

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP

Análise de Capacidade e Performance de Servidores Aplicacionais

Pedro Tiago Cardoso Teixeira

VERSÃO DEFINITIVA

Relatório de Projecto
Mestrado Integrado em Engenharia Informática e Computação

Orientador: Prof. João José da Cunha e Silva Pinto Ferreira

Julho de 2009

Análise de Capacidade e Performance de Servidores Aplicacionais

Pedro Tiago Cardoso Teixeira

Relatório de Projecto
Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: José Manuel Magalhães Cruz (Professor Auxiliar)

Arguente: Feliz Alberto Ribeiro Gouveia (Professor Titular)

Vogal: João José da Cunha e Silva Pinto Ferreira (Professor Associado)

28 de Julho de 2009

Resumo

A área do retalho é actualmente um dos grandes impulsionadores dos sistemas de informação de apoio a grandes empresas e aos processos de negócio. Esta área está fortemente envolvida e dependente da tecnologia e as necessidades, cada vez maiores, de sistemas infalíveis e de alta performance impelem a contínua procura por soluções ideais. A resposta eficiente dos sistemas de informação utilizados pelos retalhistas é sem dúvida um dos principais requisitos solicitados. Para as empresas de consultoria e implementação de sistemas de informação esta necessidade de alta performance é, definitivamente, um dos focos colossais da actividade por elas praticada.

O projecto “Análise de Capacidade e Performance de Servidores Aplicacionais”, descrito neste documento, surge precisamente no seio de uma empresa de consultoria e implementação deste tipo de sistemas de informação. A Wipro Retail, divisão da Wipro Technologies para a área de retalho, é, actualmente, uma das maiores empresas de consultoria e implementação de soluções específicas para a área do retalho. A motivação para este projecto está relacionada com os problemas de performance que são muitas vezes detectados nos sistemas de informação implementados pela Wipro Retail. A procura de satisfação e fidelização dos seus clientes força a Wipro Retail a empregar variados recursos na tentativa de resolução destes problemas, que no total cria uma despesa elevada para a empresa. A ferramenta Wipro JDBC Spy, fruto deste projecto, pretende amenizar esses custos e possibilitar a implementação de melhores soluções. Para atingir esses objectivos esta ferramenta faz uma monitorização da aplicação em causa, recolhendo dados essenciais. Estes dados podem depois ser analisados pela ferramenta que, com o uso de regras desenvolvidas com recurso ao conhecimento de colaboradores experientes da Wipro Retail, gera relatórios avaliadores da performance apresentada. Estes relatórios irão providenciar às equipas da empresa o rumo a seguir na procura de melhoria de performance.

Este documento apresenta todo o estudo efectuado que foi necessário para a realização deste projecto. A análise de metodologias de monitorização, o estudo do estado da arte das ferramentas de monitorização e a análise tecnológica efectuada, que envolve o estudo de tecnologias como o *driver* JDBC, Java Reflection API ou metodologias de *parsing* de documentos XML, foram essenciais para a realização deste projecto. O cumprimento de todos os objectivos propostos leva a crer que o Wipro JDBC Spy será uma mais-valia para a empresa, na procura das fontes problemáticas de falhas de performance dos sistemas de informação implementados.

Abstract

The retail business is, nowadays, one of the major drivers to the evolution of the information systems that support big companies and their business processes. Retailing is heavily involved and dependent on technology and its demand for flawless and high performance systems push the continuous search for optimal solutions. The efficient response of the information systems used by retailers is undoubtedly one of the main requirements requested by them. For the information systems consulting and implementation firms, retailer's need for high performance is definitely a huge focus of their activity.

The project “Análise de Capacidade e Performance de Servidores Aplicacionais”, described herein arises precisely within a consulting and implementation firm. Wipro Retail, a division of Wipro Technologies for the retail business, is one of the largest consulting and implementation firms that provide specialized solutions for that business area. The motivation for this project is related to the performance problems that are often found in information systems implemented by Wipro Retail. The search for client satisfaction and loyalty drives Wipro Retail to spend a lot of resources on the attempt of solving this performance issues. The tool Wipro JDBC Spy, result of this project, aims to mitigate these costs and enable the development of better solutions. To achieve the objectives, this tool monitors a given application, collecting essential data. These data can then be analyzed by the tool, which generates reports about performance issues. This is done using a set of rules based on the knowledge of experienced employees of Wipro Retail. The reports will provide the company's teams the way forward in seeking to improve application's performance.

This document presents the whole study that was necessary to implement this project. The analysis of methodologies for monitoring, the study of the state of the art monitoring tools, and technological analysis conducted, which involves the study of technologies such as JDBC driver, Java Reflection API or methodologies for parsing of XML documents, were essential to this project. The accomplishment of all project objectives builds confidence that Wipro JDBC Spy will be an advantage for the company, on seeking the sources of performance issues on implemented information systems.

Agradecimentos

Os meus agradecimentos são destinados a todos aqueles sem os quais não teria sido possível realizar este projecto.

Em primeiro lugar a ambas as instituições, Faculdade de Engenharia da Universidade do Porto e Wipro Retail, por me proporcionarem esta oportunidade.

Um agradecimento especial para o Engenheiro Nuno Aguiar, responsável pelo meu acompanhamento na empresa. Obrigado por partilhares comigo, um pouco do imenso conhecimento que possuis e por me indicares sempre o caminho correcto. Toda a ajuda que me deste foi essencial e, mesmo quando o tempo era pouco, nunca te recusaste a dar-me toda a atenção. Obrigado.

Ao Professor João José da Cunha e Silva Pinto Ferreira, agradeço pela ajuda na elaboração deste documento e toda a orientação durante o projecto.

Aos *trainees* da Wipro Retail. Vocês são fantásticos! Obrigado por fazerem com que nenhum dia de trabalho tenha sido um esforço mas sempre um prazer.

Ao Ricardo Machado, pela ajuda preciosa que prestou na elaboração deste documento e pela força que sempre me transmitiu.

Gostaria também de agradecer a todos os colaboradores da Wipro Retail, sem excepção, por serem sempre prestáveis e pelo fantástico ambiente que criam naquela empresa.

Por fim, quero agradecer à minha família e aos meus amigos, que sempre me ajudaram e me apoiaram na longa caminhada até a esta fase final do meu curso.

Índice

1	Introdução	1
1.1	Contexto/Enquadramento	1
1.1.1	A Área do Retalho	1
1.1.2	Enabler – Wipro Retail	2
1.2	Motivação	4
1.3	Estrutura da Dissertação	5
2	Caracterização do Problema	7
2.1	Âmbito	7
2.2	Monitorização – Objectivo: Performance	8
2.3	Descrição e Objectivos	9
2.4	Fronteira	10
2.5	Conclusões	11
3	Estado da Arte	12
3.1	Soluções Proprietárias	12
3.1.1	PerformaSure® da Quest Software®	13
3.1.2	dynaTrace® da dynaTrace Software®	16
3.2	Soluções Open Source	19
3.2.1	P6Spy	19
3.2.2	Elvyx	20
3.3	Escolha Efectuada e Justificação	22
3.4	Conclusões	23
4	Proposta de Solução	24
4.1	Especificação da Ferramenta – Wipro JDBC Spy	24
4.2	Primeira Proposta de Solução	26
4.3	Segunda Proposta de Solução	27
4.4	Solução Adoptada: Segunda Proposta de Solução	28
4.5	Análise Tecnológica	28
4.5.1	O Driver JDBC	28
4.5.2	Apache log4j	30
4.5.3	Parser XML: SAX vs DOM	31
4.5.4	Base de Dados HSQLDB	33
4.5.5	Servidor Web: NanoHTTPD	33
4.5.6	Java Reflection API	34
4.6	Conclusões	34
5	Implementação da Solução	35
5.1	Melhoramento da Performance do Elvyx	35
5.2	Fase 1: Registo dos Dados em Ficheiro XML	36
5.2.1	Registo do Caminho do Pedido do Cliente	36
5.2.2	Dados a Registar	36
5.2.3	Definição da Estrutura do Ficheiro XML	37

5.2.4	Utilização do log4j Para Fazer o Registo dos Dados.....	38
5.2.5	Transacções XA	39
5.3	Fase 2: Do Ficheiro XML para a Base de Dados.....	39
5.3.1	Alterações à Base de Dados	39
5.3.2	Commons Digester	41
5.3.3	Particularidades da Funcionalidade	41
5.4	Fase 3: Ferramentas Extensíveis de Análise de Dados.....	42
5.4.1	Ficheiro de Propriedades	42
5.4.2	Classe Executável – Analyze	43
5.4.3	Classe Abstracta – Analyzer	43
5.4.4	Classes Efectivas de Análise de Dados – Analyzers	43
5.4.5	Geração do Relatório.....	45
5.5	Melhoria da Fase 1: Cliente Web.....	47
5.6	Conclusões.....	49
6	Resultados Experimentais.....	51
6.1	Teste Demonstrativo de Execução	51
6.1.1	Teste Realizado	51
6.1.2	Resultados Obtidos.....	52
6.2	Testes – Fase 1.....	52
6.2.1	Testes Realizados	52
6.2.2	Resultados Obtidos.....	53
6.3	Testes – Fase 2.....	53
6.3.1	Testes Realizados	53
6.3.2	Resultados Obtidos.....	53
6.4	Testes – Fase 3.....	56
6.4.1	Testes Realizados	56
6.4.2	Resultados Obtidos.....	56
6.5	Comparação de Performance Utilizando Drivers Diferentes.....	56
6.5.1	Testes Realizados	56
6.5.2	Resultados Obtidos.....	56
6.6	Conclusões.....	58
7	Conclusões e Trabalho Futuro.....	59
7.1	Retrospectiva	59
7.2	Satisfação dos Objectivos	60
7.3	Trabalho Futuro	60
	Referências	61
	Anexo A – Modelo das Classes “Analyzers”	64
	Anexo B – Classe de Teste: Test4Analyzers	66
	Anexo C – Exemplo de um Ficheiro de Propriedades.....	69
	Anexo D – Exemplo de um Ficheiro XML de Registo de Pedidos	71
	Anexo E – Exemplo de um Relatório Final	80

Lista de Figuras

Figura 1-1 – Carteira de clientes da Wipro Retail.....	2
Figura 1-2 –Arquitectura de sistemas de um retalhista moderno.....	3
Figura 1-3 – Diagrama demonstrativo das áreas abrangidas pelas soluções Oracle Retail. ..	4
Figura 1-4 – Mapa de Gantt do planeamento definido.....	5
Figura 3-1 – Área de actuação do PerformaSure	13
Figura 3-2 – Exemplo de um caminho transaccional de um pedido no PerformaSure.	14
Figura 3-3 – Exemplo de árvore de chamadas de métodos no PerformaSure.	14
Figura 3-4 – Vista de SQL do PerformaSure.....	15
Figura 3-5 – Rastreio de uma transacção num ambiente heterogéneo com o dynaTrace....	16
Figura 3-6 – Reconstrução do problema até à linha de código com o dynaTrace	17
Figura 3-7 – Vistas da versão de produção e de testes do dynaTrace	18
Figura 3-8 – Padrão de desenho de software: Proxy	19
Figura 3-9 – Excerto de um ficheiro de <i>log</i> criado pelo P6Spy.....	20
Figura 3-10 – Componentes principais do Elvyx.....	21
Figura 3-11 – Cliente gráfico do Elvyx.....	21
Figura 4-1 – Esquema da ferramenta esperada e divisão em 3 fases.	26
Figura 4-2 – Arquitectura JDBC. Posição do Driver JDBC	28
Figura 4-3 – Relação entre os principais componentes do Driver JDBC	29
Figura 4-4 – Hierarquia dos <i>Loggers</i> no log4j.....	30
Figura 4-5 – Modo de funcionamento do log4j.	31
Figura 4-6 – Número de <i>tags</i> XML processadas por segundo utilizando SAX e DOM	32
Figura 4-7 – Megabytes de memória consumidos por SAX e DOM	32
Figura 4-8 – Exemplo de uma sequência pedido/resposta a um servidor Web.....	33
Figura 4-9 – Classes principais de Java Reflection.	34
Figura 5-1 – Esquema das funcionalidades a implementar na fase 1.....	36
Figura 5-2 – Esquema das funcionalidades a implementar na fase 2.....	39
Figura 5-3 – Modelo de dados da base de dados HSQLDB original do Elvyx.....	40
Figura 5-4 – Modelo de dados da base de dados HSQLDB da nova ferramenta.....	40
Figura 5-5 – Esquema das funcionalidades a implementar na fase 3.....	42
Figura 5-6 – Esquema de todas as funcionalidades da fase 1.	47
Figura 5-7 – Exemplo de utilização do cliente <i>web</i>	48
Figura 5-8 – Menu do cliente <i>web</i> do Wipro JDBC Spy.....	49
Figura 5-9 – Dados apresentados no cliente relativos aos pedidos interceptados.....	49
Figura 5-10 – Esquema da ferramenta esperada e divisão em 3 fases	50
Figura 6-1 – Testes – Fase 2 – Base de dados em modo embebido.	54
Figura 6-2 – Testes – Fase 2 – Base de dados em modo servidor.....	55
Figura 6-3 – Análise de performance do <i>driver</i> JDBC – Sobrecarga.....	57
Figura 6-4 – Análise de performance do <i>driver</i> JDBC – Funcionamento Normal.	58

Lista de Tabelas

Tabela 1-1 – Nomes das tarefas do planeamento.....	5
Tabela 2-1 – Dados de alguns clientes da Wipro Retail.....	10
Tabela 3-1 – Avaliação das ferramentas estudadas.....	22
Tabela 4-1 – Tabela comparativa entre o P6Spy (versão Wipro Retail) e o Elvyx.....	25
Tabela 5-1 – Dados registados no ficheiro XML.....	37
Tabela 6-1 – Testes Fase 1.	53
Tabela 6-2 – Testes – Fase 2 – Base de dados em modo embebido.....	54
Tabela 6-3 – Testes – Fase 2 – Base de dados em modo servidor.	55
Tabela 6-4 – Análise de performance do <i>driver</i> JDBC – Sobrecarga.	57
Tabela 6-5 – Análise de performance do <i>driver</i> JDBC – Funcionamento Normal.	57

Abreviaturas e Símbolos

3-tier	<i>Three-tier</i> – três camadas
API	Application Programming Interface
CSV	Comma Separated Values
DOM	Document Object Model
GB	Gigabyte
HSQLDB	Hyper Structured Query Language Database
IDE	Integrated Development Environment
J2EE	Java 2 Enterprise Edition
Java	Linguagem de programação da Sun Microsystems ¹
JavaScript	Linguagem de <i>Scripting</i> frequentemente utilizada em páginas <i>web</i>
JDBC	Java Database Connectivity
JNDI	Java Naming and Directory Interface
JRE	Java Runtime Environment
JVM	Java Virtual Machine
ODBC	Open Database Connectivity
PDF	Portable Document Format
ROI	Return on Investment
SAX	Simple API for XML
SQL	Structured Query Language
XML	Extensible Markup Language

¹ Em 20 de Abril de 2009 a Oracle anunciou a aquisição da Sun Microsystems.

1 Introdução

Este capítulo pretende situar o leitor no contexto do problema em questão e apresentar o porquê da necessidade deste projecto. A compreensão da área envolvente do projecto ajuda a interpretar os problemas que pretendem ser resolvidos com a realização deste projecto.

1.1 Contexto/Enquadramento

1.1.1 A Área do Retalho

A área de negócio de retalho pode ser definida como sendo a “venda de produtos ou a comercialização de serviços directamente ao consumidor final” [BDS91]. No entanto, hoje é mais do que a simples venda de uma pequena quantidade de produtos a um cliente. É toda a gestão envolvida na aquisição e prestação dos recursos necessários para colmatar todas as necessidades dos clientes [GDAVI93].

O retalho está em constante evolução, desde sempre houve mudanças na forma como o negócio é realizado e continuarão a existir novas mudanças. A forma como cada retalhista aborda a evolução do negócio e as mudanças que ele impõe a si mesmo são determinantes, não só para o contínuo crescimento do negócio, mas muitas vezes para a sua própria sobrevivência. As evoluções no negócio de retalho são, na sua maioria, um reflexo da competitividade entre os retalhistas, na procura de satisfação do cliente que, por sua vez, está também cada vez mais atento e exigente. Nos últimos anos, a procura para satisfazer os clientes, levou a que os retalhistas apostem na oferta de uma maior variedade de produtos, em locais convenientes ao cliente, nas mais diversificadas horas. Também a “experiência de compra” se torna um factor cada vez mais importante, assim como a isenção de falhas por parte do retalhista em todo o processo de compra [PHM08].

Com a evolução do negócio aumentou também a sua complexidade, de tal forma que, nos dias de hoje, o volume de informação gerado e utilizado por um médio/grande retalhista assume proporções de tal forma elevadas que seria impossível essa informação ser tratada por humanos. As tecnologias de informação surgem então como meio de tratamento desses dados em tempo útil.

A utilização de sistemas de informação orientados para este negócio possibilita aos retalhistas estarem na linha da frente, conseguindo acompanhar a evolução do negócio e satisfazer os seus clientes, aumentando a sustentabilidade das empresas e gerando valor para as mesmas.

1.1.2 Enabler – Wipro Retail

História

A Enabler foi constituída no ano de 1997 a partir da autonomização da Direcção de Sistemas de Informação da Sonae Distribuição. A actividade e experiência na concepção e desenvolvimento de sistemas de informação para a Modelo Continente proporcionaram um conhecimento elevado dos processos e sistemas de retalho. Desde então, a Enabler teve um crescimento considerável e tornou-se numa empresa de referência na integração de sistemas de informação para retalho, em toda a Europa. Em poucos anos a Enabler passou a ser uma empresa multinacional, contando na sua lista de clientes com grandes retalhistas do Reino Unido, Itália, Espanha, Alemanha e Brasil.

Em 2006, a Enabler foi adquirida pela Wipro, uma multinacional sediada em Bangalore, Índia. A aquisição da Enabler por parte da Wipro surgiu como forma de dar resposta às necessidades dos seus clientes da área de retalho. A Enabler é agora a Wipro Retail, divisão da Wipro Technologies para a área de retalho.

O crescimento da empresa passou a ser ainda mais elevado nos últimos anos e esta tem agora clientes em todo o mundo, sendo alguns deles dos mais importantes retalhistas mundiais, Alguns nomes da carteira de clientes da Wipro Retail são ilustrados na figura 1-1.



Figura 1-1 – Carteira de clientes da Wipro Retail.

Estratégia

Através da experiência adquirida ao longo dos anos em retalho e gestão de preços, *customer analytics*², *global data synchronization*³, gestão de loja e gestão da cadeia logística, a Wipro Retail ajuda os seus clientes a obterem uma ligação eficiente entre os seus processos de negócio e os sistemas de informação, apresentando soluções focadas em diversas áreas do retalho como alimentação ou moda.

Aplicando o foco do problema ao negócio e não aos sistemas de informação, a Wipro Retail opera no cliente, oferecendo soluções *end-to-end*⁴, suportando os processos de negócio vitais da empresa e integrando os sistemas legados⁵ e aplicações externas com o seu próprio *software*. Suportada por uma engenharia pragmática, a Wipro Retail foca-se em apresentar resultados e coloca os clientes ao seu lado, tornando-os parceiros, e dispondo-se a partilhar o risco com eles. A Wipro Retail apresenta uma *framework* de integração de sistemas de informação que pretende aproximar as várias áreas de acção dos retalhistas, colmatando assim todas as suas necessidades. Um retalhista apresenta, normalmente, sistemas de informação nas áreas de operações, optimização de negócio, cadeia logística, gestão de informação e ainda na integração de todas estas áreas, figura 1-2 [WIPR09].

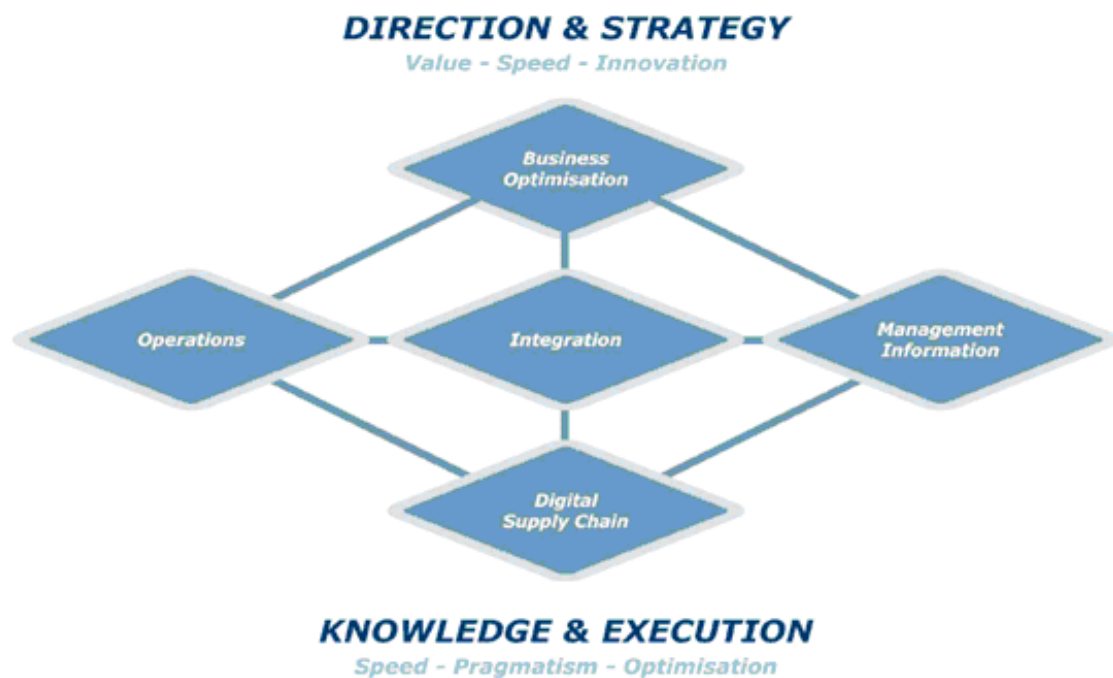


Figura 1-2 –Arquitectura de sistemas de um retalhista moderno [WIPR09].

A Wipro Retail fornece soluções para todas estas áreas, tendo como base uma *suite* (coleção de aplicações de software) da Oracle, Oracle Retail Merchandising System, que é a base de todo o sistema. Outros módulos podem ser acrescentados, como por exemplo, Oracle Active Retail Intelligence, Oracle Retail Warehouse Management System ou Oracle Retail

² Processo de análise de dados sobre o comportamento dos clientes.

³ Sincronização da informação que descreve univocamente um produto ou serviço entre parceiros transaccionais.

⁴ O fornecedor da aplicação providenciará todo o *hardware*, *software* e recursos necessários para atingir os requisitos do cliente sem que nenhum outro fornecedor esteja envolvido.

⁵ Sistemas que, apesar de serem bastante antigos, fornecem serviços essenciais e deverão ser integrados na nova solução.

Introdução

Point-of-Service, oferecendo assim soluções para as mais diversas áreas, desde a gestão de armazéns às operações, e da gestão da cadeia logística ao *business intelligence*⁶, figura 1-3. Estas ferramentas, associadas ao *know-how* existente na Wipro Retail, permitem à empresa criar soluções à medida das necessidades específicas de cada cliente, construindo, desta forma, produtos únicos, dificilmente imitáveis e com um valor enorme para os clientes.

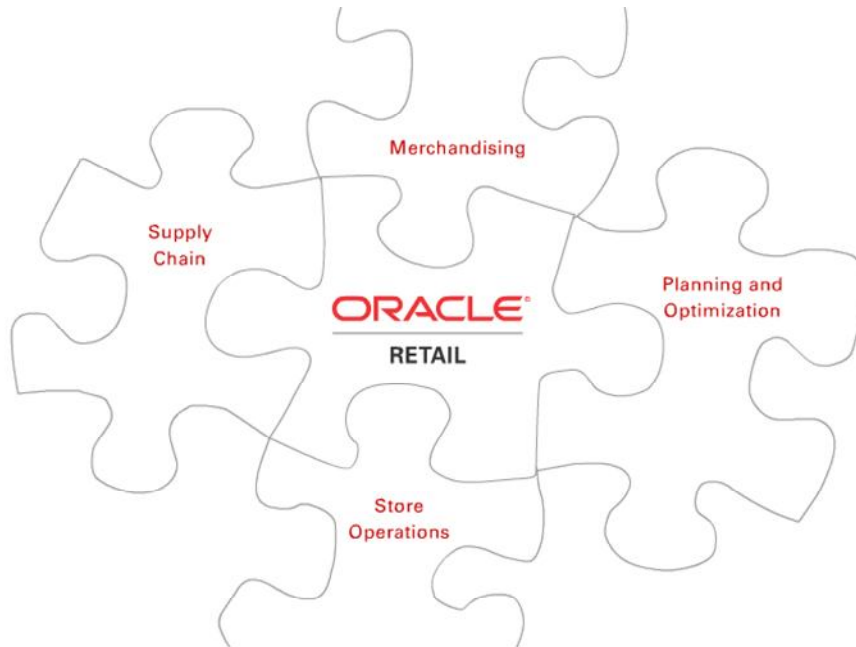


Figura 1-3 – Diagrama demonstrativo das áreas abrangidas pelas soluções Oracle Retail.

1.2 Motivação

O projecto “Análise de Capacidade e Performance de Servidores Aplicacionais”, descrito neste documento, surge como consequência das dificuldades de optimização das aplicações e das bases de dados, quer em produção, quer em implementação por parte das equipas da Wipro Retail. Estas dificuldades surgem, principalmente, devido ao volume de dados tratado e existente nas bases de dados e à complexidade das aplicações. Na sua maioria, os retalhistas têm quantidades enormes de dados a ser guardados e tratados por cada aplicação. Também, em muitos casos, são necessários tempos de resposta curtos para conseguir ter eficiência nos processos de negócio. Estes factores forçam as equipas de implementação e manutenção da Wipro Retail a dar uma enorme ênfase à performance das bases de dados e das aplicações. Esta actividade é muito penosa e muitas vezes obriga a que sejam despendidos dias de trabalho para encontrar as fontes do problema. No seguimento deste problema surge a necessidade de ter uma ferramenta que auxilie nesta actividade.

O “Wipro JDBC Spy” é a aplicação que surge como objectivo deste projecto. Esta será, para já, uma ferramenta apenas para uso interno, e tentará ser um auxílio no trabalho das equipas da Wipro Retail descrito em cima. O objectivo principal desta ferramenta passa por registar todos os pedidos que são efectuados às bases de dados, criar estatísticas sobre os mesmos e fazer uma análise desses dados para detectar problemas ou padrões de uso. De certa forma, esta ferramenta tentará sistematizar acções que são tomadas quando é encontrado um problema de performance, e tentará aplicar e agregar o *know-how* de vários elementos da Wipro Retail, de uma forma automática e bastante mais eficiente.

⁶ Conjunto de habilidades, tecnologias, aplicações e práticas que auxiliam um negócio a adquirir melhor compreensão sobre o seu contexto comercial.

Introdução

Espera-se, com o uso desta ferramenta, que seja diminuído drasticamente o tempo dispendido na procura da fonte dos problemas e que esta se torne num apoio essencial nos projectos da Wipro Retail.

Para atingir estes objectivos definiu-se o planeamento representado no mapa de Gantt, ilustrado na figura 1-4. As tarefas estão descritas na tabela 1-1.

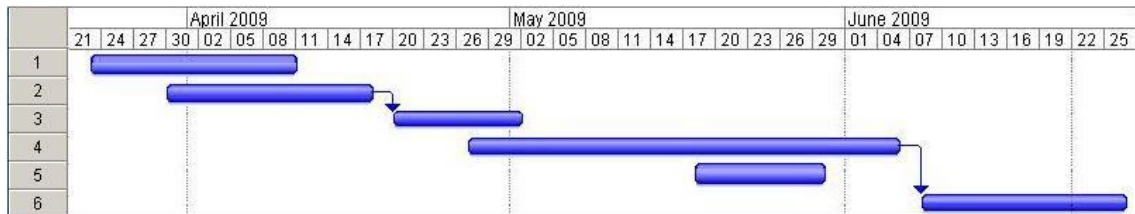


Figura 1-4 – Mapa de Gantt do planeamento definido.

Tabela 1-1 – Nomes das tarefas do planeamento.

Tarefa	Nome
1	Análise dos problemas relacionados com performance nos servidores aplicacionais
2	Estudo de ferramentas de monitorização de performance
3	Análise da ferramenta interna de recolha de dados
4	Implementação de melhorias na ferramenta interna ou noutra similar
5	Elaboração de metodologias de teste de performance
6	Escrita do relatório de projecto

1.3 Estrutura da Dissertação

Esta dissertação foi estruturada de forma a inteirar o leitor mais profundamente no projecto a cada secção que vai lendo. A informação contida numa secção será essencial para as secções seguintes. Assim, recomenda-se, numa primeira abordagem, uma leitura contínua do documento, do início ao fim.

No capítulo que aqui termina foi introduzido o tema desta dissertação e fornecida uma descrição do contexto em que este projecto se insere.

No capítulo seguinte é caracterizado e apresentada uma descrição do problema, dando a conhecer o âmbito, os objectivos gerais e a fronteira do mesmo. A leitura desse capítulo é essencial para a compreensão do problema em mãos. Aí é apresentada uma visão de alto nível da problemática envolvente.

No terceiro capítulo, estado da arte, é apresentado o estudo efectuado às melhores ferramentas existentes no mercado com objectivo de resolver o problema caracterizado no capítulo dois. É feita uma análise profunda às mais avançadas ferramentas de monitorização disponíveis no mercado, quer proprietárias, quer *open source*. No final são ainda apresentadas uma comparação entre as ferramentas e a escolha efectuada para o restante deste projecto.

Na proposta de solução, desenvolvida no quarto capítulo, são apresentadas as propostas de solução possíveis e o caminho que deve ser seguido na implementação da ferramenta pretendida assim como os requisitos principais da mesma. É ainda apresentado o estudo realizado sobre todas as ferramentas e tecnologias necessárias para a implementação da solução. Este capítulo é fundamental para a compreensão do funcionamento da ferramenta posteriormente implementada e para a compreensão de todos os métodos e todas escolhas efectuadas na fase de implementação.

No capítulo cinco é descrita a implementação efectuada, seguindo o caminho que foi realmente utilizado para essa mesma implementação. Aí são apresentadas, em detalhe, todas as

Introdução

particularidades da ferramenta e as suas funcionalidades. A leitura deste capítulo providencia um conhecimento profundo da ferramenta e do trabalho efectuado na sua implementação.

No capítulo seguinte, resultados experimentais, são apresentados os testes efectuados à ferramenta implementada. Estes testes pretendem não só avaliar o estado e a funcionalidade da ferramenta em relação aos objectivos propostos mas também demonstrar casos de uso da ferramenta.

Por último, no capítulo sete são expostas as conclusões retiradas deste projecto. Estas conclusões são apresentadas de uma forma subjectiva e crítica sobre o projecto, existindo também algumas considerações pessoais.

2 Caracterização do Problema

Este capítulo caracteriza o problema em questão. Aqui é definido o âmbito em que este se insere, é feita uma descrição global do mesmo, que inclui os objectivos gerais esperados e é definida a fronteira do projecto.

2.1 Âmbito

Uma das primeiras acções que se toma quando surgem os primeiros sinais de falha de performance nas aplicações é alterar o *hardware* existente ou adicionar maior capacidade às máquinas onde correm as aplicações. No entanto, qualquer computador, seja ele grande ou pequeno, novo ou velho, apresenta limites em todas as suas componentes. As alterações ao nível de *hardware* servirão apenas como um remedeio de um problema que mais tarde ou mais cedo se vai fazer notar. Os constantes investimentos em *hardware* dão ainda asas a que sejam descuradas as verdadeiras razões do problema: má configuração das máquinas, aplicações mal desenhadas/programadas. A primeira hipótese raramente ocorre, já que nas grandes implementações, como são o caso das implementações da Wipro Retail, existe pessoal dedicado a monitorizar e a configurar as máquinas, e mesmo que tal erro ocorra, facilmente este é detectado e corrigido. O grande problema está relacionado com *software*. Quando as aplicações não estão bem desenhadas ou programadas, facilmente são cometidos erros que provocam atrasos na execução.

A falta de optimização de *software* pode fazer com que processos consumam uma quantidade enorme de recursos, relativamente ao que deveriam consumir. Estes processos são denominados *resource hogs*, e implicam grandes consumos de processamento e má performance. Outro tipo de problemas que ocorrem frequentemente devido à má programação são os bloqueios. Estes causam má performance e um uso reduzido das capacidades de processamento. Quando um *software* está na sua fase de desenvolvimento, onde, por norma, há uma grande quantidade de recursos disponíveis e poucos dados a tratar, não são notórias as falhas de performance do sistema. No entanto, quando este entra em produção num ambiente sobrecarregado, a gestão de recursos é muito mais importante e a falta de optimização pode levar a falhas de performance de todo o sistema. Estas situações, raramente conseguem ser resolvidas com melhorias de *hardware*. Deverão ser encontradas as falhas de *software* e resolvidas estas situações, para se atingir melhorias globais no desempenho do sistema [TIMG06].

2.2 Monitorização – Objectivo: Performance

A optimização de performance deve ser tida em conta em várias etapas do ciclo de vida de uma aplicação. Erros ou falhas de performance, quando detectados e resolvidos tardiamente, implicam custos elevados e muitas vezes os resultados obtidos não são completamente satisfatórios [JACKS03]. A performance deve ser parte integrante do planeamento e desenvolvimento de uma aplicação. Devem ser antecipados requisitos de performance durante a análise e desenho das aplicações e analisado o custo/benefício do nível de performance óptimo [ORA05b]. No entanto, nas fases de planeamento, desenho e desenvolvimento, a performance não deve ser o foco principal, e muitas vezes é mais simples e eficaz ter uma fase de análise de performance após o desenvolvimento [JACKS03]. A fase de análise de performance consiste essencialmente em monitorizar a aplicação e analisar dados obtidos. Esta fase, muitas vezes, estende-se para lá da implementação e é mantida quando uma aplicação está em produção.

“Monitorização consiste em observar atentamente uma situação ou um caso individual, com objectivo de determinar quais as acções necessárias a tomar” [GUVE03].

Os seguintes elementos constituem a monitorização [GUVE03]:

- É feita durante um longo período de tempo;
- Envolve coleccionar ou receber grandes quantidades de dados;
- É feita uma observação cuidada da situação através de examinações constantes ou periódicas;
- Normas são utilizadas como referência para uma avaliação da situação ou do caso em questão;
- O resultado da monitorização é normalmente um relatório sobre a situação;
- O relatório fornece uma avaliação da situação que serve como base para as decisões necessárias a tomar.

De facto, o essencial a retirar da monitorização é o relatório final, pois é com a ajuda deste que todas as decisões serão tomadas. No entanto, os passos intermédios até à obtenção desse relatório são bastante morosos e difíceis de atingir quando feitos manualmente. As ferramentas de monitorização são cada vez mais populares e mais importantes nas áreas de informação. É usual, nos dias de hoje, ter acesso fácil e barato a boas ferramentas de *profiling*⁷ e *debugging*⁸, estas já vêm normalmente incluídas em IDE *open source* e são suficientes para a maioria das aplicações locais. As aplicações distribuídas, por outro lado, apresentam um maior número de condicionantes o que as tornam mais difíceis de monitorizar. Estas aplicações podem apresentar todos os problemas das aplicações locais e ainda problemas que fogem do contexto da aplicação em si, estando sujeitas a factores externos, como comunicações com base de dados, latências de rede ou número de utilizadores em simultâneo. Por estes motivos, as ferramentas de monitorização de aplicações distribuídas são mais escassas e, na sua maioria, proprietárias [JACKS03].

Um artigo sobre um estudo realizado pela Mercury Interactive Corporation [DREW02] aos seus *web sites*, mostra que os principais problemas das aplicações distribuídas residem em quatro grandes áreas: base de dados, servidores *web*, servidores aplicativos e rede. A nível da base de dados os principais problemas residem na insuficiência de indexação, em bases de dados fragmentadas, estatísticas desactualizadas e no mau desenho da base de dados. Os principais problemas nos servidores *web* residem em maus algoritmos, configurações incorrectas, problemas de memória e escassez de recursos ao nível de *hardware*. Nos servidores aplicativos, os principais problemas residem na má gestão da *cache*, deficiente optimização dos pedidos à base de dados, configurações incorrectas e na deficiente gestão de pedidos concorrentes. Por fim, a nível da rede, os principais problemas incluem largura de banda

⁷ Análise de performance. Investigação do comportamento de um programa através da análise de dados obtidos. O objectivo é determinar que secções do programa devem ser optimizadas.

⁸ Processo de análise e correcção de falhas existentes nos programas de software ou em hardware.

inadequada e incompatibilidades, más configurações e sub-dimensionamento de *routers*, *switches*, *firewalls* e *load balancers* [DREW02]. Este estudo ajuda a compreender as condicionantes e as variáveis implícitas à monitorização deste tipo de aplicações. Para cada um dos problemas mencionados existem várias abordagens que tentam resolver ou pelo menos amenizar os problemas. Pode-se referir, por exemplo, a boa configuração dos tamanhos de *cache*, a medição de utilização de recursos, a medição dos tempos de resposta ou os pedidos às bases de dados como abordagens a alguns dos problemas mencionados. Estas abordagens são conhecidas e, na sua maioria, o *software* existente nos servidores fornece dados suficientes para que possam ser feitas análises e tomadas decisões. A grande dificuldade e o segredo de uma boa monitorização reside na análise que é feita aos dados recolhidos [JACKS03].

Segundo Jack Shirazi, a escolha de uma ferramenta de monitorização deverá ter em atenção [JACKS03]:

- **Componentes de monitorização e logging:** Estas ferramentas devem monitorizar todos os aspectos importantes do sistema. Todas as acções importantes devem ser registadas;
- **Sobrecarga:** A monitorização do sistema não deve impor uma grande sobrecarga no desempenho do mesmo. Ainda que essa monitorização não seja sempre efectuada mas apenas regularmente, a sobrecarga deve ser diminuta, de modo a não ser percebida pelo utilizador final;
- **Mapeamento pedido-método:** A ferramenta de monitorização deve relacionar os pedidos à base de dados com os métodos associados. Isto permite dar um maior foco a partes mais problemáticas do sistema;
- **Granularidade e persistência dos dados registados:** Os dados devem ser guardados, para que seja possível dissociar a análise, do acto de monitorização em tempo real. Devem ser registados todos os dados necessários, tendo sempre em atenção que quanto maior for a granularidade maior será a sobrecarga no sistema;
- **Escalabilidade:** A ferramenta de monitorização deve ser facilmente adaptada à aplicação a monitorizar e fácil de implantar num ambiente de produção.
- **Análise de dados:** Não sendo uma funcionalidade obrigatória, a ajuda na análise dos dados será uma mais-valia para qualquer ferramenta de monitorização, já que esta é uma parte crítica do processo, e terá um peso enorme na escolha da ferramenta ideal.

2.3 Descrição e Objectivos

As aplicações implementadas pela Wipro Retail lidam com uma quantidade enorme de dados e a sua performance é fulcral para os seus clientes. Cada módulo que faz parte dos sistemas implementados tem um número de tabelas, registos, e tamanho total bastante elevados. Na sua maioria, nas implementações da Wipro Retail, são implementados no mínimo 3 a 5 módulos do Oracle Retail. Estes módulos variam em número de tabelas, registos e tamanho. No entanto, apenas a título informativo, pode-se ver na tabela 2-1 o número de tabelas e registos existentes nas bases de dados e espaço físico ocupado pelos dados existentes apenas do módulo “Oracle Retail Merchandising System” de 3 clientes da Wipro Retail.

Caracterização do Problema

Tabela 2-1 – Dados de alguns clientes da Wipro Retail⁹.

Cliente 1	Número de Tabelas	> 2200
	Número de Linhas (Milhões)	> 500
	Tamanho Total (GB)	> 41
Cliente 2	Número de Tabelas	> 3500
	Número de Linhas (Milhões)	> 4.700
	Tamanho Total (GB)	> 289
Cliente 3	Número de Tabelas	> 1700
	Número de Linhas (Milhões)	> 900
	Tamanho Total (GB)	> 66

Torna-se então necessário criar metodologias de análise de performance das aplicações. Neste caso, a envolvente do problema centra-se nos servidores aplicativos. Pretende-se desenvolver e aperfeiçoar um conjunto de metodologias que, criteriosamente, permitam efectuar testes de carga/volume e recolha de dados estatísticos sobre a componente lógica dos sistemas implementados. Este tipo de análise deve permitir a optimização dos sistemas e processos implementados e também determinar a capacidade e escalabilidade dos mesmos. Sendo assim, tem-se como objectivos:

- Utilizar um conjunto de metodologias de simulação de carga e variantes adaptadas às condicionantes existentes em cada caso prático.
- Utilizar ferramentas de recolha de dados estatísticos de desempenho e interpretação dos mesmos, adequadas aos sistemas e processos em análise.
- Melhorar uma ferramenta interna ou utilizar uma ferramenta similar de recolha de dados estatísticos de desempenho específico em arquitecturas *3-tier*¹⁰ e interpretação dos mesmos.
- Participação na elaboração de documentos de análise sobre critérios utilizados na avaliação de capacidade e performance sobre servidores aplicativos.

Para atingir estes objectivos deverá ser utilizada uma ferramenta que permita interceptar pedidos a bases de dados por parte das aplicações.

2.4 Fronteira

A fronteira de um projecto delimita até onde este poderá ir, que acções se poderão tomar e onde se espera chegar. É necessário definir algumas directrizes para conseguir levar este projecto a bom porto. Assim, o desenvolvimento deste projecto deverá obedecer às seguintes regras:

- **Deve ser em Java:** A aplicação tem, obrigatoriamente, que ser desenvolvida em Java pois será aplicada a servidores baseados em Java.
- **Não deve ser necessário alterar código da aplicação monitorizada:** As aplicações que serão monitorizadas estarão já compiladas e não deverão ser alteradas. Deve ser completamente transparente para essas aplicações o uso da aplicação de monitorização.
- **Deve ser transparente para o utilizador da aplicação monitorizada:** Os utilizadores das aplicações monitorizadas, que serão os clientes da Wipro Retail, devem continuar a utilizar as aplicações de forma normal, sem que se apercebam de quaisquer alterações. Todas as funcionalidades das aplicações monitorizadas devem manter-se intactas.

⁹ O nome dos clientes não é apresentado por razões de privacidade.

¹⁰ Arquitectura de software, em que a aplicação é dividida em 3 camadas: camada de apresentação, camada lógica e camada de dados.

- **Deve obter toda a informação de pedidos efectuados:** A aplicação de monitorização deve interceptar todos os pedidos efectuados à base de dados e guardar informação relativa a estes. Nenhum pedido deve ser descurado, de forma a obter estatísticas correctas nos relatórios finais.
- **Deve ser simples de usar, instalar e configurar:** A aplicação de monitorização deve ser simples de usar, instalar e configurar, tendo em atenção os utilizadores que a irão utilizar, que serão utilizadores com um enorme domínio nas áreas tecnológicas.
- **Deve ser parametrizável:** A aplicação de monitorização deve ser desenvolvida tendo em atenção as várias possibilidades de uso que lhe podem ser dadas. Nesse contexto deve ser o mais parametrizável possível.
- **Deve gerar relatórios com resultados da análise efectuada:** A aplicação deve gerar relatórios escritos e/ou gráficos dos resultados obtidos.
- **Deve ser possível ligar/desligar o sistema de *logging***¹¹: Deve ser possível iniciar ou parar o sistema de *logging* em tempo real. Desta forma poder-se-ão registar dados apenas quando assim for desejado.
- **Devem ser evitadas alterações de código:** Bibliotecas ou código *open source* utilizado não deve ser alterado. Sempre que possível, deverão ser estendidas as funcionalidades existentes e as alterações necessárias deverão ser efectuadas com recurso a *override*¹².

Estas regras definem a fronteira deste projecto e serão um estímulo à boa prossecução do mesmo.

2.5 Conclusões

Este capítulo apresenta o âmbito do projecto, descreve os seus objectivos e delimita a sua fronteira. As necessidades específicas da Wipro Retail e as características que uma boa ferramenta de monitorização deve apresentar são essenciais para a compreensão do problema em mãos, para a definição do estudo a efectuar e oferecem confiança em escolhas futuras.

¹¹ Processo de registo de eventos num sistema computacional. Normalmente é utilizado como método de *debugging* ou para permitir manter um histórico do estado do sistema.

¹² Característica das linguagens orientadas a objectos. Permite que uma subclasse especifique uma nova implementação de um método já fornecido por uma das suas superclasses.

3 Estado da Arte

Neste capítulo é feita uma análise de soluções idênticas àquela que se espera obter com este projecto, que são o estado da arte das ferramentas de monitorização, quer ao nível de soluções proprietárias quer ao nível de soluções *open source*. Esta análise servirá como base para a posterior comparação entre as ferramentas e para justificar a escolha feita e enunciada no final do capítulo.

3.1 Soluções Proprietárias

Com o início e crescimento das “*dot-com*” apareceram as primeiras ferramentas de monitorização de aplicações distribuídas. Muitas delas, com o passar do tempo, passaram de *open source* a ferramentas proprietárias e bastante completas. O crescimento da *web* e a procura de excelência de performance levaram a que estas ferramentas se tornassem essenciais para muitas empresas. Neste capítulo é feita uma análise a algumas das melhores ferramentas proprietárias disponíveis actualmente no mercado.

As ferramentas aqui analisadas são: PerformaSure[®] da Quest Software[®] [PSURE09], e dynaTrace[®] da dynaTrace Software[®] [DTRAC09]. Estas ferramentas foram escolhidas entre outras devido à qualidade elevada que apresentam. Apenas como referência, foram também avaliadas: Wily APM[®] da CA[®] [WILY09], Vantage Analyzer[®] da Compuware[®] [VANA09], JXInsight[®] da JInspired[®] [JXIN09] e Veritas Indepth[®] da Symantec[™]¹³ [VERIN09]. Nenhuma das ferramentas foi testada em situações reais já que estas são pagas e não são disponibilizadas versões de teste. Toda a análise feita foi baseada em artigos, documentação e registos de utilização por parte de outros utilizadores.

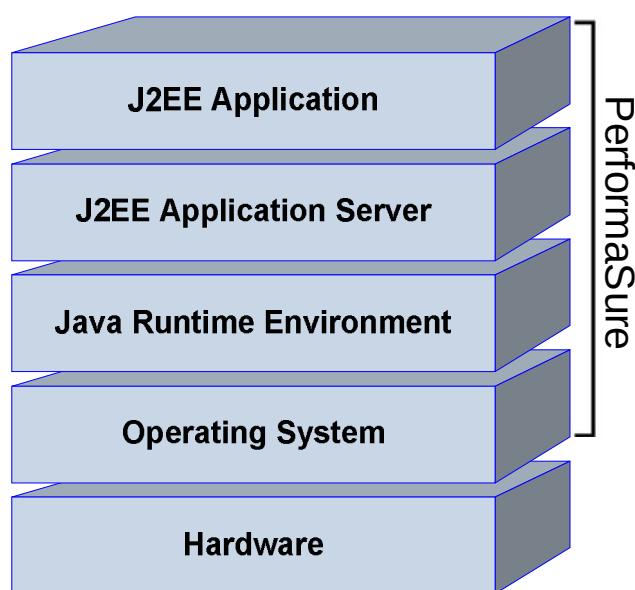
¹³ Esta aplicação foi descontinuada.

3.1.1 PerformaSure[®] da Quest Software[®]

Descrição

Esta ferramenta permite monitorizar aplicações J2EE, *n-tier*¹⁴, com *clusters*¹⁵, fornecendo dados relativos aos servidores *web*, servidores aplicativos e bases de dados possibilitando diagnósticos profundos. Combina dados transaccionais com infra-estruturas métricas que ajudam as equipas de desenvolvimento e qualidade de software a detectar a performance de uma aplicação, identificando locais problemáticos, providenciando formas de rastrear problemas de performance por vários componentes do sistema e assegurando que a aplicação está em conformidade com os tempos de resposta especificados [KMMS04].

A figura 3-1 ilustra as áreas de actuação do PerformaSure: aplicação, servidor aplicativo, ambiente onde corre a aplicação (Java runtime environment¹⁶) e sistema operativo.



The J2EE Environment

Figura 3-1 – Área de actuação do PerformaSure [KMMS04].

O PerformaSure usa uma tecnologia que permite determinar e rastrear o caminho transaccional de um pedido, mesmo que este passe por várias máquinas num cluster, figura 3-2. Seguindo esse caminho, podemos analisar as principais funcionalidades desta ferramenta.

O caminho natural iniciar-se-á nos servidores *web*, onde pode ser feita uma triagem aos tempos de reposta. Nesta fase, a ferramenta permite ver gráficos e tabelas com dados sobre os tempos de reposta a pedidos dos clientes e decompor esses mesmos dados por componentes. É fácil detectar graficamente zonas temporais susceptíveis de falhas de performance. Logo aqui, é possível ver o tempo gasto por cada pedido nas várias camadas da aplicação e detectar falhas em componentes específicos.

¹⁴ Arquitectura de software, em que a aplicação é dividida em *n* camadas, sendo o mais usual 3 camadas.

¹⁵ Conjunto de computadores que utiliza um sistema operacional distribuído. Usualmente todos os computadores do mesmo *cluster* apresentam as mesmas funcionalidades.

¹⁶ O Java runtime environment (JRE) fornece a Java Virtual Machine (JVM), um ambiente onde as aplicações Java executam.

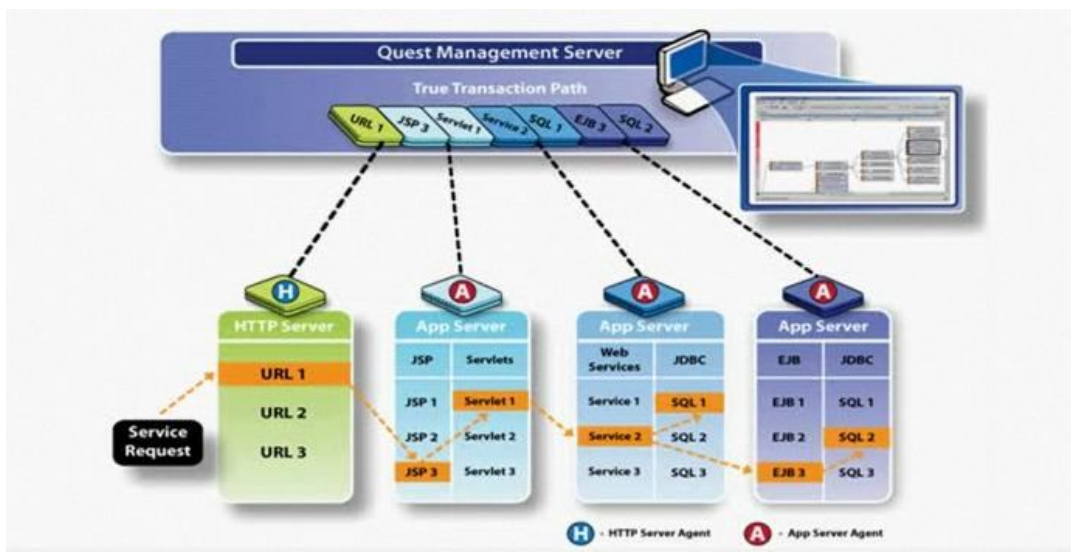


Figura 3-2 – Exemplo de um caminho transaccional de um pedido no PerformaSure [PSURE09].

O passo seguinte será analisar a árvore de chamadas de métodos para um pedido, figura 3-3. Aqui é possível ver todo o caminho de chamadas de métodos, assim como estatísticas sobre os tempos médios gastos em cada um.

Através da utilização de métricas definidas de forma automática ou manual é fácil detectar o caminho crítico de performance, assim como os pior e melhor casos de execução ou *bottlenecks* no caminho de execução. É possível ver métricas associadas a um determinado método, pedido, ou componente do sistema.

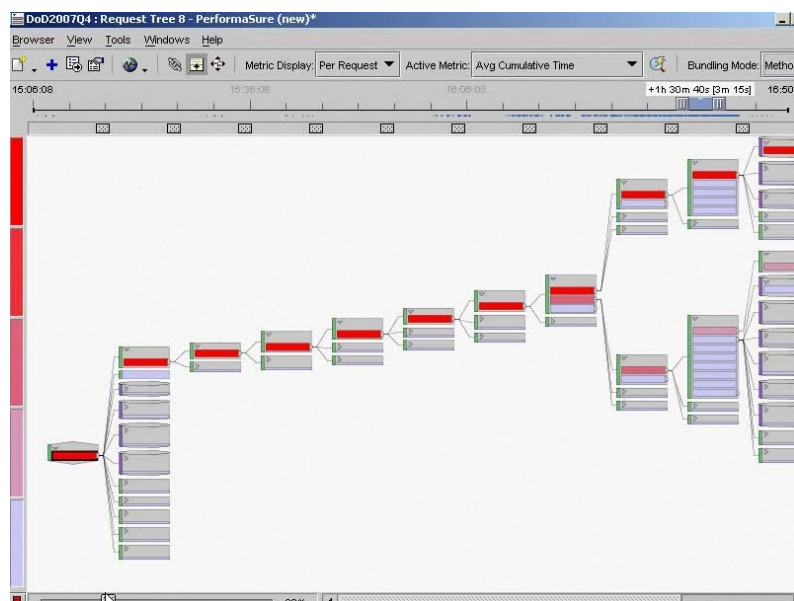


Figura 3-3 – Exemplo de árvore de chamadas de métodos no PerformaSure [PSURE09].

O próximo passo será detectar o maior problema em todo o caminho de execução. O PerformaSure tem uma funcionalidade de procura rápida, que faz isso automaticamente, para cada uma das métricas definidas. Através da ligação do PerformaSure com outra ferramenta da Quest Software, o JProbe, é possível criar automaticamente ficheiros de configuração, que podem depois ser utilizados pelas equipas de desenvolvimento, no JProbe, para testar o código a vários níveis de detalhe, até ao teste linha a linha do código. Outra forma de comunicar os

problemas às equipas de desenvolvimento é produzindo relatórios. O PerformaSure permite especificar qual a informação a ser gerada no relatório e exportá-la para os formatos PDF, XML e CSV.

Por fim, e como último passo, podemos aceder à vista de SQL, figura 3-4, onde podemos ver em detalhe todos os pedidos SQL, o número de execuções de cada pedido, os tempos máximo, mínimo e médio de execução, os tempos gastos em cada execução a preparar o pedido, a executar esse pedido e a devolver os dados de retorno. É possível ver o contexto em que estes pedidos SQL foram chamados e detectar onde e quando estes podem ser ineficientes.

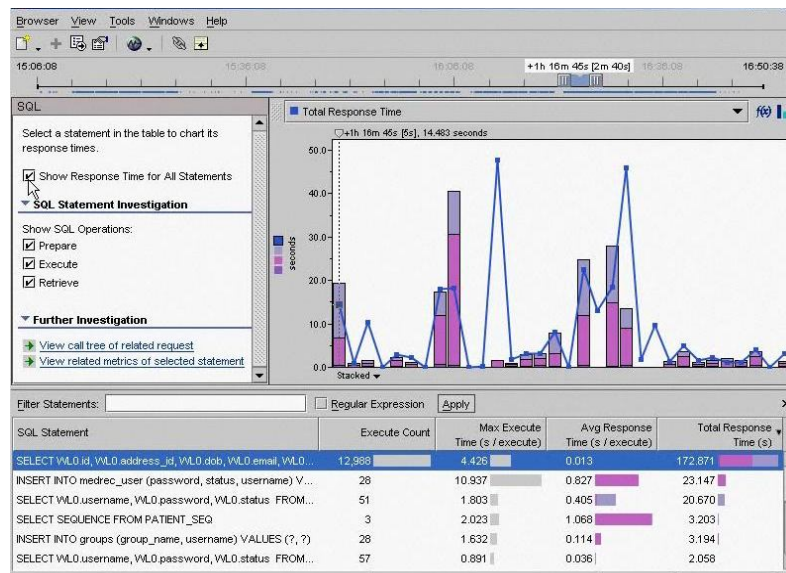


Figura 3-4 – Vista de SQL do PerformaSure [PSURE09].

Através deste exemplo de um caminho natural que seria seguido numa análise utilizando o PerformaSure foram demonstradas as principais funcionalidades desta ferramenta [PSURE09].

Principais Características

As principais características do PerformaSure são:

- Permite uma análise de todo o sistema, desde o pedido do cliente até à resposta da base de dados;
- Possibilita uma triagem rápida dos problemas;
- Possibilita a detecção prematura de problemas, possibilitando a sua correcção a um custo menor;
- Permite a monitorização durante todo o ciclo de vida de uma aplicação;
- Baseia-se em transacções de pedidos. A execução normal desta ferramenta é iniciar a análise nos pedidos do cliente, passar para uma análise na árvore de componentes e métodos e por fim para uma análise ao SQL;
- Permite detectar graficamente o caminho crítico de execução de um pedido;
- Permite gerar relatórios;
- Induz pouca sobrecarga nos sistemas. É possível definir o nível de detalhe de recolha de dados por parte da ferramenta para diminuir a sobrecarga;
- Apresenta várias vistas diferentes para vários tipos de análise;
- Permite correlacionar várias vistas, podendo, por exemplo propagar as opções escolhidas numa vista para todas as outras;
- Permite que a análise seja feita em *real-time* ou em *offline*;
- Permite a integração com o JProbe®, um *profiler* da Quest Software.

Avaliação

O PerformaSure é uma ferramenta bastante interessante e muito bem adaptada às necessidades dos sistemas distribuídos. Permite uma análise completa do sistema de uma forma intuitiva. O modo de navegação em cada uma das vistas e a ligação entre elas é simples e induz a uma boa utilização da ferramenta. Os gráficos interactivos são bastante úteis para diagnosticar rapidamente os problemas de performance. Esta ferramenta peca apenas por não permitir uma análise mais profunda e mais pormenorizada, podendo, por exemplo, detectar padrões ou deixar que o utilizador defina métodos de análise para detectar problemas específicos.

3.1.2 dynaTrace[®] da dynaTrace Software[®]

Descrição

Tal como o PerformaSure, o dynaTrace é uma ferramenta que permite a análise completa de um sistema distribuído. Esta ferramenta foi desenhada especificamente para monitorizar a performance de uma aplicação em todo o seu ciclo de vida. Existem mesmo quatro versões complementares da ferramenta: dynaTrace Development Edition, para a fase de desenvolvimento, dynaTrace Test Center Edition e dynaTrace Test Center Edition Standard para a fase de testes e dynaTrace Production Edition para a fase de produção. Cada uma destas versões possui características específicas para as fases a que estão destinadas permitindo assim que cada utilizador tenha disponível o que realmente necessita e nada mais.

O dynaTrace tenta ir para lá da aplicação monitorizada em si, e analisa todo o ambiente, desde a comunicação com outras aplicações a *web services*¹⁷. Desta forma é possível rastrear transacções individuais em ambientes heterogéneos, figura 3-5 [DTRAC09].

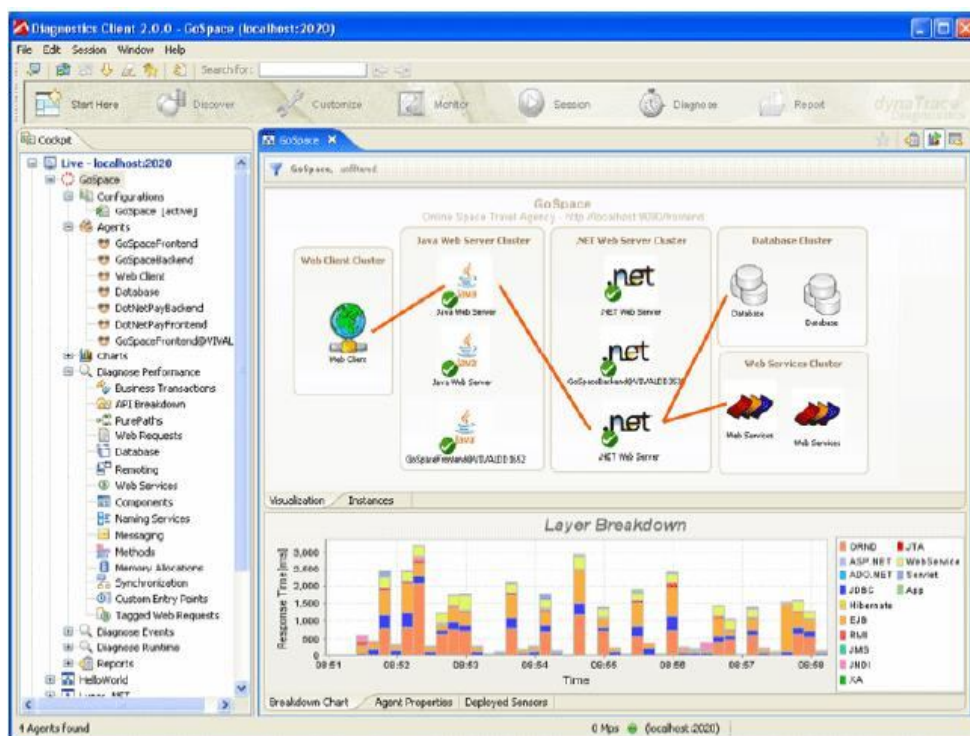


Figura 3-5 – Rastreo de uma transacção num ambiente heterogéneo com o dynaTrace [APD09].

¹⁷ Sistema de software desenhado para suportar interoperabilidade entre computadores numa rede.

A tecnologia PurePath® do dynaTrace regista todo o caminho de uma transacção. Esses dados são enviados para um servidor de diagnóstico e guardados num repositório. Desta forma, o dynaTrace consegue manter um histórico de monitorização para obter melhores estatísticas de performance. É possível reconstruir cada transacção desde o pedido do cliente, ao componente, ao método até à linha de código problemática, figura 3-6, [DTRAC09].

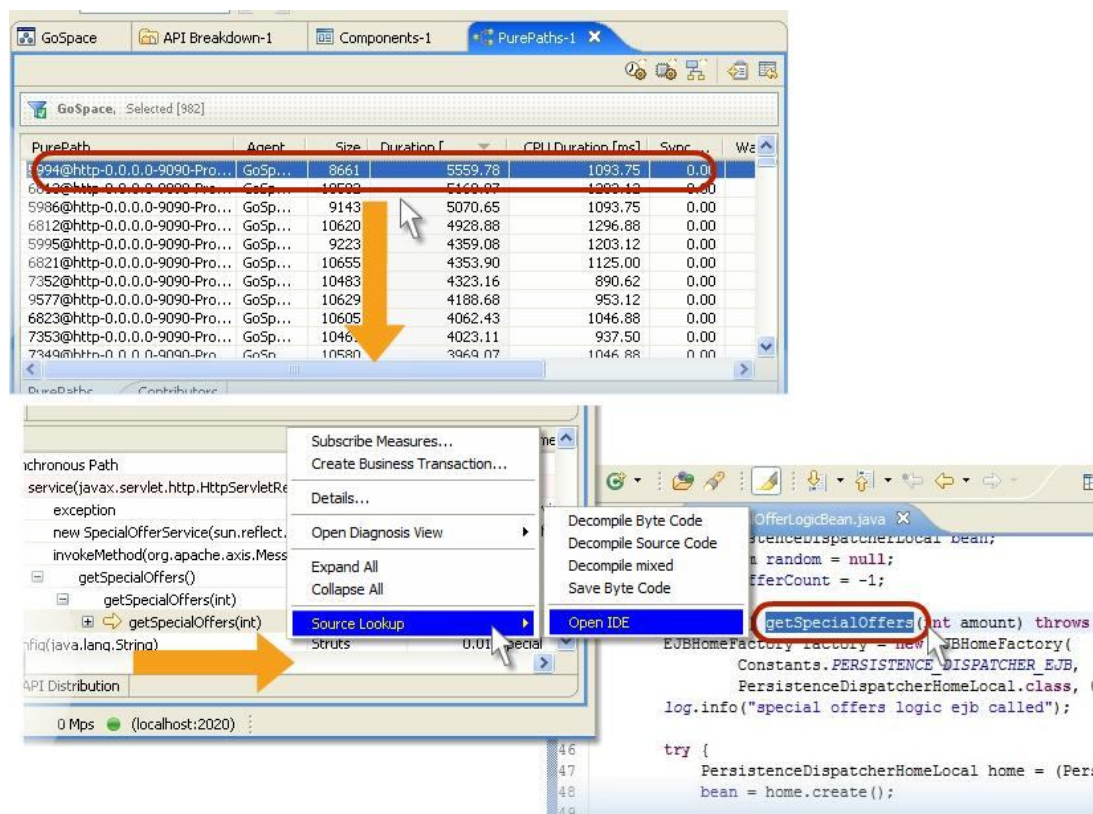


Figura 3-6 – Reconstrução do problema até à linha de código com o dynaTrace [DTRAC09].

Para cada transacção, podem ser registados: a sequência de chamada de métodos, os argumentos, valores de retorno, excepções e tempos de execução de cada um, os recursos utilizados, o tráfico de rede, chamadas remotas e atrasos de sincronização.

O dynaTrace apresenta valores de sobrecarga baixos, na ordem dos 3 a 5% [APPP09]. É ainda possível indicar um valor percentual de sobrecarga máximo que o dynaTrace poderá induzir na aplicação. Isto permite que a ferramenta seja utilizada constantemente de forma activa, mesmo em ambientes de produção com enorme carga. Nesta fase, a ferramenta é utilizada de uma forma mais automática, sem que seja necessária grande intervenção humana. Isto é possível devido aos pontos de alerta de risco que podem ser definidos na configuração do dynaTrace. O estado da aplicação pode ser consultado em dispositivos móveis e são enviadas mensagens de alerta quando um dos pontos de risco é atingido.

A versão de produção apresenta indicadores de performance de alto nível para disponibilizar uma rápida e fácil percepção do estado da aplicação, figura 3-7. Estes indicadores são ideais para a fase de produção da aplicação e para quando a monitorização é contínua. As outras versões focam-se mais na informação detalhada que é necessária para as fases de desenvolvimento e teste, figura 3-7.

O dynaTrace permite ainda especificar sensores adaptados ao problema ou à aplicação em questão e apresenta uma plataforma aberta que permite à comunidade ligar esta ferramenta com outras, como por exemplo o Eclipse ou o Visual Studio [DTRAC09].

3.2 Soluções Open Source

A exigência do mercado *web* levou a que as ferramentas de monitorização melhorassem consideravelmente e a que as empresas consumidoras deste tipo de software deixassem de apostar em aplicações *open source*. Muitas das ferramentas *open source* foram descontinuadas e hoje em dia praticamente nenhuma apresenta progressos.

As ferramentas aqui analisadas são: P6Spy [P6SPY03] e Elvyx [ELVYX08]. Foram também avaliadas: JAMon [JAMON07] e Craftsman Spy [CRAFT05]. Todas estas ferramentas têm apresentado poucos progressos sendo que a maioria está mesmo num estado descontinuado.

3.2.1 P6Spy

Descrição

O P6Spy foi uma das primeiras e mais populares ferramentas deste género. Esta é uma ferramenta simples, com um objectivo preciso: registar todos os pedidos efectuados à base de dados e tudo o que é retornado. A ideia básica do P6Spy é interceptar esses pedidos, para assim poder registá-los. Para conseguir isso, é utilizado um padrão de desenho de software denominado *proxy*. A ideia deste padrão é criar uma interface para algo, figura 3-8. Desta forma, é possível aceder ao objecto real através do proxy sem que sejam notórias quaisquer alterações [EGAM98].

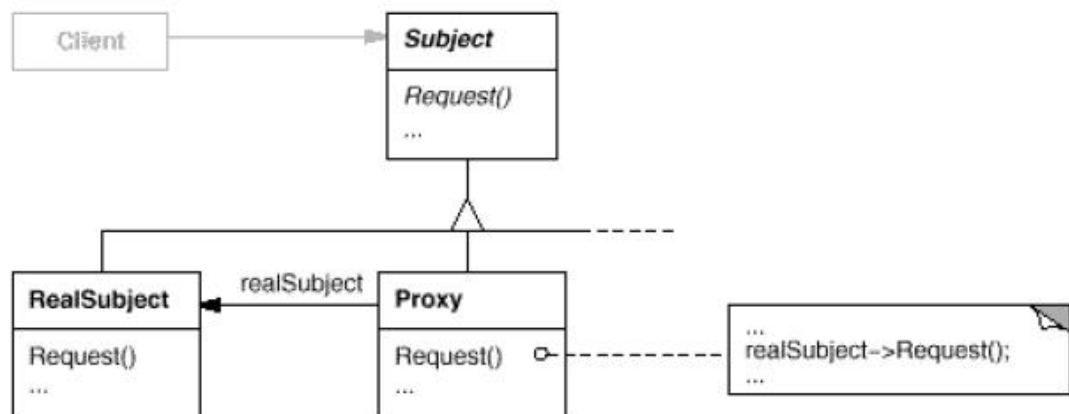


Figura 3-8 – Padrão de desenho de software: Proxy [EGAM98].

No caso do P6Spy, foi criada uma interface para o *driver* JDBC¹⁸ real. O *driver* JDBC é responsável pela comunicação de todos os pedidos à base de dados. O P6Spy regista os pedidos e os tempos de execução e depois chama o *driver* original para obter os dados.

Esta ferramenta tem dois módulos base: o P6Log e o P6Outage. O P6Log regista todos os dados num ficheiro de texto. Na figura 3-9 é representado um excerto de um ficheiro de *log* criado pelo P6Spy. Os registos de cada pedido têm a seguinte forma:

current time|execution time|category|statement SQL String|effective SQL string

¹⁸ Programa que permite uma interacção de alto nível com a base de dados

```
1243440324393|93|Statement|SELECT * FROM warehouse where id=1|SELECT * FROM warehouse where id=1|
1243440326411|56|PreparedStatement|SELECT * FROM warehouse where id=?|SELECT * FROM warehouse where id=1|
1243440334482|72|Statement|SELECT * FROM warehouse where id=1|SELECT * FROM warehouse where id=1|
1243440334498|34|PreparedStatement|SELECT * FROM warehouse where id=?|SELECT * FROM warehouse where id=1|
1243440336500|73|Statement|SELECT * FROM warehouse where id=1|SELECT * FROM warehouse where id=1|
1243440338502|34|PreparedStatement|SELECT * FROM warehouse where id=?|SELECT * FROM warehouse where id=1|
1243440338502|83|Statement|SELECT * FROM warehouse where id=1|SELECT * FROM warehouse where id=1|
```

Figura 3-9 – Excerto de um ficheiro de *log* criado pelo P6Spy.

O outro módulo, P6Outage, detecta pedidos com tempos de execução longos, e regista apenas esses pedidos. Desta forma permite evitar o tempo gasto no registo de todos os pedidos. O limite de tempo mínimo é configurável [P6SPY03].

Esta ferramenta foi utilizada anteriormente na Wipro Retail e foram introduzidas algumas inovações. Foi acrescentado o registo do caminho do pedido do cliente e foi criado um cliente *web* que dá *feedback* do estado da ferramenta. No entanto, esta ferramenta nunca foi realmente utilizada, pois as suas funcionalidades não são suficientes para as necessidades da empresa. Estas novas funcionalidades serão consideradas na escolha da ferramenta a utilizar.

Principais Características

As principais características do P6Spy são:

- Permite o registo de todos os pedidos feitos à base de dados;
- A sua utilização é transparente para o utilizador da aplicação monitorizada;
- Permite ligar/desligar o registo de pedidos;
- Tem um ficheiro de configuração onde a maioria das opções pode ser especificada;
- Não regista o caminho do pedido;
- Induz pouca sobrecarga nos sistemas;
- Encontra-se desactualizado.

Avaliação

O P6Spy é uma ferramenta simples com um conceito básico: registar todos os pedidos efectuados à base de dados. A sua simplicidade e eficácia tornam esta ferramenta bastante útil, principalmente quando o objectivo é uma monitorização simples. Os dados registados podem depois ser tratados noutras ferramentas.

Esta ferramenta foi descontinuada em 2003. A evolução tecnológica que ocorreu desde então força a que algumas alterações tenham que ser introduzidas no código para que seja possível utilizá-la na maioria dos servidores aplicativos actuais.

3.2.2 Elvyx

Descrição

O Elvyx é uma das mais recentes experiências da comunidade *open source* neste tipo de ferramentas. O conceito base desta ferramenta é idêntico ao do P6Spy. Através do padrão de desenho de software *proxy* foi criado um interface para o *driver* JDBC real permitindo registar todos os dados necessários como o pedido SQL efectivo ou tempos de execução. No entanto, esta ferramenta tenta uma aproximação maior às ferramentas proprietárias. Todos os pedidos são enviados para um servidor, que os regista numa base de dados interna. Existe ainda um cliente gráfico que permite ver os pedidos efectuados, gráficos de tempos de execução, gráficos

de distribuição de Gauss e outros dados importantes. A figura 3-10 mostra os componentes principais do Elvyx.

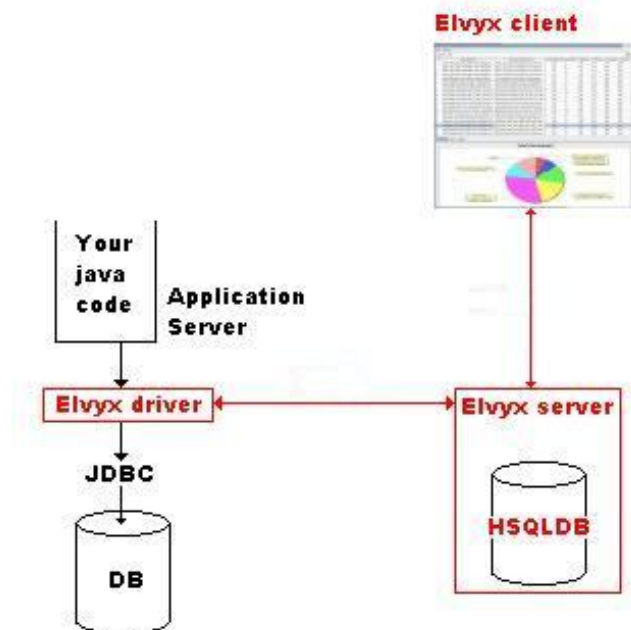
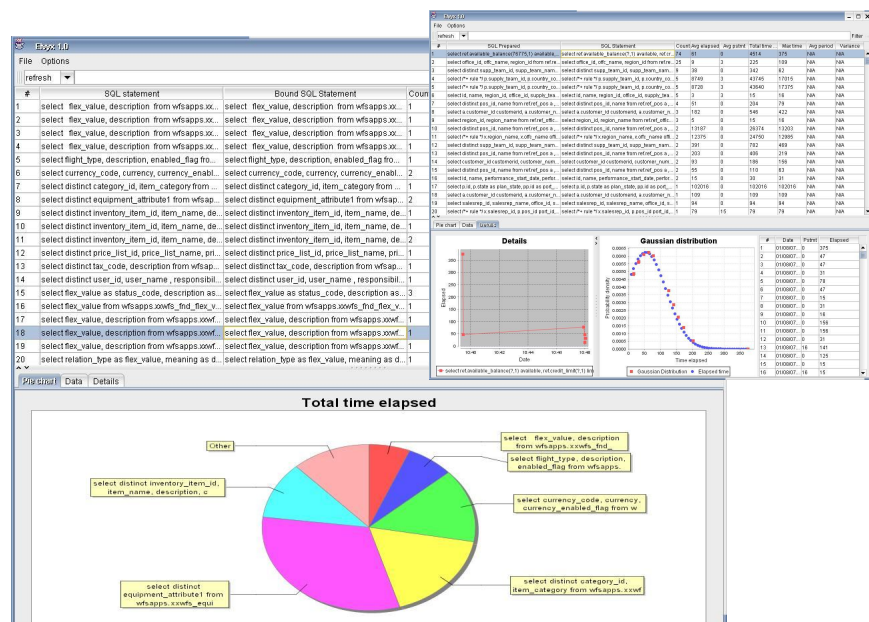


Figura 3-10 – Componentes principais do Elvyx [ELVYX08].

O servidor do Elvyx é a componente central da ferramenta. Todas as comunicações são feitas através dele. Este servidor inicia uma base de dados HSQldb – base de dados escrita em Java – que é carregada em memória quando é iniciado o servidor do Elvyx. Este mecanismo permite a persistência de dados para uma análise posterior.

O cliente do Elvyx faz pedidos constantes (tempo definido pelo utilizador) ao servidor para obter os dados da monitorização. Permite, assim, ver os resultados da monitorização quase em tempo real. O cliente pode ser iniciado sem que esteja a ser feita monitorização, permitindo analisar os dados que se encontram guardados na base de dados do Elvyx. A figura 3-11 apresenta algumas imagens do cliente gráfico.



Principais Características

As principais características do Elvyx são:

- Permite o registo de todos os pedidos feitos à base de dados;
- Cliente gráfico permite uma visualização amigável do processo de monitorização;
- A sua utilização é transparente para o utilizador da aplicação monitorizada;
- Apresenta persistência dos dados;
- Não regista o caminho do pedido;
- Induz uma sobrecarga considerável nos sistemas;
- Apresenta alguns erros no código;
- Comunidade pequena e apoio diminuto.

Avaliação

O Elvyx é uma ferramenta bastante completa e com uma arquitectura interessante. No entanto apresenta alguns problemas de performance e alguns erros que necessitam de ser corrigidos. A utilização desta ferramenta num ambiente onde os recursos disponíveis sejam mínimos e cuja utilização tem que ser cuidada é desaconselhável. Também o cliente gráfico, apesar de disponibilizar a informação de uma forma mais simpática, não apresenta um grande número de funcionalidades e não permite a manipulação dos dados para que se possam criar novas formas de análise aos mesmos.

A evolução desta ferramenta encontra-se parada desde de Fevereiro 2008. A comunidade que utiliza esta ferramenta é pequena e o apoio à mesma é diminuto.

3.3 Escolha Efectuada e Justificação

Como primeira análise, a comparação entre as ferramentas proprietárias com as ferramentas *open source*, demonstra que as grandes diferenças se encontram no melhor mapeamento pedido-método, no maior nível de detalhe apresentado e principalmente nas funcionalidades de análise de dados que as primeiras apresentam [JACKS03]. A tabela 3-1 faz uma comparação das quatro ferramentas estudadas tendo em atenção as características, enunciadas na caracterização do problema, que uma boa ferramenta de monitorização deve apresentar, segundo Jack Shirazi [JACKS03]. As notas atribuídas são de 0 a 5, sendo que 0 indica que não possui essa característica e 5 que atinge totalmente as expectativas.

Tabela 3-1 – Avaliação das ferramentas estudadas.

Característica	PerformaSure	dynaTrace	P6Spy	Elvyx
Componentes de monitorização e <i>logging</i>	4	5	2	2
Induz pouca sobrecarga no sistema	3	4	4	2
Mapeamento pedido-método	4	5	0	0
Granularidade e persistência dos dados registados	4	5	1	3
Escalabilidade	4	4	3	3
Persistência dos dados	5	5	0	5
Análise de dados	2	3	0	1

Feita esta comparação, é necessário definir qual a melhor ferramenta proprietária e qual a melhor ferramenta *open source*. Em relação às ferramentas proprietárias, qualquer uma das referidas, excluindo o Veritas Indeptyh, apresenta atributos suficientes para ser escolhida. O

Veritas Indepth será excluído desta análise por se encontrar em estado descontinuado. Qualquer uma permitirá uma boa monitorização e será uma mais-valia para a boa performance de um sistema. No entanto, o dynaTrace é superior a todas as outras em todos os aspectos referidos em cima. Antes de se poder afirmar que o dynaTrace é a escolha correcta, há ainda duas variáveis a analisar: o retorno sobre investimento (ROI) e o nível de detalhe de monitorização que é necessário, isto é, a adaptação da ferramenta às necessidades reais da Wipro Retail. Excluindo o JXInsight, nenhuma das ferramentas tem o preço discriminado, pois esse preço é negociado. Daí se subentende que esse preço será muito elevado. O preço da versão base, apenas para um computador, do JXInsight ultrapassa os €6.000. No que respeita às necessidades reais que a Wipro Retail apresenta em relação a uma ferramenta deste tipo, a análise completa e a monitorização constante não são necessárias. O foco maior é a análise à base de dados e ao caminho do pedido do cliente. A análise posterior dos dados e a liberdade de especificação de métodos de análise são, sem dúvida, os pontos essenciais que se desejam encontrar numa ferramenta deste género. Nenhuma das ferramentas se adapta totalmente às necessidades. Considerando estas duas variáveis, a Wipro Retail não vê um ROI suficiente em nenhuma das ferramentas para a poder considerar como uma opção plausível.

No que diz respeito às ferramentas *open source*, o P6Spy e o Elvyx são as duas ferramentas que mais se aproximam das necessidades da empresa. No entanto, ambas necessitam de modificações e inovações. A escolha efectiva por uma delas dever-se-á à quantidade e à facilidade de implementação dessas modificações. No entanto, a opção passará sempre por utilizar uma ferramenta *open source* que deverá ser melhorada para se adaptar totalmente às necessidades da empresa. Devem ser feitas propostas de solução para ambas as ferramentas e designados os pontos que determinarão a escolha da ferramenta final.

3.4 Conclusões

Este capítulo apresenta um estudo sobre as melhores ferramentas existentes actualmente no mercado para a detecção e resolução de problemas de performance em aplicações distribuídas que pretende servir de fonte para a escolha efectuada e para a definição do caminho a seguir no restante deste projecto.

O estudo evidencia que as ferramentas proprietárias se encontram num nível bastante elevado e abrangem todas as áreas que podem ser fonte de problemas de performance. Em relação às ferramentas *open source* conclui-se que estas não estão ao nível das proprietárias mas que podem servir como uma base interessante para novos desenvolvimentos nesta área. O facto de a evolução das ferramentas *open source* ter sido praticamente nula nos últimos anos indica que este mercado está a ser fortemente controlado pelas ferramentas proprietárias e que as soluções livres, no seu estado original, não são suficientes para a maioria das empresas. No entanto, analisando todas as variáveis, a melhor solução a tomar será adicionar novas funcionalidades a uma ferramenta *open source* e adaptá-la às necessidades da Wipro Retail.

4 Proposta de Solução

Após a análise ao estado da arte nas aplicações de monitorização é necessário escolher qual a abordagem ao problema. Neste capítulo são enunciadas as propostas de solução e as escolhas efectuadas. É ainda apresentada a análise tecnológica efectuada para a realização do projecto.

4.1 Especificação da Ferramenta – Wipro JDBC Spy

Como foi previamente referido, o P6Spy foi utilizado anteriormente pela Wipro Retail e foram aplicadas algumas modificações à ferramenta. Essas modificações foram: registo do caminho do pedido do cliente e criação de um cliente *web* que dá *feedback* do estado da ferramenta. Analisando esta versão da ferramenta P6Spy, a ferramenta Elvyx, e as necessidades actuais da Wipro Retail, os requisitos essenciais que devem ser concretizados no final do projecto são:

- **Driver JDBC actualizado:** É necessário ter um *driver* JDBC actualizado para que, através da utilização deste como *proxy*, se possa tirar partido de todas as funcionalidades do *driver* original;
- **Boa performance:** Apesar do objectivo principal desta ferramenta ser o seu uso nas fases de produção e teste, é necessária uma boa performance para evitar problemas acrescidos e uma má experiência de utilização;
- **Cliente web:** Deverá existir um cliente *web* que permita averiguar o estado da monitorização e ligar/desligar o modo de registo dos pedidos. O cliente local não é um requisito mas pode ser considerado uma mais-valia para o projecto;
- **Registo dos dados em ficheiro XML:** Deverá ser criado um ficheiro de registo dos dados em XML para facilitar a integração com outras ferramentas;
- **Registo do caminho do pedido:** Em conjunto com o registo dos pedidos SQL e dos tempos de execução deve ser registado também o caminho do pedido do cliente;
- **Persistência dos dados:** Deverão ser criados mecanismos de persistência de dados para que não seja necessário ler o ficheiro de registo sempre que se desejar correr as ferramentas de análise de dados;
- **Ferramentas extensíveis de análise dos dados:** Devem ser criadas ferramentas de análise de dados que possam ser estendidas mais tarde.

As ferramentas extensíveis de análise de dados são o objectivo essencial pois será através destas ferramentas que será efectuada a análise dos dados e obtidos os respectivos resultados da monitorização. A tabela 4-1 relaciona a versão da Wipro Retail do P6Spy e o Elvyx referindo

Proposta de Solução

quais destes requisitos fazem já parte ou não das funcionalidades da ferramenta. São ainda considerados outros aspectos importantes para a escolha da ferramenta a adoptar.

Tabela 4-1 – Tabela comparativa entre o P6Spy (versão Wipro Retail) e o Elvyx.

	P6Spy (Versão Wipro Retail)	Elvyx
Driver JDBC actualizado	Não	Sim
Boa performance	Sim	Não
Cliente web	Sim	Não
Registo em ficheiro	Sim	Não
Registo em ficheiro XML	Não	Não
Persistência dos dados	Não ¹⁹	Sim
Registo do caminho do pedido	Sim	Não
Ferramentas extensíveis de análise dos dados	Não	Não
Facilidade de extensão para cliente gráfico local	Não	Sim

Através da análise desta tabela e comparando com os requisitos enunciados em cima verifica-se que em relação ao P6Spy, versão Wipro Retail, é necessário: actualizar o *driver* JDBC, criar o registo dos dados em ficheiro XML, acrescentar persistência aos dados, e criar as ferramentas extensíveis de análise de dados; em relação ao Elvyx, é necessário: melhorar a performance da ferramenta, criar o cliente *web*, criar o registo dos dados em ficheiro XML, acrescentar o registo do caminho do pedido do cliente e criar as ferramentas extensíveis de análise de dados. Existem então duas novas funcionalidades em comum para ambas as aplicações: o registo dos dados em XML e as ferramentas extensíveis de análise de dados. O cliente *web*, a persistência de dados e o registo do caminho do pedido do cliente são funcionalidades que possivelmente poderão ser adoptadas de uma ferramenta para a outra. O principal motivo de decisão será a dificuldade de actualização do *driver* JDBC no caso do P6Spy ou a melhoria de performance no caso do Elvyx.

Após esta análise foram definidas duas propostas de solução. A primeira e inicial, utilizar o P6Spy implementando as melhorias referidas acima, iniciando o processo pela actualização do *driver* JDBC. Caso se verifique a impossibilidade ou dificuldade em completar a actualização do *driver* deve-se passar para o Elvyx, implementando as melhorias referidas acima, e iniciando o processo pela melhoria de performance da ferramenta. A principal razão pela qual foi definido este plano foi o facto de a Wipro Retail ter utilizado anteriormente o P6Spy e existir algum conhecimento interno sobre a ferramenta.

Em qualquer dos casos, espera-se obter uma ferramenta que possa funcionar em três fases, da seguinte forma: na primeira fase deve ser possível registar todos os pedidos num ficheiro XML e ter um *feedback* do estado da ferramenta; na segunda fase deve ser possível passar os dados do ficheiro XML para uma base de dados HSQldb de modo a criar persistência de dados e facilitar a terceira fase; na terceira fase deve ser possível correr as ferramentas de análise de dados que devem gerar um relatório dos problemas encontrados. Estas três fases são distintas e devem ser executadas em separado. A figura 4-1 esquematiza todas as fases e a ligação entre elas.

¹⁹ Não se considera o registo dos dados em ficheiro de texto como uma forma de persistência de dados, apesar de estes poderem ser reaproveitados mais tarde.

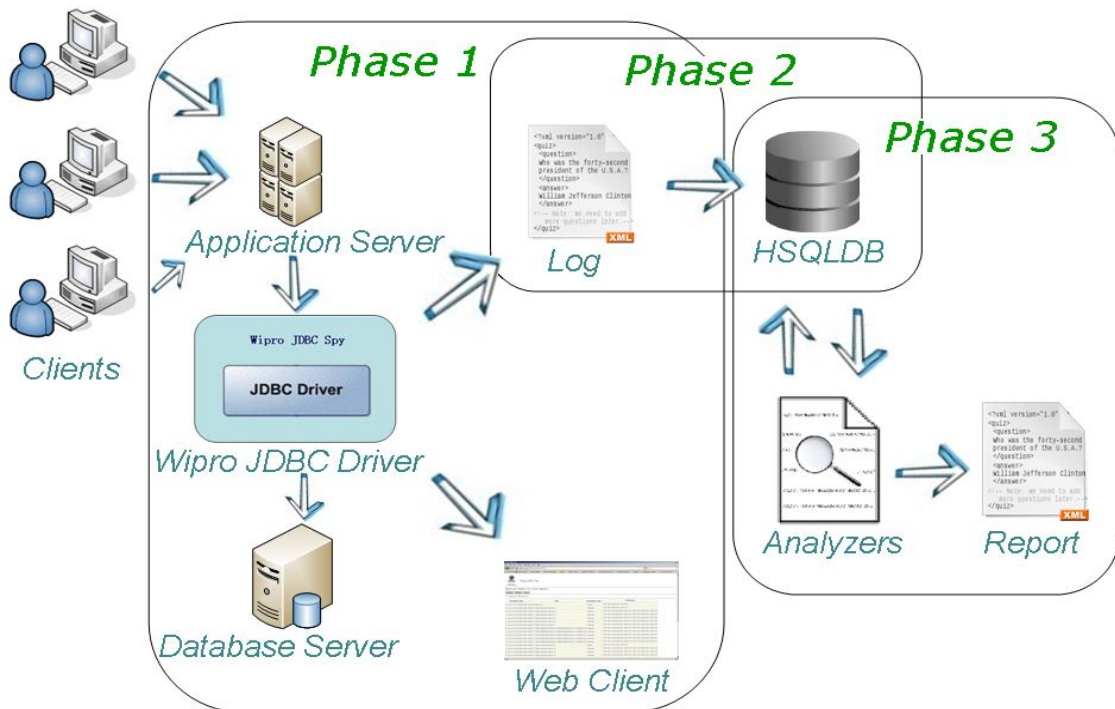


Figura 4-1 – Esquema da ferramenta esperada e divisão em 3 fases.

4.2 Primeira Proposta de Solução

A primeira proposta de solução envolve a ferramenta P6Spy. Os passos definidos são os seguintes, por esta ordem:

- Actualizar o *driver* JDBC;
- Registar os dados em ficheiro XML;
- Armazenar os dados numa base de dados HSQLDB;
- Criar ferramentas extensíveis de análise de dados.
- Melhorar o cliente *web*;

A actualização do *driver* JDBC foi definida como um *milestone* que decreta a continuidade desta proposta. O tempo máximo para atingir este objectivo deve ser de duas semanas. Como foi referido anteriormente, caso não seja possível ou seja elevada a dificuldade do cumprimento deste objectivo deve-se adoptar a segunda solução.

O segundo passo será registar os dados em ficheiro XML. Deve ser estudada a forma como é feito o registo de dados actualmente na ferramenta, e utilizar um método idêntico. A estrutura do ficheiro XML deve ser definida previamente e deve ter-se em atenção o facto de poder haver registos concorrentes no mesmo ficheiro já que poderão existir pedidos concorrentes ao *driver* JDBC. Devem ser realizados testes para verificar que pedidos concorrentes são registados correctamente.

O terceiro passo desta proposta será armazenar os dados numa base de dados HSQLDB. Os dados não deverão ser armazenados ao mesmo tempo que são registados mas apenas replicados do ficheiro XML para a base dados, quando o utilizador o desejar. O modelo da base de dados deve ser definido previamente. Deve ter-se em atenção que o número de dados a passar do ficheiro para a base de dados poderá ser bastante elevado pelo que deverá ser estudada a melhor forma de o fazer.

O quarto passo será criar ferramentas extensíveis de análise de dados. Devem ser criados métodos de análise de dados para problemas concretos. Esses métodos deverão analisar os dados contidos na base de dados e gerar um relatório dos problemas encontrados. Todas as variáveis de cada método de análise devem ser parametrizáveis de modo a poderem ser alargados a vários casos. A ferramenta deve ainda ser desenvolvida de modo a que no futuro possam ser criados novos métodos de análise de dados sem ser necessário o acesso ao código ou a recompilação do mesmo.

O quinto e último passo será melhorar o cliente *web* existente e adaptá-lo às novas funcionalidades da ferramenta.

4.3 Segunda Proposta de Solução

A segunda proposta de solução envolve a ferramenta Elvyx. Os passos definidos são os seguintes, por esta ordem:

- Melhorar a performance da ferramenta;
- Registar os dados em ficheiro XML;
- Acrescentar o registo do caminho do pedido do cliente;
- Armazenar os dados na base de dados HSQLDB existente, com acréscimo do caminho do pedido;
- Criar ferramentas extensíveis de análise de dados.
- Criar o cliente *web*;

A melhoria de performance do Elvyx foi definida como um *milestone* que decreta a continuidade desta proposta. O tempo máximo para conseguir este objectivo deve ser de duas semanas. Caso não seja possível ou seja elevada a dificuldade do cumprimento deste objectivo deve-se ponderar quais das duas soluções, se esta ou se a actualização do *driver* JDBC do P6Spy, é mais fácil ou tentar encontrar uma nova solução.

O segundo passo será registar os dados em ficheiro XML. Tal como na primeira proposta, deve ser definida, previamente, a estrutura do ficheiro XML, deve ter-se em atenção o facto de poder haver registos concorrentes no mesmo ficheiro e devem ser realizados testes para verificar que pedidos concorrentes são registados correctamente.

O terceiro passo desta proposta será acrescentar o registo do caminho do pedido do cliente. Devem ser registadas todas as funções chamadas pelo pedido do cliente, desde a função de chamada até à função que faz o pedido JDBC.

O quarto passo será armazenar os dados na base de dados HSQLDB existente. Deve ser estudado modelo de base de dados existente e modificado para que seja possível guardar o caminho do pedido do cliente. Os dados não deverão ser guardados na base de dados ao mesmo tempo que são registados no ficheiro XML mas sim replicados, mais tarde, desse ficheiro para a base de dados. Deve ter-se em atenção que ficheiro XML pode conter um número elevado de dados, pelo que deverá ser estudada a melhor forma de o fazer.

O quinto passo será criar ferramentas extensíveis de análise de dados. Tal como na primeira proposta, devem ser criados métodos de análise de dados para problemas concretos. Esses métodos deverão analisar os dados contidos na base de dados e gerar um relatório dos problemas encontrados. Todas as variáveis de cada método de análise devem ser parametrizáveis de modo a que estes possam ser alargados a vários casos. A ferramenta deve ainda ser desenvolvida de modo a que no futuro possam ser criados novos métodos de análise de dados sem ser necessário o acesso ao código ou a recompilação do mesmo.

O sexto passo desta proposta será criar um cliente *web* que permita averiguar o estado da monitorização e ligar/desligar o modo de registo de dados.

4.4 Solução Adoptada: Segunda Proposta de Solução

O estudo efectuado inicialmente no sentido de utilizar o P6Spy, versão Wipro Retail, tendo como objectivo primário a actualização do *driver* JDBC envolvido nesta ferramenta e as dificuldades iniciais de adaptação à ferramenta demonstraram que a actualização do *driver* poderia ser complicada, o que levou a que fosse testada a segunda proposta de solução. A primeira proposta não foi imediatamente descurada, mas sim posta de lado, até se confirmar a possibilidade de melhoria de performance do Elvyx. Apesar de a actualização do P6Spy estar ao alcance, o tempo dispendido a estudar o Elvyx indicou que esta ferramenta estaria mais próxima do resultado esperado, e que não seria muito complicado melhorar a performance desta ferramenta. Por estes motivos foi abandonada a primeira proposta de solução, sendo adoptada a segunda proposta.

4.5 Análise Tecnológica

Na implementação das melhorias propostas na solução adoptada estarão envolvidas várias ferramentas essenciais para atingir os objectivos esperados. Nesta secção é feita uma análise tecnológica às principais ferramentas utilizadas. Com esta análise pretende-se justificar as escolhas tecnológicas efectuadas.

4.5.1 O Driver JDBC

As aplicações utilizadas e implementadas pela Wipro Retail são, na sua maioria, em Java, ou correm em plataformas Java. Os programas Java utilizam JDBC, *Java Database Connectivity*, para efectuar todas as operações nas bases de dados. O JDBC é um API *standard*, sobre o qual cada fornecedor de bases de dados (neste caso a Oracle) fornece os *drivers* correspondentes e necessários. O *driver* JDBC disponibiliza todas as funcionalidades necessárias para comunicação com as bases de dados. Este encontra-se, então, na posição ideal entre a aplicação e a base de dados. Nesta posição, todos os pedidos, assim como os tempos de execução, resultados retornados e outros dados encontram-se disponíveis. Com o controlo do *driver* JDBC tem-se acesso a toda essa informação, sem que seja necessário modificar o código da aplicação ou até mesmo ter acesso a ele, figura 4-2 [DBOOK].

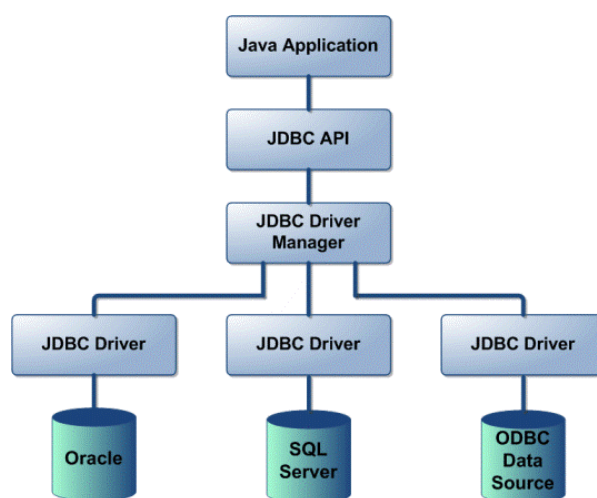


Figura 4-2 – Arquitectura JDBC. Posição do Driver JDBC [DBOOK].

Proposta de Solução

O *driver* JDBC é constituído por vários componentes, essenciais na comunicação com as bases de dados. Os principais componentes do *driver* JDBC e a sua relação estão representados na figura 4-3.

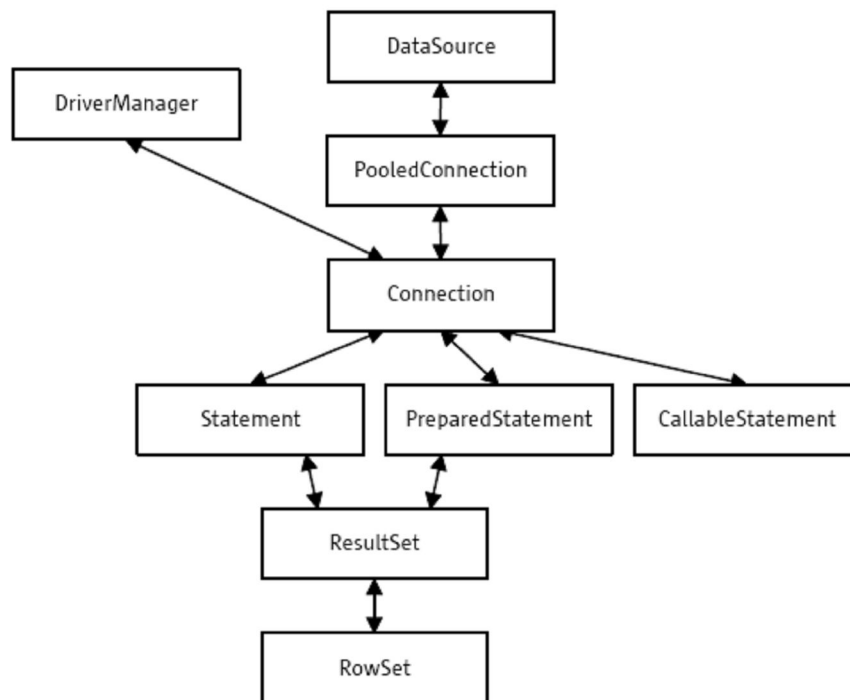


Figura 4-3 – Relação entre os principais componentes do Driver JDBC [JUGANL].

DataSource é um objecto que representa uma fonte de dados e fornece métodos de armazenamento e obtenção de dados. Algumas vantagens que se obtêm com a utilização de *DataSource* são a referência através de JNDI, a não necessidade de ter a informação especificada no código (*hardcoded*) e possibilidade de tirar partido de *pooling* de conexões²⁰ e transacções distribuídas²¹.

PooledConnection é um mecanismo gerido pela base de dados que permite reutilizar conexões criadas. Desta forma, a performance pode ser verdadeiramente incrementada.

DriverManager gere uma lista de *drivers* e compara pedidos de ligação da aplicação Java com o *driver* correcto utilizando um protocolo de comunicação. O primeiro *driver* que reconhecer o protocolo será utilizado para estabelecer a ligação à base de dados.

Connection é uma interface com todos os métodos de comunicação com uma base de dados. Toda a comunicação com a base de dados é feita através de um objecto *connection*.

Statements são objectos através dos quais se podem enviar comandos SQL à base de dados.

*PreparedStatement*s são *statements* pré-compilados na base de dados. Através da pré-compilação, os *preparedstatements* melhoram a performance de comandos SQL que são executados várias vezes.

CallableStatements são utilizados para invocar procedimentos e funções existentes na base de dados.

ResultSet representa um conjunto de linhas de dados existentes na base de dados e retornado como resposta a um pedido feito à mesma.

²⁰ Pooling de conexões pode ser visto como uma cache de ligações. Estas ligações são mantidas pela base de dados para que possam ser reutilizadas permitindo assim ligações mais rápidas do cliente à base de dados

²¹ Transacções distribuídas são transacções em bases de dados em dois ou mais computadores ligados em rede.

RowSet é um objecto que encapsula um conjunto de linhas de dados existentes quer numa base de dados quer num objecto *datasource* de outro tipo [DBOOK].

As ferramentas *open source* estudadas, através do uso do padrão *proxy*, criam novas implementações de todos estes componentes e dos respectivos métodos. Estas novas implementações chamam as classes e os métodos originais para executarem todas as acções. Este tipo de abordagem denomina-se *wrapping* e pode-se comparar a uma espécie de invólucro criado à volta das classes originais.

4.5.2 Apache log4j

O log4j é uma ferramenta escrita em Java que fornece funcionalidades que permitem fazer *logging* de dados. Esta ferramenta é formada por 3 componentes principais: *Loggers*, *Appenders* e *Layouts*. Os *Loggers* são o centro da ferramenta. Estes definem o que deve ou não ser registado e fornecem ao programador o controlo sobre o que será registado em tempo de execução. Os *Appenders* são os componentes que registam efectivamente os dados. Existe um variado número de *Appenders* que permitem registar os dados de várias formas e em vários locais, por exemplo, em ficheiro, na consola ou enviando-os através de um *socket*²². Os *Layouts* definem a formatação das mensagens escritas e são utilizados pelos *Appenders* aquando do registo dos dados.

O log4j foi desenvolvido com três objectivos em mente: flexibilidade, rapidez e confiança. A flexibilidade deve permitir registar dados diferentes para situações diferentes. Isto é conseguido através de herança, que é uma característica dos *Loggers*. Esta característica permite controlar quais as mensagens a serem registadas com grande facilidade, o que ajuda a reduzir o volume de dados a serem registados. Existem cinco níveis definidos por defeito para os *Loggers* que permitem fazer a selecção das mensagens. Esses níveis são, em ordem crescente de prioridade: DEBUG, INFO, WARN, ERROR e FATAL. Existem ainda os níveis ALL e OFF que fazem com que todas as mensagens ou nenhuma mensagem seja registada, respectivamente. A figura 4-4 mostra a relação entre os vários níveis.

Will Output Messages Of Level

	DEBUG	INFO	WARN	ERROR	FATAL
DEBUG					
INFO					
WARN					
ERROR					
FATAL					
ALL					
OFF					

Figura 4-4 – Hierarquia dos *Loggers* no log4j [LOG4J07].

A rapidez deve permitir fazer *logging* sem prejudicar gravemente a performance da aplicação. O log4j foi optimizado para ser rápido. “Num AMD Duron a 800Mhz a correr o Java JDK 1.3.1, demora 5 nanossegundos a determinar se uma mensagem deve ser registada. O registo em si é também bastante rápido, variando entre 21 microssegundos (...) e 37 microssegundos.” [LOG4J07].

Para ser de confiança o log4j deve executar sem danificar a aplicação e registar todos os eventos necessários. O log4j não garante o registo de todas as mensagens em todas as situações, pois esse registo não depende apenas do log4j (por exemplo, se a mensagem for enviada através de um socket o log4j não pode garantir que essa mensagem é lida no outro lado do socket) no

²² Extremo de um processo bidireccional de comunicação numa rede de computadores

entanto, quando o registo é em ficheiro e esse ficheiro existe e permite escrita é bastante improvável ocorrerem falhas de registo. São ainda garantidos que o log4j não irá, em altura alguma, fazer parar a aplicação e também que executará correctamente em chamadas concorrentes, ou seja, é *thread-safe* [LOG4J07]. A definição das propriedades do log4j é feita num ficheiro de propriedades que pode, facilmente, ser modificado.

O modo de funcionamento do log4j está representado na figura 4-5 e é o seguinte:

1. A aplicação usa um *Logger* para fazer o registo da mensagem, fornecendo-lhe o nível da mensagem;
2. O *Logger* verifica no ficheiro de propriedades se deve registar essa mensagem, dependendo se o nível do *Logger* é igual ou superior, em hierarquia, ao da mensagem;
3. Em caso negativo não é registada a mensagem;
4. Em caso positivo, o *Logger* chama o *Appender* associado para registar a mensagem no local definido nas propriedades;
5. O *Appender* regista a mensagem, utilizando o *Layout* definido nas propriedades.

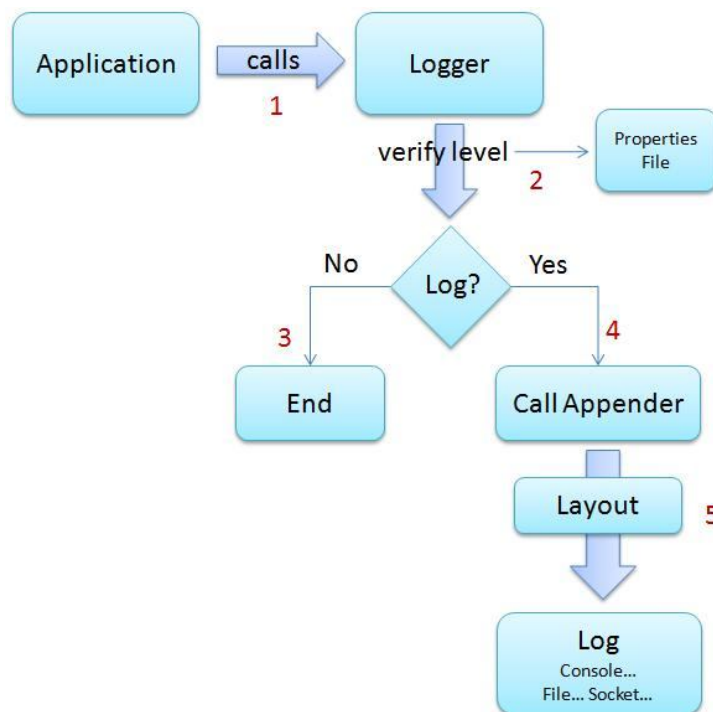


Figura 4-5 – Modo de funcionamento do log4j.

4.5.3 Parser XML: SAX vs DOM

Uma das etapas do projecto será ler os dados registados no ficheiro XML e inseri-los na base de dados HSQLDB. Apesar de não ser a opção mais eficiente, esta funcionalidade deve ser implementada em Java, pois não é essencial a rapidez nesta fase e evita a necessidade de instalação de outras plataformas. Outras soluções podem ser pesquisadas como parte de trabalho futuro. Para fazer *parsing*²³ do ficheiro XML de forma eficiente é necessário estudar os vários métodos para exercer essa tarefa e as ferramentas disponíveis.

O *parsing* de um ficheiro XML é, normalmente, efectuado com recurso a um de dois métodos distintos: SAX ou DOM. O SAX, é um API que facilita a leitura ou modificação de um

²³ Análise de um ficheiro e divisão do mesmo em pequenas porções de dados que podem mais facilmente ser armazenadas ou manipuladas.

ficheiro XML definindo um *handler*²⁴ que coloca os dados em elementos/atributos durante o processamento do ficheiro. Utilizando DOM, é criada uma representação em árvore do documento, facilitando o acesso aleatório e o controlo sobre o mesmo. O DOM, apesar de normalmente ser mais fácil de implementar, é mais lento e gasta mais recursos que o SAX [BBKCOL]. Num artigo de Padma Apparao sobre qual o *parser* mais indicado para Java [APPAR08], é demonstrado que o SAX tem uma performance bastante melhor que o DOM. Os gráficos das figuras 4-6 e 4-7 comparam o número de *tags* XML²⁵ processadas por segundo e o valor de memória consumido em Megabytes, respectivamente, utilizando SAX e DOM.

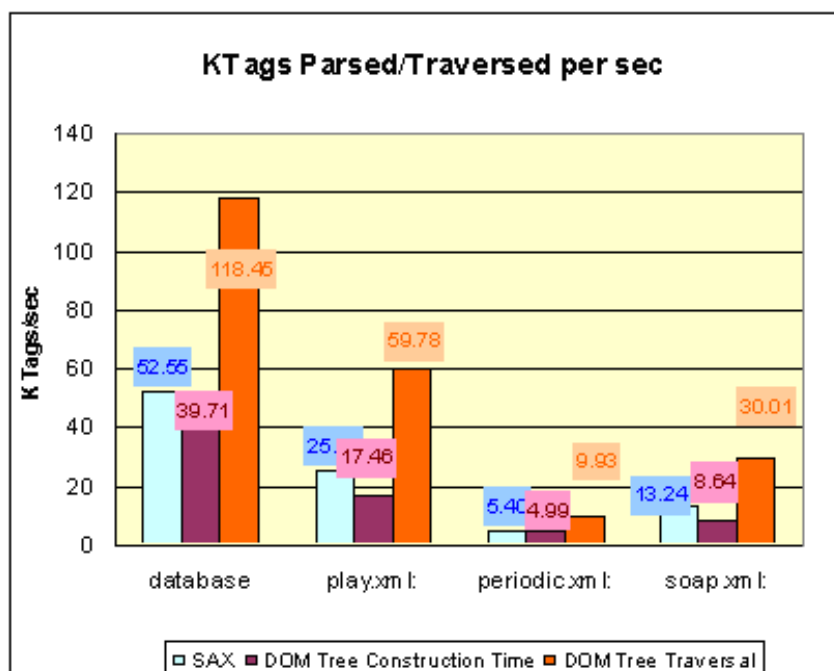


Figura 4-6 – Número de *tags* XML processadas por segundo utilizando SAX e DOM [APPAR08].

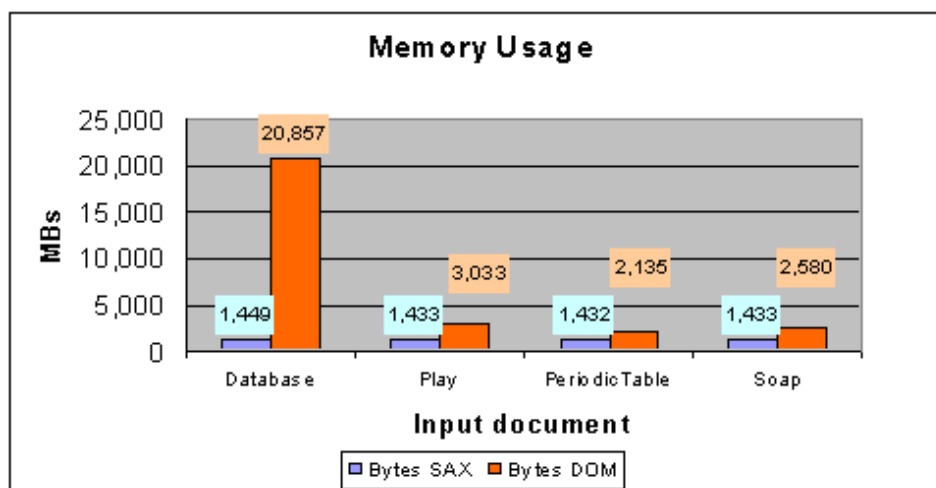


Figura 4-7 – Megabytes de memória consumidos por SAX e DOM [APPAR08].

²⁴ Subrotina que controla uma situação particular como um evento ou uma exceção.

²⁵ Palavra-chave ou termo atribuído um tipo de informação. Em XML, o uso de *tags* intercaladas define a estrutura da árvore XML.

Dado que neste projecto não é necessário ter acesso aleatório ou grande controlo sobre o ficheiro XML e ainda tendo em atenção que o tamanho do ficheiro pode ser bastante elevado, deve utilizar-se o método mais rápido e que gasta menos recursos, o SAX.

Existem várias ferramentas que permitem usar SAX para fazer o *parsing* de um ficheiro XML, entre elas: JAXP [JAXP09], Xerces-J [XERC08] e o Commons Digester [DIGES09]. Fazendo uma análise comparativa entre a performance e a facilidade de implementação das ferramentas indicadas e as necessidades do projecto, deve utilizar-se o Commons Digester devido, principalmente, à facilidade de utilização da ferramenta e à transparência da mesma. Este facto pode ser importante, caso, no futuro, seja implementada uma outra forma de leitura dos dados do ficheiro XML e consequente inserção na base de dados HSQLDB.

4.5.4 Base de Dados HSQLDB

HSQLDB (Hyper Structured Query Language Database) é um sistema de gestão de base de dados relacional escrito em Java. Este sistema permite que uma base de dados seja utilizada de forma totalmente integrada numa aplicação, sendo transparente ao utilizador e evitando a necessidade de utilizar outras aplicações ou outros sistemas de gestão de base de dados. O HSQLDB oferece um motor de base de dados rápido e pequeno que pode ser carregado em memória ou com base em disco. Suporta ainda a utilização em modo embebido, onde apenas a aplicação onde se encontra embebido tem acesso, ou em modo servidor, onde pode ser acedido por várias aplicações. O armazenamento de dados pode ser efectuado em dois modos: *memory* ou *cache*. Utilizando o modo *memory* obtém-se um melhor desempenho no entanto não é indicado para tabelas com muita informação. O modo *cache* tem um desempenho um pouco pior no entanto permite que seja armazenado um número mais elevado de dados em cada tabela [HSQLDB]. Analisando as diferenças de desempenho entre ambos os modos verifica-se que as perdas de desempenho no modo *cache* não são significativas para o problema em questão pelo que deverá utilizar-se este modo.

A utilização desta ferramenta será essencial pois facilitará a análise aos dados a ser efectuada pelas ferramentas extensíveis de análise de dados e permitirá um aumento enorme de performance dessas mesmas ferramentas quando comparado com a leitura dos dados do ficheiro XML.

4.5.5 Servidor Web: NanoHTTPD

Uma das fases deste projecto envolve a criação de um cliente *web* que permita dar *feedback* ao utilizador do estado da monitorização e da ferramenta. Para esse cliente *web* funcionar correctamente é necessário existir um servidor *web* que processe os pedidos efectuados pelo cliente. Um servidor *web* deve, entre outras coisas, poder receber pedidos de clientes através de um *web browser* e enviar a resposta para o mesmo *web browser*, figura 4-8.

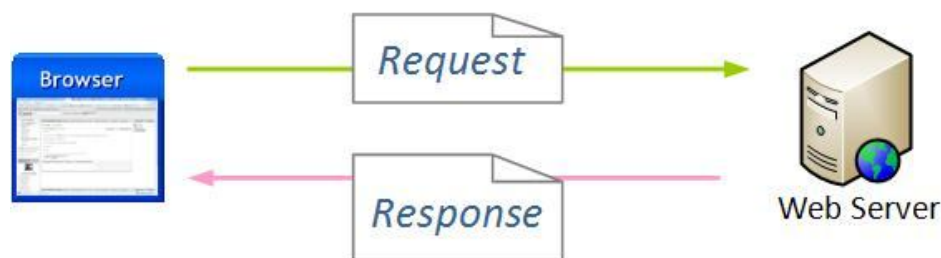


Figura 4-8 – Exemplo de uma sequência pedido/resposta a um servidor Web.

Existem inúmeros servidores *Web*, *open source*, em Java. Na sua maioria são bastante completos e apresentam bastantes funcionalidades. Os seguintes são apenas alguns exemplos de servidores *web*, *open source*, escritos em Java: Xerver [XERVER], JLHTTP [JLHTTP], Jetty [JETTY09], VQEmbWeb [VQEWEB], NanoHTTPD [NHTTPD]. Na ferramenta em questão, o desejado é ter um servidor *web* simples e embebido na própria ferramenta.

O NanoHTTPD é ideal para este projecto devido às características que apresenta:

- Apenas um ficheiro Java;
- Compatível com versões Java desde a versão 1.1;
- Suporta passagem de parâmetros através dos métodos GET e POST;
- Pequeno e rápido no processamento das respostas;

4.5.6 Java Reflection API

Java Reflection é um API que permite, tal como o nome sugere, que uma classe ou um objecto se possa auto-examinar. Através deste API é possível que um programa altere o seu estado em tempo de execução, podendo aceder a código anteriormente desconhecido. Apesar de ser menos seguro e mais lento (aproximadamente duas vezes mais lento) que o código normal, Java Reflection possibilita algo que não é possível obter sem este API: possibilita que código, possivelmente desconhecido na altura de compilação, seja executado dinamicamente. “Reflection pode ser utilizado quando se necessita de trabalhar com objectos desconhecidos previamente.” [NIKNU05] Esta propriedade será essencial na terceira fase deste projecto. Na figura 4-9 estão representadas as classes principais de Java Reflection.

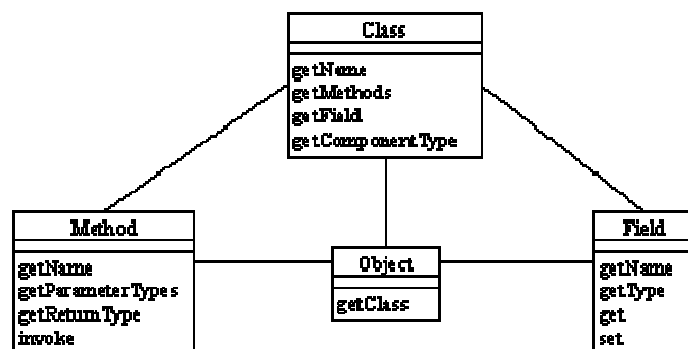


Figura 4-9 – Classes principais de Java Reflection.

4.6 Conclusões

Neste capítulo são apresentadas as prioridades e a ferramenta final que espera obter, assim como duas propostas de solução distintas e com uso de ferramentas *open source* diferentes. É apresentada qual a opção tomada e quais as razões para essa escolha. Após a apresentação dessa escolha, é exposto o estudo efectuado a todas as tecnologias necessárias para executar a proposta de solução adoptada. Este estudo é essencial para a compreensão dos passos tomados na fase de implementação e das escolhas efectuadas.

5 Implementação da Solução

Neste capítulo é feita uma descrição pormenorizada de todos os passos efectuados na implementação da solução adoptada. Todas as escolhas efectuadas são descritas aqui, assim como as maiores dificuldades encontradas. A sequência de implementação é descrita tal como foi efectuada desde o início do projecto. Para melhor compreensão e ajuste ao que foi descrito na proposta de solução, a descrição da implementação é dividida em três grandes partes, correspondentes às três fases distintas da ferramenta. A única excepção é o cliente *web* que, embora pertencendo à fase 1 da ferramenta, foi implementado no final do projecto por ter uma prioridade menor.

5.1 Melhoramento da Performance do Elvyx

A segunda proposta indicava que a melhoria de performance do Elvyx era o ponto essencial a atingir como primeiro objectivo. O estudo inicial da ferramenta Elvyx demonstrou que eram utilizados bastantes recursos em funcionalidades que não eram importantes para este projecto. Esses recursos estavam ligados essencialmente a funcionalidades externas ao *driver* JDBC, nomeadamente, ao servidor iniciado pelo Elvyx, ao cliente gráfico e principalmente ao modo de registo imediato dos dados na base de dados HSQLDB. Foram também detectados problemas nesta última funcionalidade, já que nem todos os pedidos eram registados. Após algum estudo da ferramenta para ver o funcionamento da mesma foi decidido retirar as funcionalidades referidas do modo de execução normal da ferramenta. O servidor e o cliente do Elvyx podem ser iniciados na mesma, mas apenas por opção do utilizador e não obrigatoriamente, como acontecia anteriormente. Também a funcionalidade de registo imediato dos dados na base de dados HSQLDB foi eliminada já que esse não é o objectivo deste projecto. Os dados devem ser registados, inicialmente, em ficheiro XML e só depois replicados na base de dados. Este processo melhorou consideravelmente a performance da ferramenta. Após terem sido retiradas estas funcionalidades do modo de execução normal do Elvyx verificou-se que a performance aumentou drasticamente. O primeiro passo da segunda proposta de solução tinha sido, assim, atingido.

5.2 Fase 1: Registo dos Dados em Ficheiro XML

A fase 1 da ferramenta implica o registo dos dados em ficheiro XML. Neste capítulo estão descritos todos os passos que foram efectuados para atingir este objectivo. Ainda como parte da fase 1 existe o cliente *web*. No entanto, e como foi já referido anteriormente, a descrição aqui feita segue o caminho normal da implementação efectuada. O cliente *web* foi implementado após a conclusão das três fases pois não era parte essencial da ferramenta. Por esse motivo a sua implementação é descrita apenas no final deste capítulo. A figura 5-1 esquematiza o resultado esperado no final da implementação desta fase.

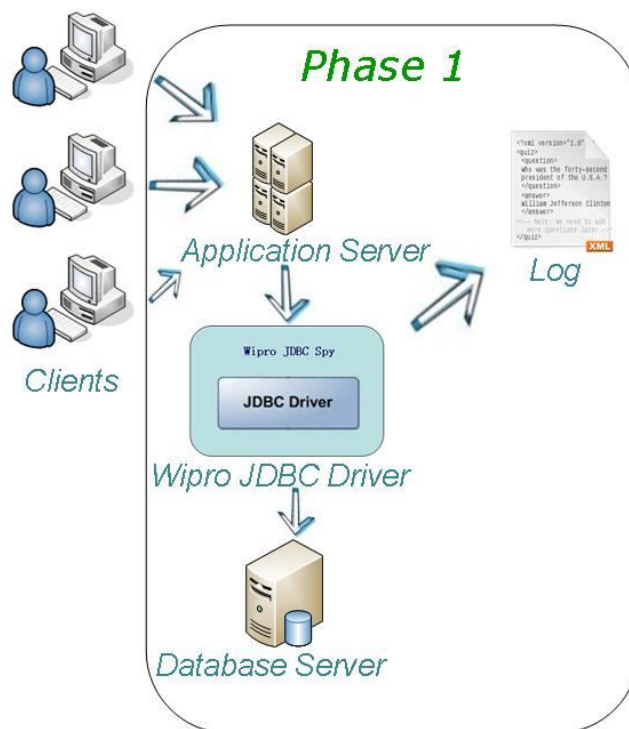


Figura 5-1 – Esquema das funcionalidades a implementar na fase 1.

5.2.1 Registo do Caminho do Pedido do Cliente

O primeiro passo tomado foi acrescentar a capacidade de detectar o caminho do pedido do cliente. Para conseguir isso, a solução adoptada foi lançar um objecto “Throwable”. Este objecto recolhe a informação de todos os métodos chamados até ao ponto no código onde este é lançado. Desta forma, obtém-se o caminho do pedido do cliente até ao primeiro método, que será aquele em que foi feito realmente o pedido. Este caminho tem que ser “tratado” pois o objecto “Throwable” é lançado dentro de um método de uma classe do Elvyx. Todos os métodos pertencentes a classes do Elvyx são retirados do caminho do pedido do cliente, pois não fazem parte do caminho real.

5.2.2 Dados a Registrar

O passo seguinte desta fase foi definir quais os dados a registar. É importante registar todos os dados necessários actual e futuramente, mas também é importante considerar apenas dados realmente necessários, pois quanto maior for o número de dados maior será a sobrecarga provocada no sistema. A tabela 5-1 contém a escolha dos dados a registar e uma pequena descrição de cada um. A sigla é o nome usado no ficheiro XML.

Tabela 5-1 – Dados registados no ficheiro XML.

Dado de Registo	Sigla	Descrição
Id de Conexão	ConnID	Identificador de conexão do cliente. Utilizado para distinguir pedidos idênticos de clientes diferentes.
Data Inicial	InitDate	Data de início de ligação do cliente ao servidor aplicacional.
Data Inicial de Execução	InitExec	Data inicial de um pedido à base de dados.
Data Final de Execução	EndExec	Data final de um pedido à base de dados.
Categoria	Category	Categoria do pedido. Pode ser Statement, PreparedStatement ou CallableStatement.
SQL	SQL	SQL real passado à base de dados.
PreparedStatement	PreparedStatement	PreparedStatement referente ao SQL real passado à base de dados.
Tamanho do ResultSet	RSetSize	Tamanho do ResultSet de retorno de dados.
Tempo decorrido	ElapsedTime	Tempo decorrido na base de dados. Equivale à data final de execução menos a data inicial de execução.
StackTrace	StackTrace	Caminho do pedido do cliente.

5.2.3 Definição da Estrutura do Ficheiro XML

Antes de iniciar a implementação do registo dos dados no ficheiro XML é essencial definir a estrutura do ficheiro para que não sejam necessárias alterações futuras.

A seguir é apresentada a estrutura do ficheiro XML. A tag **<DBEvent>** representa um pedido à base de dados, ou seja, um registo. Este ficheiro terá tantos “DBEvent” quantos os pedidos à base de dados. A estrutura aqui apresentada foi validada pela empresa.

```

<?xml version="1.0" encoding="UTF-8" ?>
<WiproSpyLog>
  <DBEvent>
    <LogDate> </LogDate>
    <ConnID> </ConnID>
    <InitDate>
      <Timestamp> </Timestamp>
    </InitDate>
    <InitExec>
      <Timestamp> </Timestamp>
    </InitExec>
    <EndExec>
      <Timestamp> </Timestamp>
    </EndExec>
    <Category> </Category>
    <SQL> </SQL>
    <PreparedStatement> </PreparedStatement>
    <RSetSize> </RSetSize>
    <ElapsedTime> </ElapsedTime>
    <StackTrace> </StackTrace>
  </DBEvent>
</WiproSpyLog>

```

5.2.4 Utilização do log4j Para Fazer o Registo dos Dados

Para fazer o registo dos dados no ficheiro XML foi utilizado o Apache log4j. Esta ferramenta, tal como já foi descrito na análise tecnológica é ideal para fazer *logging* de dados. O log4j é constituído por três elementos fundamentais: o *Logger*, o *Appender* e o *Layout*. O *Logger* utilizado neste caso foi instanciado na classe pública “WiproDBLogger“. As propriedades deste *Logger* são definidas no ficheiro de propriedades “log4jDBProperties.xml” apresentado a seguir:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <appender name="appender" class="org.apache.log4j.FileAppender">
    <param name="File" value="Spylog.xml" />
    <param name="Append" value="true" />
    <layout class="com.wipro.utils.XMLLayout" />
  </appender>
  <root>
    <priority value="info" />
    <appender-ref ref="appender" />
  </root>
</log4j:configuration>
```

As propriedades apresentadas em cima definem:

- O *Appender*, que é um “FileAppender“, o que indica que o registo será feito em ficheiro;
- O nome do ficheiro de destino, que é “Spylog.xml”;
- Que o ficheiro é *appendable*, isto é, que pode ser acrescentada informação ao ficheiro, caso este exista. Isto é importante, pois, em pedidos concorrentes o *Logger* é iniciado para cada ligação, logo, se esta opção não fosse verdadeira o ficheiro seria reescrito por cada *Logger*, pelo que no final seria apresentada a informação apenas do último *Logger* a fazer registo nesse ficheiro;
- A classe de *Layout*, que neste caso é “XMLLayout” do *package* “com.wipro.utils”. Este *Layout* foi definido no contexto deste projecto e será apresentado à frente;
- O nível de prioridade para o *Appender* definido, que neste caso é “info”. Isto indica que todos os registos com prioridade igual ou superior a INFO, ou seja, INFO, WARN, ERROR e FATAL serão registados.

De notar que podem ser definidos vários *Appenders* para o mesmo *Logger*. Isto permite, por exemplo, que os dados sejam registados num ficheiro e enviados para um socket ao mesmo tempo.

Em relação ao *Appender*, não foram necessárias quaisquer alterações já que foi utilizado o “FileAppender” do log4j.

O *Layout*, ao contrário do *Appender*, teve que ser definido, já que o *Layout* para XML original do log4j não permite obter o ficheiro desejado. Foi então necessário estender a classe “org.apache.log4j.xml.XMLLayout” e implementar o *Layout* desejado. O método principal desta classe é o método “format” que recebe um parâmetro de entrada, “LoggingEvent” e retorna um “StringBuffer”. O parâmetro de entrada contém os dados de um pedido de *logging*, entre os quais uma “String” com a mensagem de *logging*. Como o log4j está desenhado para fazer o registo de mensagens e não de vários dados, estes dados têm de ser concatenados numa “String” e depois “decifrados” no *Layout*. O “StringBuffer” é concatenado com um `<DBEvent>` e todos os dados inerentes a ele e, depois, retornado para o *Appender* que escreve essa

informação no ficheiro. De notar que o cabeçalho e o rodapé do ficheiro XML são acrescentados no final de todos os registos, para evitar sobrecarga no sistema.

5.2.5 Transacções XA

XA é uma especificação para o processamento de transacções distribuídas para bases de dados. Uma transacção distribuída é um conjunto de duas ou mais transacções relacionadas que devem ser geridas para que ambas sejam executadas correctamente, mesmo que estas sejam feitas para bases de dados diferentes [ORA07]. Esta especificação garante a integridade dos dados, que é uma característica essencial nos desenvolvimentos realizados pela Wipro Retail. A utilização do *driver* JDBC neste tipo de transacções é diferente. Nesta fase foi estudada a forma como são efectuadas estas transacções e garantido que também são registadas pela ferramenta.

5.3 Fase 2: Do Ficheiro XML para a Base de Dados

A fase 2 da ferramenta implica a replicação dos dados registados no ficheiro XML para a base de dados HSQLDB. Neste capítulo estão descritos todos os passos que foram efectuados para atingir este objectivo. A figura 5-2 esquematiza as funcionalidades que devem ser implementadas na fase 2, descrita nesta secção.



Figura 5-2 – Esquema das funcionalidades a implementar na fase 2.

5.3.1 Alterações à Base de Dados

A base de dados HSQLDB original do Elvyx permitia já guardar a maior parte da informação necessária. O modelo desta base de dados é constituído por três tabelas: SQL_PREPARED, SQL_STATEMENT e SQL_DATA. A tabela SQL_PREPARED, tal como o nome indica, guarda os *PreparedStatement* dos pedidos. Como para vários *SQLStatements* podemos ter o mesmo *PreparedStatement*, a tabela SQL_STATEMENT guarda os *SQLStatements* de cada *PreparedStatement*, existindo assim uma ligação 1 para n entre a tabela SQL_PREPARED e a tabela SQL_STATEMENT. A tabela SQL_DATA guarda os dados específicos de cada pedido SQL como o ID de conexão, as datas de início e fim do pedido ou o tempo decorrido para executar o pedido. Entre as tabelas SQL_STATEMENT e SQL_DATA existe também uma ligação 1 para n, pois para o mesmo *Statement* podem ser feitos vários pedidos, e como consequência existem vários dados referentes a cada pedido específico do cliente. A figura 5-3 representa o modelo de dados da base de dados HSQLDB original do ELvyx.

Implementação da Solução

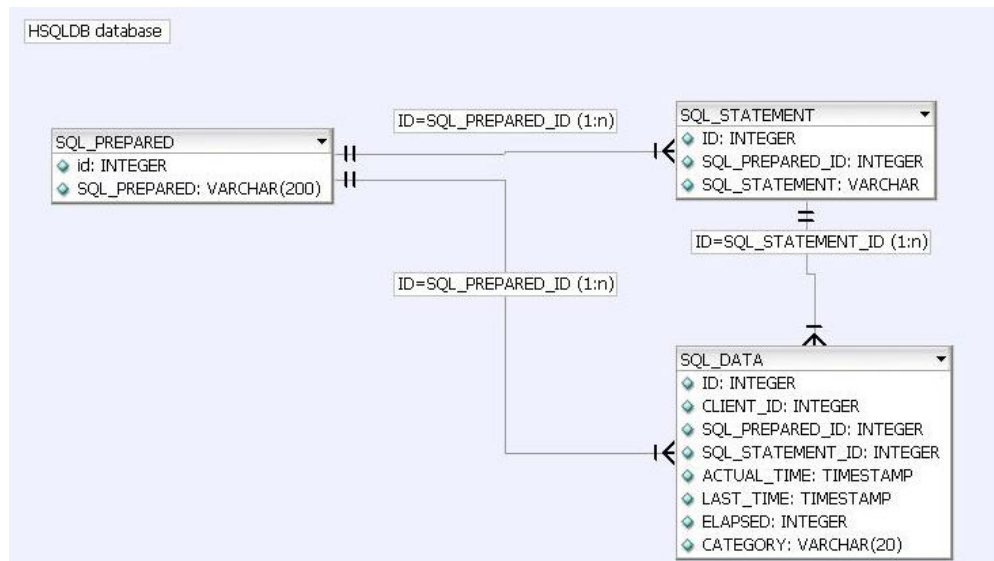


Figura 5-3 – Modelo de dados da base de dados HSQLDB original do Elvyx.

As alterações efectuadas à base de dados foram no sentido de permitir também guardar o caminho do pedido do cliente. Como este caminho é específico de cada pedido, isto é, um mesmo *SQLStatement* pode ter pedidos de caminhos diferentes, o registo do caminho do pedido do cliente faz parte dos dados específicos de um pedido, logo poderia ser integrado na tabela *SQL_DATA*. No entanto, optou-se por criar uma tabela nova para guardar esta informação, pois desta forma evitam-se alterações à ferramenta original. A nova tabela chama-se *NB_SQL_STACKTRACE* e tem uma ligação 1 para 1 com a tabela *SQL_DATA*. A figura 5-4 representa o modelo de dados da base de dados HSQLDB adaptada às necessidades da nova ferramenta.

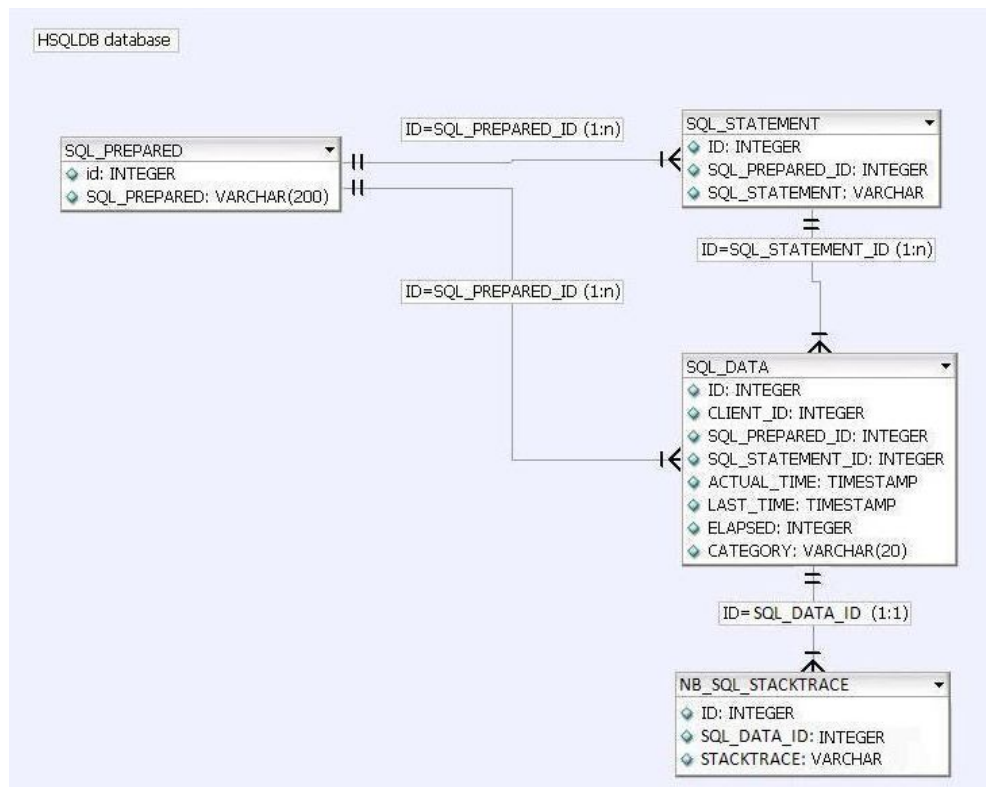


Figura 5-4 – Modelo de dados da base de dados HSQLDB da nova ferramenta.

5.3.2 Commons Digester

A ferramenta Commons Digester foi utilizada para fazer o *parsing* do ficheiro XML pois permite, de forma simples, utilizar SAX para esse mesmo objectivo. Desta forma obtêm-se os dados específicos de cada registo que são depois inseridos na base de dados.

A descrição do Commons Digester já foi feita na análise tecnológica, pelo que aqui será apenas descrito o modo como esta ferramenta foi utilizada.

A ideia base do Commons Digester é indicar, para as *tags* definidas, um método ao qual serão passados parâmetros, também eles definidos por *tags*. Utilizando um pedido à base de dados como exemplo, a forma de especificar o método e os parâmetros que serão passados é a seguinte:

```
digester.addCallMethod("WiproSpyLog/DBEvent", "addData", 10);
digester.addCallParam("WiproSpyLog/DBEvent/ConnID", 0);
digester.addCallParam("WiproSpyLog/DBEvent/InitDate/TimeStamp", 1);
digester.addCallParam("WiproSpyLog/DBEvent/InitExec/TimeStamp", 2);
digester.addCallParam("WiproSpyLog/DBEvent/EndExec/TimeStamp", 3);
digester.addCallParam("WiproSpyLog/DBEvent/Category", 4);
digester.addCallParam("WiproSpyLog/DBEvent/SQL", 5);
digester.addCallParam("WiproSpyLog/DBEvent/PreparedStatement", 6);
digester.addCallParam("WiproSpyLog/DBEvent/RSetSize", 7);
digester.addCallParam("WiproSpyLog/DBEvent/ElapsedTime", 8);
digester.addCallParam("WiproSpyLog/DBEvent/StackTrace", 9);
```

A primeira linha deste trecho de código indica que para cada *tag* `<DBEvent>` deve ser chamado o método “addData” que deve receber dez parâmetros. As linhas seguintes especificam esses parâmetros, que são obtidos de *tags* pertencentes a um `<DBEvent>`. De notar que é necessário especificar todo o caminho da árvore até ao nó desejado.

A linha de código: “`digester.parse(file);`” inicia a execução do *parsing* do ficheiro. O método “addData” é depois responsável por inserir os dados na base de dados.

Como o Commons Digester não necessita de conhecer a totalidade dos dados contidos no ficheiro, mas apenas um nó `<DBEvent>` em cada momento, este ficheiro não é carregado em memória o que permite uma performance bastante melhor, em comparação com as soluções DOM.

5.3.3 Particularidades da Funcionalidade

A implementação desta funcionalidade teve algumas particularidades que são descritas aqui. A primeira está relacionada com o nome do ficheiro XML que será alvo de *parsing*. Como esse nome pode ser alterado no ficheiro de propriedades do log4j, é necessário verificar, nesse ficheiro, qual o nome dado ao ficheiro XML. Esta funcionalidade foi implementada de forma a não ocorrerem problemas devido à alteração do nome do ficheiro XML.

A segunda particularidade está relacionada com a ligação à base de dados HSQLDB. Como foi descrito na análise tecnológica do HSQLDB, este permite a sua utilização em modo embebido ou em modo servidor. As grandes diferenças entre estes dois modos são a liberdade de acesso e a forma de iniciar a base de dados. No modo servidor qualquer aplicação pode aceder à base de dados e esta é carregada apenas uma vez, quando o servidor é iniciado. No modo embebido apenas a aplicação que inicia a base de dados tem acesso a ela e esta é carregada de cada vez que a aplicação é executada. Como é permitida a utilização de ambos os modos, foram implementadas ambas as formas de ligação. Assim, a ferramenta testa se é possível efectuar a ligação através do modo servidor. Caso não seja possível, inicializa o modo embebido, carregando a base de dados em memória.

A terceira particularidade está relacionada com a forma como foi implementada a inserção dos dados na base de dados. Toda a comunicação com a base de dados é feita através de *PreparedStatement*. Os *PreparedStatement* permitem à base de dados utilizar *cache* para

execuções repetidas do mesmo pedido. Desta forma, possibilita uma melhor performance da ferramenta.

A quarta particularidade está relacionada com a eliminação dos dados da base de dados. Esta pode ser feita eliminando os ficheiros da base de dados ou, em alternativa, executando uma outra funcionalidade implementada, que elimina todos os dados das tabelas.

Por último, devido ao tamanho, por vezes bastante elevado, do ficheiro XML, esta funcionalidade pode ser algo demorada em algumas situações. Para não deixar o utilizador sem *feedback* da ferramenta foi implementada uma funcionalidade que apresenta no ecrã a percentagem de conclusão do processo.

5.4 Fase 3: Ferramentas Extensíveis de Análise de Dados

A fase 3 da ferramenta envolve a análise dos dados registados anteriormente. Esta é a fase que cria real diferenciação em relação às outras ferramentas. O objectivo desta fase é, não só, implementar funcionalidades de análise específica de dados mas também criar ferramentas que possam ser estendidas futuramente. Neste capítulo é descrita a forma como foi implementada a base da ferramenta, que possibilita a extensão, os analisadores concretos implementados e a forma como é gerado o relatório final. A figura 5-5 esquematiza as funcionalidades que devem ser implementadas na fase 3, descrita nesta secção.

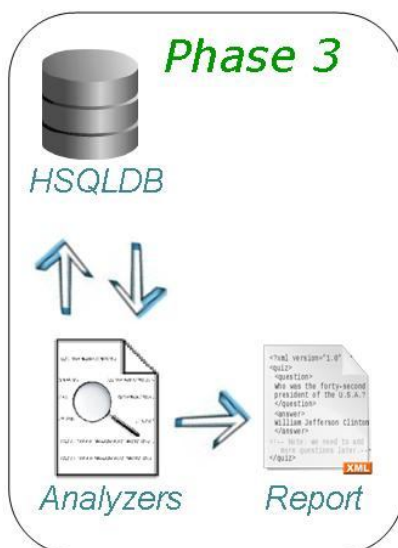


Figura 5-5 – Esquema das funcionalidades a implementar na fase 3.

5.4.1 Ficheiro de Propriedades

Nesta fase foi detectada a necessidade de utilizar um ficheiro global de propriedades que seria útil também para o cliente *web*, implementado futuramente. Este ficheiro, com o nome “wipro.spy.properties”, permite ao utilizador definir parâmetros da ferramenta. Esses parâmetros estão relacionados com as ferramentas extensíveis de análise de dados e com o cliente *web*. Em relação às ferramentas extensíveis de análise de dados, o utilizador pode definir, neste ficheiro, quais os analisadores a serem executados e, para cada um deles, pode definir os parâmetros, caso estes usem parâmetros. Em relação ao cliente *web*, o utilizador pode definir qual a porta a utilizar, o número máximo de pedidos mostrados no *browser* e ainda o estado inicial da ferramenta em relação ao registo de dados, ligado ou desligado. Todos estes parâmetros da ferramenta serão explicados no seu devido contexto nas secções seguintes. Aqui é apenas feita a referência ao ficheiro de propriedades, que ajudará a situar o leitor em alguns

exemplos que se seguem. Um exemplo deste ficheiro encontra-se em anexo, no Anexo C – Exemplo de um Ficheiro de Propriedades.

5.4.2 Classe Executável – Analyze

A classe executável “Analyze” é responsável por iniciar as ferramentas de análise de dados. Esta classe faz uso do Java Reflection API para poder carregar classes dinamicamente. As classes que vão efectivamente analisar os dados serão carregadas por esta classe, em tempo de execução, ou seja, não precisam de ser compiladas, nem sequer definidas aquando da compilação. A classe “Analyze” procura no ficheiro de propriedades da ferramenta, “wipro.spy.properties”, o nome das classes que deve carregar e depois tenta a execução do método “execute()” em cada uma delas. O nome das classes a serem carregadas deve vir à frente de “LoadClass=” no ficheiro de propriedades.

5.4.3 Classe Abstracta – Analyzer

Para definir a estrutura geral das classes efectivas de análise de dados é utilizada uma classe abstracta, “Analyzer”. Esta classe deve ser estendida por todas as classes efectivas de análise de dados. O uso de uma classe abstracta e não de uma interface deve-se ao facto de poderem ser implementados métodos com o uso de uma classe abstracta. A classe “Analyzer” implementa quatro métodos: “getConn()”, que devolve a conexão à base de dados; “getParams()”, que devolve um *array* com os parâmetros da classe derivada; “end()”, que fecha todas as conexões à base de dados e termina o ficheiro de relatório; e ainda o método “log()”, que é utilizado para registar as mensagens no relatório final. Nesta classe existe ainda o método abstracto “execute()”. Este método, sendo abstracto, tem que ser obrigatoriamente implementado pelas classes que estendam a classe abstracta. Desta forma, garante-se que o método “execute()” existirá em todas as classes que serão chamadas pela classe “Analyze” através de Reflection.

De forma a facilitar o desenvolvimento futuro de classes efectivas de análise de dados por parte de pessoas sem o conhecimento do código fonte desta ferramenta, foi desenvolvido um modelo para construção destas classes. Este tem já implementado a forma de obter a conexão à base de dados, a forma de obter os parâmetros da classe, de registar as mensagens no relatório final e de terminar todas as conexões. O modelo encontra-se definido no Anexo A – Modelo das Classes “Analyzers”.

5.4.4 Classes Efectivas de Análise de Dados – Analyzers

Como descrito anteriormente, as classes efectivas de análise de dados, “Analyzers” estendem a classe abstracta “Analyzer”, e é nestas que são efectivamente definidos os métodos de análise de dados. Nesta secção da fase 3 de implementação são descritos os “Analyzers” que foram implementados. Para cada um, é feita uma descrição de alto nível de modo a facilitar a compreensão da implementação feita.

ConsecutiveSQLsyAnalyzer

O objectivo deste “Analyzer” é encontrar pedidos à base de dados consecutivos. Estes pedidos não precisam de ser do mesmo cliente, isto é, não precisam de ter o mesmo caminho de pedido mas apenas ter o mesmo SQL e serem feitos pedidos consecutivos. Isto permite detectar, por exemplo, trechos de código que efectuam o mesmo pedido várias vezes seguidas em partes

diferentes do código. Isto pode significar erros de programação, pois provavelmente o primeiro pedido seria suficiente para todo o código consequente. A repetição dos pedidos iria gerar sobrecarga na base de dados, que, através da análise feita por este “Analyzer”, pode assim ser evitada.

ConsecutiveCallsAnalyzer

Este “Analyzer” detecta pedidos iguais dentro de um espaço de tempo máximo definido pelo utilizador. Os pedidos têm que ter o mesmo SQL e o mesmo *Stacktrace*, isto é, têm que ser feitos exactamente do mesmo local no código. Isto permite, por exemplo, detectar pedidos feitos dentro de ciclos, ou seja, pedidos executados várias vezes quando provavelmente poderiam ser feitos apenas uma vez. Tomando como exemplo uma tabela de empregados que tem o ID e o nome do empregado, um ciclo em que para cada ID de empregado é feito um pedido à base de dados para retornar o nome do empregado vai gerar sobrecarga no sistema. É preferível fazer um único pedido que retorne todos os ID e todos os nomes dos empregados e depois fazer as verificações no código. O espaço de tempo máximo definido pelo utilizador permite escalonar a análise, ou seja, detectar os pedidos iguais em que a cadência de execução é menor que o valor especificado, para evitar, por exemplo, detectar pedidos iguais que significam múltiplos pedidos de clientes, e que consequentemente não são problemáticos, ou, pelo menos, não resolvíveis através de melhorias de código.

ConsecutiveCallFromTablesAnalyzer

Este “Analyzer” segue a mesma lógica que o anterior. O objectivo é o mesmo, no entanto, este permite ao utilizador definir uma lista de tabelas chave dos pedidos. As tabelas chave são tabelas incluídas nos pedidos SQL, ou seja, para um pedido ser detectado, além das restrições já referidas no “Analyzer” anterior, na cadeia de caracteres que constitui o pedido SQL deve existir pelo menos uma das tabelas definidas pelo utilizador. Isto permite que sejam feitos relatórios apenas sobre tabelas fulcrais da aplicação monitorizada.

ExecutionTimesAnalyzer

O objectivo deste “Analyzer” é detectar o tempo total de execução e o tempo total dispendido na base dados, ou seja, o total tempo que a base de dados gastou a processar todos os pedidos registados. São apresentados os dois tempos, assim como a percentagem de tempo dispendido na base de dados. Esta análise dá uma percepção de onde a maioria do trabalho é efectuado, se no servidor aplicacional, se na base de dados. As conclusões que se podem tirar desta análise variam conforme o contexto da aplicação mas se, por exemplo, a percentagem do tempo de execução na base de dados for superior a 10%, quando não são executados procedimentos ou funções na base de dados, pode-se afirmar que são efectuados pedidos a mais à base de dados. Este “Analyzer” pode servir como uma primeira referência para detectar o bom ou mau funcionamento da aplicação monitorizada.

ExecutionTimesOnTablesAnalyzer

Este “Analyzer” detecta o tempo total dispendido na base de dados e o tempo total dispendido em tabelas específicas. O utilizador pode definir uma lista de tabelas chave. O “Analyzer” calcula o somatório do tempo dispendido na base de dados de todos os pedidos SQL cuja cadeia de caracteres contenha pelo menos uma das tabelas chave. No relatório são apresentados os dois tempos assim como uma percentagem do tempo dispendido nas tabelas

chave em relação ao tempo total dispendido na base de dados. Este “Analyzer” permite detectar se tabelas chave estão a ser utilizadas de forma excessiva. Isto pode indicar que é necessário fazer optimizações à base de dados, por exemplo em indexações.

PatternsAnalyzer

O objectivo deste “Analyzer” é detectar padrões de pedidos, isto é, detectar pedidos que ocorrem sistematicamente de x em x tempo. O utilizador define o número mínimo de pedidos que constituem um padrão, a cadência de ocorrência dos pedidos e o desvio permitido a essa cadência. O “Analyzer” encontra os pedidos que tenham o mesmo SQL e o mesmo *Stacktrace*, ou seja, pedidos efectuados exactamente do mesmo local, e que ocorram com diferença de x tempo do pedido anterior, em que x pertence ao intervalo de tempo definido pelo tempo médio de ocorrência dos pedidos (cadência) menos o desvio e o tempo médio de ocorrência dos pedidos mais o desvio. A detecção destes padrões permite compreender se há funcionalidades que são executadas de forma sistemática e que podem gerar problemas de performance.

Este “Analyzer” permite ainda que seja passado um quarto parâmetro, opcional, que define um desvio máximo do tempo médio de execução dos pedidos na base de dados. Caso seja passado este parâmetro, é calculado o tempo médio de execução dos pedidos dentro de cada padrão detectado e, para cada um dos pedidos, se o tempo de execução for maior que o tempo de execução médio mais o desvio definido ou menor que o tempo de execução médio menos o desvio definido, é apresentada uma mensagem de aviso no relatório final, com o valor do desvio real desse pedido em relação à média. Esta opção permite detectar execuções excepcionais, como por exemplo, execuções em momentos de sobrecarga do sistema.

5.4.5 Geração do Relatório

Para gerar o relatório final é utilizado mais uma vez o Apache log4j, novamente com o objectivo de gerar um ficheiro XML. A opção de utilizar XML para o relatório final deve-se à facilidade que o XML apresenta na integração com outras ferramentas, algo que pode ser necessário no futuro.

Como existem algumas diferenças nesta fase em relação à fase 1 foi necessário especificar uma nova estrutura do log4j. Os três elementos fundamentais do log4j: o *Logger*, o *Appender* e o *Layout*, foram novamente definidos para as necessidades específicas desta fase. O *Logger* utilizado neste caso foi instanciado na classe pública “WiproReportLogger“. As propriedades deste *Logger* são definidas no ficheiro de propriedades “log4jReportProperties.xml” apresentado a seguir:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <appender name="appender" class="org.apache.log4j.FileAppender">
    <param name="File" value="Report.xml" />
    <param name="Append" value="false" />
    <layout class="com.wipro.utils.XMLReportLayout" />
  </appender>
  <root>
    <priority value="info" />
    <appender-ref ref="appender" />
  </root>
</log4j:configuration>
```

Implementação da Solução

As propriedades apresentadas em cima definem:

- O *Appender*, que é um “FileAppender”, o que indica que o registo será feito em ficheiro;
- O nome do ficheiro de destino, que é “Report.xml”;
- Que o ficheiro não é *appendable*, isto é, não pode ser acrescentada informação ao ficheiro caso este exista. De cada vez que é iniciado o log4j, ou seja, de cada vez que é executada a classe “Analyze”, se o ficheiro “Report.xml” existir toda a informação contida nele é eliminada e é gerada nova informação;
- A classe de *Layout*, que neste caso é “XMLReportLayout” do *package* “com.wipro.utils”. Este *Layout* foi definido no contexto desta fase e será apresentado à frente;
- O nível de prioridade para o *Appender* definido, que neste caso é “info”. Isto indica que todos os registos com prioridade igual ou superior a INFO, ou seja, INFO, WARN, ERROR e FATAL serão registados.

Em relação ao *Appender*, não foram necessárias quaisquer alterações já que foi utilizado o “FileAppender” do log4j.

Também nesta fase foi necessário definir um *Layout* específico, já que o *Layout* para XML original do log4j não permite obter o ficheiro desejado. Foi então necessário estender a classe “org.apache.log4j.xml.XMLLayout” e implementar o *Layout* desejado. A implementação deste *Layout* apresentou algumas particularidades, já que, ao contrário do que acontecia na fase 1, nesta fase não há uma estrutura definida do ficheiro XML. Existe uma estrutura base mas é necessário permitir alguma flexibilidade, pois para novos “Analyzers” pode ser necessário gerar relatórios diferentes. A estrutura base do relatório apresenta, para cada evento (correspondente ao registo de uma mensagem por parte de um “Analyzer”): a data em que foi feito o registo, o nível do registo (DEBUG, INFO, WARN, ERROR ou FATAL), o nome do “Analyzer” que fez o registo e uma mensagem que é definida no “Analyzer”. Esta estrutura é apresentada a seguir:

```
<Event>
  <LogDate> </LogDate>
  <LogLevel> </LogLevel>
  <Analyzer> </Analyzer>
  <Message> </Message>
</Event>
```

A flexibilidade desejada foi obtida dando a possibilidade de adicionar *tags* dentro de cada `<Event>`. Desta forma é possível gerar relatórios configuráveis por quem desenvolve novos “Analyzers”, sem que seja necessário ter conhecimento sobre a forma como está definido o *Layout*. Isto é possível devido à forma como foram implementadas as funções de “log()” na classe abstracta “Analyzer”. Foram implementadas duas funções de “log()”, utilizando *overloading*²⁶ de métodos. Uma das funções, a mais básica, recebe dois parâmetros: o nível do registo e a mensagem a registar; e gera um evento com as *tags* apresentadas em cima. A outra função, a que permite personalizar o relatório, recebe quatro parâmetros: o nível do registo, a mensagem a registar, um *array* de *tags* e um *array* com as mensagens para cada *tag* do *array* de *tags*. Assim, pode ser adicionado o número de *tags* desejado por quem desenvolve novos “Analyzers”. A única coisa que é necessário saber é a existência das duas funções de “log()” e a forma como devem ser enviados os parâmetros. Com esta funcionalidade é possível gerar relatórios como o apresentado a seguir, em exemplo.

²⁶ Permite a existência de vários métodos com o mesmo nome mas com assinaturas diferentes, ou seja com número de parâmetros diferentes.

```

<Event>
  <LogDate>Mon Jun 15 14:32:51 BST 2009</LogDate>
  <LogLevel>WARN</LogLevel>
  <Analyzer>com.wipro.analyzers.PatternsAnalyzer</Analyzer>
  <Message>A pattern with 6 calls has been detected on statement
    "SELECT * FROM product order by id" with stacktrace
    Test.Test4Analyzers.executeQ(Test4Analyzers.java:107)
    ->Test.Test4Analyzers.main(Test4Analyzers.java:93).
  </Message>
  <MinimumNumberOfSQLs>5</MinimumNumberOfSQLs>
  <AverageTime>3.0 sec</AverageTime>
  <StandardDeviation>0.5 secs</StandardDeviation>
  <ExecutionTimesDeviation>100.0 msec</ExecutionTimesDeviation>
  <ExecutionTimesAverage>208 msec</ExecutionTimesAverage>
</Event>

```

5.5 Melhoria da Fase 1: Cliente Web

Esta secção apresenta a descrição da implementação do cliente *web*. Esta funcionalidade, apesar de pertencente à fase 1 da ferramenta foi apenas implementada após a conclusão da fase 3, pois não era parte crítica do projecto. Surge então como melhoria da fase 1.

Com a implementação do cliente *web* acrescenta-se assim o elemento em falta na fase 1 da ferramenta. A figura 5-6 esquematiza a fase 1 após a implementação do cliente *web*, descrita nesta secção.

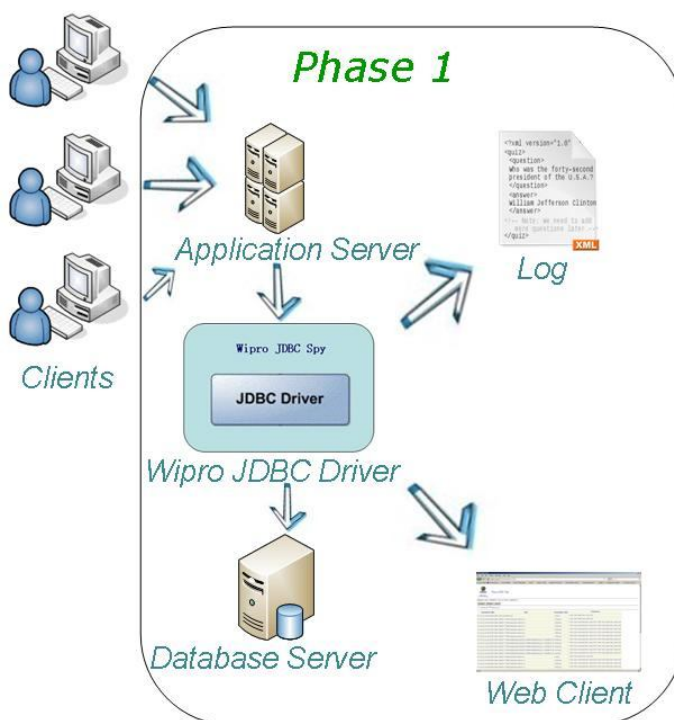


Figura 5-6 – Esquema de todas as funcionalidades da fase 1.

A ideia principal do cliente *web* é dar a possibilidade ao utilizador da ferramenta de ter acesso rápido a informação sobre o estado da monitorização. A opção de utilizar um cliente *web* e não um cliente local traz várias vantagens, como por exemplo, a simplicidade de uso, já que é utilizado um *browser*, algo a que a maioria das pessoas estão familiarizadas, a liberdade de uso, pois permite que seja utilizada em qualquer computador e não necessita de qualquer instalação

Implementação da Solução

ou a distribuição, já que permite que seja utilizado por vários utilizadores em simultâneo. O facto de ser *web* facilita ainda uma futura expansão e integração com outras ferramentas.

Como foi referido na proposta de solução, uma das alterações anteriormente implementadas pela Wipro Retail no P6Spy tinha sido exactamente a implementação de um cliente *web*. A implementação feita neste projecto é baseada nessa mesma implementação existente no P6Spy versão Wipro Retail. Para criar o cliente *web* é necessário implementar um servidor que responda aos pedidos do cliente. A implementação desse servidor, na ferramenta P6Spy foi baseada numa aplicação *open source*, o NanoHTTPD, analisada na análise tecnológica. Esta aplicação prima pela sua simplicidade e boa funcionalidade, características ideais no contexto deste projecto.

A implementação do NanoHTTPD no P6Spy obrigou a que a classe principal da aplicação fosse estendida para que fosse possível ao servidor responder aos pedidos específicos do cliente pretendido para esta ferramenta. Foi ainda desenvolvida em JavaScript a página apresentada ao utilizador. A implementação do NanoHTTPD neste projecto seguiu os traços gerais da implementação no P6Spy. Muitas das etapas foram replicadas para este caso, sendo que algumas alterações foram necessárias, devido, principalmente, ao modo distinto como as duas ferramentas guardavam os pedidos feitos à base de dados. Foram também efectuadas algumas alterações à página apresentada ao utilizador.

As principais características do cliente *web* desenvolvido para esta ferramenta são:

- Possibilita a apresentação em tempo real dos pedidos efectuados;
- Permite ver o número total de pedidos interceptados;
- Permite ver a quantidade de memória gasta;
- Permite limpar os pedidos apresentados no ecrã;
- Permite ligar/desligar o registo de pedidos.

Algumas propriedades do cliente *web* são definidas no ficheiro de propriedades da ferramenta, “wipro.spy.properties”. Essas propriedades são a porta de acesso ao cliente, que deverá ser usada no *browser*, o estado inicial da ferramenta, ligado/desligado, relativo ao sistema de *logging* e o número máximo de registos a serem apresentados no *browser*.

A figura 5-7 mostra um exemplo de utilização do cliente *web*:

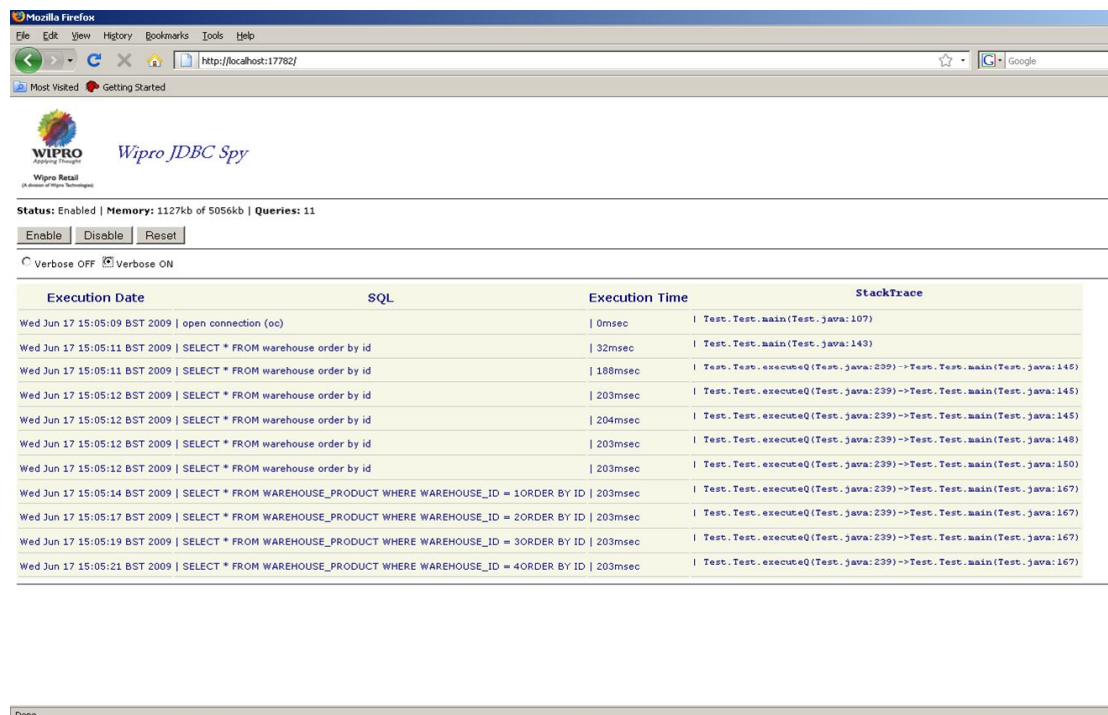


Figura 5-7 – Exemplo de utilização do cliente *web*.

Implementação da Solução

Como se pode ver na figura 5-7, o cliente *web* é constituído por duas grandes partes: um menu, em cima, onde podem ser efectuadas as opções permitidas e a parte central onde são mostrados os pedidos interceptados.

A figura 5-8 mostra o menu com maior detalhe. Podem-se ver todas as opções permitidas ao utilizador: ligar/desligar o registo de pedidos, limpar os pedidos apresentados no ecrã e activar/desactivar a apresentação dos pedidos no ecrã. É também possível ver o estado da aplicação, a memória gasta e número de pedidos registados. Este menu é sempre apresentado ao utilizador enquanto a área onde são apresentados os pedidos só é mostrada caso seja seleccionado “Verbose On”.

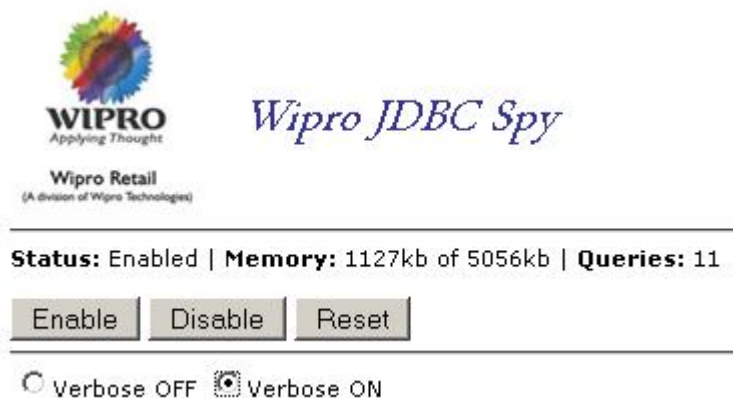


Figura 5-8 – Menu do cliente *web* do Wipro JDBC Spy.

Os dados apresentados na parte central, relativos aos pedidos apresentados no ecrã são: a data em que o pedido foi efectuado, o SQL do pedido, o tempo de execução na base de dados e o *stacktrace* do pedido. A figura 5-9 mostra essa área do cliente, num exemplo de uma monitorização, onde se podem ver com mais detalhe os dados apresentados.

Execution Date	SQL	Execution Time	StackTrace
Wed Jun 17 15:13:22 BST 2009	SELECT * FROM warehouse order by id	203msec	Test.Test.executeQ(Test.java:239)-->Test.Test.main(Test.java:145)
Wed Jun 17 15:13:22 BST 2009	SELECT * FROM warehouse order by id	204msec	Test.Test.executeQ(Test.java:239)-->Test.Test.main(Test.java:145)

Figura 5-9 – Dados apresentados no cliente relativos aos pedidos interceptados.

5.6 Conclusões

Este capítulo apresenta todas as etapas da implementação em detalhe. A proposta de solução seguida foi a utilização da ferramenta Elvyx como base do projecto. A implementação foi dividida em três fases, correspondentes às três fases de utilização da ferramenta. A única excepção foi a implementação do cliente *web* que foi feita apenas após a conclusão da implementação da terceira fase da ferramenta.

A implementação da fase 1 foi concluída com sucesso. No final desta fase a ferramenta possibilitava o registo em ficheiro XML de todos os pedidos dos clientes, incluindo o caminho do pedido.

A implementação da fase 2 foi também concluída com sucesso sendo que passou a ser possível replicar os dados do ficheiro XML para a base de dados HSQLDB.

Com a conclusão da implementação da fase 3 o caminho crítico de utilização da ferramenta ficou completo. Foi definida uma base para as ferramentas de análise de dados que pode facilmente ser estendida e passou também a ser possível fazer análises específicas aos dados registados na fase 1.

Implementação da Solução

A implementação do cliente *web* permite ao utilizador ter acesso rápido a informação relativa ao estado da monitorização e ainda efectuar algumas acções sobre a ferramenta, como ligar/desligar o registo de pedidos.

Conclui-se que a ferramenta atingiu o estado previsto na proposta de solução e encontra-se agora em estado aceitável e deverá ser alvo de testes. A figura 5-10 esquematiza o estado da ferramenta no final da implementação. Este esquema corresponde ao proposto na especificação do Wipro JDBC Spy.

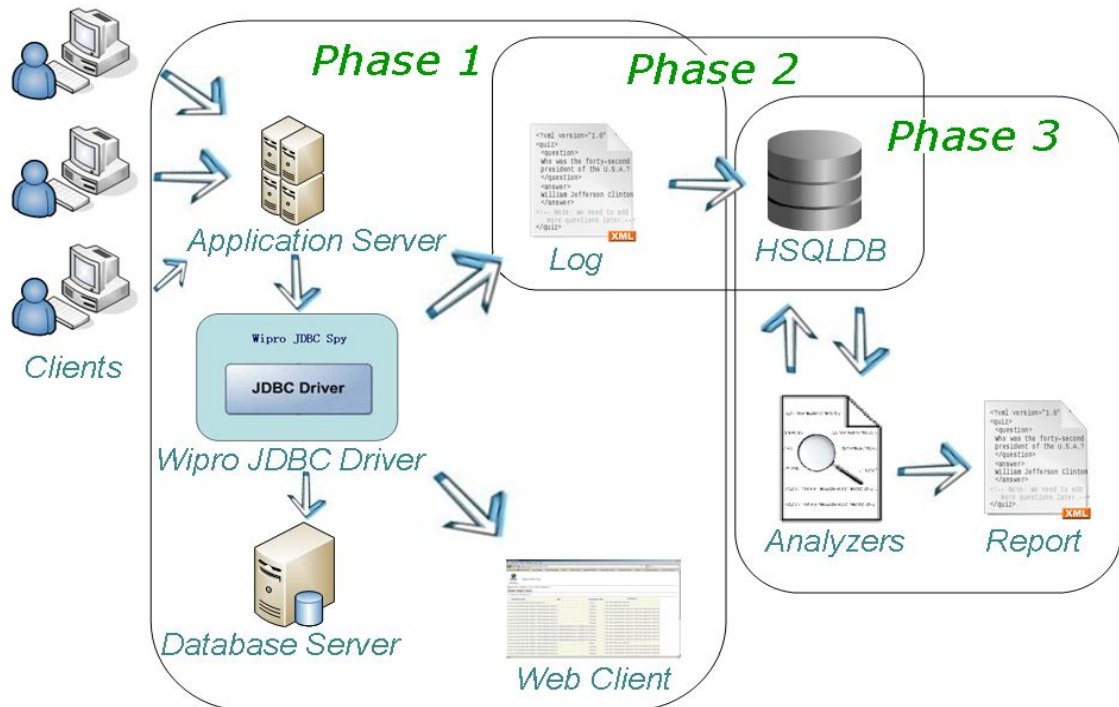


Figura 5-10 – Esquema da ferramenta obtida e divisão em 3 fases

6 Resultados Experimentais

Após a implementação é importante validar a confiança na ferramenta produzida. Com esse objectivo foram desenvolvidos testes que são apresentados neste capítulo. O teste apresentado na primeira secção deste capítulo apresenta um teste demonstrativo da execução normal da ferramenta. Os testes apresentados nas três secções seguintes formam um teste global da ferramenta, dividido pelas três fases constituintes da mesma. Estes testes visam confirmar a integridade total da ferramenta. Por último, a quinta secção, apresenta testes comparativos da performance obtida com diferentes *drivers*. Para a realização destes testes foi desenvolvida uma pequena base de dados que executa no servidor do Oracle Database 10g Express Edition [ODB10G].

6.1 Teste Demonstrativo de Execução

O teste demonstrativo de execução tem como objectivo demonstrar a execução normal da ferramenta nas suas três fases envolventes. Aqui não se tenta testar uma funcionalidade em particular mas a ferramenta como um todo.

6.1.1 Teste Realizado

A realização deste teste envolveu o desenvolvimento de uma classe de teste, que simula pedidos de clientes. Estes pedidos foram criados de forma a serem depois detectados pelas ferramentas de análise de dados. O código desta classe encontra-se em anexo, no Anexo B – Classe de Teste: Test4Analyzers. Espera-se que, ao executar esta classe, seja gerado o ficheiro XML de registo dos pedidos. Depois, deve ser utilizada a funcionalidade da fase 2 para replicar os dados para a base de dados HSQLDB. Por fim, executando a classe “Analyze” com todos os “Analyzers” definidos no ficheiro de propriedades, deve ser gerado o relatório final. O ficheiro de propriedades utilizado para este teste encontra-se em anexo, no Anexo C – Exemplo de um Ficheiro de Propriedades.

Para testar o “ConsecutiveCallsAnalyzer” foram feitos quatro pedidos dentro de um ciclo de meio em meio segundo e depois mais três pedidos dentro de um ciclo de dois em dois segundos. Definindo o parâmetro deste “Analyzer”, espaço de tempo máximo, em um segundo, os primeiros quatro pedidos devem ser detectados por este “Analyzer”, enquanto os três

seguintes não devem ser detectados. Estes sete pedidos devem ser todos detectados pelo “ConsecutiveSQLsAnalyzer”.

Para testar o “ConsecutiveCallsFromTablesAnalyzer” foram feitos quatro pedidos dentro de um ciclo de meio em meio segundo a uma tabela que consta na lista de tabelas e depois mais três pedidos dentro de um ciclo de meio em meio segundo a uma tabela que não consta na lista de tabelas do “Analyzer”. O parâmetro espaço de tempo máximo foi novamente definido em um segundo. Os primeiros quatro pedidos devem ser detectados, enquanto os três seguintes não devem ser detectados. Estes sete pedidos devem ser detectados pelo “ConsecutiveCallsAnalyzer” e pelo “ConsecutiveSQLsAnalyzer”.

Para testar o “PatternsAnalyzer” foi criado um ciclo que faz seis pedidos à base de dados, todos com cadência de 3,1 segundos. Espera-se que os tempos de execução dos pedidos na base de dados sejam próximos dos duzentos milissegundos. Os parâmetros definidos para este “Analyzer” são:

- Número mínimo de SQLs: 5;
- Tempo de cadência dos pedidos: 3 segundos;
- Desvio permitido ao tempo de cadência: 0.5 segundos;
- Desvio dos tempos de execução: 10 milissegundos.

Deve ser detectado o padrão com seis pedidos e, caso o tempo de execução de algum pedido na base de dados seja diferente do tempo médio de execução em mais de dez segundos deve ser apresentada essa diferença. Estes seis pedidos deverão também ser detectados pelo “ConsecutiveSQLsAnalyzer”.

Em relação ao “ExecutionTimesAnalyzer” e ao “ExecutionTimesOnTablesAnalyzer” os valores devem ser calculados manualmente acedendo aos dados da base de dados e comparados com os dados calculados pelos “Analyzers”.

6.1.2 Resultados Obtidos

Com a execução da classe de teste foi gerado o ficheiro XML de registo de pedidos. Este ficheiro encontra-se em anexo, no Anexo D – Exemplo de um ficheiro XML de Registo de Pedidos. Os dados deste ficheiro foram depois replicados para a base de dados HSQldb utilizando a funcionalidade da fase 2 da ferramenta. A execução do “Analyze” com o ficheiro de propriedades que se encontra no Anexo C – Exemplo de um Ficheiro de Propriedades gerou o relatório final que se encontra em anexo, no Anexo E – Exemplo de um Relatório Final. Os dados dos relatórios do “ExecutionTimesAnalyzer” e do “ExecutionTimesOnTablesAnalyzer” foram calculados manualmente e comparados com os valores registados pelos “Analyzers”. Verificou-se que para todos os “Analyzers” foram obtidos os resultados esperados.

6.2 Testes – Fase 1

Os testes da fase 1 têm como principais objectivos garantir a integridade e a totalidade dos dados registados mesmo com pedidos de múltiplos clientes. A seguir são apresentados os testes realizados e os resultados obtidos.

6.2.1 Testes Realizados

Foram desenvolvidos vários programas que simulam pedidos de vários clientes à base de dados desenvolvida. O número total de pedidos é conhecido. Com a execução simultânea dos vários clientes, a ferramenta deve criar o ficheiro de registo com o número total de pedidos e a informação deve ser íntegra. Foram executados três testes com três simuladores de clientes distintos. Em cada teste são executados os três simuladores em simultâneo, sendo que cada um

Resultados Experimentais

deles pode ser executado várias vezes no mesmo teste. Os simuladores 1, 2 e 3 executam 10, 15 e 20 pedidos respectivamente. A tabela 6-1 apresenta os três testes realizados.

Tabela 6-1 – Testes Fase 1.

Teste	Simulador De Cliente	Número De Execuções	Total de pedidos
Teste 1	Simulador 1	1	45
	Simulador 2	1	
	Simulador 3	1	
Teste 2	Simulador 1	2	85
	Simulador 2	3	
	Simulador 3	1	
Teste 3	Simulador 1	1	80
	Simulador 2	2	
	Simulador 3	2	

6.2.2 Resultados Obtidos

Para os três testes foram registados todos os pedidos. Foi também verificada a integridade do ficheiro XML, para garantir que não havia sobreposição de mensagens.

6.3 Testes – Fase 2

Os testes da fase 2 têm como objectivos garantir que apenas dados correctos são inseridos na base de dados, ou seja, que a integridade dos pedidos é mantida e ainda verificar o tempo que demora a executar a funcionalidade, em casos diferentes.

6.3.1 Testes Realizados

Os testes que visam garantir a integridade dos dados consistem em verificar que sempre que um pedido é registado têm que ser inseridos dados em todas as tabelas, excepto se estes forem repetidos ou, no caso das tabelas SQL_PREPARED e SQL_STATEMENT se já existir o mesmo *PreparedStatement* e o mesmo *Statement* respectivamente. Para realização destes testes foram alterados manualmente ficheiros de registo, criando dados propositadamente errados.

Em relação aos testes de tempos de execução, foram gerados cinco ficheiros de registo, cada um com o dobro dos dados do anterior. Assim, esses ficheiros apresentavam 100, 200, 400, 800 e 1600 registos. A replicação dos dados do ficheiro XML para a base de dados HSQldb foi testada cinco vezes para cada um destes ficheiros, quer no modo em que a base é embebida quer no modo em servidor. Após inserir os dados de cada ficheiro cinco vezes, a base de dados é limpa, ou seja, todos os dados nela contidos são eliminados, para se iniciar os testes ao ficheiro seguinte. Isto permite testar o comportamento da base de dados com persistência dos dados.

6.3.2 Resultados Obtidos

Em relação aos testes de integridade dos pedidos, foi verificado que nenhum pedido errado foi inserido na base de dados.

Resultados Experimentais

Os tempos obtidos nos testes de tempos de execução, com a base de dados a ser carregada em modo embebido são apresentados na tabela 6-2. A figura 6-1 apresenta os mesmos dados graficamente. Os tempos estão representados em milissegundos.

Tabela 6-2 – Testes – Fase 2 – Base de dados em modo embebido.

	Número de Pedidos				
	200	400	800	1600	3200
Execução 1	2315	2393	3049	4535	9509
Execução 2	2253	2628	3581	6429	16610
Execução 3	2314	2628	4004	8306	24744
Execução 4	2346	2706	4770	10293	33127
Execução 5	2487	2737	4942	12232	40351
Total	11715	3002	20346	41795	124341
Tempo Médio	2343	2693,2	4069,2	8359	24868,2

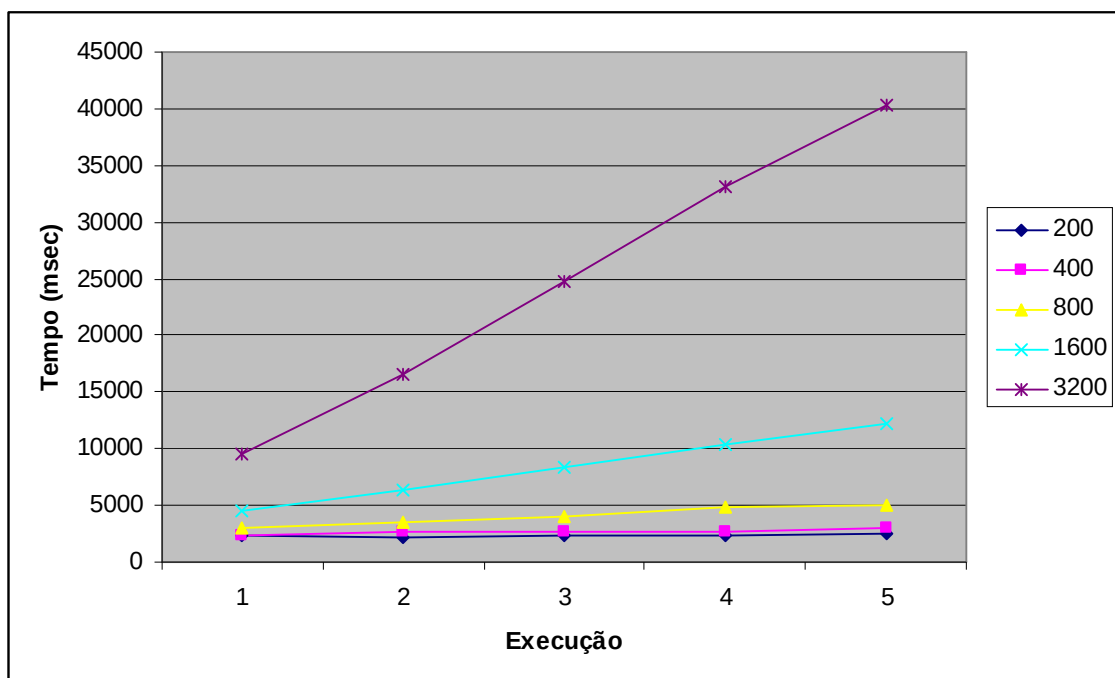


Figura 6-1 – Testes – Fase 2 – Base de dados em modo embebido.

Os tempos obtidos nos testes de tempos de execução, com a base de dados a ser carregada em modo servidor são apresentados na tabela 6-3. A figura 6-2 apresenta os mesmos dados graficamente. Os tempos estão representados em milissegundos.

Resultados Experimentais

Tabela 6-3 – Testes – Fase 2 – Base de dados em modo servidor.

	Número de Pedidos				
	200	400	800	1600	3200
Execução 1	1126	1501	2533	4864	11876
Execução 2	1064	1564	2709	6554	13953
Execução 3	1094	1729	2909	7980	16737
Execução 4	1204	1855	3463	9634	19423
Execução 5	1311	2161	3989	10876	23653
Total	5799	8810	15603	39908	85642
Tempo Médio	1159,8	1762	3120,6	7981,6	17128,4

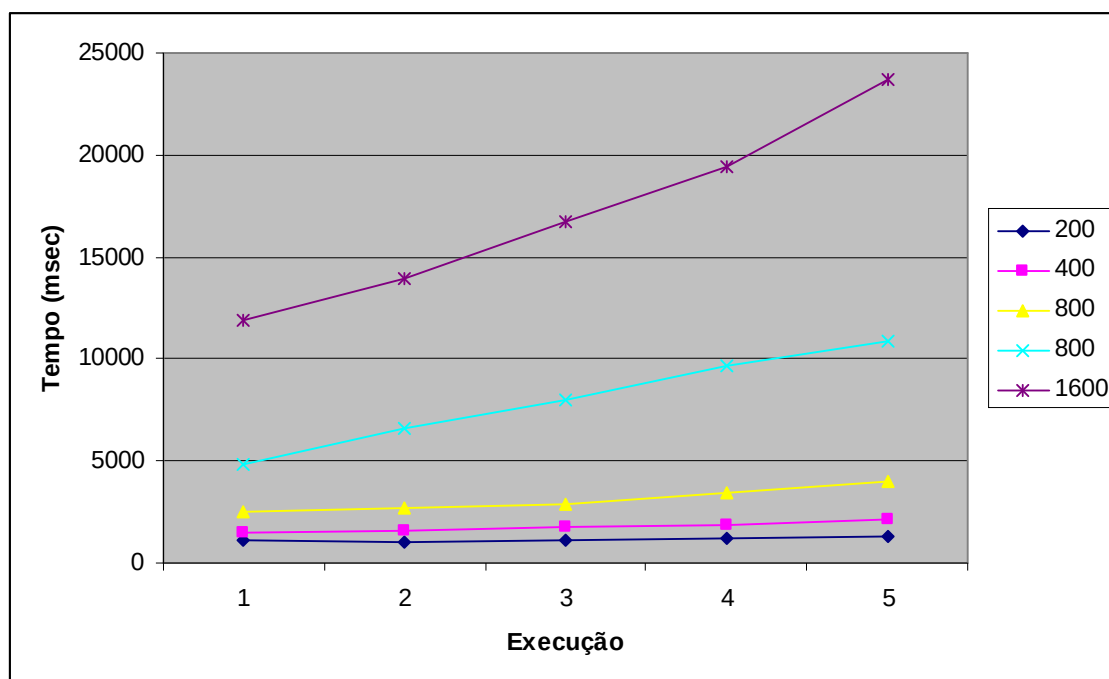


Figura 6-2 – Testes – Fase 2 – Base de dados em modo servidor.

Comparando os tempos de execução para testes com diferentes números de pedidos, verifica-se que o tempo em execução vai aumentando de forma quase exponencial com o aumento linear do número de registos.

Analisando os tempos obtidos em execuções consecutivas para o mesmo número de registos, pode-se afirmar que, quantos mais dados existirem na base de dados, maior será o tempo de execução, ainda que a variância seja pequena, principalmente para um número reduzido de registos.

Comparando os valores obtidos nos modos embebido e servidor verifica-se que o carregamento da base de dados que ocorre no modo embebido é causador de grande parte do tempo de execução, principalmente quando o volume de dados existente na base de dados aumenta.

6.4 Testes – Fase 3

Os testes da fase 3 têm como objectivos garantir que as ferramentas de análise de dados fazem uma análise correcta. Foram também efectuados testes para garantir que nenhum erro causa o aborto da aplicação.

6.4.1 Testes Realizados

Os testes que visam garantir a análise correcta dos dados foram efectuados com recursos a casos definidos. Foram definidas situações e dados concretos para cada um dos “Analyzers” implementados. O objectivo foi verificar se todos os casos eram detectados. O teste demonstrativo de execução apresenta algumas das situações definidas para testar os “Analyzers”. Os outros testes realizados não são apresentados por definirem situações idênticas.

Os testes de verificação de erros visam garantir três situações: tentar executar um “Analyzer” que não existe, passar um número de parâmetros de um “Analyzer” diferente do esperado e passar um valor num parâmetro diferente do esperado, por exemplo passar um conjunto de caracteres quando se espera um número.

6.4.2 Resultados Obtidos

Verificou-se que a análise dos dados nos casos testados é correcta para todos os “Analyzers” implementados.

Em relação aos testes de verificação de erros, a execução do “Analyzer” em questão é abortada mas todos os outros “Analyzers” são executados correctamente.

6.5 Comparação de Performance Utilizando Drivers Diferentes

O objectivo deste teste é verificar se o uso do *driver* JDBC especificado nesta ferramenta provoca grandes atrasos de resposta em relação ao *driver* normal. O outro *driver* JDBC utilizado nestes testes é o *driver* especificado pela Oracle, e ideal para executar pedidos às bases de dados Oracle.

6.5.1 Testes Realizados

Foram realizados dois tipos de testes que pretendem analisar duas situações distintas. O primeiro teste consiste em executar pedidos consecutivos, sem pausas entre cada pedido, e pretende analisar situações de sobrecarga no servidor. Para estes testes foram feitos 5000, 10000, 20000 e 40000 pedidos consecutivos com ambos os *drivers*.

O segundo teste consiste em fazer pedidos de meio em meio segundo e pretende analisar situações de execução normal do servidor. Para estes testes foram feitos 5, 10, 20 e 40 pedidos, com intervalo de meio segundo entre cada pedido, com ambos os *drivers*.

6.5.2 Resultados Obtidos

Os tempos obtidos nos testes realizados com pedidos consecutivos são apresentados na tabela 6-4. A figura 6-3 apresenta os mesmos dados graficamente. Os tempos estão representados em milissegundos e são tempos médios de resposta.

Resultados Experimentais

Tabela 6-4 – Análise de performance do *driver* JDBC – Sobrecarga.

	Número de Pedidos			
	5000	10000	20000	40000
Oracle Driver	2643	5348	10306	22740
Wipro JDBC Spy Driver	4567	8884	15922	34484

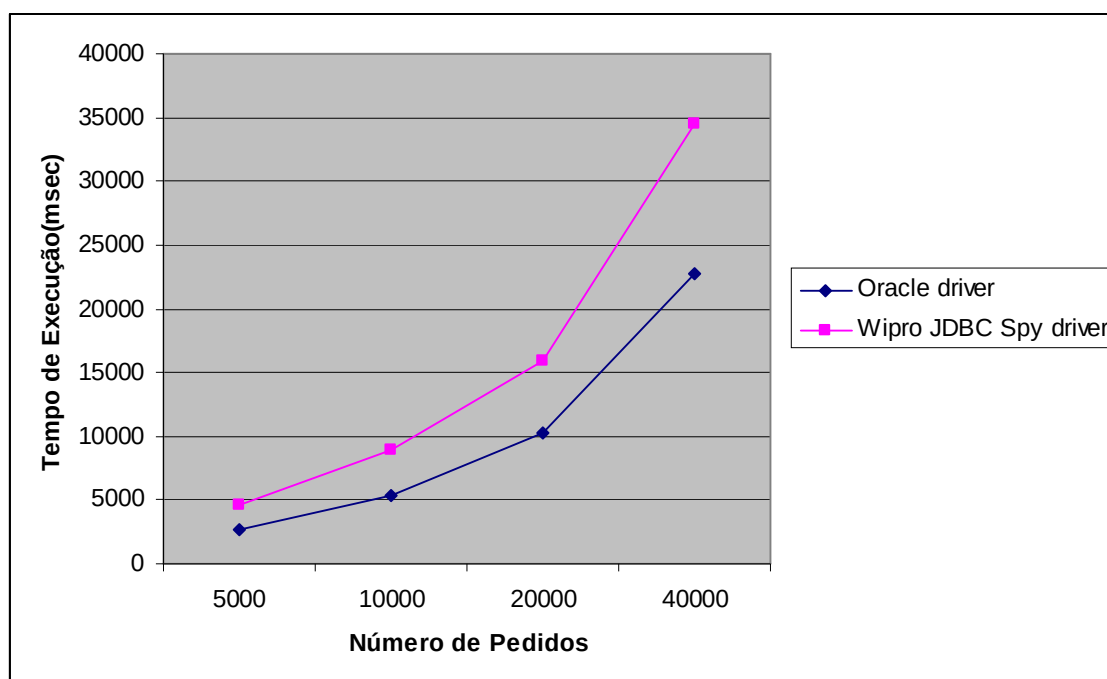


Figura 6-3 – Análise de performance do *driver* JDBC – Sobrecarga.

Os tempos obtidos nos testes realizados com pedidos de meio em meio segundo são apresentados na tabela 6-5. A figura 6-4 apresenta os mesmos dados graficamente. Os tempos estão representados em milissegundos e são tempos médios de resposta.

Tabela 6-5 – Análise de performance do *driver* JDBC – Funcionamento Normal.

	Número de Pedidos			
	5	10	20	40
Oracle Driver	2549	5051	10072	20097
Wipro JDBC Spy Driver	2565	5067	10103	20128

Resultados Experimentais

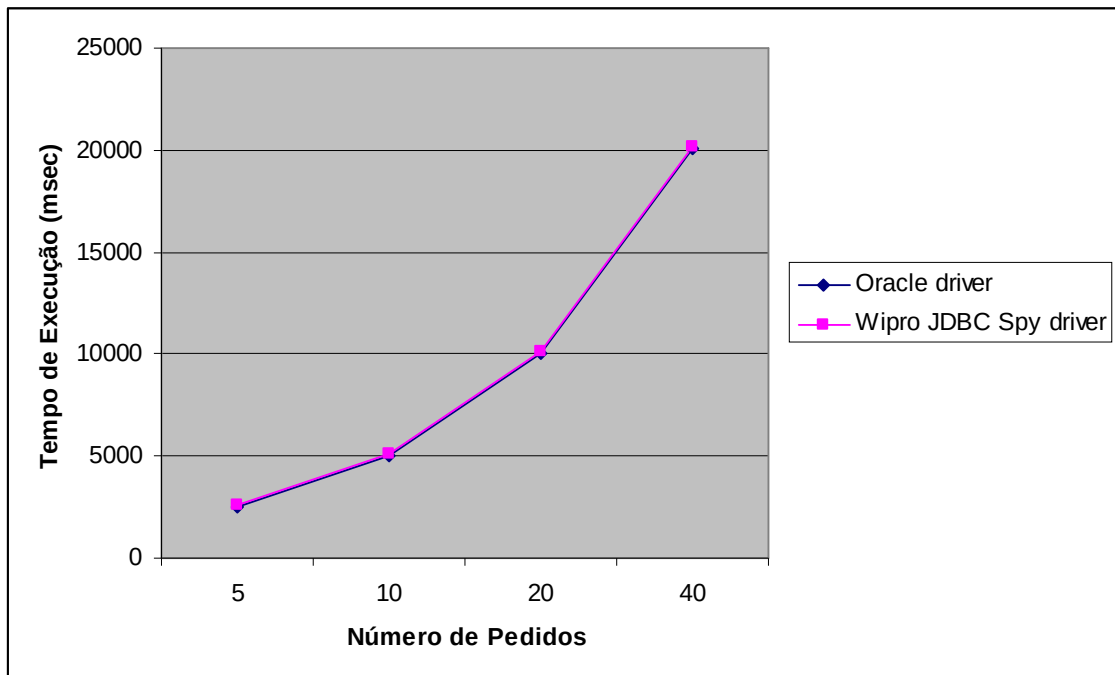


Figura 6-4 – Análise de performance do *driver* JDBC – Funcionamento Normal.

Analisando os valores obtidos nos dois testes verifica-se que o processamento extra causado pelo registo das mensagens provoca alguns atrasos quando o sistema se encontra em sobrecarga, que não são, no entanto, atrasos em valor excessivo. Verifica-se ainda que em funcionamento normal os atrasos são praticamente inexistentes.

Analisando estes valores e as fases do ciclo de vida de uma aplicação para as quais esta ferramenta é destinada, fases de desenvolvimento e de testes, verifica-se que os atrasos nos tempos de resposta não são notáveis uma vez que, nestas fases, os sistemas raramente estarão em sobrecarga.

6.6 Conclusões

Este capítulo apresenta os vários testes que foram realizados à ferramenta. O primeiro teste realizado demonstra uma sequência de utilização da ferramenta que passa por todas as fases da mesma. Os três testes seguintes testam profundamente as três fases da ferramenta. O último teste avalia a performance do *driver* JDBC especificado por esta ferramenta em comparação com o *driver* JDBC da Oracle. Para todos os testes realizados obteve-se uma resposta positiva e satisfatória.

7 Conclusões e Trabalho Futuro

Este capítulo apresenta as conclusões tiradas da realização deste projecto. Pretende-se transmitir uma retrospectiva do trabalho efectuado que proporcionará uma ligação entre o âmbito, os objectivos e os resultados obtidos. É também apresentada uma análise da satisfação dos objectivos propostos inicialmente e referido o trabalho futuro que deverá ser realizado para melhoria deste projecto.

7.1 Retrospectiva

A evolução da Internet e dos serviços *online* criam maiores necessidades de boa prestação de serviços e de rapidez na execução dos mesmos. O mundo do retalho é um dos impulsionadores da evolução nesta área, e por essa razão, também um dos mais exigentes. Os problemas de performance das aplicações são, por isso mesmo, cada vez mais frequentes e a sua resolução torna-se indispensável. O projecto descrito neste documento vai de encontro a este problema e o seu principal objectivo é ajudar na identificação da fonte dos problemas de performance.

A opção da Wipro Retail pela tentativa de desenvolver a sua própria ferramenta de apoio e resolução de problemas de performance em contraste com a utilização de uma solução já existente deveu-se, essencialmente, ao preço elevado das boas ferramentas existentes no mercado e à liberdade de adaptação a situações concretas que o desenvolvimento de uma ferramenta própria proporciona. Esta ferramenta surge como uma mais-valia e um auxiliar precioso para diversas situações problemáticas e deverá permitir a redução drástica do tempo dispendido por equipas da Wipro Retail na procura da fonte de problemas de performance.

Este projecto seguiu uma ideia utilizada em ferramentas *open source* e adicionou-lhe todo o conteúdo necessário para a obtenção da ferramenta desejada. A ferramenta Wipro JDBC Spy, produto deste projecto, permite registar todos os pedidos feitos por clientes à base de dados da aplicação monitorizada. Esses mesmos dados são depois analisados pela ferramenta e é gerado um relatório com informação essencial à descoberta das fontes problemáticas das falhas de performance.

A realização deste projecto provou ser um enorme desafio devido às várias áreas tecnológicas envolvidas mas foi também bastante enriquecedor, quer a nível profissional quer a nível pessoal. Permitiu adquirir conhecimentos importantes na área da informática mas também na área do retalho e a nível empresarial. Os conhecimentos adquiridos, como por exemplo, ao

nível de bases de dados, das soluções Oracle Retail, das transacções XA, do funcionamento dos grandes retalhistas e até das metodologias de trabalho da Wipro Retail são valores preciosos e acrescentam uma nova dimensão ao projecto, que passa de um simples projecto de desenvolvimento a uma fonte de informação e de aprendizagem essencial para o futuro do autor.

Finalmente, a nível social, a inclusão na empresa Wipro Retail foi bastante satisfatória, excedendo mesmo as expectativas. As condições de trabalho oferecidas assim como o apoio e a forma de estar de toda a equipa da empresa deram asas à boa prossecução deste projecto.

7.2 Satisfação dos Objectivos

Uma das características deste projecto foi a possibilidade dada ao autor de fazer um estudo sobre a problemática envolvente e de discutir e propor soluções e melhorias, permitindo encontrar alternativas e elevando o conhecimento pessoal. A descoberta da ferramenta *open source* Elvyx, as melhorias obtidas com o uso da base de dados HSQLDB e o uso de Java Reflection são alguns dos exemplos que foram fruto dessa característica.

Considerando as dificuldades inerentes ao projecto, os conhecimentos do autor relativamente à área envolvida, todo o estudo que foi necessário para o desenvolvimento do projecto e tendo em atenção os objectivos propostos no início do mesmo pode-se concluir que estes foram todos concretizados e que o trabalho realizado é satisfatório.

7.3 Trabalho Futuro

O trabalho futuro relacionado com este projecto passa sobretudo pela utilização da ferramenta em situações reais e pelo desenvolvimento de documentação para a empresa, relativa à ferramenta. A utilização da ferramenta em situações reais irá implicar a continuação do estudo, já iniciado, aos servidores aplicativos da Oracle e ao modo de alteração do *driver* JDBC especificado em cada aplicação existente nesses mesmos servidores. Para ser possível a monitorização de uma aplicação será necessário alterar essa especificação e passar a utilizar o *driver* JDBC da ferramenta Wipro JDBC Spy. A conclusão desta fase será essencial para o desenvolvimento da documentação relativa à ferramenta, que poderá assim ser desenvolvida completamente, considerando todas as fases e todas as configurações necessárias para a sua utilização. O conjunto destes dois factores determinará o sucesso da ferramenta no seio da empresa. Este trabalho foi já iniciado pelo autor e espera-se que seja concluído num futuro próximo.

Referências

- [APD09] *Rapidly Resolve Performance Problems in Distributed Java & .NET Applications*, 2009, <http://www.application-performance-diagnostics.com/> (Acedido em Junho de 2009)
- [APPP09] PurePath Technology, *PurePath Technology®: The quickest way from a problem's symptom to its solution*, Application Performance, 2009, <http://www.applicationperformance.com/index.php/solutions/dynatrace/44-purepath-technology> (Acedido em Junho de 2009)
- [APPAR08] Padma Apparao, *Determine the Correct XML Parser Type for a Java Application*, Intel Software Network, Maio, 2008, <http://software.intel.com/en-us/articles/determine-the-correct-xml-parser-type-for-a-java-application/> (Acedido em Junho de 2009)
- [BBKCOL] Simple API for XML (SAX) and the Document Object Model (DOM), School of Computer Science and Information Systems, Birkbeck College, University of London, <http://www.dcs.bbk.ac.uk/~mick/academic/xml/dip/sax-dom.shtml> (Acedido em Junho de 2009)
- [BDS91] J. S. Baron, B. Davies, D. Swindley, *Dictionary of Retailing*, Macmillan, 1991.
- [BHAEC02] Bernard van Haecke, *JDBC 3.0 - Java Database Connectivity*, M&T Books, 2002. ISBN: 978-0764548758
- [BLOCH05] Joshua Bloch, *Effective Java: Programming Language Guide*, pp. 158-161, Addison-Wesley, 2005. ISBN: 978-0201310054
- [CRAFT05] Craftsman Spy JDBC Driver 1.0.5, Craftsman, 2005, <http://zer0.free.fr/craftsman/spy.php> (Acedido em Junho de 2009)
- [DBOOK] DevelopersBook. JDBC Interview Questions. <http://www.developersbook.com/jdbc/interview-questions/jdbc-interview-questions-faqs.php> (Acedido em Junho de 2009)
- [DIGES09] Apache Commons Digester, Apache Software Foundation, 2007, <http://commons.apache.org/digester/index.html> (Acedido em Junho de 2009)

- [DREW02] Robb Drew, *Stopping Web Performance Thieves*, Datamation, Junho, 2002, <http://itmanagement.earthweb.com/ecom/article.php/1370691> (Acedido em Junho de 2009)
- [DTRAC09] dynaTrace®, dynaTrace Software®, 2009, <http://www.dynatrace.com/en/application-performance-management-product.aspx> (Acedido em Junho de 2009)
- [EGAM98] Erich Gamma, *Design Patterns: elements of reusable object-oriented software*, Versão CD, pp. 207-216, Addison-Wesley Longman, Incorporated, 1998. ISBN: 978-0201633610
- [ELVYX08] Elvyx, Armando Perdomo, 2008, <http://www.elvyx.com/index.html> (Acedido em Junho de 2009)
- [GDAVI93] Gary Davies, *Is Retailing What The Dictionaries Say It Is*, Manchester Business School, Manchester, International Journal of Retail & Distribution Management, Volume 21, Número 2, 1993.
- [GULCU03] Ceki Gulcu, *The complete Log4j Manual: The Reliable, Fast and Flexible Logging Framework for Java*, QOS.ch, 2003. ISBN: 978-2970036906
- [GUV03] Manuel Guzman, Bert Verstappen, *What is Monitoring*, pp. 5-6, Huridocs, 2003. ISBN: 978-9295015043
- [HSQLDB] HSQLDB – 100% Java Database, Lightweight 100% Java SQL Database Engine, 2009, <http://hsqldb.org/> (Acedido em Junho de 2009)
- [JACKS03] Jack Shirazi, *Java Performance Tuning*. pp.395-426, 445-461, 463-499 O'Reilly, Janeiro, 2003. ISBN: 978-0596003777
- [JAMON07] JAMon, Steve Souza, 2007, <http://jamonapi.sourceforge.net/> (Acedido em Junho de 2009)
- [JAXP09] Java API for XML Processing, Java.net, Sun Microsystems, 2009, <http://jaxp.dev.java.net> (Acedido em Junho de 2009)
- [JETTY09] Jetty, 2009, <http://jetty.mortbay.org/jetty/index.html> (Acedido em Junho de 2009)
- [JLHTTP] Java Lightweight HTTP Server (Web Server), JLHTTP, 2009, <http://www.freeutils.net/source/jlhttp/> (Acedido em Junho de 2009)
- [JPETU09] Java Performance Tuning, Fasterj, 2009. <http://www.javaperformancetuning.com/index.shtml> (Acedido em Junho de 2009)
- [JUGANL] Julia Gutjahr, Andreas Loew, *Scalability and Performance: JDBC Best Practices and Pitfalls*, Sun Java Center, Sun Microsystems GmbH
- [JXIN09] JXInsight®, JInspired®, 2009, <http://www.jinspired.com/products/jxinsight/> (Acedido em Junho de 2009)
- [KMMS04] Kavya Muthanna, Marina Sum, *Monitoring Application Performance in Sun Java System Application Server 7.x with PerformaSure*, Agosto 2004, <http://developers.sun.com/appserver/reference/techart/performasure.html> (Acedido em Junho de 2009)
- [LOG4J07] Apache Log4j, Apache Software Foundation, 2007, <http://logging.apache.org/log4j/1.2/index.html> (Acedido em Junho de 2009)
- [MNMB07] Mirko Novakovic, Marc van den Bogaard, *Java Performance Tools, Part2 - Profiling, Diagnosis and Monitoring*, Javamagazin, Dezembro, 2007.
- [NHTTPD] NanoHTTPD, Jarno Elonen, 2008, <http://elonen.iki.fi/code/nanohttpd/> (Acedido em Junho de 2009)

- [NIKNU05] Pat Niemeyer, Jonathan Knudsen, *Learning Java 3rd Edition*, pp. 200-211, O'Reilly Media, Inc., 2005. ISBN: 978-0596008734
- [ODB10G] Oracle Database 10g, Oracle, 2009, <http://www.oracle.com/technology/products/database/xe/index>. (Acedido em Junho de 2009)
- [ORA05a] Thomas Van Raalte, *Oracle Application Server Performance Guide, 10g Release 2 (10.1.2)*, Julho, 2005, Oracle
- [ORA05b] Theresa R. Bandy, *Oracle Application Server Concepts, 10g Release 2 (10.1.2)*, Julho, 2005, Oracle
- [ORA07] Venkatasubramaniam Iyer, Elizabeth Hanes Perry, Brian Wright, Thomas Pfaeffle, *Oracle Database – JDBC Developer's Guide and Reference, 10g Release 2 (10.2)*, Janeiro, 2007, Oracle
- [P6SPY03] P6Spy, P6Spy, 2003, <http://www.p6spy.com/> (Acedido em Junho de 2009)
- [PHM08] Philip H. Mitchell, *Discovery-Based Retail: Unlock Your Store's Potential!*, pp. 13-18, Bascom Hill Publishing Group, Agosto, 2008. ISBN: 978-0979846793
- [PSURE09] PerformaSure[®], Quest Software[®], 2009, <http://www.quest.com/performasure/> (Acedido em Junho de 2009)
- [TIMG06] Tim Gorman, *SQL Tuning for DBAs: Reactive Performance Tuning*, Evergreen Database Technologies, 2006. <http://www.evdbt.com/papers.htm> (Acedido em Junho de 2009)
- [VANA09] Vantage Analyzer[®], Compuware[®], 2009, <http://www.compuware.com/products/vantage/vantageanalyzer.asp> (Acedido em Junho de 2009)
- [VERIN09] Veritas InDepth[®], Symantec[™], 2009, <http://www.symantec.com/business/support/overview.jsp?pid=15115> (Acedido em Junho de 2009)
- [VQEWEB] VQEmbWeb, Gareth Cronin, <http://www.croninsolutions.com/vqembweb/> (Acedido em Junho de 2009)
- [WILY09] Wily APM[®], CA[®], 2009, <http://www.ca.com/us/application-performance-management.aspx> (Acedido em Junho de 2009)
- [WIPR09] Wipro Technologies, Retail, 2009. <http://www.wipro.com/retail/retail/index.htm> (Acedido em Junho de 2009)
- [XERC08] Apache Xerces Java XML Parser, Apache Software Foundation, 2008, http://projects.apache.org/projects/xerces_java_xml_parser.html (Acedido em Junho de 2009)

Anexo A – Modelo das Classes “Analyzers”

Este anexo apresenta o modelo das classes “Analyzers” desenvolvido para facilitar a construção de novas classes de análise de dados.

```

/*****
/*
/*          TEMPLATE TO CREATE ANALYZERS CLASS          */
/*
*****/

package com.wipro.analyzers;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Template extends Analyzer {

    Template analyzer;
    Connection conn;

    @Override
    public void execute() {
        try {
            analyzer = new Template();

            conn = analyzer.getConn();

            String lvl = INFO;

            /*****
            /*          DO YOUR EXECUTION CODE HERE          */
            *****/

            //If you need to use parameters uncomment the next line

```

```

//to retrieve them from the properties file
//String[] params = null;
//try {
//    params = analyzer.getParams();
//    if (params.length == 0) {
//        analyzer.log("ERROR: This analyzer needs parameters.", ERROR);
//        return;
//    }
//} catch (FileNotFoundException ex) {
//    Logger.getLogger(Template.class.getName()).log(Level.SEVERE, "wipro.spy.properties not found.", ex);
//} catch (IOException ex) {
//    Logger.getLogger(Template.class.getName()).log(Level.SEVERE, "wipro.spy.properties not found.", ex);
//}

//Remember to log using:
//analyzer.log("message", lvl);
//or
//analyzer.log("message", lvl, tag, stacktraces);

} catch (ClassNotFoundException ex) {
    Logger.getLogger(Template.class.getName()).log(Level.SEVERE,
"Class"+Template.class.getName()+"not found.", ex);
} catch (SQLException ex) {
    Logger.getLogger(Template.class.getName()).log(Level.SEVERE, "There was an error while connecting
with the database.", ex);
} finally{
    try {
        analyzer.end(conn);
    } catch (SQLException ex) {
        Logger.getLogger(Template.class.getName()).log(Level.SEVERE, "There was an error while connecting
with the database.", ex);
    } catch (FileNotFoundException ex) {
        Logger.getLogger(Template.class.getName()).log(Level.SEVERE, "wipro.spy.properties not found.",
ex);
    } catch (IOException ex) {
        Logger.getLogger(Template.class.getName()).log(Level.SEVERE, null, ex);
    }
}
}
}
}

```

Anexo B – Classe de Teste: Test4Analyzers

Este anexo apresenta a classe “Test4Analyzers” utilizada para executar os testes efectuados à ferramentas, enunciados no capítulo Resultados Experimentais.

```
package Test;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * Test Class for Analyzers - WiproJDBCSpy
 * @author Pedro Teixeira <pedro.teixeira@wipro.com>
 *
 * This test class will create data to test all the analyzers.
 *
 * @version 2009/06/13
 */
/**
 * Developed by Wipro
 */
public class Test4Analyzers {

    public static void main(String args[]) throws SQLException, InterruptedException, InstantiationException,
    IllegalAccessException {

        System.out.println("Beginning test...");
        ResultSet rset = null;

        try {
            System.out.println("Loading Class: \com.wipro.Driver\");
            Class.forName("com.wipro.Driver");
            //Class.forName("oracle.jdbc.OracleDriver");
            Connection conn;
```

```

    conn =
DriverManager.getConnection("jdbc:elvyx://localhost:4448/?elvyx.real_driver=oracle.jdbc.driver.OracleDriver&elvyx.real_jdbc=jdbc%3Aoracle%3Athin%3A%40192.168.38.128%3A1521%3AXE&user=wipro&password=admin");
//conn = DriverManager.getConnection("jdbc:oracle:thin:@192.168.38.128:1521:XE", "wipro", "admin");
conn.setAutoCommit(false);

Statement stmt = conn.createStatement();

/*****
//Test ConsecutiveCalls Analyzer
//This analyzer detects consecutive calls from exactly the same place in code in less then a given time
//NOTE: These 7 calls will also be detected by ConsecutiveSQLsAnalyzer
System.out.println("Generating ConsecutiveCalls Analyzer data...");

//Using 1000 msecs this 4 statements should be detected
for (int i = 0; i < 4; i++) {
    Thread.sleep(500);
    rset = executeQ(stmt, "SELECT * FROM warehouse order by id");
}

//Using 1000 msecs this 3 statements should not be detected
for (int i = 0; i < 3; i++) {
    Thread.sleep(2000);
    rset = executeQ(stmt, "SELECT * FROM warehouse order by id");
}
*****/

//Test ConsecutiveCallsFromTablesAnalyzer
//This analyzer detects consecutive calls from exactly the same place in code using some specified tables in less
then a given time
System.out.println("Generating ConsecutiveCallsFromTablesAnalyzer data...");

//Using 1000 msecs and searching for the tables: PRODUCT | WAREHOUSE_PRODUCT this 4 statements should
be detected
//NOTE: This will also be detected by ConsecutiveCalls Analyzer
//NOTE: These 4 calls will also be detected by ConsecutiveSQLsAnalyzer
for (int i = 0; i < 4; i++) {
    Thread.sleep(500);
    rset = executeQ(stmt, "SELECT * FROM product order by id");
}

//Using 1000 msecs and searching for the tables: PRODUCT | WAREHOUSE_PRODUCT this 3 statements should
not be detected
//NOTE: These 3 calls will also be detected by ConsecutiveSQLsAnalyzer
for (int i = 0; i < 3; i++) {
    Thread.sleep(500);
    rset = executeQ(stmt, "SELECT * FROM po_detail order by id");
}
*****/

//Test PatternsAnalyzer
//This analyzer detects patterns given a minimumSQL number, an execution time average,
//a standard deviation from that execution time average and an execution times deviation
//to detect longer calls in that pattern. Also, the calls must be from the same SQL and
//the same stacktrace
System.out.println("Generating PatternsAnalyzer data...");

//Using 5 minimum SQLs, average time of 3.1 secs, standard deviation of 500msecs and execution
//times deviation of 10msecs a pattern should be detected

```

```

//NOTE: These 6 calls will also be detected by ConsecutiveSQLsAnalyzer
for (int i = 0; i < 6; i++) {
    Thread.sleep(3100);

    rset = executeQ(stmt, "SELECT * FROM product order by id");
}
/***** /
rset.close();
stmt.close();
conn.commit();
conn.close();

} catch (ClassNotFoundException ex) {
    //Testlogger.getLogger(Test.class.getName()).log(Level.SEVERE, null, ex);
}
}

private static ResultSet executeQ(Statement stmt, String sql) throws SQLException {
    return stmt.executeQuery(sql);
}
}

```

Anexo C – Exemplo de um Ficheiro de Propriedades

Neste anexo é apresentado um exemplo de um ficheiro de propriedades do Wipro JDBC Spy. Este exemplo foi utilizado para o teste demonstrativo de execução.

```
#USED FOR TESTING ANALYZERS
```

```
#Use this file to change the Spy properties
```

```
#HTTPD Properties
```

```
#HTTPD.Initial.State should be enabled or disabled. Default is enabled  
HTTPD.Initial.State=enabled
```

```
#HTTPDport is the port to use on your browser. Default port is 17778  
HTTPDport=17782
```

```
#HTTPDmaxQueries is maximum number of queries that will be shown on your  
#browser. The last HTTPDmaxQueries are shown. Default is 100  
HTTPDmaxQueries=50
```

```
#Report Classes - Choose the classes to load and generate report  
LoadClass=com.wipro.analyzers.ConsecutiveCallsAnalyzer  
LoadClass=com.wipro.analyzers.ConsecutiveCallsFromTablesAnalyzer  
LoadClass=com.wipro.analyzers.ConsecutiveSQLsAnalyzer  
LoadClass=com.wipro.analyzers.ExecutionTimesAnalyzer  
LoadClass=com.wipro.analyzers.ExecutionTimesOnTablesAnalyzer  
LoadClass=com.wipro.analyzers.PatternsAnalyzer
```

```
#Classes parameters - Should be like ClassName.Parameter=Value  
com.wipro.analyzers.ConsecutiveCalls.timeBetweenSQLs=1000
```

```
com.wipro.analyzers.ConsecutiveCallsFromTablesAnalyzer.tables= PRODUCT |  
WAREHOUSE_PRODUCT  
com.wipro.analyzers.ConsecutiveCallsFromTablesAnalyzer.timeBetweenSQLS = 1000  
com.wipro.analyzers.ExecutionTimesOnTablesAnalyzer.tables =PRODUCT |  
WAREHOUSE_PRODUCT | PO_DETAIL  
com.wipro.analyzers.PatternsAnalyzer.minimumSQLs=5  
com.wipro.analyzers.PatternsAnalyzer.averageTime=3  
com.wipro.analyzers.PatternsAnalyzer.standardDeviation=0.5  
com.wipro.analyzers.PatternsAnalyzer.executionTimesDeviation=10
```

Anexo D – Exemplo de um Ficheiro XML de Registo de Pedidos

Este anexo apresenta um exemplo de um ficheiro XML de registo de pedidos. Este ficheiro foi obtido com a execução da classe “Test4Analyzers” apresentada no Anexo B.

```
<?xml version="1.0" encoding="UTF-8" ?>
<WiproSpyLog>
  <DBEvent>
    <LogDate>Thu Jun 18 12:23:01 BST 2009</LogDate>
    <ConnID>1</ConnID>
    <InitDate>
      <Timestamp>1245324181694</Timestamp>
    </InitDate>
    <InitExec>
      <Timestamp>1245324181694</Timestamp>
    </InitExec>
    <EndExec>
      <Timestamp>1245324181694</Timestamp>
    </EndExec>
    <Category>com.wipro.Connection</Category>
    <SQL>open connection (oc)</SQL>
    <PreparedSQL>.{oc:1:1</PreparedSQL>
    <RSetSize>0</RSetSize>
    <ElapsedTime>0 msec</ElapsedTime>
    <StackTrace>Test.Test4Analyzers.main(Test4Analyzers.java:34)
  </StackTrace>
</DBEvent>
  <DBEvent>
    <LogDate>Thu Jun 18 12:23:02 BST 2009</LogDate>
    <ConnID>1</ConnID>
    <InitDate>
      <Timestamp>1245324182226</Timestamp>
```

```

</InitDate>
<InitExec>
  <Timestamp>1245324182226</Timestamp>
</InitExec>
<EndExec>
  <Timestamp>1245324182476</Timestamp>
</EndExec>
<Category>com.elvyx.Statement</Category>
<SQL>SELECT * FROM warehouse order by id</SQL>
<PreparedSQL>SELECT * FROM warehouse order by id</PreparedSQL>
<RSetSize>0</RSetSize>
<ElapsedTime>250 msec</ElapsedTime>
  <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)      ->
Test.Test4Analyzers.main(Test4Analyzers.java:49)</StackTrace>
</DBEvent>
<DBEvent>
  <LogDate>Thu Jun 18 12:23:03 BST 2009</LogDate>
  <ConnID>1</ConnID>
  <InitDate>
    <Timestamp>1245324182976</Timestamp>
  </InitDate>
  <InitExec>
    <Timestamp>1245324182976</Timestamp>
  </InitExec>
  <EndExec>
    <Timestamp>1245324183195</Timestamp>
  </EndExec>
  <Category>com.elvyx.Statement</Category>
  <SQL>SELECT * FROM warehouse order by id</SQL>
  <PreparedSQL>SELECT * FROM warehouse order by id</PreparedSQL>
  <RSetSize>0</RSetSize>
  <ElapsedTime>219 msec</ElapsedTime>
    <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)      ->
Test.Test4Analyzers.main(Test4Analyzers.java:49)</StackTrace>
</DBEvent>
<DBEvent>
  <LogDate>Thu Jun 18 12:23:03 BST 2009</LogDate>
  <ConnID>1</ConnID>
  <InitDate>
    <Timestamp>1245324183696</Timestamp>
  </InitDate>
  <InitExec>
    <Timestamp>1245324183696</Timestamp>
  </InitExec>
  <EndExec>
    <Timestamp>1245324183915</Timestamp>
  </EndExec>
  <Category>com.elvyx.Statement</Category>
  <SQL>SELECT * FROM warehouse order by id</SQL>
  <PreparedSQL>SELECT * FROM warehouse order by id</PreparedSQL>
  <RSetSize>0</RSetSize>
  <ElapsedTime>219 msec</ElapsedTime>
    <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)      ->
Test.Test4Analyzers.main(Test4Analyzers.java:49)</StackTrace>
</DBEvent>
<DBEvent>
  <LogDate>Thu Jun 18 12:23:04 BST 2009</LogDate>

```

```

    <ConnID>1</ConnID>
    <InitDate>
      <Timestamp>1245324184415</Timestamp>
    </InitDate>
    <InitExec>
      <Timestamp>1245324184415</Timestamp>
    </InitExec>
    <EndExec>
      <Timestamp>1245324184618</Timestamp>
    </EndExec>
    <Category>com.elvyx.Statement</Category>
    <SQL>SELECT * FROM warehouse order by id</SQL>
    <PreparedSQL>SELECT * FROM warehouse order by id</PreparedSQL>
    <RSetSize>0</RSetSize>
    <ElapsedTime>203 msec</ElapsedTime>
    <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)    ->
Test.Test4Analyzers.main(Test4Analyzers.java:49)</StackTrace>
  </DBEvent>
  <DBEvent>
    <LogDate>Thu Jun 18 12:23:06 BST 2009</LogDate>
    <ConnID>1</ConnID>
    <InitDate>
      <Timestamp>1245324186620</Timestamp>
    </InitDate>
    <InitExec>
      <Timestamp>1245324186620</Timestamp>
    </InitExec>
    <EndExec>
      <Timestamp>1245324186824</Timestamp>
    </EndExec>
    <Category>com.elvyx.Statement</Category>
    <SQL>SELECT * FROM warehouse order by id</SQL>
    <PreparedSQL>SELECT * FROM warehouse order by id</PreparedSQL>
    <RSetSize>0</RSetSize>
    <ElapsedTime>204 msec</ElapsedTime>
    <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)    ->
Test.Test4Analyzers.main(Test4Analyzers.java:55)</StackTrace>
  </DBEvent>
  <DBEvent>
    <LogDate>Thu Jun 18 12:23:09 BST 2009</LogDate>
    <ConnID>1</ConnID>
    <InitDate>
      <Timestamp>1245324188825</Timestamp>
    </InitDate>
    <InitExec>
      <Timestamp>1245324188825</Timestamp>
    </InitExec>
    <EndExec>
      <Timestamp>1245324189013</Timestamp>
    </EndExec>
    <Category>com.elvyx.Statement</Category>
    <SQL>SELECT * FROM warehouse order by id</SQL>
    <PreparedSQL>SELECT * FROM warehouse order by id</PreparedSQL>
    <RSetSize>0</RSetSize>
    <ElapsedTime>188 msec</ElapsedTime>
    <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)    ->
Test.Test4Analyzers.main(Test4Analyzers.java:55)</StackTrace>

```

```

</DBEvent>
<DBEvent>
  <LogDate>Thu Jun 18 12:23:11 BST 2009</LogDate>
  <ConnID>1</ConnID>
  <InitDate>
    <Timestamp>1245324191015</Timestamp>
  </InitDate>
  <InitExec>
    <Timestamp>1245324191015</Timestamp>
  </InitExec>
  <EndExec>
    <Timestamp>1245324191218</Timestamp>
  </EndExec>
  <Category>com.elvyx.Statement</Category>
  <SQL>SELECT * FROM warehouse order by id</SQL>
  <PreparedStatement>SELECT * FROM warehouse order by id</PreparedStatement>
  <RSetSize>0</RSetSize>
  <ElapsedTime>203 msec</ElapsedTime>
  <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)      ->
Test.Test4Analyzers.main(Test4Analyzers.java:55)</StackTrace>
</DBEvent>
<DBEvent>
  <LogDate>Thu Jun 18 12:23:11 BST 2009</LogDate>
  <ConnID>1</ConnID>
  <InitDate>
    <Timestamp>1245324191719</Timestamp>
  </InitDate>
  <InitExec>
    <Timestamp>1245324191719</Timestamp>
  </InitExec>
  <EndExec>
    <Timestamp>1245324191938</Timestamp>
  </EndExec>
  <Category>com.elvyx.Statement</Category>
  <SQL>SELECT * FROM product order by id</SQL>
  <PreparedStatement>SELECT * FROM product order by id</PreparedStatement>
  <RSetSize>0</RSetSize>
  <ElapsedTime>219 msec</ElapsedTime>
  <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)      ->
Test.Test4Analyzers.main(Test4Analyzers.java:69)</StackTrace>
</DBEvent>
<DBEvent>
  <LogDate>Thu Jun 18 12:23:12 BST 2009</LogDate>
  <ConnID>1</ConnID>
  <InitDate>
    <Timestamp>1245324192438</Timestamp>
  </InitDate>
  <InitExec>
    <Timestamp>1245324192438</Timestamp>
  </InitExec>
  <EndExec>
    <Timestamp>1245324192641</Timestamp>
  </EndExec>
  <Category>com.elvyx.Statement</Category>
  <SQL>SELECT * FROM product order by id</SQL>
  <PreparedStatement>SELECT * FROM product order by id</PreparedStatement>
  <RSetSize>0</RSetSize>

```

```

        <ElapsedTime>203 msec</ElapsedTime>
        <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)      ->
Test.Test4Analyzers.main(Test4Analyzers.java:69)</StackTrace>
    </DBEvent>
    <DBEvent>
        <LogDate>Thu Jun 18 12:23:13 BST 2009</LogDate>
        <ConnID>1</ConnID>
        <InitDate>
            <Timestamp>1245324193142</Timestamp>
        </InitDate>
        <InitExec>
            <Timestamp>1245324193142</Timestamp>
        </InitExec>
        <EndExec>
            <Timestamp>1245324193361</Timestamp>
        </EndExec>
        <Category>com.elvyx.Statement</Category>
        <SQL>SELECT * FROM product order by id</SQL>
        <PreparedSQL>SELECT * FROM product order by id</PreparedSQL>
        <RSetSize>0</RSetSize>
        <ElapsedTime>219 msec</ElapsedTime>
        <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)      ->
Test.Test4Analyzers.main(Test4Analyzers.java:69)</StackTrace>
    </DBEvent>
    <DBEvent>
        <LogDate>Thu Jun 18 12:23:14 BST 2009</LogDate>
        <ConnID>1</ConnID>
        <InitDate>
            <Timestamp>1245324193861</Timestamp>
        </InitDate>
        <InitExec>
            <Timestamp>1245324193861</Timestamp>
        </InitExec>
        <EndExec>
            <Timestamp>1245324194065</Timestamp>
        </EndExec>
        <Category>com.elvyx.Statement</Category>
        <SQL>SELECT * FROM product order by id</SQL>
        <PreparedSQL>SELECT * FROM product order by id</PreparedSQL>
        <RSetSize>0</RSetSize>
        <ElapsedTime>204 msec</ElapsedTime>
        <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)      ->
Test.Test4Analyzers.main(Test4Analyzers.java:69)</StackTrace>
    </DBEvent>
    <DBEvent>
        <LogDate>Thu Jun 18 12:23:14 BST 2009</LogDate>
        <ConnID>1</ConnID>
        <InitDate>
            <Timestamp>1245324194565</Timestamp>
        </InitDate>
        <InitExec>
            <Timestamp>1245324194565</Timestamp>
        </InitExec>
        <EndExec>
            <Timestamp>1245324194784</Timestamp>
        </EndExec>
        <Category>com.elvyx.Statement</Category>

```

```

        <SQL>SELECT * FROM po_detail order by id</SQL>
        <PreparedStatement>SELECT * FROM po_detail order by id</PreparedStatement>
        <ResultSetSize>0</ResultSetSize>
        <ElapsedTime>219 msec</ElapsedTime>
        <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)    ->
Test.Test4Analyzers.main(Test4Analyzers.java:77)</StackTrace>
    </DBEvent>
    <DBEvent>
        <LogDate>Thu Jun 18 12:23:15 BST 2009</LogDate>
        <ConnID>1</ConnID>
        <InitDate>
            <Timestamp>1245324195285</Timestamp>
        </InitDate>
        <InitExec>
            <Timestamp>1245324195285</Timestamp>
        </InitExec>
        <EndExec>
            <Timestamp>1245324195488</Timestamp>
        </EndExec>
        <Category>com.elvyx.Statement</Category>
        <SQL>SELECT * FROM po_detail order by id</SQL>
        <PreparedStatement>SELECT * FROM po_detail order by id</PreparedStatement>
        <ResultSetSize>0</ResultSetSize>
        <ElapsedTime>203 msec</ElapsedTime>
        <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)    ->
Test.Test4Analyzers.main(Test4Analyzers.java:77)</StackTrace>
    </DBEvent>
    <DBEvent>
        <LogDate>Thu Jun 18 12:23:16 BST 2009</LogDate>
        <ConnID>1</ConnID>
        <InitDate>
            <Timestamp>1245324195988</Timestamp>
        </InitDate>
        <InitExec>
            <Timestamp>1245324195988</Timestamp>
        </InitExec>
        <EndExec>
            <Timestamp>1245324196192</Timestamp>
        </EndExec>
        <Category>com.elvyx.Statement</Category>
        <SQL>SELECT * FROM po_detail order by id</SQL>
        <PreparedStatement>SELECT * FROM po_detail order by id</PreparedStatement>
        <ResultSetSize>0</ResultSetSize>
        <ElapsedTime>204 msec</ElapsedTime>
        <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)    ->
Test.Test4Analyzers.main(Test4Analyzers.java:77)</StackTrace>
    </DBEvent>
    <DBEvent>
        <LogDate>Thu Jun 18 12:23:19 BST 2009</LogDate>
        <ConnID>1</ConnID>
        <InitDate>
            <Timestamp>1245324199304</Timestamp>
        </InitDate>
        <InitExec>
            <Timestamp>1245324199304</Timestamp>
        </InitExec>
        <EndExec>

```

```

    <Timestamp>1245324199492</Timestamp>
  </EndExec>
  <Category>com.elvyx.Statement</Category>
  <SQL>SELECT * FROM product order by id</SQL>
  <PreparedSQL>SELECT * FROM product order by id</PreparedSQL>
  <RSetSize>0</RSetSize>
  <ElapsedTime>188 msec</ElapsedTime>
  <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) ->
Test.Test4Analyzers.main(Test4Analyzers.java:97)</StackTrace>
</DBEvent>
<DBEvent>
  <LogDate>Thu Jun 18 12:23:22 BST 2009</LogDate>
  <ConnID>1</ConnID>
  <InitDate>
    <Timestamp>1245324202604</Timestamp>
  </InitDate>
  <InitExec>
    <Timestamp>1245324202604</Timestamp>
  </InitExec>
  <EndExec>
    <Timestamp>1245324202807</Timestamp>
  </EndExec>
  <Category>com.elvyx.Statement</Category>
  <SQL>SELECT * FROM product order by id</SQL>
  <PreparedSQL>SELECT * FROM product order by id</PreparedSQL>
  <RSetSize>0</RSetSize>
  <ElapsedTime>203 msec</ElapsedTime>
  <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) ->
Test.Test4Analyzers.main(Test4Analyzers.java:97)</StackTrace>
</DBEvent>
<DBEvent>
  <LogDate>Thu Jun 18 12:23:26 BST 2009</LogDate>
  <ConnID>1</ConnID>
  <InitDate>
    <Timestamp>1245324205919</Timestamp>
  </InitDate>
  <InitExec>
    <Timestamp>1245324205919</Timestamp>
  </InitExec>
  <EndExec>
    <Timestamp>1245324206123</Timestamp>
  </EndExec>
  <Category>com.elvyx.Statement</Category>
  <SQL>SELECT * FROM product order by id</SQL>
  <PreparedSQL>SELECT * FROM product order by id</PreparedSQL>
  <RSetSize>0</RSetSize>
  <ElapsedTime>204 msec</ElapsedTime>
  <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) ->
Test.Test4Analyzers.main(Test4Analyzers.java:97)</StackTrace>
</DBEvent>
<DBEvent>
  <LogDate>Thu Jun 18 12:23:29 BST 2009</LogDate>
  <ConnID>1</ConnID>
  <InitDate>
    <Timestamp>1245324209235</Timestamp>
  </InitDate>
  <InitExec>

```

```

        <Timestamp>1245324209235</Timestamp>
    </InitExec>
    <EndExec>
        <Timestamp>1245324209438</Timestamp>
    </EndExec>
    <Category>com.elvyx.Statement</Category>
    <SQL>SELECT * FROM product order by id</SQL>
    <PreparedStatement>SELECT * FROM product order by id</PreparedStatement>
    <ResultSetSize>0</ResultSetSize>
    <ElapsedTime>203 msec</ElapsedTime>
    <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) ->
Test.Test4Analyzers.main(Test4Analyzers.java:97)</StackTrace>
</DBEvent>
<DBEvent>
    <LogDate>Thu Jun 18 12:23:32 BST 2009</LogDate>
    <ConnID>1</ConnID>
    <InitDate>
        <Timestamp>1245324212551</Timestamp>
    </InitDate>
    <InitExec>
        <Timestamp>1245324212551</Timestamp>
    </InitExec>
    <EndExec>
        <Timestamp>1245324212754</Timestamp>
    </EndExec>
    <Category>com.elvyx.Statement</Category>
    <SQL>SELECT * FROM product order by id</SQL>
    <PreparedStatement>SELECT * FROM product order by id</PreparedStatement>
    <ResultSetSize>0</ResultSetSize>
    <ElapsedTime>203 msec</ElapsedTime>
    <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) -
> Test.Test4Analyzers.main(Test4Analyzers.java:97)</StackTrace>
</DBEvent>
<DBEvent>
    <LogDate>Thu Jun 18 12:23:36 BST 2009</LogDate>
    <ConnID>1</ConnID>
    <InitDate>
        <Timestamp>1245324215850</Timestamp>
    </InitDate>
    <InitExec>
        <Timestamp>1245324215850</Timestamp>
    </InitExec>
    <EndExec>
        <Timestamp>1245324216054</Timestamp>
    </EndExec>
    <Category>com.elvyx.Statement</Category>
    <SQL>SELECT * FROM product order by id</SQL>
    <PreparedStatement>SELECT * FROM product order by id</PreparedStatement>
    <ResultSetSize>0</ResultSetSize>
    <ElapsedTime>204 msec</ElapsedTime>
    <StackTrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) -
> Test.Test4Analyzers.main(Test4Analyzers.java:97)</StackTrace>
</DBEvent>
<DBEvent>
    <LogDate>Thu Jun 18 12:23:36 BST 2009</LogDate>
    <ConnID>1</ConnID>
    <InitDate>

```

```

        <Timestamp>1245324216054</Timestamp>
    </InitDate>
    <InitExec>
        <Timestamp>1245324216054</Timestamp>
    </InitExec>
<EndExec>
    <Timestamp>1245324216054</Timestamp>
</EndExec>
<Category>com.wipro.Connection</Category>
<SQL>commit (co)</SQL>
<PreparedStatement>{co}:1</PreparedStatement>
<RSetSize>0</RSetSize>
<ElapsedTime>0 msec</ElapsedTime>
</DBEvent>
<DBEvent>
    <LogDate>Thu Jun 18 12:23:36 BST 2009</LogDate>
    <ConnID>1</ConnID>
    <InitDate>
        <Timestamp>1245324216054</Timestamp>
    </InitDate>
    <InitExec>
        <Timestamp>1245324216054</Timestamp>
    </InitExec>
    <EndExec>
        <Timestamp>1245324216054</Timestamp>
    </EndExec>
    <Category>com.wipro.Connection</Category>
    <SQL>close connection (cc)</SQL>
    <PreparedStatement>}{cc:1:0</PreparedStatement>
    <RSetSize>0</RSetSize>
    <ElapsedTime>0 msec</ElapsedTime>
</DBEvent>
<DBEvent>
    <LogDate>Thu Jun 18 12:23:36 BST 2009</LogDate>
    <ConnID>1</ConnID>
    <InitDate>
        <Timestamp>1245324181694</Timestamp>
    </InitDate>
    <InitExec>
        <Timestamp>1245324181694</Timestamp>
    </InitExec>
    <EndExec>
        <Timestamp>1245324216054</Timestamp>
    </EndExec>
    <Category>com.wipro.Connection</Category>
    <SQL>[open-close] connection</SQL>
    <PreparedStatement>** open-close ** connection:1</PreparedStatement>
    <RSetSize>0</RSetSize>
    <ElapsedTime>34360 msec</ElapsedTime>
</DBEvent>
</WiproSpyLog>

```

Anexo E – Exemplo de um Relatório Final

Este anexo apresenta um exemplo de um relatório final. Este relatório foi obtido analisando os dados gerados pela execução da classe de teste “Test4Analyzers” apresentada no Anexo B.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <Report>
    <Event>
      <LogDate>Thu Jun 18 12:25:51 BST 2009</LogDate>
      <LogLevel>WARN</LogLevel>
      <Analyzer>com.wipro.analyzers.ConsecutiveCalls</Analyzer>
      <Message>Statement "SELECT * FROM warehouse order by id" with
stacktrace      "Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:49)" had 4 calls in less than
1000 milliseconds.</Message>
    </Event>
    <Event>
      <LogDate>Thu Jun 18 12:25:51 BST 2009</LogDate>
      <LogLevel>WARN</LogLevel>
      <Analyzer>com.wipro.analyzers.ConsecutiveCalls</Analyzer>
      <Message>Statement "SELECT * FROM product order by id" with stacktrace
"Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:69)" had 4 calls in less than
1000 milliseconds.</Message>
    </Event>
    <Event>
      <LogDate>Thu Jun 18 12:25:51 BST 2009</LogDate>
      <LogLevel>WARN</LogLevel>
      <Analyzer>com.wipro.analyzers.ConsecutiveCalls</Analyzer>
      <Message>Statement "SELECT * FROM po_detail order by id" with
stacktrace      "Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:77)" had 3 calls in less than
1000 milliseconds.</Message>
    </Event>
```

```

<Event>
  <LogDate>Thu Jun 18 12:25:51 BST 2009</LogDate>
  <LogLevel>WARN</LogLevel>
  <Analyzer>com.wipro.analyzers.ConsecutiveCallsFromTablesAnalyzer
</Analyzer>
  <Message>Statement "SELECT * FROM product order by id" with stacktrace
"Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:69)" had 4 calls in less than
1000 milliseconds. Searched for tables: [ PRODUCT |
WAREHOUSE_PRODUCT].</Message>
</Event>
<Event>
  <LogDate>Thu Jun 18 12:25:51 BST 2009</LogDate>
  <LogLevel>WARN</LogLevel>
  <Analyzer>com.wipro.analyzers.ConsecutiveSQLsAnalyzer</Analyzer>
  <Message>Statement "SELECT * FROM warehouse order by id" had 7
consecutive calls.</Message>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:49)</Stacktrace>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:49)</Stacktrace>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:49)</Stacktrace>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:49)</Stacktrace>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:55)</Stacktrace>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:55)</Stacktrace>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:55)</Stacktrace>
  </Event>
<Event>
  <LogDate>Thu Jun 18 12:25:51 BST 2009</LogDate>
  <LogLevel>WARN</LogLevel>
  <Analyzer>com.wipro.analyzers.ConsecutiveSQLsAnalyzer</Analyzer>
  <Message>Statement "SELECT * FROM product order by id" had 4 consecutive
calls.</Message>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:69)</Stacktrace>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:69)</Stacktrace>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:69)</Stacktrace>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:69)</Stacktrace>
  </Event>
<Event>
  <LogDate>Thu Jun 18 12:25:51 BST 2009</LogDate>
  <LogLevel>WARN</LogLevel>
  <Analyzer>com.wipro.analyzers.ConsecutiveSQLsAnalyzer</Analyzer>
  <Message>Statement "SELECT * FROM po_detail order by id" had 3
consecutive calls.</Message>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:77)</Stacktrace>
    <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111)-
>Test.Test4Analyzers.main(Test4Analyzers.java:77)</Stacktrace>

```

```

        <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) -
>Test.Test4Analyzers.main(Test4Analyzers.java:77)</Stacktrace>
    </Event>
    <Event>
        <LogDate>Thu Jun 18 12:25:51 BST 2009</LogDate>
        <LogLevel>WARN</LogLevel>
        <Analyzer>com.wipro.analyzers.ConsecutiveSQLsAnalyzer</Analyzer>
        <Message>Statement "SELECT * FROM product order by id" had 6 consecutive
calls.</Message>
        <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) -
>Test.Test4Analyzers.main(Test4Analyzers.java:97)</Stacktrace>
        <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) -
>Test.Test4Analyzers.main(Test4Analyzers.java:97)</Stacktrace>
        <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) -
>Test.Test4Analyzers.main(Test4Analyzers.java:97)</Stacktrace>
        <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) -
>Test.Test4Analyzers.main(Test4Analyzers.java:97)</Stacktrace>
        <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) -
>Test.Test4Analyzers.main(Test4Analyzers.java:97)</Stacktrace>
        <Stacktrace>Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) -
>Test.Test4Analyzers.main(Test4Analyzers.java:97)</Stacktrace>
    </Event>
    <Event>
        <LogDate>Thu Jun 18 12:25:51 BST 2009</LogDate>
        <LogLevel>WARN</LogLevel>
        <Analyzer>com.wipro.analyzers.ExecutionTimesAnalyzer</Analyzer>
        <Message>Total time: 34.36 sec | Time on DB: 4.162 sec | 12% of time
spent on DB</Message>
    </Event>
    <Event>
        <LogDate>Thu Jun 18 12:25:51 BST 2009</LogDate>
        <LogLevel>INFO</LogLevel>
        <Analyzer>com.wipro.analyzers.ExecutionTimesOnTablesAnalyzer</Analyzer>
        <Message>Total time on databse: 4.162 sec | Time spent on tables [PRODUCT
| WAREHOUSE_PRODUCT | PO_DETAIL]: 2.676 sec. 64% spent on the specified
tables.</Message>
    </Event>
    <Event>
        <LogDate>Thu Jun 18 12:25:51 BST 2009</LogDate>
        <LogLevel>WARN</LogLevel>
        <Analyzer>com.wipro.analyzers.PatternsAnalyzer</Analyzer>
        <Message>A pattern with 6 calls has been detected on statement "SELECT *
FROM      product      order      by      id"      with      stacktrace
"Test.Test4Analyzers.executeQ(Test4Analyzers.java:111) -
>Test.Test4Analyzers.main(Test4Analyzers.java:97).</Message>
        <SQLExecutionDate>2009-06-18 12:23:19.304</SQLExecutionDate>
        <SQLExecutionTime>188</SQLExecutionTime>
        <SQLExecutionTimeDeviation>-12</SQLExecutionTimeDeviation>
        <SQLExecutionDate>2009-06-18 12:23:22.604</SQLExecutionDate>
        <SQLExecutionTime>203</SQLExecutionTime>
        <SQLExecutionDate>2009-06-18 12:23:25.919</SQLExecutionDate>
        <SQLExecutionTime>204</SQLExecutionTime>
        <SQLExecutionDate>2009-06-18 12:23:29.235</SQLExecutionDate>
        <SQLExecutionTime>203</SQLExecutionTime>
        <SQLExecutionDate>2009-06-18 12:23:32.551</SQLExecutionDate>
        <SQLExecutionTime>203</SQLExecutionTime>
        <SQLExecutionDate>2009-06-18 12:23:35.85</SQLExecutionDate>

```

```
<SQLExecutionTime>204</SQLExecutionTime>
<MinimumNumberOfSQLs>5</MinimumNumberOfSQLs>
<AverageTime>3.0 seconds</AverageTime>
<StandardDeviation>0.5 seconds</StandardDeviation>
<ExecutionTimesDeviation>10.0 milliseconds</ExecutionTimesDeviation>
<ExecutionTimesAverage>200 milliseconds</ExecutionTimesAverage>
</Event>
</Report>
```