5
Implementação

O *middleware* Kaluana é composto pelos elementos descritos a seguir. O Gerenciador de Componentes gerencia o ciclo de vida de todos os componentes disponíveis em cada dispositivo. O Repositório de Componentes realiza a busca e mantém referência para componentes disponíveis no dispositivo, porém inativos. O Gerenciador de Adaptações é responsável por, dada qualquer notificação de mudança no contexto, verificar se adaptações dinâmicas devem ser realizadas e, caso positivo, iniciar o processo junto ao Gerenciador de Componentes. O Serviço de Configuração é a faceta pela qual todos os componentes se comunicam, para que seja possível a comunicação entre componentes executados em diferentes processos Linux.

Toda a comunicação entre diferentes componentes é realizada por meio da invocação de serviços. Ou seja, mesmo que diferentes componentes sejam executados em diferentes processos, estes podem se comunicar através de chamada de procedimentos remotos. Se os componentes, por outro lado, forem executados em um mesmo processo, também estão sujeitos a erros de comunicação remota.

Este capítulo descreve o mecanismo de chamada remota de procedimentos no Android, amplamente utilizado pelo *middleware*, e como o *middleware* utiliza este e outros mecanismos da plataforma para a representação do modelo de componentes orientado a serviços e realização de adaptação e implantação dinâmicas. O *middleware* se utiliza dos mecanismos de publicação e descoberta de serviços, da chamada remota de procedimentos e de mecanismos de ciência de contexto da plataforma.

Após a descrição destes processos, são apresentados os testes realizados e resultados obtidos.

5.1
Elementos do middleware

O ciclo de vida dos componentes, de sua instalação à sua desativação, sua interação com outros componentes, sua adaptação e sua implantação dinâmicas são controlados pelos principais elementos do *middleware* Kaluana:

o Gerenciador de Componentes, o Repositório de Componentes, o Serviço de Configuração e o Gerenciador de Adaptações. Estes elementos e seus relacionamentos são apresentados na figura 5.1 e nas seções subsequentes.

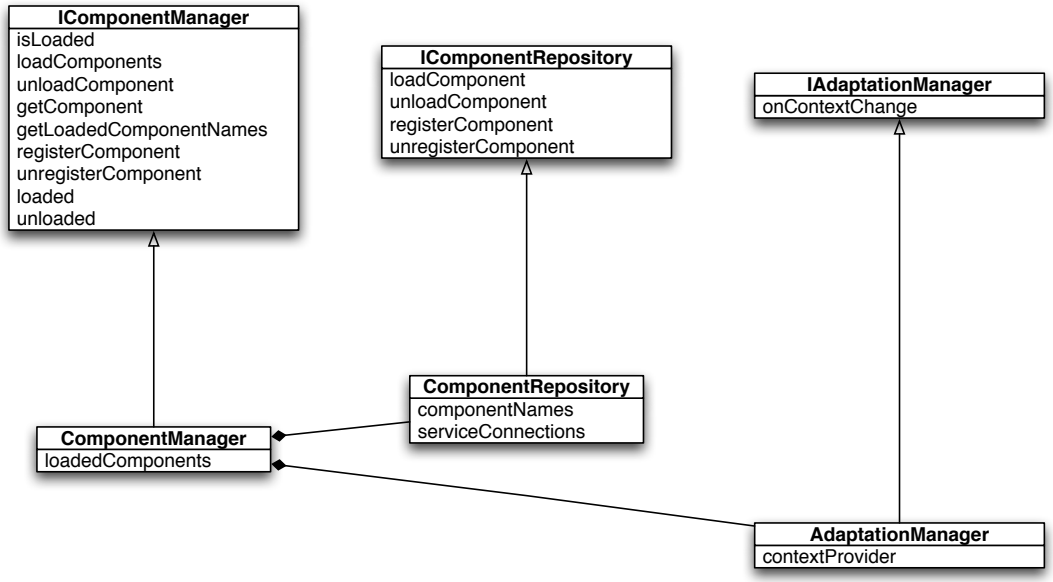

Figura 5.1: Descrição dos principais elementos do middleware

O gerenciador de componentes referencia o gerenciador de adaptações e o repositório local de componentes, e se comunicam via chamada remota de procedimentos.

O mecanismo de chamada remota de procedimentos e a comunicação assíncrona da plataforma Android são fundamentais para permitir tal arquitetura, pois todos os elementos do *middleware* bem como os componentes implementados sobre o mesmo se comunicam através de serviços, devido à necessidade de descoberta de novos componentes e serviços em tempo de execução.

O gerenciador de componentes e o gerenciador de adaptações são serviços Android, que precisam ser invocados pelo desenvolvedor de aplicações Kaluana. Os serviços de configuração dos componentes também são serviços Android, porém encapsulados pelo *middleware* Kaluana por meio de reflexão computacional, de modo que o desenvolvedor não tenha que invocá-los explicitamente.

5.1.1 Gerenciador de Componentes

O Gerenciador de Componentes é responsável por manter referências a todos os componentes ativos, bem como por atender requisições de ativação e desativação de componentes. Para isso, o Gerenciador de Componentes

funciona como um serviço oferecido pelo *middleware* Kaluana, que pode ser utilizado por qualquer aplicação.

Aplicações normalmente tentam ativar componentes no início de sua execução, de modo a formar as composições que atendam a seus requisitos. Durante a execução da aplicação, processos de adaptação dinâmica podem demandar ativações e desativações de componentes.

A cada requisição de ativação de um componente, o Gerenciador de Componentes verifica se existe um componente instalado no dispositivo com categoria ou nome semelhantes ao requisitado. Caso não haja tal componente, é iniciado um processo de implantação dinâmica, em que o *download* do componente é realizado junto a um repositório remoto de componentes, como apresentado na seção 5.1.2.

A partir deste ponto, o ciclo de vida do novo componente é iniciado, como discutido na seção 4.4.1.

Para ter acesso a todos os componentes instalados no dispositivo, o Gerenciador de Componentes mantém referência ao Repositório de Componentes.

5.1.2 Repositório de Componentes

O Repositório de Componentes conhece, por meio do registro de serviços da plataforma Android, os componentes instalados no dispositivos porém inativos. Para que seu componente possa ser descoberto pelo repositório, o desenvolvedor deve listar seu Serviço de Configuração, apresentado a seguir, no arquivo descritor do projeto. Assim, o *middleware* Kaluna é capaz de descobrir sua existência em tempo de execução.

O Repositório de Componentes é instanciado pelo Gerenciador de Componentes no início de sua execução, e é capaz de conhecer todos os componentes do dispositivo.

No caso de implantação dinâmica, o Repositório de Componentes utiliza o endereço *web* previamente cadastrado de um repositório remoto, a partir de onde pode ser feito o *download* de outros componentes, caso não existam instalados localmente.

5.1.3 Gerenciador de Adaptações

O Gerenciador de Adaptações é responsável por registrar interesse junto a um ou mais provedores de informações de contexto e, a cada mudança significativa no contexto, verificar se existem componentes ou serviços mais adequados ao novo contexto.

O Gerenciador de Adaptações mantém referência para o Gerenciador de Componentes, assim é capaz de verificar os componentes ativos no dispositivo e quais seus interesses cadastrados pelo contexto de execução.

Se for o caso, o Gerenciador de Adaptações deve notificar o Gerenciador de Componentes sobre a necessidade de uma adaptação dinâmica, que por sua vez realizará os processos de desativação do componente substituído e a ativação ou, quando cabível, a implantação do componente susbstituto.

O Gerenciador de Adaptações pode ser estendido pelo desenvolvedor de aplicações, de modo a que ofereça suporte para tipos de adaptação específicos para o domínio de cada aplicação. Deste modo, o desenvolvedor da aplicação define as regras, ou gatilhos, das adaptações dinâmicas que a aplicação realiza.

5.1.4 Serviço de Configuração

Diferentes componentes podem ser executados em diferentes processos Linux de um mesmo dispositivo, dependendo da maneira como foram implantados. A comunicação entre componentes se dá por meio de chamadas a procedimento remoto, como descrito na seção 5.2. Assim, cada componente, além dos serviços e receptáculos criados pelo seu desenvolvedor, possui também um Serviço de Configuração, implementado pelo Kaluana. Este serviço permite que, remotamente, um componente seja acessado pelo Gerenciador de Componentes e pelos demais componentes.

O serviço possui métodos para descrever a estrutura do componente, como listar seus serviços e receptáculos, para iniciar e interromper explicitamente sua execução, analisar e modificar seu estado interno e para conectá-lo e desconectá-lo a outros componentes.

O Serviço de Configuração de cada componente é referenciado pelo Gerenciador de Componentes, como mostra a Figura 5.2. Esta arquitetura permite ao Gerenciador de Componentes o acesso a componentes executados em processos distintos.

O Serviço de Configuração é também responsável por associar cada um dos outros serviços de seu componente a uma implementação específica. Este processo é realizado no processo de ativação do componente, a partir dos metadados definidos pelo desenvolvedor do componente, ou em um processo de adaptação dinâmica.

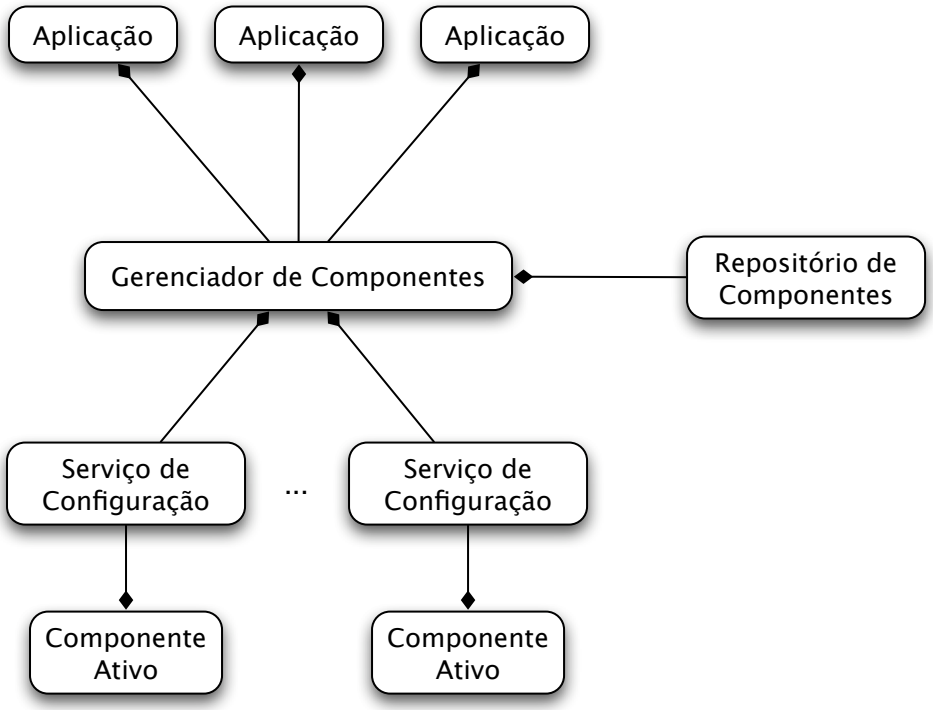

Figura 5.2: Gerenciamento de componentes

5.2 Chamada de procedimentos remotos (RPC)

O mecanismo de comunicação entre processos (IPC - *Inter Process Communication*) do Android permite à arquitetura oferecer um mecanismo de chamada de procedimentos remotos acessado por meio da linguagem Java, fundamental para a separação do processo de desenvolvimento de componentes ao processo de composição da aplicação, pois permite que componentes e aplicação se comuniquem mesmo que executem em diferentes processos.

Uma chamada remota de procedimento tem dois atores, um servidor e um cliente, executados no mesmo dispositivo.

A primeira etapa de um processo de RPC é a conversão do método chamado no *stub* cliente e seus parâmetros em uma mensagem do sistema operacional, processo este chamado de *marshalling*. Em seguida, esses dados são transmitidos pelo sistema operacional para o *stub* servidor, onde os dados são novamente transformados na representação específica da linguagem de programação, ou seja o *demarshalling*.

Os valores de retorno do método executado remotamente sofrerão o processo inverso para informar ao cliente o resultado da chamada. Todo o processo de chamada remota é de responsabilidade da plataforma Android, cabendo ao desenvolvedor a definição e implementação das interfaces envolvidas

no processo. Neste ponto, a plataforma de desenvolvimento (SDK - *Software Development Kit*) Android provê uma ferramenta que transforma os arquivos descritores de interface AIDL (*Android Interface Description Language*) em classes Java conhecidas pelos atores envolvidos na chamada.

A chamada remota de métodos é um processo síncrono, ou seja, o cliente fica bloqueado até que a chamada seja finalizada, mesmo que não haja valores de retorno, tornando o processo de chamada remota semelhante a chamadas locais em termos programáticos. Entretanto, é possível definir *callbacks* para notificação do término das chamadas, de modo a bloquear o cliente por menos tempo.

A figura 5.3 mostra o processo de chamada remota no Android. Ambos cliente e servidor contém classes que implementam a interface *android.os.IBinder*.

No processo servidor, a classe *Stub* recebe a implementação do serviço como parâmetro, definindo uma classe interna à classe Java criada pelo SDK para a interface descrita por AIDL. A classe criada pelo SDK estende a classe *android.os.Binder*, que encapsula o *stub* servidor, implementado pelo desenvolvedor do serviço.

No processo cliente, o *stub* cliente é referenciado através da chamada ao método *asInterface()* da classe *Stub*, gerada pela ferramenta AIDL. Este método recebe uma referência para o *Binder* daquele serviço, por sua vez gerado pela plataforma Android, que recebeu a implementação do serviço como parâmetro em sua instanciação.

Enquanto o processo servidor conhece a interface do serviço e sua implementação, o processo cliente só conhece a interface do serviço e recebe referência para a implementação em tempo de execução, ao requisitar pela conexão àquele serviço.

A implementação do mecanismo de adaptação dinâmica do *middleware* Kaluana estende este uso, criando diversas implementações para um mesmo *stub* servidor e separando-as em diversas classes, que podem ser substituídas de modo a prover adaptação funcional, ou seja a substituição de serviços. Retirando a implementação da classe interna ao *stub* e realizando diferentes implementações em diferentes classes, é possível substituir implementações para a mesma interface, de maneira transparente para o invocador do serviço.

Chamadas a serviços não diferenciam se o serviço é executado no mesmo processo de seu invocador ou em processo diferente. A chamada é sempre sujeita aos tipos de erros inerentes de uma comunicação remota.

Para trafegar objetos Java entre diferentes processos, a plataforma Android utiliza a interface *android.os.Parcelable* para descrever estes objetos. As-

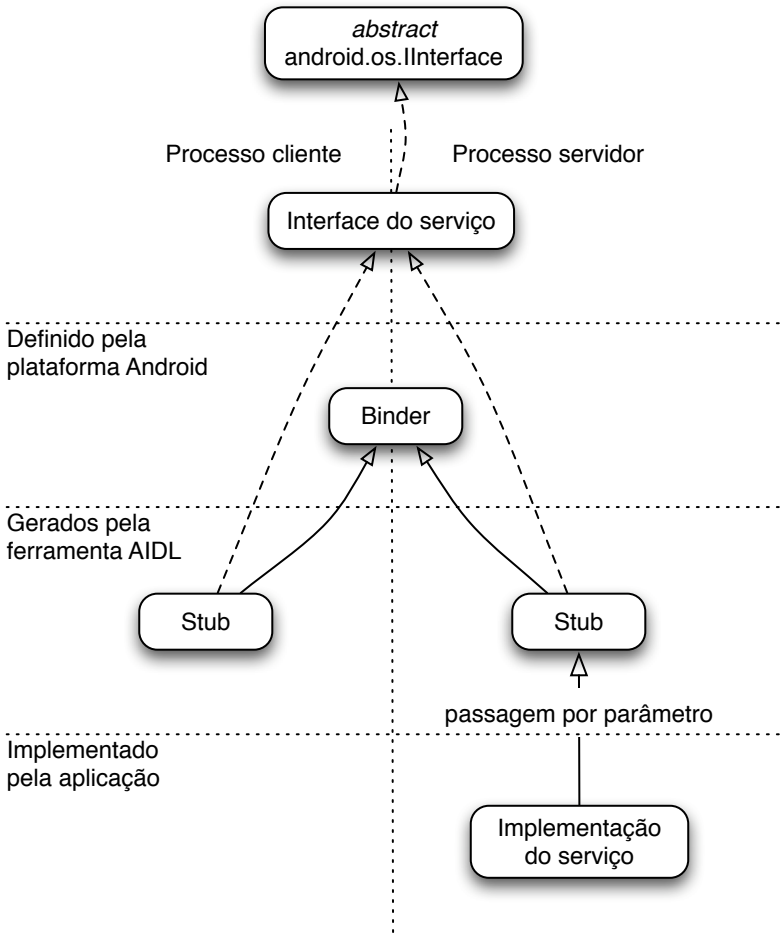

Figura 5.3: Chamada de procedimento remoto no Android

sim, o objeto pode ser representado por dados e meta-dados que caracterizam seu tipo e conteúdo, de modo a converter a representação em memória da mesma informação entre diferentes processos.

Este tipo de comunicação é utilizado no mecanismo de adaptação dinâmica do Kaluana para transmitir o estado interno de um componente para seu componente substituto.

5.3 Aspectos relevantes

Alguns aspectos da implementação são descritos nesta seção de modo a esclarecer como a arquitetura foi implementada especificamente sobre a plataforma Android.

5.3.1 Adaptação dinâmica

Para realizar a substituição de implementações de um serviço, ou adaptação dinâmica funcional, o *middleware* Kaluana estende o mecanismo de chamada remota de procedimentos da plataforma Android, de modo a permitir que diferentes classes implementem determinado serviço em diferentes momentos.

Para isso, o *middleware* Kaluana, que é responsável por instanciar todos os serviços de cada componente, o faz de forma a encapsular diferentes implementações em classes diferentes que implementam a interface do serviço. Numa aplicação típica sobre a plataforma Android, a definição de diferentes implementações para um serviço é feita em classes internas a uma mesma classe, e suas substituições são explicitamente realizadas pelo desenvolvedor.

Assim, o *middleware* Kaluana absorve a complexidade da implementação de uma troca explícita entre diferentes implementações, e o desenvolvedor não precisa se preocupar em como este processo ocorre.

Para a realização de uma adaptação funcional, é necessário armazenar todas as referências existentes para o serviço adaptado, de modo a reiniciá-lo sem que seus clientes sejam afetados. A Figura 5.4 demonstra a arquitetura que permite a adaptação dinâmica funcional, em que diferentes implementações podem ser alternadas pelo Gerenciador de Componentes, após ser notificado de tal necessidade pelo Gerenciador de Adaptações.

Kaluana se aproveita da capacidade de interrupção da execução explícita de serviços da plataforma Android para substituir a implementação de um serviço oferecido por um componente durante sua execução. Como todos os serviços de todos os componentes, bem como os receptáculos a que estão conectados são referenciados pelo *middleware* Kaluana, é possível a qualquer momento interrompê-los e iniciá-los com outra implementação.

A adaptação dinâmica estrutural, ou substituição de componentes, por sua vez é realizada por meio da substituição da implementação de um componente, todos os serviços e seu estado interno. O novo componente mantém a referência para todos os serviços que o componente anterior utilizava.

A Figura 5.5 mostra o processo de adaptação dinâmica estrutural, em que existe a substituição de um componente, seus serviços e seu estado interno. O Gerenciador de Componentes, que mantém referência aos serviços de configuração de todos os componentes, ao ser notificado da necessidade de realizar uma adaptação, realiza a troca de um componente por outro, no qual reconecta todos os serviços e receptáculos e remonta o estado interno. O processo demonstrado na figura assume que o Gerenciador de Adaptações

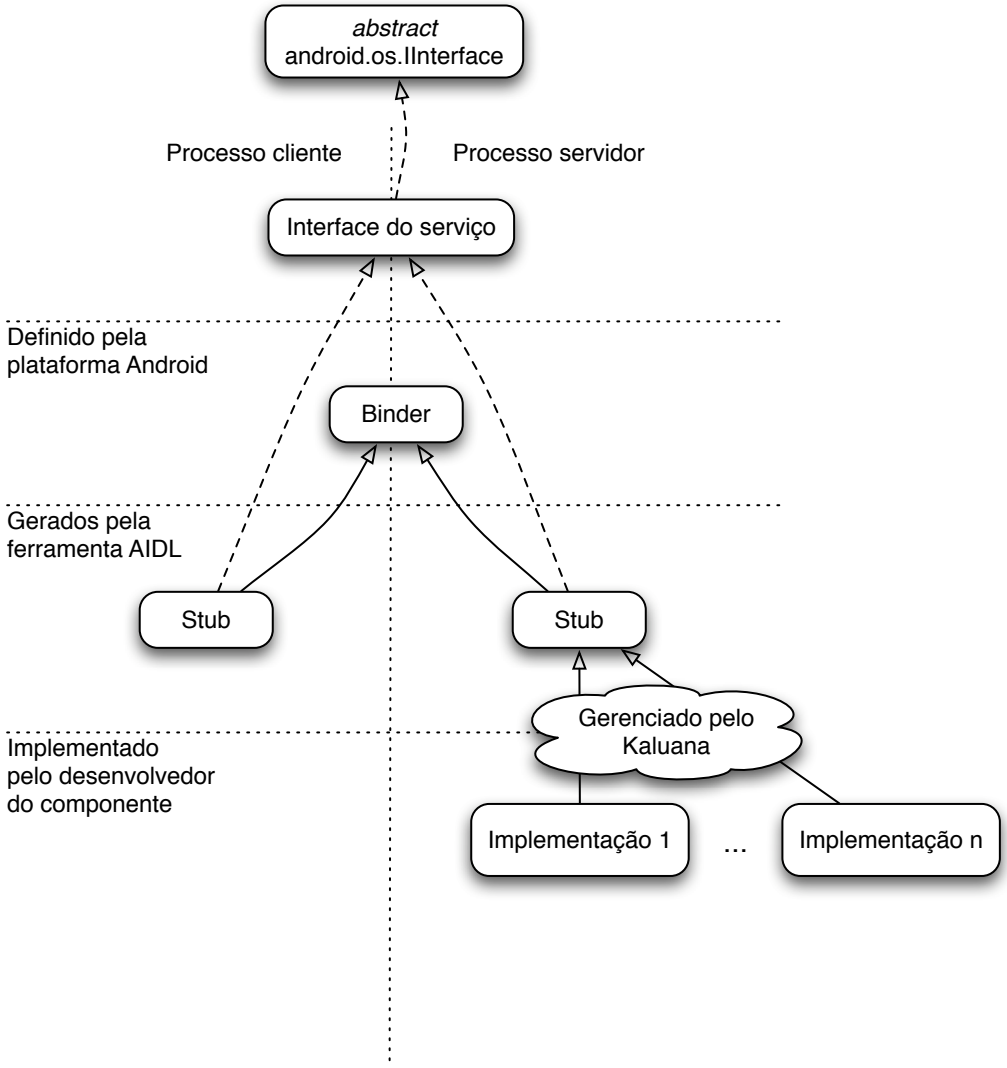

Figura 5.4: Adaptação dinâmica funcional

recebeu notificações de mudança de contexto e verificou, junto ao Gerenciador de Componentes, quais componentes se adequam ou não às novas condições.

O Repositório de Componentes utiliza o mecanismo de registro de serviços, baseado no arquivo descritor de projetos, *AndroidManifest.xml*, para realizar a descoberta de componentes e serviços adequados a cada requisição de ativação. Logo, desenvolvedores de componentes devem registrar nome e categoria de cada componente e cada serviço no arquivo de descrição de conteúdo do projeto.

Desta maneira, é possível verificar a existência de componentes e serviços disponíveis sem a necessidade de instanciar objetos ou abrir pacotes específicos para cada componente ou serviço.

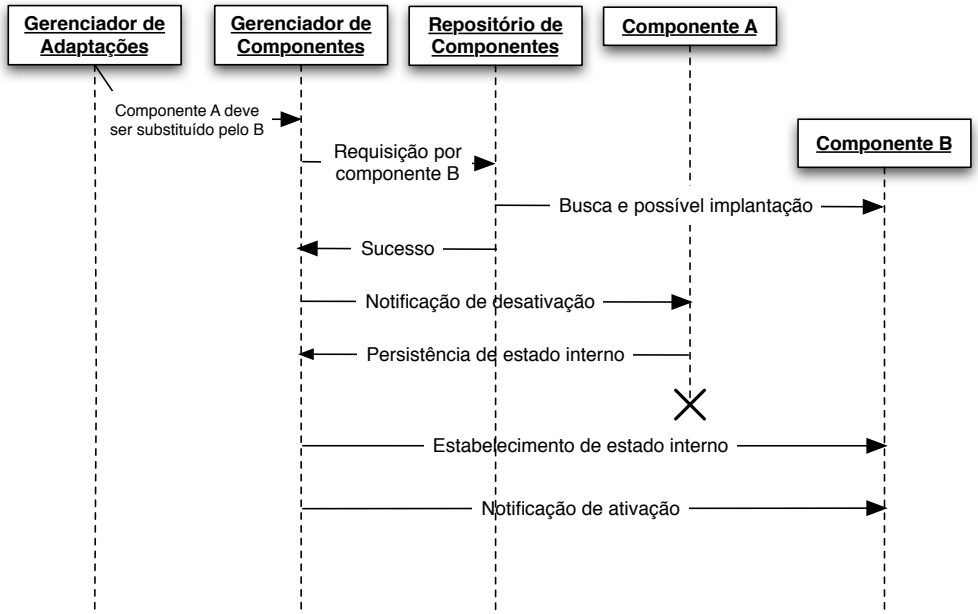

Figura 5.5: Processo de adaptação dinâmica estrutural

5.3.2 Implantação dinâmica

O processo de implantação dinâmica consiste no *download* a partir de um repositório remoto cuja URL é previamente conhecida pelo *middleware* e na instalação de um pacote APK no dispositivo móvel. Esta tarefa, entretanto, não está implementada programaticamente no SDK Android em que o *middleware* foi desenvolvido, conquanto haja interfaces para tal ainda a serem implementadas.

Para resolver este problema, a implantação dinâmica foi implementada especificamente para o ambiente do emulador Android, e esta implementação não funcionará em dispositivos reais.

Uma instalação de qualquer arquivo APK pode ser realizada através da linha de comando na máquina onde o emulador é executado. Ao solicitar a instalação de algum pacote no dispositivo, o *middleware* invoca, por meio de uma requisição HTTP, um procedimento na máquina onde o emulador é executado, que instala o pacote no emulador e retorna a mensagem de erro ou sucesso subsequente. Para eliminar problemas de concorrência, o processo de instalação é síncrono.

No futuro, pretende-se substituir este mecanismo por um processo de instalação a partir do *download* do arquivo APK, quando houver recursos programáticos que permitam sua instalação na plataforma Android.

5.3.3 Interface visual

A interface visual das aplicações executadas sobre o *middleware* Kaluana é de responsabilidade dos componentes utilizados. Cada componente pode iniciar uma interação com o usuário, que pode ser finalizada pelo próprio componente ou ao término do seu ciclo de vida, durante sua desativação.

Por exemplo, um componente que transmite notícias por *streaming* de vídeo apresentará ao usuário uma tela com o conteúdo e com interfaces de controle do vídeo exibido. Quando, por motivos de queda de qualidade da conexão do dispositivo, este componente é substituído por um componente de texto, que somente apresenta os resumos das notícias, torna-se necessária a mudança também da interface visual com o usuário.

5.3.4 Reflexão computacional

Para tornar o desenvolvimento de componentes tarefa mais simples, o *middleware* realiza as tarefas de publicação, descoberta e utilização de serviços e resolução de dependências entre componentes, além de prover abstrações como a representação de componentes por meio de classes Java.

Para tal, o *middleware* permite que componentes sejam definidos de maneira próxima a classes Java, sem esforço adicional ao desenvolvedor exceto a necessidade de definir meta-dados que indiquem ao *middleware* o nome e as depedências do componente, seus serviços e receptáculos.

A partir daí, a cada requisição de aplicação pela ativação de um componente o *middleware* inspeciona, por meio de reflexão computacional, a classe que representa o componente para vincular, por introspecção, a declaração de seus serviços às implementações adequadas, definir sua lista de receptáculos disponíveis para conexão a outros componentes e carregar suas dependências, caso existam.

Assim a complexidade adicionada ao modelo de componentes em relação a uma abordagem orientada a objetos é reduzida e o desenvolvimento de componentes mais simples e intuitivo.

Da mesma maneira, a conexão entre receptáculos e serviços de diferentes componentes se dá por meio de introspecção, em que os métodos que vinculam o receptáculo à implementação de determinado serviço são chamados em classes somente conhecidas em tempo de execução.

5.4 Testes

Esta seção define os objetivos, metodologia e resultados obtidos nos testes realizados sobre o *middleware* Kaluana e sobre as aplicações e componentes implementados, de modo a demonstrar seu correto funcionamento e sua utilidade.

5.4.1 Objetivos

Os testes propostos visam a demonstrar a correção das funcionalidades propostas neste trabalho, desde validar o modelo de componentes a verificar o comportamento do mecanismo de adaptação e implantação dinâmica.

5.4.2 Metodologia

Para validar as funcionalidades propostas pelo *middleware* Kaluana, foram criados alguns testes de integração de funcionalidades correlacionadas, divididos nas seguintes categorias: ciclo de vida dos componentes, ciência de contexto, adaptação dinâmica e implantação dinâmica.

Para verificar o correto funcionamento e a aplicabilidade do modelo proposto pelo *middleware* Kaluana, foram desenvolvidas duas aplicações dinamicamente adaptáveis baseadas em componentes. O processo de desenvolvimento destas aplicações e sua correta execução são apresentados nas seções a seguir.

Além destes, testes unitários foram realizados, por meio da ferramenta JUnit integrada à plataforma Android, para auxiliar o processo de desenvolvimento do *middleware*. A criação destes testes promoveu agilidade na detecção e correção de *bugs* na implementação do *middleware* e a implementação de aplicações permitiu a real sensação do processo de desenvolvimento, resultando em ideias para simplificá-lo, como a utilização de anotações Java para declaração de meta-dados dos componentes.

Cada um dos elementos do *middleware* é associado a um teste unitário específico, são eles: Gerenciador de Componentes, Gerenciador de Adaptações, Repositório de Componentes e Serviço de Configuração. Além destes, testes de integração verificam a correção no gerenciamento do ciclo de vida dos componentes e nos processos de adaptação e implantação dinâmicas.

A metodologia dos testes de integração e do desenvolvimento das aplicações de exemplo é mostrada nas seções subsequentes.

Ativação e composição

Para os testes de ativação e conexão de componentes, foram desenvolvidos dois componentes: *PingComponent* e *PongComponent*.

O componente *PingComponent* oferece um serviço chamado *ping* e um receptáculo chamado *pong*. O componente *PongComponent*, por sua vez, oferece um serviço *pong* e uma interface *ping*.

O teste realiza a ativação de cada um dos componentes e conecta seus serviços aos receptáculos homônimos do outro componente. Comparando o resultado da chamada de um componente ao serviço oferecido pelo outro ao resultado esperado, uma vez que conhece-se as duas implementações, verifica-se a correção no processo de ativação.

Algumas situações são possíveis e foram testadas no emulador Android.

- Ambos os componentes estão presentes no dispositivo: Os dois componentes são ativados e o requisitante é notificado do sucesso na requisição.
- Pelo menos um dos componentes não está presente no dispositivo, mas existe em um repositório remoto: Os dois componentes são ativados e o requisitante é notificado do sucesso na requisição.
- Pelo menos um dos componentes não está presente no dispositivo nem em um repositório remoto: Neste caso, o processo de ativação não é realizado para nenhum dos componentes e uma mensagem de erro é retornada ao requisitante.

Para realizar testes de dependência entre componentes, foi adicionada uma dependência no componente *PingComponent* de um terceiro componente, chamado *DependencyTestComponent*. Neste caso, existem as seguintes possibilidades:

- O componente *DependencyTestComponent* está presente no dispositivo: Os três componentes são ativados e o requisitante é notificado do sucesso na requisição.
- O componente *DependencyTestComponent* não está presente no dispositivo, mas existe em um repositório remoto: Os três componentes são ativados e o requisitante é notificado do sucesso na requisição.
- O componente *DependencyTestComponent* não está presente no dispositivo nem em um repositório remoto: O processo de ativação não é realizado para nenhum dos componentes e uma mensagem de erro é retornada ao requisitante.

Existe ainda a possibilidade de dependência cíclica, caso em que a representação das dependências entre um conjunto de componentes por meio de um grafo apresenta um ciclo. Esta situação foi testada para dois e três componentes e não resultou em erros no processo de ativação.

Ciência de contexto

Os mecanismos de ciência de contexto que o Kaluana utiliza são providos pela plataforma Android e acessados por linguagem Java. Os testes foram realizados com o auxílio da ferramenta ADB (*Android Debug Bridge*, que em conjunto com o emulador Android, provê ferramentas que simulam algumas condições de contexto, como a localização geográfica do dispositivo, e enviam essas informações às aplicações.

O conjunto de testes do *middleware* Kaluana abrange testes do serviço de localização baseado em GPS da plataforma Android. Os testes verificam o comportamento do sistema ao receber informações de coordenadas geográficas hipotéticas da localização do dispositivo.

Nos testes é possível verificar a reação do *middleware* a diferentes frequências de envio de informações. Esta frequência pode ser controlada em função de parâmetros de distância percorrida e tempo decorrido. O *middleware* Kaluana apresentou comportamento satisfatório com o valor de intervalo de tempo recomendado pela documentação da plataforma [1] de 60 segundos, que representa economia de energia. O intervalo mínimo de distância foi escolhido como 100 metros para não haver muitas requisições seguidas ao *middleware* quando o dispositivo está em movimento.

As notificações sobre mudanças no contexto são utilizadas pelo Gerenciador de Adaptações para verificar a necessidade de efetuar adaptações funcionais ou estruturais em aplicações.

Adaptação dinâmica

Dois casos podem resultar em uma adaptação dinâmica estrutural: pode existir componente mais adequado que possa substituir um componente ainda funcional, ou um componente pode deixar de funcionar e precisar ser substituído. No primeiro caso, ocorre uma adaptação de otimização, e no segundo uma adaptação corretiva.

Em primeiro lugar, é testado o algoritmo de escolha dos componentes participantes de um processo de adaptação, detalhado na seção 5.3.1.

O cenário é composto por dois componentes, um ativo e outro instalado no dispositivo, porém desativado. Dada uma mudança no contexto, as seguintes situações podem ocorrer:

- O componente ativo continua funcional e o componente inativo continua inadequado: Nenhuma adaptação é realizada.
- O componente ativo continua funcional, porém o componente inativo se torna mais adequado: O componente ativo é substituído.
- Ambos os componentes, ativo e inativo, são funcionais: Nenhuma alteração é realizada.
- O componente ativo deixa de funcionar e o componente inativo se torna adequado: O componente ativo é substituído.
- O componente ativo deixa de funcionar e não existe componente mais adequado: Nenhuma adaptação é realizada.

Para realizar os testes, as condições de funcionamento foram baseadas no exemplo descrito no estudo de caso, mostrado no capítulo 6.

Em seguida, foi verificado se o componente instanciado após o processo mantém as mesmas conexões e o mesmo estado interno do componente substituído, mostrando assim a correção do processo. A reconexão dos serviços e receptáculos foi testada verificando-se que as chamadas aos mesmos métodos de um serviço retornaram determinado valor esperado antes e depois da substituição do componente que o provê.

Para tal, o mesmo exemplo descrito no teste de ativação e composição foi modificado, de modo que um dos componentes pudesse ser substituído. Após a troca do componente *PingComponent* pelo componente substituto *PingComponent2*, as conexões do novo componente com o componente *PongComponent* continuaram sendo utilizadas.

Para testar a persistência do estado interno, foi implementado nos componentes *PingComponent* e *PingComponent2* os métodos *setInternalState()* e *getInternalState()*, respectivamente, em que os dados do componente são armazenados e recuperados. Após uma adaptação dinâmica, verificaram-se os mesmos valores nos atributos internos da classe que representa o componente substituto.

O mecanismo de adaptação dinâmica funcional foi testado por meio da realização de duas implementações distintas para o serviço oferecido pelo *PingComponent*. O *middleware* Kaluana, da mesma maneira que na adaptação estrutural, é capaz de substituir implementações de um serviço ao detectar a invalidade ou inadequação de um serviço oferecido por um componente.

Implantação Dinâmica

Este teste abrange o correto *download* e instalação de um novo componente a partir do repositório remoto, ou seja, que não está instalado no dispositivo. Também funciona como teste para o repositório, pois verifica seu correto funcionamento em caso de não existir o componente procurado.

Para a realização dos testes, foi montado um cenário em que o componente *PingComponent* existe e está ativo no dispositivo, enquanto o componente *PongComponent* não existe no dispositivo, mas está disponível em um repositório remoto. O repositório é executado na mesma máquina que o emulador Android, e por meio de linha de comando executa a instalação do pacote previamente cadastrado que contém o componente *PongComponent*.

O processo de instalação do segundo componente é transparente para a aplicação. Ao requisitar a instalação de ambos os componentes, a implantação foi realizada e a aplicação foi notificada sobre a ativação de ambos.

Em um segundo cenário, o componente não existia no dispositivo e no repositório. Desta maneira, ao requisitar a ativação de ambos os componentes, a aplicação foi notificada da inexistência do componente *PongComponent*.

O teste não abrange problemas de conectividade do dispositivo.

5.4.3 Aplicações

Para fins de testes, foram construídas duas aplicações.

A primeira aplicação, chamada *Ping Pong*, utiliza os componentes criados e consiste da interconexão entre ambos, de modo que utilizem serviços um do outro.

A segunda aplicação, sensível à localização, alterna componentes de navegação sobre mapas ao entrar e sair da universidade PUC-Rio. A aplicação, chamada *Navigator*, consiste na conexão de três componentes: dois de mapas e um de provisão de contexto, o último também utilizado pelo *middleware*. Os dois componentes de mapas são adaptáveis e pertencem à mesma categoria, deste modo sendo substituíveis.

As aplicações foram testadas no emulador Android, mostrado na Figura 5.6. A primeira, mais simples, demonstra a utilização do modelo de componentes orientado a serviços definido pelo *middleware* Kaluana. A segunda aplicação demonstra, além da utilização do modelo, os processos de adaptação e implantação dinâmicas.

Uma vez comprovada a correta execução de ambas, fica mostrado que o *middleware* permite a composição de aplicações a partir de componentes reutilizáveis.

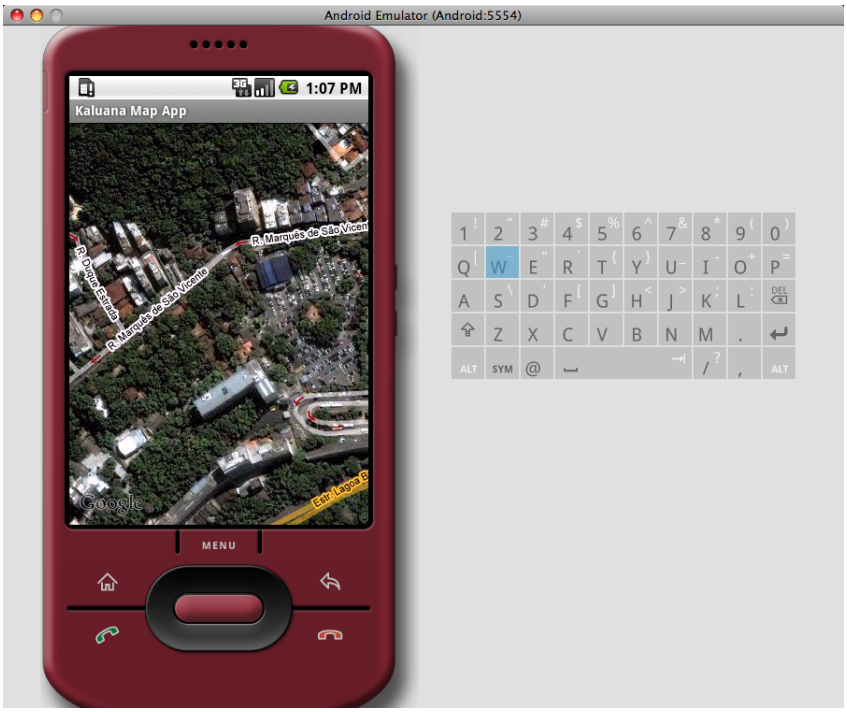

Figura 5.6: Emulador Android

A implementação destas aplicações também visa a demonstrar a simplicidade no processo de desenvolvimento, como será mostrado na discussão dos resultados.

5.4.4 Resultados qualitativos

Os testes realizados abrangeram todos os cenários descritos nas seções anteriores. Como as aplicações desenvolvidas foram executadas no emulador e seus resultados esperados foram alcançados, mostrou-se a validade e correta implementação do modelo de componentes orientado a serviços definido pelo *middleware* e do modelo de desenvolvimento proposto, dividido em duas fases.

Os mecanismos de adaptação dinâmica estrutural e funcional foram testados em diferentes situações de variação de contexto, sob condições que os componentes e serviços envolvidos no processo deveriam ou não ser substituídos, mostrando seu correto funcionamento.

O mecanismo de implantação dinâmica também mostrou-se funcional, por ser capaz de realizar o *download* e instalação de componentes de forma transparente para a aplicação.

5.4.5 Discussão

O modelo definido pelo Kaluana permite o reuso de componentes existentes no dispositivo sem a necessidade de conhecimento de suas implementações, somente de suas interfaces. Diferentemente de uma aplicação comum sobre a plataforma Android, uma aplicação sobre o *middleware* Kaluana não precisa realizar a importação de nenhum pacote Java com a implementação dos componentes de *software* que utiliza. Uma aplicação Kaluana precisa somente importar as interfaces dos serviços oferecidos pelos componentes em questão.

Pelos motivos mostrados a seguir, o desenvolvimento de aplicações e de componentes sobre o *middleware* Kaluana mostra-se mais simples em comparação com aplicações semelhantes sobre a plataforma Android que não utilizam o *middleware*, pois pouca complexidade é adicionada para adequar uma aplicação à arquitetura Kaluana e os mecanismos de adaptação e implantação dinâmica são transparentes para o desenvolvedor.

Desenvolvimento de aplicações

Isolando a descrição e a implementação de componentes e serviços, Kaluana torna o processo de reuso mais simples e robusto do que o reuso de código-fonte ou mesmo de pacotes Java, pois componentes Kaluana escondem sua implementação interna. Diferente do que acontece em pacotes Java, Kaluana só deixa disponíveis para uso as interfaces dos componentes, assim evitando a utilização indevida de dependências em função de detalhes de sua implementação.

As abstrações providas para a conexão e desconexão de diferentes componentes, que podem ser realizadas programaticamente, através de chamada de métodos das classes que representam os componentes, promovem mais clareza no entendimento da arquitetura da aplicação desenvolvida.

Desenvolvimento de componentes

A utilização de classe Java para representação de um componente e anotações Java para definir seus receptáculos e serviços simplificam o processo de desenvolvimento de componentes, por integrar todas as tarefas necessárias para sua criação no mesmo arquivo Java.

A utilização de introspecção via reflexão computacional para vincular implementações a serviços e receptáculos oferecidos pelos componentes também tem papel importante na simplificação do desenvolvimento, uma vez que não é necessário nenhum esforço de desenvolvimento, exceto anotações Java,

para vincular implementações dos serviços a componentes, nem para conectar serviços a receptáculos.

A definição de dependências por meio de anotações Java simplifica o entendimento do relacionamento entre componentes, por concentrar as dependências específicas de cada componente na classe que o representa, por meio de anotações Java.

Mecanismos de adaptação e implantação dinâmicas

A implementação dos processos de adaptação e implantação dinâmica é totalmente transparente para os desenvolvedores da aplicação e do componente. Para criar um componente adaptável, seu desenvolvedor precisa somente definir o contexto em que o componente ou cada serviço é corretamente executado, por meio do arquivo descritores de projeto, *AndroidManifest.xml*. O desenvolvedor da aplicação, por sua vez, não precisa se preocupar com nenhum detalhe.

O processo de implantação dinâmica, por sua vez, não depende de nenhum esforço do desenvolvedor da aplicação. O desenvolvedor do componente deve, entretanto, registrá-lo em um repositório a partir do qual possa ser encontrado.

A complexidade dos processos é inteiramente tratada pelo *middleware*, desde a detecção das condições de contexto, sua interpretação, a verificação da necessidade de adaptações baseada nestas condições e a substituição de componentes e serviços.