6
Estudo de caso

A utilização de um modelo de componentes orientado a serviços possibilita a construção de aplicações por meio da conexão entre componentes em tempo de execução. O *middleware* Kaluana utiliza-se deste modelo para simplificar a implementação de aplicações e componentes adaptáveis.

Os objetivos do estudo de caso apresentado neste capítulo são demonstrar a simplicidade de criação de aplicações e componentes, bem como a capacidade de adaptação e implantação dinâmicas das aplicações implementadas sobre o *middleware*.

Para tal, foi escolhida a tarefa de transformar uma aplicação *stand-alone* Android em uma aplicação colaborativa, capaz de oferecer mecanismos para que seus componentes realizem publicação e subscrição de mensagens assíncronas entre diferentes dispositivos. Esta aplicação é também adaptável segundo o contexto, substituindo componentes baseada na mudança de localização do dispositivo. Assim, a aplicação final consiste de mapas que podem ser alternados, onde podem ser marcados pontos por diferentes usuários de forma colaborativa.

Neste cenário, cada dispositivo executa uma aplicação sobre uma instância do *middleware* Kaluana, e o componente chamado *Collaboration* provê a comunicação entre os dispositivos. Este componente pode ser utilizado para a criação de uma aplicação colaborativa baseada em mapas, em que usuários podem trocar informações sobre pontos marcados em um mapa compartilhado.

O mecanismo de adaptação implementado é baseado em localização, e faz uso de um provedor de localização semântica para transformar as coordenadas geográficas obtidas da plataforma Android em representação semântica, usada pelos componentes para registrar seu interesse em determinadas localidades, representadas por domínios de localização, mostrados na seção 6.1.

Como base para o estudo de caso foram tomadas duas aplicações simples, baseadas em mapas, desenvolvidas sobre a arquitetura Android. Também foi utilizado um módulo que provê as funcionalidades de publicação e subscrição.

As duas aplicações baseadas em mapas foram transformadas em componentes, chamados *Map* e *MapPUC*. Ambos conhecem a interface oferecida pelo componente de colaboração e implementam esta interface em um de seus recep-

táculos. O componente Kaluana de colaboração que oferece como um dos seus serviços o mecanismo de publicação e subscrição foi chamado de *Collaboration*.

Os componentes baseados em mapas são adaptáveis e o *middleware* pode alterná-los a qualquer momento, pois ambos pertencem à mesma categoria. O componente *Map*, funciona corretamente em qualquer localização, enquanto o *MapPUC* só é executado adequadamente nas dependências da PUC-Rio.

A Figura 6.1 mostra a arquitetura da aplicação implementada.

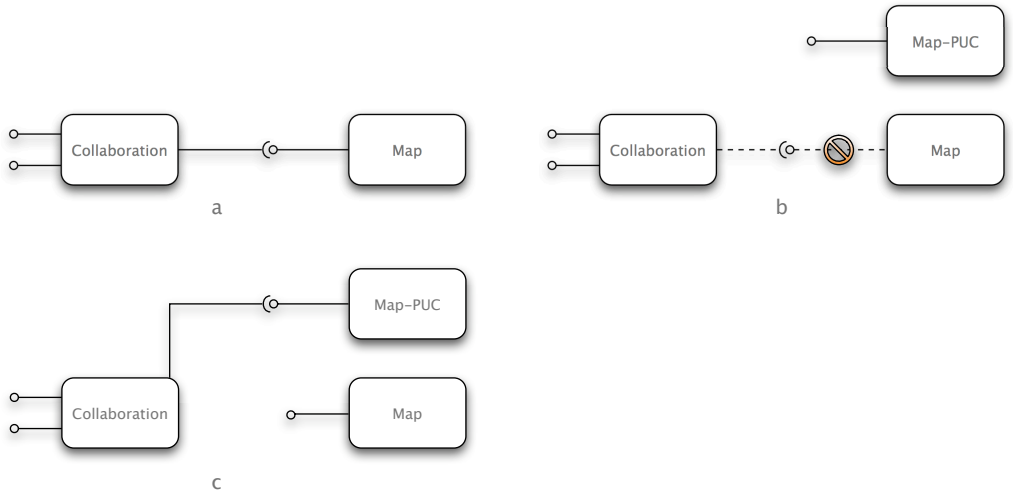

Figura 6.1: Arquitetura da aplicação

As notificações de variação de contexto foram realizadas manualmente por meio do emulador Android, verificando a reação da aplicação em função das coordenadas enviadas. Quando as coordenadas representam o espaço geográfico da PUC-Rio, o componente de navegação na PUC é ativado, enquanto que nas demais localidades, o componente genérico de mapa é mostrado. A conexão com o componente *Collaboration* é realizada em todas as adaptações.

A localização é captada a partir de diversas fontes, como sensores GPS no dispositivo ou a transformação da identificação da antena celular em coordenadas de localização. A informação proveniente do dispositivo móvel é representada por meio de coordenadas geográficas, que são transformadas em localização semântica, inteligível pelo ser humano, por meio do provedor de localização semântica.

Todos os testes foram simulados por meio do emulador do dispositivo, da ferramenta ADB (*Android Debug Bridge*) e o IDE (*Integrated Development Environment*) Eclipse.

Como exemplo de implementação do Gerenciador de Adaptações, que pode ser sobrescrito pelo desenvolvedor da aplicação, foi desenvolvido um mecanismo capaz de substituir diferentes implementações baseados em informa-

ções semânticas de localização. Estas informações de localização foram representadas por domínios de localização.

6.1 Domínios de localização

Domínios de localização definem as localizações do dispositivo que contém o componente adequadas para sua execução. Esta definição é realizada por meio de expressões de domínios de localização, de maneira semelhante à realizada no modelo Nimbus [22].

Um domínio representa a localização semântica de um lugar por meio da concatenação da identificação do lugar que o contém com a sua própria identificação. Por exemplo, o domínio que representa o Brasil é *br*, enquanto o domínio que representa o estado do Rio de Janeiro é *br.rj*, a cidade do Rio de Janeiro é representada por *br.rj.rio* e o domínio que representa a PUC-Rio é *br.rj.rio.puc*. O último domínio é um sub-domínio do primeiro.

O desenvolvedor do componente pode registrar interesse em domínios onde seu componente esteja apto à execução. Assume-se que a localização do dispositivo em todo sub-domínio do domínio registrado também implica a correta execução do componente. Ou seja, se um componente ativo registra seu correto funcionamento dentro da universidade PUC-Rio, ao detectar a saída da PUC-Rio para um super-domínio o componente pode não ser substituído, a menos que haja componente registrado para domínio mais genérico entre a PUC-Rio e o domínio atual.

Por exemplo, enquanto a localização for representada pelo domínio *br.rj.rio*, o componente registrado para *br.rj.rio.puc* poderá continuar executando, a menos que haja componente para a mesma categoria registrado para o domínio *br.rj.rio*.

A execução só se torna de fato inviável quando o seu domínio e o domínio definido pelo contexto são diferentes e nenhum dos dois é sub-domínio do outro. Por exemplo, um componente registrado para *br.rj.rio.puc* não operará em um contexto representado por *br.rj.rio.ufrj*. Neste caso, o componente será substituído por outro que registre interesse para *br.rj.rio.ufrj* ou algum de seus super-domínios, como *br.rj*.

Existe um domínio básico chamado *any*. Caso a execução de um componente não seja influenciada por mudanças na localização do dispositivo, o componente deve registrar seu contexto de execução para este domínio básico, indicando que o componente não precisa ser substituído.

A informação geométrica sobre localização geográfica provida pelo GPS do dispositivo é transformada em informação semântica por meio de um serviço

remoto provedor de localização semântica, mostrado a seguir.

Os domínios de localização devem ser conhecidos *a priori* pelo desenvolvedor dos componentes, para que possam registrar seu interesse em localidades específicas.

6.1.1 Escolha do alvo da adaptação

Se existir um componente inativo para um sub-domínio do domínio de um componente ativo e ambos tiverem a mesma categoria, ao entrar na região representada pelo sub-domínio, ocorrerá uma adaptação dinâmica de otimização, através da substituição de um componente pelo outro.

Se, por outro lado, um componente ativo não tiver interesse registrado para um novo domínio adentrado e nenhum de seus sub-domínios, entende-se que o componente não é adequado. Se houver outro componente com a mesma categoria e adequado para o novo domínio, o componente inadequado é substituído em uma adaptação dinâmica corretiva. Caso contrário, nenhuma ação é tomada, e o componente inadequado se mantém ativo.

6.2 Provedor de localização semântica

O provedor de localização semântica é um componente executado no dispositivo que funciona como fachada para um serviço *web*, em que foram cadastrados manualmente intervalos de latitude e longitude pertencentes a determinados domínios, para efeito de testes.

Para não sobrecarregar o *middleware* com verificações contínuas sobre a necessidade de adaptação enquanto o dispositivo está em movimento, foram criadas duas restrições para a realização da verificação. A primeira em função do tempo, e a segunda em função da distância percorrida. Uma nova verificação só é realizada quando o usuário se move pelo menos um determinado espaço ou no máximo quando transcorrido o período de tempo definido.

O provedor de localização pode ser estendido para um serviço colaborativo, onde os usuários podem cadastrar novos domínios para suas localidades, de modo a prover mais informações sobre localização aos desenvolvedores de componentes. Neste trabalho, entretanto, as informações foram cadastradas programaticamente, de maneira estática.

6.3 Passo a passo

Para transformar uma aplicação Android baseada em mapas em um componente Kaluana, adiciona-se ao pacote uma classe Java que represente o componente. Então, na inicialização do componente chama-se a atividade principal da aplicação, que seria chamada normalmente pela plataforma Android. O serviço de configuração deste componente deve ser registrado no arquivo descritor de projeto, *AndroidManifest.xml*.

Neste componente, definem-se seus serviços e receptáculos. No caso das aplicações baseadas em mapas, definiu-se um receptáculo que implementa a interface *publish-subscribe*, enquanto no componente *Collaboration* definiu-se um serviço que implementa a mesma interface. Uma vez definidos, receptáculos e serviços são utilizados normalmente pelos componentes.

A Figura 6.2 mostra o código-fonte do componente *Collaboration* e seu serviço *publish-subscribe*, neste exemplo chamado *sdm*.

```
package kaluana.sdm;

...

@Component(category="kaluana.collaboration")
public class Collaboration extends kaluana.impl.Component {

    @Service
    ISDM sdm;

}
```

Figura 6.2: Código-fonte do componente *Collaboration*

Com estas modificações, o componente de colaboração pode ser utilizado por outros componentes e aplicações por meio de sua interface *publish-subscribe*, desde que conheçam sua interface.

Além do componente de colaboração, outros dois componentes foram implementados. A Figura 6.3 mostra as implementações destes dois componentes, baseados em mapas, que oferecem os mesmos serviços e receptáculos, de modo a poderem ser substituídos durante a execução.

A aplicação, por sua vez, realiza a composição programática entre os componentes baseado em mapas com e sem capacidade de colaboração, como mostrado na Figura 6.4. Ao requisitar a ativação dos componentes, a aplicação define um método de *callback* que será chamado ao final do processo, em que são definidas as composições e a utilização dos componentes.

```
package kaluana.examples.navigator;

...

@Component(category=
    "kaluana.examples.navigator")
public class NavigatorComponent extends
        kaluana.impl.Component {

    @Receptacle
    ISemanticLocationService
        semanticLocation;

    @Service
    INavService navService;

    @Receptacle
    ISDM sdm;

    @Override
    public void start()
        throws RemoteException {
        ... // Start Google Maps Activity
    }

    ...
}
```

```
package kaluana.examples.navigator;

...

@Component(category=
    "kaluana.examples.navigator")
public class NavigatorPUCComponent extends
         kaluana.impl.Component {

    @Receptacle
    ISemanticLocationService
        semanticLocation;

    @Service
    INavService navService;

    @Receptacle
    ISDM sdm;

    @Override
    public void start()
        throws RemoteException {
        ... // Start PUC-Rio Map Activity
    }

    ...
}
```

Figura 6.3: Componentes baseados em mapas

```
    @Override
    public void componentsLoaded(List<String> components)
            throws RemoteException {
        sdmConfig = componentManager.getComponent("kaluana.sdm");
        mapConfig = componentManager.getComponent("kaluana.map");

        ...

        mapConfig.bindReceptacle(receptacleInfo, serviceImpl, serviceInfo);

        // Bind other services and receptacles

        mapConfig.start();
    }
```

Figura 6.4: Implementação da aplicação

6.4 Discussão

A diferença de complexidade entre implementar uma aplicação baseada em mapas diretamente sobre a arquitetura Android e uma aplicação semelhante, porém dinamicamente adaptável, sobre o Kaluana se resume à adição de duas classes Java, uma representando o componente e outra realizando a composição, e a inclusão dos arquivos AIDL que definem as interfaces destes componentes.

Os componentes de mapas colaborativos não conhecem detalhes de implementação do componente de colaboração, diferentemente do que acontece com implementações sobre a plataforma Android, em que qualquer aplicação colaborativa deve importar um pacote Java com a implementação completa do componente de colaboração. Deste modo, a amarração entre os componentes

torna-se mais fraca, uma vez que componentes não têm dependência em função da implementação de outros, mas exclusivamente em função de suas interfaces.

Entretanto, como o componente de colaboração não foi desenvolvido inicialmente como um componente, nem todas as dependências foram definidas em função de sua implementação, como as classes *Event* e *Reply*, que são compartilhadas por mais de um componente. Em uma abordagem de modelo de componentes como a do Kaluana, estes detalhes são escondidos ou transformados contratualmente em interfaces. O componente de colaboração modifica estas dependências transformando-as em interfaces também utilizadas pelos clientes.

De fato, a transformação de aplicações implementadas sobre a plataforma Android em aplicações sobre o *middleware* Kaluana requer pouco esforço e não adiciona qualquer complexidade à aplicação. A simplicidade oferecida pelo *middleware* Kaluana para a implementação de aplicações móveis dinamicamente adaptáveis advém tanto da reutilização de componentes quanto da transparência para o desenvolvedor de requisitos não-funcionais, como a publicação, descoberta e utilização de serviços.