FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Autonomic Computing – Registo de Eventos

Marcelo Fernando Magalhães Barreira

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Rui Filipe Lima Maranhão de Abreu (Doutor)

30 de Junho de 2010

Autonomic Computing – Registo de Eventos

Marcelo Fernando Magalhães Barreira

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: José Manuel Magalhães Cruz (Professor Auxiliar)

Vogal Externo: Ricardo Jorge Fernandes Chaves (Professor Auxiliar)

Orientador: Rui Filipe Lima Maranhão de Abreu (Professor Auxiliar Convidado)

_____

30 de Junho de 2010

Resumo

O tema **Autonomic Computing**, é o cerne do documento a apresentar, particularizando-se numa espécie de subtema, "**Registo de Eventos**", sobre o qual incidiu todo o desenvolvimento desta dissertação.

Da necessidade de cada vez mais as empresas pretenderem racionalizar os seus recursos, o **Autonomic Computing**, e tudo o que lhe é inerente, aparece como veículo operativo solucionador de problemas, evitando, dessa forma, um aumento de custos nas mesmas.

Por outro lado, nas sociedades modernas a simplificação e funcionalidade dos contextos informáticos, é uma constante preocupação, ao que o conceito de **Autonomic Computing** não é de todo alheio, muito pelo contrário.

Qualquer desfio informático é sempre motivador para quem fez o seu percurso académico na área, e, nesta matéria, a criação de uma API em âmbito laboral, foi uma motivação acrescida.

A API desenvolvida teve de considerar as aplicações da Empresa I2S, suas necessidades e restrições arquitectónicas, nomeadamente não influenciar o tempo da execução principal em mais de 50 milisegundos, para atingir o nível de sucesso operativo pretendido.

De forma a ser possível solucionar os diversos eventuais problemas, esta API regista eventos num formato que, para além da possibilidade de ser legível pelo homem, contenha informação suficiente e eficaz para depois os encaminhar, quer local quer remotamente, para consumidores de informação que tratarão de os interpretar, processar e analisar.

A implementação deste projecto foi concretizada, contudo, com a consciência de que poderá ainda de ser reajustada e melhorada.

Abstract

The theme **Autonomic Computing** is the core of the present document, specifying a kind of sub-theme, "**Event Log**", in which was focused the whole development of this dissertation

The need for more and more companies wishing to streamline their resources, Autonomic Computing, and anything that is inherent in it, appears as an operating problem solver vehicle, thus preventing an increase in costs.

Moreover, in the modern societies the simplicity and functionality of computing contexts are a permanent concern that the concept of Autonomic Computing is not at all indifferent, quite the contrary.

Any computer challenge is always motivating for those who graduated in this area and thus, the creation of an API in the workplace was an increased motivation.

The created API had to consider the applications of the Company I2S, its needs and architectonic constraints, for example, do not influence the time of the main execution in more than 50 milliseconds in order to achieve the intended level of operative success.

In order to be able to solve several potential problems, this API records events in a format that, apart from being readable by humans, contains sufficient and efficient information to be sent, either local or remotely, to consumers of information who will try to interpret them, process and analyse.

The implementation of this project was achieved, being aware that it can always be adjusted and improved.

Agradecimentos

Todos os trabalhos resultam de um esforço conjunto de várias pessoas que, directa ou indirectamente, contribuem para a sua realização.

Ao longo do período de desenvolvimento desta dissertação, vários foram os desafios e obstáculos a ultrapassar. Contudo, aliadas a uma motivação pessoal forte, essas dificuldades só foram possíveis superar com o apoio e contribuição de várias pessoas e instituições, razão pela qual não poderia deixar de as mencionar.

Deste modo, expresso a minha profunda gratidão:

Ao Professor Rui Maranhão, orientador deste trabalho, pela disponibilidade constante demonstrada, pela forma como acarinhou este projecto, pela confiança que depositou nas minhas capacidades, factor determinante para uma motivação efectiva.

Ao ex-Director da FEUP, Carlos Albino Veiga da Costa, pela visão alargada dos benefícios práticos que advêm da concretização protocolar com Empresas, onde é possível aliar a teoria à prática. Ao Director do Curso de Mestrado Integrado de Engenharia Informática e Computação, António Augusto de Sousa, pela atitude solicita e simpatia que sempre demonstrou durante toda a minha formação na FEUP.

À Empresa I2S, na pessoa do Engenheiro Aníbal Oliveira e do Engenheiro Jorge Miranda, por facultarem a jovens estudantes a entrada no mundo do trabalho para o qual se preparam. Ao Engenheiro Paulo Bastos, o meu profundo agradecimento pela orientação e ajuda facultadas durante todo este tempo.

Ao Bruno Costa e ao Daniel Barciela, elementos da área de Concepção Inovação e Desenvolvimento da I2S, o meu reconhecimento pela total disponibilidade, pelo apoio incansável e, fundamentalmente, pelo profissionalismo demonstrados, cujo comportamento será exemplo a seguir.

Aos meus colegas, Hélder Branco, Rui Azevedo, Adão Pinto, Sérgio Brandão e Ricardo Cardoso, agradeço a entreajuda sistemática, proporcionando um verdadeiro espírito de grupo em ambiente de trabalho.

Ao Professor da disciplina Preparação da Dissertação, Francisco Restivo, pelos conselhos pertinentes e orientações ajustadas, que muito contribuíram para o sucesso deste trabalho.

À minha família, em especial aos meus pais e irmã, Mariana, que me acompanharam sempre nos bons e nos maus momentos e a quem devo o que sou, à minha namorada, Andreia, pelo carinho, apoio e paciência demonstradas.

A todos o meu muito obrigado.

Marcelo Barreira.

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

CL	Apache Commons Logging
DI	Dependency Injection
GIS	Gestão Integrada de Seguros
I2S	Informática Serviços e Seguros
IOC	Inversion of Control
IT	Information Technology (Tecnologias da Informação)
LAP	Log, Auditing and Profile
POA	Programação Orientada a Aspectos
SO	Sistema Operativo
RMI	Java Remote Method Invocation
RPG	Report Program Generator
URL	Uniform Resource Locator
XML	eXtensible Markup Language

1 Introdução

A dependência das tecnologias e a informatização de todos os processos de trabalho, já faz parte do quotidiano da nossa realidade. Onde quer que vamos a infomática está presente e tornámo-nos dependentes dela. Contudo, e apesar de todos os processos de controlo a que um software é sujeito, a quantidade de eventuais falhas que um sistema informático pode apresentar é suficientemente significativa para despertar o um alerta, e contribuir para a nossa insegurança perante o seu uso. A possibilidade de um sistema detectar falhas, configurar-se e corrigir essa mesma falha, confere uma maior confiança aos utilizadores desse sistema. Para que essa detecção e recuperação sejam possivéis, é necessário que o sistema permita e produza informação suficiente relativa ao seu ciclo de vida, de modo a que esta seja consumida e processada posteriormente com o objectivo de restabelecer o sistema da falha em causa. O sistema deverá ser uma fonte de dados com o conteúdo suficiente para que a recuperação seja exequível, mesmo sendo em aplicações distribuídas de forma aleatória e espacialmente distante.

1.1 Contexto/Enquadramento

A crescente necessidade de continuidade dos negócios 24 horas por dia obriga a que os sistemas de informação que os suportam tenham um nível de fiabilidade, estabilidade e robustez elevados. Para que estas qualidades sejam garantidas é necessário introduzir novos modelos de monitorização das aplicações. A monitorização de soluções empresariais complexas deve ser efectuada de acordo com novos paradigmas, em que as aplicações sejam mais pró-activas e capazes de prevêr e resolver os problemas que vão surgindo.

Este documento surge no âmbito da disciplina de Dissertação leccionada no segundo semestre do quinto ano do Mestrado Integrado de Engenharia Informática e Computação na Faculdade de Engenharia da Universidade do Porto. O desenvolvimento do projecto foi realizado em conjunto com a empresa I2S e consistiu na análise, projecção e desenvolvimento de uma aplicação, adaptada à realidade da sua solução GIS e outras, possibilitando a recuperação automática no caso de ocorrência de eventos anómalos (erros).

1.2 Projecto

O objectivo principal é criar um sistema de apoio à determinação, resolução ou antecipação de problemas, que possam acontecer no uso das aplicações da I2S, num prisma de suporte por parte das equipas IT. Este sistema será composto por três sub-sistemas distribuídos, que trabalharão em conjunto para fornecer uma solução integrada, o I2S.LAP. Este projecto, I2S.LAPLocalAgent, é um sub-sistema desse mesmo sistema. O I2S.LAPLocalAgent será um agente, local ou remoto, responsável pelo registo de eventos técnicos (erros de software) e de negócio, que forem despoletados pelas aplicações da I2S, registando informação em tempo real. Será também da sua responsabilidade a ligação com os outros dois subsistemas do I2S.LAP (I2S.LAP Services e I2S.PD). Pretende-se que forneça as seguintes funcionalidades:

• Registo de eventos para os componentes das aplicações da I2S, tratando da comunicação entre o servidor onde está instalado e o I2S.LAP Service, efectuando também o registo local dos eventos.

• Entrega dos eventos registados caso o I2S.LAP Service não esteja disponível.

• Recolha de informação do ambiente de execução e das aplicações da I2S.

• Recepção e execução de instruções remotas a partir do I2S.PDA ou do I2S.LAP Services.

• Implementação de politicas de seguranças, limitando ou autorizando as acções remotas e informações que um determinado agente pode recolher, fornecer ou executar.

1.3 Motivação e Objectivos

A motivação aparece naturalmente desde que o caminho traçado seja, de facto, aquele que se pretende percorrer.

É certo que nem sempre o caminho segue em linha recta, aparecem, inúmeras vezes, encruzilhadas inesperadas (obstáculos, dificuldades….), porém há que reconhecer que o ambiente, o entusiasmo, o saber estar e o saber fazer dos colegas e professores da FEUP, foram uma ajuda preciosa no contorno dessas encruzilhadas.

A possibilidade de estagiar num efectivo contexto laboral, considerando ainda a imagem da I2S no mercado de trabalho, empresa com enormes responsabilidades na área das tecnologias informáticas, foi um factor aliciante e motivador.

Por outro lado, a perspectiva de aplicar a teoria na prática na execução de um projecto, cujo resultado não seria mero exercício prático de retórica, mas, pelo contrário, uma mais valia para as pretensões da empresa, foi duplamente motivador.

Desta forma, o primeiro e grande objectivo, desafiador de capacidades cognitivas, foi o de poder abraçar uma situação “abstracta”, estudá-la, analisá-la e incrementar os mecanismos necessários para a concretizar de forma eficiente.

Como objectivos gerais, de realçar o desenvolvimento de valores como:

- Equidade, integridade e responsabilidade
- Competência, qualidade e excelência
- Inovação, criatividade e empreendedorismo

No âmbito de objectivos mais específicos, destaca-se o desenvolvimento de um módulo que possibilitará, aquando da ocorrência de eventos técnicos (erros de software) ou de negócio, em conjunto com outros módulos, aplicar os conceitos do Autonomic Computing e encontrar soluções para as aplicações da I2S.

1.4 Estrutura da Dissertação

Este documento é composto por 5 capítulos. No primeiro é apresentada uma introdução desta dissertação, o contexto em que ela surge e a motivação que levou a que esta fosse realizada. No segundo capítulo é descrito todo a estudo inerente ao projecto, bem como os principais aspectos tecnológicos envolvidos. No terceiro capítulo é exposto o problema e uma análise da solução encontrada. No quarto é possível verificar toda a implementação inerente a este projecto. Por último, mas não menos importante, no quinto capítulo, são apresentadas todas as conclusões que advêm do desenvolvimento desta dissertação, e as possíveis melhorias futuras a que o projecto se pode/deve submeter.

2 Revisão Bibliográfica

Neste capítulo é descrito o estado da arte e apresentados os aspectos tecnológicos inerentes ao projecto descrito neste documento.

2.1 Autonomic Computing

Com a evolução dos sistemas informáticos, nomeadamente de sistemas distribuídos e heterogéneos, o risco de ocorrência de erros ao nível da instalação dos mesmos, de falhas de hardware ou software, e até mesmo de equívocos humanos, implica um aumenta das dificuldades administratrivas e de manutenção dos mesmos. Os custos inerentes a isto, em sistemas cada vez mais complexos, preocupam cada vez mais as empresas que desenvolvem este tipo de software, uma vez que isto requer que profissionais experientes e bem qualificados de IT se dediquem mais a esta ocupação do que a tarefas mais importantes para a área de negócio.

Autonomic Computing é um novo conceito que se baseia na ideia da tecnologia se gerir a si mesma, permitindo assim aos sistemas detectar, corrigir e prevenir eventuais falhas, diminuindo assim a necessidade de intervenção humana, delegando para os próprios sistemas, baseado em políticas estruturais de cada empresa, toda a gestão. [ACIBM]

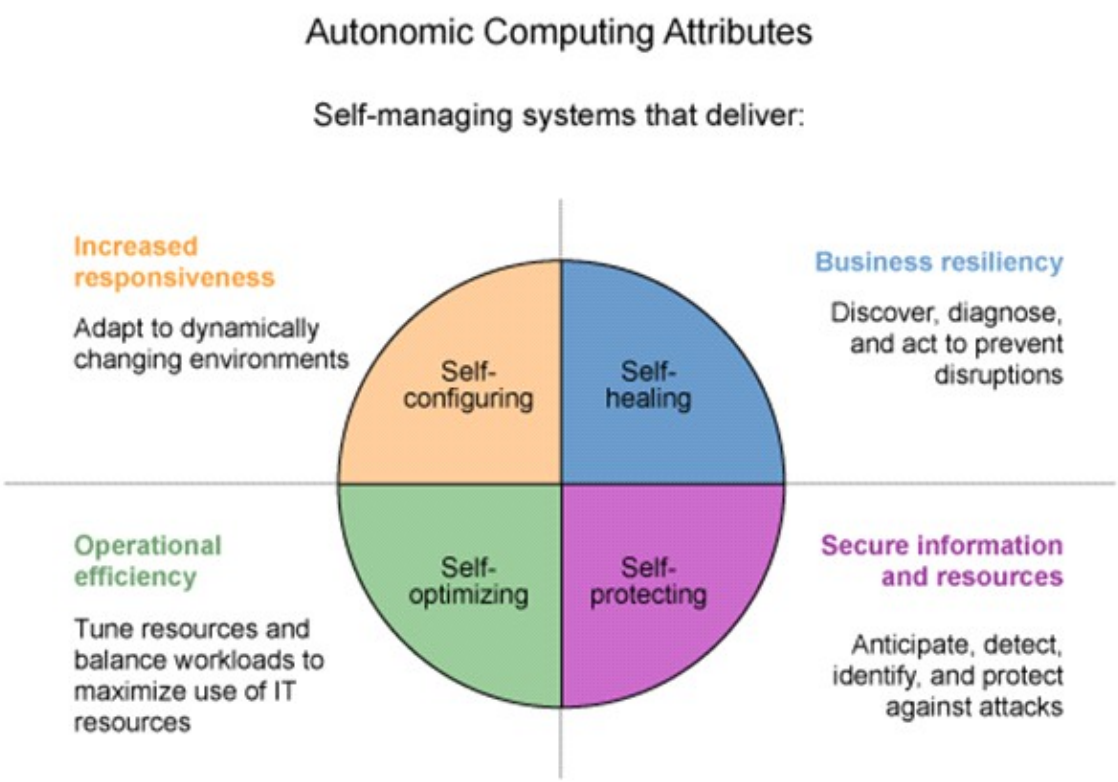

Figura 2.1: Área relativas ao conceito Autonomic Computing.

O Autonomic Computing é composto por 4 áreas distintas, mas caminham todas para o mesmo objecto.

∆ **Auto-configuração**

Esta área define a habilidade que um sistema deverá ter ao nível de alterar as suas próprias configurações em função possíveis das alterações das variáveis que definem o contexto onde este se encontra, de um modo dinâmico. De modo, as empresas conseguem garantir uma maior flexibilidade e independência de todos os seus sistemas. [ACA]

∆ **Auto-correcção**

A capacidade de detectar, analisar e corrigir uma eventual falha do sistema, confere uma maior adaptabilidade do mesmo e uma probabilidade de falha menor. [ACA]

∆ **Auto-optimização**

Em sistema complexos e de grande robustez, todas as potencialidade das máquinas onde estes executam são, na maior parte dos casos, desperdiçadas devido ao enorme processamento

exigido. A possibilidade de auto-optimização por parte de um sistema, garantindo uma aumento do seu desempenho, é visto como uma enorme potencialidade nas IT.[ACA]

∆ **Auto-protecção**

Esta área idealiza a possibilidade de um sistema antecipar, detectar e precaver os potenciais comportamentos hosteis a que este pode ser sujeito (por exemplo: vírus). Esta capacidade transmite ao utilizador uma maior confiança no uso do sistema.[ACA]

2.2 Programação Orientada a Aspectos

A Programação Orientada a Aspectos (POA), conceito criado por Gregor Kiczales, é um novo paradigma da programação que vem, não para substituir os outros paradigmas (Programação Orientada a Objectos por exemplo), mas sim com objectivo de complementar o desenvolvimento de software cada vez mais modelar e que abranja todos os interesses (*concerns*) transversais ao próprio sistema e lógica de negócio. Através da POA é possível verificar uma diminuição do código disperso no desenvolvimento de software bem como um aumento da facilidade de implementação e manutenção do mesmo [AOP97,APIBM05].

Nos últimos anos é de notar o aparecimento de várias ferramentas de POA disponíveis. Em Novembro de 2004 e através de dados disponibilizados pela Comunidade de Desenvolvimento de Software Orientado a Aspectos e Conferências (AOSD) foi possível verificar a preferências que os utilizadores dão às ferramentas AspectJ, AspectWerkz, JBoss AOP e String AOP, através da Figura 2.2, que indica o número de *posts* dos utilizadores relativamente às mesmas.

project	posts	list	url
AspectJ	150..210 each	aspectj-users at eclipse.org	eclipse.org/aspectj
AspectWerkz		user at aspectwerkz.codehaus.org	aspectwerkz.codehaus.org
JBoss AOP		aspects/jboss forum	jboss.org/products/aop
Spring AOP		springframework-user at SourceForge	www.springframework.org
abc	1..30	abc-users at comlab.ox.ac.uk	abc.comlab.ox.ac.uk
aspect#		aspectsharp-users at SourceForge	aspectsharp.sourceforge.net
AspectC++		aspectc-user at aspect.org	aspectc.org
JAC		jac-users at objectweb.org	jac.objectweb.org

Figura 2.2: Número de posts de utilizadores de ferramentas de POA, Novembro 2004.

É possível constatar a grande adopção por parte dos utilizadores às primeiras ferramentas, possivelmente demonstrando que estas são mais *userfriendly* relativamente às restantes.

	aspect declarations	inter-type declarations	advice bodies	pointcuts	static enforcement	configuration
AspectJ	code				declare error/warning	.lst inclusion list
AspectWerkz	annotation or xml		java method	string value	-	aop.xml
JBoss AOP						jboss-aop.xml
Spring AOP	xml					springconfig.xml

Figura 2.3: Comparação de sintaxe entre as principais ferramentas de POA.

Analisando a Figura 2.3, podemos verificar que apenas o projecto AspectJ utiliza código Java na codificação dos aspectos. Aspect WerKz, JBoss AOP e Spring AOP, utilização linguagem de notação, principalmente eXtensible Markup Language (XML). A possível alternação repetitiva, aquando da codificação dos aspectos, entre ficheiros XML e ficheiro Java, é uma das principais desvantagens destes últimos projectos. Contudo alguns programadores preferem a forma declarativa do XML em comparação às várias extensões da linguagem Java. De salientar que o AspectJ é o único que suporta *static enforcement*. A familiarização do AspectJ com Java leva a que o programador escreva menos e realize menos erros. Outro ponto a favor do AspectJ é que os pointcuts são considerados como classes de 1ª ordem, o que facilita o manuseamento dos mesmos.

	join point kinds and kinded pointcuts				kindless pointcuts		
	invocation	initialization	access	exception handling	control flow	containment	condi-tional
AspectJ	{method, constructor, advice} x {call, execution}	instance, static, pre-init	field get/set	handler	cflow, cflowbelow	within, withincode	if
Aspect Werkz		instance, static				within, withincode, has method/field	-
JBoss AOP		instance		(via advice)	(via specified call stack)	within, withincode, has method/ field, all	(via Dynamic CFlow)
Spring AOP	method execution	-	-	(via advice)	cflow	-	custom pointcut

Figura 2.4: Expressividade versus Simplicidade.

Numa breve análise à Figura 2.4 onde é evidenciado como é que cada ferramenta combina os *join points* com os *point cuts*, é facilmente perceptível uma maior simplicidade com o uso da ferramenta Spring AOP, uma vez que este deliberadamente limita a expressividade do seu modelo. Contudo esta limitação pode ser negativa, uma vez que na definição de alguns aspectos, implica uma certa incapacidade de expressar o que realmente o aspecto manifesta. É possível verificar uma grande proximidade entre o AspectJ e o AspectWerkz pelo que não é de estranhar que mais recentemente ocorreu a fusão destes dois projectos, originando o AspectJ5.

A nível de desempenho, o AspectJ sujeita-se a uma compilação elevada em termos de tempo e memória necessária. Por outro lado, traz benefícios durante a execução, devido ao facto de necessitar de menos esforço quando combina os pointcuts. O JBoss AOP e o Spring AOP, e contrastando com o AspectJ, levam mais tempo na execução e, por consequência, usufruem de tempos de compilação menores.

	ide	editor	views	debugger	other	libraries	docs
AspectJ	eclipse, jdeveloper, jbuilder, netbeans	highlighting, content assist, advice links	outline, visualizer, cross references	plain Java	ajdoc, ajbrowser	-	++++
Aspect Werkz	eclipse	advice links	-		-	-	+++
JBoss AOP		advice links, UI for pointcut creation	aspect manager, advised members		dynamic deployment UI, jboss framework integration	++++	++
Spring AOP		-			spring framework integration	+++	+

Figura 2.5: Ambientes de desenvolvimento, bibliotecas, documentos.

O AspectJ é, de todas as ferramentas, aquela que possuí uma maior relevância relativamente ao número de ambientes de desenvolvimento integrados compatíveis. Esta é também acompanhada pelo ajdoc, um utilitário do AspectJ que gera documentação do tipo JavaDoc.

Os pontos mais fortes das ferramentas são:

AspectJ – Documentação extensa, integração avançado em ambientes de desenvolvimento e declarações de aspectos concisas bem como verificações estáticas para os *pointcuts*.

Aspect Werkz – Mecanismo similar ao AspectJ mas usando linguagem de anotação e bom suporte para implementação dos aspectos.

JBoss AOP – Boa qualidade e quantidade nas bibliotecas que apresenta, suporte para ambiente de desenvolvimento e suporte para implementação dinâmica de aspectos.

Spring AOP – Fácil de usar e aprender, portabilidade das bibliotecas de aspectos através de servidores de aplicação e integração na *Spring Framework* aumenta a facilidade de adopção para utilizadores da mesma.

Os pontos mais fracos das ferramentas são:

AspectJ – Linguagem de extensão requer compilador e ferramentas associadas e poucas bibliotecas disponíveis.

Aspect Werkz – Declarações de aspectos e *pointcuts* pouco concisas, poucas verificações estáticas bem como poucas bibliotecas disponíveis.

JBoss AOP – Funcionalidades como refactoring no ambientes de desenvolvimento não são suportadas e poucas verificações estáticas.

Spring AOP – Pouco suporte a nível de ambiente de desenvolvimento para codificar aspectos e não é adequado para codificar aspectos refinados.

Com a fusão entre o AspectJ e o Aspect Werkz, como já referido anteriormente, a codificação dos aspectos em XML passa a ser suportada, o que torna o AspectJ5 uma ferramenta poderosa trazendo melhorias ao nível do desempenho de compilação e com capacidade de satisfazer ambas as preferências relativamente à codificação de aspectos. [AOPIBM]

2.3 Event Driven Architecture

Event-Driven Architecture (EDA) é um padrão de arquitectura de software, que promove a produção, detecção, consumo e reacção a eventos. Um evento, fundamentalmente, é uma indicação de uma ocorrência, representando genericamente mudanças de estado de um determinado recurso (exemplo: ocorrência de uma falha). Tipicamente num sistema EDA, existem dois tipos de intervenientes: os emissores de eventos e os consumidores de eventos. Sempre que ocorre alguma anomalia ao sistema, os emissores geram eventos que serão direccionados para os consumidores que tentam reagir rapidamente aos mesmos. A EDA é o suporte de comunicação de aplicações direccionadas a eventos. Num processo de negócio, um evento pode representar uma situação de negócio anómala (exemplo: caso de uma pessoa tentar levantar mais dinheiro do que o que realmente possuí na conta). [EDA]

2.3.1 CommonBaseEvent

Uma das tecnologias existente ao nível da EDA é o Common Base Event (CBE). O CBE é um formato que define a estrutura de um evento de uma forma comum e coerente. Este tem como objectivo facilitar a comunicação entre componentes representando as mensagens através de eventos, sendo que a coerência destas mensagens é definida através de um XML Schema. Eventos como o logging podem ser mapeados através do CBE. A comunicação entre componentes pode ser síncrona ou assíncrona. As propriedades definidas no modelo CBE suportam a maior parte dos elementos relevantes num evento, como a identificação do componente que está a reportar a situação, a identificação do componente que é afectado pela situação, bem como uma descrição da situação em si. Informação adicional também é suportada como identificação de eventos, identificação de componentes, entre outros. Esta capacidade de armazenamento de informação adicional permite que os eventos sejam trocados e interpretados

de uma forma determinista e apropriada, em toda a gestão de múltiplos sistemas e de negócios que consomem eventos, sem nunca perder a fidelidade.

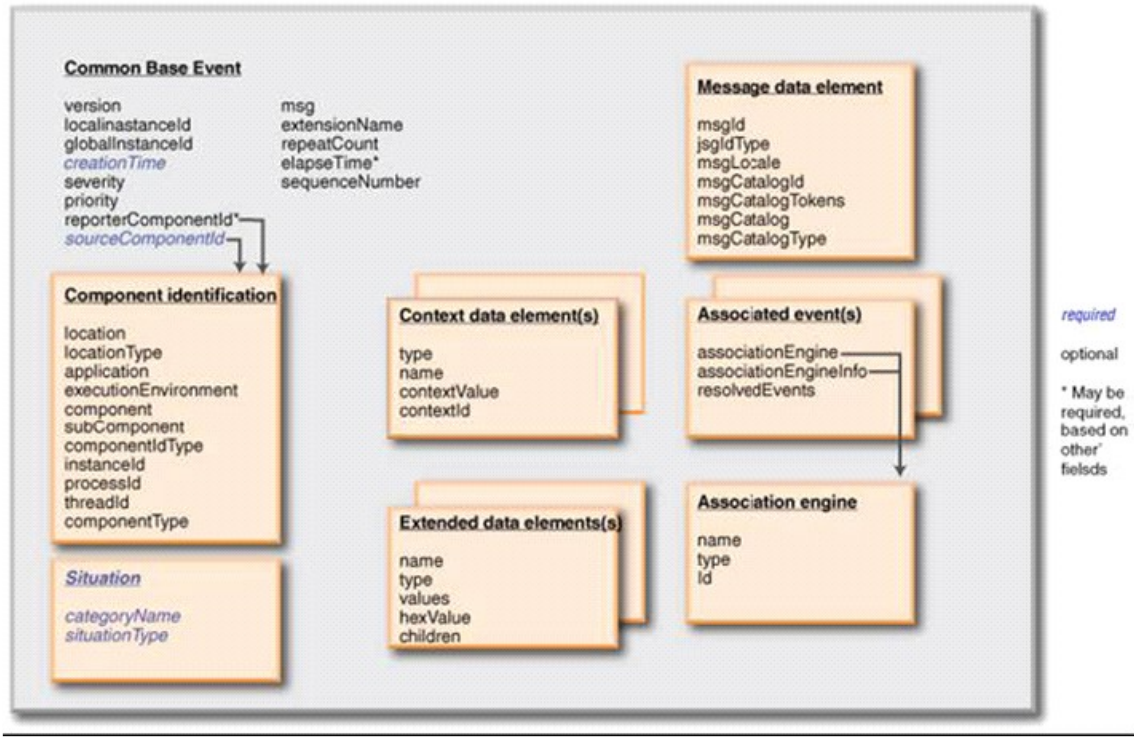

Figura 2.6: Estrutura do Common Base Event.

2.4 Apache Commons Logging

Em todo o software desenvolvido actualmente, é deveras importante controlar a lógica da aplicação, pois no caso de alguma anomalia seja descoberta, não menos importante que a sua resolução, é a sua compreensão, isto é, saber onde se deu e o porquê. O *logging* de algo em informática, não é nada mais nada menos que reportar, de forma compreensiva e legível para o ser humano, o comportamento do software. São já bastantes as implementações de sistemas de logging conhecidas. Contudo, ao utilizarmos uma delas, ficamos presos a estas, não sendo possível alterar a forma de *logging* de uma forma simples e sem necessidade de compilação.

O Apache Commons Logging (CL) é um *wrapper* que permite a possibilidade de variar entre vários sistemas de logging. Através de um ficheiro de propriedades é possível especificar que implementação de *logging* se pretende tornando a escolha do sistema completamente independente da aplicação que o está a utilizar. [ACL]

2.4.1 Log4J

O Log4J é um projecto de código aberto que disponibiliza um sistema de *logging* que permite ao utilizador a sua configuração através de um ficheiro de propriedades, sendo possível

utilizá-lo no CL. Este sistema permite que em várias classes de um projecto, o tratamento dos *logs* possa ser feito de maneira diferente, uma vez que no ficheiro de propriedades é possível configurar de uma forma hierárquica todas as classes de um projecto e o uso dos *appender* nas mesmas. O Log4J traz já consigo vários tipos de *appenders*, permitindo lançar *logs* para a consola, bem como para ficheiros, etc. Vários tipos de *layout* para os logs também podem ser configurados e usados, de forma a formatar a informação que queremos visualizar. O Log4J é composto por vários níveis de *logging*, que dependendo da configuração que se encontra no ficheiro de propriedades, podem ser ou não despoletados.

Todo o processo de detecção e registo do *log* no Log4J é executado de uma forma bastante rápida, demorando apenas 5 nanosegundos a verificar se o *log* deve ser lançado ou não, e demonstrasse bastante eficiente quando o *log* é lançado. [L4J]

2.4.2 JSR-47

O sistema de *logging* JSR-47 utiliza e é especificado por uma API, java.util.logging. Tal como o Log4J, este sistema permite a sua configuração através de um ficheiro de propriedades, diponibiliza um vasto número de níveis de *logging* e um bom leque de *handlers* (similar aos *appenders* no Log4J). Também neste sistema podemos personalizar os nossos *logs* através dos *formatters* (similar aos *layouts* no Log4J) disponibilizados. O conceito de herança entre os handlers também é aqui encontrado e definido no ficheiro de configurações. A principal vantagem deste sistema, é ser parte integrante da API do java.

2.5 Aspectos Tecnológicos

2.5.1 JAVA

Java é uma linguagem de programação orientada a objectos desenvolvida pela Sun Microsystems, e actualmente pertencente à Oracle. Para compilar e executar aplicações Java, é necessário a existência da Máquina Virtual Java (Java Virtual Machine) e um conjunto de bibliotecas que disponibilizam todos os serviços necessários a essa aplicação.

Pontos fortes:

Tecnologia multi-plataforma e portátil, devido à JVM;

Linguagem robusta, segura e de alto nível;

Existência de várias bibliotecas para Java: de XML (JAXP), de comunicação com bases de dados (JDBC) e desenvolvimento de interfaces (SWING);

Independência de um servidor para o seu funcionamento (standalone);

Existem várias plataformas Java criadas para segmentos específicos de aplicações. As plataformas mais conhecidas são:

Java SE (Java Platform, Standard Edition) – É considerada a plataforma base, uma vez que apenas inclui a JVM e as bibliotecas comuns.

Java EE (Java Platform, Enterprise Edition) – Esta edição é principalmente virada para o desenvolvimento de aplicações corporativas e aplicações Web.

Java ME (Java Platform, Micro Edition) - Esta plataforma direcciona-se ao desenvolvimento de aplicações para dispositivos móveis. [JAVA]

2.5.2 XML

eXtensible Markup Language é uma linguagem de anotação que foi concebido com o objectivo de guardar e transportar dados. Esta tem como características principais:

A possibilidade de se formar documentos minimamente estruturados de forma hierárquica.

Capacidade de se definir a sintaxe do documento, sendo que essa sintaxe é também definida pelo utilizador através de um ficheiro DTD.

Possibilidade de visualização do documento em browser, através da utilização do XSLT e aplicando as transformações ao documento.

Simplicidade e legibilidade por parte de quem lê o documento.

Possibilidade de comunicação com servidor.

Esta tecnologia é recomendada pela World Wide Web Consortium (W3C). [XML]

2.5.3 Apache Ant

O Ant é uma ferramenta de geração automática de código, baseada em Java. Toda a configuração para a geração é feita em ficheiros XML, onde se definem as tarefas a serem executadas. É possível criar “fachadas” utilizando Ant, através da especificação de uma sequência de tarefas e de possíveis dependências que possam existir, para sistemas multi-plataforma, facilitando a nível de tempo e confiança no código por parte de quem o desenvolve. Outra grande vantagem é também não depender do sistema operativo onde está a ser executado para a geração do código. [ANT]

2.5.4 Remote Method Interface

O Java Remote Method Invocation é um mecanisno de Java que permite, tal como o próprio nome indica, a invocação de métodos remotos por parte de objectos localizados em máquinas virtuais diferentes. Ao contrário de outros sistemas, que apenas cedem a possibilidade de se puder passar como argumento ou retornar dos métodos apenas determinados tipos de estruturas, o RMI permite que qualquer tipo de objecto Java seja usado no processo de chamada aos métodos.

Aplicações RMI normalmente são compostas por dois programas, um servidor e um cliente. O servidor trata de criar os objectos remotos, fazendo com que estes sejam acessíveis através de referências também criadas. Este disponibiliza uma *interface* com os serviços a que o cliente pode aceder e utilizar.Um cliente tipíco obtém essas mesmas referências e invoca os métodos que estão disponíveis a esse objecto. Este é o tipo de processo que normalmente existe numa aplicação distribuída.

Um determinado objecto para ser remoto ao nível do Java, necessita que a sua classe implemente a interface *Remote*, sendo que todos os métodos a serem invocados remotamente devem lançar a excepção *java.rmi.RemoteException*, para que o cliente consiga tratar qualquer problema relacionado com a comunicação. [JRMI]

2.5.5 Spring Framework

O Spring é uma *framework* Java/J2EE que disponibiliza um vasto número de módulos que podem ser utilizados de acordo com a orientação do projecto (módulos para Web, para POA, etc.), tornando mais simples o desenvolvimento de aplicações e promovendo boas práticas de programação. A sua arquitectura por camadas permite o uso dos módulos de uma forma complemente independente, o que aumenta a flexibilidade desta *framework.*

No Spring existe um *container* que é responsável por assumir todo o controlo do sistema (*inversion of control pattern*), estando a seu encargo a instanciação dos objectos, bem como é o responsável por preparar todas as dependências necessárias que se encontram configuradas em ficheiros XML (*dependency injection pattern)*. Através destes ficheiros de configurações XML, é possível retirar todas as vantagens da reutilização de código, herança e polimorfismo, uma vez que a instanciação dos atributos dos vários objectos podem ser definidos e alterados nestes ficheiros, sem necessidades de alteração directa no código de instanciação dos mesmos. Estes conjuntos de classes e propriedades definidas em XML têm como nome *application context*'s. A configuração de um objecto no ficheiro de propriedades denomina-se de *bean*, e possui um identificador único no XML, através do qual se pode obter a instância do objecto na codificação. [SPRG]

2.5.6 Eclipse Test and Performance Tools Platform Project (TPTP)

Este projecto providencia uma plataforma livre, constituída por framewoks e serviços poderosos, que permite o desenvolvimento de testes únicos e ferramentas de desempenho sendo facilmente esta plataforma integrada no Eclipse.

TPTP engloba todo o ciclo de vida de teste e de desempenho no desenvolvimento de uma aplicação, desde os primeiros testes até ao controlo da aplicação de produção, incluindo a edição de testes e execução dos mesmos, monitorização, detecção e capacidades de análise de logs. A plataforma suporta uma ampla gama de sistemas de computação e continuará a expandir o suporte para abranger o maior número possível de sistemas. TPTP também permite o aparecimento de modelos de ferramentas completamente novos, que simultaneamente impulsionam e integram dados através de todo o ciclo de vida produzindo recursos avançados e aumentando a produtividade. [TPTP]

2.6 Sumário

A utilização destes novos paradigmas, nomeadamente o uso de POA e de uma arquitectura orientada ao evento, poderá significar um avanço tecnológico significativo nos conceitos até hoje usados em engenharia de software.

A possibilidade de introduzir novas instruções de processamento em diferentes zonas de codificação de um software, baseadas em aspectos, poderá ser uma ferramenta de extrema usabilidade e diminuirá sem dúvida os custos temporais nesse sentido.

A utilização de formatos orientados à definição e descrição de eventos, para posterior análise por parte de sistemas capazes de detectar e corrigir eventuais erros das aplicações existentes, nomeadamente em áreas de negócio críticas, demonstra uma enorme potencialidade e fidelidade dos sistemas futuros. A própria prevenção dos mesmos, possibilitará aos técnicos das IT uma maior disponibilidade no desempenho de outras tarefas, não menosprezando a importância de responsabilidade destes no desenvolver software, segundo todas as boas práticas e padrões já existentes.

3 O Problema

3.1 Descrição do problema

Todas as aplicações da I2S registam *logs* para que, na eventualidade de acontecer alguma anomalia, seja possível detectar o erro manualmente. Contudo, dado que normalmente estes *logs* são bastante extensos, factor que dificulta a análise dos mesmos, a automatização de todo este sistema é um dos principais objectivos da I2S. Assim, impõe-se a obrigatoriedade de registar eventos, quer técnicos quer de negócio, das aplicações da I2S, com informação suficiente para que o problema seja detectado por uma estrutura completamente independente, o I2S.LAPService, e de forma automática. O I2S.LAPLogAgent será então o responsável por toda a comunicação dos eventos com o I2S.LAPService, sendo que este último poderá ter a necessidade de obter mais informação, através da interacção com o I2S.LAPLogAgent, relativa ao problema em questão.

O I2S.LAPLogAgent deverá persistir todos os eventos despoletados pela aplicação de forma a garantir que não se perca informação, perante a ocorrência de algum problema ao nível da sua execução. Caso esta situação se verifique, este será o responsável pelo envio de todos os eventos guardados, isto é, deverá ser o I2S.LAPLogAgent a gerir o envio dos eventos, e, concretamente, certificar-se que nenhuma informação é perdida em todo este processo. O I2S.LAPLogAgent deverá ser completamente independente da aplicação, garantindo que nenhuma informação relativa à comunicação com o I2S.LAPService esteja disponível no código do cliente (por exemplo, informações relativas à possível comunicação remota). Deverá também ser completamente independente do I2S.LAPService, ou seja, ambos os módulos devem permitir que a sua execução aconteça em máquinas separadas e diferentes. Na possibilidade de o I2S.LAPService se encontrar *offline*, o I2S.LAPLogAgent terá de manter todo o seu comportamento normal. Nesta situação e caso o I2S.LAPService retome a sua execução, o I2S.LAPLogAgent deverá retomar a comunicação e o envio dos eventos, dos já persistidos bem como dos eventos em tempo real.

Todas as peças que compõe o I2S.LAPLogAgent, deverão ser configuráveis, como por exemplo o local da persistência, informações relativas à comunicação, entre outros. É requisito essencial que em todo este processo a aplicação principal não seja prejudicada, isto é, o processo de chamada a esta API não deverá ser superior a 50 milisegundos desde o acontecimento do evento até ao retomar da sua execução normal. Toda a comunicação com o I2S.LAPService terá de ser segura, garantindo integridade e confidencialidade das informações que se encontram no evento. É importante também salientar, que as principais aplicações da I2S executam em JAVA1.4, pelo que o I2S.LAPLogAgent obriga a uma compatibilidade com esta mesma versão, mas também com a migração dessas mesmas aplicações para JAVA1.5.

3.2 Solução e problemas encontrados

Inicialmente, e com o objectivo de em certas zonas do código da aplicação ser necessário obter os dados de execução, pensou-se na utilização da POA. Através deste processo seria possível injectar linhas de código na aplicação principal em zonas específicas, de modo a permitir accionar o processo de execução do I2S.LAPLogAgent. Contudo, este processo poderia ter implicações ao nível da estabilidade do GIS e de outras aplicações, bem como seria necessário que todas as aplicações dos clientes da I2S,utilizadoreas desta API, fossem recompiladas. Numa análise mais aprofundada à estrutura do GIS e de outras aplicações da I2S, foi possível verificar que todos os *logs* registados eram accionados pelo CL. Assim, e uma vez que as boas práticas da I2S levam a que todas as suas aplicações tenham o cuidado de registar *logs* de todos os processos de execução, optou-se por aproveitar o CL e criar *appenders/handlers* para a utilização do Log4J e do JSR47 (ferramentas de *logging* utilizadas pela I2S no CL). Sendo o CL bem como estas duas ferramentas de *logging* configurados por ficheiros de propriedades externos, não seria preciso qualquer tipo de alteração do código da I2S, nem necessidade de novas compilações das aplicações. Assim, e utilizando os *appenders/handlers*, é possível mapear os evento no formato CBE, de forma a guardar toda a informação relevante e garantindo a integridade da mesma, pois todos os dados são "apanhados" em tempo real. O campo *threadId* do CBE é essencial para o processamento do I2S.LAPService. Porém, a noção de *threadId* em JAVA1.4 não existe. A solução encontrada foi a criação de um utilitário que sempre que fosse necessário mapear a primeira vez um CBE, este por *reflexion* determina a existência do método *getId*, da class Thread da API do JAVA. É garantido que esta verificação só é feita uma vez, devido às implicações a nível de performance que a utilização de *reflexion* implica. Caso não exista este método, é porque nos encontramos num ambiente JAVA1.4 e nesta situação esse mesmo campo do CBE é preenchido com o nome da *thread* em que o evento aconteceu. Contrariamente, e sendo o ambiente de execução JAVA1.5, o campo é preenchido com o *threadId* correspondente.

Relativamente ao envio dos CBE's para o I2S.LAPLogAgent, numa primeira fase utilizou-se comunicação síncrona. A necessidade do I2S.LAPLogAgent ser independente, levou à criação do mesmo como sendo um serviço (*service provider*). A comunicação entre a aplicação e o serviço será feita através de um proxy que apenas executa o *routing* dos eventos para o

I2S.LAPLogAgent. Este proxy acede aos métodos do I2S.LAPLogAgent através de um RMI *stub* criado, utilizando codificação em *ant*, de forma a executar as funcionalidades que este serviço disponibiliza.

Numa segunda fase, a comunicação deverá ser possível configurar para que seja feita assincronamente, utilizando para isso mecanismos de *threads*.

Relativamente às configurações das propriedades do I2S.LAPLogAgent, foi proposto inicialmente a utilização do Spring, utilizando os seus ficheiros de configurações. No entanto, estas configurações do Spring, que possibilitam a alteração do próprio modo de funcionamento do I2S.LAPLogAgent através dos ficheiros de configurações, apenas se encontram para configurar o serviço. Todas as outras peças são configuradas através de ficheiros de propriedades do *Java Properties*. Esta decisão deve-se ao facto do Spring também usar internamente o CL, consequentemente poderíamos estar perante possíveis fontes de problemas e provocar ciclos infinitos, caso utilizássemos Spring dentro do *appender/handler* ou das peças directamente ligadas a este. Assim, com o uso de ficheiro de propriedades e através do uso de *reflexion*, é possível simular o comportamento do Spring, não perdendo as suas vantagens, nem o controlo sobre eventuais falhas internas.

4 Implementação

Este capítulo apresenta todos os detalhes de implementação dos componentes relativos ao projecto. O projecto divide-se em duas partes: a lógica que se encontra do lado das aplicações que usam a API, e portanto do lado dos clientes, e o serviço I2S.LAPLogAgent. A implementação de todas as peças deste projecto foi orientada ao *command pattern*, um padrão que visa o uso de *interfaces* como objectos de negócio, com as respectivas implementações a serem declaradas através de ficheiros de configurações, utilizando para isso o Spring, no caso do serviço, e ficheiro de *properties* no caso das restantes peças. Em vários casos, como será possível constatar neste capítulo, também o *factory pattern* é utilizado.

4.1 API

A API é a responsável por apanhar, mapear e guardar os eventos localmente. Decidiu-se que estes seriam guardados num sistemas de directórios em disco.Posteriormente envia-os para o serviço. Esta é composta pelos seguintes componentes: *appender / handler* e um *proxy* que apenas têm a tarefa de enviar os eventos para o I2S.LAPLogService.

4.1.1 Appender / Handler

Para o uso das ferramentas de *logging* Log4J e JSR47, foi criado um *appender/handler*. Na figura abaixo é possível analisar o comportamente deste.

Figura 4.1: Diagrama de actividade do *appender/handler.*

Inicialmente, quando o sistema arranca, o ficheiro de propriedades do sistema de *logging* em uso é lido. Para o *appender/handler* foram criadas duas propriedades de configuração que podem ser definidas nesses mesmos ficheiros.

As propriedades são:

Δ - LapAgentHelperClassName – esta propriedade define o nome da classe de implementação da fábrica, através da qual o *appender/handler* obtem uma instância da implementação do componente que fará a ligação ao serviço I2S.LAPLogAgent.

Δ - LapAgentHelperConfigURL – esta propriedade define o URL da configuração que o componente obtido através da fábrica terá. Por exemplo, no caso de a intenção ser comunicação assíncrona da API com o I2S.LAPLogAgent, é neste URL que se deve colocar essa definição.

Através da utilização destas propriedades definidas no ficheiro de propriedades, é possivel que a implementação da fábrica, da qual se obtêm a instância da componente responsável pela comunicação com o serviço, seja alterado, fazendo com que toda a estrutura seja reutilizada. A utilização de um URL para definir onde se encontram as configurações permite a independência da localização destas, isto é, podem encontrar-se localmente no sistema de ficheiros do SO ou numa localização remota.

Aquando da invocação do *logging* por parte da aplicação a ser monitorizada, o evento é então mapeado no *appender/handler*, sendo depois enviado para o componente que será responsável pelo envio remoto do evento para o serviço.

4.1.2 LAPAgentServiceProviderDelegate

O LAPAgentServiceProviderDelegate tem como objectivo único fazer o *routing* entre a API e o I2S.LAPLogAgent. Este apenas é executado no caso de a comunicação com o serviço ser efectuada de forma síncrona. Através de um *lookup*, este componente obtêm uma referência remota do serviço, sendo assim possível invocar os métodos disponibilizados por este, utilizando para isso um RMI *stub* criado pela tecnologia Ant. Caso inicialmente falhe esta procura, o LAPAgentServiceProviderDelegate tenta de novo obter essa mesma referência um número de vezes igual ao configurado pelo utilizador. Assim, é auferida a possibilidade do *appender/handler* enviar o evento para o I2S.LAPLogAgent.

Após o envio, este tem a seu cargo a responsabilidade de informar o *appender/handler* do sucesso/insucesso do envio remoto dos eventos. Caso o insucesso do envio seja causado por problemas a nível remoto, este o LAPAgentServiceProviderDelegate tentará enviar de novo o evento um número de vezes igual ao especificado nas configurações.

A figura abaixo ilustra o comportamento deste componente.

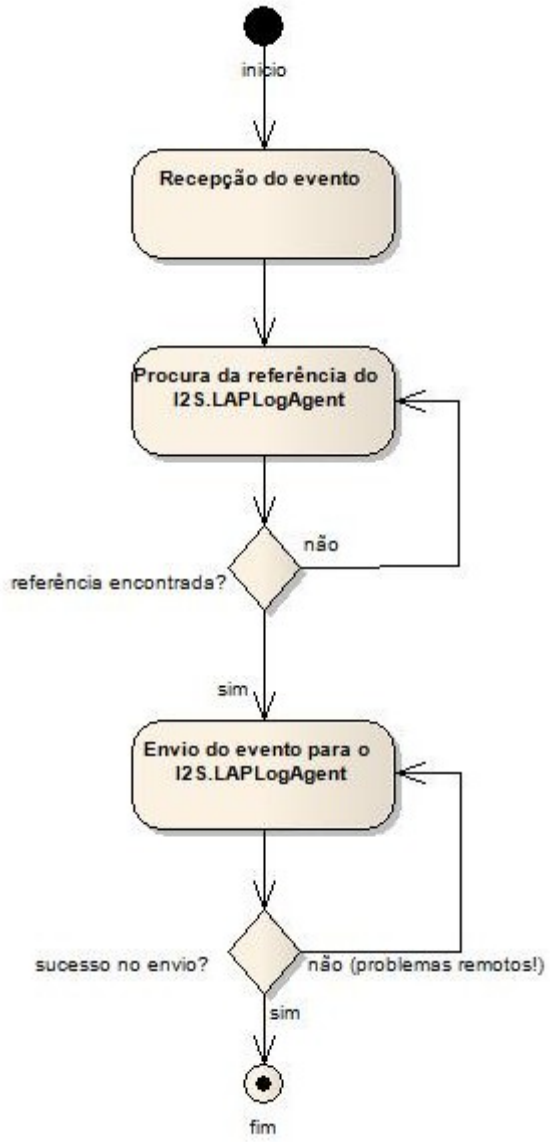

Figura 4.2: Diagrama de actividades do LAPAgentServiceProviderDelegate.

4.1.3 LAPDispatcherManager

O LAPDispatcherManager é responsável pela comunicação assíncrona entre o *appender/handler* e o I2S.LAPLogAgent. Este é uma *thread* que possuí uma *queue* interna, onde os eventos vão sendo colocados pela API e processados por ele mesmo. Este componente contém também uma *thead pool*, sendo esta disponibilizada pela *framework* Spring. O LAPDispatcherManager retira os vários eventos colocados na *queue*, e, utilizando as *threads* disponíveis que se encontram na *pool*, envia os eventos para o I2S.LAPLogAgent. Todas as peças deste componente são configuráveis, como por exemplo o número de *threads* na *pool*, o

tempo que o LAPDispatcherManager “adormece” enquanto a *queue* está vazia (com o objectivo de não existir processamento desnecessário), entre outros.

O comportamento deste componente é realçado pela figura apresentada de seguida.

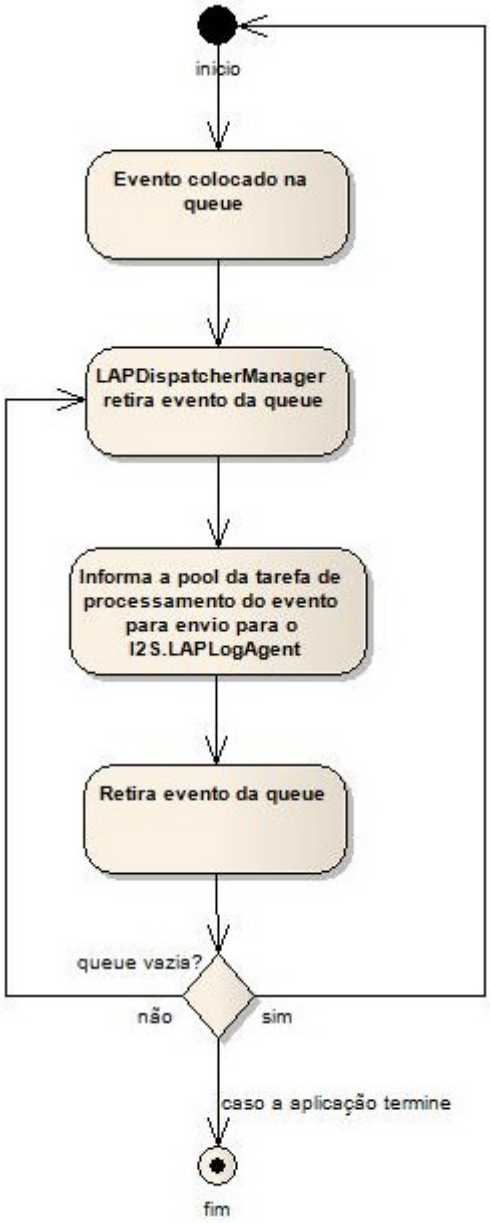

Figura 4.3: Diagrama de actividades do LAPDispatcherManager.

4.2 Serviço I2S.LAPLogAgent

O I2S.LAPLogAgent é o serviço utilizado por toda a API local, sendo este o responsável pela comunicação com o I2S.LAPService e pelo envio dos eventos para este último. É este que deve gerir os eventos que ainda não foram enviados para o I2S.LAPService e que poderão estar guardados em persistência. Devido às obrigatoriedades exigidas pela empresa, este serviço deverá no seu arranque, registar-se no *Service Manager*, uma aplicação da I2S encarregue pela gestão de todos os serviços existentes.

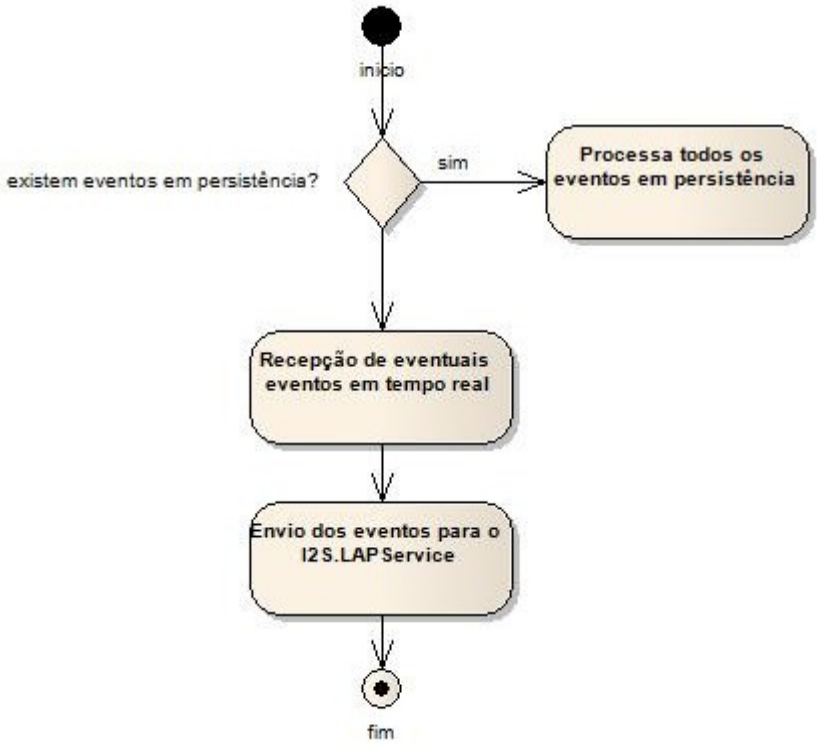

Figura 4.4: Diagrama de actividades do I2S.LAPLogAgent.

4.3 Resumo e Conclusões

Em suma, todo o sistema é capaz do pretendido e exigido pela I2S.

A possibilidade de configurar todo o funcionamento dos componentes desenvolvidos, permite uma maior flexibilidade do mesmo, sendo possível adaptá-lo em função das características da máquina onde este executa, nomeadamente em relação ao nível do paralelismo envolvido, das características pretendidas relativamente à persistência dos dados, entre outros.

De salientar também, é o facto de toda a arquitectura ter sido idealizada segundo módulos, o que permite uma fácil alteração das características das peças envolventes, sem necessidade de grandes alterações estruturais ao nível da lógica de processamento.

5 Resultados e Análise

Foram realizados alguns testes de performance ao *appender/handler*, de forma a averiguar se, o requisito de tempo de "extra processamento", no registo dos eventos, inferior a 50 milissegundos, se verificava.

Foram registados os tempos no processamento do registo de cada evento, sendo que o número de amostras, referentes à instrução de *log*, foi de 1000. Para cada uma das ferramentas de *logging*, ambas as comunicações, síncrona – através do componente LAPAgentServiceProviderDelegate - e assíncrona – utilizando o LAPDispatcherManager - com o serviço I2S.LAPLogAgent foram analisadas e traduzem-se nos tempos de processamento patentes nas tabelas seguintes. De modo a facilitar a análise, os valores que para esta contribuíram foram os da média, sendo que o valor máximo e mínimo de tempos também foi realçado.

5.1 Resultados

5.1.1 Appender

Tabela 5.1: Tempos registados para o *appender*.

	Tempo Médio	Tempo Máximo	Tempo Mínimo
Comunicação Síncrona	25	32	19
Comunicação Assíncrona	10,2	14	7

Através da Tabela 5.1, é possível verificar que a média dos resultados, na comunicação síncrona no *appender*, encontra-se abaixo do tempo mínimo requisitado pela I2S. O valor máximo neste caso é de 32 milissegundos sendo o mínimo 19 milissegundos. Na comunicação assíncrona, referente ao paralelismo no envio dos eventos para o serviço, o tempo médio é de 10,2 milissegundos, valor bastante satisfatório. O valor máximo neste caso é de 14 milissegundos, e o mínimo é de 7 milissegundos.

5.1.2 Handler

Tabela 5.2: Tempos registados para o *handler*.

	Tempo Médio	Tempo Máximo	Tempo Mínimo
Comunicação Síncrona	23,2	32	19
Comunicação Assíncrona	8,4	11	7

Na Tabela 5.2 são destacados os tempos referentes ao *handler*. Aqui os valores assemelham-se aos do *appender* em ambos os tipos de comunicação. A média de tempos, na comunicação síncrona é de 23,2, o valor máximo de 32 e o mínimo de 19 milissegundos. Já na assíncrona, o valor da média encontra-se abaixo dos 10milissegundos, sendo o seu valor de 8,4, o valor máximo é de 11 e o mínimo de 7 milissegundos.

5.2 Análise

Utilizando um grande número de tempos na amostra, é possível afirmar que estamos perante tempos com um nível credível elevado. Apesar dos tempos já de si serem bastante favoráveis, é de salientar que durante todo este processo de registo de tempos, foram enviados para a consola alguns *logs* de *debug*, de forma a ser possível acompanhar o desenrolar do processamento. Uma vez que isto também tem custos de processamento, é natural que os tempos possam ter valores um pouco acima dos exactos, apesar de que numa amostra de 1000 tempos, estes desvios possam ser desprezáveis.

É possível constatar a ocorrência de valores mais elevados na comunicação síncrona comparativamente à assíncrona. Estes valores, bastante aceitáveis, já eram de esperar, uma vez que o paralelismo existente na comunicação assíncrona permite que seja devolvido o controlo do processo da aplicação principal mais rapidamente. Contudo, mesmo no caso da comunicação síncrona, o valor máximo registado foi de 32 milissegundos no *appender* bem como no *handler*, não sendo de todo negativo relativo aos objectivos iniciais.

5.3 Conclusões

Em suma, é possível afirmar que, relativamente a um dos requisitos mais importantes para a empresa, os objectivos foram atingidos. O *appender* e o *handler* conseguem executar todo o seu processamento registo, não ultrapassando os 50 milissegundos. Na comunicação síncrona, apesar de os tempos chamarem um pouco mais a atenção, estes encontram-se dentro dos máximos estabelecidos. Na comunicação assíncrona os tempos encontram-se bastante abaixo dos 50 milissegundos, alcançando valores muito próximos dos 10 milissegundos, sendos estes valores considerados ideais pela I2S para o processamento extra.

Pode-se afirmar que a I2S, através desta API, consegue comunicar, de forma síncrona e assíncrona, para o serviço I2S.LAPLogAgent em tempos abaixo dos 50 milissegundos, não influenciando em demasia todo o processo da aplicação principal que, esta API, esteja a monitorizar.

6 Conclusões e Trabalho Futuro

Toda o estudo, investigação e análise realizadas, demonstraram que a evolução dos sistemas informáticos poderá orientar-se no sentido do conceito de detecção, recuperação e prevenção de erros automaticamente, diminuindo assim potenciais falhas dos sistemas a longo prazo e o tempo da sua recuperação.

As análises manuais de *logs* extensos farão parte do passado, e encontrar-se-ão em sistemas cujas funções passarão pela análise e recuperação dos erros das aplicações.

A importância do conteúdo da informação colectada aquando das falhas dos sistemas, tem um papel fundamental e relevante de modo a que o conceito *Autonomic Computing* faça parte integrante de todo o software desenvolvido.

Relativamente ao projecto sobre o qual incide esta dissertação, pode afirmar-se, sem falsas modéstias, que foi um sucesso. Esta constatação poderá ser um veículo motivador na aposta deste conceito por parte de outras empresas, apesar dos custos a ele inerentes.

De destacar ainda a importância de um sistema de apoio a resolução de problemas totalmente independente das aplicações que despoletam os erros, facilitando assim a possibilidade da sua integração em outras aplicações já existentes.

6.1 Satisfação dos Objectivos

Os objectivos propostos pela I2S foram atingidos.

A API desenvolvida para as aplicações da empresa poderem capturar, mapear e enviar os eventos para o serviço I2S.LAPLogAgent apresenta-se como uma solução viável.

O serviço apresenta a capacidade de gerir todos os eventos (erros de software) desencadeados pela aplicação que está a ser monitorizada, bem como a bem sucedida comunicação com o I2S.LAPService.

A independência exigida no que respeita ao I2SLAPLogAgent e à aplicação a monitorizar foi conseguida, o que leva a que o uso contínuo do GIS e outras aplicações da I2S, não implique a existência do I2S.LAPLogAgent.

6.2 Trabalho futuro

Como em qualquer projecto de desenvolvimento de software, é um risco elevado admitir como certa a sua completa finalização. Existe sempre um número, por mais reduzido que seja, de possíveis melhorias e reajustes a ser efectuado, não sendo de todo este projecto uma excepção a essa regra tão elementar nas IT.

Relativamente ao projecto em si, é de ter em consideração a influência temporal, que todo o processo de mapeamento dos eventos e respectivo envio para o I2S.LAPService, exercida sobre as aplicações monitorizadas. Auditorias sistemáticas à API e ao I2S.LAPLogAgent, poderão ser solução no controlo desses mesmos tempos.

Um aspecto a ter também em conta em termos futuros, será permitir que a recuperação das aplicações seja efectuada ao nível do I2S.LAPLogAgent, tornando-o independente do I2S.LAPService. Consequentemente, a diminuição de eventuais falhas na comunicação entre estes dois módulos seria preocupação inexistente. De realçar ainda que se a dependência entre os módulos referidos se mantiver, haveria a possibilidade de melhorar a segurança ao nível da comunicação, como por exemplo usando dados encriptados na mesma.

Por último, e no contexto da I2S, a migração do I2S.LAPLogAgent para linguagem RPG [ILERPG] será um dos objectivos futuros, uma vez que algumas das aplicações da empresa se encontram codificadas nessa linguagem.

Referências

[ACIBM] IBM. *An architectural blueprint for autonomic computing*., 2006.

[ACA] IBM. *Autonomic Computing Toolkit - User's Guide*., 2005.

[AOP97] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Videira Lopes, C., Loingtier, J.-M., and Irwin, J - *Aspect-Oriented Programming*, Junho 1997.

[AOP05] Kiczales, G., Mezini, M - *Aspect-Oriented Programming and Modular*, 2005.

[AOPIBM] Kersten, M., - *AOP tools comparison*, 2005.

http://www.ibm.com/developerworks/java/library/j-aopwork1/index.html

http://www.ibm.com/developerworks/java/library/j-aopwork2/index.html

[EDA] Michelson, Brenda M., *Event-Driven Architecture Overview*, Fevereiro 2006.

[CBESTR] Ogle, D., Kreger, H., Salahshour, A., Cornpropst, J., Labadie, E., Chessell, M., Horn, B., Gerken, J., Schoech, J., Wamboldt, M. - *Canonical Situation Data Format: The Common Base Event v1.0.1*.

[CBEBP] *Best Practices for the Common Base Event and Common Event Infrastructure http://download.boulder.ibm.com/ibmdl/pub/software/dw/autonomic/books/cbepractice/index.htm* .

[ACL] Apache Commons Logging http://commons.apache.org/logging/ .

[L4J] Gülcü, C. - *The Complete Log4J Manual* - Agosto 2003.

[JAVA] Java - http://www.java.com/en/about/.

[JRMI] RMI - http://java.sun.com/docs/books/tutorial/rmi/overview.html.

[XML] W3C - World Wide Web Consortium, http://www.w3schools.com/xml/ .

[ANT] *Apache Ant 1.8.1 Manual* - http://ant.apache.org/manual/index.html .

[SPRG] Rod Johnson, Juergen Hoeller, Alef Arendsen, Colin Sampaleanu, Rob Harrop, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Mark Fisher, Sam Brannen, Ramnivas Laddad, Arjen Poutsma - *The Spring Framework - Reference Documentation.*

[TPTP] Eclipse Test & Performance Tools Platform Project - http://www.eclipse.org/tptp/ .

Referências

[ILERPG] IBM. *WebSphere Development Studio - ILE RPG Programmer's Guide*, 2008.