Gestão Autonómica de Desempenho em Aplicações baseadas na tecnologia OSGi

João Ferreira
joao.elias.ferreira@ist.utl.pt

Instituto Superior Técnico
Universidade Técnica de Lisboa

(Orientador: Professor Luís Rodrigues)

Abstract. À medida que os sistemas de software vão ficando mais complexos, as tarefas de os configurar correctamente vão-se também tornando mais difíceis e sujeitas a erros. Importa pois desenvolver técnicas que facilitem a gestão e configuração de sistemas, automatizando tanto quanto possível estas tarefas, criando sistemas *autonómicos*. Esta dissertação propõe e avalia técnicas de computação autonómica, para simplificar a gestão e optimização da operação de sistemas de software baseadas na norma OSGi[1]. Estes sistemas devido à forte modularidade que apresentam, permitem que os diferentes módulos sejam ligados e desligados durante a sua execução alterando as funcionalidades disponibilizadas de forma a optimizar o desempenho dos componentes críticos.

1 Introdução

A arquitectura de software OSGi[1] (Open Services Gateway initiative) define um modelo de componentes dinâmicos para ambientes de programação e execução baseados na linguagem Java[2]. Os componentes da arquitectura OSGi são designados por *pacotes* (do inglês, *bundles*). Um *pacote* OSGi é um módulo de software que tem a característica de poder ser instalado, iniciado, actualizado, ou removido sem obrigar a uma reinicialização do sistema. Cada pacote fornece uma funcionalidade e define as interfaces que outros pacotes podem usar para lhe aceder. Para além disso, esta arquitectura permite também que diferentes pacotes se descubram mutuamente e que utilizem serviços fornecidos pelos restantes pacotes.

Esta arquitectura permite construir aplicações através da composição de pacotes, que são acoplados de forma fraca. Este modelo é particularmente adequado ao desenvolvimento de aplicações para a Web (do inglês, *Web applications*) que se executam integradas num servidor Web (do inglês, *Web Server*) e que, graças às características da arquitectura OSGi, suportam a remoção e introdução de novas funcionalidades em tempo de execução sem obrigar a reiniciar todo o sistema.

[1] http://www.osgi.com

Um dos problemas chave na instalação e manutenção de servidores Web é a optimização do seu desempenho. A dificuldade de prever com exactidão a carga a que o servidor estará sujeito, assim como as interferências entre as diversas aplicações que são executadas na máquina, tornam bastante complexa a tarefa de configurar adequadamente estes sistemas [3, 4]. A execução simultânea de múltiplos pacotes OSGi num único servidor, tipicamente desenvolvidos por diferentes equipas, que se invocam mutuamente de forma difícil de analisar ou prever, vem exacerbar ainda mais este fenómeno. Importa assim, estudar técnicas que permitam automatizar tanto quanto possível a gestão deste tipo de aplicações.

A crescente complexidade dos sistemas informáticos e o correspondente aumento do esforço necessário para os manter levou a IBM a promover activamente a partir de 2001 o conceito de *Computação Autonómica*[5]. O termo autonómico tem raízes na biologia, nomeadamente no Sistema Nervoso Autonómico (*Autonomic Nervous System*) que é o responsável pela regulação de tarefas vitais do corpo humano sem que seja necessária consciência que essas funções estão a ser reguladas. De forma análoga, os sistemas informáticos devem possuir um componente autonómico de gestão, capaz de assegurar um conjunto de propriedades conhecidas pelas propriedades *auto-** (do inglês, "self-*"), nomeadamente: auto-configuração, auto-optimização, auto-reparação, e auto-protecção.

Esta dissertação propõe e avalia técnicas de computação autonómica, para simplificar a gestão e optimização da operação de sistemas de software baseadas na arquitectura OSGi.

O resto deste relatório está estruturado da seguinte forma. Na Secção 2 identificam-se os objectivos e enumeram-se os resultados previstos do trabalho. A Secção 3 faz uma panorâmica sobre o trabalho relacionado. A Secção 4 propõe uma arquitectura para suportar a gestão autonómica de sistemas de software baseados na arquitectura OSGi. A Secção 5 identifica as métricas e metodologias que irão ser usadas para avaliar os resultados do trabalho. Na Secção 6 é feita a calendarização do trabalho a efectuar. Por fim a Secção 7 apresenta as conclusões.

2 Objectivos

Este trabalho aborda o problema de configurar e gerir aplicações web construídas sobre a arquitectura OSGi. A modularidade e ligação fraca entre os componentes das aplicações web construídas sobre esta arquitectura introduzem novos desafios na sua gestão, uma vez que diferentes componentes podem ser desenvolvidos por equipas diferentes e da interacção entre os mesmos podem resultar padrões de utilização de recursos difíceis de prever. Desta forma, estamos interessados em estudar técnicas que permitam automatizar tanto quanto possível a gestão destas aplicações. Mais precisamente, este trabalho pretende atingir os seguintes objectivos:

> *Objectivos:* Estudar e avaliar a aplicação de técnicas de computação autonómica para melhorar e facilitar a gestão de desempenho de aplicações Web baseadas na arquitectura OSGi.

Para atingir estes objectivos, iremos efectuar um estudo de diferentes técnicas usadas na Computação Autonómica. Iremos também estudar quais os mecanismos que devem ser incorporados na arquitectura OSGi para suportar as técnicas estudadas. Os resultados a produzir serão:

> *Resultados Esperados:* i) uma arquitectura que suporte a execução de técnicas de computação autonómica em sistemas baseados na arquitectura OSGi; ii) um protótipo desta arquitectura que concretiza um conjunto seleccionado de técnicas de computação autonómica, aplicadas a um caso de estudo a definir; iii) uma avaliação dessa arquitectura e das técnicas adoptadas.

3 Trabalho Relacionado

Esta secção descreve a norma OSGi, a arquitectura por ela definida e diferentes concretizações desta, assim como exemplos de aplicações onde esta arquitectura se adequa e foi aplicada. Posteriormente é feita uma panorâmica sobre as técnicas que permitem desenvolver sistemas autonómicos, apresenta-se uma arquitectura de referência para o desenvolvimento deste tipo de sistemas, algumas instanciações desta arquitectura e técnicas usadas nas diferentes fases de um gestor autonómico.

3.1 Referência OSGi

A referência OSGi[1] (Open Services Gateway initiative) define uma arquitectura para desenvolvimento e instalação de componentes na linguagem Java, através de um *contentor* OSGi (do inglês, *container*). A arquitectura é também conhecida por Sistema de Módulos Dinâmicos para Java (*Dynamic Module System for Java*) existindo duas especificações JSR (Java Specification Reference) para a utilização de um sistema de módulos dinâmicos numa futura referência da J2SE (Java Standard Edition) [6, 7].

A arquitectura definida pela norma OSGi está esquematizada na Figura 1. A principal componente desta arquitectura é o *pacote*, um *Java Archive*[8] (JAR) contendo classes da linguagem Java, e possivelmente outros recursos como por exemplo ficheiros HTML[2], etc. Uma característica de um pacote OSGi é a existência de um ficheiro denominado *manifesto* (do inglês, *Manifest*) contendo informação que descreve os seus atributos. Os atributos mais relevantes são:

- *Bundle-SymbolicName*, *Bundle-Name* e *Bundle-Version*: Indicam o nome e versão do pacote,
- *Bundle-Activator*: Indica a classe que contém os métodos que são executados na inicialização e paragem de um pacote.

[2] HyperText Markup Language, uma linguagem de marcação utilizada para produzir, entre outros, páginas na Web.

– *Export-Package*, *Import-Package*, *Require-Bundle* e *Bundle-RequiredExecutionEnvironment*: indicam que *Java packages* o pacote exporta e importa, quais os pacotes e ambiente de execução que são necessários para a execução do pacote.

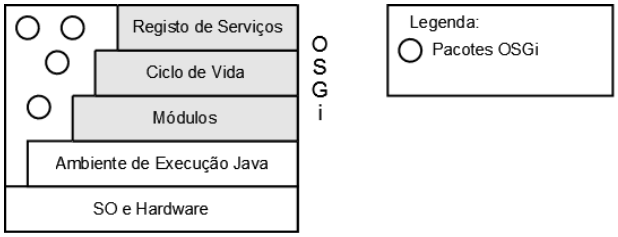

Fig. 1. Arquitectura OSGi

Para além da especificação do que é um pacote, a referência OSGi define as camadas ilustradas na Figura 1:

– *Camada Módulos:* camada responsável pelo carregamento dos pacotes OSGi e pela gestão da visibilidade dos *Java packages* entre os pacotes.
– *Camada Ciclo de Vida:* camada responsável pela gestão do ciclo de vida dos pacotes. Esta camada controla os mecanismos de inicialização e paragem dos pacotes.
– *Camada Serviços:* camada responsável pela gestão do modelo de serviços, permitindo aos pacotes registarem e procurarem serviços.

Do ponto de vista do programador o OSGi permite:

– A criação de pacotes que podem ser instalados, removidos e actualizados, inicializados e parados, sem ser necessário reiniciar o motor OSGi.
– A existência simultânea de mais que uma versão do mesmo pacote em execução num único motor OSGi.
– A existência de estruturas para o desenvolvimento de aplicações orientadas ao serviço, que permitem o registo e descoberta de serviços pelos pacotes.

O OSGi é bastante utilizado em sistemas embebidos e dispositivos de rede. No entanto a utilização do OSGi por parte da aplicação Eclipse 3.0 [9], levou a que este modelo fosse também usado em aplicações para desktop e servidores, como iremos referir de seguida. Existem diferentes concretizações do motor OSGi entre os quais: Eclipse Equinox[3], Knopflerfish[4], Apache Felix[5], e ProSyst Open Source mBedded[6].

[3] `http://www.eclipse.org/equinox/`
[4] `http://www.knopflerfish.org/`
[5] `http://felix.apache.org/`
[6] `http://www.prosyst.com/products/osgi_se_equi_ed.html`

Actualmente podem também ser encontradas diversas aplicações empresariais baseadas na arquitectura OSGi. Vários servidores aplicacionais são, hoje em dia, construídos sobre a arquitectura OSGi, (como por exemplo: Glassfish V3 da Sun[10] ou JOnAS 5 do consórcio OW2[11])). A concretização de aplicações web empresariais é realizada, neste contexto, concretizando os diversos componentes das aplicações na forma de diferentes pacotes OSGi que podem ser instalados no mesmo contentor.

3.2 Computação Autonómica e Ciclo MAPE-K

Uma das arquitecturas mais usadas para concretizar sistemas autonómicos é baseado num ciclo de controlo constituído pelas seguintes fases: monitorização do sistema, análise, planeamento, execução e conhecimento (*knowledge*) designado abreviadamente por ciclo MAPE-K[12]. A Figura 2 esquematiza esta arquitectura. Um componente do sistema capaz de executar este ciclo de controlo designa-se por *Elemento Autonómico*. Um *Elemento Autonómico* é composto por um *Gestor Autonómico* e um *Elemento Gerido*. O gestor observa o elemento gerido através da utilização de *sensores*, que capturam a informação necessária para realizar a adaptação; analisa essa informação; planeia alguma alteração no elemento gerido, se necessário; executa esse plano através de *actuadores* que modificam o elemento gerido. Os componentes que concretizam cada uma das fases do ciclo de controlo partilham conhecimento sobre o elemento gerido. Este ciclo de controlo é executado de forma autónoma, sendo a intervenção humana apenas necessária para a definição de objectivos de alto nível. De seguida é feita uma descrição mais pormenorizada de cada componente do ciclo MAPE-K.

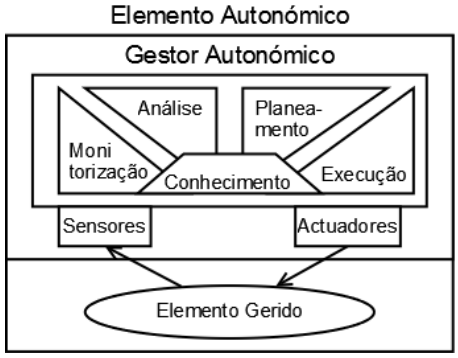

Fig. 2. Ciclo MAPEK

Monitorização A monitorização é responsável pela recolha de métricas do elemento gerido, por exemplo, valores de utilização do CPU, latência de rede,

invocação de uma função, etc. Genericamente, é possível agrupar os mecanismos de monitorização em duas categorias:

- Monitorização Passiva: O elemento gerido proporciona os mecanismos para monitorizar as métricas de interesse. Este tipo de monitorização é usado, por exemplo, para monitorizar o desempenho de um sistemas operativo, uma vez que este último exporta tipicamente interfaces que permitem aceder a informação interna como a utilização de CPU ou à memória utilizada por um determinado processo.
- Monitorização Activa: Quando o elemento gerido não oferece mecanismos de monitorização é necessário de alguma forma alterar o elemento gerido de forma a obter as métricas de interesse. Por exemplo, para detectar a alocação de objectos na máquina virtual Java, uma das alternativas é intersectar o carregamento da classe e interceder os *bytecodes* acrescentando código que façam essa monitorização.

Para realizar monitorização orientada à gestão de desempenho, podem ser tomadas em conta simultaneamente várias métricas. Do ponto de vista do sistema, as métricas com maior interesse são a utilização do CPU e da memória, enquanto que do ponto de vista do utilizador, métricas como número de pedidos respondidos por unidade de tempo ou latência dos pedidos possuem maior relevância.

Outro aspecto importante é a granularidade da monitorização. O desempenho da máquina virtual Java (JVM, do inglês, Java Virtual Machine) é naturalmente relevante para o desempenho das aplicações OSGi. No entanto a captura de informação agregada sobre a máquina virtual como um todo (por exemplo, a memória ocupada pela máquina virtual) pode não ser suficiente para aferir a contribuição de cada pacote OSGi para o desempenho global do sistema. Neste caso, seria preferível monitorizar métricas individualmente em cada pacote de forma a perceber quais os que necessitariam de sofrer alterações.

Existem duas ferramentas que podem ser utilizadas para fazer a monitorização da máquina virtual Java, nomeadamente:

- Java Management Extensions (JMX)[13]: A JMX é uma API (do inglês, Application Programming Interface), mas também uma arquitectura para a construção de aplicações de monitorização e gestão de aplicações Java assim como da JVM. Os recursos geridos são embebidos em *Componentes Geridos* (do inglês, *Managed Beans*) disponibilizando assim uma interface comum para a sua gestão. Esta arquitectura permite definir conectores para a gestão e monitorização remota. Na Secção 3.4 é abordada a JMX em maior detalhe.
- Java Virtual Machine Tools Interface (JVM TI)[14]: A JVM TI é uma API que permite a construção de aplicações de monitorização da JVM usando dois mecanismos complementares: o registo de eventos e a inserção de bytecodes. Na Secção 3.5 é abordada a JVM TI em maior pormenor.

Actualmente a norma OSGi não define qualquer pacote de monitorização. No entanto não será difícil expor a funcionalidade quer da JVM TI e da JMX como pacotes OSGi.

Análise A análise é responsável por agregar e relacionar dados provenientes da monitorização. O objectivo da análise é identificar cenários macroscópicos com base em informação desagregada (por exemplo, relações causa-efeito entre diferentes eventos). Na Secção 3.3 daremos um exemplo de um componente de análise, o *Event Distiller* do *Kinesthetics eXtreme*, que é utilizado para identificar mensagens não solicitadas num servidor de correio electrónico. Por vezes a fronteira entre Análise e Planeamento não é muito clara, existindo técnicas que são utilizadas para concretizar estas duas componentes. É por exemplo o caso das regras "if-then-else" onde a detecção de uma condição (componente de análise) e a acção a tomar (componente planeamento) podem ser concretizadas com esta técnica.

Quando consideramos aplicações OSGi, a componente de análise pode ser útil para co-relacionar diferentes métricas (e.g relação entre utilização do CPU e memória) e reconhecer situações indesejáveis ou identificar tendências para a variação de certas métricas com o decorrer do tempo. Esta informação pode ser utilizada ao nível do planeamento, para despoletar acções de forma a evitar situações indesejáveis, antes de estas acontecerem.

Planeamento O Planeamento é responsável por decidir quais as acções a tomar para atingir o estado desejado do elemento gerido. Este pode ser feito das seguintes formas:

Politicas de adaptação: Um conjunto de regras do tipo "if-then-else" que indica quais as acções a tomar quando uma determinada condição é detectada. Tipicamente, as linguagens utilizadas para descrever estas regras não permitem manter estado (embora este possa ser codificado com parte do estado do sistema que é monitorizado). Uma das principais dificuldades na definição de políticas desta forma consiste em aferir os resultados da interacção de diferentes regras. Por exemplo, num sistema com várias camadas (Web Application) as regras relacionadas com diferentes camadas podem accionar acções que causem conflitos (e.g. cada regra alocar mais recursos, mas a soma desses recursos não estar disponível). Para além disto, o número de regras da política pode aumentar rapidamente à medida que aumenta a complexidade do elemento gerido. Finalmente, a reutilização de regras é bastante dificultada visto as regras serem frequentemente dependentes do elemento gerido;

Modelo arquitectural: Uma possível solução para resolver os problemas indicados anteriormente, passa por modelar a arquitectura do elemento gerido criando um modelo arquitectural que reflecte o seu comportamento, as restrições que este impõe e o estado que se pretende atingir. Este tipo de abordagem não invalida o uso de políticas adaptação: através das métricas

monitorizadas aplicadas ao modelo arquitectural é possível verificar se as condições desejadas do elemento gerido foram violadas e, quando necessário, actuar sobre o elemento gerido. As acções a tomar podem ser especificadas através de políticas de adaptação. Apesar da complexidade da criação de modelos arquitecturais ser elevada, este esforço é geralmente compensado pois a reutilização dos componentes para descrever uma arquitectura é também elevada. Por exemplo a descrição de um servidor web pode ser reutilizada noutro servidor web com baixo esforço. Exemplos de linguagens para a descrição de modelos arquitecturais, podem ser encontrados na: Darwin[15] e Acme[16].

Coordenação de tarefas: Esta abordagem assenta na ideia de que é possível atingir um estado desejado através da coordenação de sub-tarefas que contribuam para esse fim. É semelhante à técnica anterior, no sentido que é feita uma descrição das tarefas que podem ser executadas em vez de uma descrição da arquitectura. Esta abordagem tem a vantagem de permitir a especificação de tarefas alternativas, caso uma tarefa não consiga cumprir o seu objectivo, assim como de especificar relações de ordem e paralelismo entre as tarefas. Um exemplo de uma linguagem que permite definir coordenação de tarefas pode ser encontrado na Little-Jil[17].

Qualquer uma das metodologias descrita acima poderia ser usada na implementação de uma componente de planeamento para gestão de desempenho em aplicações OSGi. Na Secção 4 iremos abordar a escolha efectuada.

Execução A componente de execução é responsável por aplicar as acções definidas ao nível do planeamento sobre o elemento gerido, recorrendo à invocação de actuadores. As acções que podem ser tomadas dependem, em muito, do elemento gerido. Podem variar desde a alteração de configurações do elemento gerido até à adição ou remoção de recursos a um componente do elemento gerido.

No âmbito das aplicações OSGi, a acção mais intuitiva é a de ligar e desligar pacotes. Na Secção 4 iremos abordar em mais detalhe as acções consideradas na nossa arquitectura.

Conhecimento A componente Conhecimento é responsável por manter informação acerca do elemento gerido. Existe uma grande relação entre as componentes Planeamento e Conhecimento, estando por vezes o conhecimento que o gestor autonómico têm do elemento gerido representado no planeamento. No entanto podemos identificar as seguintes formas de obter conhecimento sobre um elemento gerido:

- Introdução humana: O administrador do sistema pode especificar directamente o comportamento do elemento gerido. É o caso das políticas de adaptação introduzidas por um administrador, onde o conhecimento está contido nas regras usadas;

– Funções de utilidade: A utilização de uma função de utilidade para medir o "valor" de um estado do elemento gerido é uma forma de capturar o conhecimento que se tem deste. Esta técnica tem a vantagem de permitir tomar decisões, com base na comparação directa do valor de utilidade entre diversos estados. No entanto a definição da função de utilidade pode não ser trivial, visto que cada parâmetro que influencia o valor do estado deve ser quantificado. Em [18] é descrita a utilização de funções de utilidade em ambientes web;
– Aprendizagem: A utilização de aprendizagem, através da aplicação de acções sobre o elemento gerido e a recolha dos resultados obtidos, é também uma técnica possível para a obtenção de conhecimento sobre o elemento gerido que pode servir de suporte à execução do planeamento. Um problema desta técnica é que o espaço de estados possíveis a ser testado pode ser demasiado grande pelo que é necessário usar outras técnicas para o reduzir, como por exemplo introduzir conhecimento sobre o domínio. A utilização de aprendizagem pode ser executada em dois momentos: i) tempo de execução ii) tempo de desenvolvimento. Um problema da utilização de aprendizagem em tempo de execução é a introdução de penalidades no desempenho, enquanto que em tempo de desenvolvimento apresenta limitações que derivam da diferença entre o conjunto de acções testadas para a geração de conhecimento e o conjunto de acções que surgem em tempo de execução, não sendo o conhecimento gerado relevante para responder a estas acções. Em [19] são abordadas técnicas para a introdução de conhecimento utilizando aprendizagem.

3.3 Exemplos de Concretização do MAPE-K

ABLE Toolkit O *Agent Building and Learning Environment* (ABLE)[20] é uma bancada desenvolvida na linguagem Java que permite construir sistemas autonómicos usando uma arquitectura baseada em agentes. Para além do agente que concretiza o ciclo MAPE-K, o ABLE disponibiliza um conjunto de algoritmos de aprendizagem e raciocínio. O agente que implementa o ciclo MAPE-K é chamado *Autotune Agent* e tem três principais componentes: *AutotuneAdaptor*, *AutotuneController* e o *AutotuneMetrics*, como se ilustra na Figura 3. De seguida iremos analisar como o ABLE concretiza cada componente do ciclo MAPE-K.

Monitorização: O ABLE permite a definição de sensores para a obtenção do estado do elemento gerido, de forma a que a monitorização possa ser realizada de forma passiva ou activa. Mais concretamente, a monitorização é feita pelo *AutotuneAdaptor* que identifica quais as métricas que definem o estado do elemento gerido e as monitoriza, invocando para isso os sensores sobre o elemento gerido. Os indicadores de desempenho obtidos desta forma são armazenando através do *AutotuneMetric*.

Análise, Planeamento e Conhecimento: No ABLE a fronteira entre estes três componentes é bastante difusa, sendo estes componentes responsabilidade do

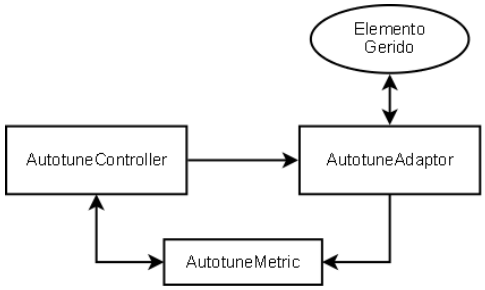

Fig. 3. Arquitectura de um Agente Autotune

AutotuneController. Este, baseia-se nos indicadores de desempenho mantidos pelo *AutotuneMetric* para obter informação sobre o estado do elemento gerido. A ABLE suporta dois modos de definição do algoritmo de análise e planeamento:

- O primeiro consiste em definir uma política de adaptação do sistema utilizando uma linguagem concebida para o efeito, a *ABLE Rule Language* (ARL), que inclui desde regras do tipo "if-then-else", até suporte para a criação de *scripts*, regras de predicados e um conjunto de motores de inferência.
- O segundo modo consiste em utilizar um algoritmo de controlo, para o qual é possível extrair de forma automática os parâmetros através de aprendizagem em tempo de desenvolvimento. O algoritmo de controlo recebe como argumentos os valores do estado desejado do elemento gerido (p.e utilização do CPU e memória) e devolve quais os valores de configuração do elemento gerido que devem ser alterados.

Execução: Para alterar o estado do elemento gerido para que este convirja para o estado desejado é necessário modificar ou adaptar configurações no elemento gerido. Assim o *AutotuneController* indica ao *AutotuneAdaptor* quais os valores a alterar e este através de actuadores efectua as alterações necessárias sobre as configurações do elemento gerido.

Exemplo de aplicação: Em [21] é descrita a utilização de agentes Autotune para gestão de desempenho do servidor web Apache. Nesta aplicação, o algoritmo de controlo relaciona os valores de configuração do sistema gerido (neste caso as variáveis *KeepAlive* e *MaxClients* do servidor Apache) e os valores actuais e passados do estado do elemento gerido (utilização do CPU e memória). Neste caso, foi possível parametrizar este algoritmo de forma automática, recorrendo a aprendizagem, conforme referido anteriormente. A avaliação do sistema demonstra que em ambientes com carga variável, os níveis de utilização do sistema são mantidos constantes.

Kinesthetics eXtreme O Kinesthetics eXtreme (KX)[22] tem como principal objectivo dar capacidades autonómicas a sistemas legados, isto é, sistemas que não são exequíveis de redesenhar de forma a adicionar comportamento autonómico. Tal pode acontecer não só porque o código fonte não está disponível mas também por não ser vantajoso fazer a sua alteração. Os autores identificaram quatro componentes principais na arquitectura: sensores, indicadores (*gauges*), controladores e actuadores. No entanto, no desenvolvimento do KX, os autores criaram diferentes módulos que podem ser utilizados em separado, cada um com a sua funcionalidade bem definida, como representado na Figura 4.

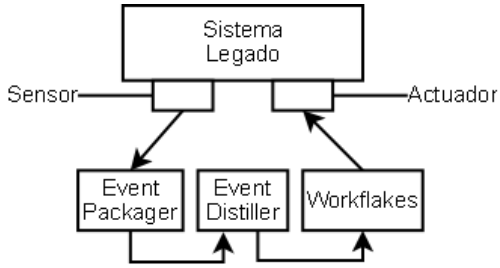

Fig. 4. Arquitectura do KX

A relação entre estes módulos e os componentes do ciclo MAPE-K é a seguinte:

Monitorização: A Monitorização é feita pelo *Event Packager*, responsável por recolher e uniformizar a informação dos diferentes sensores. O tipo de monitorização que pode ser feita apenas depende do sistema legado e da informação que este permite recolher. Enquanto os sensores dependem da tecnologia especifica do elemento gerido, o Event Packager está mais relacionado com o domínio do problema de gestão autonómica, podendo ser reutilizado em problemas idênticos.

Análise: A Análise é feita pelo *Event Distiller* que agrega, filtra e interpreta os dados da monitorização. O Event Distiller é o módulo que permite construir os indicadores. De notar que o KX apenas permite que seja feita a monitorização de um sistema com estes dois componentes (Monitorização e Análise) o que permitiria construir uma ferramenta de monitorização, sem oferecer ferramentas de suporte à auto-adaptação (o que não constituiria por si só, um sistema autonómico).

Planeamento: Este componente é concretizado pelo *Workflakes* (módulo que permite definir os controladores) que, a partir dos eventos gerados pelo *Event Distiller* e do conhecimento que tem do elemento gerido, define um plano para actuar sobre este. O planeamento é feito através da organização de tarefas que contribuam para atingir o estado desejado. As tarefas são escolhidas com base

no conhecimento que o gestor autonómico tem do elemento gerido(como este conhecimento é obtido é explicado de seguida) e como cada tarefa influencia o estado do sistema.

Uma vantagem desta abordagem é permitir que as tarefas tenham dependências entre elas e, caso a aplicação de uma tarefa não produza o resultado esperado, impedir a execução de tarefas subsequentes, tendo início a geração de um novo plano, de forma a atingir o estado desejado.

Execução: Este componente é concretizado pelo *Workflakes*, que invoca os actuadores no sistema gerido. A separação entre o *Workflakes* e os actuadores permite ao *Workflakes* abstrair-se das especificidades dos actuadores, que tal como os sensores, são dependentes do elemento gerido.

Conhecimento: O conhecimento que o gestor tem do elemento gerido é representado através de um modelo arquitectural que este possui da arquitectura do sistema gerido. Este modelo pode ser obtido *a priori* pelo administrador do sistema autonómico ou aprendido durante a execução do sistema.

3.4 Java Management Extensions (JMX)

A JMX é uma tecnologia para a plataforma Java que fornece suporte para a monitorização e gestão da JVM e de aplicações Java. Esta tecnologia está disponível na Java Standard Edition (J2SE) desde a versão 5.0, foi definida originalmente no Java Specification Request (JSR) 003 enquanto que o seu modelo de gestão remota foi definido pelo JSR 160.

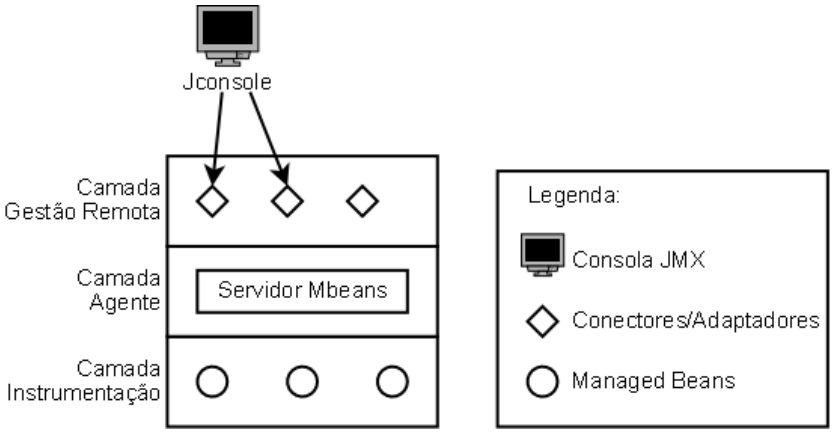

Fig. 5. Arquitectura da JMX

A JMX define uma arquitectura para a construção de aplicações que possam ser geridas e monitorizadas. A Figura 5 representa essa arquitectura que é definida por três camadas:

– Instrumentação: Os recursos a gerir e monitorizar são encapsulados como *Componentes Geridos* (MBeans). Estes garantem que os recursos são acedidos pela camada Agente por uma interface bem definida, com um comportamento normalizado através do uso de padrões de desenho pré-definidos.
– Agente: Esta camada controla o registo dos MBeans através de um Servidor de MBeans dedicado a esta tarefa, que disponibiliza estes componentes às ferramentas de monitorização e gestão.
– Gestão Remota: Esta camada permite a definição de diferentes conectores e adaptadores de forma a disponibilizar os recursos a gerir a componentes externos à JVM.

Exemplos de aplicações de gestão e monitorização que utilizam a JMX podem ser encontrados na JConsole[23] e na VisualVM[24].

3.5 Java Virtual Machine Tools Interface (JVM TI)

A JVM TI[14] é uma tecnologia que permite realizar monitorização da JVM através da criação de um agente ao qual é delegada essa tarefa. Este agente utiliza a interface JNI (do inglês, *Java Native Interface*) para comunicar com a JVM. Devido a esta concretização o agente deve ser concretizado na linguagem C ou numa similar. Na Figura 6 é ilustrada a arquitectura da JVM TI.

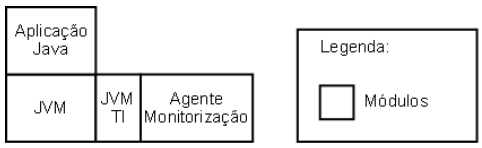

Fig. 6. Arquitectura da JVM TI

A JVM carrega o agente antes de quaisquer bytecodes, sendo o agente executado no mesmo processo da JVM. A monitorização pode ser realizada de diversas formas nomeadamente através da chamada de funções à JVM, registo de eventos na JVM e instrumentação de bytecodes. A instrumentação de bytecodes é a técnica mais eficiente visto que não implica mudança de contexto em cada acção de monitorização executada. Para além disso a sua execução pode ser também optimizada pela JVM.

3.6 Monitorização de Recursos em Aplicações Baseadas em OSGi

Em [25] são descritas técnicas de monitorização de recursos em aplicações OSGi. Os recursos monitorizados são o CPU e a memória. O contributo que se destaca é o facto da monitorização das aplicações OSGi ser feito ao nível do pacote e não ao nível da JVM como um todo. Este controlo mais fino oferece um conhecimento

mais pormenorizado sobre o funcionamento do sistema, o que permite uma gestão mais eficaz da aplicação.

Para concretizar a monitorização os autores usaram a Java Virtual Machine Tools Interface (JVM TI)[14] para criar um agente responsável pela monitorização (A JVM TI foi descrita na Secção 3.5). Usando esta tecnologia é possível associar cada *tarefa* (do inglês, *thread*) ao pacote OSGi a que pertence. Desta forma, é possível obter a utilização de CPU desse pacote através da soma do CPU utilizado por cada uma das suas *tarefas*.

O facto de arquitectura OSGi permitir que um pacote use serviços de outros pacotes torna não trivial a atribuição da utilização dos recursos a um pacote em concreto. Este problema pode ser descrito da seguinte forma: um pacote A disponibiliza um serviço e um pacote B utiliza-o. A quem devem ser atribuídos os recursos utilizados por A a pedido de B? As alternativas são:

- Atribuição Indirecta: Os recursos consumidos pelo serviço são atribuídos ao pacote que o utiliza, neste caso o pacote B.
- Atribuição Directa: Os recursos consumidos pelo serviço são atribuídos ao pacote que disponibiliza o serviço, neste caso o pacote A.

Em [25] os autores optaram pela atribuição directa, uma vez que esta faz distinção entre os recursos usados por um serviço e os recursos usados por quem o consome. No entanto para concretizar esta alternativa é necessário isolar o serviço do pacote que o consome, de forma a permitir distinguir a utilização de recursos. Este isolamento foi conseguido através da utilização de uma *proxy* que detecta a invocação do serviço, sendo necessário para isso alterar o motor OSGi.

De forma a concretizar a monitorização da memória ocupada por cada pacote os autores optaram por intersectar a alocação de objectos através da intercessão dos bytecodes aquando do carregamento das classes. Os bytecodes introduzidos realizam a contabilização do espaço em memória ocupado por cada pacote.

Como referido anteriormente, a principal contribuição do trabalho descrito em [25] centra-se nos mecanismos de monitorização. Contudo, os autores oferecem também algum suporte à adaptação, especificamente para os casos onde é possível definir um limite máximo aos recursos disponíveis a cada pacote OSGi. Para isso, foi desenvolvido um serviço (implementado num pacote OSGi) que utiliza informação disponibilizada pelo agente JVM TI de forma a detectar situações, onde os recursos utilizados por um pacote ultrapassam os limites definidos *a priori*. Pacotes que se encontrem nesta situação são alertados por este serviço, caso o subscrevam explicitamente, ficando delegada a responsabilidade em cada pacote de activar os mecanismos de adaptação adequados de forma minimizar a sua utilização de recursos.

3.7 Gestão Autonómica de Desempenho em ambientes Virtualizados

Uma área de aplicação de computação autonómica é a gestão em ambientes virtualizados. A utilização de virtualização permite a partilha de recursos físicos entre vários serviços, mantendo o isolamento entre estes. Desta forma, um único

servidor pode executar vários serviços que consomem recursos de forma isolada relativamente a outros serviços a executar na mesma máquina. As plataformas que suportam este tipo de funcionamento designam-se por Máquinas Virtuais (VM, do Inglês, *Virtual Machines*), que concretizam mecanismos que permitem simular a existência de múltiplos equipamentos e sistemas operativos num único equipamento partilhado.

Uma vantagem da virtualização é que esta permite a atribuição de recursos a cada serviço, de forma dinâmica. Assim, não é necessário fazer uma previsão da carga máxima a que cada serviço vai ser sujeito de forma a realizar de antemão uma distribuição estática de recursos considerando o pior caso. Isto permite obter uma utilização mais eficiente dos recursos disponíveis. Neste contexto, técnicas de computação autonómica podem ser utilizadas para realizar operações de adaptação de forma a redistribuir os recursos alocados a cada serviço de acordo com as condições dinâmicas de operação do sistema.

Em [26] é feita a análise de diferentes alternativas e proposta uma solução para o problema da criação de políticas para a gestão de recursos em aplicações de três camadas que se executam em ambientes virtualizados. De forma sintética:

- Uma das abordagens possíveis baseia-se na especificação manual de regras com base na experiência de um administrador de sistemas. Esta abordagem tem a desvantagem de que a especificação e manutenção destas regras é de elevada complexidade e sujeita a erro humano.
- Outra abordagem possível passa pela definição de um controlador que, dado o estado actual, consegue adaptar o sistema de forma a que este convirja para um estado desejado. A configuração dos parâmetros do controlador pode ser excessivamente complexa, no entanto esta pode ser conseguida através de mecanismos baseados em aprendizagem em tempo de execução. Contudo esta abordagem pode gerar o aparecimento de regras imprevistas e indesejáveis
- A solução proposta pelos autores é uma solução intermédia para a criação de políticas de adaptação que usa aprendizagem em tempo de desenvolvimento para criar regras de adaptação. As regras criadas podem assim ser interpretadas pelo administrador do sistema, mas a sua criação e manutenção é feita de forma automática.

O algoritmo proposto pelos autores é ilustrado na Figura 7 e consiste na geração de cargas aleatórias (usando *Rule Constructor*) que são passadas ao *Optimizer*. Este determina qual a configuração que optimiza a resposta do sistema para essa carga. A partir dos pares (carga, configuração) recolhidos é criada uma árvore de decisão utilizando o Weka[7]. Essa árvore de decisão é depois representada num conjunto de "if-then-else" encadeados que são mais facilmente interpretados pelo administrador do sistema.

De modo a gerar a melhor configuração como resposta a uma determinada carga o *Optimizer* gera um conjunto de configurações candidatas e escolhe a que oferece melhor tempo de resposta e consumo de recursos. A atribuição de

[7] `Weka(http://www.cs.waikato.ac.nz/ml/weka)` é uma ferramenta para a criação de árvores de decisão

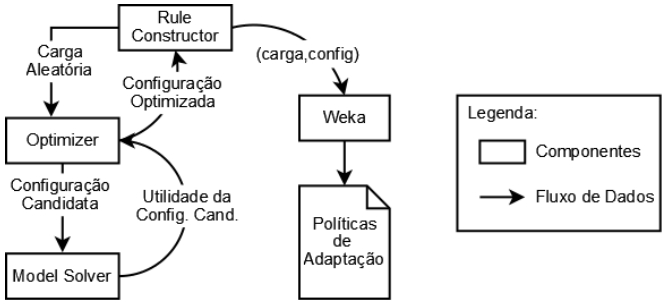

Fig. 7. Esquema do algoritmo para geração de politicas de adaptação

um valor a cada configuração é feita pelo *Model Solver* utilizando uma função de utilidade e um modelo arquitectural, que reflecte o tempo de resposta e a utilização de recursos do sistema.

O algoritmo de geração de configurações candidatas do *Optimizer* não gera todas as configurações possíveis para que estas sejam avaliadas pelo *Model Solver*, pois esta solução teria uma complexidade demasiado elevada. Para diminuir o número de configurações testadas os autores propõem um algoritmo que reduz o número de configurações geradas, mas a sua descrição ultrapassa o âmbito deste relatório.

3.8 Métricas de Desempenho

A monitorização dos valores de utilização do CPU e da memória permitem aferir a carga de um sistema informático, no entanto, não são métricas que capturem o desempenho de um sistema do ponto de vista do utilizador final. Um sistema pode manter a utilização do CPU em valores constantes mas não responder a 100% dos pedidos enquanto que outro pode ter uma utilização do CPU bastante variável mas, mesmo assim, conseguir responder a todos os pedidos. Nesta subsecção introduzimos algumas métricas que permitem caracterizar o desempenho de um sistema informático (estas métricas foram propostas em [27]):

Disponibilidade: A disponibilidade (do inglês, *availability*, por vezes também designada por *uptime*), indica a fracção do tempo que o sistema está disponível. A disponibilidade pode ser calculada com base em duas outras métricas: o tempo médio entre falhas, *MTBF* (do inglês, *Mean Time Between Failures*) e o tempo médio de reparação, *MTTR* (do inglês, *Mean Time to Repair*), de acordo com a seguinte fórmula:

$$disponibilidade = \frac{MTBF - MTTR}{MTBF}$$

Note-se que é possível melhorar a disponibilidade reduzindo a frequência das falhas ou reduzindo o tempo médio de reparação.

Cobertura: A cobertura (no original em inglês, *yield*) indica qual a fracção dos pedidos recebidos é executada. Ou seja:

$$\text{cobertura} = \frac{\text{pedidos completados}}{\text{pedidos recebidos}}$$

Esta métrica tem um valor próximo da disponibilidade, no entanto reflecte melhor a percepção da disponibilidade pelo utilizador. Um segundo de indisponibilidade num período de carga afecta bastantes pedidos diminuindo a cobertura. No entanto, a mesma indisponibilidade num período sem pedidos não afecta os utilizadores.

Abrangência: A abrangência (no original em inglês, *harvest*) reflecte qual a percentagem de dados que está disponível num determinado momento e, indirectamente, o grau de completude na resposta a um pedido.

$$\text{abrangência} = \frac{\text{dados disponíveis}}{\text{dados totais}}$$

Esta métrica reflecte o facto de ser possível diminuir a carga imposta por cada pedido diminuindo a informação retornada ao utilizador, por exemplo, fazendo uma procura sobre apenas parte da base de dados (no processamento de pesquisas), ou fornecendo conteúdos com menor definição (em servidores multimédia).

Em [27] faz-se a seguinte observação: a capacidade máxima de um dado sistema pode ser capturada de forma grosseira pelo produto da cobertura pela abrangência, ou seja:

$$\text{cobertura} \times \text{abrangência} \approx \textit{constante}$$

Esta observação captura o facto de todos os sistemas possuirem limites físicos ao processamento de pedidos, como por exemplo a largura de banda das E/S, que nenhum sistema de gestão pode ultrapassar. Em situações de sobrecarga, o melhor que um sistema de gestão pode fazer é obter um equilíbrio adequado entre a cobertura e a abrangência fornecidas.

É possível materializar estas observações para sistemas OSGi da seguinte forma:

- Em situações de sobrecarga é possível limitar a cobertura restringindo os recursos associados a certos pacotes ou, em última análise, removendo pacotes.
- Alternativamente, é possível diminuir a abrangência substituindo pacotes por outros pacotes que ofereçam o mesmo serviço mas com uma qualidade degradada com uma menor utilização de recursos.

3.9 Resumo do Trabalho Relacionado

Nesta secção foi feita uma breve panorâmica das duas tecnologias que servirão de base ao trabalho a realizar, nomeadamente a computação autonómica e a arquitectura OSGi.

Sobre a computação autonómica foi feita uma descrição da arquitectura padrão mais usada neste tipo de soluções, o ciclo MAPE-K. Esta descrição foi complementada com vários exemplos da concretização dessa arquitectura, incluindo um exemplo de utilização de computação autonómica na atribuição dinâmica de recursos em ambientes virtualizados, onde mostrámos a utilização de técnicas para a geração de políticas de adaptação.

Em relação à arquitectura OSGi, após uma descrição das suas principais características e áreas de aplicação, foram referidos trabalhos que podem ser utilizados na construção de soluções autonómicas para a gestão de desempenho de sistemas OSGI, nomeadamente, trabalho que permite realizar a monitorização deste tipo de sistemas.

4 Arquitectura

Neste trabalho propomos uma arquitectura de um gestor autonómico baseado no ciclo MAPE-K, orientado a aplicações Web concretizadas numa arquitectura OSGi. A arquitectura típica de uma aplicação web é ilustrada na Figura 8. Nesta arquitectura os clientes contactam o servidor aplicacional (através de pedidos HTTP) que encaminha os pedidos para as aplicações adequadas. A instalação da aplicação web é feita num servidor aplicacional através de um *WAR* (do inglês, *Web Archive*). O *WAR* é um *JAR* com uma determinada estrutura de ficheiros que permite o registo da aplicação web no servidor aplicacional.

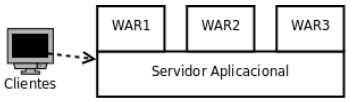

Fig. 8. Arquitectura de uma aplicação web

A concretização de aplicações Web sobre a arquitectura OSGi é ilustrada na Figura 9; neste caso o motor OSGi é instalado no servidor aplicacional também encapsulado num *WAR*. A interligação entre o motor OSGi e servidor aplicacional, que permite que os pedidos recebidos pelo servidor aplicacional sejam respondidos pelos pacotes OSGi que fornecem a funcionalidade correspondente, é feita através de uma camada (designada na figura por *ponte*) onde a aplicação web (concretizada por pacotes OSGi) regista os serviços que fornece.

A arquitectura do gestor autonómico é ilustrada na Figura 10. O gestor autonómico é ele próprio um pacote OSGi, que monitoriza e executa acções sobre o elemento gerido através de sensores e actuadores. A arquitectura definida pelo ciclo MAPE-K tem a vantagem de separar a funcionalidade que é necessário concretizar em componentes modulares, pelo que o gestor autonómico será baseado no ciclo MAPE-K. O facto de esta arquitectura ser adoptada na maioria da bibliografia existente facilita também a comparação do trabalho a realizar com o trabalho anterior.

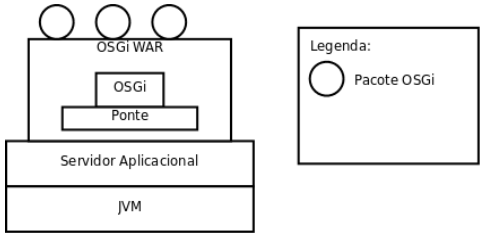

Fig. 9. Arquitectura de uma aplicação web concretizada sobre a arquitectura OSGi

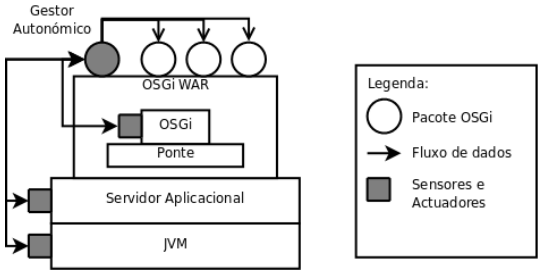

Fig. 10. Arquitectura do gestor autonómico de aplicações web concretizadas sobre arquitectura OSGi

De seguida, descrevemos com mais pormenor a concretização de cada um dos componentes do ciclo MAPE-K.

4.1 Monitorização

A monitorização pretende recolher métricas que sejam relevantes para a gestão de desempenho. No âmbito da gestão de desempenho de aplicações OSGi importa que a recolha dessas métricas seja feita ao nível dos pacotes e não apenas ao nível da JVM, pois permite aferir a contribuição de cada pacote para o desempenho global da JVM. Os sensores que irão ser concretizados são os seguintes:

- *Sensores de desempenho:* estes sensores monitorizam métricas que capturam o desempenho da aplicação web tal como este é observado pelo utilizador. Estes sensores serão instalados na camada que liga o servidor aplicacional e o motor OSGi (*ponte*).
  - sensor que monitoriza o número de pedidos respondidos por unidade de tempo de cada pacote
  - sensor que monitoriza a cobertura de cada pacote
  - sensor que monitoriza a latência dos pedidos de cada pacote
- *Sensores de utilização de recursos*: estes sensores monitorizam métricas que oferecem informação sobre a utilização de recursos a cada instante. Estas métricas são importantes porque facilitam o diagnóstico de causas que justifiquem alterações ao nível das métricas de desempenho sentidas pelo utilizador. Estes sensores serão instalados no motor OSGi e na JVM através da

utilização da ferramenta JVM TI (Secção 3.5) para a criação de um agente JVM TI.

- sensor que monitoriza a utilização do CPU
- sensor que monitoriza a utilização da memória

4.2 Execução

A componente de execução permite actuar sobre o elemento gerido. Para concretizar um maior número de acções é necessário actuar de diferentes formas nos diversos componentes do elemento gerido, nomeadamente concretizar vários actuadores que suportam as seguintes acções:

Actuadores sobre os pacotes OSGi :

- *instalar e remover pacote:* permitem remover e instalar (pacotes anteriormente removidos) da aplicação web caso seja desejável. Estas acções são já suportadas pelo motor OSGi.
- *efectuar troca de pacotes:* permite trocar um pacote por outro que ofereça melhor compromisso entre desempenho e funcionalidade. Quando um determinado pacote apresenta problemas de desempenho. é possível efectuar uma troca por outro que tenha melhor desempenho mas que no entanto ofereça uma funcionalidade reduzida ou degradada. Esta acção é diferente de remover e instalar outro pacote pois permite a transferência do estado interno para o novo pacote o que permite a manutenção do serviço sem que o utilizador se aperceba desta troca.
- *modificar parâmetros configuráveis de pacotes:* esta acção é semelhante à anterior pois consiste em alterar a configuração de um pacote para melhorar o seu desempenho. Pressupõe a existência de uma descrição dos parâmetros que podem ser configurados.

Actuadores sobre a JVM :

- *modificar a prioridade de execução das* tarefas *associadas a um pacote:* esta acção permite controlar o impacto relativo de um determinado pacote para o desempenho global do elemento gerido. Em consequência, a latência dos pedidos dirigidos a pacotes com menor prioridade poderá aumentar. Para concretizar esta acção é necessário alterar o motor OSGi para associar as *tarefas* ao pacote respectivo.
- *modificar o limite máximo de utilização de memória de um pacote:* esta acção permite que um pacote tenha um valor máximo de utilização de memória. Este actuador requer um estudo mais aprofundado, pois as suas consequências não são previsíveis, uma vez que se esse valor for atingido uma excepção de falta de memória será lançada e se o código não estiver protegido o mais provável será interromper a sua execução. Para concretizar esta acção é necessário intersectar o carregamento das classes Java e interceder os bytecodes para que faça a contagem da memória ocupada bem como a verificação se os limites foram alcançados.

Actuadores sobre o servidor aplicacional :

– *modificar o limite máximo de pedidos direccionados a um pacote:* esta acção permite rejeitar ligações antes que os pedidos sejam entregues ao pacote OSGi a que são direccionados. Para concretizar esta acção é necessário alterar o servidor aplicacional sobre o qual os pacotes estão instalados.

4.3 Análise e Planeamento

Decidimos agrupar a descrição da concretização destas duas componentes numa única subsecção, pois planeamos concretizá-las através de políticas de adaptação. Para este fim, pensamos desenvolver uma linguagem de *scripting* que permitirá especificar regras do tipo "if-then-else". Esta componente tem de ter acesso aos dados provenientes da monitorização bem como às acções que a componente execução disponibiliza.

Estas duas componentes (análise e planeamento) embora sejam concretizadas da mesma forma, através de políticas de adaptação, terão funcionalidades diferentes:

– a análise é responsável por agregar os dados provenientes da monitorização e gerar eventos de alto nível que capturem situações que envolvam várias métricas.
– o planeamento é responsável por especificar quais as acções e sobre que pacotes serão executadas quando um determinado evento é gerado pela componente análise.

5 Metodologia de avaliação do trabalho

A avaliação deste trabalho será feita sobre uma concretização da arquitectura descrita, utilizando tráfego emulado injectado sobre o sistema. O tráfego injectado simulará diferentes perfis de cargas variando os seguintes parâmetros: i) número de pedidos por unidade de tempo ii) número de clientes iii) diferentes tipos de pedidos para que diferentes conjuntos de pacotes sejam invocados.

A ferramenta que iremos utilizar para emular os perfis descritos é o JMeter[8]. Com a injecção destes perfis de tráfego no sistema, pretendemos avaliar qual a *qualidade* do estado do elemento gerido obtido. A *qualidade* será avaliada utilizando um sistema de pontuação que associa um valor a cada pedido respondido. Este valor depende da importância desse pedido no sistema, havendo pedidos que são mais valiosos que outros. Assim o valor da qualidade de um certo cenário C constituido pelo subconjunto dos pedidos respondidos $P = \{p_1, p_2, \ldots, p_n\}$ é dado por

$$qualidade = \sum_{P} valor(p)$$

Um outro aspecto deste trabalho que é importante avaliar é a penalidade introduzida pelo gestor autonómico e como se comporta o elemento gerido em

[8] `http://jakarta.apache.org/jmeter/`

termos de *qualidade* com e sem a utilização de técnicas autonómicas. Esperamos que nos perfis com carga elevada o sistema autonómico apresente um melhor valor de qualidade, pois irá privilegiar a execução dos pacotes associados aos pedidos com maior valor.

6 Planeamento do Trabalho a Realizar

O planeamento do trabalho a realizar é o seguinte:

- 10 de Janeiro - 29 Março : Desenho detalhado e concretização da arquitectura proposta, incluindo testes iniciais,
- 30 Março - 3 Maio: Execução da avaliação experimental,
- 4 Maio - 23 Maio: Escrita de um artigo descrevendo o projecto realizado,
- 24 Maio - 15 Junho: Finalização da escrita da dissertação de mestrado
- 15 Junho: Entrega da dissertação de mestrado.

7 Conclusão

Este relatório pretendeu abordar a gestão de desempenho de aplicações baseadas na norma OSGi, através da utilização de técnicas de Computação Autonómica. Para tal apresentámos uma descrição do trabalho relacionado nestas duas áreas que são ortogonais mas que podem beneficiar com a sua intersecção. Assim identificámos quais as métricas que têm relevância serem monitorizadas e quais as acções que uma aplicação OSGi pode ser sujeita por um gestor autonómico.

Desenhámos uma arquitectura para a construção de um elemento autonómico para aplicações OSGi, baseada na escrita de políticas de adaptação.

Por fim apresentámos a metodologia com que este trabalho será avaliado, identificando cenários e métricas relevantes.

Agradecimentos

Agradeço ao J. Mocito e ao J. Leitão as sugestões e comentários durante a preparação deste relatório.

References

1. The OSGi Alliance: OSGi Service Platform Core Specification, Release 4, Version 4.1. `http://www.osgi.org/Download/Release4V41` (2007)
2. Gosling, J., Joy, B., Steele, G.L.: The Java Language Specification. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1996)
3. Diao, Y., Gandhi, N., Hellerstein, J., Parekh, S., Tilbury, D.: Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server. Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP (2002) 219–234

4. van der Mei, R., Hariharan, R., Reeser, P.: Web server performance modeling. Telecommunication Systems (2001)
5. IBM: Autonomic computing: Ibm's perspective on the state of information technology. IBM Journal (2001)
6. Sun Microsystems: JSR 277: Java Module System. http://jcp.org/en/jsr/detail?id=277
7. Sun Microsystems: JSR 291: Dynamic Component Support for JavaTM SE. http://jcp.org/en/jsr/detail?id=27
8. Sun Microsystems: JAR File Specification. http://java.sun.com/javase/6/docs/technotes/guidesjar/jar.html
9. Gruber, O., Hargrave, B.J., McAffer, J., Rapicault, P., Watson, T.: The eclipse 3.0 platform: Adopting osgi technology. IBM Systems Journal (2005)
10. Sun Microsystems: Sun GlassFish Enterprise Server v3 Prelude Release Notes. http://docs.sun.com/app/docs/coll/1343.7 (2008)
11. OW2 Consortium: Jonas - White Paper v1.2. http://wiki.jonas.objectweb.org/xwiki/bin/download/Main/Documentation/JOnAS5_WP.pdf (2008)
12. IBM: An architectural blueprint for autonomic computing, fourth edition. Technical report, IBM (2006)
13. Sun Microsystems: Java Management Extensions. http://java.sun.com/javase/6/docs/technotes/guides/jmx/index.html
14. Sun Microsystems: Java Virtual Machine Tools Interface. http://java.sun.com/javase/6/docs/platform/jvmti/jvmti.html
15. Magee, J., Dulay, N., Eisenbach, S., Kramer, J.: Specifying distributed software architectures. In: Proceedings of the 5th European Software Engineering Conference, Springer-Verlag (1995) 137–153
16. Garlan, D., Schmerl, B.: Model-based adaptation for self-healing systems. In: ACM SIGSOFT Workshop on Self-Healing Systems (WOSS'02),, Charleston, SC (18-19 November 2002)
17. Lemer, B.S., McCall, E.K., Wise, A., Cass, A.G., Osterweil, L.J., Stanley M. Sutton, J.: Using little-jil to coordinate agents in software engineering. Automated Software Engineering, International Conference on **0** (2000) 155
18. Walsh, W., Tesauro, G., Kephart, J., Das, R.: Utility functions in autonomic systems. Autonomic Computing, 2004. Proceedings. International Conference on (May 2004) 70–77
19. Tesauro, G.: Online resource allocation using decompositional reinforcement learning. In: AAAI. (2005) 886–891
20. Bigus, J.P., Schlosnagle, D.A., Pilgrim, J.R., III, W.N.M.a.: Able: A toolkit for building multiagent autonomic systems. IBM Syst. J. **41**(3) (2002) 350–371
21. Diao, Y., Hellerstein, J.L., Parekh, S., Bigus, J.P.: Managing web server performance with autotune agents. IBM Syst. J. **42**(1) (2003) 136–149
22. Kaiser, G., Parekh, J., Gross, P., Valetto, G.: Kinesthetics extreme: an external infrastructure for monitoring distributed legacy systems. Autonomic Computing Workshop, 2003 (June 2003) 22–30
23. Sun Microsystems: Jconsole. http://java.sun.com/javase/6/docs/technotes/guides/management/index.html
24. Sun Microsystems: Visualvm. http://java.sun.com/javase/6/docs/technotes/guides/visualvm/index.html
25. Miettinen, T.: Resource monitoring and visualization of OSGi-based software components. PhD thesis, VTT Technical Research Centre of Finland (2008)

26. Jung, G., Joshi, K., Hiltunen, M., Schlichting, R., Pu, C.: Generating adaptation policies for multi-tier applications in consolidated server environments. Autonomic Computing, 2008. ICAC '08. International Conference on (June 2008) 23–32
27. Brewer, E.A.: Lessons from giant-scale services. IEEE Internet Computing **5**(4) (2001) 46–55