

2

Fundamentação conceitual

O *middleware* Kaluana permite a implementação de aplicações móveis dinamicamente adaptáveis, constituídas pela composição de componentes desenvolvidos seguindo o modelo Kaluana, modelo de componentes orientado a serviços. Aplicações Kaluana são adaptáveis funcional e estruturalmente, por meio da capacidade do *middleware* de realizar a substituição de serviços e componentes, respectivamente.

Aplicações móveis implementadas sobre o *middleware* Kaluana, ou aplicações Kaluana, consistem em aplicações que realizam composições entre componentes definidos sobre o modelo Kaluana e possuem controle sobre o ciclo de vida dos componentes, ou seja, podem ativar e desativar componentes após buscá-los em um repositório, também definido pelo *middleware* Kaluana.

Adaptações e implantações dinâmicas são motivadas por mudanças no contexto computacional ou de usuário, como descrito na seção 2.1, com objetivo de corrigir, otimizar ou estender o funcionamento de aplicações. Na seção 2.2, será apresentado e exemplificado o conceito de adaptação dinâmica e suas classificações quanto ao seu alvo, motivação e granularidade. O processo de implantação dinâmica, por sua vez, será apresentado na seção 2.3.

Depois de caracterizar adaptação e implantação dinâmicas, serão apresentados os conceitos de modelo de componentes, na seção 2.4, e orientação a serviços, na seção 2.5, abordagens que combinadas permitem ao mecanismo atender a requisitos de adaptação estrutural e funcional e de implantação dinâmica. O resultado desta combinação é o modelo de componentes orientado a serviços definido pelo *middleware* Kaluana, também chamado modelo Kaluana, descrito na seção 2.6.

Enquanto o modelo de componentes define unidades substituíveis de composição, promovendo a capacidade de implementar aplicações por meio de composições, a orientação a serviços implica a descoberta e uso de implementações em tempo de execução, o que torna possível, junto a técnicas de reflexão computacional, a realização de adaptações estruturais e funcionais e de implantações dinâmicas, mantendo uma fraca amarração entre os componentes e suas funcionalidades oferecidas por meio de serviços.

2.1

Contexto

Contexto [3] [4] é qualquer informação que possa ser utilizada para caracterizar a situação de uma pessoa, lugar ou objeto que possa determinar o comportamento de uma aplicação ou a realização de algum evento da aplicação que seja relevante ao seu usuário.

A ciência de contexto permite que aplicações possam adaptar-se a diferentes situações, de modo a otimizar ou corrigir seu funcionamento e melhorar sua interface com o usuário.

Informações de contexto podem ser expressas de diversas maneiras, e são resultado da interpretação de dados provenientes de um ou mais sensores, por exemplo nível de energia disponível na bateria, utilização da capacidade de processamento de um dispositivo, memória utilizada ou iluminação do ambiente onde está o dispositivo, quando cada um desses exemplos interfere na interação entre uma aplicação e um usuário.

2.2

Adaptação dinâmica

O interesse do usuário, a disponibilidade e variabilidade de recursos e a mobilidade do dispositivo e de seu usuário estão dentre as possíveis causas da necessidade de uma adaptação. Aplicações realizam adaptações para modificar seu funcionamento e adequar-se a diferentes condições de contexto.

Adaptações devidas a mudanças no ambiente de execução de uma aplicação, como no sistema operacional, em componentes de *hardware* ou no estado interno de outras aplicações, são classificadas como **adaptações reativas** (tradução livre: *adaptive adaptations*), segundo [5]. Por esta abordagem, a adaptação dinâmica ocorre para que a aplicação possa continuar atendendo a seus requisitos sob diferentes condições de execução.

Existem, ainda segundo os autores, outras razões que tornam necessárias uma adaptação dinâmica. **Adaptações corretivas** ocorrem para corrigir problemas. Neste caso, partes defeituosas do sistema são substituídas, mantendo-se inalteradas as funcionalidades do sistema. **Adaptações de extensão** ocorrem quando é necessária a adição de novas funcionalidades, não contempladas no desenvolvimento, porém mais tarde necessárias ao usuário. **Adaptações de otimização** são melhorias em implementações, substituindo componentes para melhorar o desempenho do sistema.

O momento de uma adaptação é definido segundo critérios baseados no motivo para sua realização. Devido ao alto grau de variação nas condições de contexto em dispositivos móveis, como nível de energia, qualidade ou existên-

cia de conectividade, bem como no contexto do usuário, que pode ser inferido a partir de informações como a localização do dispositivo, torna-se interessante que *frameworks* para desenvolvimento de aplicações móveis ofereçam mecanismos eficientes de adaptação dinâmica, para que desenvolvedores possam implementar comportamentos reativos a mudanças preemptivas nas condições de contexto. Por exemplo, ao detectar a perda de conectividade de um dispositivo, uma aplicação de edição colaborativa de mapas pode armazenar localmente informações criadas pelo usuário, e ao detectar a reconexão, realiza o envio dessas informações.

Além de classificar o motivo para uma adaptação dinâmica, outras duas questões são fundamentais: a granularidade e o objeto alvo da adaptação. Duas abordagens são comuns quanto à granularidade de adaptações dinâmicas: adaptações **em parâmetros** e adaptações **na composição** [6]. Na primeira, ocorrem modificações em variáveis que determinam o comportamento da aplicação. Segundo os autores, um exemplo clássico é a variação dos valores de controle de retransmissão e do tamanho das janelas no protocolo TCP. Neste caso, a otimização de uma implementação se restringe a algoritmos e componentes implementados no projeto e construção original da aplicação. A figura 2.1 mostra os tipos de adaptação de acordo com as três classificações apresentadas.

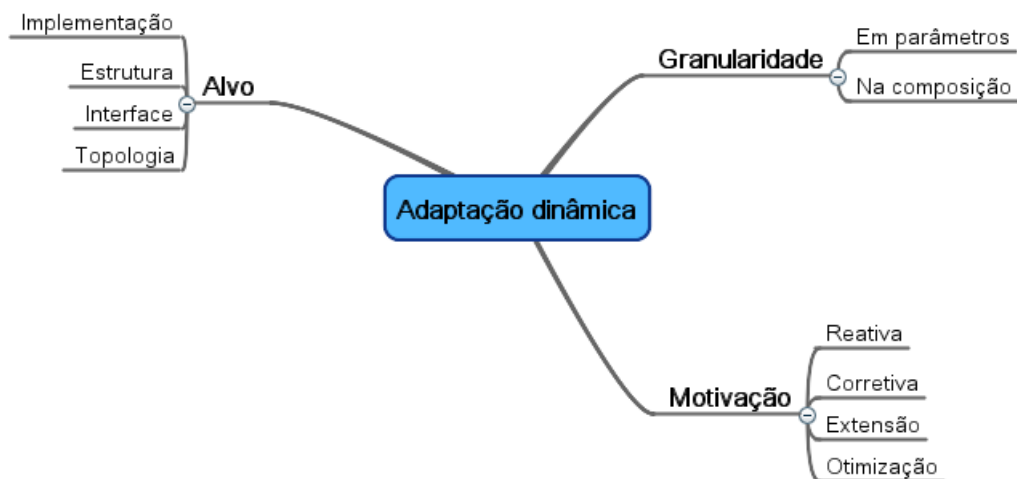


Figura 2.1: Classificações de uma adaptação dinâmica

Ao contrário da adaptação paramétrica, a adaptação na composição permite a incorporação de código desenvolvido após a implantação da aplicação, como novos algoritmos mais eficientes. Assim, oferece maior flexibilidade ao desenvolvedor e ao usuário. Este tipo de adaptação é necessário quando existe

limitação quanto ao número de componentes instanciados, como em dispositivos com pouca memória, ou quando novas implementações são necessárias para acomodar condições ou requisitos não previstos inicialmente.

Existem quatro estratégias que classificam o objeto de uma reconfiguração dinâmica na composição de uma aplicação [5]: adaptação **na implementação** dos componentes, adaptação **na estrutura** da aplicação, adaptação **na interface** de um componente ou adaptação **na distribuição** da aplicação.

A primeira é mais adequada a adaptações corretivas, desempenho e mudanças no ambiente não previstas em tempo de desenvolvimento. Nela, um componente é substituído, mantendo-se suas interfaces e funcionalidades inalteradas. Por exemplo, um componente de compactação de vídeo pode ser substituído baseado na taxa de transferência com que o vídeo é transmitido, ou um protocolo de comunicação pode ser alterado baseado na qualidade do sinal da rede sem fio.

A adaptação na estrutura da aplicação pode se dar pela introdução de um novo componente, remoção de um componente existente ou modificação na conexão entre componentes. Por exemplo, uma aplicação de notícias pode inibir um componente de *streaming* de vídeo ao perceber que o nível de energia disponível está baixo, substituindo-o por um componente que mostra uma imagem estática retirada do vídeo.

A adaptação na interface de um componente significa a introdução ou remoção de uma interface, modificando a lista de serviços provida pelo componente. Neste caso, as funcionalidades providas por um componente são modificadas.

A última estratégia, adaptação na distribuição da aplicação, se dá por questões de desempenho, como balanceamento de carga, e consiste na divisão de processamento ou armazenamento entre diferentes dispositivos físicos.

2.3

Implantação dinâmica

O processo de instalação consiste no *download* do arquivo de instalação do componente e na instalação deste arquivo. Uma vez instalado, o componente é configurado e ativado.

A implantação pode ser resultado de uma requisição da aplicação pelo uso de determinado componente inexistente no dispositivo ou de uma adaptação dinâmica.

Quando o processo de implantação é resultado de uma adaptação dinâmica, é necessário realizar no componente substituto todas as conexões que o componente substituído mantinha, bem como reconfigurar seu estado interno,

de modo a minimizar o impacto perceptível ao usuário de um processo de adaptação dinâmica. Esta tarefa é realizada durante a ativação do componente.

2.4

Modelo de componentes

Um componente de software é uma unidade binária de composição que contém interfaces contratualmente definidas e dependências explícitas. Um componente pode ser implantado separadamente e está sujeito a composições a outros componentes [7].¹

Sobre um modelo de componentes se dá a conexão de diferentes componentes, de modo a formar estas composições.

O ciclo de vida de um componente é composto por seis tarefas relacionadas [9]: instalação, ativação, atualização, adaptação, desativação e desinstalação.

Para tornar possível a interação entre componentes, cada um deve prover maneiras de descrever sua estrutura interna e a compatibilidade entre seus pontos de conexão com outros componentes: facetas e receptáculos.

A figura 2.2 mostra a representação utilizada neste trabalho da estrutura de dois componentes, provendo facetas e receptáculos, representados pelos conectores. Na figura, um receptáculo do componente A está conectado a uma faceta do componente B.

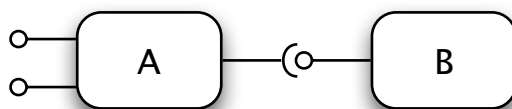


Figura 2.2: Representação de uma composição

Múltiplas instâncias de um componente podem existir, cada uma delas com determinado estado interno. Neste ponto, o conceito de componentes se aproxima ao conceito de classes em modelos orientados a objetos.

A abordagem de orientação a componentes torna clara a separação do processo de desenvolvimento de aplicações – onde realizam-se as composições – do desenvolvimento dos componentes. Logo, o desenvolvedor de aplicações não precisa conhecer detalhes sobre a implementação de um componente para usá-lo.

¹Não existe consenso na definição de componentes de software, porém esta definição é amplamente referenciada na literatura, como citado em [8].

Alguns exemplos de sistemas de componentes são COM [10], CCM [11] e Fractal [12].

2.5

Orientação a serviços

Um serviço é uma implementação associada dinamicamente a uma interface, que por sua vez é definida através da linguagem de definição de interfaces IDL (*Interface Description Language*) ou alguma variação desta.

A orientação a serviços está diretamente ligada à capacidade de descoberta e uso em tempo de execução. Existem três atores em uma abordagem orientada a serviços: o provedor do serviço, o consumidor do serviço e o registro do serviço. Três também é o número de tarefas associadas ao uso de um serviço: sua publicação, sua descoberta e seu uso. Todas são realizadas em tempo de execução.

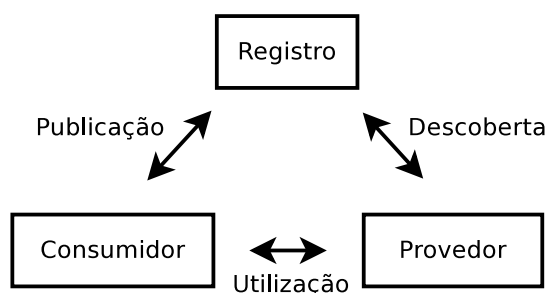


Figura 2.3: Atores e tarefas em um modelo orientado a serviços

O provedor de um serviço deve registrá-lo para torná-lo disponível ao uso, enquanto interessados em seu consumo devem buscar no registro por sua referência. Esta busca é realizada por meio da descrição do serviço, que ambos provedor e consumidor devem conhecer. Assim como ambos devem conhecer a interface do serviço.

Uma vez encontrado, o serviço pode ser utilizado depois de uma operação de amarração (*bind*). Esta operação pode resultar na criação de dois *stubs*, um no consumidor do serviço, o *stub* cliente, e um no provedor, o *stub* servidor. O *stub* cliente oferece métodos que, ao serem chamados, provocam a execução de métodos homônimos no *stub* servidor e retornam o resultado ao cliente como se fossem executados localmente.

Abordagens orientadas a serviços têm grande utilidade em dispositivos móveis, como celulares, PDAs e *MP3 players*, em que a variabilidade de recursos computacionais pode ser representada como disponibilidade de serviços, permitindo às aplicações tratar esta variabilidade como variação na disponibilidade destes serviços.

Outras características são comuns a modelos orientados a serviços, como a fraca amarração, pois o consumidor não conhece a implementação do provedor, o *late binding*, pois a implementação é conhecida somente em tempo de execução e a independência à localização, pois o serviço pode ser local ou remoto, e independente da infra-estrutura de comunicação.

Alguns exemplos de sistemas orientados a serviços são Jini [13], OSGi [14] e Android [1].

2.6

Modelo de componentes orientado a serviços

Tanto modelos orientados a componentes como modelos orientados a serviços se baseiam na composição de blocos reutilizáveis para a formação de aplicações: componentes ou serviços. Entretanto, há algumas diferenças básicas entre as duas visões quanto ao momento da definição das composições, a flexibilidade destas composições e a utilização destes blocos.

Modelos orientados a serviços prevêm a existência de diferentes implementações para um serviço, que podem ser substituídas ou tornarem-se indisponíveis em tempo de execução, promovendo assim a flexibilidade necessária para a realização de adaptações dinâmicas funcionais, ou adaptações na implementação. Apesar de modelos de componentes normalmente proverem a capacidade de adaptação estrutural, por meio da substituição de componentes, a capacidade de adaptação funcional é resultado de uma abordagem orientada a serviços.

Componentes são instanciados e seu uso se dá através de suas instâncias. Cada instância mantém conexões e estado interno próprios. Serviços, por outro lado, são utilizados sem que o consumidor tenha que explicitamente criá-los ou instanciá-los, por meio de referências para implementações já instanciadas.

Enquanto componentes descrevem contratualmente suas dependências, serviços as definem em tempo de execução.

Na tabela 2.1 são mostradas as diferenças básicas entre a composição de um sistema orientado a componentes e um sistema orientado a serviços em razão de seu momento, variabilidade e utilização dos elementos.

Modelo	Momento da definição	Flexibilidade	Definição de dependências
Componentes	Desenvolvimento	Na composição	Contratual
Serviços	Execução	Na implementação	Durante a execução

Tabela 2.1: A composição nos dois modelos: orientado a componentes e orientado a serviços

Aproveitando a capacidade de implantação, composição e instanciação de um modelo de componentes e a flexibilidade de um modelo orientado a serviços, é possível combiná-los de modo a prover tanto adaptações estruturais como funcionais, bem como a implantação dinâmica – capacidade normalmente inexistente em cada uma das abordagens.

O modelo *Kaluana*, diferentemente de alguns modelos de componentes, não amarra fortemente a implementação do componente à implementação dos seus serviços oferecidos. Assim, é possível que um componente se comporte simplesmente como um invólucro para serviços existentes *a priori*, se assim for desejado por seu desenvolvedor.

A substituição de serviços é classificada como adaptação na implementação e a substituição de componentes como adaptação na estrutura. Ambas, segundo a granularidade, são adaptações na composição das aplicações que constituem.

Os princípios de um modelo de componentes orientado a serviços são apresentados em [8] e definem a composição por meio da especificação de serviços implementados pelos componentes, ao invés das implementações dos componentes, como em modelos de componentes comuns.

Nesta abordagem, serviços são utilizados como facetas dos componentes, e receptáculos conectam a estes serviços, promovendo a composição de componentes. Serviços semelhantes podem ser agregados de modo a formar um componente, que pode implementar qualquer número de serviços.

A escolha de componentes mais adequados para realizar composições, que em modelos de componentes comuns é realizada em tempo de desenvolvimento, em modelos de componentes orientados a serviços pode ser realizada em tempo de execução, dada a dependência definida pela especificação dos serviços e não por sua implementação.