7
Conclusão

Uma das principais motivações para este trabalho é a falta de estudos detalhados sobre o uso de linguagens dinâmicas no contexto do desenvolvimento de middleware, que resulta em uma maior relutância por parte dos desenvolvedores de middleware na adoção dessas linguagens em seus projetos. Este trabalho visa suprir dessa deficiência através de duas contribuições principais:

Projeto implementação e avaliação de um middleware em uma linguagem dinâmica que oferece bons níveis de robustez, desempenho e flexibilidade. O OiL foi projetado e desenvolvido para utilização efetiva no desenvolvimento de aplicações, ao contrário de outras implementações de middleware em linguagens dinâmicas que foram desenvolvidas para utilização em experimentos de pesquisa ou como protótipo de outros sistemas, sem preocupação com qualidade. O propósito disso é fornecer uma implementação que possa ser utilizada tanto em aplicações que dependam de um middleware confiável, como também servir de base para experimentos que permitam avaliar questões como desempenho e corretude de implementação.

Identificação de características relevantes de linguagens dinâmicas para implementação de middleware. A experiência de desenvolvimento do OiL e a análise de sua implementação fornecem dados importante para identificar as características e funcionalidades mais marcantes de Lua na implementação do middleware. Em particular, podemos destacar como características positivas a tipagem dinâmica, reflexão computacional e a flexibilidade das tabelas de Lua para construção de estruturas dinâmicas. O propósito dessa identificação é demonstrar algumas vantagens da adoção dessas linguagens nesse contexto.

Uma outra contribuição deste trabalho é a utilização do arcabouço CDN como ferramenta para avaliação de flexibilidade de software, que se mostrou bastante eficaz nesse aspecto, primeiramente por permitir explicitar algumas das dificuldades encontradas pelos alunos que modificaram o OiL e tornar evidente outros problemas mais sutis como a dificuldade causada pela barreira

de abstração introduzida pelo modelo de componentes ou a pouca de expressividade do uso de tabelas em alguns pontos da implementação. Em segundo lugar, o CDN também permitiu identificar mais especificamente os aspectos da implementação que contribuem para melhorar sua flexibilidade, como os diferentes mecanismos de provisão oferecidos por Lua e a arquitetura do OiL e a facilidade para avaliações progressivas, que são ambos particularmente interessantes para experimentação e pesquisa na área de middleware.

7.1 Utilização do OiL

Como implementação, o OiL tem se mostrado bastante efetivo, sendo utilizado com sucesso em diferentes cenários, tanto em aplicações como também de base para outros projetos de middleware. Desde os seus primeiros protótipos, o OiL tem sido utilizado no desenvolvimento de aplicações. Isso contribuiu positivamente para a qualidade da implementação atual do OiL. A maior parte de seus usos em aplicações é motivada pela sua simplicidade ou facilidade de uso.

O primeiro uso do OiL foi na implementação do InteGrade (Goldchleger et al., 2004), um middleware para computação em grades que utiliza o tempo ocioso dos computadores de uma rede. Uma das principais características do InteGrade é permitir compor grades computacionais formadas por computadores que não são destinados exclusivamente para isso, como por exemplo estações de trabalho de uma instituição. O OiL é utilizado na implementação do processo LRM (*Local Resource Manager*), que executa em cada computador da grade e é responsável por controlar aquele nó da grade. Idealmente, o LRM deve utilizar poucos recursos da máquina, em especial memória, implicando em uma sobrecarga mínima no desempenho das máquinas quando estão sendo utilizadas na sua atividade principal. Através da interface C de Lua, o OiL foi utilizado para implementar uma biblioteca C++ que serve de base para a implementação do LRM, que faz chamadas CORBA. Essa biblioteca é utilizada em substituição a uma implementação de ORB CORBA em C++.

Uma outra aplicação similar, que faz uso do OiL, é o CSBase (Lima et al., 2006), um arcabouço para construção de aplicações em *clusters* heterogêneos. O CSBase foi originalmente desenvolvido para construção de aplicações da Petrobras S/A através de uma parceria com o laboratório Tecgraf da PUC-Rio. De forma similar ao cenário anterior, o OiL é utilizado na implementação do processo de controle de nós do *cluster*, contudo, neste caso, ele é escrito em Lua. A motivação principal para o uso do OiL nesse caso é sua portabilidade, pois os *clusters* onde as aplicações do CSBase executam são compostos por

centenas de nós com hardware variado, executando diferentes sabores de Unix (AIX, IRIX, Solaris, Linux, etc.) e Windows. Adicionalmente, o software de muitos desses nós é desatualizado. Como resultado disso, o custo de compilação de um ORB em C++ ou a implantação e manutenção do ambiente de execução de Java nessas diferentes plataformas é muito alto em relação à compilação de Lua e OiL. A principal razão disso é o fato de Lua ser escrita em um subconjunto bastante portável de C e o OiL utilizar uma API simples de acesso a sockets implementada sobre a API de sockets BSD, mas que também pode ser implementada em outras plataformas, como OpenVMS e BREW.

O OiL também é utilizado no MPA (Quaresma et al., 2008), um ambiente de desenvolvimento e implantação de aplicações de controle de processos industriais que é utilizado no desenvolvimento de controle de alto nível em plataformas de extração de petróleo e refinarias da Petrobras S/A. O MPA é composto por um IDE que permite o desenvolvimento de aplicações usando uma linguagem visual baseada em fluxogramas. Essa IDE se comunica com um servidor de execução instalado no ambiente de automação, tanto para enviar aplicações para serem executadas no ambiente do servidor como também para monitorar essa execução através de suporte a depuração visual da execução. Tanto a IDE como o servidor de execução do MPA são implementados inteiramente em Lua. Toda a comunicação entre esses componentes do MPA é implementada usando o suporte CORBA do OiL.

Há três motivações principais para o uso do OiL no MPA. Primeiramente, recursos do OiL como ações automáticas e *proxies* alternativos tornaram a implementação do MPA mais simples. A segunda motivação é que o MPA também adota o modelo de concorrência cooperativa do OiL para execução das aplicações, o que permite que elas sejam desenvolvidas sem as necessidade de mecanismos sofisticados de sincronização. Finalmente, a portabilidade do OiL foi útil na compilação do servidor de execução do MPA nas diferentes versões de VMS que executam nas plataformas de extração de petróleo da Petrobras.

Como ferramenta de pesquisa, o OiL também tem se mostrado muito útil. Diferentes trabalhos de pesquisa foram realizados utilizando o OiL, tanto na PUC-Rio (Theophilo et al., 2005; Mello, 2005; Nogara, 2006; Correa, 2007) como em outras universidades no Brasil (Oliveira Neto, 2008; Castor Filho et al., 2008; Silva, 2009). Alguns desses trabalhos resultaram em recursos importantes do OiL, em particular o suporte à concorrência cooperativa e a arquitetura baseada em componentes.

Adicionalmente, utilizamos o OiL em um curso sobre implementação de middleware em que os alunos tiveram a oportunidade de implementar extensões

do OiL. A experiência de uso do OiL nesse curso se mostrou satisfatória, pois permitiu que os alunos pudessem praticar muitos dos conceitos apresentados. Inclusive, uma das extensões desenvolvidas durante o curso resultou na implementação do suporte de *proxies* genéricos do OiL. Outra ferramenta utilizada em conjunto durante o curso foi o μORB[1], que consiste de implementações parciais, em C++ e Java, do protocolo GIOP de CORBA. O μORB faz parte de um livro didático (Puder et al., 2006) sobre a implementação do Mico e é utilizado como introdução à implementação do modelo de ORB de CORBA, pois a implementação do Mico se mostra muito complexa para isso. Acreditamos que, como ferramenta de ensino, o OiL seja uma boa alternativa ao uso do μORB por ser uma implementação inteiramente funcional de um ORB com suporte completo do GIOP, mas que não apresenta a mesma complexidade da implementação do Mico. O maior problema com o uso do OiL pelos alunos foi a dificuldade de navegar na implementação devido às dependências ocultas resultante do uso do modelo de componentes, assim como a pouca experiência com a linguagem Lua.

A utilização do OiL nesses diferentes projetos é uma evidência da viabilidade de linguagens dinâmicas no desenvolvimento de middleware. O uso de Lua propicia algumas características peculiares do OiL, que em muitas vezes foram decisivas para o seu sucesso nesses projetos, tais como portabilidade, eficiência e facilidade de uso. Por outro lado, a maior propensão a erros devido à tipagem dinâmica exige a utilização de mecanismos alternativos de identificação de erros ainda no desenvolvimento, tais como a utilização de testes de regressão. Contudo, acreditamos que isso não seja diferente da maior parte dos projeto em linguagens menos dinâmicas como C++ e Java.

7.2 Avaliação do OiL

A dificuldade inicial da avaliação do OiL foi definir os critérios a serem considerados. Apesar da execução de testes de desempenho exigirem cuidados especiais e apresentarem certas dificuldades, existem critérios de avaliação desempenho bem consolidados, tais como tempo de resposta, utilização de memória e escalabilidade. Nesses aspectos, o OiL apresentou resultados satisfatórios. A avaliação de desempenho realizada indica que o desempenho do OiL é comparável ao de implementações similares em Java utilizadas comercialmente, porém com uso de memória bem menor. Esses resultados são promissores visto que a implementação do OiL é mais voltada a simplicidade e portanto inclui poucas otimizações. Consequentemente, é provável que o de-

[1]http://www.mico.org/textbook/

sempenho do OiL possa ser melhorado, caso se invista um esforço maior em sua otimização.

Em relação à flexibilidade, ou mais especificamente à facilidade de modificação de software, a definição de critérios de avaliação é mais difícil. Diferentes abordagens têm sido utilizadas para avaliar facilidade de programação (Ellis et al., 2007; Nielsen e Molich, 1990; Rieman et al., 1995). Em particular, o CDN (Green, 1989) é adotado neste trabalho como critério base para avaliação de flexibilidade. O CDN se mostrou eficaz no sentido de orientar a avaliação, pois define um conjunto representativo de aspectos relevantes de facilidade de uso. Diferentemente de métodos que avaliam flexibilidade exclusivamente através da quantificação de alterações específicas no código, como é proposto em (Eden e Mens, 2006), a avaliação com CDN permite considerar também outros aspectos relacionados aos custos cognitivos dessas modificações. Isso é importante, pois modificações no software não envolvem apenas a incorporação das modificações, mas também seu planejamento e idealização. Em particular, neste trabalho, identificamos que alguns recursos que permitem facilitar a execução de modificações (*i.e.* diminuir a viscosidade), como a utilização de tipagem dinâmica, implicam em maior custo cognitivo na idealização da modificação (*e.g.* maior propensão a erros).

Por outro lado, a avaliação baseada no CDN também apresentou dificuldades importantes. O objetivo original do CDN é propor critérios que possam ser facilmente utilizados sem necessidade de treinamento especializado. Contudo, duas características do CDN dificultam isso. Primeiramente, o número de dimensões é relativamente extenso, portanto é difícil considerar todas simultaneamente durante o estudo do objeto sendo avaliado. Além disso, muitas dimensões se sobrepõem ou competem entre si, o que dificulta dividir CDN em conjuntos menores que possam ser utilizados em avaliações independentes. O grande número de dimensões também dificulta a apresentação dos resultados, que se mostram extensos e difíceis de sintetizar sem omitir aspectos importantes. Em segundo, muitas dimensões são inteiramente subjetivas e difíceis de serem metrificadas ou identificadas objetivamente. Portanto, a utilização adequada do CDN demanda um período de experiência e familiarização com cada dimensão, de forma que o avaliador possa compreender adequadamente o que cada dimensão visa identificar. Adicionalmente, há poucas ferramentas de auxílio ao uso do CDN, como o que é proposto em (Blackwell e Green, 2000). Por fim, vale ressaltar que o CDN não define um conjunto completo de critérios de facilidade de uso. Por essa razão, a avaliação apresentada neste trabalho não pode ser vista como evidência da flexibilidade do OiL, mas sim como um indicador de aspectos importantes relacionados a isso.

Apesar das limitações da avaliação de flexibilidade baseada no CDN, acreditamos que ela tenha sido eficaz em identificar alguns aspectos importantes. Em particular, as características específicas do OiL se mostraram adequadas para modificações experimentais, como melhor suporte para provisão e avaliações progressivas. Isso indica a adequação do OiL como ferramenta de ensino e como base para experimentação com outras implementações de middleware. Por outro lado, essa avaliação também permitiu identificar problemas importantes que comprometem a flexibilidade do OiL. Em particular, a maior propensão a erros relacionada à tipagem dinâmica, que só permite identificar alguns erros na execução da aplicação. Outro problema importante é o maior número de dependências ocultas devido ao uso de componentes que podem ser montados de diferentes formas pela aplicação. Se por um lado isso diminui a viscosidade da implementação, pois permite trocar partes da implementação mais facilmente, as dependências ocultas dificultam a compreensão da implementação.

Todas as três características de linguagens dinâmicas de Lua foram efetivamente utilizadas na implementação do OiL. Contudo, duas delas se mostraram mais marcantes, a tipagem dinâmica e a reflexão computacional. Em relação a tipagem estática de C++, a tipagem dinâmica oferece inúmeras vantagens, como menor prolixidade, maior visibilidade e menor viscosidade do código. A principal razão para isso é não haver necessidade de declarações explícitas de tipo, ao contrário do que ocorre em C++. Acreditamos, que a tipagem estática baseada em inferência de tipos pode permitir as mesmas vantagens da tipagem dinâmica, porém sem implicar em maior propensão a erros, que só são identificados durante a execução da aplicação. Em particular, o suporte à tipagem estática com suporte a valores dinamicamente tipados como em (Abadi et al., 1991) se mostra uma alternativa promissora para o desenvolvimento de middleware.

A reflexão computacional é amplamente utilizada no OiL como mecanismo de meta-programação para ajustar o middleware às necessidades específicas da aplicação. A principal vantagem da reflexão computacional em relação a outras formas de meta-programação é a possibilidade de ajustar o middleware na carga ou na execução da aplicação, permitindo implementações dinamicamente adaptáveis. Contudo, o suporte a reflexão computacional é um recurso difícil de ser utilizado, tanto na linguagem como no modelo de componentes, pois geralmente é muito propenso a erros, exigindo atenção especial do programador. Por exemplo, quando um componente é substituído, o programador geralmente deve lidar com a mudança do estado interno do componente, além de possíveis chamadas em execução no componente que podem nunca

cessar em alguns sistemas (Ajmani et al., 2006). Nesse sentido, acreditamos que seja necessária a utilização de abstrações de programação específicas para adaptação dinâmica, que permitam lidar adequadamente com esse tipo de complexidade. Acreditamos que o OiL seja uma boa plataforma para esse tipo de experimentação, como é proposto em (Maia et al., 2005).

7.3 Trabalhos Futuros

Este trabalho investigou a adequação de linguagens dinâmicas na implementação de middleware. Contudo, vários outros trabalhos podem ser sugeridos nesse sentido como continuidade das idéias propostas aqui, em particular, a evolução e avaliação da implementação do OiL pode fornecer novas informações sobre isso. Nesse sentido podemos sugerir linhas de trabalhos futuros em quatro frentes:

Funcionalidades do OiL Apesar do OiL ser projetado para oferecer suporte a diferentes protocolos de RMI, ele só oferece suporte a um protocolo padronizado, o GIOP 1.0 de CORBA. Portanto, é interessante validar esse suporte através da implementação de outros protocolos, como o utilizado pelo ICE (Henning, 2004) ou SOAP (Box et al., 2000), ou mesmo versões mais avançadas do GIOP que permitam fragmentação de mensagens e comunicação bi-direcional. Outra linha de investigação nessa direção é o desenvolvimento de protocolos de RMI dinamicamente tipadas como o LuDO e o PYRO, mas que sejam projetados para serem eficientes. Nesse sentido, formatos binários eficientes que permitam codificação de dados dinâmicos, como o *Google Protocol Buffers*[2], podem ser alternativas interessantes.

Além da introdução de suporte a novos protocolos, outra linha de trabalho futuro é a evolução do modelo de programação oferecido, que pode ser feito através de novos tipos de *proxy*, como por exemplo *proxies* para objetos replicados ou comunicação em grupo, ou ainda modificações na API oferecida pelo ORB. Em particular, uma linha de trabalho é implementar interfaces de programação similares às definidas pelo padrão CORBA, que incluem o modelo do POA, tal como oferecido por implementações em CORBA em linguagens dinâmicas (Object Management Group, 2002; Object Management Group, 2009). Isso é interessante para permitir comparar as vantagens da API de CORBA com a API atual do OiL, que se baseia em recursos de Lua, como o uso de meta-tabelas para implementar funcionalidades oferecidas pelo POA. Outra linha de trabalho nesse sentido é implementar mecanismos que facilitem

[2] `http://code.google.com/apis/protocolbuffers/`

o acesso a serviços básicos, como é feito no Pyro para acesso a um serviço de nomes (ver seção 6.2.2).

Implementação do OiL A implementação atual do OiL apresenta problemas e limitações que podem ser objeto de investigação futura, em particular, a busca por abordagens alternativas de implementação de componentes, como por exemplo, modelos de implementação mais simples que o utilizado pelo LOOP, que permitam diminuir a barreira de abstração da implementação do OiL. Outra alternativa são modelos com suporte para rastreamento de dependências como os que são propostos em (Kon e Campbell, 2000; Coulson et al., 2008), que podem facilitar o suporte à adaptação dinâmica do middleware, assim como permitir comparar com abordagens similares em linguagens não-dinâmicas como C++ (Coulson et al., 2002; Coulson et al., 2006; Grace et al., 2005; Costa et al., 2005).

Outro problema da implementação do OiL é a dificuldade relacionada ao ajuste do ORB através de montagens específicas de componentes. Atualmente, esse processo é propenso a erros e não permite a seleção de funcionalidades transversais. Portanto, uma linha de investigação futura é propor mecanismos que facilitem esse processo, como o modelo de *component frameworks* utilizado no OpenORB (Coulson et al., 2002), além de abordagens mais flexíveis para implementação de fucionalidades transversais, como a utilização de programação orientada a aspectos (Silva, 2009).

Por fim, o modelo de concorrência do OiL apresenta limitações importantes, em particular, a falta de integração com o sistema operacional. Um trabalho futuro nesse sentido é a implementação de um escalonador integrado ao sistema operacional que fosse capaz de interceptar chamadas de sistema bloqueantes e suspendesse apenas a execução da *thread* cooperativa que realizou a chamada, passando a escalonar as demais durante o bloqueio.

Avaliação de Flexibilidade O uso do CDN para avaliação de facilidade de programação é uma área de pesquisa recente, em particular, como forma de diminuir a dificuldade de programação. Nesse sentido, o CDN também pode ser utilizado para avaliar da facilidade de utilização do OiL em aplicações, o que permitiria identificar as vantagens do uso de linguagens dinâmicas em implementações que apenas exportam a funcionalidade de um ORB para uso em linguagens dinâmicas (Cerqueira et al., 1999; Grisby, 2008; Tackaberry, 2000).

Poucos trabalhos têm sido propostos com o intuito de utilizar o CDN como critério específico de avaliação de flexibilidade como é feito nesse traba-

lho. Nesse sentido, duas linhas de trabalho podem ser sugeridas: (a) utilização do CDN na avaliação de flexibilidade de outras implementações, em particular, implementações de middleware configuráveis e dinamicamente adaptáveis; e (b) adaptação do CDN especificamente para avaliação de flexibilidade, que pode ser feito através da identificação de dimensões mais relevantes ou mesmo a definição de um outro conjunto de dimensões baseadas na experiência do uso do CDN nesse contexto.

Ainda no sentido de avaliar a flexibilidade do OiL, outro possível trabalho futuro é avaliar a facilidade de adaptar a implementação específica do OiL ao contexto específico de aplicações de tempo real, em particular, avaliar possíveis ajustes no modelo de coleta de lixo de Lua para viabilizar a predição de execução, assim como uso de co-rotinas de Lua como modelo de *multithreading* determinístico.

Avaliação de Desempenho O OiL apresenta um desempenho satisfatório nos experimentos apresentados neste trabalho. Contudo, é importante estender esses experimentos para incluir outras implementações de middleware além de avaliar outros critérios de desempenho, como é feito em (Tůma, 2002). Nesse sentido, também é interessante avaliar o desempenho da implementação de outros protocolos no OiL, assim como comparar o desempenho do OiL com outros middleware que utilizem protocolos diferentes do GIOP como o Java RMI ou Pyro. A comparação de desempenho do OiL com implementações de CORBA em linguagens dinâmicas também é útil como forma de comparar o desempenho de um ORB em Lua com o de outras linguagens dinâmicas. Em particular, é interessante a comparação com sistemas de middleware implementados em linguagens não-dinâmicas, mas que são exportados para utilização em linguagens dinâmicas, como é feito em (Cerqueira et al., 1999; Grisby, 2008; Tackaberry, 2000). Por fim, vale ressaltar que uma contribuição importante das avaliações de desempenho é permitir identificar gargalos de desempenho na implementação do OiL, assim como permitir identificar possíveis otimizações que permitam ganhos significativos de desempenho para aplicações.