5
Mecanismo de seleção de componentes

O Kaluana Original, apresentado em detalhes no capítulo 3 deste trabalho, é um *middleware* que facilita a construção de aplicações para dispositivos móveis com suporte a adaptação dinâmica de componentes, usando a plataforma Android. No entanto, a pesar de realizar a adaptação de aplicações em tempo de execução, o Kaluana Original não possuía um mecanismo para selecionar o componente ideal a ser instanciado/ativado, de acordo com o contexto de execução do dispositivo.

O mecanismo de seleção de componentes efetua uma busca de componentes candidatos que atendam aos requisitos de compatibilidade de execução da aplicação. Estes componentes então são avaliados de acordo com seus contratos de reconfiguração e, de acordo com as restrições computacionais descritas nestes contratos, é escolhido o componente ideal de acordo com o contexto atual de execução do dispositivo. O mecanismo de seleção é importante para realizar uma instanciação mais adequada dos componentes a serem usados, de acordo com o contexto.

Cada componente do novo Kaluana deve possuir um contrato de reconfiguração. Neste contrato, são definidos seus serviços, receptáculos e suas restrições de execução. Os serviços e receptáculos de um componente são seus únicos pontos de conexão com a aplicação e com suas dependências [12].

O contrato de reconfiguração de um componente é usado para avaliar se ele é compatível com uma aplicação que esteja em processo de reconfiguração. Neste momento, são avaliados os serviços disponibilizados por um componente, assim como suas restrições computacionais.

Todo componente deve ser registrado no repositório do *middleware*, passando como referência seu contrato de reconfiguração, como é descrito na seção 5.4. Após este registro, este componente torna-se disponível para ser usado por uma aplicação.

A aplicação é responsável por iniciar a adaptação dinâmica, requisitando o componente que for necessário para sua execução. Após a requisição, o Kaluana fica responsável por avaliar o pedido e escolher, de acordo com a leitura dos contratos de reconfiguração dos componentes candidatos, o melhor componente a ser instanciado de acordo com o contexto de execução no momento da reconfiguração. O contexto de execução do Kaluana é provido através da integração do *middleware* com a plataforma Mobilis, onde o serviço de gerenciador de contexto (CMS) realiza a avaliação do contexto provendo detalhes do contexto do dispositivo para o Kaluana.

5.1. Restrições computacionais para a execução de um componente

Os componentes registrados no Kaluana podem possuir restrições de execução indicando o contexto mínimo para que ele seja executado. Estas restrições representam os requisitos não funcionais de cada componente [68] e ajudam o *middleware* a definir qual o melhor componente a ser instanciado durante a requisição de reconfiguração de uma aplicação. O componente é escolhido sempre de acordo com o contexto computacional encontrado no dispositivo no momento da reconfiguração. Desta maneira, um componente pode ser ideal em um momento e, alguns minutos depois, não ser o mais indicado para ser usado. Isto acontece, pois o contexto computacional de um dispositivo varia a todo o momento, como por exemplo, a sua quantidade de energia, a quantidade de memória disponível, ou o tipo de conectividade sem fio.

O desenvolvedor de um componente fica responsável em realizar testes previamente e definir quanta energia ou memória é necessária para executá-lo. O desenvolvedor também deve definir qual o tipo de conexão é necessária para que seu componente execute, assim como a velocidade de conexão necessária para a execução.

Alguns critérios de comparação foram convencionados para escolher qual o componente disponível atenderia a requisição da melhor maneira, considerando o contexto computacional do dispositivo no momento da requisição. Os tipos de restrição criados para esta dissertação foram: restrição de energia, restrição de

memória e restrição de conectividade. Cada uma delas é descrita em mais detalhes abaixo:

- Restrições de energia

  Para a comparação de restrições de energia, são verificados apenas os níveis mínimos de energia necessários para executar o componente. Neste ponto, é considerado o melhor componente, aquele que tiver a menor restrição de energia. Por exemplo: Caso o componente *A* necessite de, no mínimo, 30% de energia para executar e o componente *B* precise de no mínimo 40% de energia, o componente *A* será considerado melhor por ter uma restrição menor.

- Restrições de memória

  Os pontos considerados quando comparando restrições de memória são dois: Quantidade de memória do dispositivo e nível mínimo de memória disponível no momento.

  Com respeito à primeira restrição, é selecionado o componente que necessitar de uma menor quantidade de memória física no dispositivo. Por exemplo: Se o componente *A* necessita que o dispositivo tenha, no mínimo, 64Mb de memória física e o componente *B* necessita de 56Mb de memória física, o componente *B* é escolhido por ser menos restritivo. Assim como na primeira comparação, a segunda restrição escolhe o componente que necessite de um menor nível de energia disponível. Por exemplo: Se o componente *A* necessita que o dispositivo tenha, no mínimo, 64Mb de memória física livre e o componente *B* necessita de 56Mb de memória física livre, o componente *B* é escolhido por ser menos restritivo.

- Restrições de conectividade

  As restrições de conectividade são as que possuem a comparação mais complexa. É tomado como princípio básico que os dispositivos **sempre** possuam a possibilidade de se conectar

com conexões GPRS, 3G ou *Wifi*. Sendo assim, caso um dispositivo esteja conectado usando GPRS em certo momento, isso significa que nenhuma rede 3G ou *Wifi* está disponível naquele momento. Visto que, em sua maioria, as redes *Wifi* são gratuitas e possuem uma velocidade superior a redes 3G, elas são consideradas as mais restritivas.

Deste modo, as redes são comparadas do nível menos restritivo para o mais restritivo como: GRPS->3G->*Wifi*. Isto significa que se um componente necessita apenas de uma conectividade GPRS, ele é menos restritivo que um componente que necessita de uma conectividade 3G ou *Wifi*.

Assim, caso sejam comparados 3 componentes (*A*, *B* e *C*) onde o componente *A* necessite apenas de conectividade GPRS, o componente *B* requisite, no mínimo, de uma conectividade 3G e o componente *C* tenha a necessidade de uma conectividade *Wifi*, o primeiro será escolhido como o menor pois ele poderá usar tanto GPRS, 3G e *Wifi* enquanto os outros não poderão usar todas os tipos de conectividade.

A introdução do conceito de restrições computacionais possibilita uma comparação entre diferentes componentes candidatos à instanciação, permitindo que o Kaluana possa escolher entre diferentes componentes, qual o mais adequado a ser instanciado no momento da reconfiguração de uma aplicação.

Outra vantagem é que o conceito de restrições computacionais permite que o Kaluana não faça a instanciação de componentes que possuem incompatibilidade com o contexto computacional do dispositivo durante a reconfiguração. Por exemplo: caso um componente que necessite de, no mínimo 30% de energia disponível, ele não será instanciado se o nível de energia do dispositivo for menor que 30% mesmo que ele seja o único componente disponível.

5.2. Contrato de reconfiguração de um componente

O mecanismo de reconfiguração de uma aplicação baseia-se na compatibilidade de um componente com a necessidade de um serviço por um *software* usado pelo usuário. Na versão original do Kaluana [12], os contratos de reconfiguração dos componentes eram feitos através da anotação @Component, a qual recebia como parâmetro uma *string* informando a categoria em que aquele componente pertencia. Por exemplo, caso o desenvolvedor crie um componente para o mapa da PUC-Rio, ele poderia classificar este componente como @Component(category="kaluana.examples.maps") ou @Component(category="kaluana.examples.maps.puc-rio").

No entanto, esta especificação de um componente mostra-se bastante limitada, pois pode causar conflitos como o exemplo em que dois componentes pertencem a uma mesma categoria, mas implementam serviços diferentes. Isto pode gerar erros no *software* que só serão notados em tempo de execução, quando a aplicação tentar usar um serviço que não é implementado pelo componente que foi instanciado e fornecido a ela.

Um exemplo básico disso seria um software estar executando um componente da categoria “kaluana.examples.maps” que usa o mapa da PUC-Rio e, ao notar que o usuário saiu da universidade, a aplicação requisitar uma reconfiguração e receber outro componente desta mesma categoria. Neste momento, caso vários componentes estivessem classificados na categoria "kaluana.examples.maps" (e.g. *google maps*, o componente que disponibiliza o mapa específico de outra instituição ou qualquer outro componente de mapa) o Kaluana original não teria como diferenciar estes componentes de mapa.

Para tratar desse problema, neste trabalho foi desenvolvido outro tipo de contrato de reconfiguração de um componente onde estes são classificados através dos serviços e receptáculos que possuem e não mais através de uma *string* que representa uma categoria.

O contrato de reconfiguração de um componente possui a referência para os serviços, receptáculos e restrições de execução de um componente, as quais são explicadas de maneira mais aprofundada na seção 5.1. A Figura 15 mostra um exemplo da criação do contrato de reconfiguração de um componente. Neste

exemplo, o componente criado disponibiliza o serviço *MapPUCService*, possui o receptáculo *GoogleMapService* e possui uma restrição de energia que informa que ele executa apenas com o mínimo de 30% de energia disponível.

```
List<IRestriction<RestrictionType>> componentRestrictions =
  new ArrayList<IRestriction<RestrictionType>>();
List<String> serviceNames = new ArrayList<String>();
List<String> receptacleNames = new ArrayList<String>();
serviceNames.add("MapPUCService");
receptacleNames.add("GoogleMapService");
componentRestrictions.add(new EnergyRestriction(30));
```

Figura 15. Definição do contrato de reconfiguração de um componente

Desta maneira, quando é feita uma requisição de para o Kaluana, o *middleware* retorna sempre um componente que possua o serviço requisitado pela aplicação, diminuindo a possibilidade de incompatibilidade durante a reconfiguração.

5.3. Integração do Kaluana com a plataforma Mobilis

As aplicações dinamicamente adaptáveis no Kaluana são reconfiguradas de acordo com as informações de contexto durante o momento da adaptação. Para isto é avaliado o contexto computacional do dispositivo, além das restrições de execução de cada componente candidato.

Para flexibilizar a ciência ao contexto do *middleware* Kaluana, foi realizada uma integração com a plataforma Mobilis, de maneira que o desenvolvedor do *middleware* não fosse responsável pelo desenvolvimento do código referente à ciência ao contexto. Para isto, o Kaluana se cadastra como assinante de determinados dados/eventos de contexto junto ao CMS, transferindo a responsabilidade de gerenciar os provedores de contexto correspondentes para o CMS. A vantagem de usar o CMS como gerenciador de informações de contexto é modularizar melhor o sistema de modo a permitir que o desenvolvimento deste gerenciador seja feito em paralelo com o desenvolvimento do próprio *middleware* Kaluana. Isto promove uma modularização melhor de código.

No momento da inicialização do repositório de componentes do Kaluana, são registrados no CMS os interesses nos provedores de contexto necessários para a avaliação das políticas de adaptação dos componentes. Neste instante, é

verificado se o CMS está ativo e então, são registrados os interesses necessários. A Figura 16 mostra como é feito o registro do consumidor de informações de contexto no Kalauana. No exemplo, o consumidor é adicionado como observador do CMS e aguarda a resposta sobre a ativação dos provedores de contexto disponíveis. Quando esta informação está disponível, uma *callback* é chamada e o consumidor registra os interesses necessários.

```
protected void registerKaluanaConsumer() {
  try {
    myConsumer = new KaluanaContextConsumer(getApplicationContext(), msgHandler);
    myConsumer.addObserver(this);
    update(null, new Boolean(ContextConsumer.isActive()));
  }
  catch (Exception e) {
    Log.e("cms-client", "error while trying to create consumer - " + e.getMessage(), e);
  }
}

@Override
public void update(Observable observable, Object data) {
  boolean active = (Boolean) data;
  if (active) {
    try {
      myConsumer.addContextInformationInterest("this.battery.level");
      myConsumer.addContextInformationInterest("this.energy.level");
      myConsumer.addContextInformationInterest("this.conectivity.type");
    }
    catch (Exception e) {
      Log.e("cms-client", "error while trying to add consumer - " + e.getMessage(), e);
    }
  }
}
```

Figura 16. Registro do consumidor de informações de contexto no Kaluana

Os interesses dos provedores de contexto dependem do conteúdo necessário para avaliar as políticas de adaptação de cada componente. Por exemplo, caso existisse definido no *middleware* Kaluana apenas a restrição de energia para a execução de um componente, não faria sentido avaliar provedores de contexto com informações referentes ao nível de ruído, pois este tipo de informação não seria aproveitada na avaliação dos componentes.

Para este trabalho, foram criados três tipos de interesses de contexto para recuperar o contexto do dispositivo: nível de energia do dispositivo, quantidade de bateria e tipo de conectividade sem fio. Estes interesses foram escolhidos de acordo com as restrições de execução criadas, descritas na seção 5.1.

5.4. Registro de componentes no Kaluana

Para que um componente seja reconhecido pelo Kaluana e possa ser instanciado durante a adaptação dinâmica de uma aplicação, ele deve ser registrado por uma aplicação criada pelo desenvolvedor do componente. Esta *aplicação registro* consiste *apenas* em um código simples que captura uma instância do gerenciador de componentes do Kaluana e realiza o registro dos componentes que ela declara. O registro de um componente é feito através da criação de uma entrada no repositório local do Kaluana, onde é guardado o contrato de reconfiguração do componente.

Este cadastro é feito pelo desenvolvedor, que registra o componente no repositório, evitando que o Kaluana necessite descobrir novos componentes que possam ser instanciados, através de uma varredura constante no dispositivo. Isto diminui a carga de processamento do *middleware* para a busca de um serviço que seja compatível com uma aplicação, pois não é mais necessário usar reflexão computacional para descobrir as características de um componente (serviços, receptáculos e restrições), sendo estas cadastradas no repositório durante seu registro. A Figura 17 mostra um exemplo do registro de um componente no Kaluana. No exemplo, é cadastrado o componente *MapPUCAppComp*, que disponibiliza o serviço *MapPUCService* e não possui receptáculos nem restrições.

```
componentManager = IComponentManager.Stub.asInterface(service);
try {
  List<String> serviceNames = new ArrayList<String>();
  serviceNames.add("MapPUCService");
  componentManager.registerComponents("MapPUCAppComp", serviceNames,
    null, null);
}
catch (RemoteException e) {
  e.printStackTrace();
}
```

Figura 17. Exemplo de registro de um componente no Kaluana

Este registro pode ser realizado de duas maneiras: *manualmente* ou *automaticamente*. Quando ele é feito de forma manual, o usuário adquire o arquivo *apk* do componente e realiza a instalação manual a partir de um cartão de memória [69]. No entanto, quando a instalação é feita de forma automática, o próprio Kaluana faz o *download* e o registro do componente no dispositivo. Este procedimento é descrito com mais detalhes na seção 5.6 desta dissertação.

Por motivos de desempenho, o repositório do Kaluana não guarda o objeto do componente instanciado. O contrato de reconfiguração é constituído apenas de *strings* que indicam as características do componente (serviços, receptáculos e restrições). Isto torna o repositório mais leve fazendo com que, mesmo que possua uma quantidade grande de componentes cadastrados, não ocupe um espaço significativo no dispositivo. O fato de as aplicações que realizam o registro de componentes lidarem apenas com *strings* faz com que a serialização dos registros seja mais simples por se tratar de objetos simples de java.

A princípio, um componente registrado permanece no repositório por tempo indefinido. Caso o desenvolvedor não queira que seu componente seja disponibilizado pelo Kaluana, ele deve criar uma aplicação para "descadastrá-lo" do *middleware*. Esta aplicação é similar a aplicação que faz o cadastro do componente.

5.5. Requisição de um componente por uma aplicação

A interpretação do contrato de reconfiguração dos componentes é feita quando uma aplicação requisita que um componente seja carregado. Neste momento, o *ComponentManager* intercepta a requisição e a repassa para o *ComponentRepository* tratá-la. Este então procura, em seu repositório, pela entrada que atende da melhor maneira a aplicação.

A decisão de qual é o componente que atende da melhor maneira a requisição é feita de acordo com a avaliação das restrições de execução dos componentes registrados no repositório, como explicado na seção 0. Uma aplicação pode requisitar um componente de duas maneiras: através de seu nome ou indicando o nome de um ou mais serviços necessários.

Para que o componente seja requisitado através de seu nome, a aplicação deve ter o conhecimento prévio do componente e fazer sua requisição através do nome declarado no Android. Este nome é representado pelo nome que o desenvolvedor do componente definiu no manifesto da aplicação que faz o registro no Kaluana, como no exemplo da Figura 18, onde é declarado o componente *GoogleMapComponent*.

```
<service android:name="kaluana.examples.google.map.comp.src.GoogleMapContainer">
    <intent-filter>
        <action android:name="GoogleMapComponent"></action>
    </intent-filter>
</service>
```

Figura 18. Declaração de um componente no manifesto do Android.

Um exemplo do procedimento de requisição de um componente através de seu nome pode ser visto na Figura 19.

```
List<String> componentNames = new ArrayList<String>();
componentNames.add("GoogleMapComponent");
componentManager.loadComponentsByName(componentNames, getListener());
```

Figura 19. Requisição de um componente por nome

Neste caso, a aplicação receberá apenas as diversas versões de um mesmo componente, caso existam, não tendo a possibilidade de usar componentes diferentes que sejam equivalentes. Por este motivo, esta é uma implementação mais restrita, onde a aplicação possui uma variedade menor de opções de reconfiguração. Porém a aplicação tem a certeza de que o componente desejado é exatamente aquele o qual foi requisitado.

De maneira análoga, a aplicação pode fazer a requisição do componente através do nome de um ou mais serviços necessários para sua execução. Para este procedimento, a aplicação faz a requisição de um componente que implemente certo *serviço* e/ou possua certo *receptáculo* em seu contrato. Para isto, ela faz uma requisição ao *ComponentManager*, passando uma lista com o nome do *serviços* e/ou *receptáculos* desejados, como é mostrado na Figura 20.

```
List serviceNames = new ArrayList<String>();
List receptacleNames = new ArrayList<String>();
serviceNames.add("MapPUCService");
receptacleNames.add("GoogleMapComponent");
componentManager.loadComponentsByService(serviceNames,
  receptacleNames, getListener());
```

Figura 20. Requisição de um componente pelo nome de um serviço necessário

O gerenciador repassa o pedido para o repositório, onde o pedido será avaliado. Este então fará uma busca de todos os componentes que possuem estes serviços/receptáculos em seus contratos de reconfiguração.

Uma lista é montada com todos os componentes candidatos, que atendem os pré-requisitos necessários para a requisição da aplicação. Após a montagem desta lista, são verificadas as restrições de cada componente, como descrito na seção 0.

Dentre os componentes cujas restrições de execução são compatíveis com o contexto computacional do dispositivo no momento da requisição, é feita uma comparação para escolher qual atende melhor a aplicação, de acordo com os critérios de comparação convencionados para este trabalho.

Caso o desenvolvedor de outro sistema queira usar algum outro critério de escolha para o melhor componente a ser usado, bastaria estender a classe *ComponentRepository* e reimplementar o método *findBestComponent(ArrayList<String> componentNames)*. Este método recebe como parâmetro, o nome dos componentes que possuem em seu contrato de reconfiguração os serviços requisitados pela aplicação e é responsável por toda a inteligência necessária para avaliar qual o melhor componente a ser instanciado de acordo com os critérios de comparação de restrições de execução descritos na seção 5.1.

5.6. Equivalência de componentes

Devido à nova maneira de buscar e instanciar um componente Kaluana, os componentes agora podem possuir uma equivalência entre si, gerando uma maior flexibilidade na hora de reconfigurar uma aplicação. Dois componentes são ditos equivalentes, com relação a uma requisição, caso possuam em seus contratos de reconfiguração pelo menos o(s) serviço(s) requisitado(s) pela aplicação, independente se possuem outros serviços declarados.

Por exemplo, caso um componente *A* implemente o serviço *x* e o componente *B* implemente o serviço *x* e o serviço *y*, os componentes serão considerados equivalentes pelo Kaluana caso a aplicação faça a requisição de um componente que implemente o serviço *x*. No entanto, estes mesmos componentes não serão considerados equivalentes caso a aplicação necessite do serviço *y* pois o componente *A* não possui implementação para este serviço.

Este tipo de equivalência só se aplica caso a requisição seja feita através do nome de um serviço implementado pelo componente, não sendo possível no caso de a requisição seja feita pelo nome do componente.

O Kaluana não considera as restrições de execução dos componentes para avaliar sua equivalência visto que elas não são parte da definição de sua interface

pública de execução e sim a definição de seus requisitos não funcionais de desempenho [68]. No entanto, as restrições são usadas durante a seleção do componente, para avaliar qual dos componentes candidatos está mais apto a ser instanciado. Para isso é usado o critério de comparação, descrito na seção 5.2, para avaliar qual o melhor componente candidato, a partir de suas restrições computacionais.

5.7. *Download* e instalação de componentes, a partir de um servidor remoto

Uma adaptação dinâmica é gerada a partir de uma mudança do contexto computacional do dispositivo. Esta mudança de contexto pode ocorrer a partir da mudança de localização do usuário, da alteração na quantidade de memória disponível ou mesmo pela diminuição brusca na quantidade de energia que o dispositivo possui. Durante a reconfiguração de uma aplicação, o Kaluana, a princípio, busca os componentes necessários, em um repositório local. No entanto, quando um componente não é encontrado localmente, é feita uma busca por componentes em um repositório remoto localizado em um servidor *http*.

O *download* de um componente é feito através de uma conexão *http* com um *WebService* do qual o *middleware* copia um arquivo *.apk* para o dispositivo. Este arquivo contém o código do componente e a aplicação que realiza seu registro no repositório do Kaluana. Após o *download* da aplicação de registro, o *middleware* tenta realizar a instalação do arquivo no dispositivo. Para isto, é requisitado ao usuário que aceite instalar este componente em seu dispositivo, como é mostrado na Figura 21. Por motivos de segurança, o Android não permite que seja feita a instalação de uma aplicação no dispositivo sem confirmação do usuário.

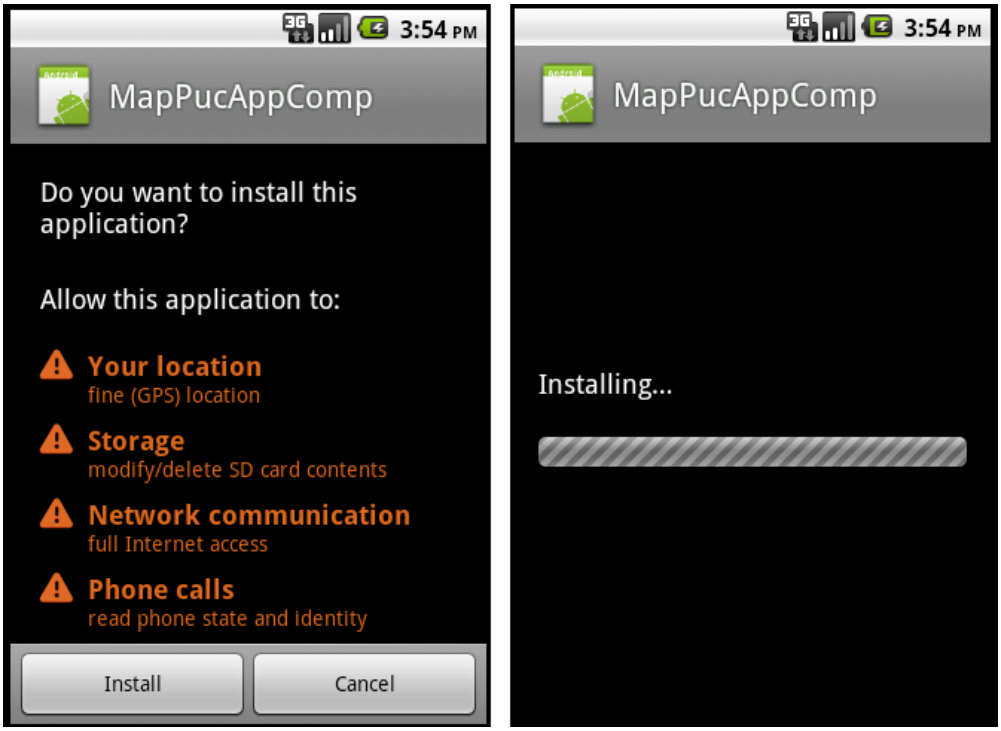

Figura 21 Instalação de um componente novo no dispositivo

Ao final do processo de instalação, o usuário tem a escolha de permitir que a aplicação utilize imediatamente o componente novo, selecionando o botão *Open* ao final da instalação, como é mostrado na Figura 22.

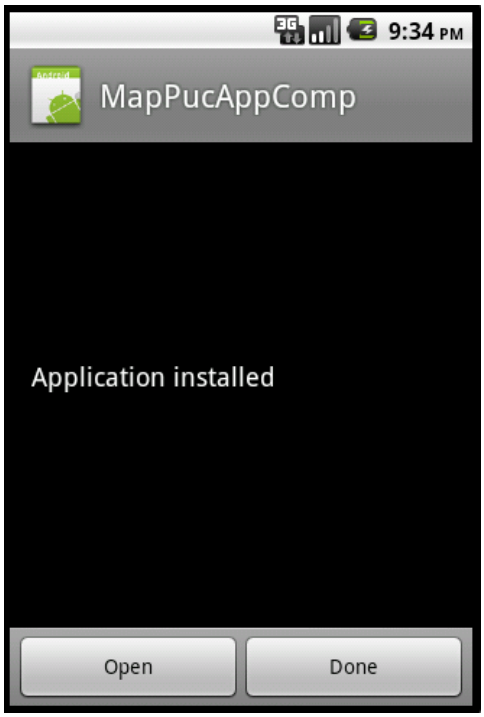

Figura 22 Usuário escolhe se deseja usar a aplicação instalada ou não

Este componente, mesmo não sendo usado imediatamente, não é descartado pelo *middleware*. Seu contrato de reconfiguração fica registrado no repositório de componentes do Kaluana para que ele possa ser usado em outra reconfiguração, caso necessário. Desta maneira, o Kaluana evita que todo o processo de *download* e instalação seja realizado múltiplas vezes para o mesmo componente, economizando energia do dispositivo.

A atual versão do Kaluana considera que o componente baixado de um servidor é sempre o componente ideal para ser instalado. No presente trabalho, não foi realizado o serviço de escolha e avaliação de um componente em um servidor remoto, pois isso implicaria em tratar aspectos que fogem ao escopo do projeto, como o tratamento de mudanças de contexto durante o processo de seleção de um componente no servidor. Por exemplo, um componente com a restrição de que o dispositivo deve possuir um mínimo de 30% de energia é indicado como o melhor componente para ser instanciado em certo momento e seu *download* é feito de um servidor remoto. No entanto, após ser registrado o componente, suas restrições podem ter sido violadas caso a energia tenha caído para 29% no meio tempo. Neste caso, o componente não pode ser mais instanciado e um novo *download* teria que ser realizado.