6
Resultados experimentais

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.

O objetivo dos resultados experimentais apresentados neste capítulo é demonstrar a simplicidade da criação de aplicações e componentes, bem como a capacidade do *middleware* de tomar decisões a respeito do melhor componente a ser oferecido para uma aplicação, de acordo com o contexto computacional do dispositivo no momento da adaptação.

Para tal, foi desenvolvida uma aplicação Android baseada no Kaluana, que faz uso de mapas e se adapta dinamicamente de acordo com a localização do usuário, para que o usuário possa ter uma informação mais específica e relacionada com seu contexto.

Foram criados também testes de desempenho para avaliar o *overhead* gerado pelas novas implementações, de acordo com as buscas de componentes e avaliação de contratos.

É demonstrado também o *middleware* Kamaiurá, criado sobre o *middleware* Kaluana para prover a reconfiguração de componentes não só nos dispositivos como também em servidores remotos que provêm a execução de componentes necessários para a aplicação. Estes servidores remotos são designados para executar parte do processamento dos componentes, desonerando o dispositivo do processamento de partes necessárias para a execução da aplicação, que tenham uma execução intensiva.

6.1.
Testes de desempenho

Esta seção, descreve alguns dos testes realizados para avaliar a solução implementada para este trabalho. Para isto, foram criados dois testes que medem o

overhead gerado pelas novas implementações feitas no novo Kaluana. Em todos os resultados apresentados, as aplicações cliente e servidor foram inicializadas em um emulador Android versão 2.1 executando em um Windows XP (Intel Core 2 Quad 2.5GHz e com 4Gb de RAM) operando em uma rede local cabeada a 100Mbps.

6.1.1. Busca de componentes

Este teste demonstra um comparativo entre o código do Kaluana original e o Novo Kaluana. Para isto, foi tomado o tempo de execução de uma a 100 requisições de componentes sequencialmente simulando a requisição de um componente por *n* aplicações. Este teste avalia o tempo de resposta do *middleware* para estas *n* requisições considerando a diferença de avaliação entre as duas versões do Kaluana. Esta avaliação calcula o tempo de resposta de uma requisição considerando não só o tempo de resposta como também a capacidade do *middleware* de consumir itens da fila de requisições no caso de um número grande de pedidos.

O Kaluana original faz a requisição do componente para o Android e retorna o primeiro componente encontrado mesmo que haja outros disponíveis. O Novo Kaluana faz uma busca dos *k* componentes que atendem a requisição, avalia o contrato de reconfiguração destes componentes lendo suas restrições computacionais para depois escolher o componente que melhor atende a requisitando, retornando este para a aplicação requisitora.

A Figura 23 mostra os resultados para a requisição de componentes ao Kaluana sequencialmente. Nos gráficos são feitos comparativos entre os tempos de requisição de componentes para ambas as versões do código do *middleware*. No gráfico, foram esboçados os tempos médios de resposta do *middleware* com intervalo de confiança assintótico simétrico de 95%. Para isto, cada teste foi executado 50 vezes usando os tempos médios calculados a partir dos dados destas execuções.

Nota-se que existe um *overhead* maior de processamento para a requisição de componentes no Novo Kaluana em relação ao código do Kaluana Original.

Esta diferença se deve ao algoritmo de busca e avaliação de componentes candidatos (que atendem aos requisitos da aplicação).

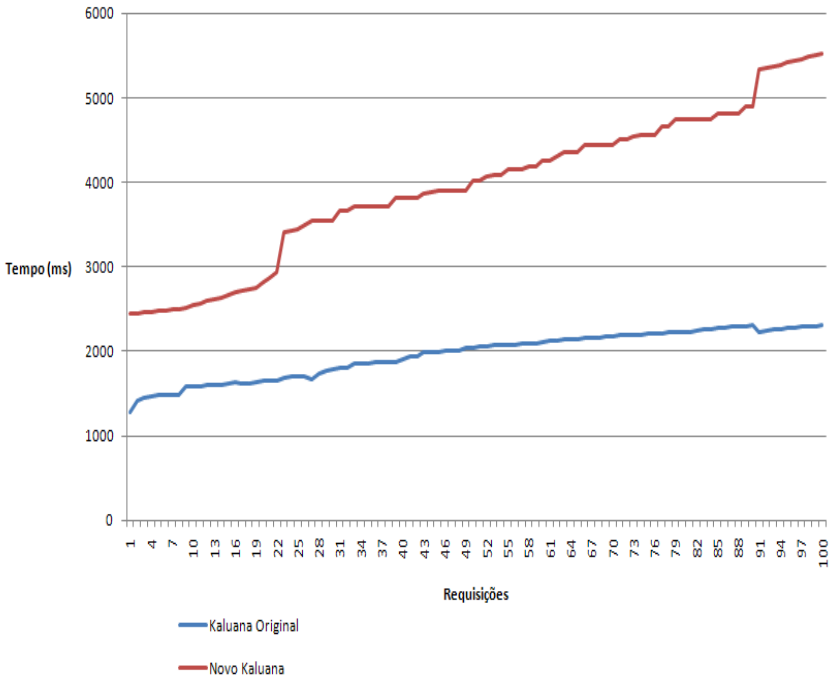

Figura 23 Requisição de componentes sequencialmente

6.1.2. Requisições múltiplas

O segundo teste realizado avalia a performance do Novo Kaluana ao receber requisições múltiplas simultâneas. Para este teste foram tomados os tempos de resposta de 1 a 100 requisições simultâneas de componentes de modo a tentar estressar o *middleware* de acordo com sua escalabilidade. Foram criadas *n threads*, disparadas simultaneamente, realizando a requisição de um componente ao *midddleware*, simulando a requisição de *n* aplicações simultâneas.

A Figura 24 mostra os resultados obtidos nos testes. Pode-se notar nos gráficos que o tempo de requisição aumenta quase linearmente de acordo com a quantidade de requisições paralelas, o que mostra uma pequena perda de desempenho com o aumento de requisições. No gráfico, foram esboçados o tempo médio de resposta do *middleware* com intervalo de confiança assintótico simétrico

de 95%. Para isto, cada teste foi executado 50 vezes usando os tempos médios calculados a partir dos dados destas execuções.

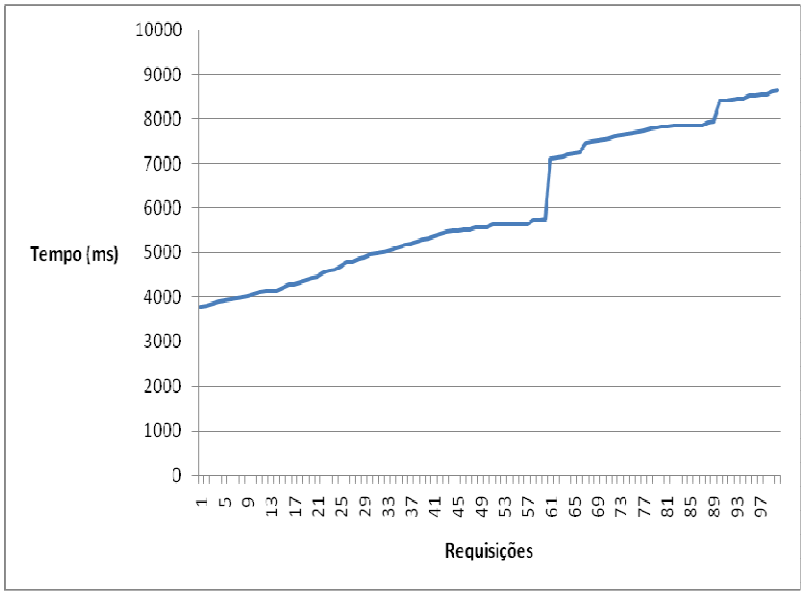

Figura 24 Requisição paralela de componentes

6.2. Aplicação exemplo

A aplicação criada para exemplificar os requisitos implementados no Kaluana durante este trabalho é baseada em componentes de mapa intercambiáveis. A implementação desta aplicação visa demonstrar a facilidade de desenvolvedores de aplicações e de componentes em implementar aplicações dinâmicas usando o Kaluana como base.

A aplicação exibe a localização do usuário a todo o momento em um mapa, usando a interface de GPS do dispositivo. O *google maps* é usado como mapa base, o qual na aplicação é representado pelo componente *GoogleMapComponent*. Este componente mostra o mapa básico do google, porém com um marcador que representa a posição atualizada do usuário (vide Figura 25).

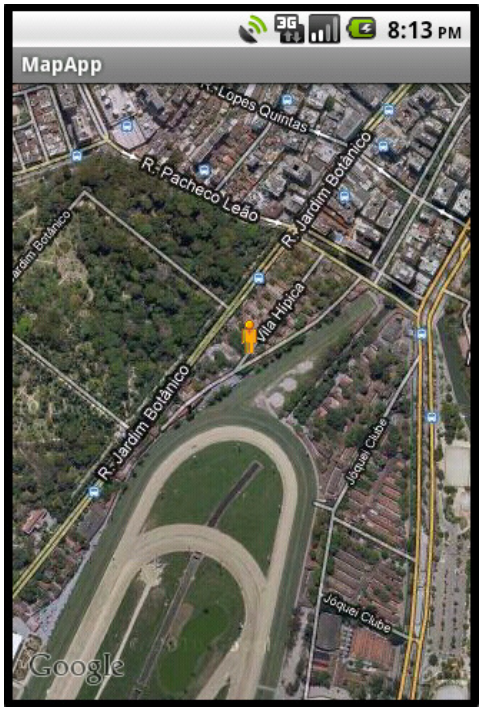

Figura 25. Componente default, *GoogleMapComponent*

A aplicação possui também áreas previamente demarcadas, que representam lugares que possuem componentes específicos que exibem mapas detalhados do local. Para este trabalho foi demarcada apenas a área da PUC-Rio, pois foi a única área em que um mapa detalhado foi conseguido.

Toda vez que a aplicação nota que o usuário entrou em uma área demarcada, ela faz a requisição do componente de mapa específico daquela área ao Kaluana. O desenvolvedor da aplicação deve conhecer o nome do componente ou o nome do serviço de mapa daquela área, para realizar a requisição.

Sempre que o usuário passa por uma área demarcada, a aplicação pergunta se ele deseja exibir o mapa detalhado daquele local. Para evitar inconveniências, caso o usuário recuse um mapa por duas vezes seguidas, a aplicação marca aquele componente como descartável e não realiza mais a requisição naquela área. A Figura 26 mostra um exemplo da requisição da permissão de um usuário para exibir um mapa detalhado quando uma área demarcada é cruzada.

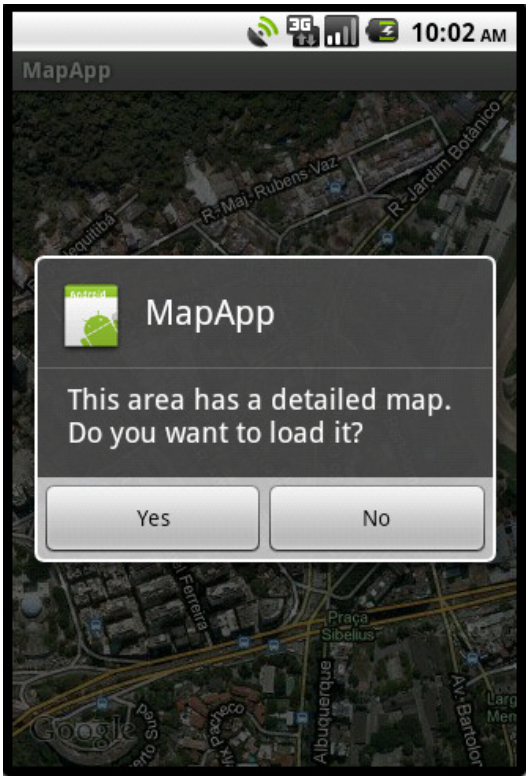

Figura 26. Requisição de carregamento de um novo componente

6.3. Casos de uso

Com o objetivo de realizar testes que abrangessem todas as extensões criadas para o *middleware* Kaluana, foram criados dois casos de uso para a aplicação exemplo, descrita na seção 6.2. Estes casos de uso simulam um usuário comum ao longo de sua participação em um congresso na PUC-Rio.

6.3.1. Ator percorre uma área não demarcada

6.3.1.1. Descrição

Este caso de uso permite que o ator percorra uma área da cidade que não seja demarcada pela aplicação como contendo um mapa detalhado.

6.3.1.2. Atores

Qualquer usuário

6.3.1.3. Pré-condições

O usuário deve estar executando a aplicação *MapApp*, instalada em conjunto com o Kaluana, em seu dispositivo móvel. O dispositivo deve possuir o sistema operacional Android, algum tipo de conexão sem fio e acesso a interface de GPS.

6.3.1.4. Fluxo Básico

Durante o tempo em que o ator percorre uma área não demarcada, seja a pé ou com veículo automotivo, o aplicativo exibe o mapa do *Google*. Este mapa é representado pelo componente default da aplicação (*GoogleMapComponent*) que exibe apenas o mapa do *Google* com um marcador exibindo a posição atual do ator, como exemplificado na Figura 25.

6.3.2. Ator entra em uma área demarcada

6.3.2.1. Descrição

Este caso de uso permite que o ator aceite ou recuse o uso de um mapa detalhado ao entrar em uma área demarcada.

6.3.2.2. Atores

Qualquer usuário

6.3.2.3. Pré-condições

O usuário deve estar executando a aplicação *MapApp*, instalada em conjunto com o Kaluana, em seu dispositivo móvel. O dispositivo deve possuir o sistema operacional Android, algum tipo de conexão sem fio e acesso a interface de GPS.

6.3.2.4. Fluxo Básico

Ao entrar em uma área demarcada, é informado ao ator que existe um mapa detalhado daquela área e ele é questionado se deseja exibir este mapa detalhado, como exemplificado na Figura 26**.**

1. Ator entra em uma área demarcada

2. Aplicação informa que aquela área possui um mapa detalhado e pergunta se o ator deseja carregá-lo.

3. Ator aceita o carregamento do mapa detalhado

4. Aplicação requisita uma reconfiguração dinâmica para carregar o componente que exibe o mapa detalhado da área em que o usuário se encontra.

5. Kaluana recebe a requisição da aplicação e verifica em seu repositório se o componente está disponível para instanciação ou se existe um componente equivalente.

6. Kaluana avalia o contexto computacional do dispositivo

7. Kaluana avalia as restrições de execução dos componentes que atendem a requisição da aplicação.

8. Kaluana escolhe o componente a menor restrição computacional

9. Kaluana instancia o componente escolhido.

10. Kaluana retorna o componente para a aplicação.

11. Aplicação exibe o mapa detalhado para o ator (Figura 27).

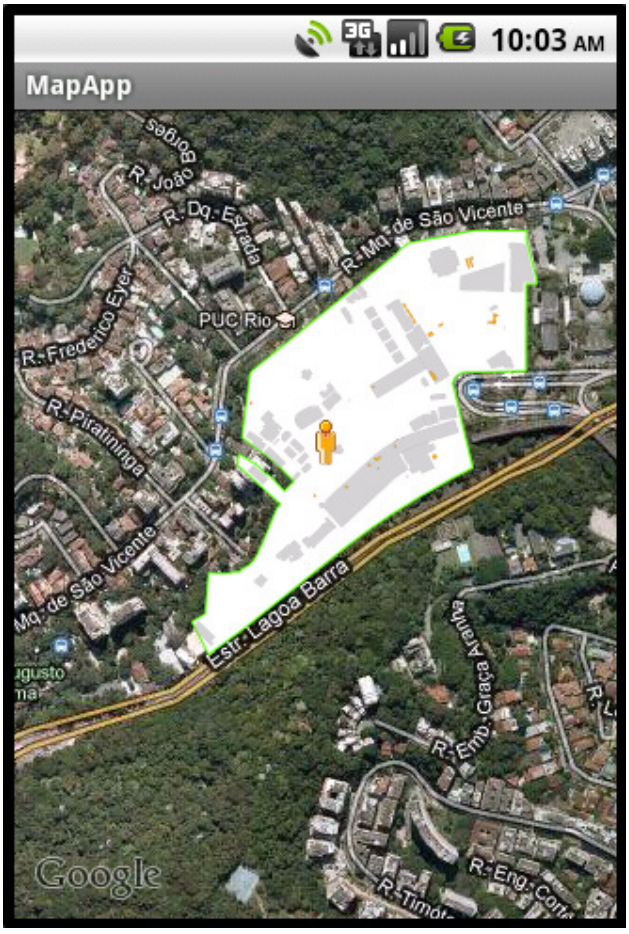

Figura 27. Mapa detalhado da PUC-Rio

6.3.2.5. Fluxos Alternativos

A. Ator cancela a exibição do mapa detalhado de uma área.

   Este fluxo acontece no passo dois do fluxo básico. Neste caso, a aplicação não realiza a requisição do componente ao Kaluana e exibe o componente *default*.

B. Kaluana não encontra, no repositório local, o componente requisitado.

   Este fluxo acontece no passo cinco do fluxo básico. Neste caso o seguinte fluxo acontece:

   1. Kaluana envia uma requisição *http* a um repositório remoto pré-definido

   2. Repositório remoto responde a requisição retornando o componente escolhido.

3. Caso de uso continua no passo nove do fluxo básico.

C. Kaluana não encontra o componente requisitado.

Este fluxo acontece no passo cinco do fluxo básico. Neste caso, o Kaluana retorna um erro de carregamento à aplicação.

D. Kaluana encontra o componente requisitado, mas o contexto computacional não atende as restrições de execução definidas no contrato de reconfiguração do componente. Este fluxo acontece no passo 7 do fluxo básico.

1. Kaluana avalia as restrições de execução de um componente equivalente ao requisitado. Caso todos os componentes avaliados possuam alguma infração de suas restrições de execução, o caso de uso segue para o fluxo alternativo B.

6.4. Middleware Kamaiurá

O *middleware* Kamaiurá, proposto em [70], foi criado como um serviço adicional executando sobre o *middleware* Kaluana descrito aqui. O objetivo do Kamaiurá é para evitar que uma aplicação pare de executar definitivamente em virtude da violação das restrições de execução de alguns de seus componentes. Por exemplo, quando o nível de bateria ou memória é insuficiente para que haja a instanciação de um novo componente disponível no repositório do Kaluana, o Kamaiurá provê uma interface de reconfiguração que executa os serviços do componente em servidores remotos, retornando para a aplicação apenas o resultado da execução.

Provendo-se das funcionalidades do Kaluana, especialmente as propostas nessa dissertação, a nova extensão do *middleware* Kaluana possibilita o tratamento da adaptação dinâmica das aplicações, através da reconfiguração e da redistribuição dos componentes das aplicações entre um dispositivo cliente e um servidor da rede. Essa extensão foi construída no topo do Kaluana, a fim de utilizar as facilidades que esse *middleware* dispõe para a construção de aplicações móveis cientes de contexto. A Figura 28 mostra a organização da arquitetura do *middleware* Kamaiurá sobre o Kaluana.

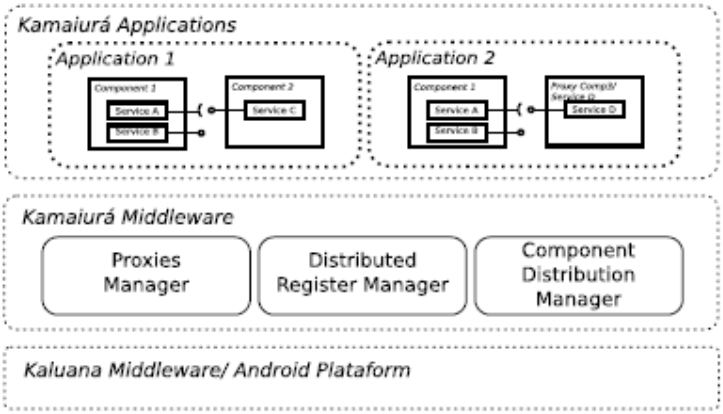

Figura 28. Arquitetura do Kamaiurá sobre o *middleware* Kaluana

Na figura, são mostrados os três serviços principais do Kamaiurá:

- *Proxies Manager.* Este gerenciador é responsável por criar e gerenciar *proxies* locais de modo que estes possam retransmitir, de forma transparente, as chamadas a métodos do serviço original do componente, alocado em um servidor remoto
- *Distributed Register Manager.* Em ambientes distribuídos, com múltiplos provedores de serviço, antes de um dispositivo cliente poder acessar qualquer componente ou serviço, ele deve descobrir onde este serviço está executando. A tarefa deste gerenciador é ficar responsável por implementar esta descoberta de protocolos de serviço, de modo a acessar componentes e seus serviços em seus respectivos provedores.
- *Component Distribution Manager.* Este é o serviço principal do *middleware* Kamaiurá. Ele é responsável por determinar e executar a distribuição/migração de componentes de uma aplicação. Para isto, a distribuição depende dos recursos disponíveis no dispositivo móvel e seus objetivos. Quando um recurso específico de um dispositivo estiver abaixo do adequado para a execução de um componente, este gerenciador indica como os componentes devem ser redistribuídos.

O Kamaiurá faz uso da **noção de restrições computacionais** de execução de cada componente, introduzida neste trabalho, para determinar qual é a melhor redistribuição dos componentes das aplicações que devem ser reconfiguradas. Por exemplo, caso seja necessário redistribuir certo número de componentes, o

Kamaiurá faz a requisição dos componentes ideais pelo Kaluana (avaliados a partir da comparação de seus contratos de reconfiguração), para que possa então realizar a redistribuição nos demais nós da rede. Fica a cargo do Kaluana, avaliar as políticas de adaptação dos componentes candidatos e prover os componentes considerados como menos restritivos, para que então o Kamaiurá possa redistribuí-los remotamente.

A quantidade de recursos disponíveis no dispositivo cliente, os componentes das aplicações, a forma como esses componentes estão interligados para compor as aplicações (isto é, o **contrato de reconfiguração**) e as restrições computacionais de cada componente são as informações de entrada para a execução do algoritmo de redistribuição dos componentes. Esse algoritmo irá determinar quais componentes podem permanecer executando no cliente e quais componentes devem ser movidos para o servidor, de modo que se minimize o custo total de redistribuição desses componentes.

A remoção dos componentes que estão implantados localmente para a sua execução no servidor, implica na criação de *proxies* para esses componentes remotos a fim de que possam ser acessados de maneira transparente pela aplicação, como se estivessem executando localmente no cliente.

Além do uso da noção de restrições computacionais, do conceito de contrato de reconfiguração, a nova extensão do Kaluana utiliza também como entrada os componentes escolhidos para compor a aplicação segundo a definição do **conceito de equivalência de interfaces públicas,** aqui apresentado. Isto é necessário pois através deste conceito, é possível utilizar-se componentes semanticamente equivalentes como candidatos para a redistribuição remota. Sem a noção de equivalência, caso houvesse os mesmos componentes disponíveis para atender uma requisição, apenas um deles seria considerado como candidato a uma reconfiguração, restringindo a abrangência e a eficácia do algoritmo proposto.

Sendo assim, as três principais extensões ao Kaluana original implementadas neste trabalho de dissertação foram de extrema importância para o Kamaiurá, de forma a permitir a reconfiguração e a redistribuição dos componentes das aplicações.

6.5. *Guideline* de criação de uma aplicação dinâmica usando o Kaluana

Para criar uma aplicação dinamicamente adaptável usando o *middleware* Kaluana, os desenvolvedores das aplicações devem seguir alguns pré-requisitos necessários para que o Kaluana consiga realizar a troca de seus componentes de *software* em tempo de execução.

O Kaluana toma como base algumas premissas para o gerenciamento de uma aplicação:

1. Cada aplicação deve saber o nome do componente ou o nome de um serviço implementado pelo componente que ela necessita

   Isto é necessário para realizar a requisição de um componente no Kaluana. A pesar de realizar o gerenciamento dos componentes de uma aplicação, o *middleware* não tem (e não deve ter) o conhecimento do que os componentes executam, evitando uma amarração do código do *middleware* com o código da aplicação.

   A aplicação deve conhecer previamente os componentes necessários para sua execução e realizar a requisição destes componentes de acordo com os métodos definidos na seção 5.5.

2. Assume-se que a aplicação define o momento de cada reconfiguração

   Portanto, no Kaluana não é feito o gerenciamento completo do ciclo de vida dos componentes. Sendo assim, é tarefa da aplicação estabelecer o momento exato de requisitar uma reconfiguração dinâmica, seja para implantar, como para descartar um componente. Por exemplo, na aplicação exemplo descrita na seção 6.1, uma reconfiguração dinâmica é requisitada toda vez que um usuário cruza uma área demarcada como contendo um mapa detalhado.

   Isto evita que o Kaluana necessite ficar consultando regularmente às aplicações para ver se elas necessitam de uma reconfiguração, o que consumiria recursos do dispositivo.

3. A aplicação é responsável por vincular os receptáculos de um componente requisitado.

   A pesar de o middleware realizar a instanciação dos componentes, o gerenciamento da utilização de um componente é sempre feito pela aplicação. Isto evita que o Kaluana necessite ter conhecimento da implementação do componente.

A partir destas premissas, a aplicação precisa definir os seguintes pontos para poder realizar a requisição de um componente dinamicamente:

- Referenciar o *jar* do Kaluana em seu *build path*
- Manter uma referência para a interface do gerenciador de componentes no objeto que executa a requisição.
- Criar um *intent* [71] utilizando o nome da interface do gerenciador de componentes e fazer o *bind* da aplicação neste serviço, como mostrado na Figura 29.

```
Intent intent =
  new Intent(kaluana.api.control.IComponentManager.class.getName());
bindService(intent, mServiceConnection, Context.BIND_AUTO_CREATE);
```

Figura 29 Realizando o *bind* de uma aplicação no serviço de gerenciamento de componentes do Kaluana

- Criar uma *callback* para a conexão de serviço do Android. Esta *callback* é passada como parâmetro para o *bind* da aplicação no serviço de gerenciamento de componentes, representado na Figura 29 pelo parâmetro *mServiceConnection*, e será chamada quando o Android terminar de conectar a aplicação.
- Criar um *BroadcastReceiver* [72] para receber os componentes instanciados pelo Kaluana e cadastrar a aplicação como listener deste objeto. Isto é necessário, pois o Android não permite observadores remotos em sua implementação. Todos os objetos transitados pela rede devem ser enviados pelo servidor através de um *broadcast* e capturados pelo receptor através de um *BroadcastReceiver* no lado do cliente.

  Este *receiver* é responsável por receber a notificação do Kaluana ao final do carregamento de todos os componentes requisitados

pela aplicação. Por este motivo, ao registrar a o *receiver* no contexto da aplicação, é necessário usar sempre o identificador "*kaluana.components.loaded*". A Figura 30 mostra um exemplo deste registro.

```
Receiver rec = new Receiver();
rec.addIcomponentManagerListener(getListener());
getApplicationContext().registerReceiver(rec,
  new IntentFilter("kaluana.components.loaded"));
```

Figura 30. Registro do *receiver* de notificação do carregamento dos componentes requisitados.

- Implementar a interface *IComponentManagerListener*. Esta é a interface que implementa a *callback* utilizada na notificação de que o carregamento dos componentes foi concluído. O desenvolvedor da aplicação deve implementar o método *componentsLoaded(List<String> components)* que é onde ele fará a manipulação dos componentes carregados pelo *middleware* durante a reconfiguração dinâmica. Neste método são realizadas as conexões dos receptáculos dos componentes e é iniciada sua utilização pela aplicação.