FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Framework de Monitorização de Interfaces

Tiago Filipe Rodrigues Ramos

Mestrado Integrado em Engenharia Informática e Computação

Orientador: António Miguel Pontes Pimenta Monteiro

Julho de 2010

Framework de Monitorização de Interfaces

Tiago Filipe Rodrigues Ramos

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Ademar Manuel Teixeira de Aguiar (Professor Auxiliar)

Vogal Externo: Álvaro Manuel Reis da Rocha (Professor Associado)

Orientador: António Miguel Pontes Pimenta Monteiro (Professor Auxiliar)

_____

23 de Julho de 2010

Resumo

No mundo contemporâneo, verifica-se a existência de uma grande adesão a tecnologias de informação por quase todos os sectores da sociedade, como um meio de facilitar a gestão e execução de actividades relacionadas. O sector da saúde não se apresenta como uma excepção, na medida em que vários estabelecimentos, hospitalares ou de menor dimensão, recorrem a meios tecnológicos para armazenar e gerir toda a informação decorrente da sua actividade, como recursos humanos, materiais ou financeiros, para dinamizar a qualidade de atendimento e prestação de cuidados a pacientes. Para tal, as soluções utilizadas para estes fins requerem um grau de fiabilidade elevado, de modo a evitar ao máximo erros que poderão ter consequência no desenrolar dos processos terapêuticos dos pacientes. Desta forma, os processos de comunicação entre estas soluções são extremamente críticos e devem estar disponíveis em qualquer altura ou lugar.

Na empresa Glintt – Healthcare Solutions, SA, empresa de desenvolvimento de software para instituições de saúde, a avaliação da qualidade e funcionamento destes tipos de processos é um processo custoso, que requer uma grande quantidade de tempo de dedicação por parte dos seus recursos humanos, sendo por isso susceptível a erro humano.

Este documento apresenta uma proposta de solução para a criação de uma *framework* de uma plataforma genérica de monitorização de recursos lógicos que suportam estes processos, nomeadamente bases de dados e sistemas de ficheiros.

Abstract

In the contemporary world, information technologies are being embraced by a large scale from almost all sectors of society, as a way of improving the management and execution of their related activities. The healthcare sector is not an exception, for every modern hospital or a small healthcare facility apply technological assets to store and manage all information regarding the used resources, either human, material or financial, in order to improve the quality of attendance and patient care. Therefore, the solutions used to achieve these goals require a high reliability rate, to reduce the chances of interfering with the patients' therapeutical processes. So, the communication between these kinds of solutions is extremely critical.

In the company Glintt – Healthcare Solutions, SA, a healthcare software development company, the assessment of the quality and performance of these kinds of processes takes a serious amount of time to make by its human resources and it is susceptible to human error.

This document presents a generic solution proposal for a framework which aims at obtaining, evaluating and presenting the state of the logical resources which support these processes, such as databases and file systems.

Agradecimentos

A todos os que estiveram presentes.

Conteúdo

Lista de Figuras

Lista de Tabelas e Código

Abreviaturas

CLI	Common Language Infrastructure
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
EDM	Entity Data Model
ET	Error Threshold
FDD	Feature Driven Development
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
LAN	Local Area Network
MEP	Message Exchange Pattern
ORM	Object-Relational Mapping
RAD	Rapid Application Development
RIA	Rich Internet Application
SBS	Service Based Systems
SCADA	Supervisory Control And Data Acquisition
SCOM	System Center Operations Manager
SGBD	Sistema de Gestão de Base de Dados
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
W3C	World Wide Web Consortium
WCF	Windows Commnication Foundation
WT	Warning Threshold
XML	Extensible Markup Language

Capítulo 1

Introdução

A elevada dependência em tecnologias de informação para a execução de actividades, tanto a nível empresarial como a nível do utilizador comum, implica a necessidade de atenção especial ao modo como os sistemas se comportam. Desta forma, é preponderante ter conhecimento, em qualquer momento de funcionamento dos seus recursos físicos e lógicos (hardware e software, respectivamente), do seu estado e da eficácia das tarefas pelas quais se encontram responsáveis.

Neste contexto surge o conceito principal do projecto, *Monitorização*. Este termo entende-se como o acto de observar e controlar o funcionamento de algo [Info09], vigiar a ocorrência de determinados eventos. Pode ser aplicado nas mais diversas áreas, como tráfego rodoviário, segurança, protecção civil ou mesmo na área ambiental [Chu07].

Neste sentido, adequa-se a justificação da sua aplicação no domínio das tecnologias de informação. Existem no mercado diversos sistemas catalogados como sistemas de monitorização, cuja actividade principal passa por controlar recursos computacionais usados na actividade quer de uma estação de trabalho, como utilização de memória e de CPU (*Central Processing Unit*), quer no funcionamento geral de uma LAN (*Local Area Network*), através, por exemplo, do controlo de tráfego e de velocidade de transmissão de dados. Exemplos destes sistemas incluem Microsoft System Center Operations Manager [SCOM09], Hyperic HQ [Hype09], IBM WebSphere Business Monitor [IBM10] e SAP Business Process & Interface Monitoring [SAP10]. Outra aplicação importante é a monitorização a nível operacional numa linha de produção [YYS09], onde os sistemas SCADA (S*upervisory Control And Data Acquisition*) são populares.

1.1 Contexto

O trabalho insere-se numa proposta da empresa Glintt – Healthcare Solutions SA. Esta empresa, uma divisão da Glintt - Global Intelligent Technologies, cujas sectores de actividade incluem Administração Pública, Banca e Seguros, Construção e Promoção Imobiliária, Farmácia, Indústria, Logística e Distribuição, Saúde e Telecomunicações, foca-se sobretudo, como o seu nome indica, nas áreas da saúde, fornecendo soluções tecnológicas centradas em hospitais públicos e privados (grupos hospitalares), bem como clínicas e hospitais privados de pequena dimensão. [GHS09]

1.2 Motivação e Objectivos

O funcionamento das aplicações da empresa proponente, integradas nos mais variados locais a nível nacional, implica a comunicação entre diversos componentes, não relacionados entre si. Por definição, este aspecto revela a utilização de “interfaces” [Webo09], pelo que doravante o processo de comunicação entre componentes será denominado por este termo. Assim, um exemplo de uma interface poderia ser a comunicação entre uma solução clínica presente num serviço de urgência e o sistema central de pacientes, onde poderiam ser registadas as admissões de novos pacientes nesse serviço.

Actualmente, a monitorização de interfaces requer a utilização de bastante tempo por parte de recursos humanos da empresa para averiguar o estado dos componentes de comunicação. Por conseguinte, o objectivo principal do trabalho passa pela criação de uma plataforma de monitorização automática das interfaces de soluções da empresa, presentes em várias instituições cliente. Esta plataforma deve ser dotada da maior flexibilidade possível, de forma a possibilitar a monitorização de qualquer tipo de interface. Para além disso, deve possibilitar a configuração das interfaces que se pretendam monitorizar. Esta plataforma deve permitir a interacção com um utilizador, no formato gráfico, para que este possa ter acesso ao estado actual da monitorização das interfaces configuradas no momento. Deve também possibilitar a inclusão de um módulo de alertas, para responder aos casos de detecção de situações anómalas no funcionamento das interfaces monitorizadas. Esta detecção surge na sequência da extracção de conhecimento de eventos que ocorram nos processos de comunicação.

O presente documento apresenta uma proposta de solução para esta plataforma de monitorização de interfaces. Para tal, foi investigada a utilização de *software* de monitorização proprietário e a criação de uma solução própria, construída de base. Mais precisamente, o modo como ambas as abordagens se adequavam ao objectivo em questão. De seguida, procedeu-se à implementação de um protótipo funcional como meio de justificar e demonstrar a proposta de solução.

1.3 Estrutura da Dissertação

O presente documento encontra-se estruturado sob duas linhas gerais: a apresentação de uma proposta de uma *framework* para responder ao problema descrito e o desenvolvimento de um protótipo funcional, que pretende demonstrar uma implementação efectiva da *framework*.

Assim, o segundo capítulo apresenta uma abordagem teórica do problema, descrevendo aspectos gerais para a criação da uma possível solução.

O capítulo 3 apresenta uma proposta de solução para o problema, de um ponto de vista lógico, onde se descrevem os seus principais componentes e a sua interacção.

O quarto capítulo descreve as tecnologias que estiveram na base da criação de um protótipo funcional.

O capítulo 5 apresenta os detalhes de desenvolvimento do protótipo funcional.

O sexto capítulo apresenta as conclusões da realização do trabalho e perspectivas de trabalho futuro.

Por fim, as referências e anexos finais apresentam os documentos que estiveram na origem da recolha de informação para o trabalho e informação adicional relativa à implementação do protótipo, respectivamente.

Capítulo 2

Monitorização de Interfaces

Este capítulo tem como objectivo a exposição dos aspectos principais do problema de um modo mais detalhado, o estado da arte do tema principal, a monitorização, bem como as linhas orientadoras que estiveram na base da criação do método principal que guia a plataforma de monitorização.

2.1 Interfaces

Para a apresentação mais clara de uma interface, considere-se o seguinte exemplo de uma solução clínica (Figura 2.1), presente num serviço de urgência, que se pretende que esteja integrada com o sistema central de doentes, isto é, o sistema onde são registadas todas as admissões de doentes numa qualquer parte de uma instituição de saúde. Existe, então, a necessidade de uma interface de “admissão de doentes”, que representa o processo de comunicação da solução do serviço de urgência com o sistema central. Desta forma, como se pode entender, esta interface é extremamente importante, uma vez que, no caso de um doente admitido para uma urgência não transitar para a aplicação clínica, este poderá não ser tratado devidamente, quer pelo desconhecimento do seu histórico de informação de saúde, quer pela impossibilidade de prosseguir para os restantes circuitos clínicos, como pedidos de exames, administração terapêutica, entre outros. Nenhum destes circuitos informatizados pode funcionar devidamente sem a correcta passagem dos dados relativos ao doente, já que todos dependem directamente de si.

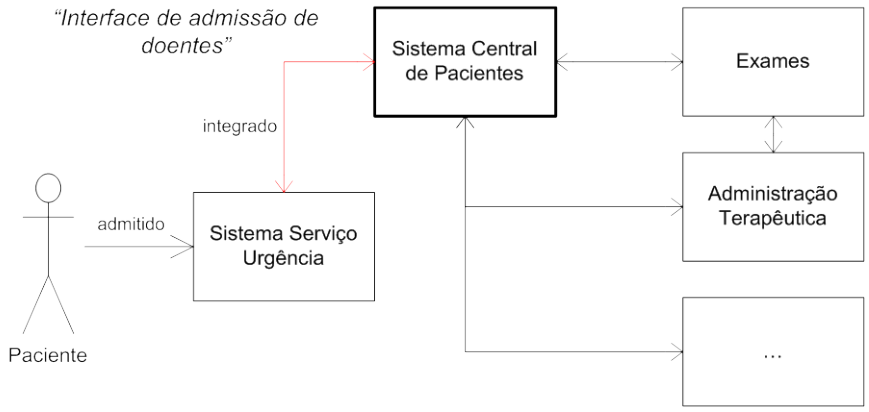

Figura 2.1 – Exemplo de interface

De um modo mais claro, o processo de comunicação entre componentes poderá ser generalizado segundo o diagrama apresentado na Figura 2.2, que representa uma visão geral da estrutura de interfaces de uma reputada instituição de saúde nacional.

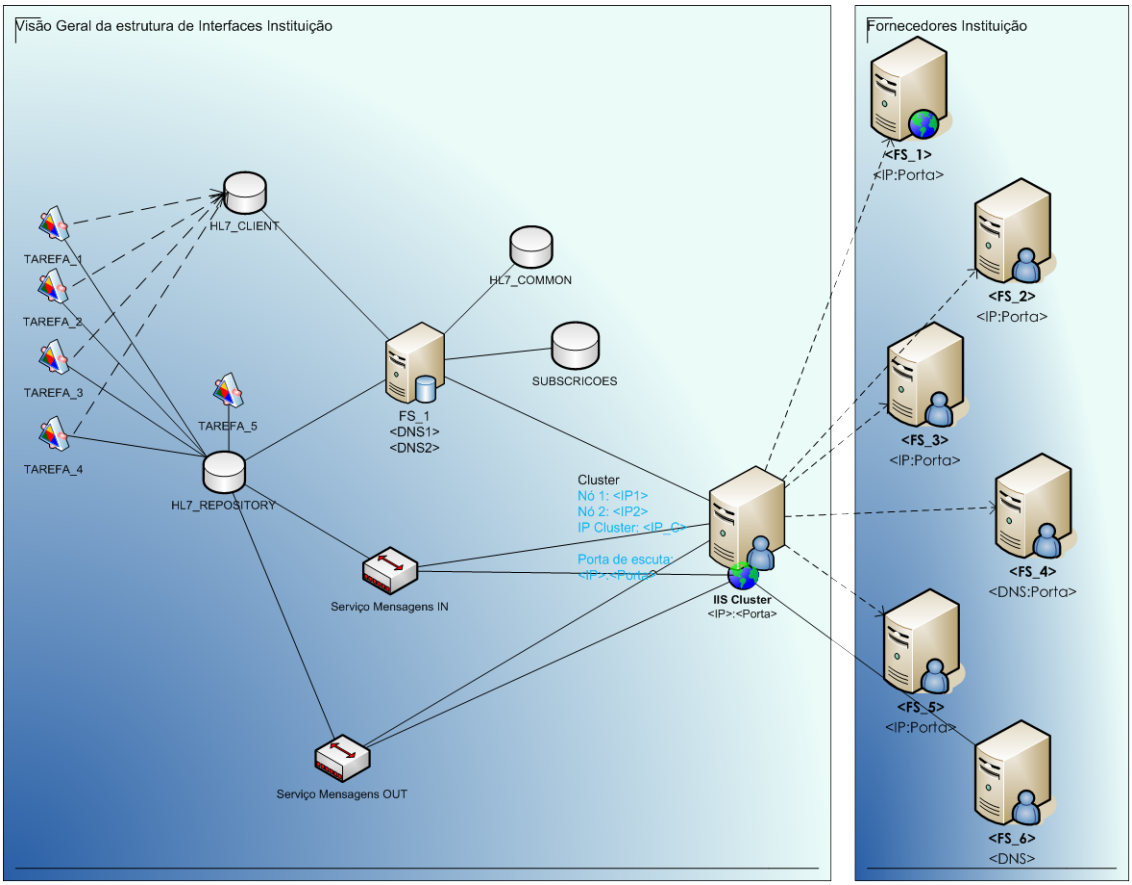

Figura 2.2 – Visão geral da estrutura de interfaces de uma instituição de saúde

Efectuando uma breve análise, verifica-se que o funcionamento de uma interface passa, sobretudo, pelo armazenamento de informação em bases de dados e sistemas de ficheiros, e respectiva circulação e partilha. Essa circulação deve-se sobretudo à execução de tarefas (*jobs*) em bases de dados e serviços em sistemas de ficheiros. Como serviço entenda-se serviço *Windows*, uma vez que se considerará, doravante, sistemas de ficheiros em ambiente *Windows*. Em suma, um serviço *Windows* procede à execução de uma ou várias aplicações, cujo objectivo é fornecer funcionalidades de longa duração, sem necessidade de intervenção por parte de um utilizador [MSDN10a]. Esta abordagem adequa-se ao cenário da execução de tarefas de modo autónomo, sem interferir de forma visível com a utilização do sistema operativo. Para além disso, podem ser iniciados automaticamente no arranque da máquina onde o sistema operativo se encontra instalado, como também podem ser terminados e reiniciados. Esta distinção do conceito de serviço deve-se principalmente à utilização de outros tipos de serviços na especificação da solução, como se poderá verificar no capítulo 4. Desta forma, doravante, o conceito de "serviço", mencionado de modo isolado, referir-se-á a um serviço *Windows*.

Em suma, estes dois elementos, bases de dados e sistemas de ficheiros apresentam-se como os principais âmbitos de acção para a monitorização de interfaces, ou seja, para atingir o objectivo principal do problema é necessário averiguar o bom estado dos recursos que se enquadrem nestes dois âmbitos.

2.2 Monitorização

O conceito de monitorização é o cerne da plataforma, pelo que se justifica que toda a implementação do objectivo principal se tenha iniciado pela definição clara de todos os aspectos necessários para a sua inclusão no contexto da plataforma. Deste modo, apresenta-se de seguida, de forma resumida, o resultado principal da investigação efectuada nesta área, que se apresentou como a base da avaliação de linhas orientadoras, que serviram como alicerces para a criação de um método geral que conseguisse responder às necessidades gerais da plataforma de monitorização.

2.2.1 Soluções existentes e métodos de monitorização

A maioria dos sistemas de monitorização é criada com um objectivo específico (avaliação de desempenho de componentes, depuração, entre outros), e com um modelo computacional em

mente, pelo que se apresenta como um desafio uma construção que seja suficientemente flexível para ser aplicada no contexto onde se insere a plataforma de monitorização proposta.

Como referido anteriormente, exemplos destes sistemas incluem Microsoft System Center Operations Manager [SCOM09]. Esta plataforma, desenvolvida pela Microsoft, é a sua solução para a monitorização de sistemas, pois foi desenvolvida para ser executada em parceria com ferramentas da empresa, mas permite também o seu funcionamento noutros sistemas que não se enquadrem nesse âmbito. Actualmente na versão 2007 R2, a plataforma permite a monitorização de aplicações e infraestruturas distribuídas, permitindo a aferição do seu estado de funcionamento. Para além deste sistema, outro exemplo que se encontra implementado comercialmente é o Hyperic HQ [Hype09]. Desenvolvido pela empresa Spring Source, este sistema enquadra-se no mesmo âmbito do SCOM, permitindo também a monitorização de aplicações e infraestruturas distribuídas. A empresa disponibilza duas versões deste sistema, Hyperic HQ Enterprise e Hyperic HQ Open Source. A principal diferença entre estas duas versões consiste nas funcionalidades disponibilizadas consoante a licença necessária, licença paga para a primeira versão e totalmente grátis para a segunda, com possibilidade de descarregamento do seu código de desenvolvimento. Porém, a versão de código aberto apresenta funcionalidades limitadas, principalmente ao nível de tratamento da escalabilidade dos sistemas monitorizados [Hype09]. Outros exemplos de implementações comerciais de sistemas de monitorização são o IBM WebSphere Business Monitor [IBM10], a solução da IBM para a monitorização de plataformas, seguindo as mesmas linhas dos dois sistemas de monitorização referidos, e o SAP Business Process & Interface Monitoring [SAP10], um módulo de monitorização de processos de negócio definidos nas soluções da empresa SAP AG. Por fim, outra aplicação importante de um sistema de monitorização situa-se no nível operacional de uma linha de produção [YYS09], onde sistemas SCADA são uma alternativa possível para controlo da eficácia dos processos e respectiva gestão.

Para além disso, algumas considerações podem ser efectuadas relativamente a estudos realizados na área de sistemas de monitorização, nomeadamente na área de monitorização de requisitos, que se considera como o seu estado da arte [Robi05], especialmente no âmbito de arquitectura e sistemas baseados em serviços (SBS). O motivo pelo qual este tipo de sistema em particular foi seleccionado como tema de pesquisa principal na investigação do estado da arte é referido no capítulo 3.

Em primeiro lugar, Sheehan, Malony e Shende [SMS99] avançam que todos os sistemas de monitorização efectuam pelo menos, numa aplicação em tempo de execução, uma de duas operações básicas: a observação de dados do programa ou a interacção com código executado. Desta forma, é implícita a existência de uma entidade computacional que fornece acesso ao contexto do programa e um mecanismo para se envolver com o código executado, à qual atribui o

nome de *agente de monitorização*. Por sua vez, a existência desta entidade justifica a existência de uma entidade adicional, o *cliente de monitorização*, que comunica com o agente, dirigindo as suas acções e interpretando a informação monitorizada. Para não competir com os recursos computacionais da aplicação, o cliente reside num contexto diferente do agente.

Ghezzi e Guinea [GG09] apresentam algumas formas para captar dados de monitorização de diversas fontes em tempo de execução. Por um lado, colocando "sondas" em locais apropriados do fluxo de dados, interceptando todas as mensagens que circulem nesses locais, nomeadamente ao nível de mensagens SOAP (*Simple Object Access Protocol*) que cheguem ou partam de sistemas onde existam implementações de serviços. Por outro lado, captando dados de eventos a um nível de abstracção inferior, como é o nível do motor de execução. Todos os eventos gerados a esse nível são registados para análise imediata ou posterior.

Spanoudakis [CS09,MS04,MS06,MS09] propõe uma arquitectura baseada em eventos [CS09], à semelhança de [GG09], e que se apresenta na Figura 2.3.

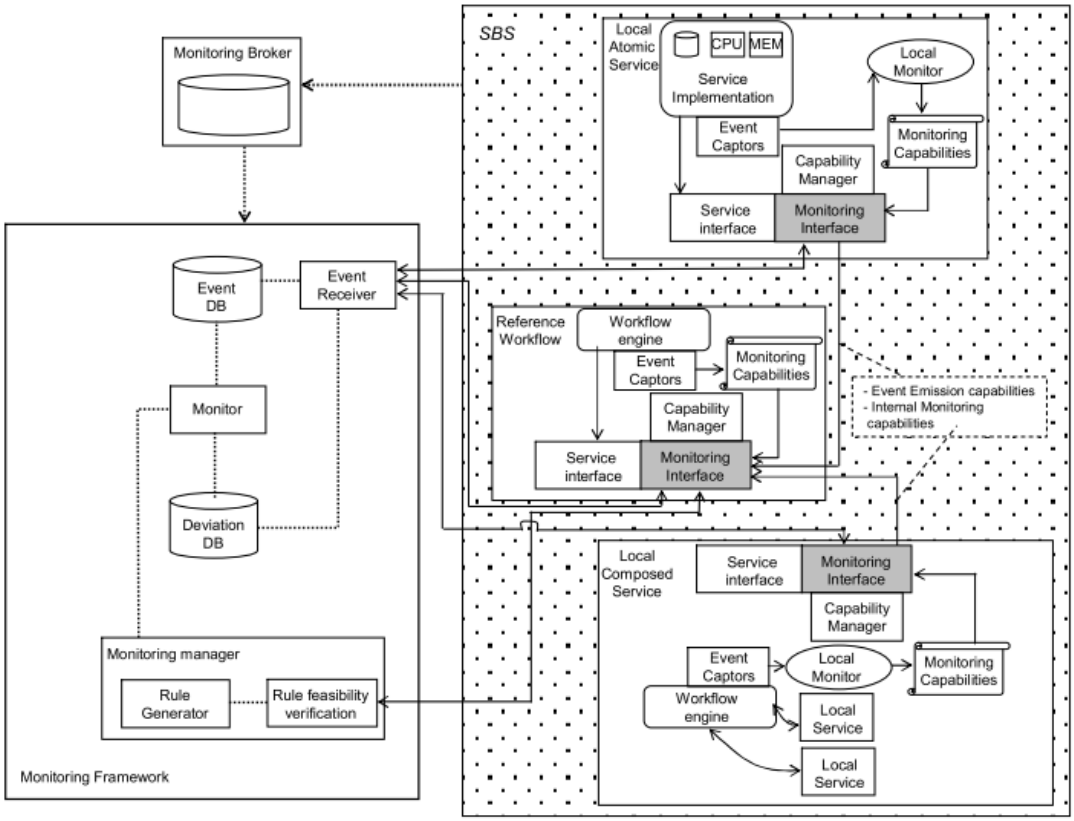

Figura 2.3 – Arquitectura proposta em [CS09]

Desta forma distingue dois tipos de serviços diferentes, os serviços atómicos e os serviços compostos (definidos por um *workflow* de encadeamento de execução de serviços). A arquitectura define a presença de monitores locais para cada serviço com o objectivo de captar os seus

eventos, criando assim uma interface de monitorização local. Para além disso, considera uma *framework* para captar todos os eventos locais de cada serviço a partir das suas interfaces de monitorização. Esta recolha de dados encontra-se unicamente restringida pelo facto dos captadores de eventos terem de possuir uma interface de comunicação comum. Estes eventos são armazenados numa base de dados específica para esse efeito. Cabe ao componente *Monitor* analisar e validar os eventos captados. Para tal, as interfaces de monitorização devem registar as referências no *Monitoring Broker*, uma base de dados.

O processo de análise e validação dos eventos consiste na sua comparação com regras de monitorização (expressas, por exemplo, em fórmulas de cálculo de eventos [MS06,MS09]), e quaisquer falhas encontradas serão armazenadas numa base de dados respectiva (*Deviation DB*). Desta forma, seria a partir da informação armazenada nesta base de dados que seriam geradas mensagens de erro captadas pelo funcionamento dos componentes de monitorização.

Outra abordagem encontrada passa pela aplicação de sistemas inteligentes multi-agente para executar monitorização e interpretação de dados. Alguns estudos nesta área foram executados [Mang05, MMM01], mas devido a alguma indefinição quanto à sua eficácia, apenas são mencionados como uma alternativa possível.

No cômputo geral, nenhuma das abordagens mencionadas pelos autores do material de investigaçãofoi sujeita a implementação prática, pelo que se desconhecia à partida a sua eficácia para a plataforma de monitorização pretendida. Assim, a abordagem para a solução do problema passou por utilizar alguns conceitos descritos pelos autores, generalizados na área da monitorização, principalmente o conceito de agente, cuja relevância será apresentada nos capítulos seguintes. Assim sendo, esta abordagem possuiu a finalidade de encontrar uma estrutura e métodos de monitorização próprios, adequados às condições que se verificam na realidade das instituições clientes.

2.2.2 Linhas orientadoras

Poder-se-á então obter algumas conclusões, a partir da investigação efectuada. No contexto em que o presente documento se insere, o conceito de “monitorização” entende-se como o acto de vigilância de processos de comunicação (interfaces), com o intuito de obter informação sobre o estado, para que seja possível aferir se os processos de comunicação decorrem, ou não, com normalidade.

Desta forma, é possível efectuar-se uma analogia importante, ao considerar-se um serviço, tal como referido na Figura 2.3, como um recurso. Surgem então algumas questões:

- Quais os intervenientes do processo de comunicação, ou seja, quais os recursos envolvidos?
- Qual a sua localização?
- Quais os parâmetros a ter em conta para a sua especificação?
- Qual é o seu estado actual?
- O estado actual é erróneo? Se sim, o que poderá ser efectuado para o corrigir?

Para que o processo de monitorização se efectuasse de modo adequado, a solução teve de fornecer resposta a estas questões. Por conseguinte, dever-se-ia realizar uma monitorização em tempo de execução da interface, ou seja, no momento que existe troca de informação entre os intervenientes do processo de comunicação [CS09], isto é, ao nível dos recursos identificados, para que seja possível averiguar o seu funcionamento adequado.

Assim sendo, para que fosse possível a criação da plataforma, foram identificadas cinco acções que surgiram como linhas orientadoras para a sua implementação:

- Identificação de recursos a monitorizar: onde se definem quais os recursos que devem ser monitorizados, como meio de averiguar o estado da interface ao qual pertencem;
- Recolha de dados: a actividade de monitorização recolhe dados classificáveis de modo autónomo, que correspondam à situação actual do recurso em questão;
- Avaliação de dados recolhidos: após a recolha de dados, deve ser possível proceder à aferição de situações erróneas ou situações que se estabeleçam segundo os padrões normais da actividade do recurso;
- Apresentação dos dados recolhidos e resultado da avaliação: deve ser possível observar, de um modo gráfico, os dados recolhidos e respectivas avaliações;
- Acção sobre recursos: deve ser possível, caso seja necessário, tomar acções sobre os recursos, com o objectivo de resolver situações anómalas e/ou outras quaisquer situações que se considerem adequadas, como a recolha de informação a partir de um pedido originado por um utilizador humano.

2.2.3 Acções de monitorização

Assim, verifica-se que a enumeração das linhas orientadoras corresponde, no fundo, à sequência de passos necessários para o funcionamento do método principal da plataforma de monitorização. Em suma, definem-se os recursos a monitorizar, obtêm-se dados a partir desses recursos, procede-se à avaliação dos dados, e, posteriormente, tanto os dados como a avaliação

são apresentados. Por fim, opcionalmente, actua-se sobre os recursos com o intuito de corrigir e/ou obter mais informação. Graficamente, na Figura 2.4,

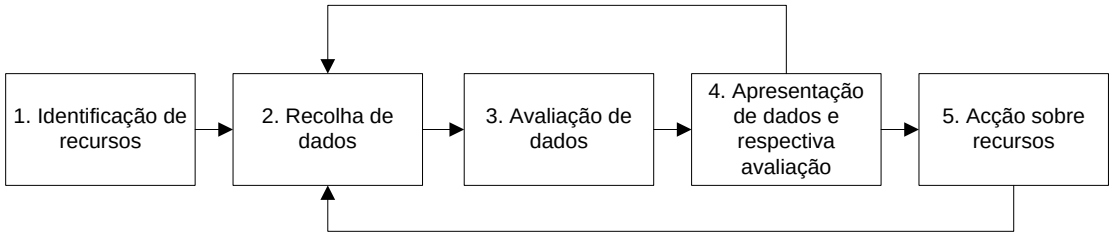

Figura 2.4 – Fluxo de acções de monitorização

Desta forma, no cômputo geral, a sequência destes passos corresponde às acções principais para recolha e apresentação de toda a informação necessária para responder ao problema em questão. Este método foi considerado como dotado do máximo de flexibilidade possível, como meio de possibilitar um comportamento dinâmico, considerando situações de alteração de parâmetros, como modificação de recursos e/ou regras de avaliação de dados.

Todavia, o principal obstáculo para o funcionamento adequado da sequência apresentada passou pelo facto de existirem diferentes tipos de recursos, que poderão pertencer a contextos completamente díspares, sujeitos a processos e regras diferenciadas para obter e avaliar resultados, bem como para acção directa nesses recursos. Assim, a especificação da plataforma de monitorização, apresentada no próximo capítulo, descreve uma possível abordagem para resolver este obstáculo.

2.3 Conclusão

Neste capítulo foram apresentados alguns pormenores importantes relativamente à plataforma de monitorização, como por exemplo, a exemplificação do que é, de facto, uma interface, assim como uma apresentação geral dos resultados obtidos pela investigação efectuada na área dos sistemas de monitorização. Para além disso, descreveu-se um método geral para proceder à monitorização de recursos, método esse que está na base de todo o funcionamento da plataforma.

O capítulo seguinte apresenta a especificação geral da plataforma de monitorização, ou seja, a descrição dos seus componentes principais. Contudo, a especificação mais detalhada da implementação da proposta de solução será apenas apresentada no capítulo 5.

Capítulo 3

Plataforma de Monitorização

A criação do método geral de monitorização de recursos foi um passo importante para a especificação da plataforma, mas, como referido em 2.2.3, sujeita a alguns obstáculos que justificam a apresentação de uma possível solução para a sua resolução. Assim, este capítulo apresenta a especificação da plataforma de comunicação, que basicamente consiste na *framework* que intitula este documento.

Para que fosse possível a implementação efectiva da plataforma de monitorização proposta, foi necessário especificar alguns aspectos relacionados, como, por exemplo, quais as entidades que iriam interagir com a plataforma, quais as suas responsabilidades, qual a relação entre si, bem como a sua disposição física.

3.1 Entidades

Através da análise da Figura 2.4, pode concluir-se que, para a execução efectiva das acções, surge como uma necessidade a delegação de responsabilidades a algum tipo de entidade. Considerando a Figura 2.2 e a Figura 2.3 verifica-se que, para a monitorização de uma interface, seria necessário actuar sobre um recurso, para obter resultados sobre o seu funcionamento. Assim, identifica-se um *Recurso* como uma entidade física que será alvo de monitorização e/ou acção, e que servirá como fonte de dados para avaliação e apresentação. No caso respectivo do problema, pode corresponder a bases de dados, tarefas em execução em bases de dados, unidades de disco, directórios de sistemas de ficheiros e serviços *Windows*, ou outros, a identificar. Analisando as abordagens mencionadas em 2.2.1 e a Figura 2.3, a abordagem de Sheehan, Malony e Shende [SMS99] adequa-se quase na totalidade ao conceito de interface de monitorização local, ou seja,

acção directa sobre o alvo de monitorização para recolha e transmissão de dados. Logo, usando o termo *Agente*, como [SMS99], identifica-se outro tipo de entidade diferente.

Um *Agente* será uma entidade lógica responsável pela recolha, avaliação e transmissão de dados de monitorização para posterior apresentação. Actua sobre um ou vários recursos para executar as suas funções e surge como uma das entidades mais importantes, uma vez que é a partir da sua actividade que se podem retirar conclusões sobre o estado dos recursos a monitorizar.

Após a identificação de recursos, considerando a sequência de passos no método de monitorização, surgem a recolha e transmissão de dados, para apresentação a um utilizador comum. Contudo, verifica-se a necessidade de conhecer o local para onde os dados serão transmitidos. Assim, considerou-se a entidade especial *proxy*, que, resumidamente, efectua a recepção e retransmissão de mensagens enviadas por agentes. Basicamente, cada instituição cliente deverá ser contemplada com uma ou mais destas entidades, que deverão ser exclusivas da instituição, para que os agentes consigam transmitir todos os dados que recolham e avaliem. A criação desta entidade foi uma das principais decisões de desenho da plataforma, detalhada mais pormenorizadamente no capítulo 5.

Assim sendo, um agente enviará mensagens para um *proxy*, que as redirecciona para o componente seguinte. Consequentemente, *Aplicação de Monitorização* corresponde à aplicação que agrega as informações obtidas em todas as instituições cliente onde existam agentes em funcionamento, sob o ponto de vista gráfico. Um utilizador interage com esta entidade para obter informação da actividade de um ou mais agentes, ou seja, para aferir o estado de funcionamento de recursos. Esta interacção inclui também pedidos a agentes para que executem acções sobre os recursos que estejam a monitorizar.

Por fim, a entidade que completa este conjunto encontra-se supra referida. O *Utilizador* corresponde à única entidade corpórea, ou seja, todos os utilizadores humanos que interajam e/ou configurem qualquer outro tipo de entidade estão incluídos nesta categoria. No âmbito da plataforma em questão, surgem três tipos de utilizadores:

- Utilizador de implementação: apresenta-se como um colaborador da empresa, cuja principal responsabilidade é iniciar o processo de monitorização numa instituição cliente. Procede a configurações, tais como a definição de recursos a monitorizar e as regras que servirão de base à avaliação de dados recolhidos;
- Utilizador de suporte: colaborador da empresa, que procede ao auxílio no funcionamento da plataforma numa instituição cliente;
- Cliente: colaborador de uma instituição cliente, que poderá consultar a aplicação de monitorização, mas apenas os dados que estejam relacionados com a instituição com a

qual colabora. Também poderá interagir com os agentes que estejam em execução sobre recursos da instituição.

3.2 Agentes

O funcionamento da plataforma de monitorização é desencadeado pelas actividades de agentes, que actuam sobre recursos pelos quais se encontram responsáveis, recolhendo, avaliando e transmitindo dados sobre o seu estado.

Como referido anteriormente, são considerados, à partida, dois âmbitos distintos para os recursos: bases de dados e sistemas de ficheiros.

Assim, a principal decisão de desenho da plataforma passou por definir que âmbitos diferentes teriam tipos de agente diferentes, ou seja, existe um tipo de agente específico para um tipo de recurso específico. Esta decisão surge naturalmente, uma vez que o método de recolha de dados numa base de dados será diferente do método de recolha de dados no contexto de um sistema de ficheiros.

Desta forma, surgem dois tipos de agente: agente de bases de dados e agente de sistemas de ficheiros, denominados *DBAgent*, e *FilesystemAgent*, respectivamente.

3.2.1 Decisões de desenho

Em primeiro lugar, é visível a diferença desta abordagem e a proposta de Sheehan, Malony e Shende [SMS99] e a arquitectura de Spanousakis [CS09], uma vez que, em ambas as situações apresentadas pelos autores, os responsáveis pela recolha de informação delegavam a responsabilidade de avaliação de dados recolhidos a uma entidade diferente. No caso da plataforma de monitorização, tomando como base a arquitectura de Spanoudakis, essa tarefa pertenceria à *Aplicação de Monitorização*, entidade que surge como o destinatário dos dados recolhidos pelos agentes. Contudo, os agentes, na plataforma de monitorização, também avaliam os dados que recolhem. Tal deve-se, sobretudo, à necessidade que um utilizador que se enquadre na categoria de *Cliente* também possa controlar o funcionamento do agente e ao modo como a plataforma é implementada fisicamente (ver 3.4.2.2).

Para além disso, os fluxos de acções apresentados tiveram como meta, exclusivamente, dotar a plataforma do máximo de flexibilidade possível, para possibilitar a adição de elementos ainda imprevistos nesta plataforma e que poderão surgir no futuro.

Este aspecto, o desenho e implementação de agentes, surge como o cerne da plataforma, pois os dados recolhidos serão aqueles que estarão na base de qualquer avaliação e conclusão que se poderá retirar do funcionamento das interfaces pretendidas.

3.2.2 Especificação funcional

Para que um agente proceda de modo adequado às acções pretendidas, é necessário efectuar um passo absolutamente vital, a sua configuração. Em suma, a configuração de um agente define o seu âmbito de acção, o(s) recursos(s) onde deve actuar e o modo como deve actuar. Consequentemente, surge um novo conceito, *Tarefa*, que consiste numa acção que um agente pode efectuar. Logo, a configuração do modo como um agente deve actuar consiste na configuração das tarefas que deverá realizar. Tal facto é um dos principais aspectos com que um agente se distingue no universo de agentes, pois agentes de tipos diferentes executarão tarefas diferentes, evidentemente.

Resumindo, com o auxílio da Figura 3.1, para a configuração de um agente é necessário,

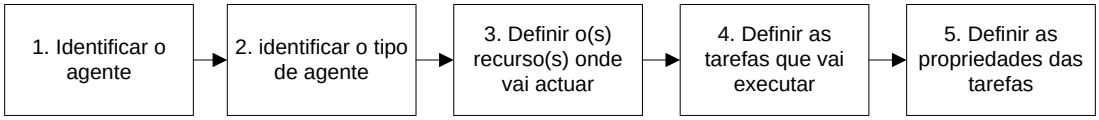

Figura 3.1 – Passos para a configuração de um agente

Ou seja, um exemplo da estrutura de uma possível configuração poderá ser:

```
Agente
   Identificador = ’Agente1’
   Tipo = ‘<Tipo de agente>’
   Recursos
          Recurso
                 Identificador = ‘Recurso1’
                 Tarefas
                        Tarefa
                               <Definição de tarefa>
                        Tarefa
                               <Definição de tarefa>
                        …
          Recurso
                 Identificador = ‘Recurso2’
                 Tarefas
                        Tarefa
```

```
                <Definição de tarefa>
        Tarefa
                <Definição de tarefa>
        …
…
```

Código 3.1 – Estrutura de uma possível configuração de um agente

Um dos aspectos principais da plataforma é a flexibilidade, ou seja, a plataforma deve ser capaz de monitorizar novas interfaces com o mínimo esforço possível. Este aspecto consiste, basicamente, na monitorização de novos recursos, consequentemente, da adição de novos agentes. Como não existe uma definição estática dos tipos de recursos existentes, foi necessário definir um comportamento geral dos agentes que iriam actuar sobre recursos, de forma a possibilitar a criação rápida de novos tipos de agente. Desta forma, a solução proposta consiste num fluxo de acções comum a todos os tipos de agente existentes, prevendo, também, que os agentes que poderão vir a existir no futuro tenham o mesmo comportamento. A Figura 3.2 apresenta o fluxo de acções referido.

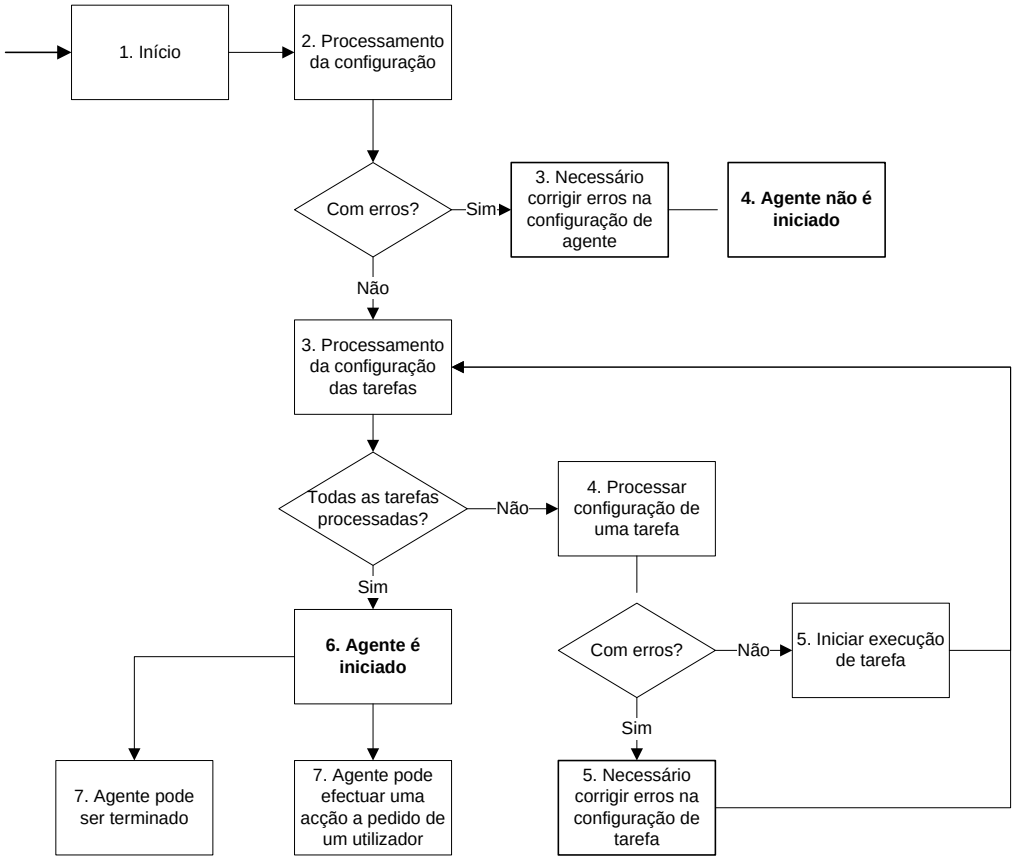

Figura 3.2 – Fluxo de execução de um agente

Assim sendo, um agente apenas poderá ser correctamente iniciado se o processamento das suas tarefas, ou seja, a ordem para que as suas tarefas sejam executadas, seja completado com sucesso. Para tal, será necessária a correcta configuração das propriedades do agente. Estas propriedades referem-se à correcta definição do identificador, do tipo de agente e recursos. Contudo, é visível que um agente pode ser correctamente iniciado mesmo que existam erros na configuração das suas tarefas. Tal permite que tarefas correctamente configuradas sejam iniciadas mesmo que existam erros noutras. Caso tal aconteça, o agente notifica esse facto, gerando eventos, apresentados sob a forma descrita em 5.1.

3.2.3 Tarefas

O passo 4 da Figura 3.2 é preponderante para se poder processar as tarefas do agente, dado que existem tarefas específicas para cada tipo concreto de agente, e só assim o passo 2 da mesma figura pode ser efectuado de modo adequado.

Para a configuração de uma tarefa, são necessários os seguintes passos:

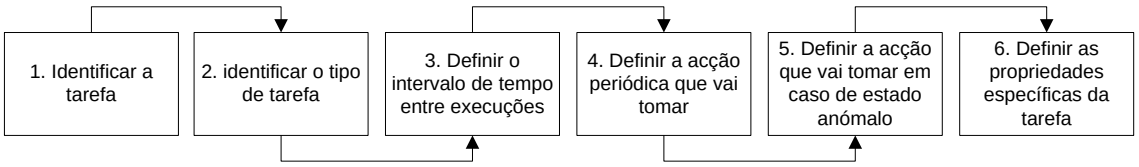

Figura 3.3 – Passos para configuração de uma tarefa

Em suma, uma tarefa será executada de modo periódico até que o agente que se encontra responsável termine a sua execução, tal como representado no passo 3 da Figura 3.3.

Um exemplo da estrutura de uma possível configuração de tarefas poderia ser:

```
Tarefas
  Tarefa
        Identificador = ’Tarefa 1’
        Tipo = ‘<Tipo de tarefa>’
        Intervalo = ‘<intervalo_em_segundos>’
        Acção Periódica = ‘acção_a_tomar’
        Acção Erro = ‘acção_a_tomar_para_erro’
        Propriedades
              <Lista de propriedades>
  Tarefa
```

```
Identificador = ’Tarefa 2’
Tipo = ‘<Tipo de tarefa>’
Intervalo = ‘<intervalo_em_segundos>’
Acção Periódica = ‘acção_a_tomar’
Acção Erro = ‘acção_a_tomar_para_erro’
Propriedades
        <Lista de propriedades>
…
```

Código 3.2 – Exemplo de estrutura de configuração de tarefas de um agente

Cada tipo de agente é responsável por agendar o início de execução da acção periódica de uma tarefa, específica do tipo de agente que efectua o agendamento. Porém, o modo de execução de tarefas, de forma análoga à execução de agentes, também obedece a um comportamento comum, independentemente das actividades que terá de realizar, como a Figura 3.4 expressa.

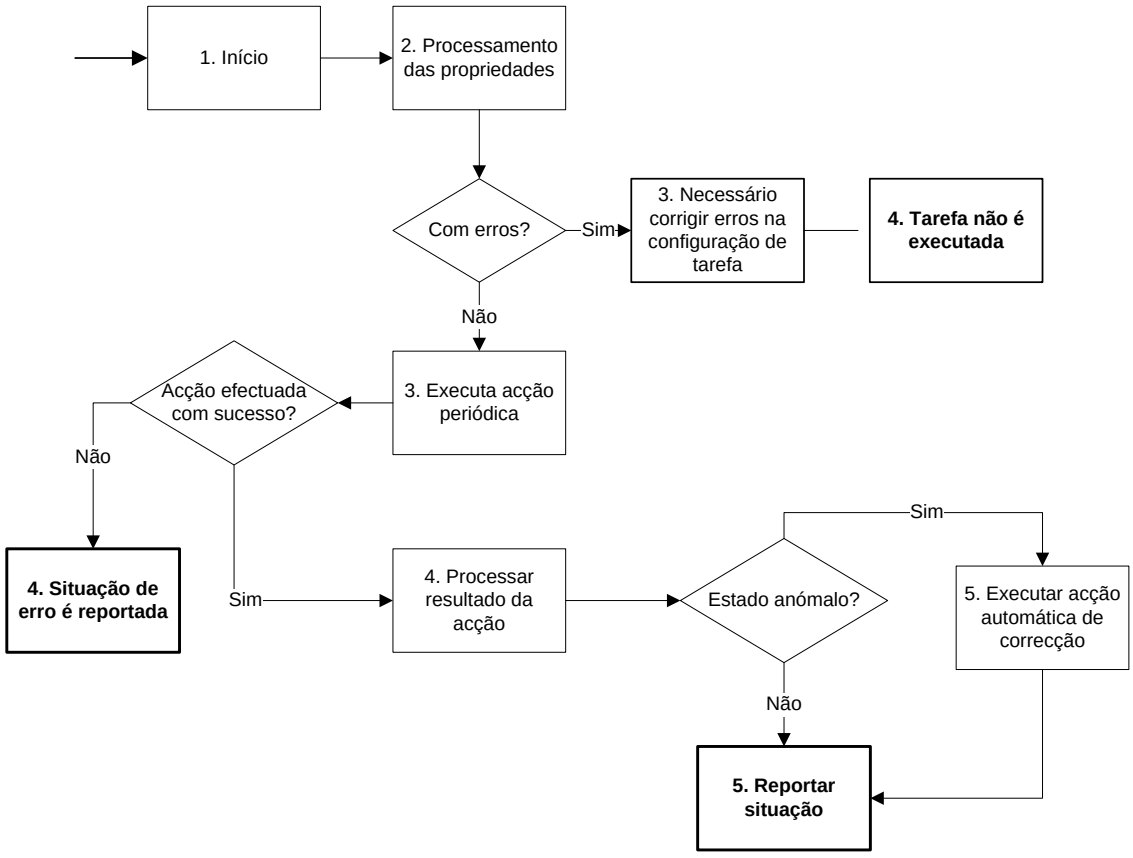

Figura 3.4 – Fluxo de execução de uma tarefa

Assim sendo, o aspecto que diferencia as tarefas executadas será as acções executadas, a acção periódica e a acção a tomar em caso de ser encontrada uma situação anómala, dado que o fluxo de execução de acções é geral para todas as tarefas existentes.

3.2.4 Agente de base de dados

Um agente de base de dados (*DBAgent*), actua, como o seu próprio nome indica, em bases de dados, executando todos os tipos de tarefas que se julguem adequadas para este tipo de contexto em particular. Como anteriormente referido, um agente deve reportar dados mensuráveis, passíveis de serem sujeitos a uma avaliação quantitativa ou qualitativa, ou situações de erro, no caso de não conseguir obter dados que possam ser sujeitos a estes tipos de avaliação.

No caso de bases de dados, exemplos de dados mensuráveis podem incluir, entre outros, espaços ocupados por *tablespaces*, ou um conjunto de tuplos de uma tabela que obedeçam a um critério. No fundo, depende da configuração que se fizer do agente, ou, mais especificamente, das suas tarefas.

Relativamente a este contexto, para que um agente de base de dados agende e execute uma tarefa, a sua configuração deve especificar bases de dados como recursos, e as tarefas que serão executadas para cada base de dados. Neste caso, um agente de base de dados apenas possui uma tarefa específica, denominada *MonitoringResourceTask*, que, traduzindo, ilustra de uma forma clara a acção a tomar, ou seja, a tarefa de monitorização de um recurso, uma base de dados.

Um agente de base de dados tem, como acção a tomar, uma *query* ou um procedimento, ou seja, um comando que executa na base de dados como forma de obter um resultado. Este *query*/procedimento depende daquilo que se pretende que o agente execute e que deve estar correctamente estruturado para que a sua execução devolva um resultado lógico, como, por exemplo, um valor numérico para averiguar o espaço ocupado por um *tablespace* em particular, no caso de uma *query*, ou o número de tuplos afectados, no caso de um procedimento. O mesmo se aplica para a acção a tomar no caso de encontrar um erro.

Para além disso, o aspecto que torna a configuração de uma tarefa específica para o contexto de uma base de dados é as propriedades específicas da tarefa, ou seja, conforme referido na possível estrutura da configuração de uma tarefa, em 3.2.2, a sua lista de propriedades.

Assim sendo, a plataforma permite a inclusão de qualquer tipo de propriedade, pelo que tudo depende do tratamento que estas propriedades terão quando a configuração de uma tarefa for processada.

3.2.5 Agente de sistema de ficheiros

Um agente de sistemas de ficheiros (*FilesystemAgent*), de forma similar ao agente de base de dados, actua sobre o contexto específico de um sistema de ficheiros. Para a plataforma de monitorização, apenas são considerados sistemas de ficheiros relativos a ambientes *Windows,*

Da mesma forma, um agente de sistemas de ficheiros também reporta dados passíveis de serem sujeitos a avaliação, para que seja possível reportar situações anómalas, caso tal se justifique.

Cenários possíveis neste contexto podem ser, por exemplo, a averiguação do espaço livre de uma unidade de disco, o espaço ocupado por um directório, o número total de ficheiros que estejam num directório há mais tempo do que um limite determinado, ou ainda o estado de execução de um serviço. De facto, não existe um conjunto pré-definido do tipo de dados que deve ser obtido pela recolha.

Uma acção, em agentes sistemas de ficheiros, consiste na definição do nome de um método implementado pelo agente, nome que deverá ser conhecido no momento da configuração do agente, o que requer o conhecimento prévio dos nomes dos métodos disponibilizados por este tipo de agente.

As tarefas para estes agentes obedecem também ao comportamento geral de tarefas, e, uma vez mais de forma análoga aos agentes de bases de dados, também permitem a inclusão de qualquer tipo de propriedade, sujeitas também ao modo como serão processadas pelo agente ao qual a tarefa configurada pertence.

3.3 Aplicação de monitorização

A aplicação de monitorização é a entidade gráfica, aquela que é responsável por apresentar, ao utilizador humano, os resultados das actividades dos agentes. Em suma, recebe, regista e apresenta os resultados reportados pelos agentes. Para além disso, como será de fácil compreensão, é a ponte entre um utilizador humano e os agentes, uma vez que, para além de apresentar os resultados da execução das suas tarefas, também permite que um utilizador humano controle o funcionamento de um agente, como por exemplo, ordenando que este execute uma tarefa inesperada que não considerou na sua configuração.

Basicamente, esta aplicação agrega os resultados das tarefas de todos os agentes, de todas as instituições clientes. A aplicação tem como principal requisito o facto de não ser uma aplicação local, mas sim uma página w*eb*, se possível acessível em qualquer local com acesso à Internet.

A apresentação da informação é efectuada graficamente, onde são apresentados os recursos monitorizados, bem como o seu estado, se foram detectadas anomalias ou não, tendo como base os resultados obtidos pelos agentes. Dado que este componente é de relativa complexidade, poderão ser identificados componentes secundários que estruturam a aplicação de monitorização.

3.3.1 Base de dados

A fonte de informação de toda a plataforma, que servirá como base para os dados apresentados pela aplicação de monitorização é uma base de dados.

A decisão de ser utilizada este tipo de plataforma de colecção de dados, relativamente a alternativas possíveis, como um sistema de ficheiros, por exemplo, deve-se ao facto do seu alojamento não ser considerado como uma restrição pela empresa proponente da plataforma. Para além disso, a plataforma beneficia de algumas vantagens que uma base de dados permite [DBAD10], como redundância de dados reduzida, logo, maior consistência e integridade de dados, e acesso a dados facilitado pelo uso de linguagens de interrogação de bases de dados, por exemplo.

Em suma, é utilizada uma base de dados relacional, cuja estrutura é o mais minimalista possível, uma vez mais devido ao facto de se pretender que a plataforma seja dotada de flexibilidade. Possíveis alterações nos tipos de dados em circulação evitarão, assim, a reestruturação da base de dados, de forma a permitir albergar as alterações efectuadas. A estrutura será pormenorizada no capítulo 5.

Nesta base de dados são armazenadas todas as informações relevantes para a definição das interfaces monitorizadas. Assim sendo, armazenam-se informações sobre as instituições cliente, informações sobre os agentes responsáveis por monitorizar recursos dessas instituições cliente, como também os resultados da execução das tarefas por parte desses mesmos agentes. Uma vez que a presente secção se foca na base de dados, doravante o resultado de execução de uma tarefa será intitulado *relatório*, designação utilizada na plataforma.

Para além disso, é adequado referir como são inseridas informações nesta base de dados. A introdução de informações relativas a instituições cliente e relatórios é bastante linear. Resumidamente, as informações das instituições são inseridas por um utilizador que se enquadre na categoria *Utilizador de implementação*, que insere directamente as informações necessárias,

dado que este tipo de informação apenas é relevante para colaboradores da empresa. Quanto a relatórios, estes também são inseridos directamente na base de dados, segundo o método apresentado no capítulo 5.

Por fim, a informação de agentes é a única que sofre restrições. Desta forma, um aspecto importante, e que, de forma premeditada, foi omisso anteriormente, é o facto de os agentes só enviarem relatórios quando se encontrarem registados na aplicação de monitorização. Por conseguinte, o primeiro passo de um agente, após processar a sua configuração, será registar-se na aplicação de monitorização, isto é, ser inserido na base de dados. Só após a confirmação deste registo é que o agente envia relatórios. Tal deve-se ao facto de, assim, se assegurar que apenas agentes que estejam em funcionamento efectivo sejam inseridos na base de dados, evitando assim o registo de relatórios de agentes sem qualquer registo. Para além de violar uma regra de integridade referencial de uma base de dados relacional, seria perigoso para o controlo dos agentes activos na aplicação de monitorização, não sendo possível obter uma lista efectiva dos agentes em execução numa dada instituição.

Este aspecto também suscita uma nova situação. E se o agente já tiver sido registado antes? Nesse caso, é assinalada ao agente a sua situação afirmativa de registo, mas este não é inserido na base de dados, porque já se encontra presente.

Assim sendo, a Figura 3.2 é actualizada, como se pode verificar na Figura 3.5.

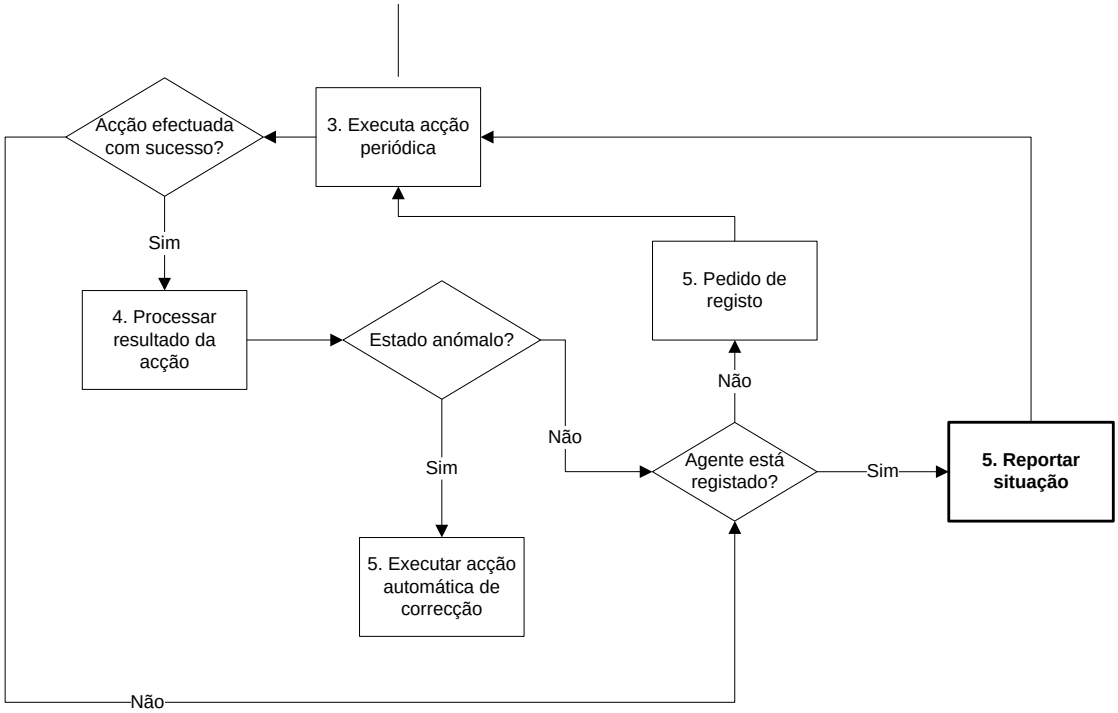

Figura 3.5 – Fluxo de execução de um agente (actualizado)

3.3.2 Gestor de mensagens

Este componente secundário corresponde ao responsável pela recepção de mensagens que são enviadas por agentes, isto é, envios de relatórios ou pedidos de registo. Por conseguinte, é o responsável pela verificação da existência e posterior inserção de agentes na base de dados, quando estes efectuam pedido de registo, segundo o processo descrito. Por outro lado, dado que é o responsável pelo acesso à base de dados para acções de inserção, também será o responsável por proceder a actualizações do estado de funcionamento dos recursos na aplicação de monitorização, uma vez que irá inserir os relatórios enviados por agentes na base de dados.

Para além disso, é o responsável pelo envio de mensagens que tenham origem na aplicação de monitorização, destinadas a agentes. Como mencionado, estas mensagens consistem em pedidos efectuados por *Utilizadores* para que os agentes executem uma acção.

Em suma, este componente secundário actua como uma caixa de correio da aplicação de monitorização, sendo o seu ponto de entrada e de saída de mensagens, de e para agentes, respectivamente.

3.4 Arquitectura

Após a enumeração dos componentes principais, é importante referir o modo como se relacionam, não só do ponto de vista lógico, ou seja, compreender os emissores e receptores de dados nos processo de comunicação entre entidades, como também compreender como se dispõem do ponto de vista físico, ou seja, como se deve proceder à instalação das entidades para que possam efectivamente executar as acções pelas quais se encontram responsáveis.

3.4.1 Arquitectura lógica

Através da descrição efectuada ao longo do presente capítulo, é possível obter-se uma noção aproximadamente clara do modo como as entidades se relacionam para comunicar as suas acções e receber resultados de acções de outras entidades.

Podem ser diferenciados dois tipos de fluxos, um fluxo unidireccional, que consiste no envio de mensagens de agentes para a aplicação de monitorização, que, no fundo, consistem em relatórios. Para além disso, verifica-se também a existência de um fluxo bidireccional, onde existe

troca de mensagens entre agentes e a aplicação de monitorização. A sua utilização realiza-se quando existem mensagens de registo de agentes, que necessitam de obter uma confirmação do seu pedido, como também quando existem mensagens originadas pela aplicação de monitorização, que, por sua vez, necessita de receber uma resposta do agente ao qual se dirigiu.

A Figura 3.6 ilustra, do ponto de vista lógico, esta relação. São apresentados, os principais fluxos relativos às principais acções que as entidades que compõem a plataforma executam, sob a forma de cores. Refira-se que a figura apresenta apenas a arquitectura lógica para uma única instituição. Dado que se pretende que existam várias, o bloco *Instituição* repete-se tantas vezes quanto o número de instituições cliente cujos recursos se pretendam monitorizar.

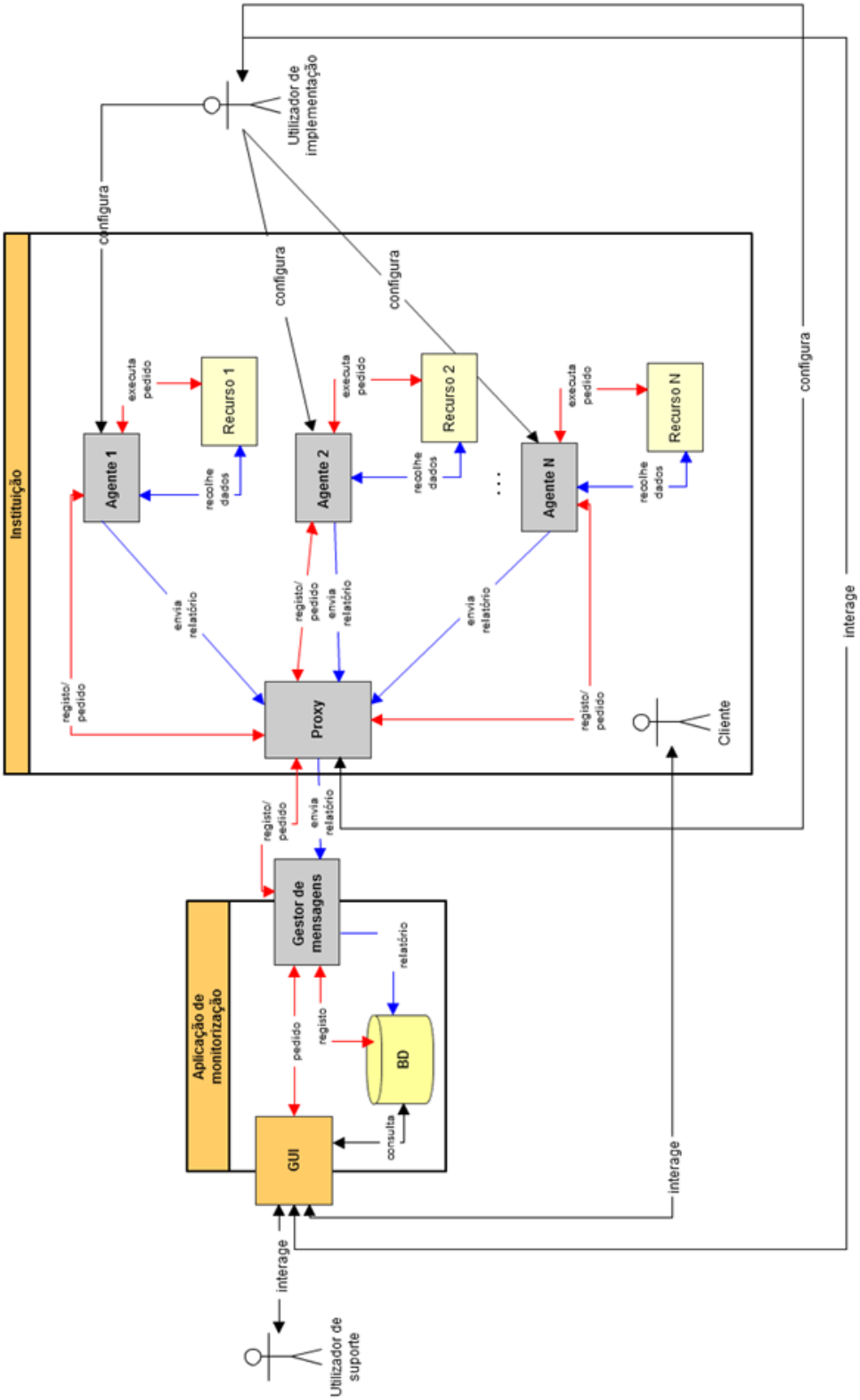

Figura 3.6 – Arquitectura lógica

3.4.2 Arquitectura física

Do ponto de vista físico, a descrição efectuada nos capítulos anteriores permite concluir que a plataforma de monitorização é um sistema distribuído, na medida em que não só os recursos de uma instituição não se encontram centralizados num único local, como são contempladas várias instituições, que, naturalmente, não se encontram concentradas num único ponto geográfico. Dado o âmbito de actividade da empresa proponente, estas encontram-se distribuídas nos mais diversos pontos do país.

3.4.2.1 Decisões de implementação

Assim, uma vez que todo o âmbito de funcionamento da plataforma é em ambientes *Windows*, foi efectuado um levantamento de possibilidades para que as entidades conseguissem executar as suas tarefas, nomeadamente as entidades representadas nos blocos cinzentos da Figura 3.6. Um aspecto preponderante é o facto de todas estas entidades não necessitarem de intervenção por parte de um utilizador humano para poderem ser executadas, ou seja, são entidades que não apresentam uma GUI (*Graphical User Interface*) para interacção com o utilizador, sendo dotadas de autonomia para poderem iniciar. Assim sendo, foram contempladas duas alternativas, a implementação das entidades como aplicações de consola ou como serviços Windows. A escolha não apresentou dificuldade. Uma aplicação em consola necessita de ser explicitamente iniciada pelo utilizador, apresentando, naturalmente, uma consola. Um aspecto que não é necessário é a existência de uma consola durante a sessão Windows na qual foi iniciada, pois os agentes, *proxies* e gestores de mensagens poderão encontrar-se em execução em *background*, em sessões próprias, não interferindo com a utilização regular do sistema operativo. Este aspecto é totalmente resolvido por um serviço Windows, que cumpre todos os requisitos referidos. Desta forma, um serviço Windows pode ser automaticamente iniciado pelo sistema operativo, quando este é iniciado, ou explicitamente controlado pelo utilizador. Este aspecto possibilita que o funcionamento da plataforma esteja sempre activo, mesmo quando a execução dos serviços é interrompida.

3.4.2.2 Distribuição geográfica

A Figura 3.7 apresenta o modo como a plataforma é implementada fisicamente.

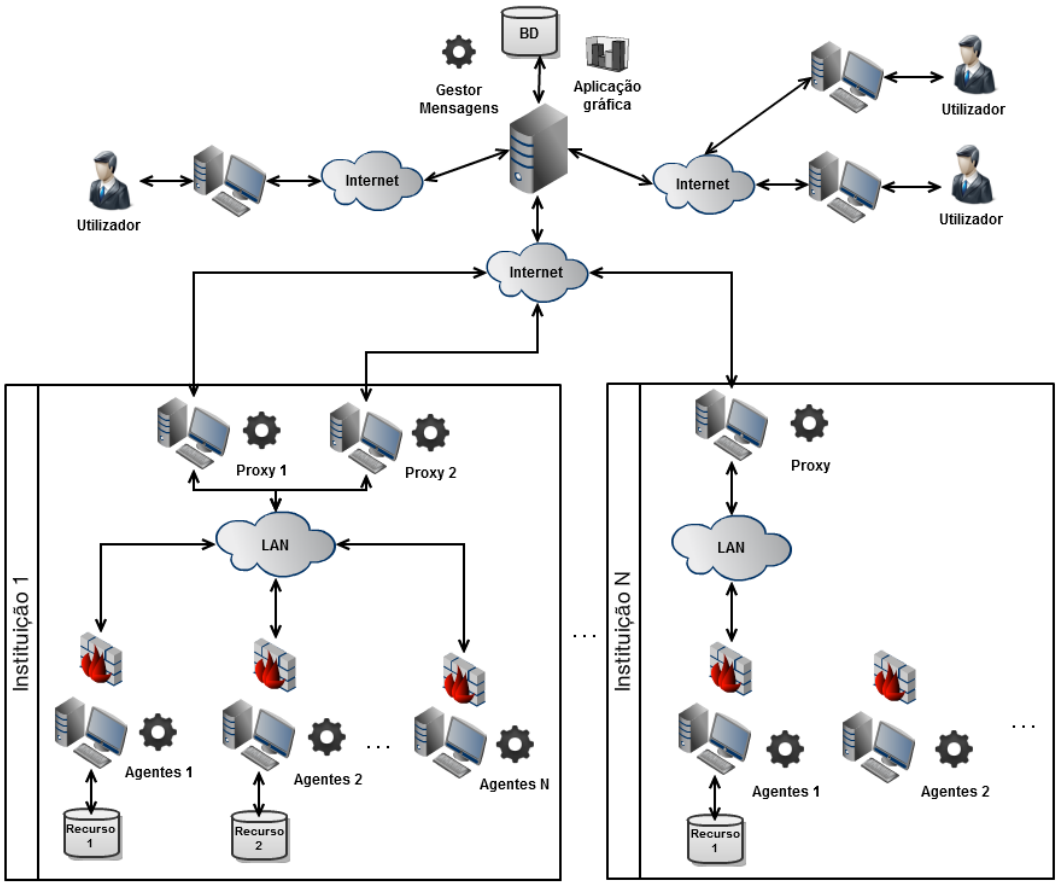

Figura 3.7 – Arquitectura física

A escolha de serviços Windows, de facto, contempla quase todas as funcionalidades necessárias para a plataforma, uma vez que estes serviços permitem a recolha de dados, o redireccionamento de mensagens, bem como acesso à base de dados, ou seja, o fluxo principal de dados da plataforma. Porém, a apresentação da actividade da plataforma, sob o ponto de vista gráfico também é um aspecto importante. Assim sendo, fisicamente, a aplicação de monitorização, como representado pela Figura 3.7, que inclui a aplicação gráfica, a base de dados e o serviço de gestão de mensagens apresentam-se como o conjunto agregador de dados, o local onde todos os fluxos da plataforma terminam ou são iniciados. Este facto permite afirmar que a aplicação de monitorização é o componente central da plataforma, o que é visível na Figura 3.7.

3.5 Conclusão

O presente capítulo descreveu, de uma forma geral, os principais componentes da plataforma de monitorização e a sua funcionalidade geral. Para além disso, foi apresentada a sua arquitectura, quer de um ponto de vista lógico, onde se descrevem os relacionamentos entre as várias entidades, com identificação dos fluxos de dados principais, quer de um ponto de vista físico, onde, sob forma esquemática, foi apresentado um mapa geral da plataforma de monitorização.

Porém, alguns aspectos carecem de detalhe técnico. Tais aspectos serão apresentados nos próximos capítulos, onde serão enumeradas as tecnologias principais, como também os detalhes mais relevantes para a implementação de um protótipo funcional.

Capítulo 4

Tecnologias de suporte

Este capítulo tem como objectivo a apresentação do estado da arte das principais tecnologias utilizadas para a criação de um protótipo funcional, que justifique que as decisões tomadas para efectuar a sua implementação respondem aos objectivos da plataforma de monitorização. Uma vez que surge antes da especificação efectiva do protótipo, não existe uma relação óbvia entre si, pelo que tentar-se-á, tanto quanto o possível, enquadrar uma tecnologia com a sua aplicação prática na plataforma.

4.1 Considerações Gerais

As escolhas das tecnologias que foram utilizadas para a implementação de um protótipo funcional encontraram-se restringidas pelo facto da empresa na qual o projecto se realizou pretender que a plataforma de monitorização fosse implementada com tecnologias pertencentes à Microsoft. Antes da sua selecção, foram exploradas algumas alternativas, nomeadamente a utilização de alguns sistemas de monitorização mencionados no capítulo 1.

Assim, foram tomados em consideração dois sistemas de monitorização: Microsoft System Center Operations Manager [SCOM09] e Hyperic HQ [Hype09]. Quanto ao primeiro, a razão deve-se ao facto de ser um sistema de monitorização da Microsoft, o que ia ao encontro da principal restrição mencionada. O segundo, por permitir que uma versão disponibilizada pela empresa que implementou o produto fosse de código aberto, facilitando a alteração das suas funcionalidades. O Microsoft System Center Operations Manager é mais rico, disponibilizando mais funcionalidades, passíveis de serem estendidas e/ou adicionadas, através de um SDK [SCOM10]. Por outro lado, necessita de uma licença paga. Quanto ao Hyperic HQ, o seu principal aspecto favorável é a possibilidade de ser extensível, tendo em conta o código aberto mencionado antes, mas a sua implementação é efectuada recorrendo à plataforma Java, o que impediu à partida a sua selecção. Desta forma, à primeira vista, o Microsoft System Center

Operations Manager apresentou-se como o candidato mais viável para ser a base da plataforma de monitorização. Porém, a necessidade da aplicação de monitorização possuir uma interface gráfica com uma determinada estrutura, impediu a selecção desta solução da Microsoft, uma vez que, apesar de permitir a extensão de funcionalidades, não permite a edição das vistas para o utilizador.

Assim, como não foram encontrados mais sistemas de monitorização que se enquadrassem nas restrições consideradas, e uma vez que a Microsoft não disponibiliza mais sistemas dedicados exclusivamente à área da monitorização, a decisão final foi a criação de uma plataforma de monitorização de raiz, permitindo maior flexibilidade na definição e implementação das funcionalidades desejadas.

4.2 XML

A linguagem XML (*Extensible Markup Language*) revela-se como uma parte fundamental para a plataforma de monitorização, uma vez que é a linguagem utilizada para a descrição de muitos elementos essenciais para a construção de um protótipo funcional.

Esta linguagem, desenvolvida por iniciativa do World Wide Web Consortium (W3C) [W3C10] em 1996, tem como função a estruturação e armazenamento de informação sob a forma hierarquizada, com flexibilidade para a criação de qualquer estrutura numa forma textual.

Esta linguagem adequa-se no contexto da plataforma de monitorização, visto que:

- Possibilita um elevado grau de interoperabilidade, uma vez que um dos seus principais objectivos no seu desenho é a independência de sistemas;
- Permite a padronização de estruturas de dados para sistemas que necessitem de comunicar entre si;
- Não se encontra restringida por licenças, é uma linguagem não proprietária;
- A sua legibilidade permite a fácil edição para um utilizador comum;
- O seu uso encontra-se já bastante generalizado, e visto que é suportada por um vasto conjunto de aplicações e organizações, existem diversas ferramentas livres para a criação e edição de documentos formatados nesta linguagem.

Desta forma, as sucessivas trocas de mensagens entre as entidades da plataforma de monitorização conduziram à escolha desta linguagem como meio de normalizar o seu conteúdo.

4.3 .NET *Framework*

Actualmente na versão 4, lançada em Abril de 2010, a *Framework* .NET da Microsoft, apresenta-se como uma plataforma que fornece uma vasta gama de possibilidades para a construção de aplicações em ambientes Windows [NET10]. Assim, a utilização desta plataforma adequa-se na totalidade para a criação da plataforma de monitorização, uma vez que, como foi referido, o seu contexto inclui-se neste universo.

A grande complexidade desta plataforma não permite a sua total descrição, visto que não se enquadra no âmbito deste documento. Porém, existem alguns componentes desta plataforma que devem ser abordados, pois a implementação do protótipo da plataforma de monitorização foi totalmente efectuada com a sua utilização.

Em suma, o cerne do seu ambiente de execução consiste numa especificação desenvolvida pela Microsoft, a *Common Language Infrastructure* (CLI) [CLI10], que permite a criação de ambientes de execução e desenvolvimento onde linguagens e bibliotecas podem funcionar em conjunto. Tal deve-se à criação de um ambiente de execução virtual, que encapsula aplicações criadas neste ambiente do sistema operativo, possibilitando que aplicações criadas em linguagens que obedeçam à especificação CLI possam ser executadass sem necessidade de reescrita. Neste conjunto de linguagens encontram-se, entre outras, C#, VB.NET, ou ainda a recém-criada F#. Assim sendo, a figura Figura 4.1 ilustra, globalmente, como a especificação é usada na prática.

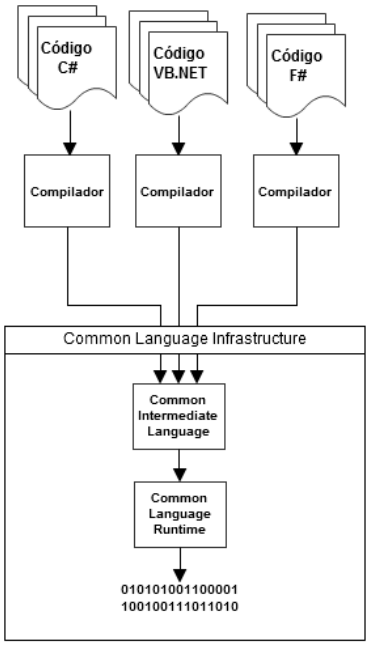

Figura 4.1 – Visão Geral da Common Language Infrastructure

Assim, todo o código-fonte escrito em linguagens de alto nível, como as que estão mencionadas na Figura 4.1, é compilado na linguagem *Common Intermediate Language* (CIL), posteriormente executada no ambiente *Common Language Runtime (CLR)*. Este ambiente é a base de toda a execução de todo o código escrito em linguagens suportadas pela plataforma.

A plataforma, genericamente, poderá ser estruturada da seguinte forma [Yvas10]:

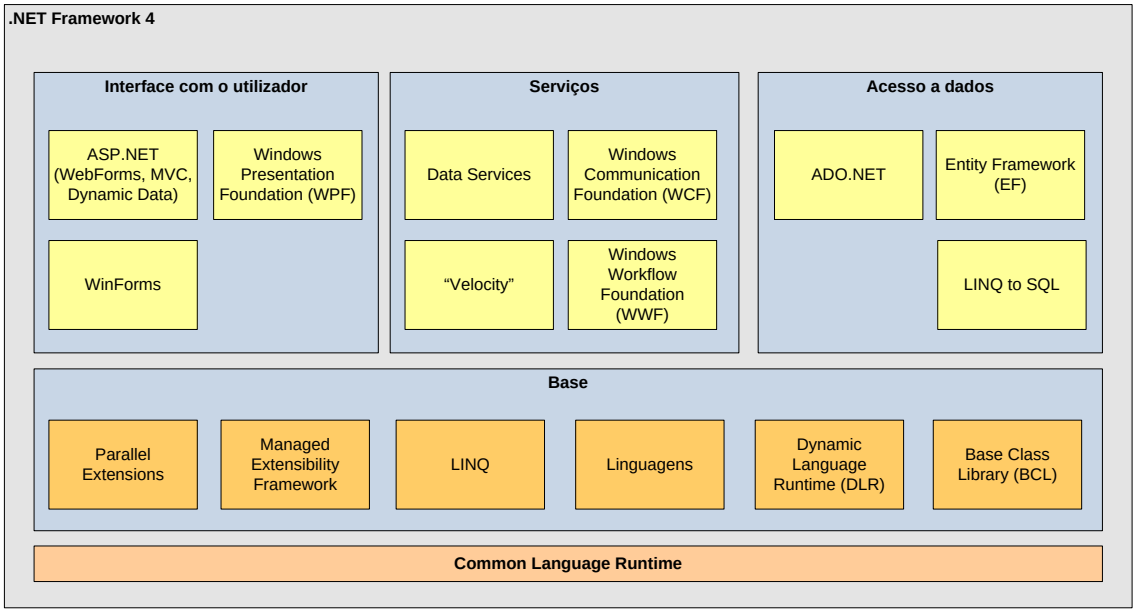

Figura 4.2 – Estrutura da *Framework* .NET

Desta forma, a *framework* .NET, disponibiliza várias ferramentas com o intuito de resolver o maior número de problemas possível quando se pretende criar uma aplicação, fomentando a flexibilidade oferecida pela CLI.

Como a oferta de ferramentas é de facto extensa, apenas algumas foram utilizadas na implementação do protótipo funcional, que serão apresentadas de seguida. Porém, o modo como foram efectivamente aplicadas será mencionado no capítulo 5.

4.3.1 Linguagem de programação: C#

A plataforma foi, na sua totalidade, implementada usando a linguagem C#. Esta linguagem, que se rege pela especificação CLI, é, como referido acima, uma das linguagens de alto-nível usadas pela plataforma .NET e foi a linguagem seleccionada para a criação de todos os componentes da plataforma. A razão deve-se à sua utilização generalizada pela empresa proponente, pelo que não foram exploradas alternativas.

Esta linguagem apresenta-se como a linguagem orientada a objectos da plataforma .NET, surgiu como a resposta da Microsoft à linguagem Java [CSha10], tendo sido apresentada pela primeira vez no ano 2000.

Segundo a especificação da linguagem [CSha10], um dos seus principais objectivos é a sua utilização para o desenvolvimento de componentes para serem implementados em sistemas distribuídos, o que vai ao encontro dos objectivos da plataforma de monitorização.

4.3.2 Serviços

Um dos aspectos fundamentais da plataforma de monitorização é a comunicação entre entidades, nomeadamente na troca de mensagens relativas a pedidos de registo, relatórios ou ainda pedidos de acções. Dado que a plataforma se trata de um sistema distribuído, a selecção para implementação desta funcionalidade recaiu sobre a solução da plataforma .NET para responder a esta necessidade. Consequentemente, surge o *Windows Communication Foundation* (WCF).

4.3.2.1 Windows Communication Foundation

O conceito de WCF é, no fundo, o mesmo que *Web Services*, fornecer serviços que permitem que clientes possam aceder a funcionalidades que só estão disponíveis em locais remotos. Uma vez que se trata de um universo complexo e bastante alargado, apresentar-se-ão apenas os aspectos mais relevantes, dado que não se enquadra no objectivo deste documento.

WCF surgiu como uma solução da Microsoft para capacitar o consumo de fornecedores de serviços sem restrições de protocolos e formatos de mensagens, o que acontece com os *Web Services* [BCR08]. Este facto permite uma maior liberdade para o desenvolvimento de aplicações que necessitem de serviços para responder aos seus requisitos.

Assim, no cômputo geral, um serviço é um conjunto de *endpoints*, pontos de acesso que expõem funcionalidades acessíveis por clientes que desejem usufruir da sua oferta. Assim, os clientes enviam mensagens para estes *endpoints*. Os serviços esperam por mensagens em endereços especificados pelos *endpoints* de acordo com um formato previamente acordado entre clientes e serviço [BCR08].

Figura 4.3 – Comunicação entre cliente e serviço WCF

Para que um cliente consiga aceder ao serviço, necessita de conhecer o ABC do *endpoint* com o qual deseja comunicar. Desta forma:

- "A" (*Address*), o local para onde as mensagens deverão ser enviadas para que o *endpoint* de destino as receba;
- "B" (*Binding*), o modo como as mensagens serão enviadas. Define o tipo de canal de comunicação que será criado para que as mensagens sejam transaccionadas. Um canal possui diversos elementos, mas um dos mais importantes é o protocolo de transporte (HTTP, TCP, entre outros);
- "C" (*Contract*), o que pode ser comunicado para que o serviço entenda as mensagens recebidas. Consiste no conjunto de operações que o serviço disponibiliza no *endpoint* que expõe e que podem ser consumidas por clientes. Este conjunto de operações designa-se por contrato, que são mapeados em métodos de classes implementados pelo *endpoint*.

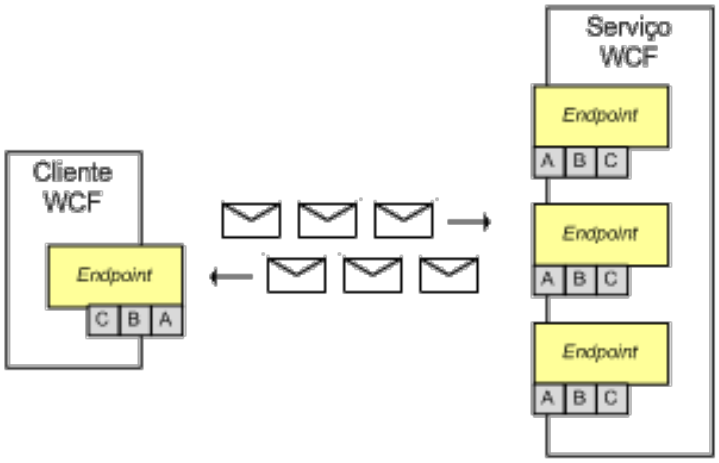

Figura 4.4 – Comunicação entre *endponts* de cliente e serviço WCF

Assim, para a definição de serviços, especificam-se os ABC dos seus *endpoints*.

Para que um serviço seja executado, será necessário que seja albergado num processo em execução num sistema operativo. Neste sentido, enquadram-se os serviços *Windows*, por exemplo.

4.3.3 Acesso a dados

Como referido na secção 3.3.1, a aplicação de monitorização usa uma base de dados relacional para armazenar as entidades principais da plataforma de monitorização e o resultado da actividade de agentes. Esta base de dados, criada em Microsoft SQL Server, que, relembrando, vai ao encontro do requisito do uso de tecnologias Microsoft para a implementação do protótipo, necessita de um meio para se conseguir escrever e obter informação da plataforma.

Um dos blocos de funcionalidades da plataforma .NET é as ferramentas disponibilizadas para acesso a dados.

4.3.3.1 Entity Framework

Framework de ORM (*Object-Relational Mapping*), abstracção que permite trabalhar com diversos Sistemas de Gestão de Bases de Dados (SGBDs) como se fossem objectos. Efectua a reunião entre duas projecções de um modelo de entidades abstracto [MSDNB10], a projecção sobre um esquema relacional de uma base de dados e a projecção sobre uma linguagem de programação orientada a objectos. Do ponto de vista do programador, esta abstracção permite actuar na base de dados como se estivesse a manipular objectos de classes de uma forma transparente, facilitando a tarefa de programar acções sobre uma base de dados. Neste sentido surge o conceito de EDM (*Entity Data Model*), o modelo conceptual, que efectua um mapeamento 1:1 entre entidades e tabelas de uma base de dados. Desta forma, um tuplo de uma tabela será representado por um objecto e as suas colunas serão propriedades desse objecto.

4.3.3.2 LINQ to Entities

Para proceder ao acesso propriamente dito, a *Entity Framework* estende um componente pré-existente na plataforma .NET, LINQ. Em suma, LINQ fornece capacidades de interrogação (*query*) às suas linguagens. No caso específico da plataforma de monitorização, LINQ *to Entities* permite o suporte LINQ para interrogações e manipulações do modelo conceptual da *Entity Framework* [MSDN10b].

4.3.4 Interface com o utilizador

A *framework* .NET fornece várias soluções para a construção de aplicações com interface gráfica. Para a plataforma de monitorização, o seu cariz gráfico conduziu à necessidade de seleccionar uma solução para a criação da interface com o utilizador e que se enquadrasse no aspecto do sistema distribuído da plataforma. Assim, a plataforma é apresentada a um utilizador como uma página *Web*.

4.3.4.1 Silverlight

Para a implementação da aplicação *Web*, a escolha recaiu em Silverlight, a implementação multi-*browser* e multi-plataforma da f*ramework* .NET para a construção e implementação de RIA (R*ich Interactive Applications*) sobre a *Web* [MSDN10c]. Uma das vantagens é a delegação de um ambiente de execução para o lado do cliente que acede à aplicação, reduzindo o grau de processamento no lado do servidor. [MSDN10d]

No ponto de vista geral, Silverlight pode ser representado pela seguinte arquitectura [MSDN10d]:

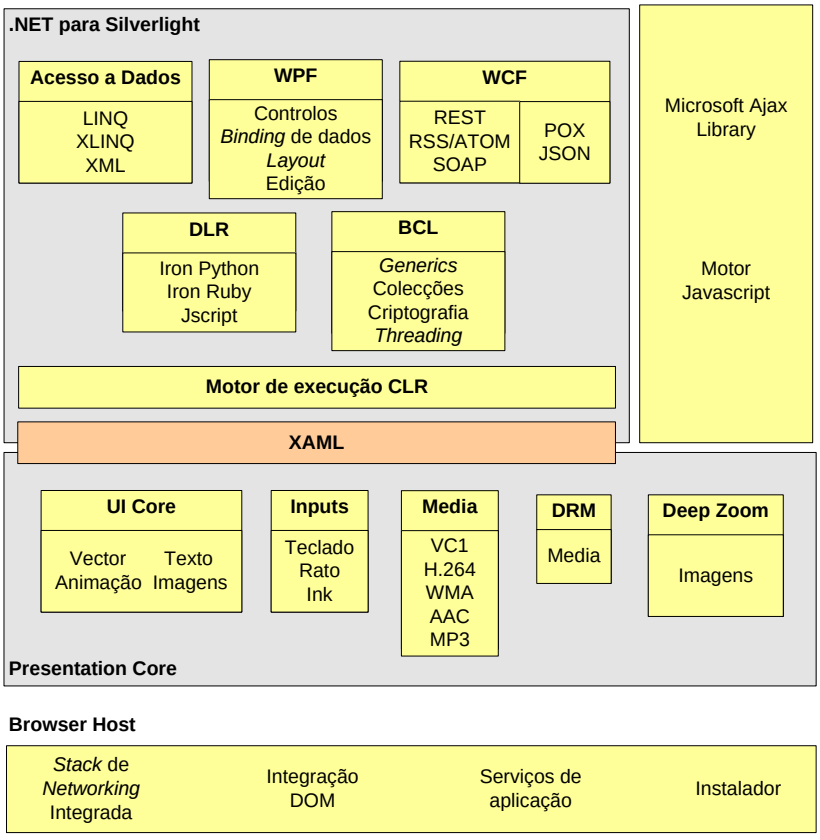

Figura 4.5 – Arquitectura da plataforma Silverlight

A plataforma Silverlight apoia-se em algumas funcionalidades da *framework* .NET, às quais combina um conjunto de componentes e serviços orientados à interacção com o utilizador (*Presentation Core*). Nestes componentes incluem-se controlos de *inputs* do utilizador, controlos UI para uso em aplicações *Web*.

A especificação do *layout* dos controlos gráficos efectua-se com o uso da linguagem XAML, uma linguagem declarativa baseada em XML, que permite a descrição de objectos e propriedades e a sua relação, em XML. Tal facto permite a separação da especificação do *layout* da apresentação da aplicação do código que gere toda a lógica de negócio inerente à interacção com o utilizador.

4.4 WCF RIA Services

Na descrição da f*ramework* .NET, é evidente que a estruturação dos blocos principais das funcionalidades da f*ramework* e os requisitos de plataforma de monitorização vão ao encontro da criação de uma aplicação multi-camada. Por um lado, temos a camada de acesso a dados, situada no lado do servidor (que aloja a base de dados), onde se encontra implementada alguma lógica de negócio. Por outro lado, temos uma aplicação Silverlight a ser executada no lado do cliente, ou seja, do lado do utilizador da plataforma de monitorização. Esta aplicação, que também implementará alguma lógica de negócio para processamento de eventos gerados pelo utilizador, constituirá a camada de apresentação. Assim, qual é o modo para unir estas duas camadas? A resposta passa pela criação de uma camada intermédia para efectuar a ligação. A Microsoft disponibiliza uma resposta para este problema, que denominou por WCF RIA Services [MSDN10f]. Para que a aplicação possa fornecer uma maior experiência de utilização ao utilizador, a lógica de negócio presente no lado do cliente deve ter conhecimento da lógica de negócio do lado do servidor. Esta situação implica uma maior manutenção, uma vez que seria necessário duplicar a lógica de negócio do lado do servidor, no cliente. Este facto suscita algumas dificuldades de manutenção, dado que possíveis alterações da lógica no servidor teriam de ser replicadas na lógica situada no cliente [MTB10]. Para evitar a tarefa de proceder a manutenção da mesma lógica em camadas diferentes, a melhor solução seria que o cliente pudesse ter acesso à lógica mais actual do servidor, sem que para isso seja necessária a replicação mencionada. Assim, consequentemente, WCF RIA Services permitem que tal aconteça. Deste modo, utiliza a f*ramework* .NET para que o desenvolvimento de lógica de negócio no servidor gere código para a camada de apresentação, como meio de sincronizar as lógicas nas duas camadas.

No caso particular da plataforma de monitorização, os WCF RIA Services actuam sobre o *Entity Model*, expondo aos clientes as entidades a que poderão aceder, isto é, expõem um serviço de domínio (*domain service*) [MSDN10g].

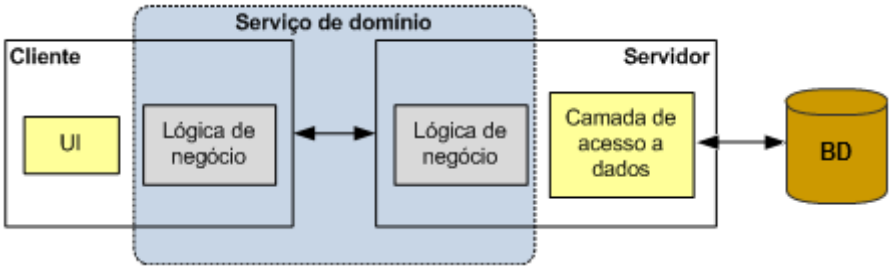

Figura 4.6 – Versão simplificada de aplicação de n camadas

4.5 Xsd2Code

Esta ferramenta, um Add-On do *Integrated Development Environment* (IDE) Microsoft Visual Studio, tem como objectivo transformar esquemas XML (XML *Schemas*) em estruturas hierárquicas de classes [X2C10], gerando código em diferentes linguagens. Esta funcionalidade permite uma leitura fácil de um ficheiro XML que esteja de acordo com o esquema esperado, procedendo à acção *Deserialize*. Assim, o conteúdo de um ficheiro XML é transformado em objectos, onde *tags* de XML serão propriedades e o seu valor será o conteúdo definido entre as *tags* respectivas. Por outro lado, permite também a acção contrária, ou seja, a transformação da estrutura hierárquica em código XML (acção *Serialize*).

4.6 Conclusão

Este capítulo apresentou as principais tecnologias usadas para a implementação de um protótipo funcional.

Todas as tecnologias apresentadas, à excepção da linguagem XML, são do universo da Microsoft, enquadrando-se numa das principais restrições da plataforma de monitorização, a utilização exclusiva de tecnologias da empresa. Desta forma, foram referidos os contextos nos quais são utilizadas, ou seja, qual o papel que representam no âmbito da arquitectura multi-
-camada da plataforma de monitorização.

O próximo capítulo apresenta os principais detalhes de implementação do protótipo funcional, onde as tecnologias foram postas em prática.

Capítulo 5

Implementação

Este capítulo apresenta os principais detalhes de implementação de um protótipo funcional, onde foram aplicadas as tecnologias referidas no capítulo anterior. Desta forma, serão apresentados os principais detalhes para cada componente da plataforma, uma vez que todos representam um papel importante para o seu funcionamento.

5.1 Aspectos gerais

A implementação do protótipo funcional foi efectuada na fase de desenvolvimento especificada para o efeito, com a duração de 10 semanas.

A abordagem tomada passou, em primeiro lugar, pela especificação conceptual de cada componente, como meio de validar a sua relevância no âmbito da plataforma de monitorização. Para além disso, foi necessário estabelecer prioridades entre os componentes a implementar, de forma a assegurar que, na sequência de componentes apresentados, aqueles cuja implementação estivesse concluída permitissem efectuar testes de modo a verificar os fluxos de mensagens. Desta forma, a sequência de implementação foi:

- Agentes;
- *Proxy*
- Modelo de entidades (*Entity Model*)
- Gestor de mensagens
- Serviço de domínio
- Aplicação de monitorização

Esta sequência foi criada considerando a importância que cada componente tinha no âmbito global da plataforma. Assim, a implementação de agentes foi a tarefa mais prioritária, uma vez que é a partir dos agentes que são originados os dados. De seguida, a criação de *proxies* para

poder implementar as tarefas de envio de dados por parte dos agentes. O passo seguinte foi a criação do modelo de entidades, pois a *Entity Framework* permite a criação de bases de dados a partir de um modelo de dados editado por um IDE. Desta forma, foi necessário implementar o Gestor de mensagens para que fosse possível a utilização da base de dados para armazenar a informação efectivamente recolhida pelos agentes. Este fluxo de implementação permitiu que fosse possível o seu teste usando como meio auxiliar consolas para apresentação de actividades. Assim sendo, o último passo consistiu na criação da aplicação de monitorização, ou seja, a camada de apresentação do lado do cliente. Porém, visto que seria necessário o teste à lógica de negócio neste lado da plataforma, foi necessária a implementação da camada intermédia simultaneamente.

Para efeitos de teste, os serviços Windows que representam agentes, *proxies* e gestores de mensagens geram eventos que são registados no sistema operativo. O sistema operativo utilizado para efeitos de desenvolvimento e teste foi o Microsoft Windows 7 *Enterprise*. Para consulta destes eventos, recorreu-se ao componente *Event Viewer*.

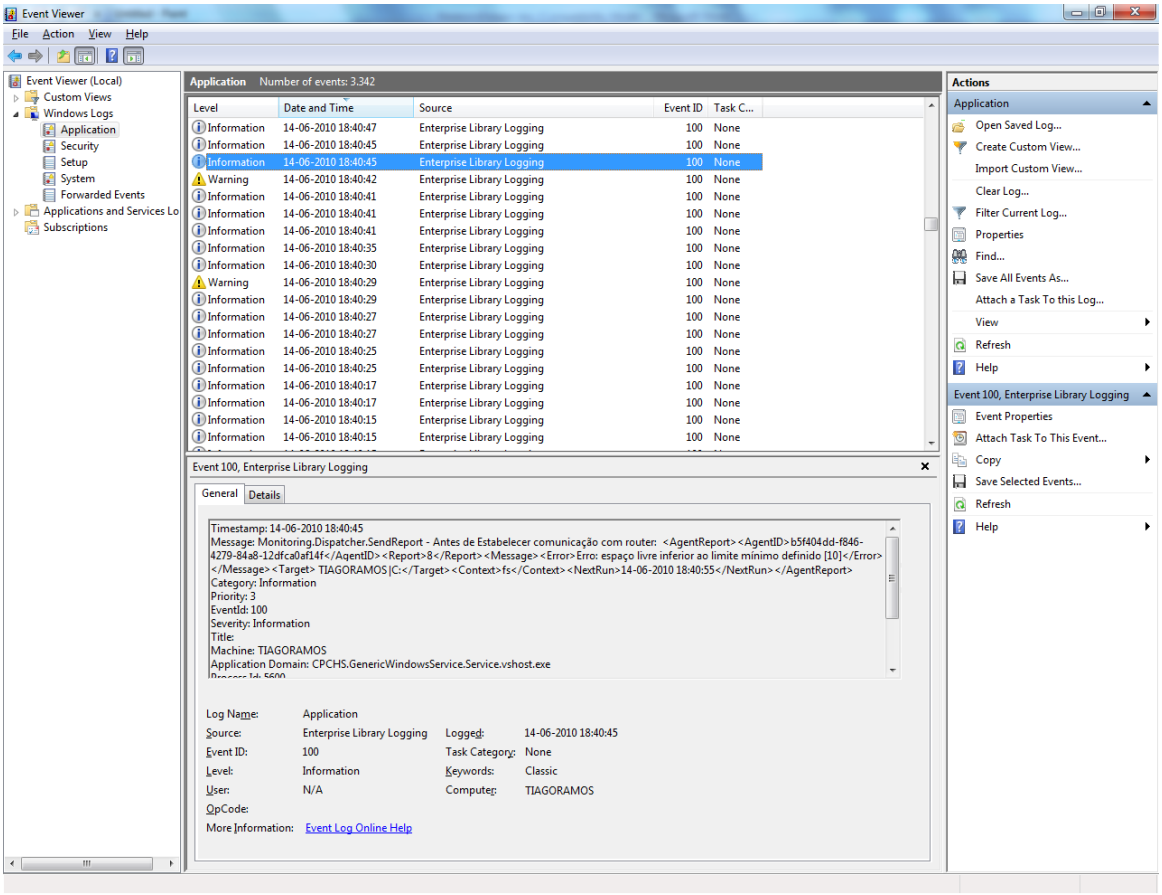

Figura 5.1 – Event Viewer

Quanto à gestão do trabalho, foi, tanto quanto possível, utilizada a abordagem FDD (*Feature Driven Development*) [Amb10], com iterações semanais.

5.2 Agentes

A especificação de agentes constituiu-se como a tarefa inicial para a implementação do protótipo funcional. Tal deve-se ao facto de, como referido em capítulos anteriores, as tarefas executadas por agentes serem as actividades que dão origem a todos os eventos da plataforma.

Esta especificação teve em linha de conta a necessidade de estes serem dotados da maior flexibilidade possível para facilitar a adição de novos agentes que poderão ser considerados no futuro.

Em suma, a execução de agentes é efectuada por um serviço Windows, que processa um ficheiro de configuração, criando os agentes que estejam especificados nesse ficheiro.

5.2.1 Configuração

A configuração de agentes é efectuada recorrendo a um ficheiro XML, onde constam as informações principais dos agentes, ou seja, a sua identificação, o seu tipo (agente de base de dados ou agente de sistema de ficheiros), os recursos sobre os quais irá actuar e as tarefas que irá efectuar nesses recursos. Porém, visto que, de facto, os tipos de recursos dependem do tipo de agente, apenas a identificação e tipo apresentam-se como o esquema comum para todos os agentes.

Basicamente o esquema para a configuração de agentes é:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="AgentsConfig">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Agent" maxOccurs="unbounded" >
          <xs:complexType mixed="true">
            <xs:sequence minOccurs="0">
              <xs:element name="Type" type="xs:string" />
              <xs:element name="ID" type="xs:string" />
              <xs:any minOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Código 5.1 – Esquema XML de configuração comum de agentes

Desta forma, o número total de agentes corresponde ao número total de blocos *Agent* que existir na configuração.

Como será evidente, o tipo de recursos e tarefas de cada tipo de agente serão diferentes. Assim, a configuração específica será efectuada após identificação do agente, o que justifica a inclusão do bloco do tipo *any*. Este bloco deverá estar em concordância com o que estiver definido em *Type*.

5.2.1.1 Agente de base de dados

Se *Type* estiver definido como *DBAgent*, o esquema do bloco de tipo *any* deve obedecer a uma estrutura definida. O esquema desta estrutura é:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="Databases"xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Databases" msdata:IsDataSet="true" msdata:Locale="en-US">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Database">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ID" type="xs:string" minOccurs="1" />
              <xs:element name="Module" type="xs:string" minOccurs="1" />
              <xs:element name="Company" type="xs:string" minOccurs="1" />
              <xs:element name="Tasks" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Task" minOccurs="0" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                         <xs:element name="TaskID" type="xs:string" minOccurs="0" />
                          <xs:element name="Type" type="xs:string" minOccurs="0" />
                          <xs:element name="Interval" type="xs:string" minOccurs="0" />
                          <xs:element name="Action" type="xs:string" minOccurs="0" />
                          <xs:element name="OnError" type="xs:string" minOccurs="0" />
                          <xs:element name="Properties" minOccurs="0" maxOccurs="1">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:any minOccurs="1" maxOccurs ="unbounded" />
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
```

```
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Código 5.2 – Esquema XML de configuração de agentes de bases de dados

Em suma, no bloco *Databases* definem-se os recursos sobre os quais o agente de base de dados irá actuar. Assim, devem existir tantos blocos *Database* quantas as bases de dados que se pretendam monitorizar. Os blocos *Module* e *Company* definem a forma de acesso à base de dados, ou seja, a partir destes dois parâmetros é obtida a sua *string* de conexão. Para tal, é utilizada uma biblioteca da empresa para efectuar este tipo de acção.

De seguida definem-se as tarefas a executar nas bases de dados. O esquema de definição das tarefas será igual para todos os tipos de agente. Desta forma, o modo como as tarefas irão variar depende das propriedades definidas, ou seja, do bloco de tipo *any*.

5.2.1.2 Agente de sistema de ficheiros

Se *Type* estiver definido como *FilesystemAgent*, de forma análoga ao anterior, o esquema da configuração do agente deve obedecer à seguinte estrutura:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="Units" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Units" msdata:IsDataSet="true" msdata:Locale="en-US">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Unit">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ID" type="xs:string" minOccurs="1" />
              <xs:element name="Tasks" minOccurs="0" maxOccurs="1">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Task" minOccurs="0" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="TaskID" type="xs:string" minOccurs="0" />
                          <xs:element name="Type" type="xs:string" minOccurs="0" />
                          <xs:element name="Interval" type="xs:string" minOccurs="0" />
                          <xs:element name="Action" type="xs:string" minOccurs="0" />
                          <xs:element name="OnError" type="xs:string" minOccurs="0" />
                          <xs:element name="Properties" minOccurs="0" maxOccurs="1">
                            <xs:complexType>
                              <xs:sequence>
```

```
                        <xs:any minOccurs="1" maxOccurs ="unbounded" />
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
```

Código 5.3 – Esquema XML de configuração de agentes de sistemas de ficheiros

As principais diferenças serão os nomes dos blocos que definem os recursos, especificamente *Units* e *Unit*. Este nomes referem-se a uma unidade de disco, que se constitui como o repositório de recursos. Neste caso, serão directórios. No caso das bases de dados, referem-se a *tablespaces*, por exemplo.

A mudança de nomes foi simplesmente um meio de tornar a leitura do ficheiro de configuração mais fácil, visto que, com a inclusão de vários agentes, a complexidade do ficheiro de configuração aumentará, e, desta forma, é possível identificar mais facilmente o tipo de recurso sob monitorização.

5.2.2 Detalhes de implementação

Como referido no capítulo 3, a execução de agentes é efectuada por um serviço Windows. Este serviço executa os fluxos apresentados no mesmo capítulo. Assim, a especificação de um serviço Windows efectua-se criando uma classe que implemente a interface *ServiceBase*, do *namespace System.ServiceProcess* [MSDN10h]. Mais especificamente, a implementação dos métodos *OnStart* e *OnStop*, ou seja, as acções a efectuar quando um serviço é iniciado ou terminado, respectivamente.

Para o serviço de agentes, o método *OnStart* instancia uma classe do tipo *AgentManager*, cuja responsabilidade é processar o ficheiro de configuração, instanciar os agentes e iniciá-los.
Em suma, as classes existentes para processamento de agentes são relacionadas segundo o diagrama de classes da figura seguinte seguinte.

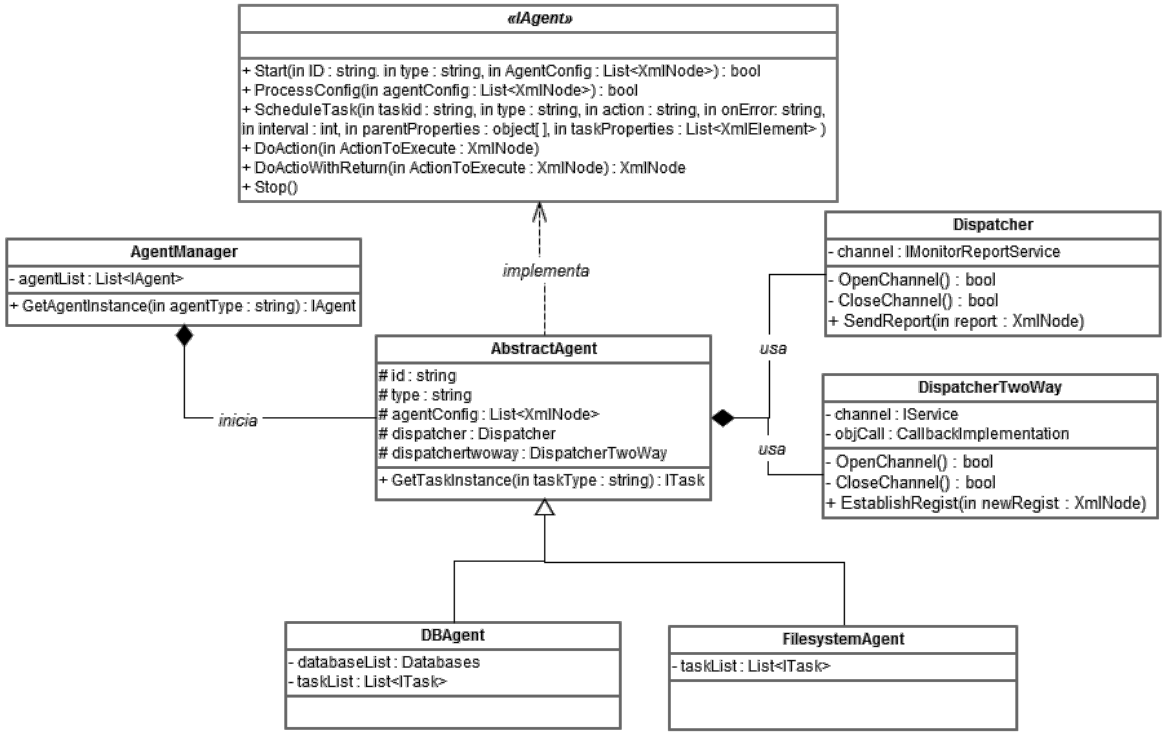

Figura 5.2 – Diagrama de classes do serviço de agentes (Agentes)

Assim, um agente, ao processar o ficheiro de configuração, analisa o campo *Type* da sua configuração e instancia um objecto do tipo que esteja definido nesse campo. Desta forma, a necessidade de instanciação de um objecto cujo tipo se desconhece à partida foi o motivo pelo qual se achou necessária a inclusão da implementação de interfaces, dado que não é possível instanciar classes abstractas (*AbstractAgent*). Assim, a instanciação de um objecto do tipo *IAgent* garante a abstracção necessária para a criação de um objecto genérico que represente um agente. De seguida, a invocação do método *Start* inicia a sua execução.

Contudo, a aplicação deve conhecer o tipo específico de agente que o objecto do tipo *IAgent* representa para que a execução do método Start seja efectuada segundo a implementação correcta. Nesse sentido surge o método *GetAgentInstance*, que utiliza a classe *System.Activator* [MSDN10i] para obter a referência de um objecto a partir de uma biblioteca. Desta forma, cada agente encontra-se definido numa biblioteca própria, que será acrescentada à plataforma de monitorização, o que permite que, para a criação de novos agentes, apenas será necessário incluir novas bibliotecas.

O fluxo de funcionamento é aquele que foi apresentado na secção 3.2.2, ou seja, após o início de funcionamento de um agente, este processa a sua própria configuração para iniciar as suas tarefas. Resumindo, o método *ProcessConfig* obtém as tarefas de um agente e *ScheduleTask* inicia-as. Para tal, a relação entre tarefas é visível a partir do seguinte diagrama de classes.

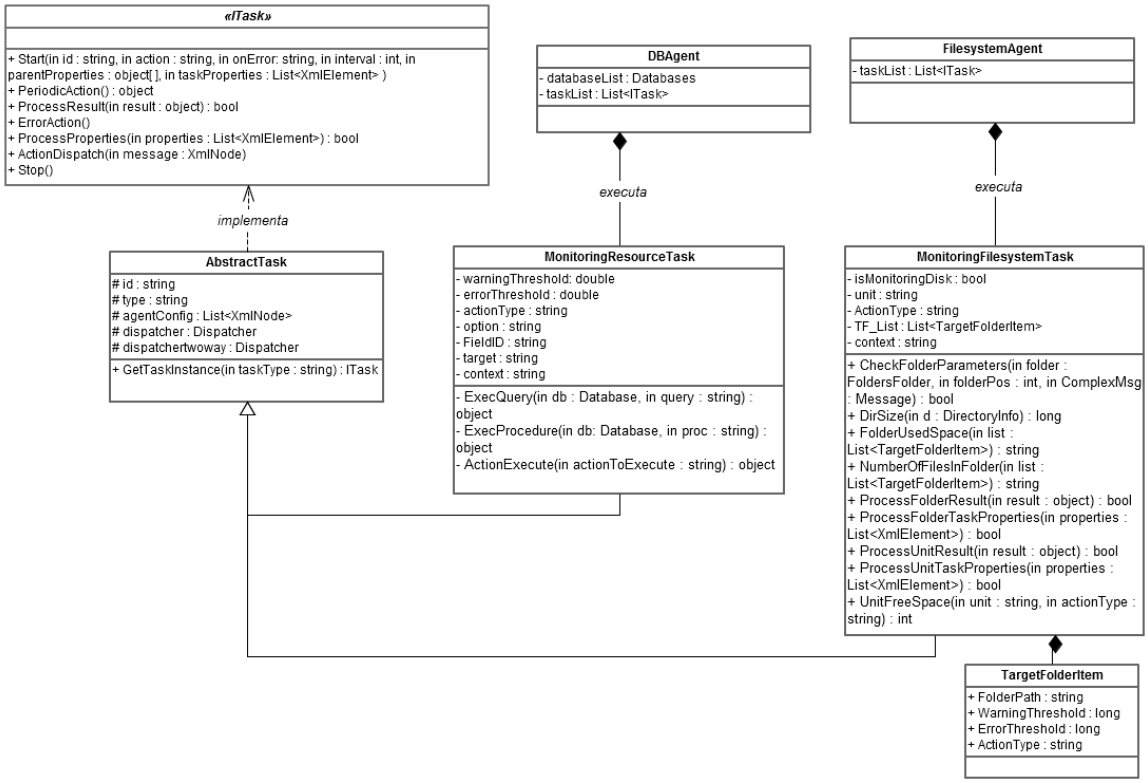

Figura 5.3 – Diagrama de classes do serviço de agentes (Tarefas)

De modo análogo, cada agente instancia um objecto consoante o que for definido no bloco *Type*, na configuração de uma tarefa (bloco *Tasks*), e invoca o seu método *Start*. Cada tarefa executa o fluxo apresentado na secção 3.2.3. Em suma, executa *ProcessProperties* para validar as suas propriedades, e, em caso de sucesso, iniciar a sua actividade. Assim, cada tarefa possui um temporizador, cujo intervalo de tempo de execução é definido pelo bloco *Interval* da sua configuração. A execução consiste na invocação do método *PeriodicAction*, que irá efectuar a tarefa periódica atribuída ao agente (bloco *Action*), o que corresponde à sua tarefa de monitorização. Segundo o fluxo mencionado no capítulo 3, o resultado da sua monitorização deverá ser processado. O resultado de *PeriodicAction* será processado por *ProcessResult*. Este processamento tem em consideração as propriedades da tarefa que se encontra em execução.

Cada tipo de tarefa terá um funcionamento diferente. A partir da Figura 5.3, a decisão foi associar, para cada agente, as suas tarefas específicas. Porém, como no cômputo geral a implementação de uma tarefa seguirá princípios comuns, optou-se por incluir uma classe abstracta, que agirá como uma super-classe de tarefas.

Assim, uma tarefa executará tanto *PeriodicAction*, *ProcessResult* e *ErrorAction*, cujas implementações serão fornecidas por cada classe representante de uma tarefa, herdeira dos membros e métodos da super-classe *AbstractTask*. Estas implementações serão dependentes do contexto onde a classe de tarefa se situa. Assim, *MonitoringResourceTask* é uma implementação

de uma tarefa associada a um agente de bases de dados. Especificamente, esta classe executa uma *query* ou um procedimento na base de dados à qual se refere, justificando, assim, a definição prévia dos recursos a monitorizar relativamente às tarefas. Assim sendo, uma tarefa terá a noção do recurso que se encontra sob monitorização, uma vez que o agente que a inicia informa-a das suas propriedades associadas, como o seu identificador, a base de dados e os dois parâmetros para acesso (*Module* e *Company*). Esta informação é representada pelo *array* denominado *parentProperties*, visível no método *Start* de *ITask*. Para além disso, foi atribuído a *ProcessResult* a maior flexibilidade possível, dado que não foi atribuído um esquema específico para propriedades de uma tarefa, como é evidente a partir da inclusão de blocos *any* na sua configuração. Assim, foram consideradas as seguintes propriedades:

- *Target*: recurso sobre o qual a tarefa se irá debruçar;
- *ActionType*: acção a executar. Pode ter um de dois valores: *Query*, para interrogações à base de dados, ou *Procedure*, para executar um procedimento;
- *FieldID:* nome de coluna utilizada para analisar o resultado. Necessário no caso de serem executas *queries*;
- *ErrorThreshold* (ET): se o resultado de *Action* (tarefa periódica) devolver um valor numérico, esta propriedade será considerada para aferir se o resultado ultrapassa um limite correspondente a uma situação anómala;
- *WarningThreshold* (WT): semelhante à propriedade anterior, mas considera situações em que se trata de uma situação a ter em atenção, mas não é uma situação anómala. Neste caso, o processamento destas duas propriedades verifica a magnitude dos valores, isto é, se ET for um valor inferior a esta propriedade, significa que valores intermédios destas as duas propriedades correspondem a situações que se enquadram naquelas a ter em atenção, pelo que valores inferiores a ET são situações anómalas. Caso estas situações se verifiquem, irá ser automaticamente executada a acção definida em *OnError*, o que poderá permitir que situações anómalas sejam automaticamente corrigidas.

  Estes aspectos aplicar-se-ão também nos casos inversos, ou seja, casos em que ET é um valor superior a WT.

Relativamente a tarefas de agentes de sistemas de ficheiros, foi criada *MonitoringFilesystemTask*. Neste caso, os tipos de processamento são idênticos à tarefa do tipo *MonitoringResourceTask*, mas as propriedades diferem. Neste sentido, foi tomada a seguinte abordagem. Ao contrário da acção periódica de uma tarefa relativa a bases de dados, que consiste na execução de uma *query* ou de um procedimento definido no bloco *Action* do ficheiro de configuração, a acção periódica de um agente de sistema de ficheiros é o nome de um método. Desta forma, a configuração de tarefas de agentes deste tipo só serão correctamente efectuadas no caso de se conhecerem à partida os métodos existentes. Neste caso, existem três:

- *NumberOfFileInFolder*: onde se verifica e devolve o número total de ficheiros que se encontrem dentro de um directório há mais de um certo tempo;
- *FolderUsedSpace*: verifica e devolve o espaço ocupado por um directório;
- *UnitFreeSpace*: espaço livre de uma unidade de disco.

Neste sentido, foram distinguidas duas abordagens, monitorização de uma unidade de disco e de directórios. O motivo passa por se considerar que as propriedades necessárias para uma unidade de disco ou para directórios são diferentes. Assim, se a acção periódica da tarefa for *UnitFreeSpace*, as propriedades necessárias serão:

- *ActionType*: método de extracção do espaço livre. *ByPercentage* se se quiser que o espaço livre seja obtido como uma percentagem; *ByValue*, se se pretender que o espaço livre seja obtido com valores absolutos;
- *ErrorThreshold*: propriedade semelhante à sua homónima, respeitante a *MonitoringResourceTask;*
- *WarningThreshold*: propriedade semelhante à sua homónima, respeitante a *MonitoringResourceTask.*

Se a acção periódica consistir nas duas restantes, a lista de propriedades deverá seguir uma estrutura pré-definida. Assim, decidiu-se, como forma de facilitar a configuração de tarefas, considerar que é possível executar a mesma acção sobre vários directórios simultaneamente, evitando a configuração individual de cada um. Desta forma, as propriedades para as acções *NumberOfFilesInFolder* e *FolderUsedSpace*, seguem o seguinte esquema:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="TargetFolders" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="TargetFolders" msdata:IsDataSet="true" msdata:Locale="en-US">
    <xs:complexType>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element name="Folder">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="FolderPath" type="xs:string" minOccurs="1" />
              <xs:element name="WarningThreshold" type="xs:string" minOccurs="1" />
              <xs:element name="ErrorThreshold" type="xs:string" minOccurs="1" />
               <xs:any minOccurs="0" maxOccurs ="unbounded" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Código 5.4 – Esquema XML de configuração das tarefas *NumberOfFilesInFolder* e *FolderUsedSpace*

A tarefa *MonitoringFilesystemTask* poderá ser executada em vários locais, uma vez que o bloco *TargetFolders* tem como objectivo a definição de vários directórios que estejam incluídos na unidade definida no bloco *Unit* do ficheiro de configuração do agente respectivo. Para cada directório, são definidas as seguintes propriedades:

- *FolderPath*: definição do caminho do directório sobre o qual se pretende obter informação;
- *WarningThreshold*: semelhante à mesma propriedade relativa a *UnitFreeSpace*;
- *ErrorThreshold*: semelhante à mesma propriedade relativa a *UnitFreeSpace*.

5.3 Base de dados

A base de dados utilizada na plataforma possui o esquema representado pelo seguinte diagrama de classes.

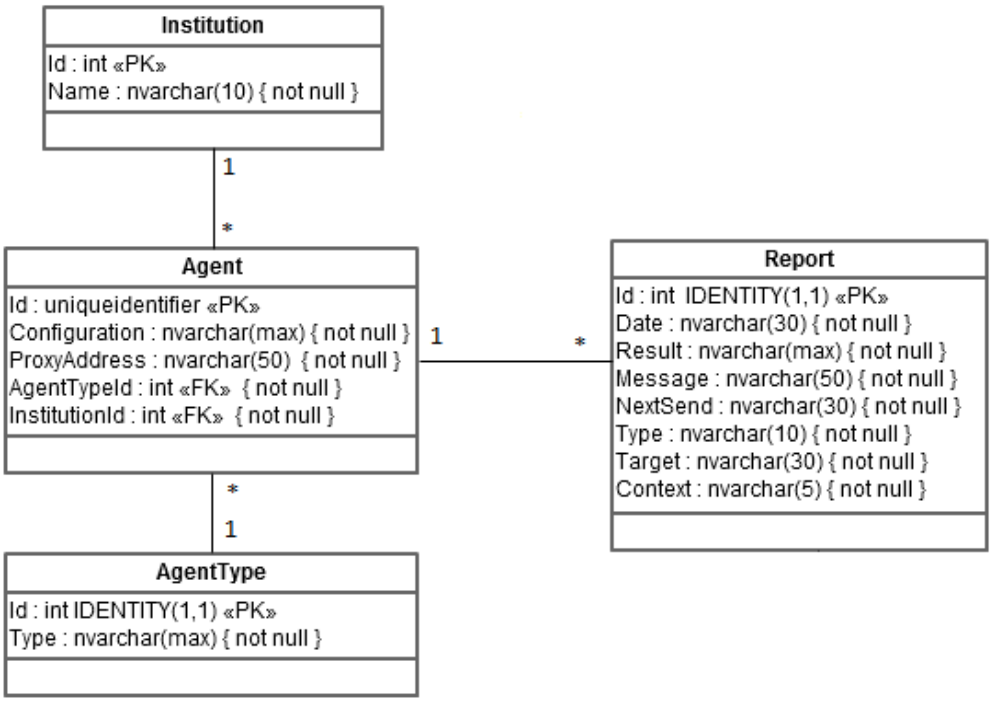

Figura 5.4 – Diagrama de classes da base de dados

De facto, verifica-se que o esquema utilizado é bastante simples. Contudo, toda a informação necessária encontra-se presente. Assim, uma instituição possui vários agentes, cada um deles representante de um tipo (base de dados ou sistema de ficheiros). Para além disso, um agente possuirá uma configuração, que, no fundo, corresponde ao código XML que define os recursos e tarefas que deverá processar. Para além disso, como será necessário aceder aos relatórios de determinados agentes, os relatórios encontram-se associados aos agentes que procederam ao seu

envio.. Estes relatórios são representados por *Report*. Os seus membros representam a informação mais importante que um relatório deve possuir, ou seja, a data em que foi recebido (*Date*), o resultado que obteve (*Result*), uma mensagem indicativa do resultado do processamento do resultado obtido (*Message*), a próxima data em que irá processar a mesma tarefa (*NextSend*) e a categorização do resultado (*Type*). Esta categorização poderá possuir quatro tipos diferentes:

- *Error*: detectou uma situação anómala;
- *Warning*: detectou uma situação que deverá ser alvo de atenção;
- *OK*: situação normal;
- *Complex*: esta categoria corresponde à situação em que um relatório refere-se simultaneamente a vários recursos. Concretamente, quando existem, por exemplo, vários directórios definidos nas propriedades de uma tarefa de monitorização de sistemas de ficheiros. Assim sendo, apresenta, para cada pasta, uma das três categorias acima mencionadas.

Quanto aos membros restantes, *Target* corresponde ao(s) recurso(s) que foi (foram) monitorizado(s) e *Context* será o âmbito em que o relatório se enquadra, ou seja, se é relativo ao resultado de monitorização de uma base de dados ou de um sistema de ficheiros

5.4 Serviços

A comunicação entre as várias entidades da plataforma de monitorização é rigorosamente necessária para que o fluxo de resultados de monitorização e de pedidos de registo de agentes, como também de pedidos para que agentes executem uma acção, seja efectuado.

Tal como referido no capítulo 4, a solução para a implementação deste aspecto da plataforma é a utilização de serviços WCF. Para o consumo de um serviço, um consumidor deverá ter conhecimento do ABC do *endpoint* que disponibiliza o serviço. Este facto é estritamente vital para que o consumo se possa efectuar. Desta forma, um serviço WCF necessita de ser albergado por um processo em execução no sistema operativo. Como agentes, *proxies* e *gestores de mensagens* irão ser executados por um serviço Windows, adequa-se perfeitamente que estes serviços alberguem os serviços WCF utilizados para a comunicação.

É neste aspecto que foi efectuada uma restrição: a impossibilidade do serviço de agentes estabelecer *endpoints*, ou seja, disponibilizar serviços WCF. Este facto indica que este serviço não disponibiliza serviços WCF, sendo apenas consumidor. A razão pela qual tal se sucede deve-se ao facto de a disponibilização de serviços WCF necessitar de abrir portas de comunicação, o que, segundo as políticas da empresa, foi estabelecido como algo desaconselhável, uma vez que o

serviço de agentes é executado em instituições clientes. Assim, foi necessário encontrar uma alternativa para se conseguir obter o fluxo pretendido. Desta forma, o estabelecimento de comunicação é sempre efectuado pela iniciativa do serviço de agentes, impedindo, assim, que haja a necessidade de disponibilizar serviços WCF.

Outro aspecto relevante diz respeito a uma das principais decisões relativamente à criação de serviços WCF, que é a escolha do elemento B, *Binding*. Este elemento define o protocolo que o canal de comunicação que irá ser aberto para o consumo de serviço e a resposta (opcional) deverão ter. Por defeito, são disponibilizados alguns protocolos [MSDN10j], como por exemplo *BasicHttpBinding*, que representa um protocolo adequado para comunicação com *Web Services*, que usa HTTP (*Hypertext Transfer Protocol*); *WSHttpBinding*, usa HTTP e adequa-se para contratos que não são *duplex* (criação de um canal adicional para resposta, ou seja, estabelecimento de comunicação bidireccional); *WSDualHttpBinding*, semelhante ao anterior, mas considera contratos *duplex*; *NetTcpBinding*, usa TCP (*Transmission Control Protocol*) como protocolo de transporte. Apesar de existirem mais soluções, apenas estes quatro foram tomados em consideração no momento da selecção de um *binding*, por se considerar que reuniam as especificações necessárias, à partida, para implementar as funcionalidades requeridas. Os dois primeiros foram rejeitados, pois não permitem o estabelecimento de canais bidireccionais [MSDN10j]. Por outro lado, segundo as especificações de *WSDualHttpBinding*, um canal bidireccional neste *binding* corresponde, no fundo, a dois canais distintos, um de saída e outro de entrada. Assim, esta opção também foi rejeitada, uma vez que, para criar um canal de entrada, seria violada a principal restrição na criação dos serviços WCF, a abertura temporária de uma porta de comunicação. Assim, a última opção foi a seleccionada, pois permite a criação de apenas um canal para comunicação bidireccional. Por outro lado, este *binding* também é aconselhável para o seguinte cenário. No mundo virtual contemporâneo, é obrigatória a existência de *firewalls* para protecção de acessos indesejados. Neste sentido, terminais onde esteja a ser executado o serviço de agentes também devem estar protegidos por *firewalls*, o que pode dificultar o consumo de serviços WCF que se encontrem disponíveis em terminais remotos. Por conseguinte, segundo as indicações da equipa criadora de WCF da Microsoft [MSDN10k], a Tabela 1 apresenta as suas recomendações:

Tabela 1 – Selecção de transporte e protocolo de troca de mensagens

Restrições de *firewall*	Servidor disponível	Servidor com *firewall* gerida	Servidor com *firewall outbound-only*
Cliente disponível	Qualquer transporte e MEP	Qualquer transporte e MEP	Não suportado
Cliente com firewall gerida	**Qualquer trasporte não-dual, MEP Duplex requer transporte TCP**	Qualquer trasporte não-dual, MEP Duplex requer transporte TCP	Não suportado
Cliente com *firewall outbound-only*	Qualquer trasporte não-dual, MEP Duplex requer transporte TCP	Qualquer trasporte não-dual, MEP Duplex requer transporte TCP	Não suportado

A situação que melhor se aplica às condições mais usuais é a que se encontra destacada na Tabela 1. Seguindo as indicações, uma vez que é necessário um protocolo de troca de mensagens - MEP (*Message Exchange Pattern*) *duplex*, visto que poderá existir um padrão de pedido-resposta no consumo de serviços, a melhor solução parece ser o *binding NetTcpBinding*, o que vai ao encontro da escolha inicial.

5.4.1 Agentes e *Proxies*

De acordo com o fluxo apresentado na secção 3.2.3, uma tarefa, após executar a sua acção periódica e processar o resultado obtido, necessita de enviar este resultado para um *proxy*. Neste ponto entra a entidade *Dispatcher*. Em suma, um objecto do tipo *Dispatcher* é o responsável pelo consumo de serviços por parte de um agente. Como referido na secção 4.3.2.1, um consumo de serviço necessita de conhecer, à partida, os ABC dos *endpoints* da(s) entidade(s) que disponibiliza(m) o(s) serviço(s) pretendido(s). Neste sentido, um agente deve conhecer os ABC dos serviços disponibilizados pelos *proxies*. O elemento *B* é conhecido, de acordo com a Tabela 1. Por outro lado, *C* também, uma vez que foram definidos na criação do protótipo da plataforma de monitorização (mais detalhes adiante), restando conhecer *A*. A definição deste elemento deverá ser efectuada no ficheiro de configuração do serviço de agentes. Não se deve confundir este ficheiro de configuração com a configuração dos agentes referida antes, pois um serviço Windows tem associado um ficheiro de configuração próprio, onde se definem opções para a sua execução, como localizações de ficheiros auxiliares, por exemplo. Desta forma, a definição dos elementos *A*

de serviços WCF disponibilizados pelo *proxy* com quem irá comunicar é uma das opções que deverão ser configuradas.

Um aspecto preponderante para que o consumo de um serviço seja efectuado com sucesso é a definição dos contratos. Um contrato consiste na identificação de uma interface que a entidade que disponibilizar o(s) serviço(s) WCF deve implementar, ou seja, a definição de um contrato indica que os métodos associados a uma interface compõem as operações que são disponibilizadas num *endpoint*. Assim, num endereço específico (elemento *A*), poderão ser consumidas várias operações relativas a um só contrato.

Analogamente, todos os serviços Windows que constam da plataforma de monitorização possuem um objecto do tipo *Dispatcher* próprio para consumo de serviços. Porém, como será evidente, a implementação de objectos deste género deverá ser individualizada, uma vez que o consumo de serviços variará consoante o serviço Windows considerado. Um *Dispactcher* deverá ter conhecimento das operações dos contratos pretendidos, uma vez que só deste modo poderá proceder ao seu consumo.

Por outro lado, importa referir outro aspecto importante. Um contrato poderá definir uma resposta, ou seja, a implementação de um padrão pedido-resposta. Para tal, a definição de interface indica que deverá existir um contrato adicional de resposta, intitulado contrato de *callback*. Neste caso, se a implementação das operações que poderão ser consumidas é da responsabilidade da entidade que as disponibiliza, a implementação das operações de um contrato de *callback* é da responsabilidade da entidade que efectuou o consumo. É neste sentido que se justificou a escolha do *binding NetTcpBinding*, visto que para disponibilizar um contrato de *callback* seria necessário a criação de um canal de comunicação adicional se se considerasse os restantes *bindings*. Com *NetTcpBinding* será utilizado o mesmo canal para consumo de serviços e disponibilização de contratos de *callback*.

A figura seguinte ilustra a relação Agente/*Proxy* relativamente à disponibilização e consumo de serviços:

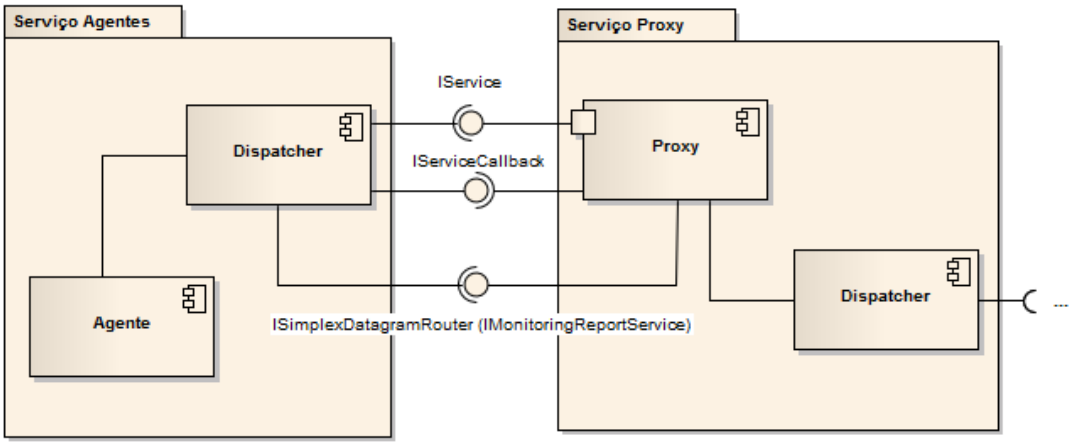

Figura 5.5 – Contratos de serviços WCF entre agentes e *proxies*

Desta forma, a comunicação entre agentes e *proxies* poderá ser especificada em duas formas, comunicação unidireccional e comunicação bidireccional. Em suma, a comunicação bidireccional consistirá em consumo de serviços WCF disponibilizados por um *proxy* e que não necessitem de resposta. Desta forma, para efectuar uma divisão clara desta duas especificações, foram criados dois tipos de *Dispatcher* no serviço de agentes. De acordo com o que se encontra apresentado na Figura 5.2, *Dispatcher* e *DisptacherTwoWay*, para comunicação unidireccional e bidireccional, respectivamente. Dado que comunicação unidireccional não necessita de contratos de *callback*, decidiu-se efectuar a divisão, pois não faria sentido uma implementação deste tipo de contratos, algo que é obrigatório no caso do contrato disponibilizado por uma entidade fornecedora de serviços WCF especificar um contrato de *callback*.

Relativamente a comunicação bidireccional, esta reflecte operações que necessitam de resposta por parte de um *proxy*. No caso do protótipo funcional, consiste no pedido de registo. Um agente só iniciará o envio de relatórios sobre a sua actividade se o consumo do serviço respectivo for efectuado com sucesso, isto é, se obtiver uma resposta afirmativa do *proxy*. Deste modo, *Proxy* disponibiliza o contrato *IService*, cuja única operação é:

- *Regist*: que recebe a configuração do agente que se prentende registar e que será inserida na base dados caso este não esteja presente.

No consumo deste serviço WCF, o serviço *proxy* regista a instância do canal de comunicação iniciado e associa-o ao agente que o estabeleceu, através do identificador que o agente envia na mensagem de registo. Desta forma, manterá o registo de um canal se for necessário comunicar com o agente posteriormente. A mensagem consiste num bloco de texto em formato XML, com o seguinte esquema:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="AgentRegist">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="InternalID" type="xs:string" minOccurs="0" />
        <xs:element name="Type" type="xs:string" minOccurs="0" />
        <xs:element name="Configuration" minOccurs="0" type="xs:string"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Código 5.5 – Esquema XML de mensagem de registo de um agente

Um agente deverá implementar o contrato de *callback* definido para *IService*, *IServiceCallback*. Este contrato implementa todas as operações necessárias para que um *proxy* comunique com um agente através da instância do canal bidireccional que guardou no momento do pedido de registo de um agente. Especificamente, estas operações são:

- *Ack_RegistProxy*: indica se o registo efectuado pelo agente teve, ou não, sucesso. Será a partir do resultado desta operação que se iniciará o processo de envio de relatórios;
- *RequestAction_Sync*: pedido de acção para que um agente execute uma tarefa a pedido. Após essa execução, o agente deverá devolver imediatamente o resultado;
- *RequestLifeSigns*: pedido a um agente para que informe se se encontra em execução. Necessário para controlo dos agentes activos por parte da aplicação de monitorização.

Por outro lado, a comunicação unidireccional consiste no envio de relatórios relativos às actividades de monitorização do agente. Neste caso, no fluxo de relatórios, um *proxy* apenas actua no seu redireccionamento para um *gestor de mensagens*, não existindo quaisquer procedimentos adicionais.

A versão 4 do WCF permite a disponibilização de *endpoints* para esse fim. Para tal, disponibiliza um *endpoint* cujo contrato é System.ServiceModel.Routing.ISimplexDatagram [MSDN10l]. Quanto ao agente, este deverá ter conhecimento do contrato disponibilizado no terminal que receberá o redireccionamento efectuado pelo *proxy*, ou seja, *IMonitorReportService*, do *gestor de mensagens*.

Quanto ao conteúdo de relatórios, estes também consistem em texto em formato XML, cujo esquema é:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="AgentReport" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="AgentReport">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="AgentID" type="xs:string" minOccurs="1" />
        <xs:element name="Report" type="xs:string" minOccurs="1" />
        <xs:element name="Target" type="xs:string" minOccurs="1" />
        <xs:element name="Context" type="xs:string" minOccurs="1" />
        <xs:element name="Message" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:choice>
              <xs:element name="OK" type="xs:string" minOccurs ="0" />
                <xs:element name="Error" type="xs:string" minOccurs="0" />
              ..<xs:element name="Warning" type="xs:string" minOccurs="0" />
                <xs:element name="Complex" minOccurs="0">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="Warning" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                      <xs:element name="OK"  type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
```

```
                    <xs:element name="Error" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:complexType>
        </xs:element>
        <xs:element name="NextRun" type="xs:string" minOccurs="1" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Código 5.6 – Esquema XML de um relatório

O significado dos blocos é em tudo semelhante ao esquema da classe *Report*, apresentado na Figura 5.4.

5.4.2 *Proxies* e Gestores de mensagens

Os *Proxies* deverão comunicar com um gestor de mensagens para que a plataforma de comunicação possa interagir com agentes. De forma análoga à comunicação com agentes, os *proxies* serão consumidores de serviços WCF disponibilizados pelo gestor de mensagens com quem se deseja comunicar. O processo é em tudo semelhante, dado que também será necessário especificar os endereços dos *endpoints* dos serviços WCF do gestor de mensagens no ficheiro de configuração do serviço *proxy*. Os *Proxies* redireccionam relatórios de agentes, assim como os pedidos de registo de agentes, como também pedidos de acção para agentes, originados na aplicação de monitorização. A relação entre *proxies* e gestores de mensagens é mais complexa, uma vez que ambos os serviços necessitam de estabelecer canais bidireccionais para disponibilizar e consumir serviços WCF para executar as suas acções. Desta forma, a Figura 5.6 apresenta, no modo geral, os contratos disponibilizados por ambos os serviços para estabelecer o fluxo Agente/Gestor de mensagens e vice-versa.

Especificamente, um gestor de mensagens disponibiliza três contratos: *IMonitorService*, *IMonitorReportService* e *IMonitorCommandService*.

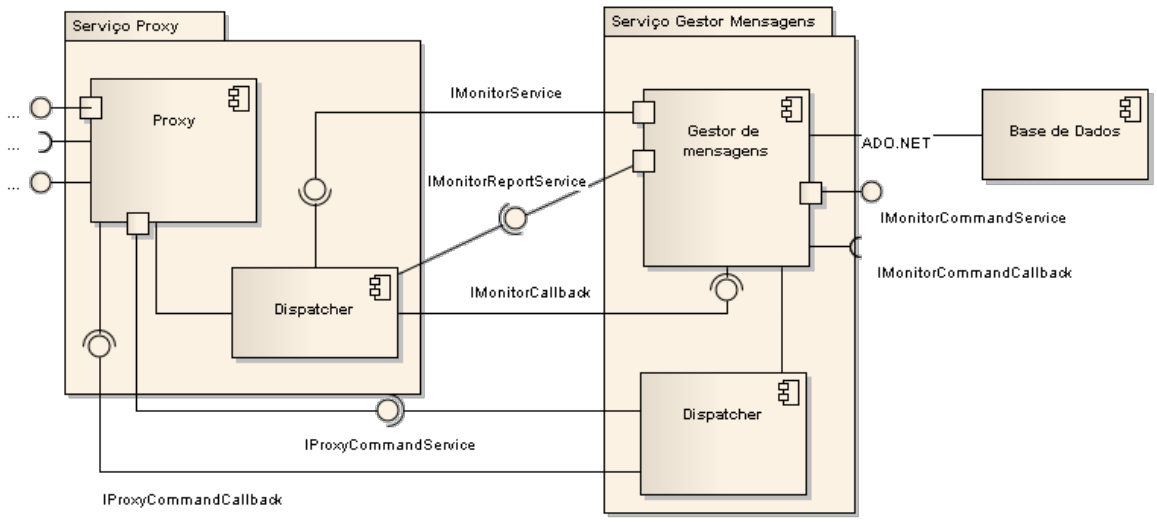

Figura 5.6 – Contratos de serviços WCF entre *proxies* e gestores de mensagens

Relativamente ao primeiro, *IMonitorService*, as suas responsabilidades consistem na recepção de pedidos de registo por parte de agentes, verificação da sua existência na base de dados e resposta desta verificação ao *proxy* que enviou o pedido. Estas acções consistem na única operação do contrato, *RoutedRegisterMessage*. Por outro lado, é necessário enviar uma resposta ao agente relativamente ao resultado da verificação do seu pedido de registo. Neste sentido, um *proxy* implementa as operações do contrato de *callback* para esta função, *IMonitorCallback*. Neste caso em particular, apenas é disponibilizada uma operação, *Ack_Regist*, que informa um agente se o seu pedido foi, ou não, processado com sucesso, através da instância de canal bidireccional que o *proxy* armazenou para o agente ao qual o pedido de registo pertence, no momento em que o agente iniciou o processo.

Quanto a *IMonitorReportService*, este deverá receber um relatório redireccionado por um *proxy* e inseri-lo na base de dados, através da sua única operação, *MonitoringReport*. Por fim, *IMonitorCommandService* é o contrato para que seja enviada uma ordem para um agente. A sua única operação é *SendOrderToAgent*. Neste sentido, um *proxy* também deve fornecer serviços WCF para que um pedido chegue a um agente. Assim, surge o contrato *IProxyCommandService*, com a operação *AgentActionOrder_Sync*. Assim, o *proxy* envia ao agente a ordem, recorrendo, também, à instância de canal bireccional estabelecida no momento do seu registo.

5.4.3 Fluxos de consumo

Após uma descrição geral dos serviços WCF disponibilizados pelos diversos serviços Windows da plataforma de monitorização, adequa-se o estabelecimento de um fluxo de consumos para proceder às principais acções. Neste caso, poder-se-ão identificar três acções principais: pedido de registo, envio de relatório e pedido de acção.

5.4.3.1 Registo

A figura seguinte ilustra o fluxo necessário para que um agente se registe na base de dados principal da plataforma de monitorização.

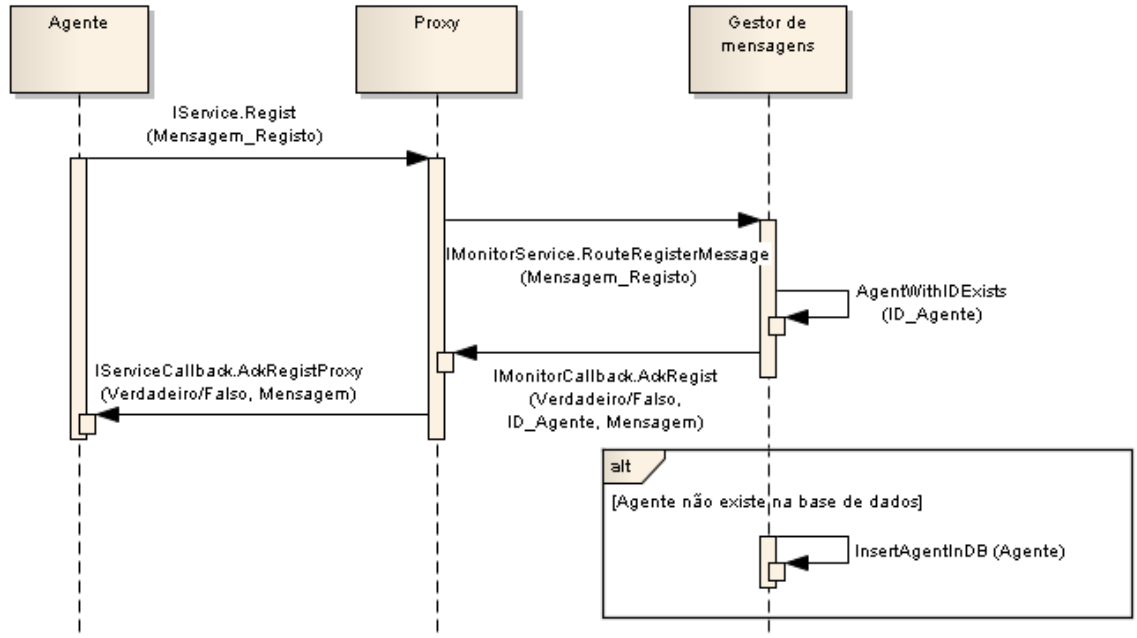

Figura 5.7 – Sequência de consumos de serviços WCF para registo de um agente

Desta forma, o fluxo apresentado na Figura 5.7 indica a existência de um padrão de pedido resposta relativamente aos pedidos de registos por parte de agentes.

A verificação da existência do agente na base de dados é efectuada analisando o conteúdo mais recente da tabela onde se armazenam informações de agentes, inquirindo a presença do identificador do agente que efectua os registos. Em caso negativo, as informações do agente que se pretende registar são inseridas na base de dados da plataforma de monitorização e o agente é notificado que foi registado (Verdadeiro).

5.4.3.2 Envio de relatório

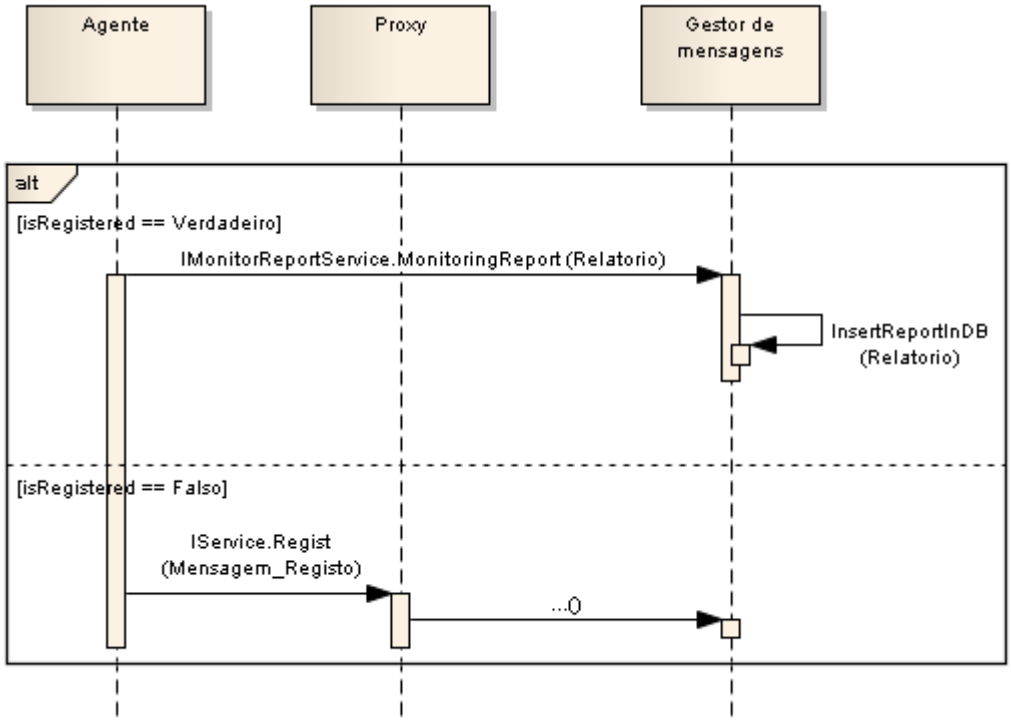

Figura 5.8 – Sequência de consumos de serviços WCF para envio de um relatório

Um agente só iniciará o envio de relatórios da(s) sua(s) tarefa(s) de monitorização se lhe for dado o aval para tal, dependendo, assim, do valor do primeiro parâmetro enviado na operação *AckRegistProxy* do contrato de *callback*. Este valor irá actualizar uma propriedade associada ao *Dispacher* de um agente, *isRegistered*. Esta propriedade actuará como uma *flag*, indicando ao agente o estado do seu registo. Assim, um agente envia relatórios se esta propriedade for verdadeira. Contudo apesar de a Figura 5.8 aparentar a existência de um consumo directo da operação *MonitoringReport*, na verdade o *proxy* intermédio efectua a operação de redireccionamento do relatório. Na implementação do agente, a noção é que existirá um acesso directo ao serviço WCF do gestor de mensagens, mas a realidade é que o consumo deste serviço WCF é na verdade efectuado pelo *proxy*.

5.4.3.3 Pedido de acção

Até ao momento foram apresentados os fluxos relativos aos consumos de serviços WCF com iniciativa em serviços de agentes. Porém, também é possível que esta iniciativa seja da aplicação

de monitorização. Assim, gestores de mensagens e *proxies* disponibilizam serviços WCF para o efeito, como referido mais acima neste capítulo. Desta forma, o fluxo tem a seguinte estrutura:

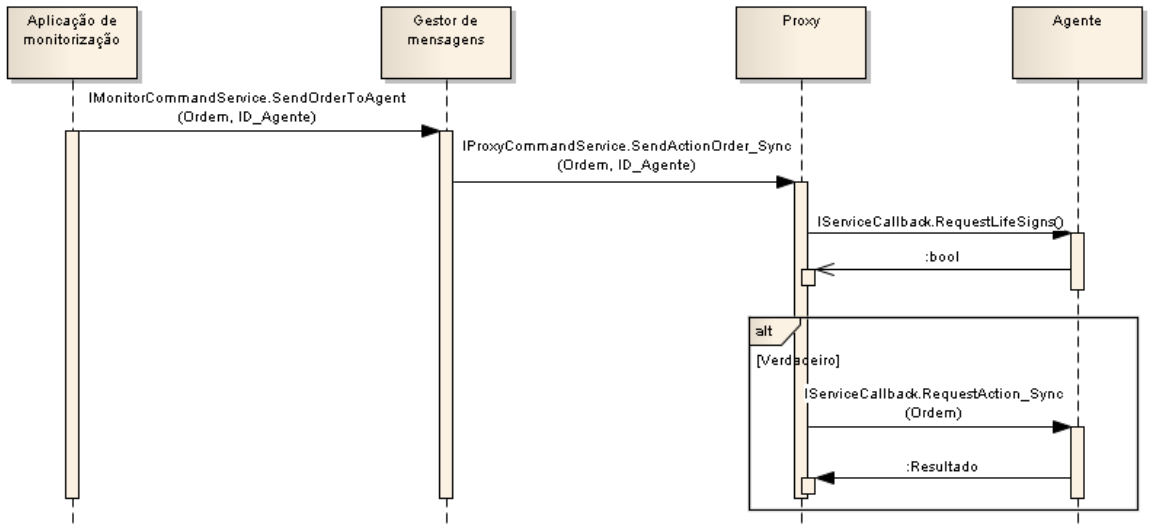

Figura 5.9 – Sequência de consumos de serviços WCF para pedido de uma acção

Por conseguinte, este fluxo inverso representa as situações onde existam pedidos para que agentes executem determinadas acções.

Como se pode notar, apenas se encontra representado um sentido neste fluxo. Todavia, o resultado deveria ser a construção de um fluxo adicional, com o sentido contrário, como meio de obter resposta do agente ao pedido. Tal não se verifica, pois este sentido não foi implementado no protótipo funcional. Assim, apenas foi construído o fluxo no sentido representado pela Figura 5.9 como meio de provar a possibilidade de processamento por parte de agentes de mensagens que surgissem a partir da aplicação de monitorização. Daí a inclusão do método *DoAction*, visível na interface *IAgent* da Figura 5.2.

É então possível à aplicação de monitorização processar informação que provém das actividades de agentes e vice-versa.

5.5 Aplicação de Monitorização

Em último lugar é necessário efectuar algumas considerações sobre a implementação da aplicação de monitorização. Esta aplicação é o componente da plataforma de monitorização com quem um *Utilizador* interage. Neste sentido, no protótipo funcional foram implementadas algumas funcionalidades para demonstrar a aplicação prática de uma interface gráfica para a plataforma de monitorização. Assim, uma interface gráfica é um indicador valioso para retirar conclusões imediatas sobre o estado de funcionamento dos recursos monitorizados, um dos

principais requisitos da plataforma. Neste sentido, a construção de indicadores visuais suficientemente claros para obter ilações instantâneas foi uma das principais preocupações. Esta construção consistiu, sobretudo, na utilização de ícones representativos de estados semelhantes aos utilizados em quase todos os sistemas operativos, nomeadamente Windows, o sistema operativo no qual a plataforma de monitorização opera. Assim, as associações da cor vermelha a estados de erro, amarela a estados de aviso e verde a situações normais são as principais linhas orientadoras, associações comuns na utilização de sistemas com forte componente gráfica.

Para a criação de vistas para o utilizador, foi utilizado o seguinte critério: atribuir maior prioridade a situações que representem situações anómalas, pois estas são aquelas que devem ser abordadas em primeiro lugar, pelo que em qualquer vista com menções a erros, estes aparecem sempre com prioridade sobre todas as outras situações e indicações.

A aplicação de monitorização possui o seguinte mapa:

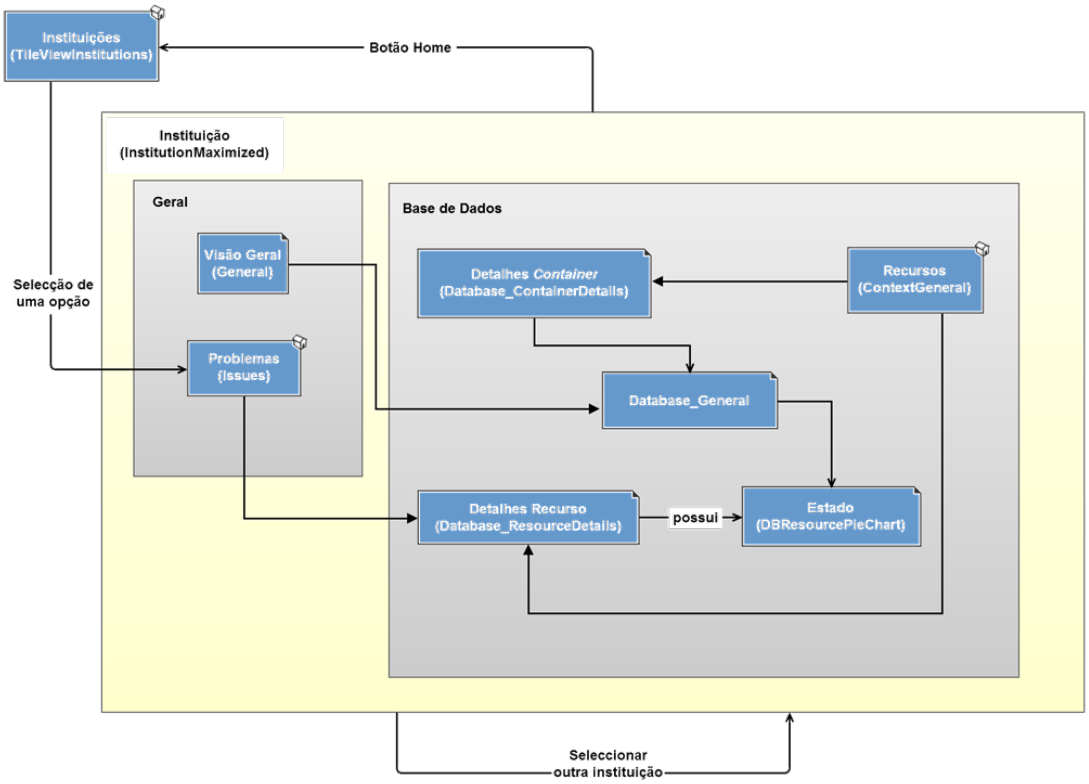

Figura 5.10 – Mapa da aplicação de monitorização

Como se pode verificar, foi efectuada uma divisão relativamente aos âmbitos que os recursos monitorizados poderão ter. Assim, cada âmbito possui um espaço próprio para representação de informação, apesar de, no protótipo funcional, apenas se ter considerado o âmbito de bases de dados.

Porém, apresentou-se como um requisito a necessidade de obter informação geral de todos os recursos monitorizados, através de uma página de entrada com os eventos mais importantes ocorridos. Nesse sentido justifica-se a inclusão do grupo *Geral*. Este grupo tem como objectivo apresentar as informações mais relevantes relativamente a todos os recursos sobre os quais existem agentes configurados para actuar. Assim, são apresentadas em primeiro lugar as situações anómalas (*Issues*), aquelas que requerem maior atenção com vista a serem resolvidas.

Outro aspecto importante foi considerar que recursos pertencem a um *container*. Em suma, quando se configura um agente de base de dados, por exemplo, define-se primeiro a base de dados e depois as tarefas que serão executadas sobre os seus elementos (tabelas, *tablespaces*, entre outros). Neste caso, uma base de dados seria um *container* e os seus elementos, os recursos.

Database_General corresponde a um painel onde se apresenta, sob forma gráfica, o estado de recuros. Este painel actua como um resumo dos estados, para sua aferição imediata. Para além disso, existem vistas com informações mais específicas de recursos (*Database_ResourceDetails*), onde se apresentam os relatórios enviados por agentes relativos a esse recurso e a data do último envio.

Para a apresentação de informação, o acesso à base de dados é uma acção vital. Mencionou--se anteriormente que o ambiente de execução da aplicação é delegado para o cliente. Assim, um cliente acede a um *browser* que deverá ter um *plug-in* de *Silverlight* instalado e o módulo da aplicação de monitorização deverá ser transferido para esse terminal. Contudo, o alojamento da aplicação de monitorização encontra-se num servidor remoto, pelo que existe a necessidade de interligar estes dois componentes. Neste ponto actuam os WCF RIA Services [MSDN10f]. A utilização desta tecnologia consistiu na criação de uma classe *MonitorDomainService*, onde são especificadas as acções que se pretende que sejam efectuadas sobre a base de dados. Esta especificação foi efectuada recorrendo a *LINQ to Entities* [MSDN10b]. Especificamente, para além das habituais operações CRUD (*Create, Read, Update, Delete*), foram necessárias operações adicionais, como consulta de configuração de agentes de uma instituição específica, que é a fonte de dados para a aplicação ter conhecimento dos seus *containers* e respectivos recursos sob monitorização, bem como do resultado dos relatórios que estes agentes foram enviando até ao momento da consulta. Assim, esta informação, o resultado dos relatórios, é o critério para que a aplicação de monitorização apresente informação sobre se um determinado recurso se encontra em estado de erro ou não. Uma vez que é necessário obter a informação mais recente o mais depressa possível, o acesso à base de dados, consoante o seu propósito, é efectuado de forma periódica, para poderem ser obtidos os últimos dados disponíveis.

A classe *MonitorDomainService*, alojada no servidor da aplicação, necessita de ser acedida no controlo da UI da aplicação de monitorização, alojada no cliente. Desta forma, considera-se o conceito de *Context*, que, basicamente, é a camada intermédia entre apresentação e acesso a

dados. Desta forma, o acesso ao *Context* da classe permite que a UI aceda aos seus métodos de modo transparente, abstraindo a separação física entre as duas camadas, facilitando a criação de aplicações em *Silverlight* com necessidade de acesso a uma base de dados, pelo que se considera que WCF RIA Service é uma solução para RAD (*Rapid Application Development*) em *Silverlight* [Bro10].

Concluindo, a aplicação de monitorização apenas foi criada para apresentar, de um modo geral, a actividade de agentes de bases de dados em formato gráfico e a utilidade das tecnologias utilizadas para responder aos requisitos físicos da plataforma. Contudo, alguns aspectos não foram considerados, como a inclusão do resultado das actividades de monitorização de agentes de sistemas de ficheiros e o controlo das funcionalidades apresentadas dependendo do tipo de utilizador (*Utilizador de implementação*, *Utilizador de suporte* e *Cliente*).

5.6 Conclusão

Neste capítulo foram apresentados alguns detalhes de implementação de um protótipo funcional criado para justificar a aplicabilidade da abordagem tomada para desenvolver o objectivo principal do trabalho, a criação de uma plataforma de monitorização. Assim, foram descritos os seus principais componentes e o modo como foram criados, bem como a descrição da sua relação para a implementação dos principais fluxos de dados.

Contudo, não foram implementadas todas as funcionalidades necessárias para a conclusão do desenvolvimento da plataforma.

Capítulo 6

Conclusões e Trabalho Futuro

Algumas considerações finais devem ser efectuadas relativamente ao desenvolvimento da plataforma de monitorização. Neste capítulo efectua-se uma análise da satisfação dos objectivos propostos no desenvolvimento do trabalho e as perspectivas de trabalho futuro.

A plataforma de monitorização surge na sequência da necessidade de diminuir o tempo ocupado por colaboradores da empresa na monitorização dos recursos utilizados nos ambientes das suas instituições clientes. Esta monitorização é, de facto, importante para os objectivos de negócio da empresa, uma vez que o funcionamento das suas soluções deve ser dotado da maior fiabilidade possível. Como meio de garantir o estado normal do funcionamento destes recursos, devem ser recolhidos dados de forma a avaliar o mais rapidamente possível este estado e actuar de forma célere, caso tal seja necessário. Para tal, a plataforma de monitorização foi estruturada sob a forma de uma *framework*, através da criação de alguns componentes gerais que poderão ser interligados para se atingir o objectivo de monitorizar recursos. Em suma, estes componentes são: agentes, que recolhem, avaliam, enviam informação sobre recursos e executam acções que lhes sejam fornecidas; *proxies*, que recebem e redireccionam informação de agentes para gestores de mensagens e vice-versa; gestores de mensagens, que recebem informações de agentes, redireccionadas por *proxies*, inserindo-as na base de dados, e enviam informações a agentes, através do contacto com *proxies*; base de dados, onde se armazenam os aspectos relevantes da plataforma, como instituições cliente, os seus agentes e as informações que estes vão enviando ao longo da sua execução; e aplicação de monitorização, que apresenta graficamente as informações recolhidas por agentes.

Estes componentes foram criados tendo em conta que a extensão e/ou alteração das suas funcionalidades é deixada nas mãos dos responsáveis pela sua aplicação. Desta forma, o caso mais evidente é a estruturação de agentes, responsáveis por actuar sobre recursos que compõem interfaces e recolher dados sobre o seu funcionamento. Os seus âmbitos de acção não foram restringidos uma vez que não foram definidos de forma clara quais os tipos de recursos sobre os

quais iriam actuar. Apesar de, na prática, terem sido restringidos alguns passos na ligação entre componentes, como o seu fluxo de registo, o modo como estes efectuam a sua acção periódica depende exclusivamente da forma como são configurados. Assim sendo, a estruturação de agentes é, de facto, o ponto mais importante relativamente à flexibilidade da plataforma. Deste modo, para estender a sua funcionalidade, utilizando o esquema de configuração apresentado, basta criar uma nova biblioteca onde se definem as acções específicas do agente, ou seja, as suas tarefas, acrescentando-a de seguida ao conjunto de bibliotecas utilizadas pelo serviço Windows onde os agentes são efectivamente executados. Por outro lado, esta indefinição quanto aos âmbitos de execução dificultou a criação da plataforma de monitorização, mas o facto de ter sido estruturada para considerar cada âmbito individualmente indica que se possui liberdade suficiente para ser estendida.

Assim, o protótipo funcional da plataforma de monitorização permite a recolha de dados de recursos reais e a sua apresentação no formato gráfico, pelo que, no cômputo geral, o objectivo primordial do trabalho foi atingido. Para além disso, o principal aspecto que que poderá ser um indicador de inovação é a criação do protótipo com a utilização do estado da arte das tecnologias mencionadas.

Contudo, alguns aspectos menos positivos verificam-se. O principal destaque que deve ser efectuado é a falta de testes da plataforma nos locais onde se prevê a sua utilização efectiva, pelo que a eficácia absoluta da especificação da plataforma carece ainda de alguma prova. Contudo, tal deve-se, sobretudo, à necessidade da plataforma conter funcionalidades suficientes para que seja viável a sua instalação em locais onde irá funcionar em situações reais, o que, de facto, ainda não se verifica na prática. Especificamente, a falta de controlo sobre permissões no acesso às funcionalidades da plataforma, como também a apresentação gráfica de informação da actividade de agentes de sistemas de ficheiros. Porém, este facto apresenta-se como deliberado, pois apenas foi efectuada a tentativa de interligar todos os componentes e não aprimorar de modo definitivo as suas funcionalidades, pelo que o objectivo princiapal foi atingido. Por outro lado, estas funcionalidades ainda se encontram em aberto, pelo que seria impossível o seu desenvolvimento definitivo.

Desta forma, prevê-se a continuidade do desenvolvimento da plataforma, onde as actividades mais prementes passam, principalmente, por concluir o desenvolvimento dos agentes de base de dados e sistemas de ficheiros, adicionando funcionalidades ainda ausentes, nomeadamente acções a efectuar a partir da selecção de opções na aplicação de monitorização. De facto, um agente consegue receber mensagens criadas remotamente, mas, na prática, não efectua qualquer acção. Para além disso, o desenvolvimento do controlo de permissões permitirá a instalação efectiva dos componentes da plataforma necessários para monitorizar recursos em locais reais, especificamente agentes e *proxies*. Para além disso, poderão ser incluídas outras tarefas

associadas aos dois tipos de agentes, visto que, por exemplo, sobre um directório de um sistema de ficheiros poderão ser efectuadas diversas acções adicionais que não foram consideradas. A estruturação dos agentes deste tipo permite que a inclusão de novas funcionalidades se efectue sem grande esforço, o mesmo aplicando-se a agentes de bases de dados.

Uma vez que âmbitos adicionais ainda não se encontram definidos, a criação de agentes de novos tipos ainda se encontra em aberto, o que se apresenta como um aspecto a considerar seriamente no futuro.

Referências

[Amb10] "*Feature Driven Development (FDD) and Agile Modeling*". Scott W. Ambler. Ambysoft. Web. 16 Mar. 2010. <http://www.agilemodeling.com/essays/fdd.htm>

[BCR08] Bowen, Chris, Richard Crane, e Steve Resnick. *Essential Windows Communication Foundation (WCF): For .NET Framework 3.5 (Microsoft .NET Development Series).* New York: Addison-Wesley Professional, 2008.

[Bro10] "*Enterprise Patterns with WCF RIA Services*". Michael D. Brown. Microsoft Developer Network Magazine. Web. 29 Abr. 2010.
<http://msdn.microsoft.com/en-us/magazine/ee336308.aspx>

[Chu07] Chu, Yu-Chi. "*A Multi-layered System Architecture for Environmental Monitoring Data Management – Taiwan's Experience*". EnviroInfo 2007 Conference. Systems Research Institute, Polish Academy of Sciences, Varsóvia, Polónia. 14 Set. 2007.

[CLI10] "*Introduction to the Common Language Infrastructure*". C# Online.NET. Web. 5 Abr. 2010. <http://en.csharp-online.net/Introduction_to_the_Common_Language_Infrastructure>.

[CS09] Comuzzi, Marco, e George Spanoudakis. "*A Framework for Hierarchical and Recursive Monitoring of Service Based Systems*." Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services (2009): 383-388.

[CSha10] "C# Language Specification". C# Online.NET. Web. 10 Mar. 2010.
<http://en.csharp-online.net/CSharp_Language_Specification>.

[DBAD10] "*Database - Advantages & Disadvantages*". Web. 10 Mar. 2010. <http://www.cl500.net/pros_cons.html>.

[GG09] Ghezzi, Carlo, e Sam Guinea. "*Run-Time Monitoring in Service-Oriented Architectures*". Test and Analysis of Web Services. New York City: Springer Berlin Heidelberg, 2009. 237-264.

[GHS09] "*Glintt - Saúde*". Glintt. Web. 18 Nov. 2009. <http://www.pararede.com/sectores/saude>.

[Hype09a] "*Open Source Systems Monitoring and Management Software | Hyperic*". Systems Monitoring, Server Monitoring & Systems Management Software | Hyperic. Web. 26 Nov. 2009. <http://www.hyperic.com/products/open-source-systems-monitoring.html>.

[IBM10] "*IBM - WebSphere Business Monitor*". IBM. Web. 26 Jan. 2010. <http://www-01.ibm.com/software/integration/wbimonitor/features/?S_CMP=wspace>.

[Info09] "*Definição de monitorizar*." Infopédia - Dicionários e Enciclopédia em língua portuguesa. Web. 20 Dez. 2009. <http://www.infopedia.pt/lingua-portuguesa/monitorizar>.

[Mang05] Mangina, Eleni. "*Intelligent agent-based monitoring platform for applications in engineering*". International Journal of Computer Science and Applications 2.1 (2005): 38-48.

[MMM01] Mangina, Eleni, S. D. J. McArthur, e Jim R. McDonald. "*COMMAS (COndition Monitoring Multi-Agent System)*." Autonomous Agents and Multi-Agent Systems 4.3 (2001): 279 - 282.

[MS04] Mahbub, Khaled, e George Spanoudakis. "*A framework for requirents monitoring of service based systems*." Proceedings of the 2nd international conference on Service oriented computing (2004): 84 - 93.

[MS06] Mahbub, Khaled, e George Spanoudakis. "*Non-intrusive monitoring of service-based systems*". International Journal of Cooperative Information Systems 15.3 (2006): 325 - 358.

[MS09] Mahbub, Khaled, e George Spanoudakis. "*Monitoring WS-Agreements: An Event Calculus-Based Approach*". Test and Analysis of Web Services. New York City: Springer Berlin Heidelberg, 2009. 265-306.

[MSDN10a] "*Introduction to Windows Service Applications?"*. Microsoft Developer Network. Web. 27 Mar. 2010. <http://msdn.microsoft.com/en-us/library/d56de412%28VS.80%29.aspx>.

[MSDN10b] "*LINQ to Entities*". Microsoft Developer Network. Web. 17 Abr. 2010. <http://msdn.microsoft.com/en-us/library/bb386964.aspx>.

[MSDN10c] "*Silverlight Overview*". Microsoft Developer Network. Web. 18 Abr. 2010. <http://msdn.microsoft.com/en-us/library/bb404700%28v=VS.95%29.aspx>.

[MSDN10d] "*Silverlight Architecture*". Microsoft Developer Network. Web. 18 Abr. 2010. <http://msdn.microsoft.com/en-us/library/bb404713%28VS.95%29.aspx>.

[MSDN10e] "*XAML Overview (WPF)*". Microsoft Developer Network. Web. 18 Abr. 2010. <http://msdn.microsoft.com/en-us/library/ms752059.aspx>.

[MSDN10f] "*WCF RIA Services*". Microsoft Developer Network. Web. 29 Abr. 2010. <http://msdn.microsoft.com/en-us/library/ee707344%28VS.91%29.aspx>.

[MSDN10g] "*Middle Tier*". Microsoft Developer Network. Web. 20 Nov. 2010. <http://msdn.microsoft.com/en-us/library/ee707348%28v=VS.91%29.aspx>.

[MSDN10h] "*ServiceBase Class (System.ServiceProcess)*". Microsoft Developer Network. Web. 20 Abr. 2010. <http://msdn.microsoft.com/en-us/library/system.serviceprocess.servicebase.aspx>

[MSDN10i] "*Activator Class (System)*". Microsoft Developer Network. Web. 20 Abr. 2010. <http://msdn.microsoft.com/en-us/library/system.activator.aspx>

[MSDN10j] "*Configuring System-Provided Bindings*". Microsoft Developer Network. Web. 27 Abr. 2010. <http://msdn.microsoft.com/en-us/library/ms731092.aspx>

[MSDN10k] "*Working with NATs and Firewalls*". Microsoft Developer Network. Web. 27 Abr. 2010. <http://msdn.microsoft.com/en-us/library/ms731948.aspx>

[MSDN10l] "*A Developer's Introduction to Windows Communication Foundation 4*". Microsoft Developer Network. Web. 28 Abr. 2010.
<http://msdn.microsoft.com/en-us/library/ee354381.aspx>

[MSDNB10] "*Why use Entity Framework?*". Danny Simmons. MSDN Blogs. Web. 16 Abr. 2010. <http://blogs.msdn.com/b/dsimmons/archive/2008/05/17/why-use-the-entity-framework.aspx>.

[MTB10] "*Silverlight and WCF RIA Service (1 - Overview)*". Mike Taulty. Mike Taulty's Blog. Web. 29 Abr. 2010. <http://mtaulty.com/CommunityServer/blogs/mike_taultys_blog/archive/2010/05/04/silverlight-and-wcf-ria-services-1-overview.aspx>

[NET10] "*Microsoft .NET Framework*". Microsoft Corporation. Web. 02 Abr. 2010. <http://www.microsoft.com/net>.

[Robi05] Robinson, William N.. "*A requirements monitoring framework for enterprise systems*". Requirements Engineering 11.1 (2005): 17-41.

[SAP10] "*SM300 - Business Process & Interface Monitoring*." SAP. Web. 26 Jan. 2010. <www12.sap.com/services/education/catalog/globaltabbedcourse.epx?context=[[|SM300||1||G|]]|> .

[SCOM09] "*System Center Operations Manager 2007 (SCOM) - Platform Monitoring*". Microsoft Corporation. Web. 18 Dez. 2009. <http://www.microsoft.com/systemcenter/operationsmanager/en/us/default.aspx>.

[SCOM10] "*System Center Operations Manager 2007 R2 SDK*". Microsoft Corporation. Web. 12 Mar. 2010. <http://msdn.microsoft.com/en-us/library/cc268402.aspx>.

[SMS99] Sheehan, Timothy J., Allen D. Malony, and Sameer S. Shende. "*A Runtime Monitoring Framework for the TAU Profiling System*". Proceedings of the Third International Symposium on Computing in Object-Oriented Parallel Environments (1999): 170 - 181.

[W3C10] "*Extensible Markup Language (XML)*". World Wide Web COnsortium. Web. 07 Mar. 2010. <http://www.webopedia.com/TERM/I/interface.html>.

[Webo09] "*What is interface?*". Webopedia Computer Dictionary. Web. 18 Nov. 2009. <http://www.webopedia.com/TERM/I/interface.html>.

[X2C10] "*Xsd2Code .net class generator from XSD schema*". Codeplex. Web. 22 Abr. 2010. <http://xsd2code.codeplex.com>

[Yvas10] "*Overview Of .Net 4.0*". Sanjay Yvas. Web. 02 Abr. 2010. <http://www.slideshare.net/rsnarayanan/overview-of-net-40-sanjay-vyas>.

[YYS09] Yao, Yachuan, Yi Yao, e Hong Song. "*The Remote Monitoring System Based on the OPC Technology*". International Workshop on Intelligent Systems and Applications, 2009. ISA 2009. (2009): 1-3.

Anexos

Anexo A Diagrama de classes completo

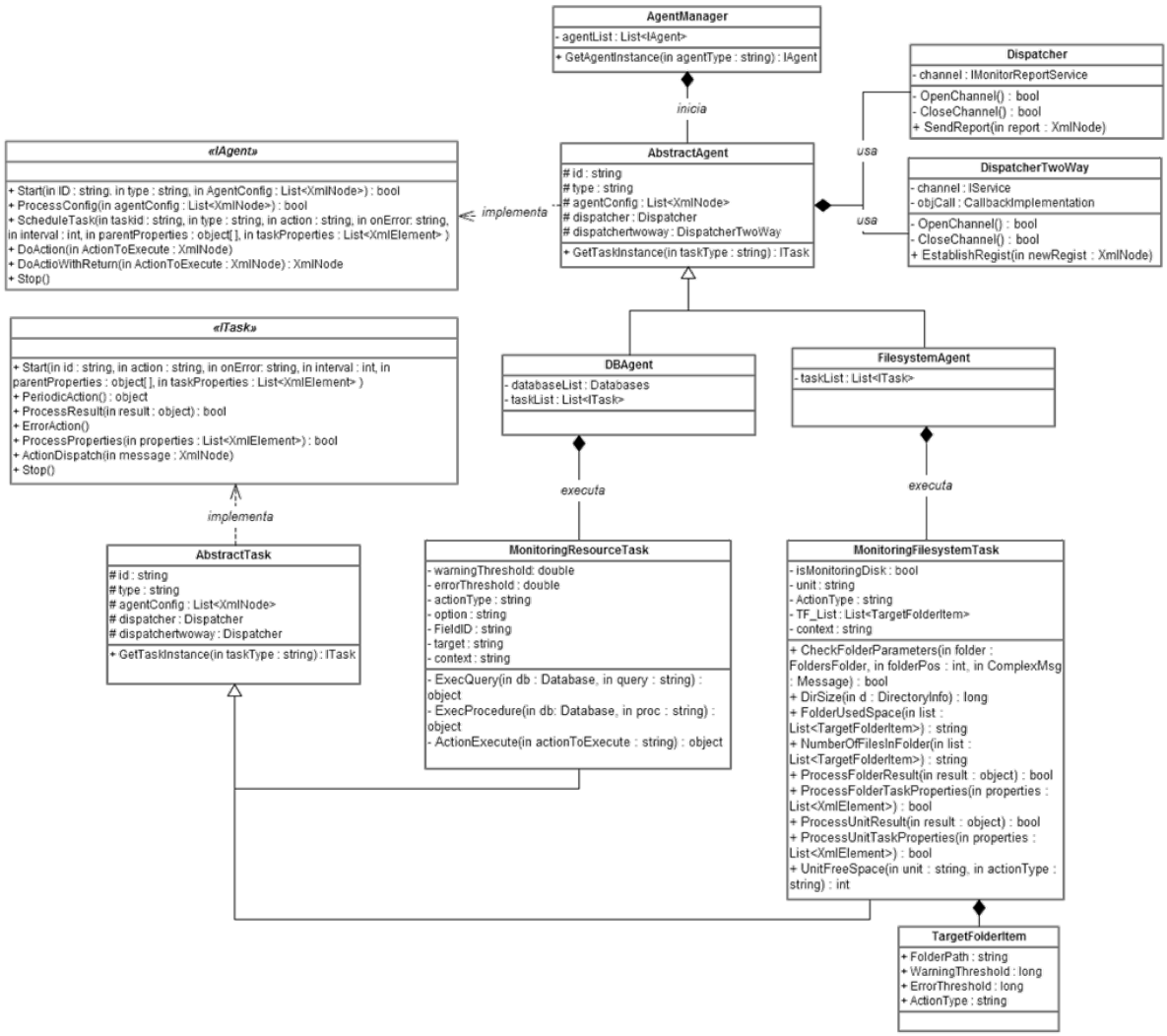

Figura A.1 – Diagrama de classes completo do serviço de agentes

Anexo B Aplicação de Monitorização (Imagens)

Figura B.1 – Página de entrada da aplicação de monitorização (*TileViewInstitutions*)

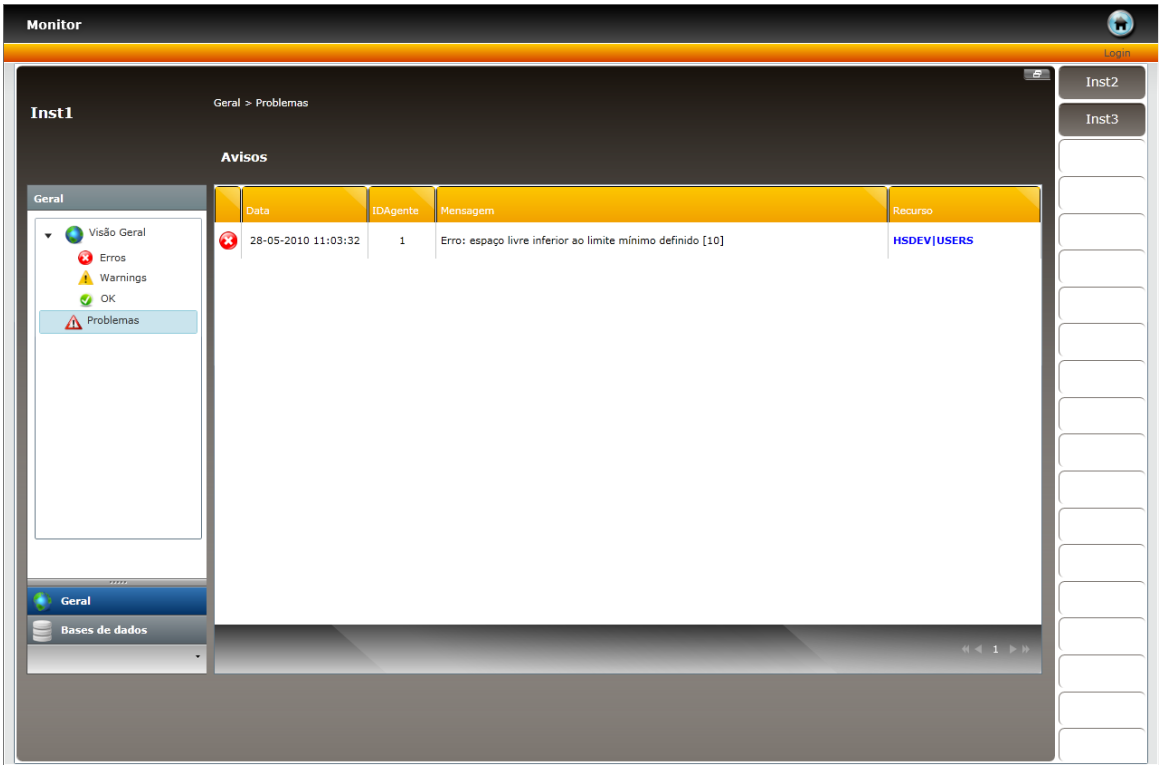

Figura B.2 – Página de listagem de problemas (*Issues*)

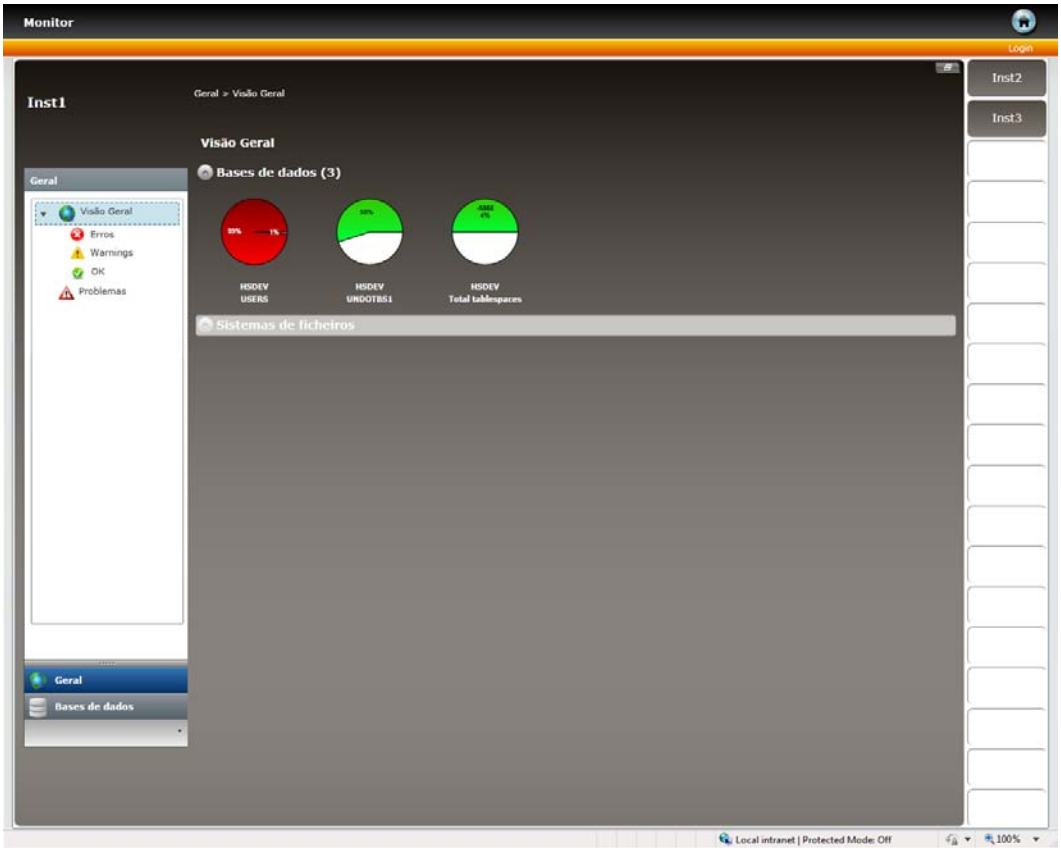

Figura B.3 – Página do estado geral de todos os recursos sob monitorização de uma instituição (*General*)

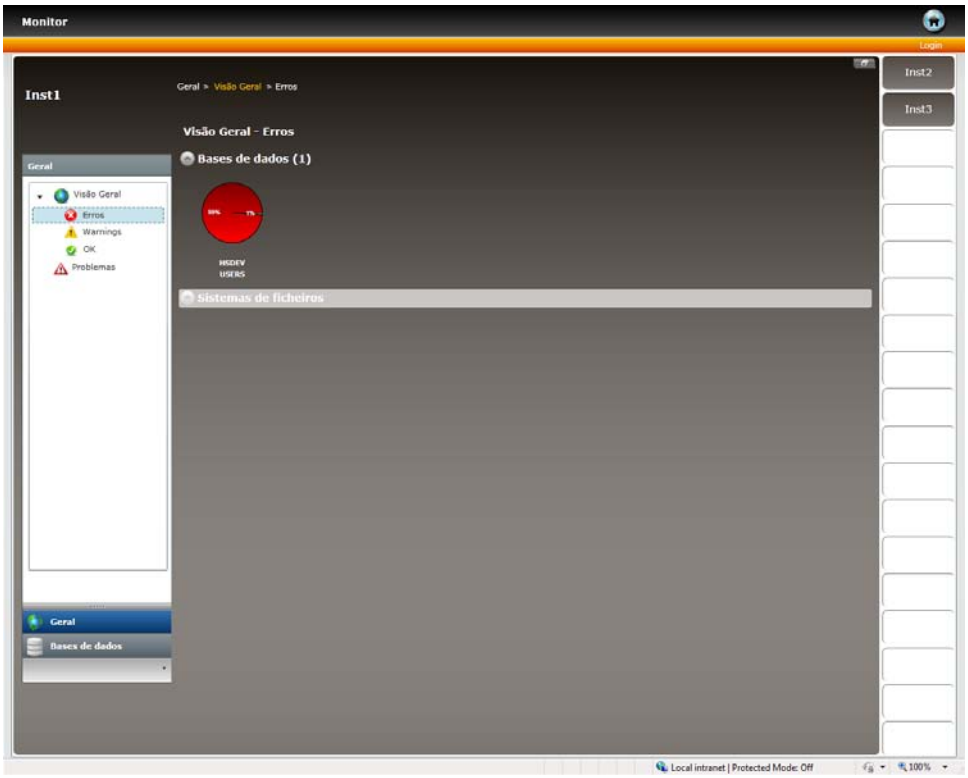

Figura B.4 – Página de estado geral com aplicação de filtro (*General*)

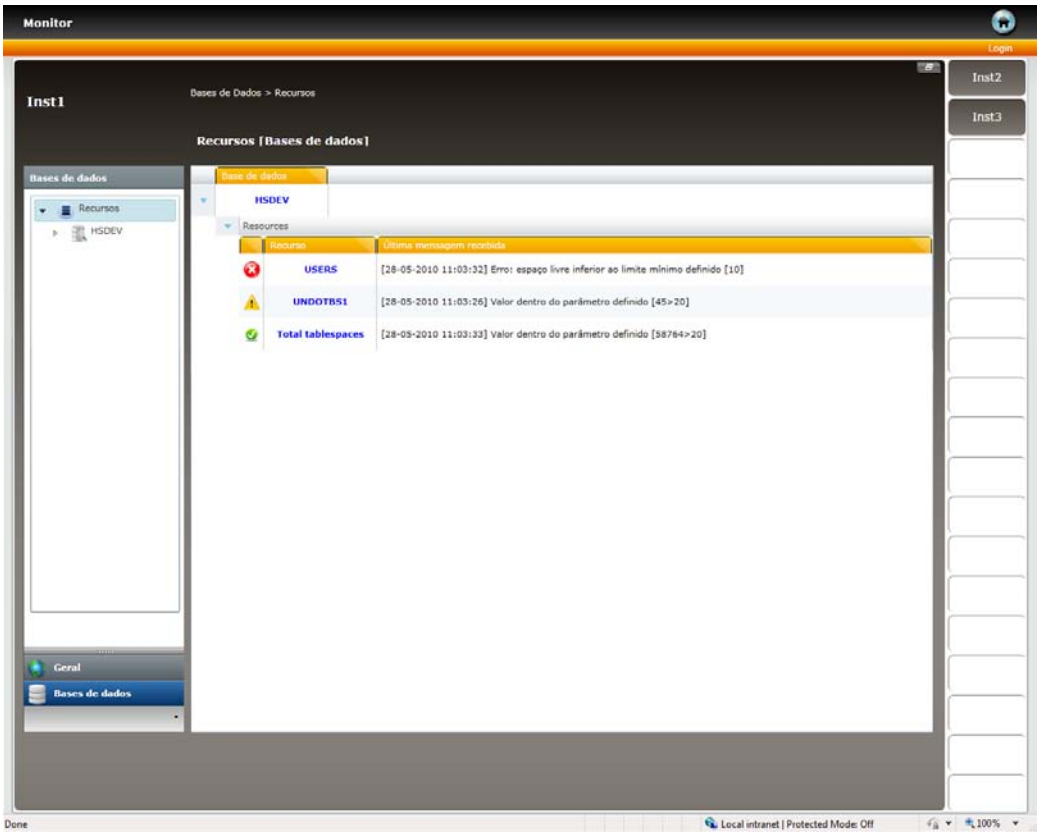

Figura B.5 – Página de listagem de todos os recursos de bases de dados sob monitorização (*ContextGeneral*)

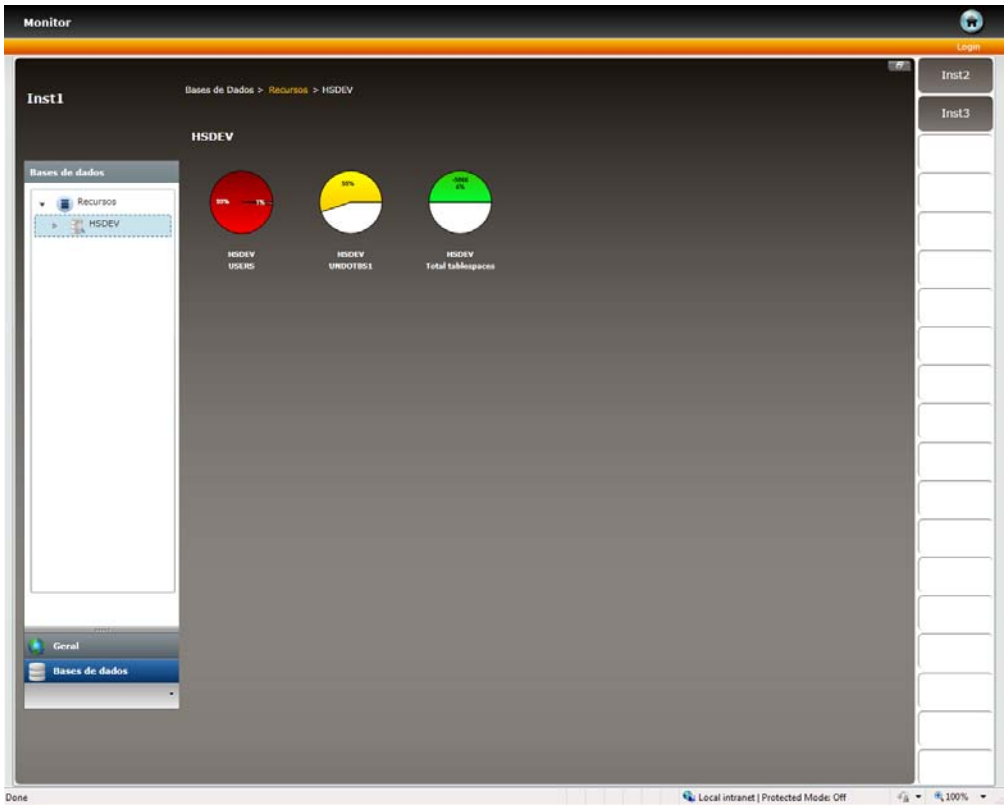

Figura B.6 – Página de detalhes de um *container* – base de dados (*Database_ContainerDetails*)

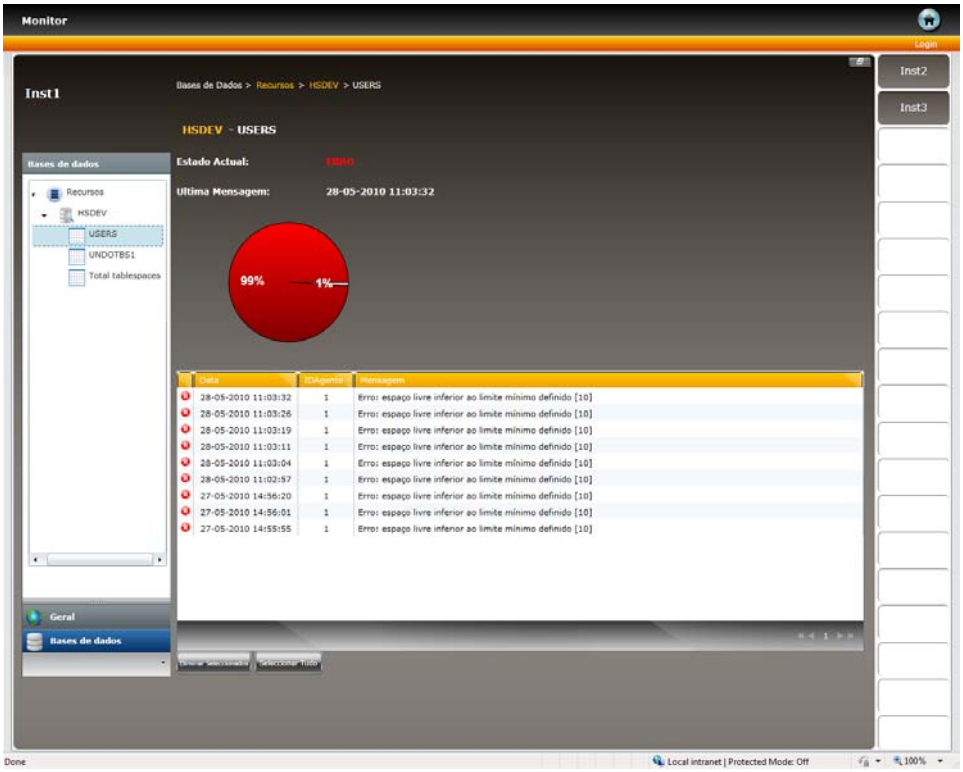

Figura B.7 – Página de detalhes de um recurso – base de dados (*Database_ResourceDetails*)

Anexo C Configuração de serviço de agentes

```
<?xml version="1.0"?>
<AgentsConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Agent>
    <Type>FilesystemAgent</Type>
    <ID>b5f404dd-f846-4279-84a8-12dfca0af14f</ID>
    <Units>
      <Unit>
        <ID>C:</ID>
        <Tasks>
          <Task>
            <TaskID>Numero de ficheiros em pasta</TaskID>
            <Type>MonitoringFilesystemTask</Type>
            <Interval>15</Interval>
            <Action>NumberOfFilesInFolder</Action>
            <OnError></OnError>
            <Properties>
              <TargetFolders>
                <Folder>
                  <FolderPath>Users\tramos\desktop</FolderPath>
                  <WarningThreshold>500 000 000</WarningThreshold>
                  <ErrorThreshold>1 000 000 000</ErrorThreshold>
                  <Time>45</Time>
                </Folder>
                <Folder>
                  <FolderPath>Users\tramos\tracing</FolderPath>
                  <WarningThreshold>500 000 000</WarningThreshold>
                  <ErrorThreshold>1 000 000 000</ErrorThreshold>
                  <Time>45</Time>
                </Folder>
              </TargetFolders>
            </Properties>
          </Task>
          <Task>
            <TaskID>Espaço total pasta</TaskID>
            <Type>MonitoringFilesystemTask</Type>
            <Interval>18</Interval>
            <Action>FolderUsedSpace</Action>
            <OnError>
            </OnError>
```

```
          <Properties>
            <TargetFolders>
              <Folder>
                <FolderPath>Users\tramos\desktop</FolderPath>
                <WarningThreshold>500 000 000</WarningThreshold>
                <ErrorThreshold>1 000 000 000</ErrorThreshold>
              </Folder>
              <Folder>
                <FolderPath>Users\tramos\tracing</FolderPath>
                <WarningThreshold>500 000 000</WarningThreshold>
                <ErrorThreshold>1 000 000 000</ErrorThreshold>
              </Folder>
            </TargetFolders>
          </Properties>
        </Task>
        <Task>
          <TaskID>Espaço livre unidade</TaskID>
          <Type>MonitoringFilesystemTask</Type>
          <Interval>10</Interval>
          <Action>UnitFreeSpace</Action>
          <OnError>
          </OnError>
          <Properties>
            <ActionType>ByPercentage</ActionType>
            <WarningThreshold>20</WarningThreshold>
            <ErrorThreshold>10</ErrorThreshold>
          </Properties>
        </Task>
      </Tasks>
    </Unit>
  </Units>
</Agent>
<Agent>
  <Type>DBAgent</Type>
  <ID>8c0e1b24-9a6b-4a08-acc5-752335dd236d</ID>
  <Databases>
    <Database>
      <ID> HSDEV </ID>
      <Module>Modulo1</Module>
      <Company>Company1</Company>
      <Tasks>
        <Task>
          <TaskID>Task1_Agent1</TaskID>
```

```
            <Type> MonitoringResourceTask </Type>
            <Interval> 30  </Interval>
            <Action> select round(100 * ( (df.totalspace - tu.totalusedspace)/
df.totalspace)) "% Free"
            from
            (select tablespace_name,
            round(sum(bytes) / 1048576) TotalSpace
            from dba_data_files
            group by tablespace_name) df,
            (select round(sum(bytes)/(1024*1024)) totalusedspace, tablespace_name
            from dba_segments
            group by tablespace_name) tu
            where df.tablespace_name = tu.tablespace_name and
df.tablespace_name='USERS'</Action>
            <OnError>
            </OnError>
            <Properties>
              <Target>USERS</Target>
              <ActionType>Query</ActionType>
              <Option>ByPercentage</Option>
              <FieldID>% Free</FieldID>
              <ErrorThreshold>10</ErrorThreshold>
              <WarningThreshold>20</WarningThreshold>
            </Properties>
          </Task>
          <Task>
            <TaskID>Task2_Agent2</TaskID>
            <Type> MonitoringResourceTask </Type>
            <Interval> 20  </Interval>
            <Action> select round(100 * ( (df.totalspace - tu.totalusedspace)/
df.totalspace)) "% Free"
            from
            (select tablespace_name,
            round(sum(bytes) / 1048576) TotalSpace
            from dba_data_files
            group by tablespace_name) df,
            (select round(sum(bytes)/(1024*1024)) totalusedspace, tablespace_name
            from dba_segments
            group by tablespace_name) tu
            where df.tablespace_name = tu.tablespace_name and
df.tablespace_name='UNDOTBS1'</Action>
            <OnError>
            </OnError>
```

```
            <Properties>
              <Target>UNDOTBS1</Target>
              <ActionType>Query</ActionType>
              <Option>ByPercentage</Option>
              <FieldID>% Free</FieldID>
              <ErrorThreshold>10</ErrorThreshold>
              <WarningThreshold>20</WarningThreshold>
            </Properties>
          </Task>
        </Tasks>
      </Database>
    </Databases>
  </Agent>
</AgentsConfig>
```

Código C.1 – Exemplo de uma configuração completa do serviço de agentes

Anexo D Mensagens

D.1 Registo

```
<AgentRegist>
     <InternalID>8c0e1b24-9a6b-4a08-acc5-752335dd236d </InternalID>
     <Type>DBAgent</Type>
     <Configuration>
       <Databases>
              <Database>
                     <ID> HSDEV </ID>
                     <Module>Modulo1</Module>
                     <Company>Company1</Company>
                     <Tasks>
                            <Task>
                                   <TaskID>Task1_Agent1</TaskID>
                                   <Type> MonitoringResourceTask </Type>
                                   <Interval> 7  </Interval>
                                   <Action> select round(100 * ( (df.totalspace -
tu.totalusedspace)/ df.totalspace)) &amp;quot;% Free&amp;quot;
            from
            (select tablespace_name,
            round(sum(bytes) / 1048576) TotalSpace
            from dba_data_files
```

```
            group by tablespace_name) df,
            (select round(sum(bytes)/(1024*1024)) totalusedspace, tablespace_name
            from dba_segments
            group by tablespace_name) tu
            where df.tablespace_name = tu.tablespace_name and
df.tablespace_name=&apos;USERS&apos;</Action>
                              <OnError></OnError>
                              <Properties>
                                     <Target>USERS</Target>
                                     <ActionType>Query</ActionType>
                                     <Option>ByPercentage</Option>
                                     <FieldID>% Free</FieldID>
                                     <ErrorThreshold>10</ErrorThreshold>
                                     <WarningThreshold>20</WarningThreshold>
                              </Properties>
                       </Task>
                <Tasks
        </Database>
    </Databases
</AgentRegist>
```

Código D.1 – Exemplo de uma mensagem completa para registo de um agente

D.2 Relatório

```
<AgentReport>
    <AgentID>8c0e1b24-9a6b-4a08-acc5-752335dd236d</AgentID>
    <Target>BD1|Tabela1</Target>
    <Report>80</Report>
    <Context>db<Context>
    <Message>
       <OK>Valor dentro dos parâmetros definidos</OK>
    </Message>
    <NextRun>10-06-2010 15:34:23</NextRun>
</AgentReport>
```

Código D.2 – Exemplo de uma mensagem completa de um relatório

Anexo E Estrutura geral de serviços WCF

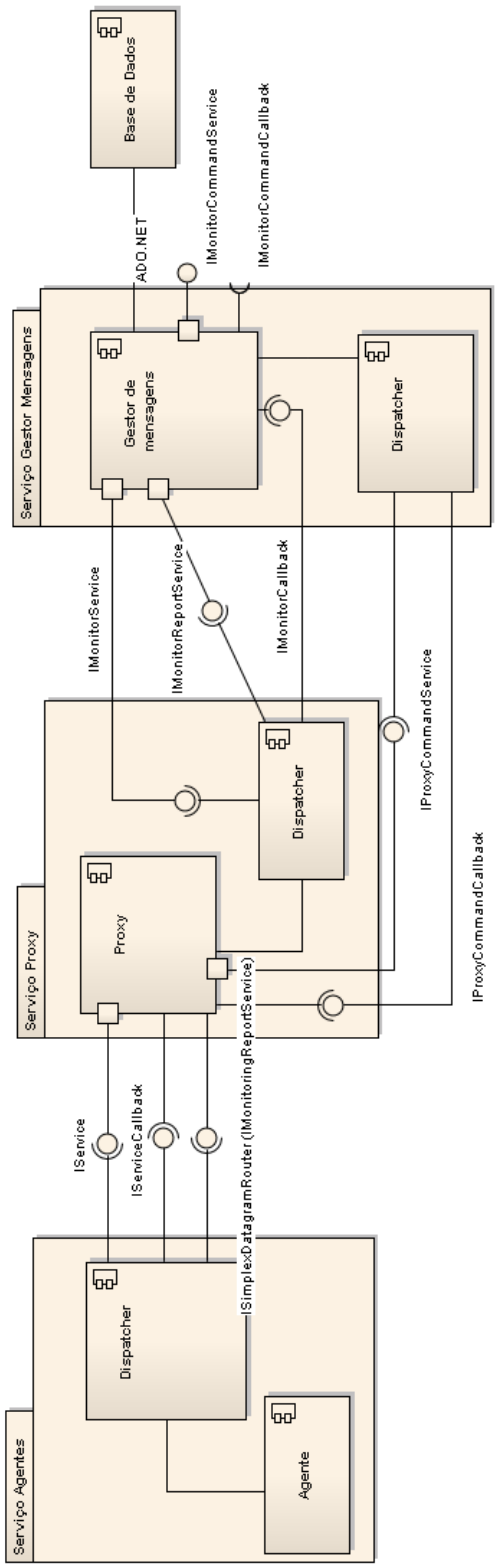

Figura E.1 – Mapa completo dos contratos de serviços WCF

Índice Remissivo