7
Conclusões

O objetivo inicial deste trabalho foi avaliar a viabilidade da implementação de mecanismos de adaptação dinâmica, como os implementados pelo *LuaCCM*, utilizando a linguagem Java. Para tanto criamos uma implementação em Java do sistema de componentes SCS baseada em *esqueletos dinâmicos*, que forneceu um nível de indireção adequado para a criação desses mecanismos. Apesar da necessidade de alguns ajustes nos recursos de adaptação do *LuaCCM*, devido a particularidades da linguagem Java e do sistema SCS, conseguimos implementar satisfatoriamente todas as funcionalidades propostas.

A avaliação da solução desenvolvida foi divida em três partes de acordo com as referências utilizadas para a comparação. Inicialmente analisamos o mecanismo de *binding* dinâmico implementado, tendo como referências a versão convencional do SCS Java e o sistema *LuaORB*, de forma a avaliar a sua flexibilidade e facilidade de uso, além de mensurar o impacto que ele causa no desempenho das aplicações. De forma geral, podemos concluir que esse mecanismo simplifica o processo de desenvolvimento das aplicações e contribui com a flexibilidade do sistema ao promover o desacoplamento entre a implementação e a estrutura de seus componentes. A sobrecarga no desempenho se mostrou tolerável e pouco representativa, especialmente no caso de aplicações distribuídas que já são submetidas à sobrecarga dos mecanismos de comunicação.

Na segunda parte da avaliação, examinamos o contêiner dinâmico onde estão implementados os mecanismos de adaptação dinâmica, tendo como principal referência o *LuaCCM*, que definiu e implementou originalmente esses mecanismos. Para comparar as duas soluções, buscamos implementar com o SCS adaptável os mesmos exemplos de uso utilizados pelo *LuaCCM* para demonstrar suas funcionalidades. Essa comparação nos mostrou que não existem limitações na linguagem Java que impeçam a implementação de nenhum dos mecanismos de adaptação propostos, mas a rigidez do mecanismo

de verificação de tipos dessa linguagem por diversas vezes dificulta bastante o processo de adaptação, prejudicando a facilidade de uso da solução. Por outro lado, é uma expectativa comum que a agilidade proporcionada pela linguagem Lua tenda a tornar o processo de aplicação de adaptações complexas mais propenso a erros, uma vez que o código adaptativo não é compilado e que não existem restrições que induzam a uma modularização adequada desse código.

Por fim, avaliamos o uso das linguagens Lua e Java para a composição do código adaptativo, ambas suportadas pela solução desenvolvida. Os resultados da comparação nos mostraram que a execução das funcionalidades adaptadas com a linguagem Lua apresentaram um desempenho inferior ao das adaptadas com a linguagem Java, devido à sobrecarga introduzida pelo *binding* entre as duas linguagens e à natureza interpretada da linguagem Lua. O tempo de aplicação de adaptações fornecidas como código Lua foi similar ao de adaptações fornecidas como *bytecode* Java e inferior ao das fornecidas como código Java. Em relação à facilidade de uso e flexibilidade, observamos que a linguagem Lua torna mais simples o desenvolvimento de código adaptativo pois conta com mecanismos mais flexíveis de verificação de tipos.

Pudemos constatar ao longo do desenvolvimento do trabalho que a verificação estática de tipos da linguagem Java foi o maior obstáculo para a implementação dos mecanismos de adaptação dinâmica. Boa parte das funcionalidades oferecidas por esses mecanismos envolve alterações nas interfaces dos componentes e a verificação estática impede que tipos existentes sejam modificados em tempo de execução para refletir essas alterações. Para contornar essa limitação, foi necessária a introdução de um nível de indireção entre a definição do componente e sua implementação, de forma que a ligação entre os componentes fosse efetuada de maneira dinâmica e pudesse ser manipulada em tempo de execução. A utilização de uma linguagem dinamicamente tipada e com declarações fracas de tipo simplificaria a implementação de tais mecanismos, pois como os tipos podem ser alterados livremente, não seria necessário implementar a ligação dinâmica entre os componentes.

A decisão de adotar um mecanismo de verificação baseado na compatibilidade estrutural de tipos foi fundamental para aumentar a flexibilidade da solução. Esse tipo de verificação permite que se utilize na implementação dos componentes objetos que não foram necessariamente projetados para esse fim, bastando que haja compatibilidade entre os métodos disponibilizados pelo objeto e os requeridos pelo componente. Desta forma, caso haja compatibilidade, é possível utilizar classes legadas sem nenhuma modificação para implementar um componente. Como nossa verificação de compatibilidade estrutural

só ocorre no momento em que o componente recebe uma chamada e se restringe à operação invocada, é possível ainda utilizar implementações parciais do componente, recurso valioso para a execução de testes e desenvolvimento de protótipos.

7.1 Trabalhos Futuros

No decorrer do desenvolvimento deste trabalho identificamos uma série de pontos de melhoria e extensão, que não foram implementados por fugir do escopo definido. Inicialmente, uma das limitações que nos deparamos foi a ausência do suporte à comunicação baseada em eventos por parte do SCS. O sistema SCS se destaca pela simplicidade do seu modelo de componentes aliada à flexibilidade proporcionada por seu modelo de execução, porém acreditamos que a sua especificação precisa evoluir para acompanhar os demais sistemas de componentes existentes, como o CCM.

A medida que a infra-estrutura das aplicações e o próprio *hardware* sobre o qual elas são instaladas evolui, a sobrecarga de desempenho causada pelos mecanismos de adaptação dinâmica tende a se tornar insignificante. Porém, antes que esses mecanismos possam ser incorporados de maneira definitiva ao processo de desenvolvimento de aplicações, uma série de questões ainda precisam ser endereçadas. Uma das principais preocupações diz respeito à segurança desses mecanismos, uma vez que o uso mal intencionado deles compromete completamente as aplicações. Desta forma se faz necessário o desenvolvimento ou integração de mecanismos que garantam aos recursos de adaptação dinâmica requisitos básicos de segurança da informação como controle de acesso, integridade, autenticidade e disponibilidade. Além da questão da segurança, para que os mecanismos de adaptação possam ser utilizados em sistemas distribuídos de maneira confiável, é necessária a implementação de mecanismos que garantam a consistência do sistema durante a aplicação das adaptações. Antes de se aplicar uma adaptação, é necessário verificar se o sistema está apto a recebê-la, checando se não existem requisições em andamento que serão afetadas pela adaptação. Além disso é necessário tratar as requisições recebidas durante a aplicação da adaptação, redirecionado elas para um componente que possa processá-las, enfileirando elas até que o componente adaptado esteja apto a respondê-las ou ainda sinalizando um erro de forma adequada.

Outro ponto que consideramos bastante relevante é a estruturação das

adaptações aplicadas. Como vimos nos exemplos do capítulo 6, adaptações em uma aplicação freqüentemente envolvem adaptações pontuais em vários de seus componentes. Na solução desenvolvida neste trabalho, as adaptações são aplicadas de maneira *ad hoc* e não existem formas de relacionar as operações que compõem uma adaptação maior. Além disso, caso uma adaptação introduza erros no sistema, não existem mecanismos que permitam restaurar o seu estado anterior. O sistema Comet [9] propôs abstrações, implementadas posteriormente também pelo *LuaCCM*, que permitem agrupar as adaptações que devem ser aplicadas em um componente para que ele desempenhe um novo papel e aplicar esses grupos de adaptações de maneira sistemática. Acreditamos que com a ajuda de uma linguagem declarativa que permita descrever de maneira mais fácil abstrações como essas, seja possível implementar no SCS Java adaptável mecanismos semelhantes.

Seria extremamente útil oferecer também um mecanismo similar a um sistema de versionamento, que permitisse inspecionar em tempo de execução as adaptações aplicadas a um componente e restaurar versões passadas dele, caso algum erro tenha sido introduzido em alguma das adaptações aplicadas. Um desafio introduzido por esse tipo de mecanismo é a necessidade de absorver o código fonte das adaptações aplicadas em tempo de execução na base de código da aplicação. Nos casos em que uma adaptação introduz uma nova funcionalidade ou substitui uma existente, a incorporação do código fonte não traz maiores problemas, pois basta criar um novo artefato ou adicionar uma nova versão de um artefato existente à base. Versionar interceptadores é um processo mais complexo, pois é necessário armazenar informações sobre o período em que ele foi aplicado e sobre sua relação com os artefatos interceptados e demais interceptadores. Outro problema que merece atenção é a gerência das versões da aplicação geradas a partir das adaptações aplicadas, que devem agregar adaptações relacionadas e diferenciar alterações temporárias de definitivas.

Há ainda alguns pontos que podem ser melhorados no SCS Java adaptável de forma a simplificar o seu uso e incrementar sua flexibilidade. Em primeiro lugar, acreditamos que é possível eliminar a necessidade de que as implementações de facetas dos componentes declarem um construtor que receba como parâmetro a sua API interna, ganhando com isso a possibilidade de reutilizar classes legadas sem precisar alterá-las. Para isto, podemos implementar um mecanismo de fábrica de instâncias que permita injetar essa dependência na implementação das facetas de forma ajustável. Para uma classe que receba a referência de um componente externo como parâmetro de uma operação, por

exemplo, podemos implementar um fábrica de instâncias que funcione como um adaptador, obtendo a referência do componente externo pela API interna do componente e passando ela para classe legada através da operação adequada. Outro recurso interessante seria a eliminação da necessidade de geração de *stubs* para os clientes dos componentes, que pode ser alcançada com a introdução de um elemento no cliente que intercepte e mapeie as mensagens trocadas com os componentes, de maneira análoga ao mecanismo de *esqueletos dinâmicos* implementado. Por fim, um mecanismo do *LuaCCM* que não foi implementado mas seria de imensa utilidade é o que permite aplicar adaptações em um tipo de componente, replicando estas adaptações a todas as suas instâncias.

No momento da conclusão deste trabalho, uma implementação em Lua do sistema SCS com suporte a adaptação dinâmica de componentes estava em desenvolvimento em nosso grupo de estudo. Pretendemos oferecer interoperabilidade entre essa implementação e a desenvolvida neste trabalho de forma que adaptações descritas em Java e em Lua possam ser aplicadas tanto em componentes desenvolvidos em Java quanto nos desenvolvidos em Lua.