4
Arquitetura do middleware

Ao utilizar um modelo de componentes orientado a serviços, o *middleware* Kaluana divide o processo de desenvolvimento de aplicações em duas etapas: a primeira é o desenvolvimento dos componentes, a segunda é a formação de aplicações a partir de composições.

O desenvolvimento de componentes consiste na implementação de classes Java cujos atributos representem os serviços e receptáculos oferecidos por cada componente, além da implementação de seu comportamento interno e das interações com os demais componentes.

As composições, realizadas em tempo de execução, são definidas programaticamente pelo desenvolvedor de cada aplicação. A aplicação é resultado de uma composição. Sua implementação consiste em buscar os componentes reutilizáveis de interesse, representados por objetos Java, e conectá-los por meio do serviço de configuração que o *middleware* oferece para cada componente.

Ao realizar esta separação, o *middleware* Kaluana não só provê as funcionalidades de adaptação e implantação dinâmica como, ao utilizar abstrações adequadas e mecanismos de reflexão computacional, simplifica o processo de desenvolvimento de aplicações móveis.

Componentes têm suas interfaces e receptáculos definidos contratualmente e estes são seus únicos pontos de conexão com outros componentes e com as aplicações. As interfaces são implementadas por serviços, aos quais receptáculos de outros componentes conectam-se.

O *middleware* utiliza reflexão computacional para obter informações estruturais e internas aos componentes para instanciar seus serviços e receptáculos, bem como resolver suas dependências. Desta maneira, oferece ao desenvolvedor de componente a abstração de requisitos não-funcionais, como publicação, descoberta e uso de serviços e resolução de dependências entre componentes.

A capacidade de realização de composições em tempo de execução permite o funcionamento do mecanismo de adaptação estrutural dinâmica. Utilizando mecanismos de ciência de contexto, o modelo de componentes pode modificar sua estrutura de acordo com as condições computacionais ou do

usuário. No Kaluana, a percepção ao contexto é resultado da utilização de mecanismos de acesso ao contexto computacional providos pela plataforma Android, mostrada na seção 4.2, por exemplo informações sobre localização a partir de sensores GPS e da rede de telefonia celular.

A Figura 4.1 mostra a arquitetura do Kaluana. Aplicações são composições realizadas por meio de componentes gerenciados pela camada de *middleware*. O *middleware*, por sua vez, utiliza a plataforma Android para acessar informações de contexto, bem como aproveita-se de sua orientação a serviços para realizar a conexão entre componentes. As composições são formadas e alteradas em tempo de execução, baseadas nas informações providas durante o desenvolvimento de aplicações e componentes por meio das abstrações oferecidas pelo *middleware* para definição de seus pontos de conexão.

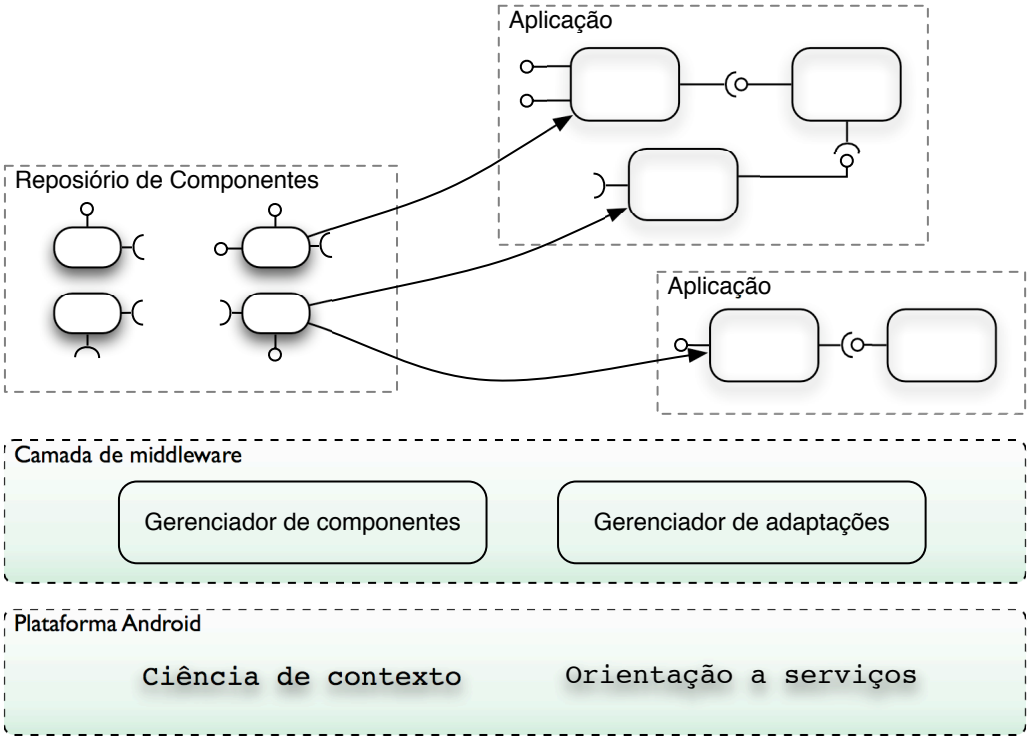

Figura 4.1: Arquitetura do middleware Kaluana

A arquitetura do *middleware* Kaluana é proposta como complementar à plataforma Mobilis [2], que provê colaboração e ciência de contexto, porém não é implementado sobre um modelo de componentes. Com a integração ao Kaluana, a arquitetura pode oferecer funcionalidades de adaptação e implantação dinâmicas, além do reuso de componentes.

Algumas premissas quanto à utilidade e aplicabilidade do *middleware* são explicitadas na seção a seguir. Em seguida, são apresentados o modelo de componentes orientado a serviços implementados pelo *middleware*, as técnicas

utilizadas para prover abstrações e oferecer agilizade aos desenvolvedores e o funcionamento dos mecanismos de adaptação e implantação dinâmica.

4.1 Premissas

O *middleware* é desenvolvido em Java sobre a plataforma Android, como mostra o capítulo 5. Sendo assim, sua aplicabilidade limita-se a aplicações móveis construídas sobre esta plataforma.

Componentes, assim como seus serviços oferecidos, não possuem versão. Apesar do atual modelo, deseja-se que componentes ofereçam diversas versões e suas dependências sejam explícitas em função das versões dos componentes e serviços.

O *middleware* assume a conectividade do dispositivo durante processos de implantação dinâmica, uma vez que é necessário o acesso ao repositório remoto de componentes para seu correto funcionamento.

4.2 A plataforma Android

O modelo e o *middleware* Kaluana foram definidos e implementados sobre a plataforma móvel Android, baseada em Linux. A plataforma é orientada a serviços, e oferece facilidades como comunicação assíncrona entre processos e diversas fontes de informação de contexto.

Uma aplicação Android é composta por atividades (*Activities*) e serviços (*Services*). Cada aplicação Android, incluindo suas atividades e serviços, executa um processo Linux próprio e pode ser implantada no dispositivo móvel por meio de um arquivo de extensão .apk.

Atividades e serviços são iniciados por meio de intenções (*Intents*), mensagens assíncronas trafegadas entre diferentes processos ou dentro de um mesmo processo, cujo conteúdo é representado por um objeto do tipo *android.content.Intent*. Atividades e serviços registram interesse, por meio do arquivo *AndroidManifest.xml*, em tempo de desenvolvimento, em determinados tipos de intenção, representados por *Strings*.

O desenvolvedor de uma aplicação inicia explicitamente uma atividade ou serviço por meio de uma intenção. Se houver pelo menos uma atividade ou serviço com interesse registrado para o conteúdo daquela intenção, um deles será iniciado.

4.2.1 Atividades

Atividades representam interfaces visuais da aplicação com o usuário. Toda a interação com o usuário se dá através das atividades. Por exemplo, um mapa mostrado na tela do dispositivo é resultado de uma atividade, implementada em uma classe que estende *com.google.android.maps.MapActivity*, que por sua vez estende *android.app.Activity*. Se sobre este mapa for mostrado um formulário para criação de um ponto de interesse, o formulário será exibido por outra atividade.

Cada atividade é associada a uma janela, que pode ter qualquer tamanho até o limite da tela do dispositivo. Uma mesma atividade pode mostrar, além de sua janela principal, caixas de diálogo ou alertas ao usuário, mas não mais que isto. Qualquer outra janela será exibida por outra atividade.

O conteúdo visual de uma atividade é controlado por visões (Views), objetos derivados da classe *android.view.View*. Cada visão é reponsável por um espaço retangular da janela. Visões podem ser alternadas e modificadas a qualquer momento pela atividade.

Tipicamente, toda aplicação Android tem uma atividade principal, apresentada ao usuário quando a aplicação é iniciada.

4.2.2 Serviços

Serviços, por outro lado, não têm interface visual. São tarefas que executam por período de tempo indeterminado. Um serviço pode, por exemplo, tocar uma música enquanto o usuário utiliza o navegador web do dispositivo, ou ainda pode prover informações sobre a localização sempre que o usuário se move. Serviços estendem a classe *android.app.Service*.

Os serviços oferecem interfaces descritas pela linguagem AIDL (*Android Interface Description Language*), pelas quais podem ser utilizados, após conectado a seu invocador. Conexões com serviços podem existir dentro de um mesmo processo ou entre processos diferentes. As conexões são criadas pelos métodos *bindService()* ou *startService()*, providos pelo objeto de contexto Android, explicado adiante. Ambas as chamadas podem criar uma instância de um serviço, se uma já não existir.

Ao conectar a um serviço utilizando-se o método *bindService()*, a plataforma retorna, de forma assíncrona, uma referência do tipo *android.os.IBinder*, que corresponde à referência para sua implementação. Essa referência será utilizada pelo *stub* do cliente para acessar os métodos definidos no arquivo AIDL descritor da interface do serviço. O cliente, que também deve conhecer

a interface, pode utilizar o método *asInterface()* do *stub* para fazer chamadas diretamente à referência da interface local.

Por exemplo, a interface do serviço *Location Service*, descrita no arquivo *ILocationService.aidl* e implementada na classe *ILocationService* pode ser utilizada por meio de *ILocationService.Stub.asInterface(IBinder service)*. A chamada a este método retorna um objeto do tipo *IlocationService*, cujas chamadas a seus métodos serão executadas pelo serviço remoto e seus resultados retornados ao invocador do serviço.

4.3 Modelo de componentes orientado a serviços

O *middleware* Kaluana define um modelo de componentes sobre a plataforma Android, que por sua vez é orientada a serviços. Um componente oferece e consome qualquer quantidade de serviços. O uso dos serviços se dá por meio de receptáculos, que conectam-se aos serviços de outros componentes.

Cada serviço implementa uma determinada interface. Para usar um serviço, um componente deve ter um receptáculo que espere ser conectado à mesma interface.

Além de receptáculos e serviços, componentes também possuem dependências em função de outros componentes, que são resolvidas durante o processo de ativação, como mostrado na seção 4.4.

O processo de composição é realizado através de uma interface de programação (*Application Programming Interface*). O desenvolvedor da aplicação tem acesso à estrutura de cada componente e aos métodos utilizados para realizar conexões entre serviços e receptáculos. Uma vez ativos, componentes podem ser conectados pelo desenvolvedor de aplicações, formando composições em tempo de execução.

O gerenciador de componentes, mostrado na seção 5.1.1, mantém referências para todos os componentes ativos no dispositivo, através de seus serviços de configuração. Deste modo o *middleware* pode comparar a condição de contexto a cada momento com as condições requeridas por cada componente, e decidir por adaptar um ou mais componentes.

A próxima seção descreve o ciclo de vida de componentes, bem como a maneira que o *middleware* os gerencia.

4.4 Componentes e seu ciclo de vida

Componentes oferecem serviços e receptáculos, que representam seus pontos de conexão com outros componentes e com o *middleware*. Com-

ponentes são representados por classes que estendem a superclasse *kaluana.impl.Component*. Esta classe define métodos para sua configuração, ativação, utilização e desativação. Estes métodos instanciam seus serviços, receptáculos e dependências, e são utilizados pelo *middleware* ou pela aplicação em cada uma das etapas de seu ciclo de vida.

O desenvolvedor do componente não precisa implementar nenhum método para tornar disponíveis ou utilizar serviços e receptáculos, todos são implementados pelo *middleware*. Além destes, alguns métodos de *callback*, como os chamados após a ativação ou antes da desativação de um componente, podem ser opcionalmente implementados. Assim, é possível desenvolver um componente de maneira simples, sem a necessidade de sobrescrever nenhum método desta classe.

Pela arquitetura do *middleware* Kaluana, cada componente define um serviço específico para sua ativação e configuração. Deste modo, a classe que representa o componente não deve ser acessada diretamente, pois diferentes componentes podem ser executados em diferentes processos.

Para permitir seu acesso remoto, todo componente possui um serviço de ativação e configuração, o Serviço de Configuração, que implementa métodos homônimos e simplesmente os delega para a classe que representa o componente. Este serviço, apresentado na seção 5.1.4, é implementado pelo Kaluana e transparente para o desenvolvedor do componente.

As características dos componentes, como a lista de seus serviços e receptáculos, bem como suas dependências são definidas em tempo de desenvolvimento por meio de meta-dados nas classes que o representam. Estes meta-dados são então obtidos pelo *middleware* durante a execução por meio de reflexão computacional. Para construir o componente a partir destas características, o *middleware* realiza uma busca pela definição de serviços, receptáculos e dependências na classe que representa o componente.

Kaluana utiliza anotações Java para descrição de meta-dados referentes ao componente, como seus receptáculos e serviços oferecidos. Foi considerado que utilizar anotações torna o desenvolvimento de componentes mais ágil, uma vez que o desenvolvedor tem uma visão mais clara da estrutura de um componente pela síntese desses meta-dados na definição da classe Java que o representa.

Serviços são definidos por meio da anotação *@Service* e receptáculos por meio da anotação *@Receptacle* sobre campos da classe Java que implementam a interface de cada serviço e receptáculo. Ambas as anotações devem ser utilizadas em atributos cujo tipo represente um serviço Android.

No início do ciclo de vida de um componente, Kaluana procura por todos

os seus serviços e os instancia, se já não tiverem sido iniciados. Quando uma aplicação solicita a conexão entre diferentes componentes, Kaluana injeta a dependência do serviço no receptáculo do componente que o utilizará.

A classe que representa o componente também deve ser anotada com *@Component*, e opcionalmente descrever dependências a outros componentes por meio da anotação *@Dependencies*, de maneira análoga ao componente mostrado na Figura 4.2. A figura mostra a implementação do componente de nome *PingComponent*, que oferece o serviço *ping*, que implementa a interface *IPingService*, e o receptáculo *pong*, que implementa a interface *IPongService*.

O *middleware* Kaluana, por meio de reflexão computacional, inspeciona a classe que representa o componente para descobrir seus serviços e, antes da ativação do componente, vincula uma implementação específica a cada serviço oferecido por cada componente. Assim, é possível descobrir, dada a interface do serviço, as suas implementações disponíveis no dispositivo. Assim, o *middleware* seleciona, no processo de ativação do componente ou de uma adaptação dinâmica, a implementação mais adequada para cada faceta oferecida pelo componente.

```
12 @Component(category="kaluana.examples.ping")
13 @Dependencies({@Dependency(name="pongDependency",
14         componentCategory="kaluana.examples.pong")})
15 public class PingComponent extends kaluana.impl.Component {
16
17     @Service
18     IPingService ping;
19
20     @Receptacle
21     IPongService pong;
22
23 }
24
```

Figura 4.2: Exemplo da implementação de um componente

Representar componentes por meio de classes Java e utilizar mecanismos de reflexão computacional para realizar abstrações são maneira habitual de descrever aspectos de controle e tendem a simplificar o processo de desenvolvimento.

4.4.1 Ciclo de vida

O ciclo de vida de um componente é composto por seis etapas bem definidas: instalação, ativação, utilização, adaptação, desativação e desinstalação [9]. A Figura 4.3 demonstra estes processos.

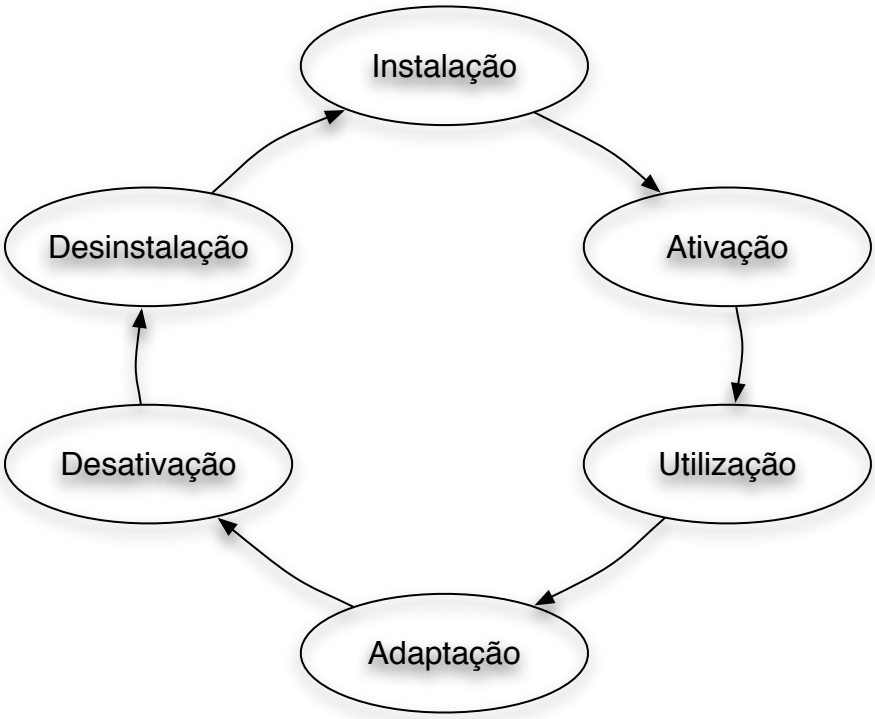

Figura 4.3: Exemplo da implementação de um componente

O modelo Kaluana não prevê, entretanto, a desinstalação de um componente, uma vez que ele pode novamente ser ativado após ter sido desativado.

A implantação dinâmica é composta pelos processos de instalação e ativação, enquanto a adaptação dinâmica pode implicar nas tarefas de desativação de um componente e ativação de outro, inclusive podendo resultar em uma implantação dinâmica, quando o componente não está previamente instalado no dispositivo.

O *middleware* Kaluana implementa mecanismos para tornar transparentes ao desenvolvedor de componentes aspectos referentes à descoberta, publicação e utilização de serviços, bem como adaptação e implantação dinâmica. Assim, torna mais simples tanto o desenvolvimento de componentes como a composição pelas aplicações.

Instalação

A instalação de um componente Kaluana consiste no desempacotamento e instalação do arquivo .APK, específico à plataforma Android, no dispositivo (ou no emulador) para tornar possível seu uso. Um arquivo .APK pode encapsular qualquer quantidade de componentes, além de uma ou nenhuma aplicação.

A instalação é realizada no processo de implantação dinâmica, após a

realização do *download* do arquivo que contém o componente buscado em um repositório remoto, como mostrado na seção 4.6. O processo de instalação é iniciado quando o Gerenciador de Componentes requisita ao Repositório de Componentes a ativação de um componente inexistente no dispositivo.

Ativação

A ativação consiste na instanciação da classe que representa o componente e na criação, em tempo de execução, dos seus serviços e receptáculos baseada nas definições indicadas em tempo de desenvolvimento.

O processo de ativação, realizado pelo Gerenciador de Componentes, pode ocorrer como resultado da solicitação de um componente pela aplicação ou de uma adaptação dinâmica. No segundo caso, o processo deve implicar na criação das mesmas conexões que o componente anterior matinha, bem como no reestabelecimento de estado interno similar ao do componente substituído.

O procedimento de restauração das conexões no componente substituto é realizado pelo *middleware* Kaluana e é transparente para o desenvolvedor do componente. O re-estabelecimento do estado interno, por sua vez, fica a cargo do desenvolvedor do componente, como mostrado na seção 4.5.2.

Aplicações podem requisitar a ativação de componentes de duas maneiras: a partir seu nome ou de sua categoria. A categoria define quais serviços e receptáculos um componente oferece. Um componente só pode substituir outro se ambos pertencerem à mesma categoria. A categoria é definida como uma expressão que representa semanticamente o conjunto de funcionalidades daquele componente, enquanto o nome representa a implementação específica daquele componente.

Cada componente tem uma categoria, representada por uma expressão criada pelo seu desenvolvedor. O desenvolvedor da aplicação, por sua vez, deve conhecer a expressão que representa a categoria do componente que deseja utilizar. Caso o desenvolvedor da aplicação queira utilizar um componente específico, e outros da mesma categoria não lhe sejam úteis, a requisição pode ser realizada em função do nome do componente desejado, que deve ser a concatenação de sua categoria com o nome da classe que o representa, separados por ponto.

Por exemplo, dois componentes de visualização de mapas alternativos, para diferentes localidades, dentro ou fora do espaço físico da universidade PUC-Rio, podem ter seus nomes *PUCNavigatorComponent* e *NavigatorComponent*, respectivamente, enquanto ambos pertencem à categoria *kaluana.examples.navigator*.

Como a requisição por um componente pode demandar sua implantação,

o processo de ativação é realizado de forma assíncrona. Ou seja, a aplicação requisita ao *middleware* a ativação de determinado componente e é notificada, ao fim do processo, do sucesso ou falha na ativação. Para buscar o componente adequado à cada requisição, o *middleware* utiliza o registro de serviços da plataforma Android, realizado por arquivos descritores dos pacotes instalados, como mostrado na seção 4.2. Deste modo, é possível realizar a busca por componentes de maneira semelhante à busca por serviços, a partir de um registro, sem a necessidade de verificar o conteúdo de pacotes ou instanciar objetos.

Antes de retornar à aplicação a referência para o serviço de configuração do componente, o *middleware* realiza a instanciação de todos os seus serviços. Somente quando todos os serviços de um componente forem iniciados, o Gerenciador de Componentes é notificado sobre a ativação deste componente, e pode repassar a notificação a quem solicitou tal ativação. Os receptáculos, por sua vez, só se conectarão aos respectivos serviços ao longo da utilização do componente.

Para garantir o correto atendimento de todas as requisições de ativação de componentes, o *middleware* empilha todas as chamadas, que podem requisitar múltiplos componentes, e inicia o processo para cada componente requisitado. A cada componente ativado assincronamente, o *middleware* verifica na pilha quais das chamadas foram concluídas com êxito e notifica o requisitante sobre o término da chamada. Assim, nenhuma das requisições deixa de ser atendida, bem como o número de notificações geradas é idêntico ao número de requisições.

Para resolver dependências entre componentes, a cada requisição, o *middleware* verifica as dependências do componente requisitado e as empilha na mesma chamada, caso ainda não existam. Sendo assim, só haverá sucesso na chamada se todos os componentes definidos como dependências forem instanciados. O empilhamento de chamadas garante que a ativação de um componente se dará após a resolução de suas dependências.

Ao final da ativação, o método *onActive()* do componente é chamado pelo *middleware*, assim finalizando seu processo de ativação. A implementação deste método é opcional, e o desenvolvedor do componente pode utilizá-lo para iniciar alguma tarefa específica ao início da utilização daquele componente, como iniciar ou uma execução em *background* ou sua interface visual (vide seção 5.3.3).

Outro método, *start()* pode ser opcionalmente implementado pelo componente e chamado pela aplicação para realizar tarefas específicas, tais como iniciar uma interface de usuário ou uma tarefa em *background*.

Utilização

O desenvolvedor de aplicações pode verificar os serviços e receptáculos disponíveis e realizar conexões entre componentes. Para isso, o desenvolvedor pode utilizar métodos providos pelo próprio componente. Estes métodos são implementados pelo *middleware* Kaluana e são transparentes para o desenvolvedor do componente.

Os métodos *getServiceNames()* e *getReceptacleNames()*, oferecidos pelo Serviço de Configuração de cada componente, retornam uma lista com os nomes de todos os serviços e receptáculos, respectivamente. A partir de qualquer um desses nomes, é possível buscar a referência para cada serviço e cada receptáculo por meio dos métodos *getServiceInfo()* e *getReceptacleInfo()*. Estes métodos retornam classes que representam a definição de um serviço e um receptáculo oferecido pelo componente, respectivamente.

Uma vez escolhidos um serviço e um receptáculo, é necessário mais um passo para conectá-los. O método *getService()* retorna uma referência para o *stub* cliente do serviço no processo Linux executado pelo requisitante, e esse *stub* deve ser passado ao componente que contém o receptáculo por seu método *bindReceptacle()*, juntamente às definições do serviço e dos receptáculos conectados.

Uma vez definidas todas as conexões, componentes passam a manter referências a serviços de outros componentes, assim podendo realizar as tarefas a que a aplicação se propõe. Serviços também podem ser chamados diretamente da aplicação, caso o desenvolvedor da aplicação assim o deseje.

A partir do momento em que a aplicação realiza a conexão entre um receptáculo e um serviço, por meio de reflexão computacional o *middleware* associa a implementação do serviço no componente provedor ao atributo que representa o receptáculo na classe cliente. Ou seja, o desenvolvedor do componente não precisa se preocupar com a descoberta dos serviços, uma vez que as implementações dos serviços serão injetadas em tempo de execução. Basta indicar quais são os receptáculos do componente e o *middleware* se encarrega de instanciar os serviços mais adequados, segundo as conexões realizadas pela aplicação.

Nenhum código de inicialização dos serviços ou receptáculos é implementado pelo desenvolvedor do componente. Toda a lógica de inicialização é controlada pelo *middleware* Kaluana. A única exigência é a implementação de um método *setter*, que recebe uma referência para a interface implementada pelo atributo e vincula a referência ao atributo, para cada receptáculo do componente, para que o *middleware* possa acessá-los.

Desativação

Na desativação de um componente Kaluana, todos os serviços do componente são interrompidos e seus receptáculos desconectados de serviços remotos. Em seguida, todos os seus serviços são finalizados explicitamente pelo *middleware* Kaluana. O componente, entretanto, continua instalado no dispositivo móvel, para o caso de vir a ser utilizado posteriormente.

De maneira análoga à ativação, ao término da utilização o desenvolvedor pode julgar necessário liberar algum tipo de recurso, o que pode ser implementado no método *onInactive()*. Um componente pode ser ativado novamente após ter sido desativado, uma vez que o Repositório de Componentes preserva a referência para este componente.

A aplicação pode explicitamente desativar um componente por meio de seu método *stop()*. A chamada a este método implica a execução do método *onInactive()*. A desativação é realizada pelo Gerenciador de Componentes, quando solicitado pelo Gerenciador de Adaptações ou por uma aplicação.

4.5 Mecanismo de adaptação dinâmica

Adaptações dinâmicas são realizadas pelo Gerenciador de Adaptações, descrito na seção 5.1.3. O Gerenciador de Adaptações é um serviço do *middleware* que utiliza um ou mais componentes provedores de contexto e verifica a existência de componente mais adequado ou a incapacidade de algum componente ativo em atender às condições de contexto a cada vez que detecta mudança significativa no contexto.

Duas situações podem motivar uma adaptação dinâmica. Um componente ainda inativo pode ser mais adequado à execução no contexto atual, e substituirá outro, que ainda poderia funcionar corretamente naquele contexto, ou um componente atual deixa de funcionar em uma nova condição de contexto, e será substituído por outro que possa continuar a execução da aplicação. Por exemplo, um componente que mostra o mapa da cidade do Rio de Janeiro ainda pode ser utilizado quando o usuário se localiza nas dependências da PUC-Rio, mas pode ser substituído por um componente que mostre o mapa da PUC-Rio. Por outro lado, um componente que mostra o mapa da PUC-Rio torna-se inadequado quando o usuário está em outra localidade.

O primeiro caso é classificado como adaptação de otimização, enquanto o segundo como adaptação corretiva. Ambas são também reativas, pois são resultado da mudança do contexto computacional.

Se não houver componente instalado no dispositivo capaz de atender à necessidade de uma adaptação dinâmica ou à requisição de ativação a partir

de uma aplicação, um processo de implantação dinâmica é iniciado, buscando em um repositório remoto por um componente que tenha as caracterísiticas necessárias para operar sob o contexto atual do dispositivo. O processo é mostrado na seção 4.6.

A figura 4.4 mostra um exemplo de adaptação estrutural dividido em três etapas. No primeiro momento, o componente A utiliza um serviço do componente B. Depois de uma mudança no contexto, o componente B passa a não ser mais adequado. Assim, o componente B é substituído pelo componente C. Em seguida, o componente C realiza as mesmas conexões que B mantinha, neste caso, com A.

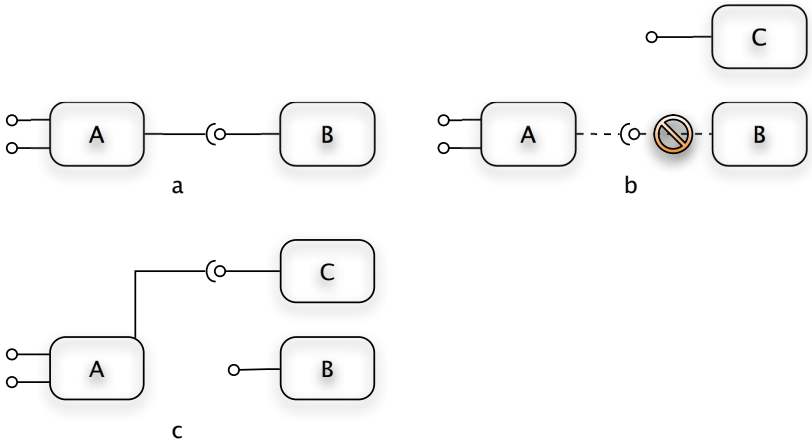

Figura 4.4: Etapas de uma adaptação dinâmica

Por exemplo, determinado componente provedor de localização, dependente da disponibilidade de sensores GPS, pode ser substituído por um componente semelhante, porém dependente de tecnologia *wi-fi* para realizar triangulação e determinar o posicionamento do dispositivo. O Gerenciador de Adaptações pode ser utilizado para definir políticas de adaptação e o momento em que cada substituição é realizada, baseado na disponibilidade das informações provenientes dos sensores.

O Gerenciador de Adaptações pode ser estendido pelo desenvolvedor da aplicação de modo a oferecer comportamentos baseados em informações de contexto específicas. O Gerenciador de Adaptações implementado no estudo de caso desta dissertação é um exemplo dentre diversas implementações possíveis.

4.5.1 Ciência de contexto

Diversas informações de contexto físico e computacional podem, em princípio, ser obtidas diretamente a partir dos dispositivos móveis, como taxa

de utilização de memória e CPU, conectividade e energia disponível até a localização do dispositivo, temperatura e luminosidade do ambiente.

Muitos tipos de informação de contexto são úteis e podem melhorar a experiência de usuários com aplicações móveis, e são resultado de análise de dados provenientes de um ou mais tipos de sensores ou de medições realizadas sobre a própria aplicação.

A plataforma Android permite o acesso programático a alguns dados de contexto, como a localização do dispositivo, nível de energia e conectividade, ou ainda a agenda de contatos do usuário.

Como o objetivo desta dissertação é o mecanismo de adaptação e implantação dinâmica, a implementação do *middleware* limitou as fontes de dados sobre contexto ao serviço de localização do emulador Android, baseado na simulação de sensores GPS.

4.5.2 Persistência de conexões e estado interno

Em um processo de adaptação dinâmica, o componente substituído provavelmente possui um estado interno a ser persistido, além das conexões a outros componentes por meio de seus serviços e receptáculos.

Essas conexões e estado interno devem ser reconstruídos após o processo de adaptação, de modo a minimizar a interferência do processo de adaptação dinâmica no comportamento das aplicações, além de manter a consistência em seu estado de execução.

A preservação das conexões é realizada automaticamente pelo *middleware* Kaluana, que, ao desativar qualquer componente, mantém armazenadas informações suficientes para reconstruir todas as conexões no novo componente.

O estado interno de um componente pode ser persistido pelo seu desenvolvedor, por meio de facilidades provenientes da plataforma Android de comunicação entre processos (IPC - *Inter Process Communication*), como mostrado na seção 5.2. O *middleware* Kaluana se aproveita dessas facilidades para permitir que um componente, em seus métodos de ativação e desativação, carreguem e persistam dados relevantes referentes ao seu estado interno, podendo assim reconstruí-los no componente substituto sem ônus ao usuário da aplicação.

O *middleware* Kaluana provê um mecanismo para a transferência do valor associado a alguns tipos básicos da linguagem Java entre diferentes componentes. Para transferir tais dados, o desenvolvedor do componente deve definir o protocolo utilizado para representação de seus objetos e fica responsável pelos processos de *marshalling* e *unmarshalling* destes objetos, baseado nos tipos básicos da linguagem. O desenvolvedor deve definir quais

objetos serão armazenados e a ordem em que a transmissão destes objetos entre os componentes ocorrerá.

4.6 Mecanismo de implantação dinâmica

A implantação dinâmica consiste no *download* de um pacote instalável da plataforma Android que contém um ou mais componentes de interesse e sua instalação no dispositivo em que o *middleware* Kaluana é executado. O processo ocorre quando nenhum componente presente no repositório de componentes local pode atender a uma requisição de ativação.

A implantação foi criada como um processo síncrono, como explicado em mais detalhes na seção 5.3.2. Ou seja, somente um pacote de componentes é implantado por vez no dispositivo. O processo de ativação, entretanto, é realizado de forma assíncrona, e seu resultado é informado ao requisitante por meio de *callbacks*. Ou seja, a aplicação sempre solicita a ativação de um ou mais componentes, que podem estar instalados no dispositivo ou não. Quando todos os componentes requisitados forem ativados, a aplicação será notificada. O mesmo procedimento se aplica em caso da ativação a partir de uma adaptação dinâmica.

O *middleware* Kaluana não define sequência de implantação dos componentes. O requisitante será notificado sobre o sucesso de uma requisição quando todos os componentes de tal requisição estiverem ativos, independente da ordem em que foram requisitados e ativados.

O modelo Kaluana assume que existe um componente mais adequado para a execução em cada condição de contexto. Assim, mesmo que mais de um componente possa atender de forma satisfatória as condições atuais, um deles é mais indicado, baseado nos critérios definidos pelo desenvolvedor do próprio componente. Se houver dois componentes que antedam exatamente às mesmas condições de contexto, o *middleware* escolhe indiferentemente um deles.

Se não houver, no repositório remoto, componentes aptos a anteder às condições requisitadas, uma condição de erro é retornada à aplicação, que poderá então decidir por utilizar um componente mesmo que inadequado, ou simplesmente não realizar determinada composição.