1
Introdução

Há vários anos, vem crescendo a necessidade de desenvolvimento de programas distribuídos. Em resposta a essa necessidade, diversas tecnologias como CORBA (*Common Object Request Broker Architecture*) [1], COM [2] e JavaBeans [3] surgiram.

Essas tecnologias permitem a aplicações reagirem dinamicamente a mudanças no ambiente em que executam. Em um cenário típico, um cliente poderia procurar servidores que implementem o serviço de que precisa em tempo de execução. Caso o servidor encontrado venha a falhar, uma nova busca pode ser feita, o que constituiria uma implementação simples de um mecanismo de tolerância a falhas. Tais alterações de comportamento em tempo de execução são aqui chamadas de *adaptação dinâmica*.

Outra situação que pode requerer adaptação dinâmica é a escassez de recursos, como largura de banda, CPU, etc. Isso é particularmente verdade em enlaces sem fio, onde problemas como interferência, migração de estações e desconexões são constantes.

De especial interesse para esta dissertação, CORBA define uma arquitetura aberta e amplamente difundida, que oferece suporte à programação em ambientes heterogêneos de maneira transparente. Dentre os diversos serviços CORBA existentes, destacam-se, no suporte à adaptação dinâmica, o repositório de interfaces e o serviço de *trading*.

O repositório de interfaces é responsável por armazenar, de maneira centralizada e organizada, as informações das interfaces em uso. Essas interfaces são especificadas em IDL (*Interface Description Language*) — uma linguagem definida pela OMG que faz parte do padrão CORBA. Outra função do repositório de interfaces é o de permitir, em tempo de execução, a inspeção das assinaturas dos métodos das interfaces, atributos, listagem dos módulos, etc.

O serviço de *trading*, por sua vez, pode ser visto como um repositório de ofertas de serviço. Essas ofertas consistem em uma referência para os objetos que as implementam, e um conjunto conhecido de propriedades. Desse modo, pode-se anunciar novas ofertas de serviço, passando como argumentos a referência à implementação do serviço e suas propriedades. Analogamente, um cliente interessado em buscar ofertas de serviço pode fazê-lo especificando os devidos valores para as propriedades de interesse.

Embora seja popular, a arquitetura CORBA requer investimento considerável em esforços de aprendizado e em desenvolvimento de aplicações. Vários trabalhos têm sido propostos com o objetivo de amenizar esse inconveniente. Alguns, como o LuaRep [4] e o LuaTrading [5], oferecem acesso facilitado ao repositório de interfaces e ao *trader*[1], respectivamente. Outros, como o LuaMonitor [6] implementam recursos úteis ao desenvolvimento de aplicações.

Além dos monitores, descritos no próximo capítulo, o LuaMonitor apresenta uma proposta de *proxy inteligente*. Esse *proxy* é capaz de se adaptar automaticamente a mudanças de propriedades relevantes ao usuário, mas até então essa idéia não havia sido totalmente explorada.

Como continuação dessa série de trabalhos, esta dissertação se aprofundou mais na idéia do *proxy inteligente*, chamado aqui de *LuaProxy*. Como recursos adicionais, pode ser especificado o nível desejado para cada recurso, fornecer código de adaptação personalizado, etc. Além disso, cliente e servidor podem usufruir da agregação de *stubs* em tempo de execução. Esses *stubs* oferecem uma implementação alternativa para a chamada dos métodos remotos, e podem ser empregados com o objetivo de explorar as particularidades de um determinado serviço, no sentido de prover, por exemplo, mais eficiência ou segurança nas chamadas de métodos.

O restante da dissertação está organizado da seguinte maneira: o Capítulo 2 descreve as ferramentas nas quais este trabalho se baseou. O Capítulo 3 descreve a arquitetura do proxy inteligente; o mecanismo de *upload* e *download* de *stubs* é discutido no Capítulo 4. A seguir, o Capítulo 5 mostra uma aplicação-exemplo e os resultados obtidos. Finalmente, o Capítulo 6 conclui a dissertação e discute algumas possibilidades de trabalhos futuros.

[1] Um *trader* é um processo que implementa o serviço de *trading*.