Departamento de Ciências e Tecnologias da Informação

# VISCTE: Visualização Interativa de Serviços e Componentes em Tempo de Execução

Ivo Emanuel Carvalho de Albuquerque

Dissertação submetida como requisito parcial para obtenção do grau de mestre

Mestre em Engenharia Informática

Orientador:
Doutor André Leal Santos, Professor Auxiliar, ISCTE-IUL

Outubro, 2014

Departamento de Ciências e Tecnologias da Informação

# VISCTE: Visualização Interativa de Serviços e Componentes em Tempo de Execução

Ivo Emanuel Carvalho de Albuquerque

Dissertação submetida como requisito parcial para obtenção do grau de mestre

Mestre em Engenharia Informática

Orientador:
Doutor André Leal Santos, Professor Auxiliar, ISCTE-IUL

Outubro, 2014

# VISCTE: Visualização Interativa de Serviços e Componentes em Tempo de Execução,

Ivo Emanuel Carvalho de Albuquerque

Outubro
2014

# Agradecimentos

A realização desta dissertação de mestrado contou com contributos importantes que tornaram a realização deste trabalho possível e pelos quais expresso um grande agradecimento.

Ao Professor Doutor André Santos, orientador da dissertação, agradeço a confiança, o apoio, a partilha do saber e as valiosas contribuições para o trabalho.

À Professora Doutora Isabel Alexandre, docente da cadeira de Introdução à Investigação em Engenharia, pela partilha de conhecimento e orientação prestada durante a fase de investigação desta dissertação.

A todos os meus amigos pelo apoio e incentivo demostrado.

Aos meus pais e irmãos, pelo apoio incondicional que me deram durante toda a minha experiência académica.

À minha namorada, pela ajuda na revisão ortográfica deste trabalho e por tudo o que representa para mim.

# Resumo

Um sistema de larga escala baseado em componentes de software pode ser composto por centenas de componentes que interagem entre si. A visualização de diagramas de sistemas com estas características sofre de problemas relacionados com a dimensão dos diagramas, dado que não é eficiente nem útil manipular diagramas com centenas de nós e ligações. As ferramentas de visualização de componentes de software baseados em OSGi já existentes representam uma informação meramente estática dos componentes e dependências. Estas ferramentas integram mecanismos de foco, como forma de resolver o problema de visualização de centenas de nós e ligações, contudo, o utilizador tem que realizar uma exploração manual do fragmento de componentes que pretende visualizar.

Nesta dissertação é proposta uma solução que passa pela implementação de um mecanismo que produza diagramas de fragmentos do sistema com base em eventos ocorridos em tempo de execução. Existem certos tipos de interações entre componentes que só são determinados em tempo de execução, logo, a análise estática de dependências não fornece uma ajuda satisfatória. O mecanismo proposto pretende, através de uma demonstração de funcionalidade ou casos de teste, registar as interações entre componentes ocorridas durante um período de execução determinado pelo programador, para gerar uma representação diagramática localizada do comportamento do sistema. Foi desenvolvido um protótipo que concretiza o mecanismo proposto sob forma de um *plug-in* para o sistema de desenvolvimento Eclipse, tendo sido o mesmo experimentado de forma bem-sucedida num contexto industrial.

**Palavras-chave:** software, componentes, ferramentas de visualização, OSGi, *plug-in*, Eclipse

# Abstract

A large-scale software system based on components may be composed of hundreds of components, which interact with each other. The diagram visualization of systems with these characteristics normally suffers from problems related to the diagram size, since it is neither efficient nor useful to manipulate diagrams with hundreds of nodes and links. Visualization tools for software components based on existing OSGi generate a static representation of the components and their connections. Visualization tools for software components based on OSGi integrate focus mechanisms as a way to solve the problem of viewing hundreds of nodes and links, however the user must perform a manual exploration of the fragment components that he or she wants to view.

The solution proposed in this dissertation is based on a mechanism that produces diagram fragments based on runtime execution events. There are certain types of interactions between components that are only determined at runtime, so the static analysis of dependencies does not provide a satisfactory aid. The proposed mechanism enables developers, through a demonstration of functionality or test case execution, to obtain a diagrammatic representation of the interactions between components that occurred during a delimited period of time. The proposed mechanism was realized in the form of a plug-in for the Eclipse development enrironment, and the latter was experimented successfully on industrial settings.

**Keywords:** software, components, visualization tools, OSGi, plug-in, Eclipse

# Índice

# Índice Figuras

# Índice Tabelas

# 1 Introdução

## 1.1 Contexto

O crescente uso de software está a aumentar cada vez mais as exigências dos sistemas de informação, tais como, a usabilidade, robustez, confiança, flexibilidade, adaptabilidade e integração. Como tal, o software torna-se cada vez maior e complexo. O principal desafio em engenharia de sistemas é lidar com a complexidade e adaptar-se rapidamente às mudanças. A integração de componentes previamente construídos num sistema de software baseado em componentes, permite uma gestão mais eficaz da complexidade (Crnkovic & Larsson, 2002). Um componente de software pode ser definido como sendo uma peça de código ou um conjunto de tipos de dados abstratos, que são utilizados como uma unidade de uma composição que constitui uma aplicação de software (Luer & Rosenblum, 2001). É constituído por classes que podem ser públicas ou privadas. As classes que representam as interfaces são responsáveis pela comunicação entre componentes e constituem a única parte pública. Todas as outras classes, são privadas e não acessíveis a outros componentes. Desta forma, nenhum conhecimento sobre a estrutura e comportamento de um componente pode ser assumido para uma análise dinâmica (Martin, Giesl, & Martin, 2002).

*Open Services Gateway Initiative* (OSGi) é um conjunto de especificações que definem um modelo de componentes e serviços (McAffer, VanderLei, & Archer, 2010). Existem diversas implementações das especificações OSGi. Equinox é a plataforma de suporte a aplicações desenvolvidas em Eclipse e talvez a implementação OSGi mais utilizada. É considerada uma implementação de referência das especificações do modelo OSGi pela sua robustez e escalabilidade. Um sistema de software baseado no modelo OSGi é constituído por um conjunto de componentes conhecidos por *bundles*. Os *bundles* são auto descritivos, declaram a API pública, definem dependências com outros *bundles* em tempo de execução e escondem a sua implementação interna. Os sistemas de software baseados no modelo OSGi são dinâmicos, no sentido em que os *bundles* que o constituem podem mudar ao longo do tempo. Um *bundle* pode ser instalado, desinstalado e atualizado a qualquer momento. Toda esta alteração de estados surge como um contínuo fluxo de eventos acessível a todos os *bundles*. Por exemplo, quando um novo *bundle* é instalado, outros *bundles* poderão estar interessados nas suas contribuições.

## 1.2 Problema

Os programadores e analistas precisam de um modelo visual de suporte que os ajude a projetar e raciocinar sobre sistemas de software baseado em componentes complexos (Grundy & Hosking, 2000). Estas ferramentas de visualização podem incluir técnicas utilizadas no software convencional, mas necessitam de técnicas adicionais específicas para software baseado em componentes, tais como: (1) visualização da interconexão entre componentes; (2) propagação e manipulação de eventos; e (3) alteração das estruturas dos componentes (Grundy, Mugridge, & Hosking, 1998). Contudo, as ferramentas de visualização ainda contêm alguns problemas que os programadores e analistas enfrentam na visualização de diagramas em grande escala. Estes problemas vão desde a difícil orientação, quantidade limitada de elementos visíveis no ecrã ao mesmo tempo que são mostrados os seus detalhes, detalhes insuficientes ou outros problemas (Holy, Jezek, Snajberk, & Brada, 2012).

A apresentação visual do software baseado em componentes pode ser de dois tipos: estática e dinâmica. A visualização estática consiste na apresentação do comportamento dos componentes de software utilizando o seu código fonte, por outro lado, a visualização dinâmica reflete o comportamento do software em tempo de execução (Martin, Giesl, & Martin, 2002). Os sistemas de software baseados em componentes OSGi são dinâmicos, dado que o conjunto de componentes que o compõe pode sofrer alterações ao longo do tempo. Um componente OSGi pode ser instalado, desinstalado, e modificado a qualquer altura. Estas características proporcionam um sistema modular simples mas poderoso que permite criar outros sistemas de software (McAffer, VanderLei, & Archer, 2010). Os modelos visuais de suporte existentes para o modelo OSGi contêm informações relevantes de um ponto de vista de análise das dependências entre componentes, mas a informação apresentada é apenas estática. Desta forma, os programadores e analistas de software não conseguem saber que componentes estão a ser utilizados e qual o seu comportamento no contexto de determinadas funcionalidades ou casos de teste.

A visualização de software oferece uma forma mais conveniente de descobrir eventuais discrepâncias entre o desenho e a implementação de software. No entanto, como as técnicas de análise de software são raramente ensinadas aos programadores e as ferramentas de análise são muitas vezes separadas das ferramentas de desenvolvimento, os programadores evitam o uso desses ferramentas e as discrepâncias passam despercebidas

(Martin, Giesl, & Martin, 2002). Estas discrepâncias provocam problemas na arquitetura de software, que na maioria das situações não são imediatamente percetíveis, mas que originam problemas a longo prazo e que dificultam a evolução do sistema de software.

## 1.3 Solução

No âmbito desta dissertação foi elaborado um mecanismo de visualização da interação entre componentes de software em tempo de execução, mais concretamente no modelo OSGi. Foi desenvolvida uma prova de conceito que demonstra que é possível reduzir os problemas de visualização existentes, na representação de interação entre componentes em sistemas de software baseados em componentes complexos. A originalidade do mecanismo criado pretende-se com a possibilidade de produzir diagramas de representação de fragmentos do sistema de software, com base em demostrações de funcionalidades ou casos de teste. Desta forma, é possível esconder a informação que não é necessária e tornar os diagramas menos complexos para que os utilizadores possam analisar as relações mais facilmente. O resultado prático da técnica aplicada é o mesmo do estudo realizado por Holy, Jezek, Snajberk e Brada, apesar das suas diferenças. No estudo realizado, os autores defendem que os principais responsáveis pela complexidade gerada nos diagramas de software são os elementos que detêm um grande número de ligações com outros elementos, e de forma a reduzir essa complexidade, esses elementos devem ser movidos para uma área à parte. Desta forma, é possível reduzir o número de ligações presente no diagrama, tornando-o menos complexo e mais compreensível ao olho humano (Holy, Jezek, Snajberk, & Brada, 2012).

As ferramentas de análise de software surgem como um meio de suporte aos programadores e analistas. O estudo realizado por Silito, Murphy e De Volder visa perceber como os programadores compreendem os sistemas de software, e de que forma as ferramentas de análise se deveriam comportar. No estudo mencionado, foi feita a analogia entre um sistema de software e um grafo de elementos. Foram descobertas várias questões que os programadores fazem durante a análise do código de um sistema de software, sobre o qual possuem nenhum ou pouco conhecimento: as primeiras questões focam a descoberta da parte inicial do grafo; as segundas sobre os elementos relacionados com o elemento inicial; as terceiras focam em perceber o número de elementos e as suas relações; as últimas questões baseiam-se na forma em como os elementos do subgrafo adquirido interagem entre si e se relacionam com o resto do sistema (Sillito, Murphy, &

De Volder, 2006). O protótipo desenvolvido faz uso de um mecanismo que viabiliza a criação de diagramas de interação de componentes em tempo execução, para que os programadores e analistas possam determinar as contribuições de cada componente e responder às questões mais endereçadas.

## 1.4 Avaliação

A prova de conceito foi concebida para a tecnologia Equinox, sob forma de um *plug-in*, permitindo aos programadores e analistas de software a geração de diagramas no ambiente de desenvolvimento Eclipse, e oferecendo uma forma útil e conveniente, de mitigar as diferenças de desenho e implementação de software. Elaborou-se um caso de estudo sobre um sistema de componentes de escala industrial com vista a provar se o protótipo desenvolvido é aplicável e útil num contexto complexo e de grande dimensão. O caso de estudo realizado provou que o protótipo desenvolvido é escalável, em contextos de sistemas de componentes de escala industrial, e útil na tarefa de análise de comportamento localizado de sistemas de componentes em tempo de execução.

Em comparação com ferramentas de análise estática de componentes e dependências, o protótipo revela que a abordagem tomada proporciona uma solução viável na redução da complexidade de diagramas de sistemas de componentes de larga escala. O estudo realizado mostrou que o protótipo consegue gerar uma representação localizada do comportamento do sistema com base na execução de uma determinada funcionalidade, e desta forma, demonstra a possibilidade de medir o nível de complexidade inerente às funcionalidades dependendo do nível de complexidade dos diagramas gerados.

## 1.5 Estrutura do documento

O documento encontra-se estruturado da seguinte forma: o Capítulo 2 pretende introduzir e explicar os conceitos relacionados com o software baseado por componentes e da especificação OSGi; o Capítulo 3 sumariza a literatura associada ao tema abordado desta dissertação e os seus trabalhos relacionados; o Capítulo 4, através de um exemplo simples de componentes explica os conceitos chave, o modelo de dados de captura e a forma como os programadores e analistas podem utilizar o mecanismo implementado; o Capítulo 5 introduz de ponto de vista técnico as opções tomadas na fase de desenvolvimento e apresenta todos os componentes que constituem o mecanismo

desenvolvido; o Capítulo 6 recorre à utilização de um caso de estudo, para avaliar a aplicabilidade da ferramenta desenvolvida num contexto de um sistema de software de larga escala; por último, o Capítulo 7 apresenta as conclusões e trabalho futuro no âmbito desta dissertação.

# 2 Engenharia de software baseada por componentes

O conceito de componentes de software foi introduzido pela primeira vez em 1968, por Doug McIlroy, durante a primeira conferência de Engenharia de Software (ES) – *NATO Software Enginnering Conference* (Crnkovic, Stafford, & Szyperski, 2011). Os componentes detêm um lugar preponderante na história da ES e as suas técnicas envolvem alguns princípios fundamentais da engenharia. A abordagem *top-down* visa decompor grandes sistemas em partes mais pequenas – componentes – e a abordagem *botton-up* em compor as pequenas partes de software em grandes sistemas. Mais tarde, em 1998, na conferência de engenharia *International Conference on Software Engineering*, a Engenharia de Software Baseada em Componentes (ESBC) foi introduzida como uma área da ES. Os princípios fundamentais da ESBC são: (1) a construção de software é feita através da composição de componentes de software pré-existentes; (2) os componentes são criados para ser reutilizáveis; (3) a expansão das aplicações de software é feita através da substituição de componentes.

## 2.1 Conceitos

É importante clarificar alguns dos conceitos básicos da ESBC, dado que a diversidade de conceitos e de definições pode gerar confusão, visto que muitos dos conceitos e definições ainda não foram explicados com precisão ou testados na prática (Crnkovic & Larsson, 2002). A ESBC é centrada no conceito componente. Outros conceitos, como interface, contrato ou *framework* estão diretamente relacionados com o desenvolvimento de software baseado por componentes. Nesta secção, será apresentado um sumário destes conceitos, definições e das suas relações.

### *2.1.1 Componente*

Componente é o conceito chave da ESBC. É necessária uma definição precisa de componente de forma a perceber os princípios base da ESBC. A literatura oferece diversas definições, mas a maioria falha ao dar uma definição intuitiva e ao invés disso são focados os aspetos gerais do componente (Crnkovic & Larsson, 2002).

A definição de Szyperski enumera as principais características e princípios que constituem um componente: "*A software component is a unit of composition with*

*contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party*" (Szyperski, 1998). Desta definição, é possível retirar os seguintes princípios: (1) deve existir uma distinção clara entre o ambiente interno e externo do componente; (2) a comunicação entre componentes deve ser feita através de interfaces; (3) o componente deve especificar claramente as suas interfaces e a sua implementação deve ser encapsulada e não acessível através do ambiente externo ao componente.

D'Souza e Wills definem componente como, uma peça reutilizável de software que é desenvolvida de forma independente e que pode ser combinada com outros componentes para produzir maiores unidades de software (D'Souza & Wills, 1999). A especificação de um componente assegura a sua integração, manutenção e atualização, e como tal deve ser composto pelos seguintes elementos:

- Um conjunto de interfaces fornecidas ao, e solicitadas pelo, ambiente que servem como meio de comunicação entre componentes.
- Código executável, que pode ser ligado a código de outros componentes através de interfaces.

### *2.1.2 Interface*

A interface de um componente pode ser definida como a especificação de um ponto de acesso (Crnkovic & Larsson, 2002). Os clientes acedem a serviços fornecidos pelo componente através destes pontos de acesso. Se um componente tem vários pontos de acesso, cada um representa um serviço diferente oferecido pelo componente, e é esperado que existam várias interfaces. Uma interface é apenas uma especificação, e como tal, não oferece uma implementação concreta das suas ações. Esta distinção torna possível alterar a implementação sem alterar a interface e adicionar novas interfaces (e implementações) sem impactos na implementação já existente. Desta forma, é possível melhorar a performance e a adaptabilidade do componente.

Os componentes podem exportar e importar interfaces para ambientes comuns entre componentes. Uma interface exportada descreve os serviços disponibilizados pelo componente ao ambiente, por sua vez, uma interface importada especifica os serviços exigidos pelo componente ao ambiente.

### *2.1.3 Contrato*

As interfaces providenciam apenas informação relativa aos serviços que são fornecidos, omitindo assim a informação sobre o comportamento do próprio componente (Crnkovic & Larsson, 2002). Uma descrição mais assertiva do comportamento do componente pode ser obtida através de contratos. Um contrato contém uma lista de regras que o componente visa manter. Para cada funcionalidade que o componente disponibiliza, um contrato lista todas as regras que devem ser respeitadas pelo cliente (pré-condições) e aquelas que o componente garante respeitar (pós-condições). As pós-condições são regras que se mantêm inalteráveis, e as pós-condições constituem uma definição do comportamento do componente.

A utilização de contratos permite a especificação da interação entre componentes e suporta a reutilização de componentes com base no comportamento dos mesmos. Os contratos permitem aos programadores isolar e especificar, num nível elevado de abstração, as funções dos diferentes componentes num determinado contexto. De notar que, os contratos e interfaces são conceitos diferentes. Uma interface é composta por um conjunto de operações que especifica um serviço fornecido por um componente, enquanto um contrato específica os aspetos comportamentais de um componente ou a interação entre diferentes componentes.

### *2.1.4 Framework*

A ESBC consiste em construir software através da composição de componentes. De modo a tornar este conceito possível, é necessário que exista um contexto onde os componentes possam ser utilizados. As *frameworks* são um meio utilizado para fornecer determinados contextos (Crnkovic & Larsson, 2002).

A literatura oferece diversas definições de *framework*. Uma *framework* pode ser considerada como "esqueleto de um sistema de software*"* (Johnson, 1997) ou "arquitetura que fornece um modelo para sistemas de software" (Jacobson, Griss, & Jonsson, 1997). Tkach e Puttick (Tkach & Puttick, 1995) definem três tipos diferentes de *frameworks*: (1) as *frameworks* técnicas, (2) industriais e (3) aplicacionais. *Smalltalk Model-View Controller (MVC)* é um exemplo prático do primeiro tipo de *framework*, que consiste no conceito de uma vista gráfica que serve para apresentar informação e num controlador que gere todas as manipulações a um determinado modelo de dados (a informação a ser

apresentada). Os outros dois tipos representam dois níveis de modelos de *frameworks*, o primeiro num nível industrial, baseado numa determinada área de negócio, enquanto o segundo é baseado em determinados tipos de domínios aplicacionais como certos problemas de negócio.

O modelo e a *framework* de um componente são dois conceitos que muitas das vezes são confundidos. A sua distinção foi discutida por Bachman (Bachmann, et al., 2000). Um modelo do componente define um conjunto de especificações e convenções utilizados pelo programador do componente, enquanto que a *framework* do componente é uma infraestrutura de suporte ao modelo do componente.

## 2.2 OSGi

Um sistema Java é composto por classes e interfaces que estão organizados por pacotes (*packages*). A linguagem define um conjunto de regras que determinam a gestão da visibilidade entre as várias classes, interfaces, métodos e atributos. No entanto, existem dois problemas com esta estrutura: os *packages* são demasiado granulares para serem considerados módulos e os *Java Archives* (JARs) – tipicamente construídos por vários pacotes – constituem apenas um mecanismo de entrega que não contém semântica em tempo de execução. *Open Services Gateway Initiative* (OSGi) visa resolver estes problemas – com um conjunto de especificações que definem um modelo de componentes e serviços. Estas especificações permitem desenvolver um modelo onde as aplicações são (dinamicamente) compostas por vários componentes reutilizáveis (OSGi, 2014). A Figura 1 mostra um exemplo de uma aplicação de software composta por um conjunto de componentes que possuem dependências entre si. Este conceito tem dois efeitos imediatos: os programadores têm a oportunidade de poder especificar como querem gerir determinados módulos de código e existe informação em tempo de execução para fazer cumprir esses espectativas (McAffer, VanderLei, & Archer, 2010). Nesta secção, vão ser explorados os conceitos base do OSGi e as suas relações. Serão abordados os conceitos *bundle* - a sua estrutura e o seu ciclo de vida – serviços, pontos de extensão e *framework* OSGi, focando essencialmente nos seus princípios fundamentais e modo de operação.

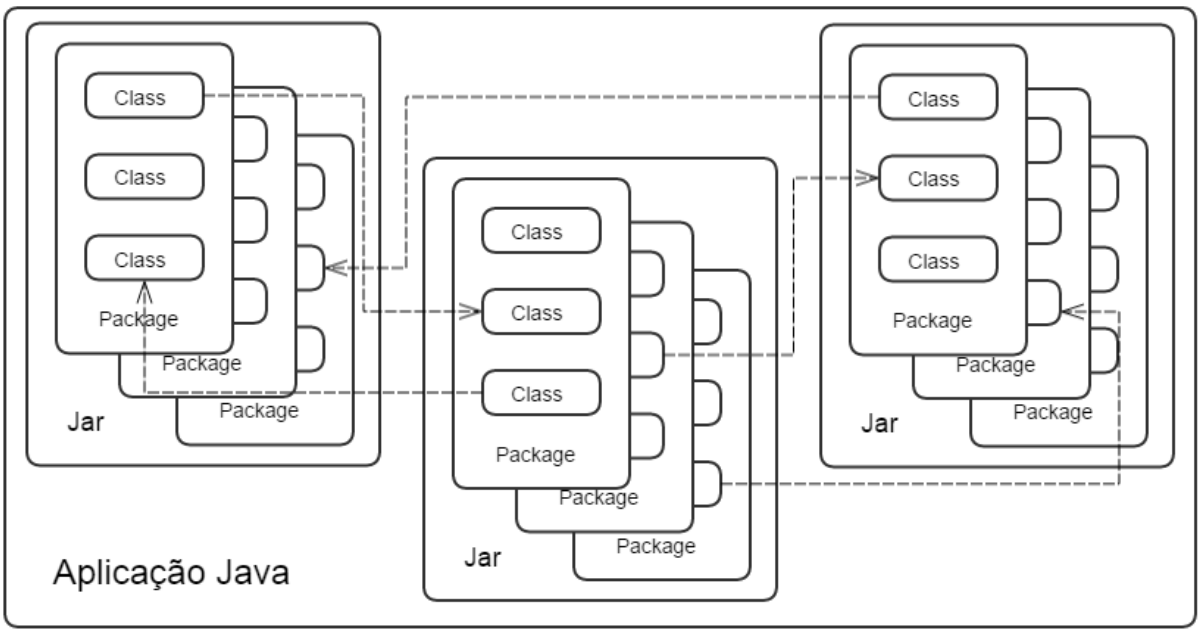



**Figura 1** - Aplicação Java com dependências entre classes.

### *2.2.1 Bundle*

Um sistema OSGi é constituído por um conjunto de componentes conhecidos por *bundles*. Os *bundles* são auto descritivos, declaram uma API pública, definem dependências com outros *bundles* em tempo de execução e escondem a sua implementação interna (McAffer, VanderLei, & Archer, 2010). Como é possível ver na Figura 2, uma aplicação OSGi não possui início nem fim, é simplesmente composta por um conjunto de *bundles*.

Sistemas baseados em OSGi são dinâmicos, dado que o conjunto de *bundles* que o compõe pode sofrer alterações ao longo do tempo. Um *bundle* pode ser instalado, desinstalado, e modificado a qualquer altura. Estas características proporcionam um sistema modular simples mas poderoso para criar sistemas de software adaptáveis. A composição por módulos e os componentes OSGi estão na base do sucesso do Eclipse como plataforma de desenvolvimento da linguagem de programação Java.

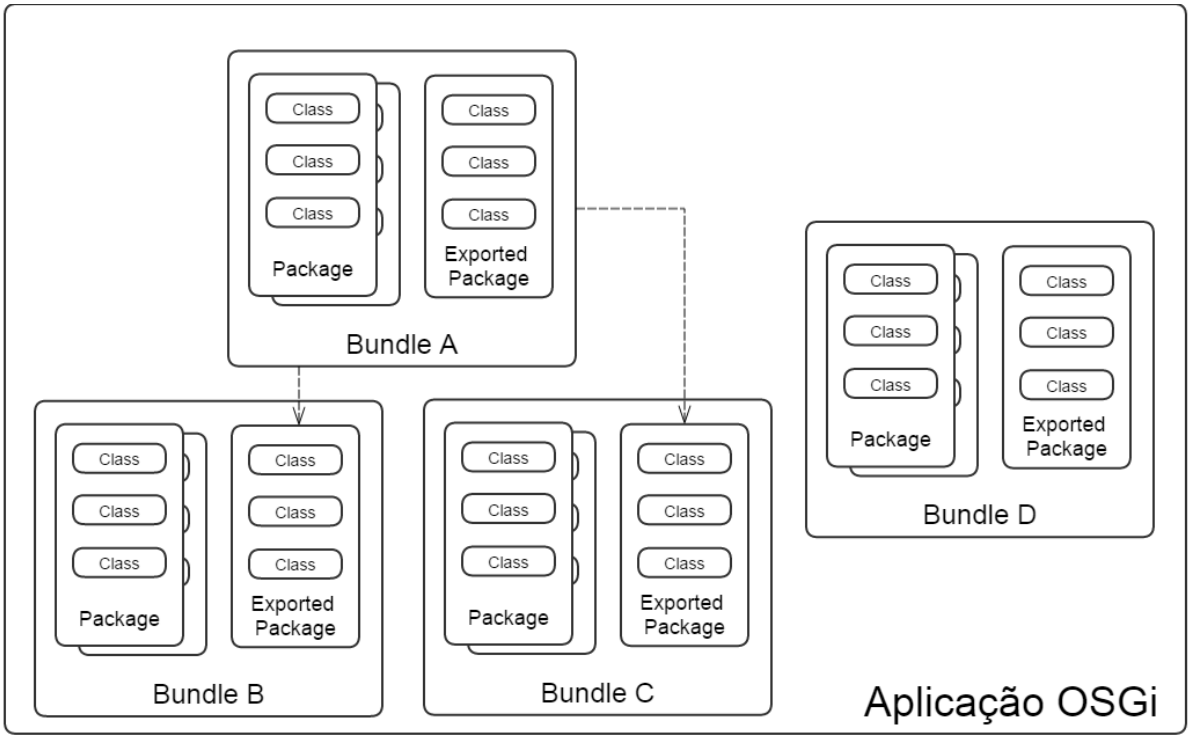



**Figura 2** - Representação de uma aplicação OSGi através de um conjunto interdependente de *bundles*.

#### *2.2.1.1 Anatomia*

Um *bundle* é composto por um conjunto de ficheiros. Os requisitos e contribuições do *bundle* encontram-se definidos no ficheiro META-INF/MANIFEST.MF. O conteúdo do ficheiro MANIFEST.MF do *bundle* `org.eclipselabs.osgi.rtviz.view` tem a seguinte estrutura:

```
(…)
Bundle-SymbolicName:
org.eclipselabs.osgi.rtviz.view;singleton:=true
Bundle-Version: 0.1
(…)
Export-Package: org.eclipselabs.osgi.rtviz.view
Import-Package: org.eclipselabs.osgi.rtviz.bundlespy,
org.eclipselabs.osgi.rtviz.platformlistener
(…)
```

Todos os ficheiros MANIFEST.MF devem conter os cabeçalhos *Bundle-SymbolicName* e *Bundle-Version*. A combinação destes cabeçalhos identifica univocamente o *bundle* na plataforma OSGi. A modularidade do *bundle* encontra-se expressa através dos cabeçalhos *Export-Package* - que define os serviços disponibilizados pelo componente – e o *Import-*

*Package* – que define os serviços necessários para o *bundle* ser lançado na plataforma. Os cabeçalhos adicionais, como o *Bundle-Copyright, Bundle-Name*, e *Bundle-Vendor* são utilizados meramente para efeitos de documentação (McAffer, VanderLei, & Archer, 2010).

#### *2.2.1.2 Ciclo de vida*

O OSGi segue uma lógica de funcionamento fundamentalmente dinâmica. Os *bundles* podem ser instalados, inicializados, parados, e desinstalados num sistema em execução. A Figura 3 oferece uma definição clara do ciclo de vida de um *bundle*.

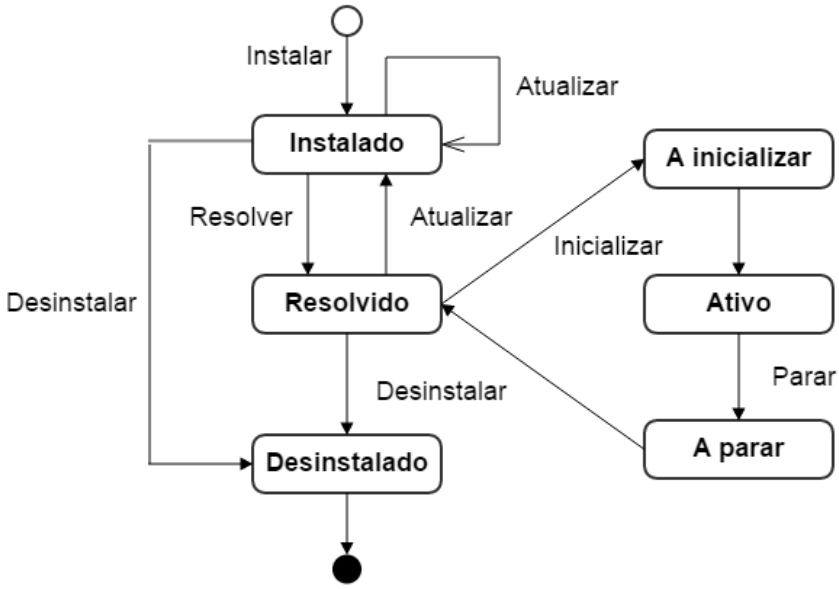



**Figura 3** - Ciclo de vida de um *bundle*.

Todos os *bundles* começam o seu ciclo de vida no estado instalado. Se todas as suas dependências forem atingidas o seu estado muda para resolvido. Quando um *bundle* se encontra no estado resolvido, todas as suas classes podem ser carregadas e executadas. Após o processo de inicialização – estado a inicializar – e da alteração do estado para ativo, o *bundle* fica ligado com o resto do sistema. A partir deste momento, o *bundle* consegue utilizar serviços de outros *bundles*, previamente instalados e inicializados, e as suas contribuições ficam disponíveis a outros *bundles*. Num determinado momento – por exemplo, durante o desligar do sistema – os *bundles* ativos são parados e o seu estado muda para parado. Com o estado parado, os *bundles* podem voltar a ser inicializados ou desinstalados do sistema (McAffer, VanderLei, & Archer, 2010).

Todas as alterações de estados criam um fluxo de eventos. Os *bundles* suportam um comportamento dinâmico através da escuta destes eventos e da resposta que dão às alterações que foram realizadas. Por exemplo, quando um novo *bundle* é instalado, outros *bundles* podem estar interessados nas suas contribuições.

### *2.2.2 Serviços*

Os sistemas baseados em OSGi são compostos por *bundles* auto descritivos como já foi mencionado anteriormente. Os *bundles* podem colaborar diretamente com outros *bundles*, mas para que a colaboração seja possível, tem que existir uma terceira entidade que estabelece a colaboração entre *bundles*. Na plataforma OSGi, o mecanismo que visa realizar esta tarefa é o *service registry*. O *service registry* é responsável por registar e gerir os *bundles* que definem interfaces, *bundles* que implementam e registam serviços, e *bundles* que descobrem e utilizam serviços. O *service registry* gere as colaborações de forma anónima – o *bundle* que providencia um serviço não sabe quem está a usar, e um *bundle* que está a utilizar um serviço não sabe a quem pertence (McAffer, VanderLei, & Archer, 2010).

O aspeto dinâmico do comportamento do serviço é, na maioria das situações, gerido juntamente com o ciclo de vida dos *bundles* envolvidos. Por exemplo, quando um *bundle* é inicializado, descobre, obtém serviços, instancia e regista serviços. Da mesma forma, quando um *bundle* é parado, os serviços que providencia ficam indisponíveis e liberta todos os serviços em utilização. A Figura 4 mostra um exemplo de uma colaboração entre componentes OSGi utilizando serviços.

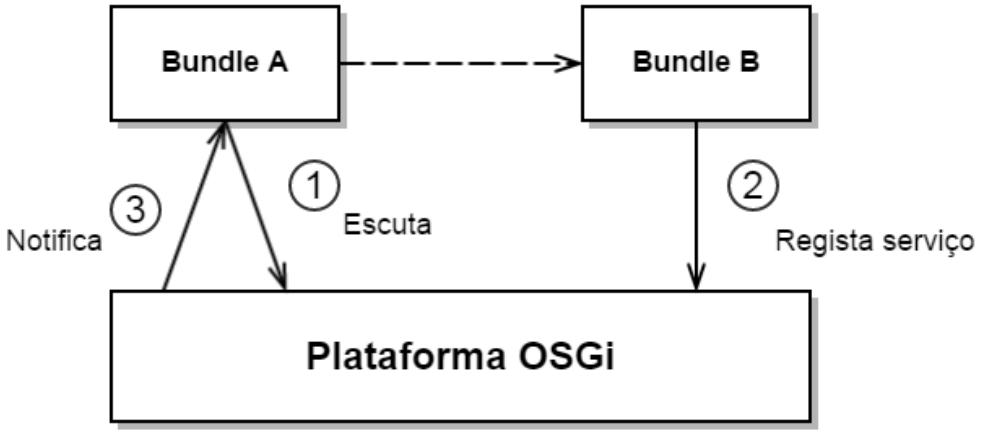



**Figura 4** - Exemplo de colaboração entre *bundles* utilizando serviços.

A figura mostra uma dependência entre *bundles*. Mais concretamente, o *bundle* A depende do *bundle* B e necessita de utilizar um serviço providenciado pelo *bundle* B. Ambos estão instalados na mesma plataforma OSGi, desta forma, e a qualquer momento, o *bundle* A pode escutar os eventos da plataforma e esperar que o *bundle* B registe o serviço que necessita. Dado que, os *bundles* podem ser inicializados a qualquer momento – por exemplo, quando determinada funcionalidade é ativada – após a inicialização, o *bundle* B regista o serviço. Após o registo, a plataforma notifica todos os *bundles* interessados em utilizar o serviço incluindo o *bundle* A, e este passa a utilizar o serviço registado.

### *2.2.3 Implementação OSGi*

Uma aplicação OSGi é composta por um conjunto de um ou mais *bundles* executados numa *framework* OSGi. A *framework* é responsável pela resolução de dependências entre *bundles*, carregamento de classes, registo de serviços, e gestão de eventos (McAffer, VanderLei, & Archer, 2010). A *framework* OSGi é materializada num componente conhecido como *system bundle*. A representação da *framework* OSGi como um *bundle* permite a visualização de toda a plataforma OSGi como um conjunto de *bundles*. O *system bundle* contém todas as características básicas de um *bundle* comum, mas difere no facto em que o seu ciclo de vida não pode ser gerido. Fica ativo quando a *framework* inicializa e continua ativo até que a *framework* é terminada.

Existem quatro implementações da especificação OSGi que se destacam (McAffer, VanderLei, & Archer, 2010):

**Equinox** – Equinox constitui a plataforma base da ferramenta de desenvolvimento Eclipse e é talvez a implementação OSGi mais utilizada. É também considerada uma implementação de referência das especificações OSGi.

**Felix** – Originalmente conhecido como o projeto Oscar, o projeto de código aberto Felix providencia uma implementação da *framework* e serviços OSGi.

**Knopflerfish** – O projeto de código aberto Knopflerfish fornece uma implementação da *framework* OSGi e algumas implementações de serviços.

**mBedded Server** – É uma implementação comercial da *framework* R4.x OSGi da ProSyst. A ProSyst oferece algumas implementações de serviços OSGi.

**Concierge** – Concierge é uma implementação de código aberto da especificação OSGi otimizada para pequenos sistemas de software.

# 3 Estado da arte

Hoje em dia, as aplicações de software conseguem atingir facilmente as centenas ou milhares de componentes, tornando-se maiores e mais complexas. A visualização da estrutura do software em grande escala através de diagramas não é fácil de perceber, devido à complexidade que é gerada através do grande número de elementos e ligações. Apesar das diversas ferramentas disponíveis para ajudar os programadores a desempenhar as suas tarefas, e dos diversos estudos que foram conduzidos para perceber como os programadores compreendem os sistemas de software, pouco é conhecido sobre o tipo de questões específicas que os programadores colocam quando estão a desenvolver código.

Em 2006, Silito, Murphy e De Volder conduziram dois estudos nesta área: um primeiro, em que que envolveu programadores a trabalharem sobre um sistema com o qual não tinham qualquer conhecimento; e um segundo, em que se envolveu o estudo sobre programadores a realizar alterações sobre o seu próprio código (Sillito, Murphy, & De Volder, 2006). A Figura 5 mostra as questões colocadas pelos programadores resultantes da realização dos dois estudos.

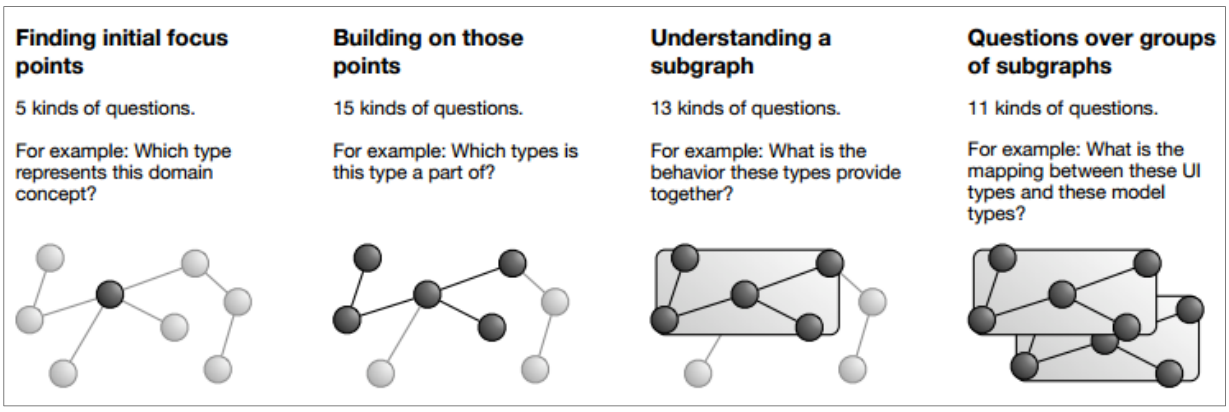



**Figura 5** - Exemplo da aplicação das quatro categorias de questões a um grafo (Sillito, Murphy, & De Volder, 2006).

As questões colocadas foram organizadas em quatro categorias: (1) questões sobre encontrar os pontos iniciais de entrada do sistema; (2) questões sobre a construção desses pontos; (3) questões apontadas para perceber um determinado subgrafo; (4) questões relacionadas sobre vários subgrafos. Considerando o código como sendo um grafo de entidades – métodos e atributos – e as relações entre eles – referências e chamadas aos

métodos e atributos – para responder às questões colocadas, é necessário considerar um determinado subgrafo do sistema. As questões da primeira categoria focam-se em descobrir uma determinada parte inicial do grafo. As questões da segunda categoria são sobre uma entidade e todas as entidades relacionadas. As questões da terceira categoria são sobre perceber o número de entidades e as suas relações. Por fim, as questões sobre a quarta e última categoria são sobre grupos de entidades interligadas entre elas, como se relacionam entre elas e com o resto do sistema.

### *3.1.1 Simplificação da representação de interação entre componentes*

Muito frequentemente, apenas um pequeno número de componentes está interligado com um grande número de outros componentes. Esses componentes são, na maioria das vezes os responsáveis pela complexidade gerada na representação visual, quer porque o utilizador não consegue perceber e identificar outras ligações na mesma área ou porque ocupam demasiado espaço nos diagramas gerados. (Holy, Jezek, Snajberk, & Brada, 2012)

A técnica proposta por Holy, Jezek, Snajberk e Brada, de forma a reduzir a complexidade encontrada nos diagramas de software em grande escala, é mover os componentes que possuem um maior número de ligações para uma área à parte. Desta forma, é possível reduzir o número de ligações presente no diagrama, tornando-o menos complexo e mais compreensível ao olhar humano. Os componentes que são separados mantêm todos os seus detalhes e são escolhidos símbolos para os representar, esses símbolos serão anexados em todos os componentes que possuírem ligações com os componentes em causa (Holy, Jezek, Snajberk, & Brada, 2012).

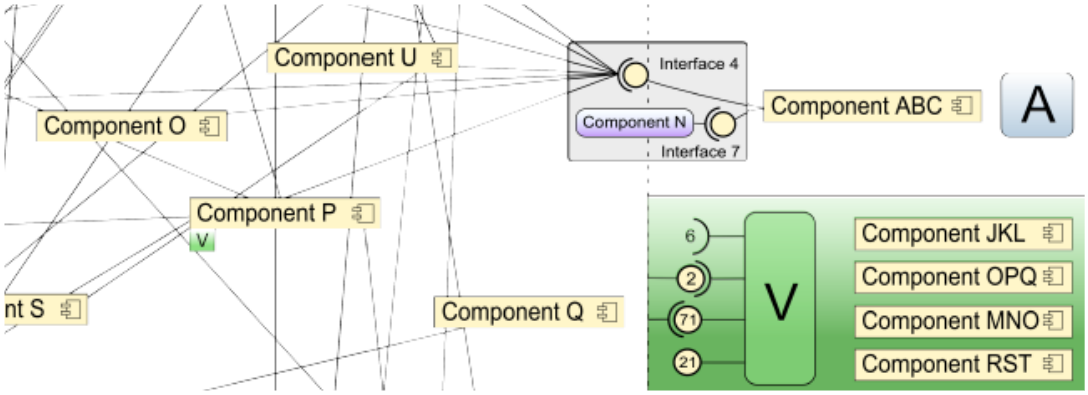



**Figura 6** - Exemplo da aplicabilidade da técnica proposta no contexto de um diagrama de componentes (Holy, Jezek, Snajberk, & Brada, 2012).

### 3.1.2 PDE Incubator Dependency Visualization

*PDE Incubator Dependency Visualization* é uma ferramenta de visualização que tem como objetivo fornecer um conjunto de pontos vista para ajudar com tarefas de análise de dependências entre *plug-ins*. Em particular, os pontos de vista dão apoio cognitivo para pessoas que tentam perceber as dependências entre os seus *plug-ins* (The Eclipse Foundation, 2009). Apesar do facto de a ferramenta estar devidamente integrada numa ferramenta de desenvolvimento (Eclipse) sob forma de *plug-in*, de fornecer informações relevantes de um ponto de vista de análise das dependências entre *plug-ins*, a informação apresentada é apenas estática. Os engenheiros de software não conseguem saber que componentes estão a ser utilizados e qual o seu comportamento no contexto de uma determinada funcionalidade.

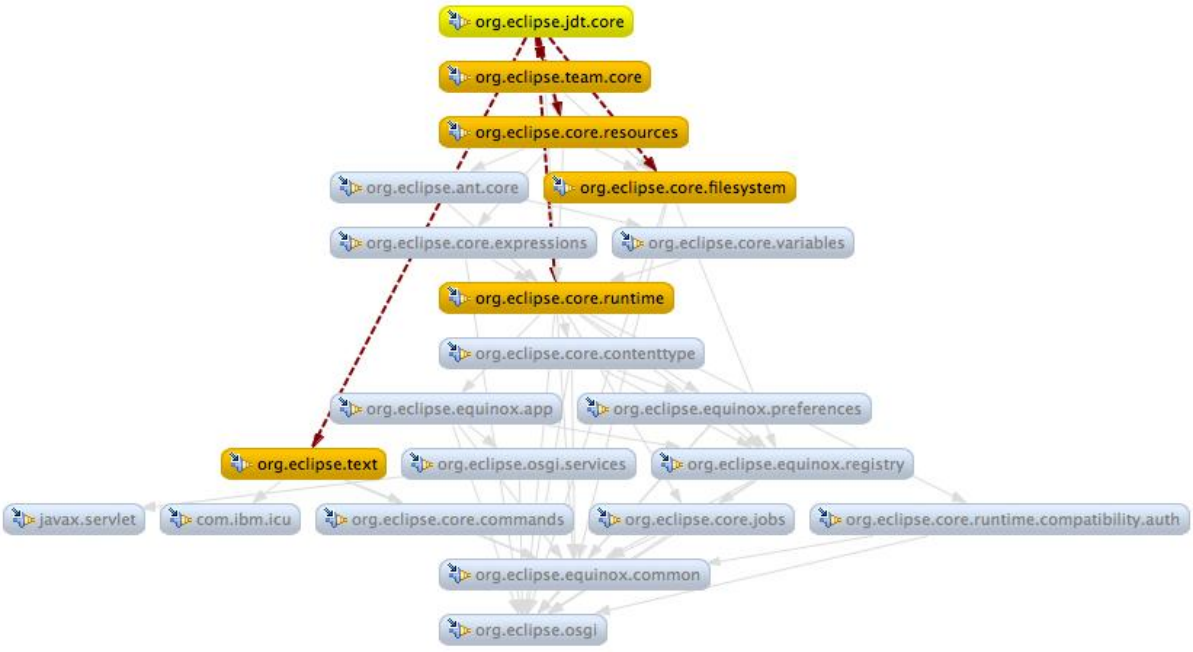



**Figura 7** - Interface da ferramenta PDE Incubator Dependency Visualization. (The Eclipse Foundation, 2009)

### 3.1.3 AIVA

*Advanced-Interactive Visualization Approach* (AIVA) é a implementação de uma abordagem, que consegue representar visualmente as estruturas de dados de uma forma percetível e ao mesmo tempo manter um acesso fácil dos detalhes. É utilizada uma

representação gráfica que permite visualizar os seus componentes e as suas relações. A sua notação é similar ao UML, mas mais simplificada, mostrando apenas informação relevante no âmbito de software baseado por componentes. Para simplificar a representação visual são escondidas informações que o utilizador não deseja visualizar e são mostrados detalhes dos componentes apenas a pedido do utilizador (Snajberk, Holy, & Brada, 2013).

A ferramenta aplica conceitos e técnicas interessantes no contexto da representação visual da interação de componentes em software: a integração da ferramenta sob forma de um *plug-in* para o Eclipse, para ser mais acessível aos utilizadores e incentivar ao seu uso; a utilização de uma notação simplificada do UML para componentes de software; a informação representada visualmente conter apenas a informação pretendida pelo utilizador. Contudo, a aplicabilidade desta ferramenta está restrita ao uso de uma análise estática do código fonte do software, e não permite realizar uma análise em tempo de execução.

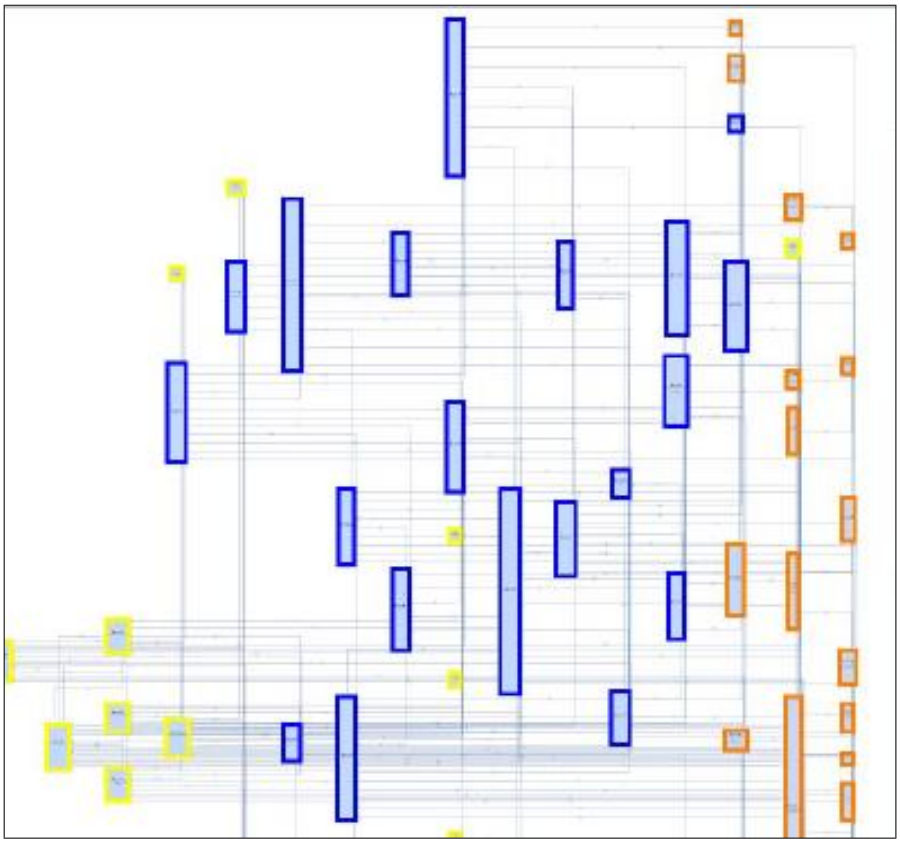


**Figura 8** - Interface da ferramenta AIVA (Snajberk, Holy, & Brada, 2013).

### *3.1.4 HOTAGENT*

HOTAGENT apresenta-se como uma ferramenta de desenvolvimento e visualização de software baseado por componentes. O modelo de componentes utilizados pela ferramenta HOTAGENT não é um modelo conhecido, como o EJB (*Enterprise Java Beans*), CCM (*CORBA Component Model*) ou COM+ (*Common Object Model*), ao invés disso, é utilizado um modelo de componentes próprios e simplificado. A componente de visualização da ferramenta HOTAGENT gera uma representação visual, que visa analisar o comportamento em tempo de execução de um software baseado por componentes previamente construídos pela ferramenta HOTAGENT (Martin, Giesl, & Martin, 2002).

A ferramenta HOTAGENT utiliza o seu próprio modelo de componentes de forma a simplificar a informação relevante e a representação visual. A representação da interação entre os componentes de software, é feita na análise dos comportamentos dos componentes em tempo de execução. Contudo, a utilização da ferramenta HOTAGENT não é viável no contexto de modelos de componentes mais conhecidos, como por exemplo o modelo OSGi.

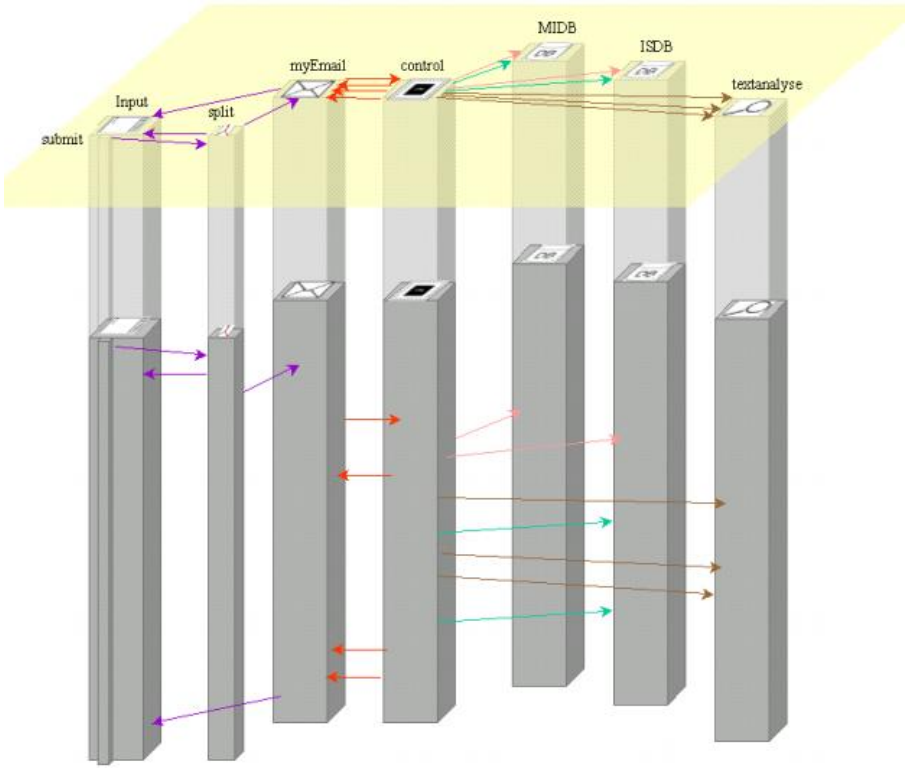



**Figura 9** - Interface da ferramenta de visualização HOTAGENT (Martin, Giesl, & Martin, 2002).

### 3.1.5 *SoftArch*

*SoftArch* é uma ferramenta que providência uma visualização estática e dinâmica das arquiteturas de software tradicionais. A visualização dinâmica é gerada sob forma de anotação e/ou animação, para dar mais informação às formas visuais geradas pela visualização estática. *SoftArch* utiliza um modelo de meta-dados dos diversos tipos de componentes da arquitetura de um software, que podem ser modelados e representados de uma forma visual (Grundy & Hosking, 2000).

Os autores salientam a importância da análise estática e dinâmica e a ferramenta criada serve os propósitos mencionados. A ferramenta *SoftArch* utiliza diagramas para representar a arquitetura de software de um ponto de vista estático e dinâmico. Contudo, a ferramenta foi criada para dar suporte às arquiteturas de software tradicional e não serve os propósitos do software baseado por componentes.

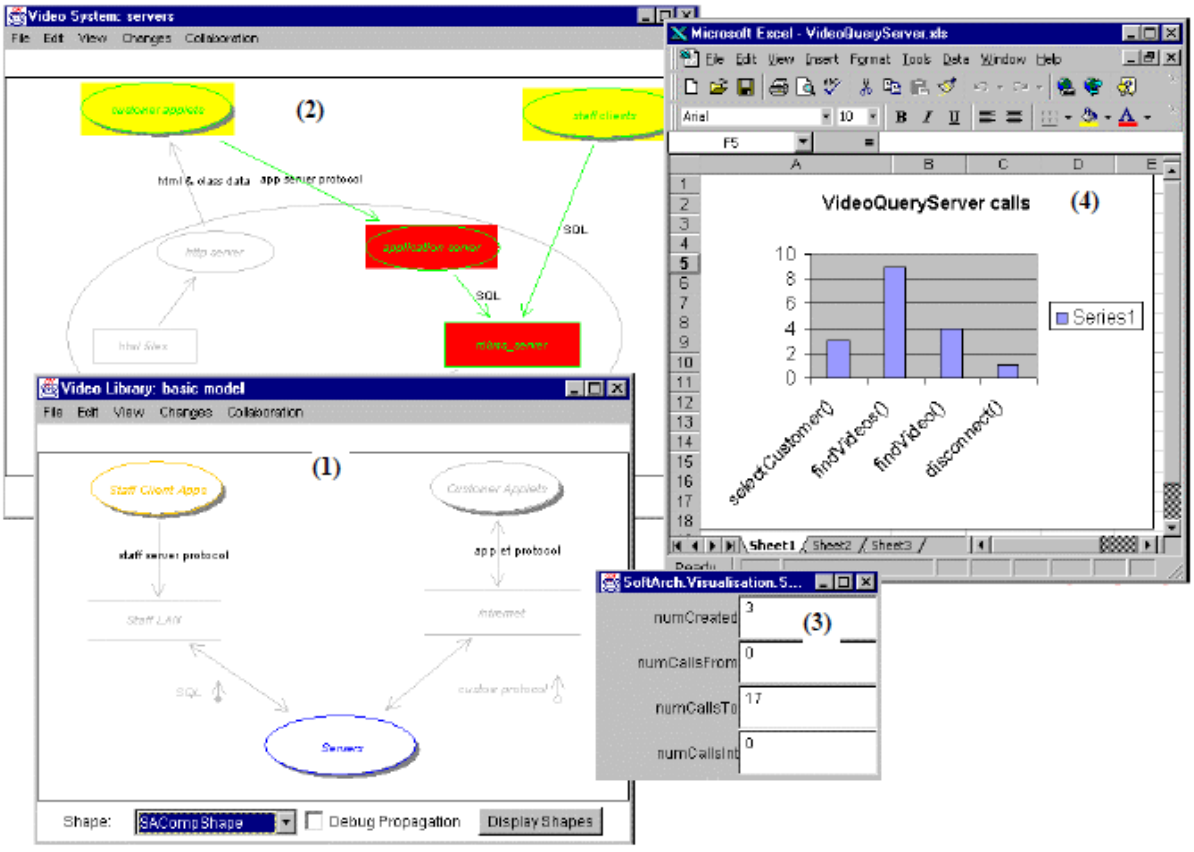



**Figura 10** - Interface da ferramenta SoftArch (Grundy & Hosking, 2000).

### 3.1.6 CoCA-Ex

O trabalho apresentado apresenta a ferramenta *Complex Component Applications Explorer* (CoCA-Ex). A ferramenta CoCA-Ex pretende resolver problemas relacionados com a grande complexidade de diagramas de componentes, em sistemas de software em grande escala. Os autores defendem que a complexidade de visualização de componentes de sistemas de larga escala está, muitas das vezes, associada a um conjunto de componentes que se encontra ligado a um grande número de componentes (Holy, Snajberk, Brada, & Jezek, 2013).

A técnica proposta pretende reduzir a complexidade visual através da remoção dos componentes com um grande número de ligações do diagrama principal para uma área à parte. Quando o utilizador move os componentes do diagrama principal para a área à parte, as linhas que representam as ligações do componente são ocultadas e o componente é considerado um componente conhecido. Após a remoção do componente, o utilizador pode continuar a explorar o sistema de componentes. Com esta abordagem, os autores conseguem reduzir o número de ligações do diagrama gerado, tornando-o menos complexo e mais compreensível ao olhar humano. Contudo, a abordagem dos autores visa analisar o sistema de componentes de uma perspetiva estática das relações entre os componentes. Desta forma, não é possível gerar uma representação localizada do comportamento do sistema em análise.

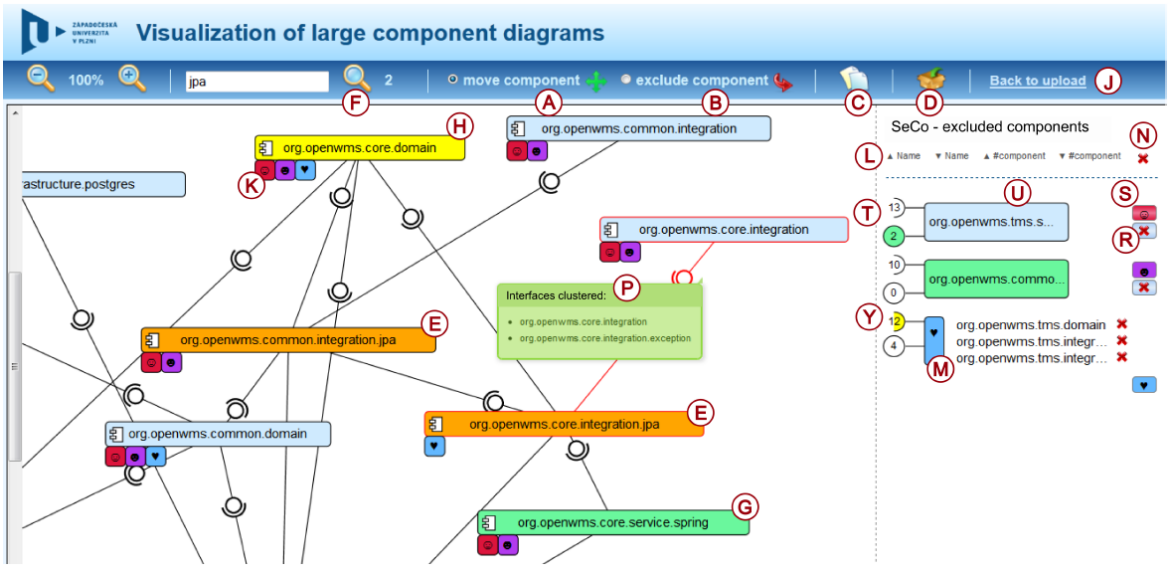



**Figura 11** - Interface da ferramenta CoCA-Ex (Holy, Snajberk, Brada, & Jezek, 2013).

# 4 Conceito

Um sistema de larga escala baseado em componentes de software pode ser composto por centenas de componentes que interagem entre si. A visualização de diagramas de sistemas com estas características sofre de problemas relacionados com a dimensão dos diagramas, dado que não é eficiente nem útil manipular diagramas com centenas de nós e ligações. O mecanismo de Visualização Interativa de Serviços e Componentes em Tempo de Execução (VISCTE) visa solucionar este problema, através da produção de diagramas de fragmentos do sistema com base na execução do mesmo.

Existem certos tipos de interações entre componentes que só são determinados em tempo de execução, logo, a análise estática de dependências não fornece uma ajuda satisfatória. O mecanismo VISCTE pretende, através de uma demonstração de funcionalidade ou casos de teste, indicar as interações entre componentes ocorridas durante um período de execução, para gerar uma representação localizada do comportamento do sistema.

## 4.1 Exemplo

A forma mais simples de explicar os conceitos do mecanismo VISCTE é através de um exemplo simples de componentes. O exemplo escolhido baseia-se numa plataforma de transferência de imagens pela Internet. A plataforma é composta por vários elementos, entre os quais dois servidores web (A e B), três servidores de imagens – alta, média e baixa resolução – e um controlador de performance. A plataforma tem à sua disposição dois servidores web e três servidores de imagem, que oferecem redundância e uma alternativa em situações de aumento de tráfego para garantir velocidades de transferência razoáveis. O controlador de performance é responsável por gerir a escolha de servidores de imagem em função das velocidades de transferência praticadas e a sua utilização é opcional. Os servidores web só podem utilizar um servidor de imagem de cada vez. A escolha do servidor de imagem recai sobre os servidores web quando o controlador de performance não é utilizado. Neste cenário, existe uma dependência estática entre o servidor web e o servidor de imagem escolhido. Quando o controlador de performance é utilizado, a dependência entre os servidores web e os servidores de imagem é dinâmica. A escolha dos servidores de imagem pode variar ao longo do tempo em função das

velocidades de transferência – por exemplo, se o controlador de performance necessitar de garantir velocidades de transferência razoáveis devido a um aumento de pedidos de transferência, pode alterar entre o servidor de imagem de alta resolução para o servidor de imagem de média resolução. A Figura 12 representa os elementos da plataforma sob forma de componentes e as suas dependências estáticas.

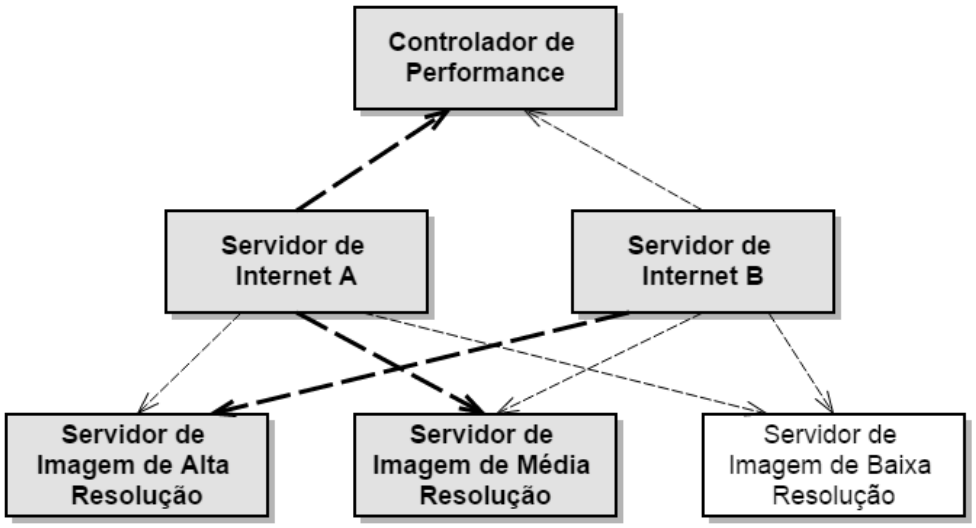



**Figura 12** - Exemplo de componentes e dependências estáticas da plataforma de transferência de imagens.

A figura mostra os componentes que compõem o sistema de componentes do exemplo da plataforma de transferência de imagens e as suas dependências estáticas. Os componentes e dependências, evidenciados a sombreado e negrito respetivamente, constituem um exemplo concreto do sistema de componentes em tempo de execução. O exemplo escolhido mostra que a determinado ponto, o servidor de Internet A está a utilizar o controlador de performance e o servidor de imagem de média resolução. Por sua vez, o servidor de Internet B não utiliza o controlador de performance e utiliza o servidor de imagem de alta resolução. A representação estática de componentes e das suas dependências oferece informação útil mas é limitada devido à dinâmica do modelo OSGi. Este tipo de representação oferece informação sobre os componentes existentes num sistema baseado no modelo OSGi e das suas dependências, contudo, não consegue transmitir toda a veracidade comportamental dos componentes envolvidos. Pré-condições como, um servidor de Internet pode utilizar apenas um servidor de imagem de cada vez ou que a utilização do controlador de performance é opcional, são impossíveis de determinar após a análise estática de componentes e das suas dependências. Uma representação em

tempo de execução (dinâmica) é muito mais rica em informação. Existem certos tipos de interações entre componentes que só são determinados em tempo de execução – por exemplo, um servidor de Internet utiliza apenas um servidor de imagem de cada vez – logo, a análise estática de dependências não fornece uma ajuda satisfatória. O mecanismo VISCTE permite, através de uma demonstração de funcionalidade ou casos de teste, a visualização das interações entre componentes ocorridas durante um período de execução, para gerar uma representação dinâmica e localizada do comportamento do sistema.

## 4.2 Modelo de dados

Existe um fluxo de eventos de componentes e serviços durante a execução de um sistema de componentes baseados no modelo OSGi. Os eventos gerados podem ser utilizados para representar a interação dos componentes e serviços com o resto do sistema. O mecanismo implementado visa escutar e capturar alguns eventos chave para gerar uma representação em tempo de execução de interação entre componentes: inicialização e término de um componente; registo e cancelamento de um serviço; obtenção e libertação de um serviço. A Figura 13 representa o modelo de dados utilizado para capturar esta informação com a notação UML.

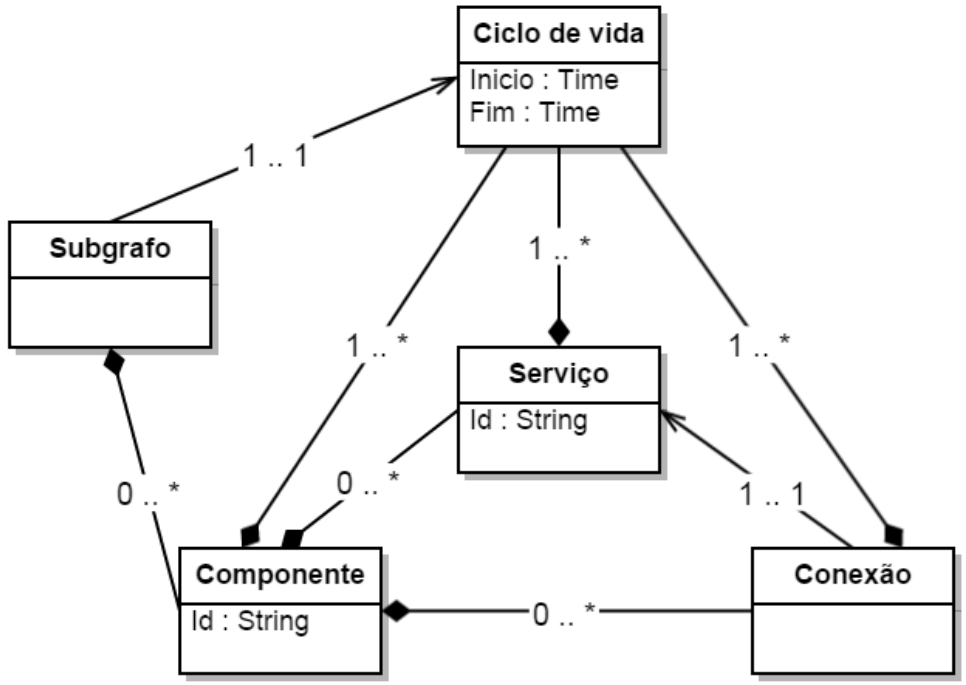



**Figura 13** - Modelo de dados em notação UML utilizado na captura do fluxo de eventos gerados pelo sistema de componentes baseados no modelo OSGi.

O modelo de dados é composto por vários elementos e está centrado na representação de interação de um fragmento de componentes durante um dado período de tempo. Esse período de tempo é chamado período de captura. Durante este período, o sistema baseado no modelo OSGi gera eventos que notificam alterações de estado de componentes e serviços, e o modelo de dados do mecanismo implementado é responsável por guardar essa informação. Fora desse período, todos os eventos são descartados.

O componente é o elemento nuclear do modelo de dados utilizado. O diagrama gerado será composto apenas por componentes e pelas suas relações. Um componente é adicionado ao modelo quando é gerado um evento a notificar a inicialização de um componente, assim como, é removido quando surge um evento que indique o término do mesmo. O mesmo componente pode ser inicializado e terminado várias vezes durante o período de captura. O elemento ciclo de vida permite que o modelo de dados guarde informação das várias vezes que um componente, serviço ou obtenção de serviço – conexão – é adicionado ou removido do sistema. Um componente pode disponibilizar e adquirir vários serviços após a sua inicialização. O registo de um serviço é representado pela relação componente-serviço e a obtenção de um serviço registado por um outro componente, é representada através da relação componente-conexão-serviço.

O modelo de dados do mecanismo VISCTE é atualizado quando a plataforma OSGi do sistema de componentes em análise notifica um dos seguintes eventos: inicialização de componentes, término de componentes, registo de serviços, cancelamento de serviços, obtenção de serviços e libertação de serviços. O exemplo da Secção 4.1 permite explicar as alterações no modelo de dados resultantes dos eventos gerados pela plataforma OSGi. Quando os servidores de Internet (A e B), o controlador de performance e os servidores de imagem de alta, média e baixa resolução são inicializados, a plataforma OSGi gera eventos de inicialização para cada componente respetivo. Após a receção destes eventos, e para cada evento rececionado, é criado um objeto componente e um objeto ciclo de vida no modelo de dados. Cada objeto componente contém um identificador (Id) que corresponde ao *bundle symbolic name* do respetivo componente. Por sua vez, cada objeto ciclo de vida contém uma data de início de uma data de fim. A data de início é preenchida na criação do componente, com a data do evento de inicialização rececionado, e a data de fim é preenchida quando for recebido um evento de término associado ao componente. No entanto, se o componente for inicializado novamente após o seu término, um novo objeto ciclo de vida é criado no modelo de dados. O componente controlador de performance regista um serviço, que permite determinar o melhor servidor de imagem a utilizar em

função da velocidade de tráfego praticada, durante o processo de inicialização. Consequentemente, todos os componentes interessados recebem uma notificação do registo do dado serviço. O mecanismo VISCTE, após a receção da notificação de registo do serviço, cria um objeto serviço e um objeto ciclo de vida no modelo de dados e associa o objeto serviço ao componente previamente criado. O objeto ciclo de vida é associado ao serviço criado e contém as mesmas características que um objeto ciclo de vida associado a um componente explicado anteriormente, isto é, a data de início é preenchida na receção do evento de registo de serviço e a data de fim quando o serviço é cancelado. Os servidores de Internet, interessados em utilizar o serviço registado, requisitam a sua utilização. O evento de aquisição de serviços é considerado um evento chave, e deste modo, o mecanismo atualiza o modelo de dados e cria um objeto conexão, associado ao componente que requisitou a utilização do serviço e associado ao serviço requisitado, e cria um objeto ciclo de vida associado ao objeto conexão com as mesmas características que os objetos ciclo de vida anteriores.

## 4.3 Interface com o utilizador

O protótipo VISCTE faz uso de uma interface minimalista com o único objetivo de se tornar autoexplicativa e intuitiva. O protótipo foi desenvolvido sob forma de uma vista e de um *plug-in* que integra o ambiente de desenvolvimento Eclipse. O *plug-in* tem as mesmas características que uma vista tradicional do Eclipse, podendo o utilizador adicionar, remover e movimentar a vista a qualquer momento. A Figura 14 mostra a interface gráfica do protótipo VISCTE, que é composta por vários elementos: (1) área de desenho, que tem o propósito de representar o subgrafo do sistema de componentes OSGi em análise sob forma de um diagrama de componentes e dependências; (2) componentes que representam os *bundles* em execução do sistema em análise; (3) legendas que contêm informação adicional dos componentes e serviços (4) dependências que representam a utilização de serviços entre componentes; (5) botão de captura, que permite ao mecanismo inicializar ou parar o processo de criação e representação de um subgrafo do sistema; (6) botão de recuo no histórico de captura, que possibilita a navegação para trás no histórico de interações entre componentes capturada durante o período de captura; (7) botão de avanço no histórico de captura, que possibilita a navegação para a frente no histórico de interações entre componentes.

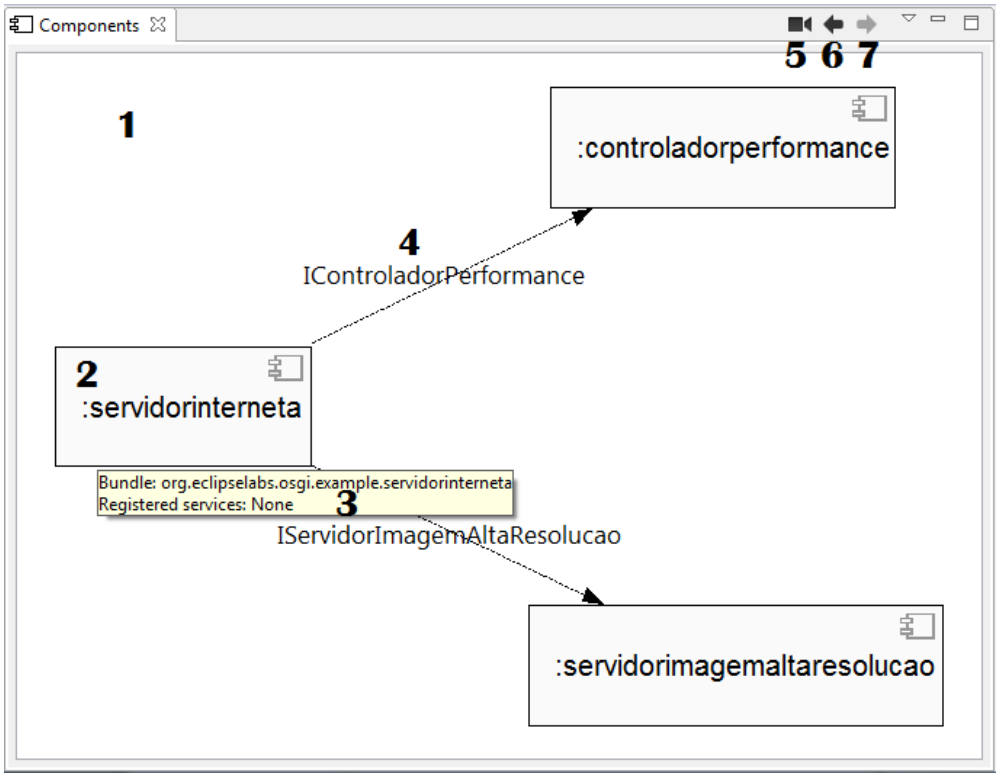



**Figura 14** - Interface gráfica do protótipo VISCTE sob forma de uma vista no ambiente de desenvolvimento Eclipse.

O protótipo desenvolvido possibilita a ativação e desativação do modo de captura. Durante o modo de captura, guarda todo o histórico dos eventos chave gerados pela plataforma OSGi e as resultantes interações dos componentes e serviços com o sistema. Este mecanismo permite aos programadores e analistas isolar os fragmentos de sofware pretendidos após a execução de determinadas funcionalidades ou casos de teste.

Os comandos de navegação permitem que, após uma série de eventos e interações, os programadores e analistas possam rever todo o histórico de interações, passo a passo. Esta ação é bastante útil por várias razões. Na maioria das vezes, as interações entre os componentes resultantes da demostração de uma funcionalidade, são tão rápidas que não são percetíveis à visão humana. Permite também, associar uma sequência de interações de componentes a um caso de uso para uma análise posterior.

A representação dos componentes que interagem e compõem o sistema de componentes a analisar é feita utilizando a notação UML componentes. A identificação unívoca de um componente é feita recorrendo ao *bundle symbolic name*. Dado que, em alguns casos, o *bundle symbolic name* pode atingir um número de caracteres muito elevado – por exemplo, `org.eclipselabs.osgi.example.servidorinterneta` – é adotada uma

simplificação do identificador original. A simplificação adotada pelo mecanismo VISCTE visa representar os componentes através da última parte do *bundle symbolic name* respetivo, por exemplo o *bundle symbolic name* `org.eclipselabs.osgi.example.servidorinterneta` é representado por `servidorinterneta` e a descrição completa do *bundle symbolic name* está disponível através de uma legenda disponibilizada pelo protótipo, tal como a informação dos serviços registados pelo componente, se existirem.

A notação UML utilizada para representar dependências foi escolhida para descrever as interações entre componentes, por exemplo, quando o componente A obtém um serviço providenciado pelo componente B, o componente A passa a depender do componente B. Existe uma relação cliente-fornecedor em todas as interações entre componentes que envolvam a utilização de serviços de outros componentes. A dependência de um componente para outro, pode simbolizar a utilização de um ou mais serviços. A representação das dependências entre componentes é feita através de uma seta a tracejado – notação UML – e identificada através do identificador do serviço ou serviços utilizados pelo componente cliente. Como os serviços possuem um identificador único que se assemelha ao *bundle symbolic name*, o protótipo desenvolvido procede à mesma simplificação que é feita nos identificadores dos componentes, isto é, é apenas demonstrada a última parte do identificador.

## 4.4 Cenários de utilização

A demostração da aplicabilidade do mecanismo implementado sobre o exemplo fictício, a plataforma de transferência de imagens pela Internet, vai servir para complementar alguns dos conceitos introduzidos ao longo deste capítulo. Procedeu-se à escolha de um conjunto de cenários concretos, que envolvem a geração de eventos indispensáveis para a compreensão da aplicabilidade do protótipo desenvolvido, e para a representação gráfica de interação dos componentes envolvidos no sistema de componentes baseados no modelo OSGi.

### *4.4.1 Inicialização de um componente*

Um dos componentes da plataforma de transferência de imagens pela Internet é o controlador de performance. O controlador de performance não depende de mais nenhum

componente, e como não possui nenhuma dependência, pode ser resolvido e inicializado sozinho. O cenário escolhido envolve a geração do evento de inicialização do componente controlador de performance sobre um ambiente OSGi sem componentes inicializados. A Figura 15 mostra o diagrama de componentes, composto por um único componente, gerado pelo VISCTE quando aplicada ao cenário descrito.

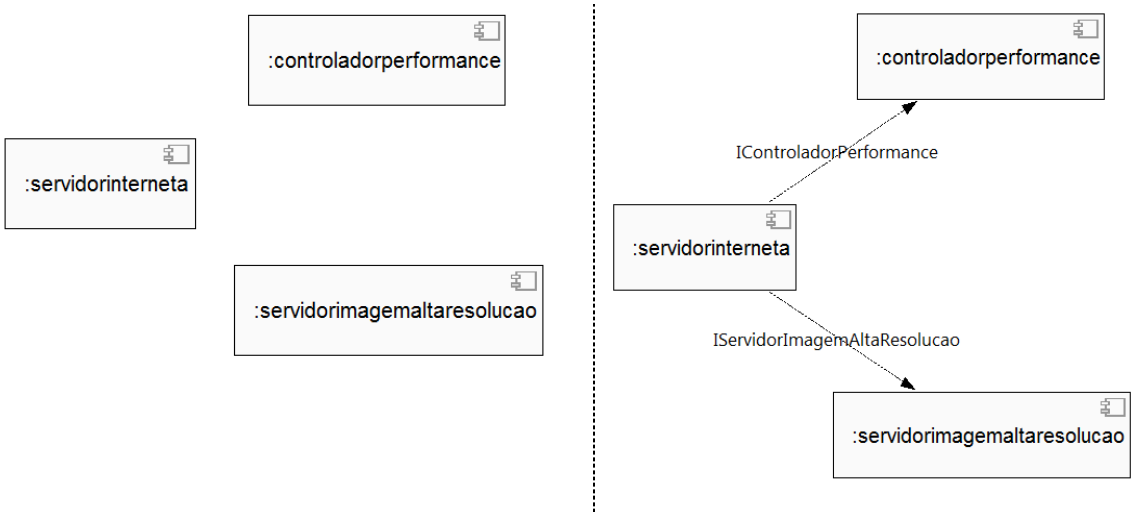

**Figura 15** – Representação de um componente e respetiva legenda num cenário de inicialização de um único componente.

É possível identificar a representação do controlador de performance que constitui o único componente inicializado no ambiente do sistema OSGi em análise. O componente é representado pelo *bundle symbolic name* simplificado – `controladorperformance` – e a legenda oferece a informação do *bundle symbolic name* completo e dos serviços registados pelo componente. A visualização da inicialização de componentes em tempo de execução é bastante útil, dado que, o conceito de *lazy loading* é bastante comum em sistemas de componentes baseados no modelo OSGi. O conceito *lazy loading* implica que, por vezes, determinados componentes apenas são instalados e inicializados quando determinadas ações ocorrem. Por exemplo, no sistema fictício da plataforma de transferência de imagens, os componentes correspondentes aos servidores de imagens só são carregados quando os servidores de Internet decidem utilizar os seus serviços.

### *4.4.2 Aquisição de serviços*

A utilização dos serviços disponibilizados pelos servidores de imagens – baixa, média ou alta resolução – por parte dos servidores de Internet constitui um dos requisitos para a plataforma de transferência de imagens pela Internet estar funcional. Os servidores de imagem disponibilizam o mesmo serviço – acesso às imagens – a única diferença está na resolução das imagens. Inicialmente, os servidores de Internet escolhem os servidores de imagem de alta resolução, mas esta escolha pode variar em função da velocidade do tráfego, se os servidores de Internet estiverem a utilizar o serviço de gestão de tráfego

disponibilizado pelo controlador de performance. Para demonstrar a utilização do protótipo VISCTE aplicada num cenário, onde existam registo e aquisição de serviços, foi escolhido um exemplo de inicialização da plataforma de transferência de imagens fictício. No exemplo de inicialização escolhido ocorreu a seguinte sequência de eventos: (1) inicialização do servidor de Internet A; (2) inicialização do servidor de imagem de alta resolução; (3) servidor de Internet requisita o serviço de gestão de tráfego providenciado pelo controlador de performance; (4) aquisição do serviço de acesso de imagens disponibilizado pelo servidor de imagens de alta resolução pelo servidor de Internet A. A Figura 16 mostra uma representação visual interativa dos componentes com o auxílio do protótipo desenvolvido. Dado que, a sequência de eventos descrita no cenário atual é muito rápida, e a visualização de todas as interações entre componentes no diagrama gerado pelo protótipo ISCTE pode ser impercetível ao olho humano, o acesso de um fragmento no tempo torna-se uma funcionalidade bastante útil. O protótipo permite a qualquer momento, através da funcionalidade de *playback*, a possibilidade de aceder ao histórico dos diagramas gerados através dos botões de navegação embutidos na interface gráfica. O antes representado na tabela abaixo, mostra um fragmento do sistema no tempo, onde todos os componentes se encontram inicializados e os serviços registados. Por sua vez, o depois mostra o fragmento final após a aquisição de serviços descrita do cenário da secção atual.

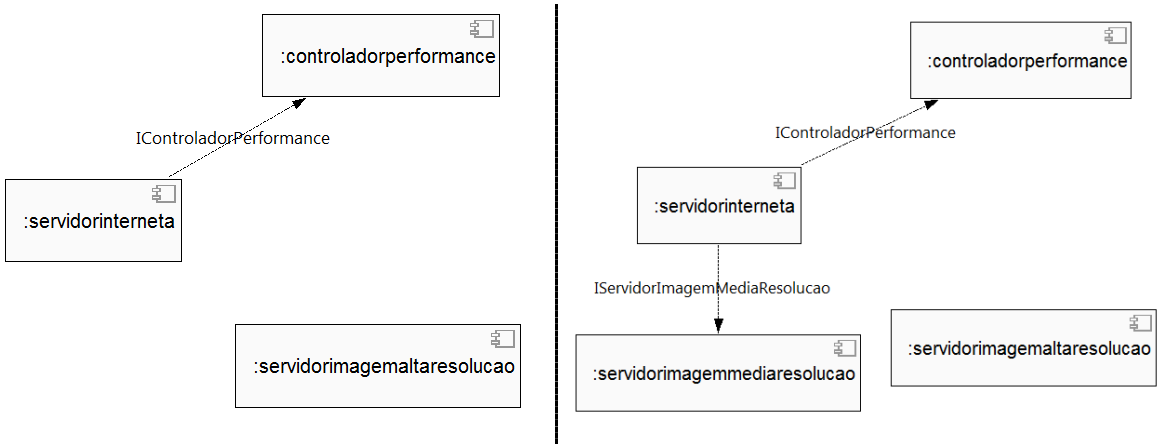



**(Antes)** **(Depois)**

**Figura 16** - Evolução do diagrama gerado pelo protótipo VISCTE após a aquisição de serviços.

### 4.4.3 *Libertação de um serviço e aquisição de outro*

A escolha do servidor de imagens a ser utilizado pode variar em função da velocidade do tráfego, quando os servidores de Internet estão a utilizar o serviço de gestão de tráfego disponibilizado pelo controlador de performance. Nesta situação, o controlador de performance verifica periodicamente as velocidades de tráfego associadas às ligações dos servidores de Internet. O controlador de performance pode, a qualquer momento, decidir que é necessário alterar a resolução das imagens para melhorar a velocidade de tráfego praticada pelos clientes. Neste cenário, vai-se assumir que a velocidade de tráfego diminuí devido ao aumento das ligações ao servidor de Internet A. De seguida, o controlador de performance apercebe-se da alteração, inicializa o servidor de imagens de média resolução e dá a indicação ao servidor de Internet A para libertar o serviço do servidor de imagem de alta resolução e passar a utilizar o serviço do servidor de média resolução. A Figura 17 representa a evolução do diagrama gerado que representa o fragmento do sistema de *bundles* que está a ser analisado. O fragmento à esquerda (Antes) representa um fragmento do sistema no tempo após a libertação de um serviço. É possível observar que a dependência que existia entre o servidor de Internet A e o servidor de imagens de alta resolução desapareceu. O fragmento à direita (Depois) representa o sistema de *bundles* após o servidor de Internet adquirir o serviço do servidor de imagens de média resolução.




**(Antes)** **(Depois)**

**Figura 17** - Representação iterativa dos fragmentos do sistema de bundles após a libertação e aquisição de um serviço.

### 4.4.4 *Término de um componente*

Num cenário normal, são utilizados dois servidores de Internet para oferecer uma alternativa quando as velocidades práticas não são viáveis, e servir de redundância numa situação de falha. O fragmento do sistema de *bundles* à esquerda representado na **Figura 18** - Diagramas gerados pelo protótipo VISCTE num cenário de utilização de dois servidores de Internet e após o término de um deles.

(Antes) mostra o sistema de *bundles* num cenário onde existem dois servidores de Internet, e ambos estão a utilizar o serviço de controlo de tráfego disponibilizado pelo controlador de performance. De notar que, não é obrigatório que os servidores de Internet estejam a utilizar o mesmo servidor de imagem. Isto porque, um dos servidores de Internet pode estar a ser alvo de mais pedidos de acesso e, desta forma, a velocidade de tráfego ser mais lenta. Neste cenário, é simulada uma situação de falha de um dos servidores de Internet. O servidor de Internet A deixou de funcionar e o componente associado foi terminado. O fragmento do sistema em análise à direita representado na Figura 18 (Depois) mostra as alterações do sistema de componentes após o término do componente servidor de Internet A. De notar que, após o término do componente, todas as suas dependências desapareceram. Isto porque, todos os serviços que estavam a ser utilizados foram libertados antes do *bundle* ser terminado. O mecanismo do protótipo VISCTE intercetou esta sequência de eventos e procedeu à alteração do diagrama como mostra a figura abaixo.

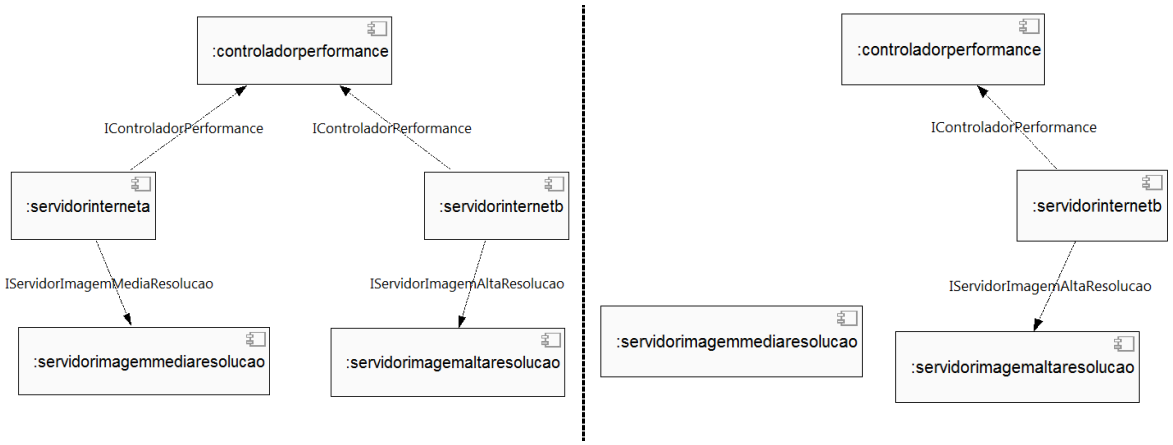



**Figura 18** - Diagramas gerados pelo protótipo VISCTE num cenário de utilização de dois servidores de Internet e após o término de um deles.

# 5 Concretização

A visualização de comportamento de software oferece uma forma útil de identificar incoerências entre a arquitetura planeada e implementação existente. No entanto, como as ferramentas de análise de software são na maioria das vezes separadas das ferramentas de desenvolvimento, os programadores frequentemente não chegam a utilizar verdadeiramente essas ferramentas. Esta foi a motivação que levou à integração do protótipo desenvolvido num ambiente de desenvolvimento de software, mais especificamente no ambiente de desenvolvimento Eclipse (The Eclipse Foundation, Eclipse, 2014). Dado que, o mecanismo desenvolvido pretende visualizar a interação entre componentes baseados na especificação OSGi, a escolha do ambiente de desenvolvimento Eclipse é óbvia. A plataforma de suporte a aplicações desenvolvidas em Eclipse – Equinox – é considerada uma implementação de referência das especificações do modelo OSGi e é talvez a implementação mais utilizada. Neste capítulo, será feita uma descrição mais técnica e concreta da implementação do protótipo VISCTE. Serão abordados tópicos referentes à arquitetura do protótipo desenvolvido e detalhados aspetos relacionados com a implementação dos componentes que o constituem.

## 5.1 Arquitetura

A plataforma de suporte a aplicações desenvolvidas no ambiente de desenvolvimento Eclipse é denominada por Equinox. É considerada uma implementação de referência pela sua robustez e escalabilidade e é talvez a implementação OSGi mais utilizada (McAffer, VanderLei, & Archer, 2010).

O mecanismo desenvolvido é composto por vários componentes que interagem entre si. Os componentes foram desenvolvidos de acordo com a especificação OSGi, de forma a viabilizar a análise dos sistemas de componentes baseados no modelo OSGi que o protótipo visa analisar. A *framework* OSGi oferece mecanismos indispensáveis para a análise comportamental dos sistemas OSGi. Esta, ainda possibilita que, a qualquer momento, os componentes que constituem um determinado sistema OSGi – *bundles* – tenham acesso a informação relativa aos *bundles* e serviços existentes, bem como, receber notificações de eventos específicos da plataforma OSGi. O mecanismo desenvolvido funciona através da interação de três componentes: um *bundle* que constitui a vista gráfica

(*Visualizer*), um *bundle* que monitoriza as notificações da framework OSGi (*BundleMonitor*), e por último um *bundle* que monitoriza as ações de outros *bundles* (*ServiceMonitor*) que constituem o sistema em análise. A Figura 19 representa de que forma o protótipo VISCTE integra o sistema de desenvolvimento Eclipse e o sistema baseado na especificação OSGi que o utilizador visa analisar.

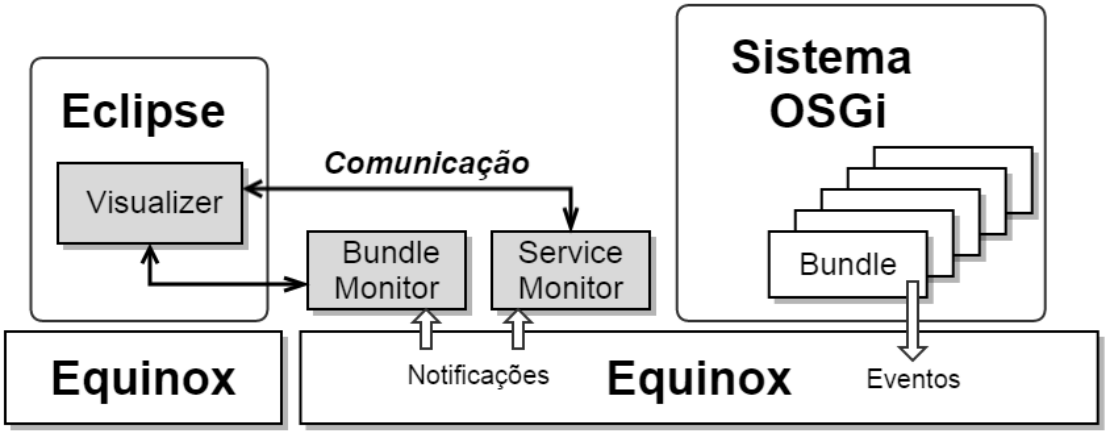



**Figura 19** - Integração dos componentes (a sombreado) do protótipo VISCTE no Eclipse e sistema de componentes em análise.

O componente *Visualizer* integra, sob forma de um *plug-in*, o ambiente de desenvolvimento Eclipse e providencia ao utilizador uma interface gráfica que mostra, através da geração de diagramas, a interação de componentes do sistema OSGi em análise, tal como ilustrado no Capítulo 4. A integração do protótipo desenvolvido no Eclipse representa um incentivo à utilização de ferramentas de análise enquanto se desenvolve software. Os componentes *BundleMonitor* e *ServiceMonitor*, apesar de não fazerem parte do conjunto de componentes que constitui o sistema OSGi em análise, integram a mesma instância da *framework* OSGi sobre qual o sistema está a ser executado. A integração dos componentes na mesma instância onde o sistema em análise está a ser executado é fundamental para o funcionamento do protótipo. Esta integração, permite que a ferramenta tenha acesso aos eventos OSGi notificados pela *framework* e monitorizar as ações dos componentes que compõem o sistema. Dado que, o processo de execução da *framework* Equinox do sistema Eclipse e do sistema OSGi em análise é diferente, é necessário estabelecer um canal de comunicação entre os componentes *BundleMonitor e ServiceMonitor* e o componente *Visualizer*. Este canal serve, essencialmente, para os componentes *BundleMonitor* e *ServiceMonitor* notificarem o componente *Visualizer* dos eventos chave entre componentes no sistema OSGi em análise. Uma vez que, ambos os

sistemas estão a correr em processos diferentes no sistema operativo, tem que existir comunicação entre processos utilizando um mecanismo que permita transferir informação entre os mesmos. O protótipo desenvolvido utiliza *sockets* TCP, como mecanismo de comunicação entre processos, onde os componentes *BundleMonitor* e *ServiceMonitor* enviam pacotes de dados que notificam o componente *Visualizer* dos eventos OSGi ocorridos no sistema de componentes em análise.

### *5.1.1 BundleMonitor*

A representação visual das interações entre componentes do sistema OSGi em análise depende dos eventos gerados pela própria plataforma onde o sistema está a ser executado. A plataforma OSGi gera um fluxo de eventos sempre que existem alterações ao sistema. Contudo, apenas um subconjunto dos eventos gerados interessa ao protótipo VISCTE, dado que, os diagramas de fragmentos do sistema gerados pelo protótipo focam apenas a representação de interação de componentes e serviços. Os eventos que o mecanismo desenvolvido está interessado em escutar são: inicialização e término de um *bundle*, porque um *bundle* só existe impacto significativo no sistema após a sua inicialização; registo e cancelamento de serviços possuem a sua relevância, dado que, podem estar diretamente associado à execução de determinada funcionalidade; obtenção e libertação de serviços, que representam a utilização de serviços e as dependências entre componentes. A *framework* OSGi disponibiliza, a todos os componentes, um meio de escutar os eventos relacionados com a inicialização e término de componentes, bem como, o registo e cancelamento de serviços. Contudo, não possui um mecanismo de escuta sobre os eventos que abrangem a obtenção e libertação de serviços. O componente *BundleMonitor* tem como finalidade escutar os eventos através dos meios providenciados pela *framework* OSGi e notificar o componente *Visualizer* da ocorrência dos mesmos. Para que isto seja possível, é necessário existir uma integração do componente *BundleMonitor* na plataforma OSGi onde o sistema alvo está a ser executado e, desta forma, sempre que ocorrer um dos eventos descritos, a plataforma notifica todos os bundles interessados, incluindo o componente *BundleMonitor* que após a receção desses eventos notifica de imediato o componente *Visualizer*. Na Figura 20 é representado um exemplo da ocorrência de eventos OSGi e a respetiva notificação para o componente *BundleMonitor*. No exemplo representado ocorre a inicialização de um componente (*Bundle* A) e registo de um serviço (Serviço S1) e de seguida, dado que o componente *BundleMonitor* está integrado na

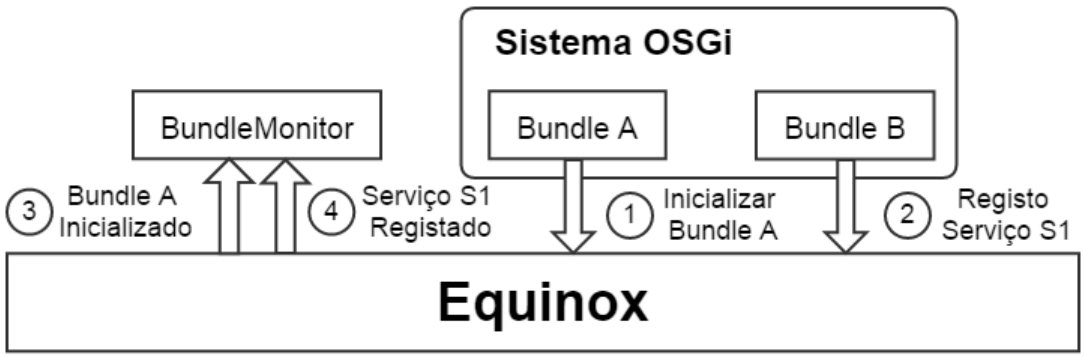



**Figura 20** – Integração e funcionamento do *bundle BundleMonitor* com o sistema OSGi em análise.

mesma plataforma que o sistema OSGi alvo e subscreveu os eventos de alteração de estado de bundles e serviços, a plataforma notifica o *BundleMonitor*.

### *5.1.2 ServiceMonitor*

O *bundle ServiceMonitor* é utilizado para colmatar a ausência de notificações da plataforma OSGi em cenários concretos. Como explicado na Secção **Error! Reference source not found.**, a *framework* OSGi disponibiliza a todos os *bundles* instalados, a possibilidade de receber notificações sobre o inicialização e término de componentes e registo, e cancelamento de serviços. No entanto, a *framework* OSGi não possui notificações que alertam quando um componente obtém ou liberta um determinado serviço e estas notificações são relevantes no contexto desta dissertação porque as dependências entre componentes representadas pelo mecanismo VISCTE constituem a utilização de serviços. Por exemplo, recorrendo ao exemplo fictício da plataforma de transferência de imagens pela Internet da Secção 4.1, só é possível representar a dependência entre os servidores de Internet e os servidores de imagem se o mecanismo implementado conseguir intersetar os eventos de obtenção de serviços. Da mesma forma, só é possível remover as dependências entre componentes se o mecanismo implementado tiver conhecimento dos eventos de libertação de serviços. O componente *ServiceMonitor* consegue colmatar a ausência deste tipo de notificações, através da instrumentação do sistema em análise e utilizando programação orientada a aspetos, mais concretamente AspectJ (The Eclipse Foundation, AspectJ, 2014). A integração na mesma plataforma OSGi onde o sistema em análise está a ser executado, permite que o componente *ServiceMonitor* tenha acesso aos outros componentes que compõem o sistema alvo e, com o módulo dos aspetos, é possível

alterar o comportamento da aplicação, injetando um comportamento adicional sobre determinados pontos de execução. Assim sendo, a utilização dos aspetos permite, intersetar todas as chamadas dos componentes que constituem o sistema em análise, a métodos da *framework* OSGi que permitam adquirir ou libertar a utilização de serviços e aplicar um comportamento adicional. O comportamento adicional pretende ser o mais simples possível, dado que o protótipo desenvolvido pretende minimizar o impacto no sistema de componentes em análise. Desta forma, quando um método que visa a obtenção ou libertação de um serviço é intersetado o componente *ServiceMonitor* simplesmente notifica o componente *Visualizer*. A Figura 21 mostra um exemplo da integração e aplicabilidade do componente *ServiceMonitor*. O exemplo escolhido mostra que, através dos aspetos, é possível interferir com o comportamento de um componente (obtenção de um serviço) e adicionar um comportamento adicional (envio da notificação da obtenção do serviço ao componente *Visualizer).*

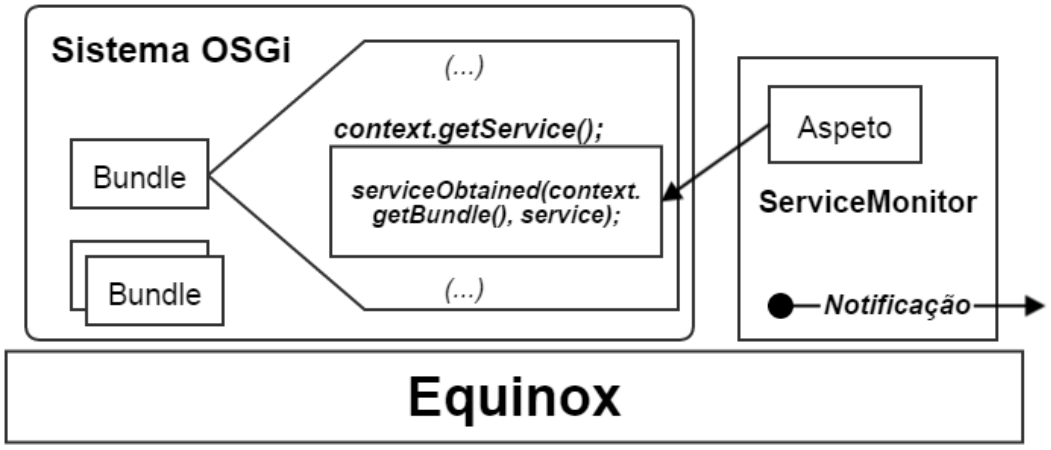



**Figura 21** - Integração e funcionamento do componente *ServiceMonitor* com o sistema em análise.

### *5.1.3 Visualizer*

O componente *Visualizer* constitui a interface gráfica do mecanismo desenvolvido e é responsável pela produção e representação de diagramas de fragmentos do sistema. O ambiente de desenvolvimento Eclipse possibilita a integração de componentes sob forma de *plug-ins*. A integração do componente *Visualizer* é feita através do ponto de extensão `org.eclipse.ui.views` que o Eclipse disponibiliza para adicionar uma nova vista. Uma vista é tipicamente utilizada para transmitir algum tipo de informação, abrir um editor, ou

mesmo até, mostrar as propriedades de um editor ativo. No protótipo desenvolvido, a vista criada tem como finalidade transmitir a informação da interação entre componentes através da geração de diagramas e é composta pela área de desenho e por um conjunto de ações.

O componente *Visualizer* é composto pela contribuição de vários *plug-ins*. Dado que, as técnicas de desenho de diagramas não representam o principal foco desta dissertação, recorreu-se à utilização do *plug-in* ZEST (The Eclipse Foundation, ZEST, 2014) que tem como objetivo a representação gráfica de grafos compostos por nós e ligações entre os mesmos. A *framework* do ZEST contém um mecanismo de geração de diagramas embutido, onde é possível definir um conjunto de características: elementos que compõem o diagrama e suas dependências, algumas definições visuais e até mecanismos de representação visual. O *Eclipse Modeling Framework* (EMF) (The Eclipse Foundation, Eclipse Modeling Framework (EMF), 2014) foi utilizado para criar o modelo de dados de captura representado na Figura 13 e gerar todas as classes necessárias para manipular a informação que permite desenhar o diagrama de componentes. O componente *Visualizer* depende fundamentalmente da contribuição de dois componentes: *BundleMonitor* e *ServiceMonitor*. Os componentes *BundleMonitor* e *ServiceMonitor* enviam informação (notificações) sobre os eventos ocorridos no sistema OSGi em análise que permitem a produção de fragmentos do sistema. Após a receção das notificações, o componente *Visualizer* decide se processa ou não as notificações, com base se o protótipo desenvolvido se encontra ou não em modo de captura. Se a protótipo se encontrar em modo de captura, o componente *Visualizer* modifica o modelo de dados e atualiza o diagrama de componentes e dependências. Dado que, todos os eventos processados estão associados a uma data e hora, é possível a qualquer momento rever o histórico do diagrama de componentes através dos botões de navegação no histórico de eventos.

O protótipo pretende gerar uma representação localizada do comportamento do sistema centrado na execução de funcionalidades ou casos de testes. O componente *Visualizer* possibilita a navegação do diagrama para artefactos específicos da implementação correspondentes, como por exemplo, componentes (ficheiro MANIFEST), interfaces ou implementações concretas de serviços. A Figura 22 mostra a integração do componente *Visualizer* sob forma de um *plug-in* no ambiente de desenvolvimento Eclipse e a possível navegação do diagrama para artefactos específicos da implementação.

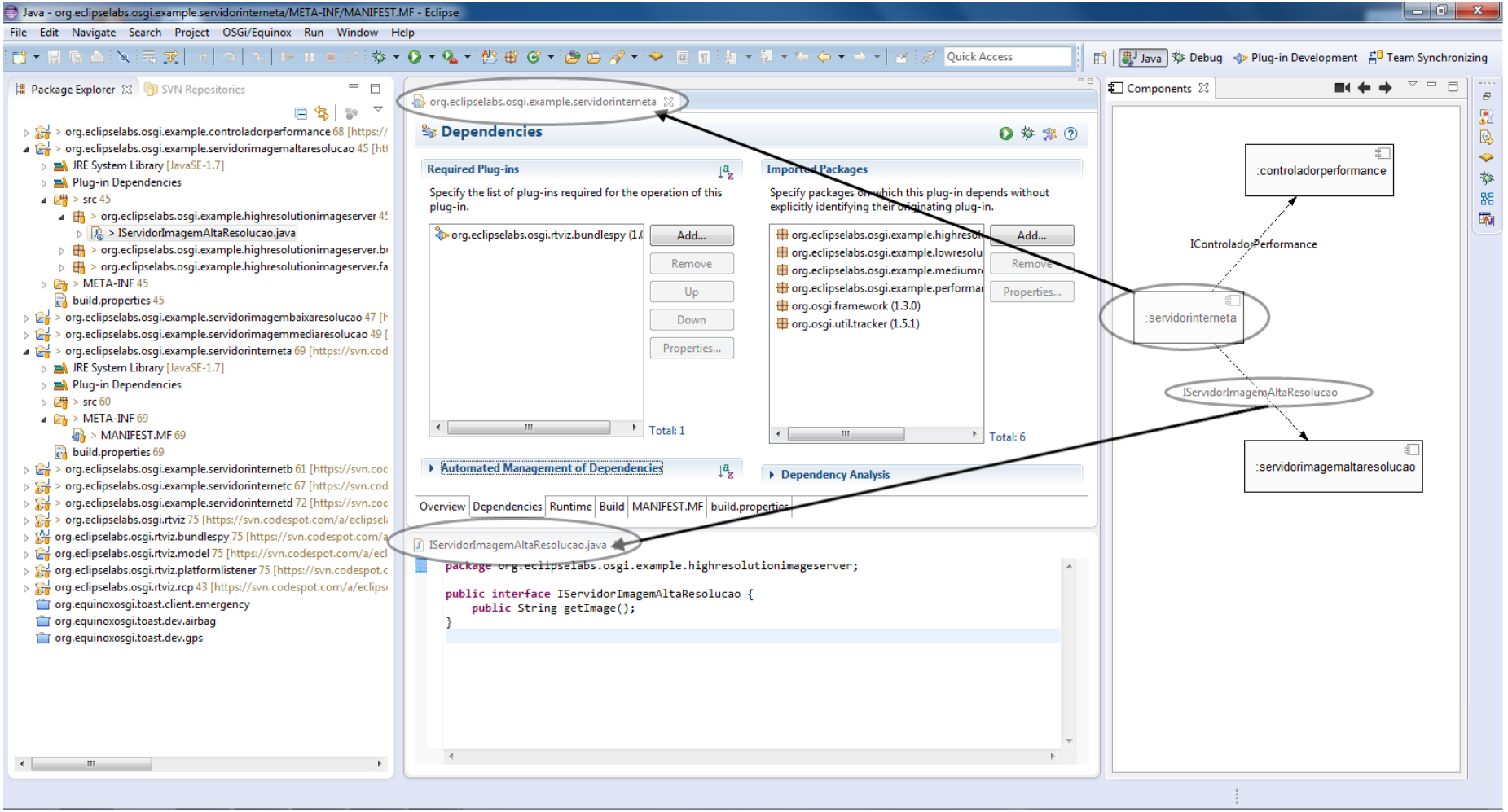



**Figura 22** - Integração do bundle *Visualizer* sob forma de um *plug-in* do tipo vista no ambiente de desenvolvimento Eclipse e demostração da navegação entre o diagrama e artefactos da implementação.

### *5.1.4 Comunicação*

Os componentes *BundleMonitor* e *ServiceMonitor* e o componente *Visualizer* encontram-se em dois processos do sistema operativo distintos, e por esta razão, é necessário um mecanismo de comunicação entre processos. O protótipo desenvolvido faz uso de *sockets* TCP para estabelecer um canal de comunicação entre estes componentes. Desta forma, todas as notificações sobre os eventos ocorridos são feitas através do envio de pacotes de dados como mostra a Figura 23.

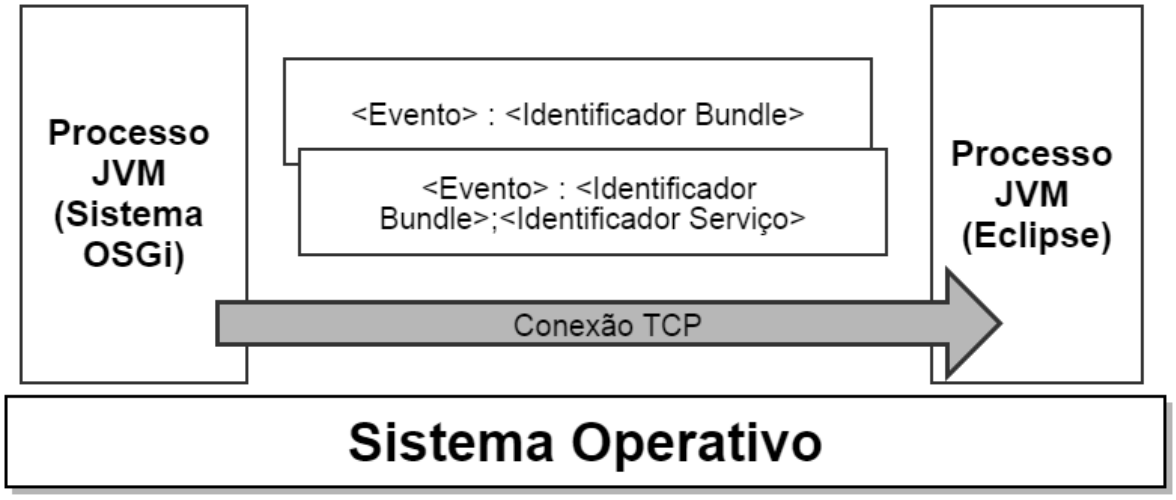



**Figura 23 -** Comunicação entre processos através do envio de pacotes de dados por um *socket* TCP.

Os pacotes de dados enviados pelo componente *BundleMonitor* são compostos pela informação do evento e pelo identificador do componente ou serviço associado. O componente monitoriza toda a informação associada à inicialização e término de componentes, e nesta situação, o pacote de dados contém a informação da inicialização ou término de um componente – *Bundle Started* ou *Bundle Stopped* respetivamente – e do *bundle symbolic name* do componente respetivo como mostra a figura acima. O componente *BundleMonitor* é também responsável por monitorizar toda a informação de eventos associada ao registo e cancelamento de serviços, e da mesma forma, o pacote de dados a enviar contém a informação do tipo de evento – *Service Registered* ou *Service Unregistered* – o *bundle symbolic name* do componente que regista ou cancela o serviço e do identificador unívoco do serviço associado. Desta forma, o componente *Visualizer*

consegue identificar o tipo de evento notificado e guardar um identificador do componente e serviço associados ao evento notificado. Posteriormente, o componente *Visualizer* decide, com base se o protótipo tem ou não o modo de captura ligado, se guarda a informação recebida no modelo de dados ou se a informação recebida é descartada.

A informação enviada pelo componente *ServiceMonitor* é composta pelo tipo de evento, pelo identificador do componente que obtém ou liberta o serviço, pelo identificador do serviço associado e pela implementação do serviço concreta. O componente interseta os métodos de obtenção e libertação de serviços dos componentes que constituem o sistema em análise e envia um pacote de dados com o tipo de evento – *Service Obtained* e *Service Released* – o *bundle symbolic name* do componente que obteve ou libertou o serviço, o indentificador do serviço respetivo (*interface*) e do identificador da implementação do serviço concreta. Desta forma, o componente *Visualizer* consegue identificar o tipo de evento notificado e guardar um identificador associado ao componente e serviço associado ao evento respetivo, e posteriormente decide, com base se o modo de captura está ou não ligado, se guarda ou não a informação recebida no modelo de dados.

## 5.2 Limitações

O protótipo desenvolvido apresenta algumas limitações. A visualização de interação entre componentes em tempo de execução é exclusivamente aplicável a sistemas de componentes baseados na especificação OSGi e encontra-se limitado aos eventos de inicialização e término de componentes, registo e cancelamento de serviços e obtenção e libertação de serviços. Numa tentativa de simplificar o diagrama gerado e a comunicação entre os componentes *BundleMonitor* e *ServiceMonitor* e o componente *Visualizer*, a informação apresentada é também limitada à identificação dos componentes e serviços. O protótipo encontra-se ainda limitado à execução no ambiente de desenvolvimento Eclipse ou sistemas de componentes com as mesmas características.

# 6 Caso de estudo: TransNet

Com vista a avaliar o protótipo desenvolvido, foi levada a cabo uma experiência de utilização do mesmo no âmbito de um sistema de componentes de escala industrial.

## 6.1 Contexto

A Coriant [1] é uma empresa com a área de negócio centrada nas redes e telecomunicações, a qual oferece soluções de equipamento de rede e serviços para grandes operadores de redes móveis, sendo o seu principal centro de desenvolvimento e investigação de software em Alfragide, Lisboa.

O autor desta dissertação trabalha na Coriant, e dado esse facto, foi-lhe autorizado o acesso ao código fonte de um dos produtos da empresa que foi desenvolvido com a tecnologia OSGi. O sistema alvo baseado em componentes é denominado de TransNet e acumula cerca de 10 anos de desenvolvimento. O sistema TransNet é uma ferramenta automática de auxílio ao planeamento, otimização, licenciamento de redes e geração relatórios relacionados com estas atividades como ilustra a Figura 24.

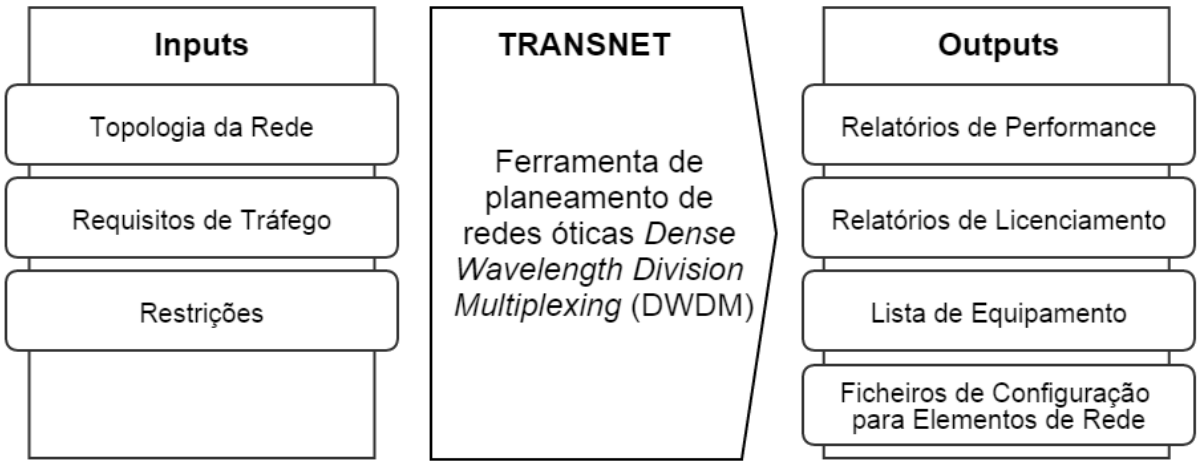



**Figura 24** – Representação diagramática de *inputs* e *outputs* relacionados com a aplicação da ferramenta de planeamento de redes TransNet.

[1] www.coriant.com

## 6.2 Objetivos

A utilização do protótipo no contexto do caso de estudo escolhido pretende avaliar se a solução técnica desenvolvida é escalável para um sistema de industrial, isto é, se o protótipo desenvolvido funciona não apenas em sistemas desenvolvidos artificialmente (utilizados durante a fase de implementação), mas também, em sistemas de escala industrial com vários anos de desenvolvimento e uma dimensão e complexidade muito maiores.

O caso de estudo escolhido visa também avaliar a usabilidade e utilidade do mecanismo proposto de um prisma industrial. Pretende-se saber se, os conceitos que integram o protótipo desenvolvido providenciam uma mais-valia num contexto industrial de desenvolvimento de software.

## 6.3 Caraterização do sistema

Durante a evolução da plataforma Eclipse como ambiente de desenvolvimento, foi pensada numa abordagem, para que os componentes que a compõem pudessem ser utilizados para construir praticamente qualquer aplicação que necessitasse de uma infraestrutura para suportar *plug-ins*. O conjunto mínimo de *plug-ins* necessários para construir uma aplicação cliente do tipo *Rich Client* é conhecido por *Rich Client Platform* (RCP) (The Eclipse Foundation, Rich Client Platform, 2014). O sistema de componentes TransNet é uma aplicação RCP, desta forma, o TRASNET faz uso do modelo dinâmico de *plug-ins* e da biblioteca de criação de interface gráfica – *The Standard Widget Toolkit* (SWT) (The Eclipse Foundation, SWT: The Standard Widget Toolkit, 2014) – que incorpora a plataforma de desenvolvimento Eclipse. O modelo dinâmico de *plug-ins* viabiliza a adaptabilidade do sistema de componentes TransNet, que desta forma, consegue produzir versões diferentes do produto de acordo com o operador de rede móvel (AT&T, Verizon, etc).

De forma a caracterizar o sistema TransNet em termos da sua dimensão, a Tabela 1 apresenta um conjunto de dados recolhidos com a ajuda da ferramenta SonarQube (SonarQube, 2014).

***Caracterização da dimensão do sistema TransNet***

| | |
|---|---|
| *Linhas de código* | **1.525.859** |
| *Ficheiros* | **11.615** |
| *Classes* | **12.797** |
| *Métodos* | **130.733** |
| *Componentes* | **90** |
| *Serviços* | **199** |

**Tabela 1** – Informação que caracteriza o sistema de componentes TransNet utilizando a ferramenta SonarQube (SonarQube, 2014).

## 6.4 Integração do VISCTE

Dadas as características do sistema de componentes TransNet, foi possível integrar o componente *Visualizer* do protótipo VISCTE embebido no próprio sistema. O sistema TransNet partilha do mesmo modelo de *plug-ins* e faz uso da mesma interface gráfica que o Eclipse. Desta forma, pretendeu-se demostrar que é possível integrar o protótipo, tanto no Eclipse como em qualquer outro sistema de componentes que partilhe as mesmas características. Este tipo de integração não implica quaisquer alterações à implementação do sistema alvo. A Figura 25 mostra o protótipo VISCTE integrado sob forma de um *plug-in* no sistema de componentes TransNet. A funcionalidade de adição de elementos de rede (ONN-X96) ao mapa de planeamento de rede gerou um fragmento do sistema constituído por cinco componentes e relações entre si. No contexto desta funcionalidade, o componente *commands* utilizou um serviço (*IplacerService*) que o componente *proxy* forneceu para posicionar os elementos de rede no mapa. O componente *datamodel* foi inicializado e obteve vários serviços do componente *common* (componente que contém serviços utilitários). Durante a execução da funcionalidade de adição de elementos de rede, o componente *datamodel* e *common* utilizaram o mesmo serviço (*IEqmInfoProvider*) providenciado pelo componente *eqm* para obter informação dos equipamentos de rede selecionados.

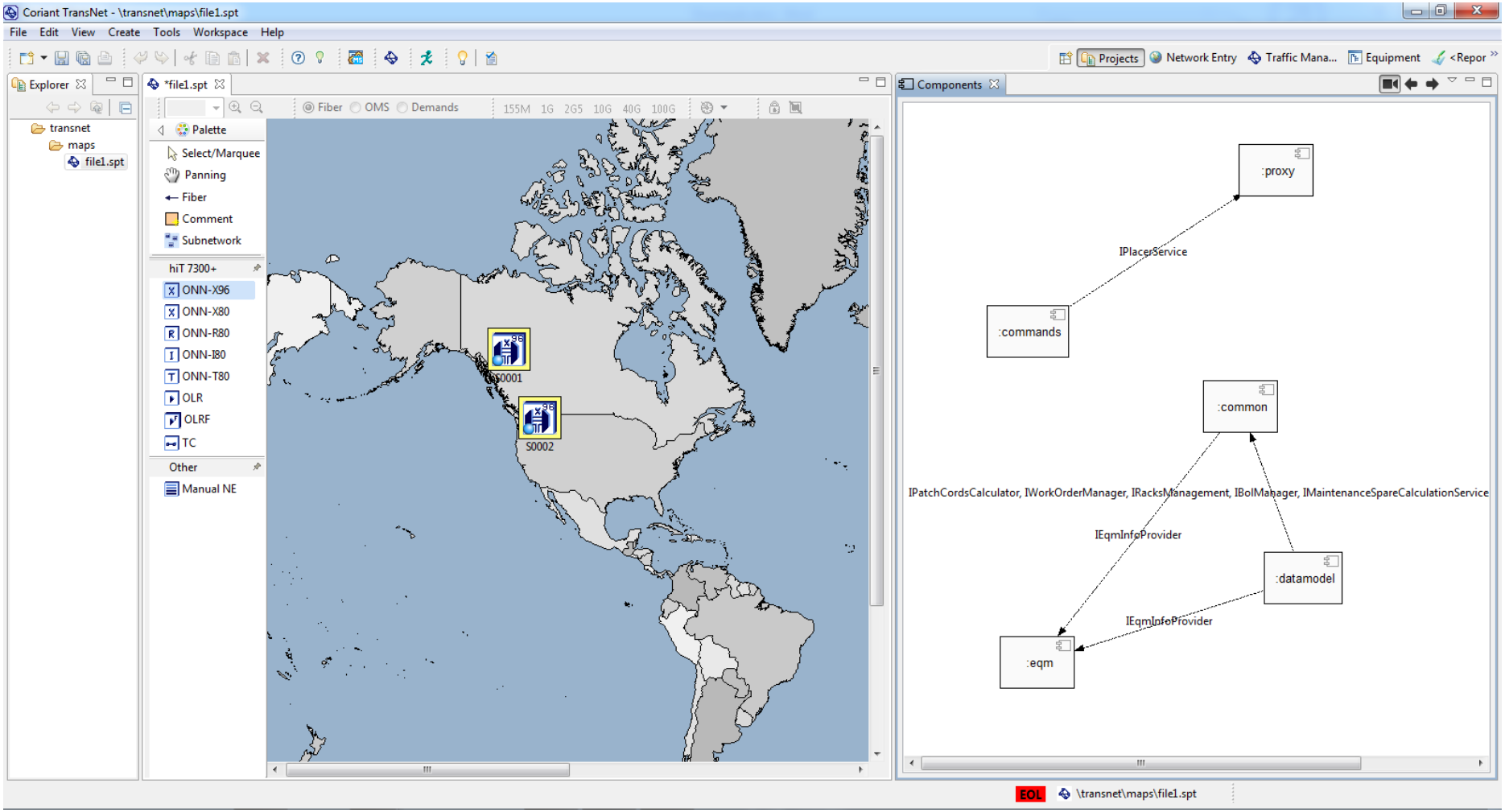



**Figura 25** - Protótipo VISCTE integrado sob forma de um *plug-in* do tipo vista no sistema de componentes TransNet.

## 6.5 Cenários de utilização

Com vista a ilustrar a forma como um programador ou analista pode interagir com o VISCTE para analisar os componentes do TransNet, esta secção apresenta dois casos de utilização.

### *6.5.1 Abertura do painel de pesquisa*

Este cenário de utilização foca apenas os eventos de inicialização de componentes. Neste cenário foi ativado o modo de captura, que o protótipo desenvolvido disponibiliza, durante a execução da funcionalidade de abertura do painel de pesquisa presente no sistema TransNet. O painel de pesquisa escolhido é na realidade, uma vista que faz parte do subconjunto de *plug-ins* que a plataforma de desenvolvimento Eclipse disponibiliza para as aplicações RCP. Desta forma, os componentes esperados na abertura do painel de pesquisa fazem parte da aplicação mas não representam componentes específicos do sistema de componentes TransNet. Durante o período de captura, constatou-se que a plataforma OSGi, sobre a qual o sistema TransNet estava a ser executado, enviou eventos de inicialização de cinco componentes responsáveis pela inicialização e integração de uma vista que corresponde ao painel de pesquisa. Os eventos foram enviados porque a plataforma onde o sistema de componentes TransNet está assente adota a implementação *lazy loading*, a qual define que, os componentes são inicializados quando necessários e não durante o arranque do sistema de componentes. A Figura 26 representa o fragmento do sistema de componentes TransNet gerado pelo protótipo VISCTE após a execução da funcionalidade de abertura do painel de ajuda. A figura representa um diagrama com os cinco componentes utilizados no contexto da funcionalidade de abertura de um painel de pesquisa descrito no caso de utilização.

O protótipo VISCTE possibilita, através da funcionalidade *playback*, navegar no histórico de eventos capturados durante o período de captura. O protótipo dá aos programadores e analistas o controlo do cenário capturado, que permite uma análise mais profunda dos eventos ocorridos, entre outras vantagens, como por exemplo saber a ordem de inicialização dos componentes no contexto de determinado cenário.

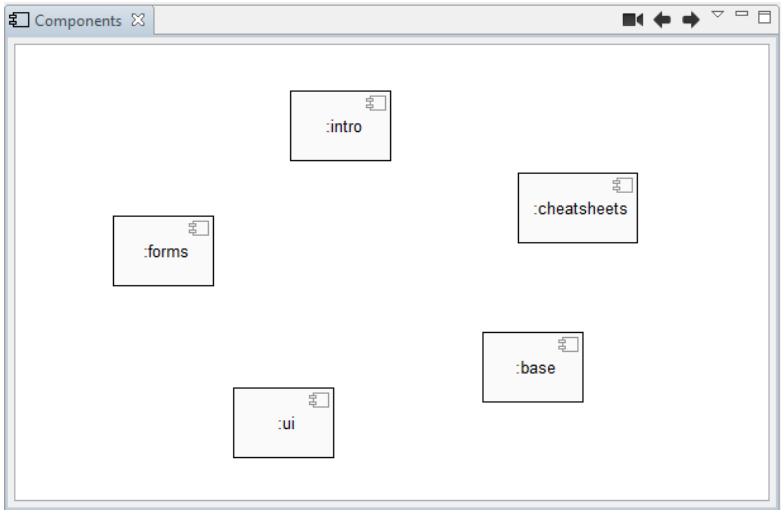



**Figura 26** - Fragmento do sistema de componentes TransNet gerado pelo protótipo VISCTE após a execução da funcionalidade de abertura do painel de pesquisa.

Neste cenário não é possível comparar o resultado do protótipo VISCTE com o *plug-in* de visualização de dependências estáticas (The Eclipse Foundation, PDE Incubator Dependency Visualization, 2009), dado que, o *plug-in* de visualização de dependências estáticas não apresenta os *bundles* utilizados na abertura do painel de pesquisa por estes não pertencerem ao sistema de componentes TransNet.

### *6.5.2 Criação de um mapa de planeamento de redes*

Este cenário de utilização envolve eventos de inicialização de componentes e obtenção de serviços. A funcionalidade de criação de um mapa de planeamento de redes foi capturada pelo protótipo desenvolvido e, durante o período de captura, verificou-se que a plataforma OSGi, sobre a qual o sistema TransNet está a ser executado, notificou os componentes que compõem o protótipo desenvolvido da inicialização de componentes e obtenção de serviços. A Figura 27 mostra o diagrama gerado pelo protótipo VISCTE resultante da interação entre componentes no contexto da funcionalidade descrita. A criação de um mapa de planeamento de redes resulta da interação dos seguintes componentes: o componente *query* cujo propósito é a realização de pesquisas à base de dados; o componente *eqm* que proporciona um serviço (*IEqmInfoProvider*) que fornece informação dos equipamentos de rede disponíveis no mapa de planeamento; o componente *serialization* que fornece um serviço (*ISptManager*) de serialização de objetos; o componente *discovery* que providencia um serviço (*IEquipmentDiscovery*) de descoberta

de equipamentos de rede; componente *common* que fornece um conjunto de serviços utilitários comuns ao sistema; por último, o *bundle* central desta funcionalidade (*datamodel*) que utiliza todos os serviços referidos, incluindo um serviço que o próprio *bundle* fornece.

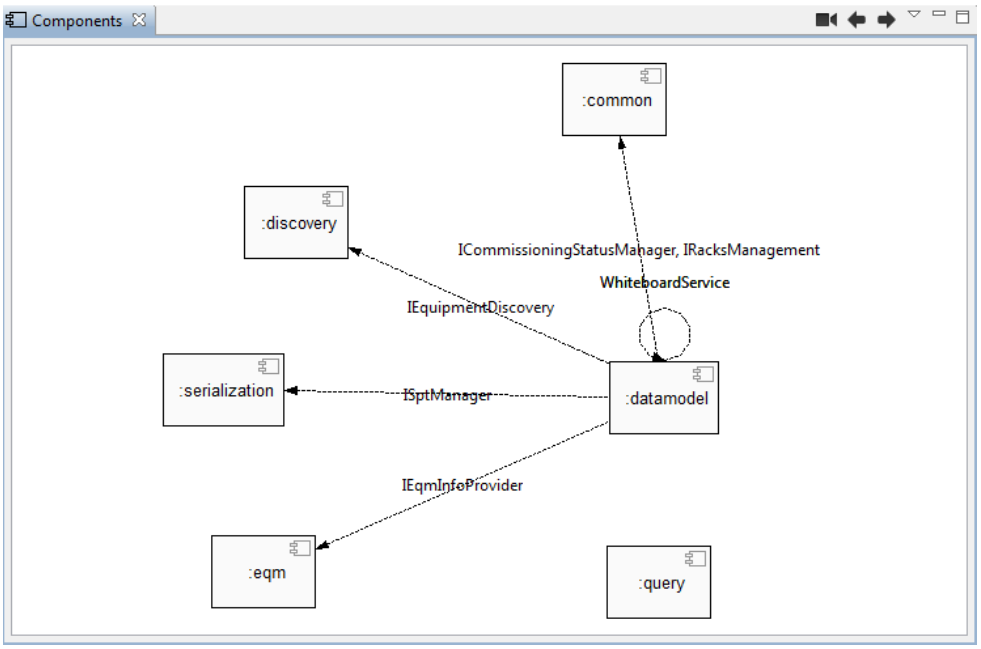



**Figura 27 -** Diagrama gerado pelo protótipo VISCTE aplicado ao cenário de utilização criação de um mapa de planeamento de rede do sistema de componentes TransNet.

Com vista a dar uma ideia superficial da comparação com visualização estática de dependências, a Figura 28 mostra o diagrama gerado pelo *plug-in* de visualização de dependências estáticas de componentes (The Eclipse Foundation, PDE Incubator Dependency Visualization, 2009). A figura apresenta todos os componentes que compõem o sistema TRASNET e as respetivas dependências estáticas, com o componente central da funcionalidade de criação de um mapa de planeamento de rede (*datamodel*) e as suas dependências estáticas em foco. O diagrama representado mostra que, em sistemas de larga escala, os mecanismos de foco que integram a ferramenta de visualização estática são insuficientes para reduzir a complexidade do diagrama dado a sua extensão. Contudo, a abordagem do protótipo desenvolvido provou-se bastante eficiente na redução da complexidade de diagramas de componentes de software complexos através da geração de fragmentos do sistema localizados com base na execução de funcionalidades. Existem certos tipos de interações entre componentes que só são determinados em tempo de execução, logo, a análise estática de dependências não fornece uma ajuda satisfatória pois

apenas oferecem uma representação estática dos componentes e das suas ligações. O protótipo desenvolvido prova que é possível, através de uma demonstração de funcionalidade ou casos de teste, indicar as interações entre componentes ocorridas durante um período de execução, para gerar uma representação localizada do comportamento do sistema de modo a identificar eventuais discrepâncias entre a arquitetura planeada e implementação dos sistemas de software.

A localização de um determinado artefacto num ambiente de desenvolvimento de um sistema de componentes de grande escala pode não ser tarefa fácil. O método de localização normalmente utilizado pelos programadores é o método de pesquisa manual, no qual, o tempo despendido nesta tarefa costuma estar diretamente relacionado com o tamanho do sistema, quanto maior o sistema, maior o tempo despendido. Neste contexto, o mecanismo do protótipo desenvolvido que visa representação localizada do conjunto de componentes e serviços utilizados durante uma funcionalidade e a respetiva navegação do diagrama para artefactos de implementação é extremamente útil. Não só o programador consegue restringir o conjunto de componentes e serviços que estão verdadeiramente a ser utilizados no contexto de uma funcionalidade, como também, consegue navegar entre o diagrama gerado e os artefactos dos elementos do diagrama correspondentes.

**Figura 28** - Diagrama de componentes do sistema TransNet gerado pelo *plug-in* de visualização de dependências estáticas (The Eclipse Foundation, PDE Incubator Dependency Visualization, 2009).

## 6.6 Observações

O caso de estudo demonstra empiricamente que o protótipo VISCTE é adequado para ser utilizado em sistemas de componentes de escala industrial. A integração e experimentação do protótipo com sistema de componentes TransNet mostram que, o protótipo desenvolvido pode ser considerado uma ferramenta útil de análise comportamental de sistemas de componentes de grande escala em tempo de execução.

Foram encontradas algumas dificuldades no âmbito do caso de estudo efetuado. A plataforma OSGi não disponibiliza eventos sobre a obtenção e libertação de serviços, e desta forma, foi necessário encontrar uma solução para solucionar esta limitação. A solução escolhida baseou-se na utilização de aspetos para controlar as chamadas a métodos de obtenção e libertação de serviços. Surgiram alguns problemas de integração com o sistema TransNet na qual, a maioria, correspondeu à falta de conhecimento da configuração e ambiente de desenvolvimento do sistema. Foram encontrados alguns problemas no âmbito da experimentação realizada e relacionados com a implementação da especificação OSGi, adotada durante a fase de desenvolvimento do sistema de componentes TransNet. Verificou-se que, em tempo de execução, a implementação do sistema TransNet não termina *bundles* ou liberta serviços quando já não estão a ser utilizados. Ao invés disso, os *bundles* são mantidos inicializados, mesmo após a sua inicialização, e os serviços são obtidos mesmo que os *bundles* já possuam uma referência para os mesmos. Desta forma, não é possível encontrar e demonstrar casos de utilização em que o foco da demonstração fosse o término de um *bundle* ou o cancelamento e a libertação de um serviço.

Embora não tenham sido realizados testes de performance minuciosos sob forma de avaliar o impacto do protótipo VISCTE sobre o sistema de componentes a analisar, verificou-se que não existem alterações de comportamento percetíveis ao olho humano quanto à execução de um sistema de componentes em larga escala a ser executado com e sem a integração do protótipo desenvolvido.

## 6.7 Vantagens

O protótipo desenvolvido, quando comparado com o *plug-in* de visualização de dependências estáticas, prova que a produção de fragmentos do sistema de componentes, com base em demonstrações de funcionalidades ou casos de teste, proporciona uma

solução viável na redução de complexidade de diagramas de componentes em larga escala. No entanto, existe uma relação direta entre a complexidade gerada e a implementações das funcionalidades, dado que, a geração de fragmentos do sistema são baseados na execução de uma ou mais funcionalidades. Desta forma, o protótipo em desenvolvimento consegue não só transmitir a veracidade comportamental do sistema em análise, como também, reduzir a complexidade de diagramas, apresentando aos programadores e analistas uma fração tipicamente bastante reduzida dos componentes e serviços do sistema.

Os conceitos aplicados ao mecanismo de desenvolvimento, nomeadamente a produção de diagramas representando fragmentos do sistema em tempo de execução, mostram que é possível gerar localização fiável e instantânea dos componentes e serviços utilizados no contexto de funcionalidades ou casos de testes executadas. O protótipo desenvolvido incorpora um mecanismo de navegação entre o diagrama e os artefactos da implementação correspondentes, dado que, a correspondência manual entre os elementos do diagrama e os artefactos é demorada em sistemas mais complexos.

# 7 Conclusões e trabalho futuro

Esta dissertação focou-se no desenvolvimento de um mecanismo de visualização de interação entre componentes em tempo de execução, mais concretamente no modelo de componentes que segue a especificação OSGi. O protótipo desenvolvido teve como objetivos a resolução de problemas relacionados com a complexidade de diagramas de sistemas de componentes em larga escala e com a falta de mecanismos de visualização de interação entre componentes em tempo de execução para o modelo de componentes OSGi. Com o intuito de solucionar ambos os problemas, foi proposta uma solução integrada no ambiente de desenvolvimento Eclipse, capaz de produzir diagramas de fragmentos do sistema através de uma demostração de funcionalidade ou casos de teste, com vista a gerar uma representação localizada do comportamento do sistema. O protótipo desenvolvido foi avaliado com base num caso de estudo com um sistema de componentes que utiliza a tecnologia OSGi de escala industrial chamado TransNet, proprietário da empresa Coriant. O caso de estudo realizado provou que a solução técnica desenvolvida é escalável para um sistema de grandes dimensões, e demonstrou a adequação do mecanismo proposto num contexto de desenvolvimento industrial.

Após a realização deste trabalho de Mestrado, conclui-se que a abordagem proposta é exequível dado o cumprimento dos objetivos propostos. Apesar das dificuldades e das limitações, o caso de estudo do protótipo de Visualização Interativa de Serviços e Componentes em Tempo de Execução revelou dados positivos. Revelou-se fazível a implementação de um mecanismo de monitorização de eventos OSGi em tempo de execução e a representação diagramática desses eventos através de um *plug-in* integrado no Eclipse. A técnica proposta para reduzir a complexidade de diagramas de sistemas de grande escala revelou-se eficiente, tendo reduzido o número de componentes de modo significativo através de uma representação localizada da execução do sistema.

O mecanismo desenvolvido nesta dissertação trata-se unicamente de uma prova de conceito que possui um conjunto de limitações apesar dos bons resultados. O mecanismo incorpora um comando de ativação do modo de captura e a possibilidade de navegação no histórico de eventos ocorridos. Outras duas funcionalidades poderiam ser adicionadas no contexto de trabalho futuro. A adição de filtros para omitir determinados componentes ou serviços e a possibilidade de guardar uma captura realizada num ficheiro com um dado formato, como por exemplo XML.

Do ponto de vista técnico, a utilização de programação orientada por aspetos para efeitos de instrumentação (componente *ServiceMonitor*) introduz algumas dificuldades de integração, dado que o sistema sob análise tem que ser executado numa configuração com dependências adicionais. Como solução técnica alternativa, valerá a pena investigar formas de acrescer uma plataforma OSGi (p.e. Equinox) com o objetivo de permitir a notificação de eventos de aquisição e libertação de serviços. Desta forma, a solução não dependeria de um processo de instrumentação para monitorizar serviços, tornando mais simples as configurações necessárias para executar um sistema a analisar.

Também como trabalho futuro, e com vista a aprofundar o nível de detalhe das interações entre componentes, o protótipo poderia incluir uma vista separada para representar as chamadas aos métodos e respetivos atributos ocorridos durante o modo de captura. Deverá ser analisada a possibilidade de integrar o mecanismo desenvolvido noutros ambientes de desenvolvimento. Embora o componente visualizador tenha sido concretizado para o ambiente de desenvolvimento Eclipse, os outros componentes que monitorizam os eventos podem ser reutilizados no contexto de outros ambientes de desenvolvimento, para os quais seria apenas necessário implementar visualizadores. Por fim, a aplicação do conceito de mecanismo proposto nesta dissertação poderia ser estudada no âmbito de outros modelos e tecnologias para concretização de sistemas baseados em componentes.

# 8 Bibliografia

Bachmann, F., Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J., . . . Wallnau, K. (2000). *Technical Concepts of Component-Based Software Engineering.* Software Engineering Institute.

Crnkovic, I., & Larsson, M. (2002). *Building Reliable Component-based Software Systems.* Artech House.

Crnkovic, I., Stafford, J., & Szyperski, C. (2011). Software Components beyond Programming: From Routines to Services. *Proceedings of the Computer Society Press Los Alamitos* (p. 22). IEEE.

D'Souza, D., & Wills, A. C. (1999). *Objects, Components and Frameworks: The Catalysis.* Boston, MA, USA: Addison-Wesley Longman Publishing Co.

Grundy, J., & Hosking, J. (2000). High-level static and dynamic visualisation of software architectures. *Proceedings of the International Symposium on Visual Languages* (pp. 5-12). IEEE.

Grundy, J., Mugridge, W., & Hosking, J. (1998). Static and Dynamic Visualisation of Software Architectures for Component-based Systems. *Proceedings of the Symposium on Visual Languages.* IEEE.

Holy, L., Jezek, K., Snajberk, J., & Brada, P. (2012). Lowering Visual Clutter in Large Component Diagrams. *Proceedings of the 16th International Conference on Information Visualisation (IV),* (pp. 36-41).

Holy, L., Snajberk, J., Brada, P., & Jezek, K. (2013). A Visualization Tool for Reverse-engineering of Complex Component Applications. *Proceedings of the 29th IEEE International Conference in Software Maintenance (ICSM)* (p. 4). IEEE.

Jacobson, I., Griss, M., & Jonsson, P. (1997). *Software reuse: architecture, process and organization for business success.* NY, USA: ACM Press/Addison-Wesley Publishing Co.

Johnson, R. (1997). *Frameworks = (Components + Patterns).* NY, USA: ACM New York.

Luer, C., & Rosenblum, D. (2001). WREN - An Environment for Component-Based Development. *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering* (pp. 207-217). ACM SIGSOFT Software Engineering Notes.

Martin, L., Giesl, A., & Martin, J. (2002). Dynamic component program visualization. *Proceedings of the Ninth Working Conference on Reverse Engineering,* (pp. 289-298).

McAffer, J., VanderLei, P., & Archer, S. (2010). *OSGi and Equinox: Creating Highly Modular Java Systems.* Series Editors.

OSGi, A. (2014). *The OSGi Architecture*. Obtido de OSGi Alliance: http://www.osgi.org/Technology/WhatIsOSGi

Sillito, J., Murphy, G., & De Volder, K. (2006). Questions Programmers Ask During Software Evolution Taks. *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, (pp. 23-34).

Snajberk, J., Holy, L., & Brada, P. (2013). Visualization of Component-Based Applications Structure Using AIVA. *Proceedings of the Software Maintenance and Reengineering (CSMR)*, (pp. 409-412).

SonarQube. (2014). *SonarQube*. Obtido de SonarQube: http://www.sonarqube.org/

Szyperski, C. (1998). *Component Software—Beyond Object-Oriented Programming.* Boston: Addison-Wesley Longman Publishing Co.

The Eclipse Foundation. (13 de 04 de 2009). *PDE Incubator Dependency Visualization*. Obtido em 25 de 10 de 2013, de Eclipse: http://www.eclipse.org/pde/incubator/dependency-visualization/index.php

The Eclipse Foundation. (2014). *AspectJ*. Obtido de Eclipse: http://www.eclipse.org/aspectj/

The Eclipse Foundation. (2014). *Eclipse*. Obtido de Eclipse: https://www.eclipse.org

The Eclipse Foundation. (2014). *Eclipse Modeling Framework (EMF)*. Obtido de Eclipse: http://www.eclipse.org/modeling/emf/

The Eclipse Foundation. (2014). *Rich Client Platform*. Obtido de Eclipse: http://wiki.eclipse.org/Rich_Client_Platform

The Eclipse Foundation. (2014). *SWT: The Standard Widget Toolkit*. Obtido de Eclipse: http://www.eclipse.org/swt/

The Eclipse Foundation. (2014). *ZEST*. Obtido de Eclipse: http://www.eclipse.org/gef/zest/

Tkach, D., & Puttick, R. (1995). *Object Technology in Application Development.* Addison Wesley Longman.