1

Introdução

Sistemas de computação fatalmente necessitam evoluir de alguma forma [1], seja para se adequarem a novas necessidades ou mesmo para aperfeiçoar o seu funcionamento. Além disso, como as mudanças nas necessidades do sistema podem ser freqüentes e rotineiras, a tarefa de evoluir um sistema deve ser rápida, barata e eficiente (*i.e.*, deve atender às novas necessidades satisfatoriamente). Essa evolução de software tem sido tópico de pesquisa há algum tempo, de onde surgiram modelos e ferramentas para construção de sistemas mais flexíveis e adaptáveis a mudanças [2, 3, 4].

Entretanto, com o avanço tecnológico e o surgimento de sistemas maiores, mais complexos e mais essenciais[1], a tarefa de construir sistemas flexíveis e adaptáveis vem se tornando substancialmente mais complexa. Novas pesquisas têm sido conduzidas no sentido de desenvolver técnicas para construção de sistemas mais flexíveis, além de abstrações de programação que facilitem o desenvolvimento dessas aplicações [5].

1.1

Grau de Flexibilização

As dimensões de um sistema de computação são reflexo direto da quantidade de requisitos a serem atendidos. Portanto, à medida que a tecnologia permite a construção de sistemas maiores e mais robustos, maior é a probabilidade de alteração em seus requisitos e portanto maior é a necessidade de evolução desses sistemas. Entretanto, apesar de flexibilidade e desempenho serem características sempre desejáveis, elas geralmente são antagônicas. Por essa razão, muitas das técnicas de flexibilização de sistemas são aplicadas a apenas um subconjunto do sistema que é previamente classificado como mais provável de sofrer alteração. Esse conjunto de alterações previstas durante o projeto do sistema para, por exemplo, permitir inclusão de novas funcionalidades ou adequações em virtude de alterações nos requisitos do sistema, são chamadas de *alterações antecipadas*[6].

Entretanto, à medida que o tamanho dos sistemas cresce, a tarefa de prever alterações em seus requisitos e identificar pontos de flexibilização se

[1]Nesse contexto, essenciais referem-se a sistemas que devem manter-se disponíveis continuamente, por fornecerem serviços essenciais à execução de determinada tarefa.

torna mais difícil, uma vez que mais partes do sistema se tornam sujeitas a alterações. Disso surge a necessidade de utilizar técnicas que permitam adaptar sistemas em virtude de alterações não antecipadas, ou seja, alterações não previstas no projeto inicial do sistema. Além disso, é necessário que adaptações em virtude de alterações não antecipadas sejam introduzidas de forma organizada e modularizada, de forma a minimizar os impactos da alteração no sistema como um todo. Caso contrário, as alterações podem resultar numa arquitetura pouco adequada, comprometendo a eficiência do sistema e dificultando sua manutenção. Isso acarreta a necessidade de reengenharia, cujo custo pode ser muito alto.

1.2 Abstrações de Programação

Sistemas maiores também implicam um acréscimo de complexidade no seu desenvolvimento, que pode ser resultado de um maior número de requisitos ou de características não funcionais a serem atendidos, como a necessidade de distribuição em diversas unidades de processamento. Por isso, existe a necessidade de fornecer abstrações que permitam administrar essa complexidade. Em particular, essas abstrações devem permitir separar as responsabilidades e características do sistema em partes distintas, de forma a permitir identificar facilmente as partes do sistema relacionadas a determinada funcionalidade ou característica.

O paradigma de orientação a objetos define um conjunto de conceitos e abstrações que foram utilizados com certo sucesso na administração da complexidade de sistemas. Entretanto, o modelo orientado a objetos apresenta muitos problemas em relação a modularidade e evolução de software [7, 8, 9]. Em especial isso se deve a alta granularidade dos objetos, que são unidades geralmente bastante simples com poucas responsabilidades. Por isso, as responsabilidades do sistema são implementadas por diversos objetos que conjuntamente são responsáveis por uma funcionalidade do sistema. Dessa forma, quando alguma funcionalidade ou característica do sistema evolui, é necessário rastrear todas as partes envolvidas. No caso de características não funcionais como persistência, segurança ou eficiência, esse problema é ainda mais sério, pois essas características se espalham por todo o sistema.

Baseado em conceitos de arquitetura de sistemas, o modelo de componentes de software [4] tenta complementar o modelo de orientação a objetos. Componentes são unidades de construção de sistemas com funcionalidades e responsabilidades bem definidas e portanto mais complexos que objetos. Sistemas baseados em componentes são construídos juntando-se componentes que são interconectados através de *conectores* por onde as interações entre os componentes são realizadas. Os conectores definem os serviços e dependências de cada componente. As abstrações do modelo de componentes visam promover o desacoplamento e a modularização das funcionalidades do sistema em componentes bem definidos, de forma a permitir a reutilização desses componentes

na construção de outros sistemas com funcionalidades similares.

Outro modelo de programação é o modelo orientado a aspectos [10], que define técnicas e abstrações que visam promover um melhor grau de modularização, não somente com a separação das funcionalidades em componentes, como também através da modularização das características que se espalham por todo o sistema, como por exemplo sincronização, comunicação, depuração, etc. Para tanto, é definido o conceito de *aspecto*, que define uma característica do sistema, cuja implementação se espalha por todos os componentes do sistema. No modelo de programação orientada a aspectos, um aspecto é definido como um código a ser executado em diversos pontos bem definidos do sistema, denominados de *pontos de junção*. Os pontos de junção podem ser uma chamada a um método ou a leitura de um atributo. O código adicionado ao sistema através dos pontos de junção implementa uma nova característica do sistema (*i.e.*, aspecto) de forma modular e organizada. Dessa forma, é possível modificar características do sistema através da simples alteração do aspecto relacionado.

1.3 Adaptação Dinâmica

Além de maiores e mais complexos, os sistemas de computação tendem a se tornar cada vez mais essenciais, ou seja, precisam estar disponíveis por longos períodos de tempo. Como exemplo de tais sistemas, temos aplicações de missão crítica, de processamento intenso, comércio eletrônico, apoio a vida, etc. Em tais sistemas, não é possível ou desejável interromper o seu funcionamento para realizar alguma adaptação do sistema em virtude de alterações em seus requisitos ou para correção de erros. Entretanto, é inevitável que essas adaptações sejam necessárias. Dessa necessidade, surge o conceito de *adaptação dinâmica*, que consiste em alterar o comportamento do sistema enquanto ele esteja executando.

A pesquisa em adaptação dinâmica consiste basicamente em definir técnicas e modelos que permitam construir sistemas capazes de serem alterados em tempo de execução, ou seja, dinamicamente adaptáveis. Diversas técnicas têm sido propostas para permitir alterar o comportamento de sistemas em tempo de execução. Essas técnicas vão desde o uso de bibliotecas dinâmicas (*e.g.*, *Dynamic Link Library* - DLL) até o uso de linguagens interpretadas e geração dinâmica de código (*e.g.*, geração dinâmica de bytecodes Java). Entretanto, a adaptação dinâmica introduz alguns problemas em conseqüência da flexibilidade adicionada aos sistemas. Em particular, é necessário definir abstrações e modelos que permitam administrar a complexidade induzida pela possibilidade de alterar dinamicamente as aplicações.

O modelo de componentes de software define abstrações que visam administrar a complexidade dos sistemas e permitir uma rápida construção de aplicações através da composição de componentes, além de facilitar a manutenção desses sistemas devido às dependências bem definidas e o desacopla-

mento dos componentes. Essas características contribuem para que o modelo de componentes de software seja bastante adequado para a construção de sistemas dinamicamente adaptáveis. Diversos trabalhos propõem mecanismos de adaptação dinâmica baseados no modelo de componentes, como é discutido no capítulo 3.

1.4 Um Framework para Evolução Dinâmica de Aplicações

A medida que o tamanho, a complexidade e a necessidade de alta disponibilidade das aplicações aumentam, é necessário que a tecnologia de desenvolvimento de aplicações evolua adequadamente. Em especial, é necessário desenvolver aplicações dinamicamente adaptáveis, com alto grau de flexibilidade, de maneira que sejam capazes de se adaptar a alterações não esperadas. Além disso, é necessário utilizar abstrações de programação que permitam administrar a complexidade dessas aplicações e tornar seu desenvolvimento mais rápido e barato.

Este trabalho se propõe a estudar como diferentes mecanismos de adaptação dinâmica se enquadram ao modelo de componentes de software, tendo como prioridade a simplicidade de uso desses mecanismos. Para tanto, é proposto o *LuaOrb Adaptation Framework* - LOAF, que oferece ferramentas e recursos para construção de sistemas baseados em componentes dinamicamente adaptáveis. As ferramentas do LOAF são baseadas em diferentes técnicas de adaptação, permitindo diferentes graus de flexibilidade. Em particular, as ferramentas do LOAF auxiliam tanto na adaptação em virtude de alterações esperadas, através de reconfigurações, como também na adaptação em virtude de alterações não esperadas, através de alterações na implementação dos componentes da aplicação.

O LOAF é baseado no modelo de componentes de CORBA (CCM) [11] e estende esse modelo definindo recursos para construção e manipulação de componentes dinâmicos. Com base nessa extensão, é proposta uma ferramenta que permite definir alterações na estrutura e implementação de componentes. Essas alterações são definidas com uso de papéis, onde as alterações são especificadas de forma organizada, mantendo a modularização das funcionalidades do sistema. Além disso, através do uso de abstrações denominadas protocolos, é possível definir como os componentes modificados são combinados para compor as novas funcionalidades por meio de novas interações.

Além do suporte a componentes dinâmicos, o LOAF também define uma extensão da linguagem Lua para configuração dinâmica de componentes CCM. Essa extensão é apresentada como uma ferramenta que simplifica a utilização da linguagem Lua como linguagem de configuração de componentes CCM. Através dessa ferramenta, é possível definir roteiros para realizar reconfigurações em sistemas de componentes CCM através de alterações nas suas interconexões e atributos, sem a necessidade de levar em consideração todos os detalhes do modelo CCM. Essa ferramenta é proposta para auxiliar

a especificação de adaptações em virtude de alterações esperadas através de reconfigurações dinâmicas.

O LOAF se propõe a fornecer uma infra-estrutura para construção de sistemas baseados em componentes que forneça recursos de adaptação dinâmica de forma simplificada. Apesar de que a capacidade de adaptação dinâmica introduzir um aumento de complexidade no desenvolvimento, aplicações desenvolvidas através do LOAF são projetadas sem tratar esse tipo de complexidade, pois os recursos de adaptação são implementados pela infra-estrutura. Ou seja, a complexidade relacionada às alterações em tempo de execução só é tratada no momento em que haja a necessidade de alterar dinamicamente o sistema. Além disso, o LOAF simplifica a realização dessas alterações através de abstrações de programação.

1.5 Estrutura do Texto

Este trabalho está organizado da seguinte forma: o capítulo 2 apresenta uma introdução à teoria de componentes de software e ao modelo de componentes CCM, que é o modelo adotado neste trabalho; no capítulo 3 é feita uma introdução das técnicas de adaptação dinâmica; o modelo de componentes dinâmicos do LuaCCM, que compõe a infra-estrutura de adaptação do LOAF, é discutido no capítulo 4; as ferramentas do LOAF para controlar adaptações são discutidas no capítulo 5; no capítulo 6 são apresentados alguns exemplos de uso dos recursos do LOAF na construção de aplicações dinamicamente adaptáveis; no capítulo 7 são apresentadas as conclusões e as considerações finais do trabalho, juntamente com uma comparação com trabalhos relacionados e a enumeração de trabalhos futuros.