1
Introdução

Middleware é um termo cunhado no final da década de 60 (Naur e Randell, 1968), que é freqüentemente empregado para designar uma camada de software que oferece uma infra-estrutura para construção de aplicações mais adequada que a oferecida pelo software básico de uma plataforma. Originalmente, o middleware visava implementar correções ou ajustes no software básico necessários para a construção de certas aplicações. Atualmente, o middleware desempenha um papel mais amplo, facilitando a construção das aplicações de diversas formas, seja oferecendo um conjunto de funcionalidades necessárias, ou ferramentas de auxílio à programação, ou um ambiente para implantação, execução e manutenção das aplicações. Tipicamente, um middleware oculta detalhes e diferenças entre versões ou implementações do software básico, facilitando a portabilidade e a manutenção das aplicações.

No seu sentido mais amplo, o conceito de middleware confunde-se com outras infra-estruturas de software, como arcabouços, ambientes de desenvolvimento, servidores de execução de aplicações, entre outras. Neste trabalho, trataremos por middleware uma infra-estrutura de software que integre um conjunto de funcionalidades, ferramentas ou serviços sob um modelo de programação ajustado para um tipo específico de aplicação. O modelo de programação refere-se à forma com que o middleware assume ou impõe que a aplicação seja implementada, que pode se resumir ao uso de um dado paradigma de programação (*e.g.* orientado a objetos (Object Management Group, 2002), troca de mensagens (Carzaniga et al., 2000), declarativo (Montanari et al., 2004; Soares et al., 2007)), padrão arquitetural (Schmidt et al., 2000), ou mesmo uma linguagem de programação específica (Grosso, 2001; Jong, 2007). A seção 2.1 apresenta uma discussão mais detalhada sobre a caracterização de middleware adotada neste trabalho.

Um desafio no projeto de middleware é a definição dos recursos e modelo de programação oferecidos para a aplicação, visto que os recursos necessários podem variar consideravelmente, mesmo para aplicações similares. Além disso, a evolução natural da aplicação pode alterar suas necessidades e, conseqüentemente, demandar novos recursos do middleware ou mesmo um modelo de

programação diferente. Como solução, alguns sistemas de middleware incorporaram uma variada gama de funcionalidades e recursos, mesmo que sejam desnecessários em algumas de suas aplicações. Um exemplo disso é o padrão CORBA para middleware de distribuição, que integra uma ampla variedade de recursos (Henning, 2006). Uma desvantagem dessa abordagem é a complexidade resultante do middleware, que se reflete tanto na sua implementação quanto no modelo de programação oferecido, tornando o middleware difícil de ser implementado, adaptado ou mesmo utilizado. Além disso, a inclusão de um conjunto amplo de recursos não garante que outros não sejam necessários em novas aplicações ou na evolução de aplicações existentes.

Por algum tempo, o foco da implementação de middleware foi o seu desempenho (Vinoski, 2003). Implementações de middleware eram escritas em linguagens de alto desempenho como C++. Mais recentemente, esse foco vem sendo desviado para implementações que incorporem facilidades de alteração e extensão. Um exemplo disso são os sistemas de middleware reflexívos (Kon et al., 2002), que são aqueles que oferecem mecanismos através dos quais a aplicação pode inspecionar e alterar seu comportamento durante sua execução. Questões como portabilidade, possibilidade de configuração, extensibilidade e evolução dinâmica ganharam maior importância.

Como um reflexo disso, linguagens mais dinâmicas como Java passaram a ser utilizadas mais freqüentemente na implementação de middleware (Brose, 1997; Pellerin et al., 2005; Group, 2007), apesar do seu uso geralmente implicar em um desempenho inferior da aplicação. Recursos oferecidos por Java, como as facilidades para carga dinâmica de código compilado, facilitam a extensão da implementação do middleware através da inclusão de novos recursos ou substituição do comportamento pré-definido. O suporte parcial para reflexão computacional oferecido por Java também facilita que a implementação do middleware realize verificações dinâmicas de tipos sobre seus componentes internos ou oferecidos pela aplicação, permitindo mensagens de erro mais claras ou mesmo que o middleware possa identificar e contornar eventuais problemas de configuração.

Algumas das primeiras linguagens dinâmicas utilizadas no desenvolvimento de software foram criadas na década de 60 com o intuito de facilitar a utilização de sistemas de computação por pessoas com pouca ou nenhuma experiência em programação de computadores. Exemplos dessas linguagens incluem MUMPS (Greenes et al., 1969), desenvolvida para criação de software de administração hospitalar, e BASIC (BASIC, 1964), criada para auxiliar o ensino de programação para pessoas sem formação em áreas de tecnologia. Com a evolução do software de computadores pessoais, a habilidade de programação

por parte dos usuários dos sistemas de computação se tornou menos necessária e a popularidade desse tipo de linguagem diminuiu. Mais recentemente, a popularidade de linguagens dinâmicas ressurgiu com o crescimento de aplicações para Web. A linguagem Java (Gosling e McGilton, 1996) ganhou popularidade devido a funcionalidades relacionadas às suas características dinâmicas, como carga dinâmica de código e maior portabilidade devido ao seu modelo de execução baseado em máquinas virtuais. Posteriormente, novas linguagens dinâmicas surgiram e algumas ganharam popularidade tanto em aplicações Web (Vinoski, 2002; Garrett, 2005) como em outros domínios (Stothard, 2000). As razões para o sucesso dessas linguagens nesses diferentes usos são variadas, mas alguns de seus defensores clamam que elas sejam mais produtivas, expressivas e flexíveis (Hirschi, 2007; Yegge, 2009). Além disso, o sucesso do uso de linguagens dinâmicas no desenvolvimento de sistemas distribuídos na Web sugere sua adequação para sistemas distribuídos em geral. A seção 2.2 apresenta uma discussão mais detalhada sobre a caracterização de linguagens dinâmicas neste trabalho.

Apesar dessa popularização de linguagens dinâmicas no desenvolvimento de aplicações distribuídas na Web e o maior interesse em sistemas de middleware flexíveis, utilização dessas linguagens no desenvolvimento de middleware tem sido investigada por poucos grupos. Ao contrário disso, as linguagens dinâmicas são comumente desconsideradas nesse tipo de aplicação por serem consideradas ineficientes (Geelan, 2007) ou muito propensas a erro devido à tipagem dinâmica. Contudo, poucos trabalhos tem sido realizados no sentido de validar essas considerações no contexto específico de middleware. Este trabalho visa suprir essa deficiência, através de um estudo de caso prático, que consiste na análise detalhada da implementação de um middleware escrito numa linguagem dinâmica. Este estudo foi realizado em três etapas. Inicialmente, desenvolvemos o OiL (Maia et al., 2006), um middleware de distribuição escrito inteiramente na linguagem dinâmica Lua (Ierusalimschy, 2006). No desenvolvimento do OiL, são tratados aspectos como desenvolvimento orientado a objetos e baseado em componentes em Lua, uso do recurso de co-rotinas de Lua para implementar suporte a *multithreading* e o uso do suporte reflexão computacional de Lua para implementar mecanismos de interceptação na implementação do middleware. O foco do desenvolvimento do OiL é utilizar os recursos de Lua para prover uma implementação robusta que seja simples, porém flexível, ou seja, fácil de ser adaptada a diferentes usos. O principal propósito do OiL neste trabalho é servir de base para as análises apresentadas, que visam identificar as características específicas de Lua que influenciam a implementação de um middleware como o OiL, em particular, as características

que a destacam como linguagem dinâmica.

Numa segunda etapa deste estudo consiste na definição da forma de avaliação da implementação do middleware, em particular em relação a sua flexibilidade. Para tanto, propomos o uso de Dimensões Cognitivas de Notações (CDN, do inglês *Cognitive Dimensions of Notations*). O CDN é um arcabouço de avaliação proposto originalmente para avaliar a facilidade de uso de notações e linguagens na produção de artefatos (Green, 1989). Atualmente, ele é utilizado também na avaliação da facilidade de uso de linguagens de programação (Clarke, 2001), implementações (Clarke e Becker, 2003), interfaces de programação de bibliotecas e *frameworks* (Clarke, 2006), etc. Neste trabalho, o CDN é adotado como critério de avaliação de flexibilidade, que é vista como a facilidade de modificar o software.

Na última etapa do estudo, é feita uma análise da implementação do OiL através da comparação com a implementação de um middleware similar escrito em C++, o Mico (Puder et al., 2006). Para tanto analisamos como diferentes recursos são implementados nesses dois sistemas de middleware considerados, identificando facilidades e dificuldades na realização dessas implementações, de acordo com os critérios de facilidade definidos pelo CDN. O objetivo disso é identificar diferenças importantes nas implementações que influenciam sua flexibilidade, em especial, as diferenças na arquitetura dessas implementações e suas origens nos recursos de suas respectivas linguagens de programação. Para ilustrar o impacto no desempenho devido ao uso de uma linguagem dinâmica, são realizados testes elaborados com o intuito de avaliar alguns aspectos relacionados ao tempo de resposta e utilização de memória do middleware. Com isso, pretendemos identificar e ilustrar algumas das vantagens e problemas reais do uso de linguagens dinâmicas na implementação do middleware.

As contribuições deste trabalho podem ser resumidas a (1) uma implementação robusta de ORB (*Object Request Broker*) escrita inteiramente na linguagem dinâmica Lua, cuja arquitetura, principais recursos e um breve relato do seu desenvolvimento são apresentados neste texto; (2) identificação da influência das características de linguagens dinâmicas na flexibilidade e no desempenho dessa implementação, através da comparação com outras implementações similares em linguagens menos dinâmicas; e (3) interpretação dos critérios de facilidade de uso definidos pelo CDN para serem utilizados como critério de avaliação de flexibilidade de um middleware.

1.1 Estrutura do Texto

O texto desta tese está organizado da seguinte forma. Inicialmente, o capítulo 2 apresenta algumas definições e conceitos básicos usados neste trabalho, em particular, a interpretação do termo middleware, a definição de linguagens dinâmicas e a caracterização de Lua como tal, além de uma conceitualização de flexibilidade de software. Esse capítulo também apresenta as dimensões cognitivas que compõem o CDN, assim como a influência de cada uma delas na flexibilidade do software e como elas são influenciadas pelas linguagens dinâmicas. O capítulo 3 apresenta o OiL, seus principais recursos e a organização da sua implementação. No capítulo 4, é apresentada a avaliação de flexibilidade do OiL com base no CDN através de comparações com o Mico, cuja implementação é apresentada nesse capítulo. No capítulo 5, é apresentada a avaliação de desempenho do OiL comparativamente a outros sistemas similares implementados em linguagens menos dinâmicas como Java e C++. O capítulo 6 apresenta alguns trabalhos relacionados relevantes. Finalmente, o capítulo 7 apresenta as conclusões da tese.

1.2 Convenções Tipográficas

O texto dessa tese segue as seguintes convenções tipográficas:

- As definições de termos ou expressões, com significado especial no texto, inclusive as utilizadas sem tradução para o português, são escritas em *itálico*.
- Texto de saída de programas e identificadores de elementos em trechos de código apresentados no texto ou que fazem parte da definição das linguagens de programação citadas no texto são escritos usando `caracteres de espaçamento fixo`.