3
Trabalhos relacionados

Adaptação e implantação dinâmicas são requisitos de aplicações em diversos domínios. Diversas abordagens são capazes de promover adaptação e implantação em tempo de execução. Alguns projetos que provêem adaptação e implantação dinâmica têm sua abordagem para prover adaptação dinâmica em ambientes móveis estudada neste capítulo. Os conceitos básicos, implementações e ferramentas utilizadas nestes projetos serão descritos e comparados com a arquitetura implementada neste trabalho.

Este capítulo apresenta sistemas baseados em modelos de componentes ou orientados a serviços, que provejam suporte à adaptação dinâmica. Os trabalhos apresentados também devem poder ser implementados sobre ambientes móveis.

A motivação e o alvo de adaptações dinâmicas variam entre os projetos, e farão parte dos critérios de comparação. O modelo definido por cada sistema será outro critério de comparação. O terceiro quesito de comparação será a plataforma sobre qual cada um dos sistemas foi desenvolvido e se aplica.

3.1
OpenCOM

OpenCOM [15] é um modelo de componentes baseado em um subconjunto do modelo Microsoft COM [16]. O modelo é independente de linguagem, ou seja, existem implementações em diversas linguagens, como C, Cobol ou Java. Por ser mais leve do que o modelo da Microsoft, é normalmente utilizado em dispositivos móveis.

Definindo explicitamente dependências entre componentes, o modelo permite a reconfiguração de composições em tempo de execução, realizadas por mecanismos definidos pelo próprio modelo. O modelo também prevê mecanismos de interceptação de chamadas para tornar possível a adição de comportamentos sem a necessidade de reconfiguração de conexões entre componentes.

O modelo define facetas, receptáculos e conexões para realizar composições. Implementações do modelo se utilizam de mecanismos de reflexão com-

putacional para prover facilidade nos mecanismos de gerenciamento de componentes e conexões.

3.2 SCA

SCA (*Service Component Architecture*) [17] é implementado sobre um modelo orientado a serviços baseado em conjuntos de regras, tarefas, tipos de componentes e interfaces. Sua arquitetura distribuida é composta por componentes que se comunicam por chamada remota de procedimentos.

Cada serviço define requisitos em função de outros serviços, em forma de referências, além de propriedades que podem definir ou modificar seu comportamento. SCA provê um modelo para a implementação e composição de serviços que, junto a referências e propriedades, compõem componentes SCA.

Um módulo SCA é um pacote que contém serviços e componentes SCA, além de serviços externos e pontos de entrada, que podem ser acessados por aplicações externas que o utilizam.

A arquitetura é independente de linguagem, existindo implementações em diversas linguagens, como C, JAVA, PHP e Cobol.

3.3 Gravity

O projeto *Gravity* [8], assim como o mecanismo proposto neste trabalho, define um modelo de componentes orientado a serviços para atender à demanda por adaptação dinâmica de acordo com a disponibilidade de serviços. Em seu modelo, componentes oferecem e consomem serviços e todas as interações entre componentes ocorrem através de serviços. O protótipo do projeto *Gravity* foi implementado sobre o *framework* OSGi, em Java.

A adaptação dinâmica no *Gravity* é baseada na disponibilidade dos serviços e componentes. Durante a execução de aplicações, serviços podem tornar-se disponíveis ou indisponíveis de acordo com condições explícitas no ambiente de execução, e aplicações devem se adaptar de modo a utilizar ou deixar de utilizar estes serviços e componentes. Substituindo serviços, implementações de funcionalidades são modificadas, resultando em adaptações funcionais. A substituição de componentes resulta em adaptações estruturais.

Enquanto adaptações dinâmicas no *Gravity* são resultado da disponibilidade e indisponibilidade de serviços, neste trabalho são resultado de mudanças no contexto computacional ou de usuário. Por exemplo, um *browser* pode precisar se adaptar ao exibir determinados tipos de conteúdo. Para exibir um

documento PDF, por exemplo, o browser precisaria fazer o *download* e instalar um *plugin* específico, adaptando-se assim em extensão.

Por outro lado, *Gravity* não provê mecanismos de ciência a contexto, o que poderia também funcionar como motivação para realização de adaptações dinâmicas e persistência de estado interno, o que tornaria mais suave a troca entre serviços.

Gravity difere do trabalho proposto na descrição dos componentes e suas composições, que são realizadas em arquivos descritores escritos em XML, enquanto que neste trabalho a descrição e composição são realizadas programaticamente. Composições no *Gravity* podem ser tratadas também como componentes, oferecendo e consumindo serviços, diferindo também neste ponto da arquitetura apresentada.

Gravity permite a implantação dinâmica de componentes através da instalação de arquivos JAR a partir de URLs, de maneira semelhante ao que acontece no presente trabalho com a instalação de arquivos APK, específicos da arquitetura Android.

3.4 AlfredO

AlfredO [18] é uma arquitetura que permite o desenvolvimento de aplicações separadas em três camadas, de tal forma que a camada de apresentação possa ser executada em um dispositivo móvel, com poucos recursos computacionais, enquanto as outras, lógica e de informação, sejam executadas em outro dispositivo eletrônico, chamado pelos autores de dispositivo alvo (*target device*).

Além de promover escalabilidade e flexibilidade por meio da distribuição das diferentes camadas em diferentes dispositivos, a arquitetura provê independência ao dispositivo, separando as abstrações de interface visual de suas implementações. Ou seja, existe uma aplicação no dispositivo móvel que transforma a abstração de uma interface visual em componentes visuais.

A arquitetura também atende a requisitos de segurança, pois as camadas lógica e de informação estão desacopladas da camada de apresentação, executadas em dispositivos diferentes. Existe a possibilidade, entretanto, de migrar explicitamente a camada lógica do dispositivo alvo para o dispositivo móvel quando uma execução local é preferível e não afeta a segurança do sistema.

Por meio desta separação em camadas, *AlfredO* realiza adaptação dinâmica em sua topologia, modificando em tempo de execução a estrutura de aplicações para atender a requisitos de integração ubíqua entre diferentes dispositivos eletrônicos.

AlfredO é implementado por meio de R-OSGi [19], *middleware* que permite a distribuição de módulos OSGi de uma aplicação em diferentes dispositivos. Os módulos OSGi, por sua vez, implicam a componentização da aplicação e permitem atualizações e extensões de aplicações Java de maneira simples.

O *framework* OSGi, também sobre a plataforma Java, gerencia aspectos referentes à implantação de serviços, como dependência entre pacotes. O sistema estende o conceito de pacotes Java, presentes somente em tempo de desenvolvimento, ao tempo de execução como módulos OSGi, ou *bundles*. Deste modo, soluciona problemas de dependência por isolar o *classpath* de cada um dos módulos, além de permitir a instalação, atualização e remoção de módulos durante sua execução.

OSGi utiliza serviços dinâmicos, criados por meio de objetos Java que implementam qualquer interface. Por meio da arquitetura baseada em serviços o *framework* realiza a amarração, em tempo de execução, de implementações para determinadas interfaces. Assim, o *framework* resolve as dependências entre diferentes módulos durante a execução.

A arquitetura provê percepção de contexto dos dispositivos móveis para verificar a capacidade de processamento, conectividade e memória livre dos dispositivos e definir se são capazes de executar, além da camada de apresentação, também a camada lógica, se isso não interferir na segurança do sistema.

A arquitetura não provê persistência de estado das camadas que sofrem adaptação topológica, uma vez que, após sua utilização, toda a informação referente à aplicação no dispositivo móvel é descartada.

3.5 Roam

Roam [20] é um framework, implementado em Java, de adaptação dinâmica topológica, uma vez que permite a migração de componentes de software entre diferentes dispositivos mantendo seu estado interno. Implementado em Java, utiliza RMI (*Remote Method Invocation*), serialização e reflexão para promover, em tempo de execução, a instanciação dinâmica de diferentes implementações do mesmo componente, cada uma adequada a determinado tipo de dispositivo.

A arquitetura permite também a distribuição de aplicações para suprir limitações computacionais de dispositivos móveis, migrando componentes que necessitam de mais recursos para dispositivos mais adequados, como servidores.

A interface visual das aplicações sobre esta plataforma é construída de modo a tornar-se independente do dispositivo por meio de um *toolkit* provido

pelo sistema *Roam*.

As aplicações podem ser construídas por meio da composição de três tipos diferentes de componentes: múltiplos e dependentes do dispositivo (M-DD); único e dependente do dispositivo (S-DD); ou único e independente do dispositivo (S-DI). Assim, é possível determinar o componente mais adequado para execução em cada dispositivo.

A arquitetura define componentes, como o *middleware* apresentado neste trabalho, compostos de modo a formar aplicações – chamadas *Roamlets*.

Da mesma maneira que proposto no presente trabalho, a implementação mais adequada é escolhida em tempo de execução em função do contexto computacional.

3.6 iPOJO

iPOJO [21] é um *framework* de componentes orientado a serviços, implementado em Java sobre a plataforma OSGi, que permite o desenvolvimento de aplicações adaptáveis. A arquitetura de componentes orientada a serviços se assemelha à do *middleware* Kaluana em termos da composição entre os componentes. Entretanto há diferenças no aspecto de configuração dos componentes. Enquanto o *middleware* Kaluana define um serviço padrão que todo componente implementa, iPOJO trata a configuração através da definição de um *container* para cada componente, que gerencia a disponibilidade dos serviços providos pelo componente.

Seu nome deriva da expressão *injected Plain Old Java Objects*, pois os objetos Java que representam componentes são complementados com código binário injetado para prover a transparência ao tratamento da publicação, descoberta e uso de serviços.

A abordagem do *container* no iPOJO é semelhante à do serviço de configuração no Kaluana, pois são o ponto de conexão dos componentes com o *middleware*. A utilização de abstrações que representam componentes como classes Java e a utilização de reflexão computacional para complementar essas classes com funcionalidades referentes a gerenciamento de serviços são outros pontos onde iPOJO se assemelha ao Kaluana.

A composição no *framework* iPOJO é realizada por meio de arquivos descritores em XML, que definem quais classes representam componentes e seus serviços oferecidos e requisitados. No Kaluana, a composição é realizada de forma programática.

3.7 Comparações

A tabela 3.1 mostra a comparação entre os sistemas apresentados em termos dos quesitos discutidos no início deste capítulo.

Sistema	Adaptação dinâmica	Modelo	Plataforma
OpenCom	Estrutural	Modelo de componentes	Independente
SCA	Funcional	Orientado a serviços	Independente
Gravity	De extensão, estrutural	Modelo de componentes orientado a serviços	OSGi
AlfredO	De extensão, na topologia	Orientado a serviços	R-OSGi
Roam	Reativa, na topologia	Modelo de componentes distribuído	Java
iPOJO	Reativa, estrutural	Modelo de componentes orientado a serviços	OSGi
Kaluana	Reativa, funcional e estrutural	Modelo de componentes orientado a serviços	Android Java

Tabela 3.1: Tabela comparativa entre os trabalhos relacionados

OpenCOM é um modelo de componentes que prevê reconfiguração dinâmica, desta maneira provendo capacidade de adaptação dinâmica estrutural. SCA é um modelo orientado a serviços que define componentes como conjuntos de serviços agrupados.

O projeto *Gravity* tem como objetivo a extensibilidade e, portanto, provê a implantação e utilização de novas funcionalidades em tempo de execução. Para isso, utiliza modelo de componentes orientado a serviços, de maneira semelhante ao *middleware* Kaluana. O projeto, entretanto, não dá suporte à ciência de contexto e persistência de estado.

AlfredO tem como objetivo promover integração ubíqua entre diversos dispositivos eletrônicos com desempenho, flexibilidade e capacidade de adaptação, de modo a maximizar as interações entre os dispositivos. Assim, utiliza uma abordagem orientada a serviços. Para tal, promove modificação na topologia das aplicações, migrando camadas de serviços entre diferentes dispositivos utilizando a plataforma R-OSGi [19].

O projeto *Roam* tem como objetivo permitir a migração de aplicações entre diferentes dispositivos, motivado pela continuidade na execução de aplicações em diferentes dispositivos de um mesmo usuário. Deste modo, promove adaptação reativa na topologia da aplicação, respaldado por um modelo de componentes.

iPOJO é um *framework* de componentes orientado a serviços para desenvolvimento de aplicações dinamicamente adaptáveis, que tem por objetivo a simplicidade no desenvolvimento de aplicações. Para tal, o *framework* representa componentes por meio de classes Java. Os meta-dados das classes que

representam componentes descrevem os seus pontos de conexão e configurações, de maneira semelhante ao *middleware* Kaluana.

A abordagem de *Gravity*, *AlfredO* e *iPOJO* utiliza orientação a serviços para dar suporte à adaptação estrutural e funcional. *Roam* implementa um sistema distribuído para realizar adaptações topológicas. *OpenCOM* e *SCA*, por sua vez, definem modelos de componentes capazes de prover reconfiguração, oferecendo assim suporte à adaptação dinâmica.