1
Introdução

Diversos fatores podem levar um sistema de software a necessitar de alterações ao longo de sua vida útil. Quando um sistema passa a ser utilizado por uma quantidade maior de usuários, freqüentemente surgem erros que passaram desapercebidos na fase de testes e demandam alterações corretivas. Com o seu maior uso, também é comum que novas necessidades sejam identificadas e traduzidas em novos requisitos para o sistema. A própria evolução natural do hardware pode exigir alterações arquiteturais nas aplicações instaladas para que estas contemplem novas configurações.

Os processos envolvidos na manutenção de sistemas em produção normalmente são complexos e exigem que estes sistemas sejam interrompidos por algum período de tempo, que pode variar de acordo com a abrangência da alteração e com a tecnologia utilizada. Esse período de indisponibilidade pode ser inaceitável em aplicações que exijam um alto grau de disponibilidade, como aplicações de missão critica, de processamento intenso, comercio eletrônico, entre outras [1].

Com a popularização de sistemas de comércio eletrônico e a grande quantidade de aplicações científicas fazendo uso de grades computacionais, a demanda por mecanismos que permitam efetuar alterações sem tirar as aplicações do ar tende a crescer cada vez mais. Por outro lado, o desenvolvimento de mecanismos deste tipo é complexo e foge do escopo da maioria das aplicações. Acreditamos que o ideal seria fornecer essa facilidade sob a forma de serviços providos pela camada de *middleware*.

Diversos trabalhos têm sido desenvolvidos pelo nosso grupo de pesquisa no sentido de definir mecanismos e abstrações que ofereçam um melhor suporte a alterações dinâmicas de aplicações, aliando técnicas adequadas de modularização, que possibilitam gerenciar a complexidade das alterações, a mecanismos de adaptação dinâmica, que permitem efetuar essas alterações

sem tirar as aplicações do ar.

Um dos primeiros esforços nessa direção foi o desenvolvimento do sistema LuaOrb. LuaOrb é uma infra-estrutura de desenvolvimento de software cujo núcleo é composto por *bindings* entre a linguagem interpretada Lua [2] e diferentes sistemas de middleware. Atualmente, LuaOrb conta com *bindings* para CORBA, COM, Java e .NET. Nesse projeto, a linguagem Lua desempenha o papel de uma linguagem de composição unificadora, que permite tanto a conexão dinâmica de componentes quanto a interoperabilidade entre componentes de diferentes middlewares. O mecanismo de composição oferecido por LuaOrb realiza em tempo de execução as tarefas de conexão, adaptação, implementação e verificação de novos tipos de componentes, e trata de uma maneira uniforme componentes de diferentes middlewares, permitindo que esses componentes sejam conectados de forma transparente [3]. O núcleo do sistema LuaOrb, assim como o modelo de composição que o fundamenta estão descritos em [4].

Baseado no LuaOrb, alguns mecanismos de mais alto nível para dar suporte a adaptação dinâmica foram investigados, tais como a extensão dinâmica de servidores CORBA [5], *smart proxies*, um mecanismo extensível para monitoração distribuída [6], e um contâiner dinamicamente adaptável para o Modelo de Componentes CORBA (CCM) [7]. Também já avaliamos [7, 8] o uso das abstrações de *papéis e protocolos* [9] para adaptar aplicações baseadas em componentes CCM.

Para avaliar as ferramentas e técnicas investigadas, alguns experimentos em diferentes domínios de aplicação foram desenvolvidos, tais como sistemas de gerenciamento de redes [10], CAD colaborativo [11, 12], visualização distribuída [13], computação ubíqua [14, 15, 16, 17, 18], e computação em grade [19, 20].

Todos esses estudos, porém, têm sido baseados na linguagem Lua [21], explorando intensivamente os recursos tipicamente oferecidos por uma linguagem interpretada. Seria interessante contar com recursos deste tipo também em outras linguagens, para que desenvolvedores com esse tipo de necessidade possam escolher a linguagem mais adequada ao domínio da aplicação a ser desenvolvida.

Neste trabalho, avaliamos a aplicação de algumas das soluções identificadas por nosso grupo em um sistema de componentes desenvolvido em Java [22], comparando os resultados obtidos com os de um sistema similar implementado em Lua. A linguagem Java foi escolhida por ser uma linguagem orientada a

objeto amplamente adotada, especialmente no desenvolvimento de aplicações para a Internet, com características bastante diferentes de Lua: enquanto Java é uma linguagem estática e fortemente tipada, Lua é uma linguagem interpretada, sem declarações de tipo e tipada dinamicamente.

1.1 Objetivos

O objetivo principal deste trabalho é avaliar a viabilidade da implementação em Java de um conjunto de mecanismos de adaptação dinâmica desenvolvidos previamente em outros trabalhos de nosso grupo com a linguagem Lua. Para tanto, desenvolvemos uma implementação de um sistema de componentes existente que ofereceu um nível de flexibilidade adequado para a implementação dos mecanismos de adaptação identificados. Quando este trabalho foi iniciado, não estava claro que essa implementação seria possível, pois o sistema estático de tipos de Java impõe uma série de restrições que poderiam impedir ou dificultar a solução desenvolvida de oferecer o mesmo grau de flexibilidade oferecido pelos mecanismos implementados em Lua. Os mecanismos de adaptação originais sofreram algumas alterações para que se tornassem compatíveis com a linguagem Java e com o sistema de componentes escolhido, além de incorporar algumas melhorias.

Outro objetivo importante do trabalho é a realização de um estudo para avaliar a utilização das linguagens Lua e Java na implementação de mecanismos de adaptação e na produção de código adaptativo, comparando a solução desenvolvida com outra equivalente implementada em Lua. O primeiro aspecto estudado foram as dificuldades encontradas no processo de desenvolvimento e as possíveis limitações impostas pelo uso da linguagem Java. Em seguida, analisamos a flexibilidade dos mecanismos oferecidos para o desenvolvedor, comparando-os com os oferecidos em Lua. Finalmente analisamos o desempenho das soluções, incluindo na comparação aplicações desenvolvidas sem os mecanismos de adaptação, para mensurar a sobrecarga imposta por eles.

1.2 Contribuições

A principal contribuição deste trabalho é a implementação de um sistema de componentes escrito em Java que permita a desenvolvedores criar aplicações dinamicamente adaptáveis de maneira simples e fornecer mecanismos flexíveis para que eles efetuem essas adaptações. Em um primeiro momento, criamos uma implementação do sistema de componentes SCS em Java baseada em *esqueletos dinâmicos*, que ofereceu um grau de flexibilidade adequado para que ocorra a adaptação dinâmica. Em seguida, os mecanismos de adaptação foram implementados, dando origem à versão adaptável do sistema SCS.

Ao comparar a solução desenvolvida em Java com uma similar desenvolvida em Lua, pretendemos oferecer argumentos para auxiliar desenvolvedores de aplicações com requisitos de adaptação dinâmica na tarefa de selecionar uma linguagem de programação adequada às suas necessidades. Os dois principais fatores que um programador deve levar em consideração neste caso são a flexibilidade oferecida pela linguagem, que deve permitir efetuar as alterações necessárias de maneira simples porém eficaz, e o impacto do uso dos recursos de adaptação sobre o desempenho das aplicações. Realizamos estudos comparativos avaliando as soluções sobre o ponto de vista desses dois aspectos, de forma a identificar os pontos positivos e negativos das linguagens.

1.3 Estrutura do Texto

No capítulo 2, examinamos alguns trabalhos relacionados que abordam de alguma forma o tema tratado nesta dissertação. Analisamos inicialmente quatro trabalhos que apresentam abordagens diferentes para a questão da adaptação dinâmica, entre eles os sistemas de *middleware DynamicTAO* (seção 2.1), *OpenCOM* (seção 2.2) e *Comet* (seção 2.3) e o gerador de programas adaptáveis *TRAP/J* (seção 2.4). Em seguida, analisamos mais profundamente o sistema LuaCCM [23] (seção 2.5), que serviu de referência para a solução descrita neste trabalho, apresentando o modelo de componentes CCM [24] (2.5.1), a partir do qual ele foi desenvolvido, e os mecanismos de adaptação dinâmica e abstrações propostos por ele (2.5.2).

No terceiro capítulo, descrevemos o sistema de componentes SCS [25], as-

sim como sua implementação em Java, o SCS Java, que foram utilizados como base para o desenvolvimento deste trabalho. Em seguida, no quarto capítulo, apresentamos a versão do SCS com *binding* dinâmico, desenvolvida para fornecer um nível de indireção adequado para a implementação dos mecanismos de adaptação propostos, apresentados no quinto capítulo. Finalmente, apresentamos os resultados dos experimentos realizados com a solução desenvolvida no capítulo 6, e as conclusões e trabalhos futuros propostos no capítulo 7.