Aspect Open-ORB: Um Middleware Reflexivo Orientado a Aspectos

Nélio Cacho[1,2], Thaís Batista[1], Alessandro Garcia[2], Cláudio Sant´anna[2]

[1]Departamento de Informática e Matemática Aplicada (DIMAp)
Universidade Federal do Rio Grande do Norte (UFRN)
Campus Universitário – Lagoa Nova – 59.072-970 - Natal – RN

[2]Computing Department – Lancaster University
United Kingdom

n.cacho@lancaster.ac.uk, thais@ufrnet.br, a.garcia@lancaster.ac.uk, c.santanna@lancaster.ac.uk

Resumo. *Esse artigo apresenta o Aspect Open-ORB, uma plataforma de middleware baseada no Open-ORB que combina reflexão computacional com programação orientada a aspectos para permitir a separação dos conceitos transversais e, assim, melhorar o reuso e a capacidade de customização das plataformas de middleware reflexivas. O Aspect Open-ORB é implementado usando duas estratégias distintas: (i) uma estratégia interpretada utilizando a linguagem Lua em combinação com AspectLua; (ii) uma estratégia compilada que emprega a linguagem Java em combinação com AspectJ. De forma a avaliar os benefícios de cada estratégia, este trabalho apresenta uma série de comparações que traduzem as diferenças em termos de modularidade, memória utilizada e tempo de execução.*

1. Introdução

Plataformas de middleware (Bernstein 1996) vêm sendo bastante utilizadas no desenvolvimento de aplicações distribuídas por simplificar o processo de desenvolvimento oferecendo uma abstração em relação aos detalhes de distribuição e um conjunto de serviços comumente usados por aplicações distribuídas. Middlewares são geralmente estruturados em torno de padrões de projeto, modelo de componentes e, muitas vezes, usam reflexão computacional visando permitir a customização do próprio middleware através da introdução e remoção de funcionalidades.

Nesse contexto, o principal problema é que tais mecanismos usados na estruturação do middleware não são capazes de reduzir a complexidade e conseqüentemente aumentar o reuso dessas plataformas em ambientes móveis que são caracterizados pela existência de vários *conceitos transversais* (Kiczales 1997), tais como persistência/sincronização, tratamento de erros, mobilidade e sensibilidade ao contexto. O inevitável entrelaçamento dos conceitos transversais gerado pela utilização conjunta dos mecanismos convencionais de estruturação do middleware leva a uma limitação do poder de customização dessas plataformas, uma vez que aumenta o acoplamento entre os módulos e impede o amplo reuso e a remoção/introdução de funcionalidades.

De forma a resolver esse problema, o presente trabalho tem como objetivos: (i) *analisar* como a capacidade de customização de uma plataforma de middleware pode ser melhorada pela combinação das estratégias convencionais (padrões de projeto, modelo de componentes e reflexão computacional) com um paradigma emergente, a

Programação Orientada a Aspectos (POA) (Kiczales 1997), (ii) *definir* uma arquitetura que suporte a clara separação dos conceitos transversais, reuso e customização dinâmica do middleware e (iii) *validar* tal arquitetura por meio da implementação de dois protótipos e da avaliação dos benefícios e limitações de tais protótipos em termos de um conjunto de métricas que consideram separação de conceitos, acoplamento, coesão, tamanho, desempenho, utilização de memória e capacidade de customização.

Esse artigo está organizado da seguinte forma. A seção 2 apresenta os conceitos básicos relacionados com esse trabalho: middleware reflexivo, Open-ORB, POA, AspectJ e AspectLua. A seção 3 define a arquitetura do Aspect Open-ORB. A seção 4 apresenta a avaliação do Aspect Open-ORB. A seção 5 apresenta uma comparação com os trabalhos relacionados. A seção 6 contém as conclusões.

2. Conceitos Básicos

2.1 Middleware Reflexivo e Open-ORB

O principal objetivo da *reflexão* (Smith 1982) em sistemas computacionais é permitir que um sistema possa executar algum processamento em benefício próprio, para modificar ou ajustar sua estrutura ou comportamento. *Middlewares reflexivos* (Blair 1998, Parlavantzas 2000) usam reflexão computacional visando um maior poder de adaptação do middleware. A reflexão está diretamente ligada aos detalhes de implementação de um sistema, uma vez que promove a separação entre as funcionalidades fornecidas pelos objetos (*nível-base*), e os detalhes de sua implementação (*meta-nível*).

Middlewares construídos a partir dessa abordagem, tais como Open-ORB (Blair 1998), OpenCOM (Parlavantzas 2000), dynamicTAO (Kon 2000) são geralmente estruturados em termos de três camadas: infra-estrutura básica, infra-estrutura de componentes e aplicação. Na camada de infra-estrutura básica estão as classes que definem a estrutura de reflexão juntamente com as classes que definem o modelo de componentes. Na camada de infra-estrutura de componentes, os elementos definidos na camada básica são instanciados para compor o *meta-level* e os serviços que implementam a conectividade do middleware. Finalmente, na camada de aplicação o usuário utiliza a API provida pelas camadas anteriores para estruturar a aplicação desejada.

2.2 Programação Orientada a Aspectos

A programação orientada a aspectos (POA) é um paradigma emergente de desenvolvimento de software que busca melhorar a modularidade e reuso em sistemas de software. Estes benefícios são alcançados através da separação dos conceitos transversais (*crosscutting concerns*) e conceitos básicos de um sistema. Exemplos tradicionais de conceitos transversais são tratamento de erros, persistência, distribuição, etc.

AspectJ (Team 2002) é uma extensão Java para suportar a POA. Em AspectJ os conceitos transversais são separados da aplicação e modelados em um elemento denominado *aspecto*. Um mecanismo de composição (*weaving*) realiza a junção dos aspectos com os conceitos básicos para compor o sistema. AspectLua (Cacho et al. 2006) é uma extensão da linguagem Lua (Ierusalimsky 1996) para suportar a POA.

AspectLua provê um conjunto de funcionalidades que potencializam e facilitam a utilização da POA: (1) inserção e remoção de aspectos em tempo de execução; (2) definição de ordem de precedência entre aspectos; (3) a possibilidade de usar *wildcards*; e (4) a possibilidade de associar aspectos com elementos não declarados (*anticipated join points*) (Cacho et al. 2006).

3. Aspect Open-Orb

Neste trabalho utilizamos a POA em conjunto com a reflexão computacional para customizar a infra-estrutura básica do Open-ORB. O objetivo dessa customização visa facilitar o processo de inserção/remoção de funcionalidades em ambientes móveis com grande variabilidade de requisitos. Entretanto, tal customização só é possível se houver uma perfeita combinação entre a POA e reflexão computacional. Com o objetivo de prover tal combinação, a arquitetura Aspect Open-ORB usa POA para customizar a infra-estutura básica e reflexão computacional para customizar as duas outras camadas.

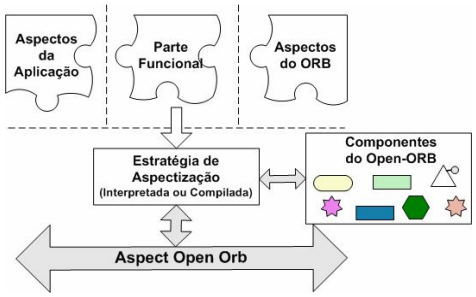

Figura 1: Arquitetura de uma aplicação sobre o Aspect Open-ORB

A Figura 1 ilustra a arquitetura de uma aplicação desenvolvida sobre o Aspect Open-ORB. Inicialmente são definidos os aspectos relativos ao ORB (Configuração dos elementos e dependências), seguido pela definição do código funcional da aplicação e, por último, pelos aspectos da aplicação (Segurança, Persistência, etc). Estes três elementos são combinados em duas estratégias (intepretada e compilada) para gerar a implementação final baseada nos requisitos da plataforma alvo.

A estratégia interpretada é baseada nas características dinâmicas da linguagem Lua e no conjunto de elementos fornecidos pelo AspectLua e LOpenOrb (uma implementação Lua do OpenOrb). Esta estratégia baseia-se na idéia de que as funcionalidades da aplicação são definidas pelas funções invocadas pelo código da aplicação. Dessa forma, AspectLua deve controlar os métodos invocados e verificar suas dependências, de modo que estas sejam carregadas e executadas de acordo com a necessidade. Os detalhes de implementação desta estratégia estão descrito em (Cacho 2005). Por sua vez, a estratégia compilada é baseada nas características da linguagem Java e na forma como AspectJ e JOpenOrb (uma implementação Java do OpenOrb) podem gerar um modelo adequado para reduzir o número de configurações para as plataformas alvo. Os detalhes desta estrategia estão descritos em (Cacho 2006b).

A distinção entre as estratégias é motivada pelo diferente suporte provido por cada uma das linguagens frente a variabilidade de requisitos. Na linguagem Lua, cada plataforma possui um interpretador próprio que provê a mesma API. Isto permite desenvolver um único LOpenOrb para todas as plataformas, lidando apenas com restrições de memória. Por esse motivo, a estratégia de aspectização do LOpenOrb foca

na questão da inserção dinâmica de novas funcionalidades, buscando assim otimizar o uso dos recursos.

Por outro lado, a solução Java fornece um conjunto diferenciado de APIs para cada plataforma alvo. Ou seja, aplicativos que utilizem CLDC (*Connected Limited Device Configuration*), que é uma especificação voltada para dispositivos extremamente restritos em termos de memória, largura de banda e segurança, não podem requerer recursos avançados com capacidades reflexivas. Por exemplo, a implementação proposta do JOpenOrb é inadequada para dispositivos móveis com baixos recursos que utilizam J2ME com a configuração CLDC, uma vez que utiliza vários recursos que não são suportados por essa configuração, tal como a interface *Iterator* e o pacote *java.lang.reflection*. As soluções propostas neste trabalho resolvem este problema por meio dos mecanismos da POA que permitem configuração do JOpenOrb de acordo com a plataforma alvo, evitando assim, a definição e duplicação de múltiplas versões de um mesmo middleware.

4. Avaliação

A avaliação das soluções envolveu a comparação dos dois protótipos originais LOpenOrb e JOpenOrb com os dois protótipos que implementam a arquitetura Aspect Open-ORB. A versão Lua do Aspect Open-ORB demonstrou uma melhor capacidade de customização, na comparação com o LOpenOrb, ao permitir a personalização da camada de infra-estrutura básica. Como conseqüência, a versão Lua do Aspect Open-ORB também reduziu o tempo de execução e a memória consumida (Cacho 2005). Para a comparação entre as duas versões Java, utilizamos um conjunto de métricas de software (Sant'Anna 2003) que capturam importantes características de modularidade do sistema, tais como métricas de separação de conceitos (SoC), acoplamento, coesão e tamanho.

Os resultados das medições mostraram que a modularidade da implementação Aspect Open-ORB aumentou significantemente em comparação com a versão JOpenOrb. Isto fica evidente pela redução em 50% do nível de entrelaçamento da maioria dos conceitos transversais (Cacho 2006b). Em termos de desempenho, a implementação Java do Aspect Open-ORB reduziu o tempo de instanciação do middleware em 28%, mas por outro lado introduziu uma perda de 11% sobre o desempenho do JOpenOrb. Na comparação entre as versões LOpenOrb e JOpenOrb, a versão Java apresentou um melhor desempenho enquanto que a versão Lua permitiu uma melhor capacidade de customização. No caso das implementações Lua e Java do Aspect Open-ORB, as duas abordagens apresentaram similar desempenho enquanto que a versão Lua manteve a hegemonia no suporte a customização.

5. Trabalhos Relacionados

Em termos de trabalhos relacionados, a idéia de aspectizar plataformas de middleware vem sendo explorada em vários artigos para suportar diversos propósitos. Por exemplo, Jacobsen (Godil 2005, Zhang 2005) utiliza POA para resolver o problema dos conceitos transversais e assim melhorar o desempenho da plataforma alvo. O Lasagne (Truyen

2001) é um middleware orientado a aspectos para customização de aplicações dinâmicas e sensíveis ao contexto. O JAC (Pawlak 2004) é um middleware orientado a aspectos implementado em Java que implementa comunicação por RMI. O JBOSS AOP é um framework orientado a aspectos que pode ser integrado com um dos serviços do servidor JBOSS. Aspectos são definidos por classes Java e relacionados com o código da aplicação via documentos XML ou anotações Java.

A reflexão computacional é também suportada por outras abordagens como o dynamicTAO (Kon 2000) que é um middleware reflexivo que suporta a customização de componentes para habilitar reconfiguração dinâmica. FlexiNet (Hanssen 1999) é uma plataforma desenvolvida em Java para permitir a invocações de métodos remotos via uma infra-estrutura de *binding.* Tal infra-estrutura permite que uma aplicação controle a pilha de protocolo e as *constraints* que são usadas para construir um *binding* para cada objeto. O Jonathan (Dumant 1999) é uma plataforma de middleware organizado em quatro frameworks (binding, comunicação, recursos e configurações) que permitem a introdução de novos tipos de bindings e protocolos de comunicação.

Na comparação com as abordagens citadas acima, apenas a abordagem provida por Jacobsen suporta a aspectização da infra-estrutura básica. No entanto, nossa abordagem distingue-se da provida por Jacobsen em vários aspectos: (i) a versão Java do Aspect OpenORB prioriza a modularidade em detrimento do desempenho fornecido pela solução do Jacobsen, (ii) por sua vez a versão Lua prioriza a capacidade dinâmica de inserir e remover aspectos, algo que as soluções do Jacobsen não suportam, e (iii) por fim a nossa versão orientada a aspectos é obtida pela total aspectização da versão orientada a objetos, enquanto que Jacobsen aspectiza apenas os aspectos que podem melhorar o desempenho da middleware, ou seja, a utilização da versão do Jacobsen em ambientes limitados pode exigir uma nova aspectização.

Em relação a customização da infra-estrutura de componentes e aplicação, percebemos que todas as abordagem usam a mesma técnica para as duas camadas. As soluções FlexiNet, dynamicTAO, Jonathan, Lasagne e JBoss AOP suportam customização dinâmica pela utilização de mecanismos diversos como reflexão computacional, *binding factories* e aspectos para permitir a inserção e remoção de funcionalidades na infra-estrutura de componentes e aplicação. No entanto, por limitações da própria linguagem de implementação, como Java e C++, tais abordagens estão restritas a aplicação da customização para pontos bem definidos na arquitetura ou framework utilizados. Por outro lado, as abordagens JAC e Jacobsen não suportam a customização dinâmica, uma vez que o processo de *weaving* implementado por essas abordagens é restrito a tempo de carga ou compilação, não permitindo assim a flexibilidade das abordagens dinâmicas.

6 Conclusões

Neste trabalho apresentamos uma proposta para integração de reflexão computacional e Programação Orientada a Aspectos no contexto de desenvolvimento de uma plataforma de middleware. Diferentemente das abordagens tradicionais, nossa proposta explora duas estratégias: (1) utiliza POA para implementar a customização da infra-estrutura básica; (2) utiliza reflexão computacional para implementar a customização das demais camadas do middleware. No sentido de avaliar como esta abordagem é afetada pela

linguagem de programação utilizada, implementamos neste trabalho dois protótipos de parte da arquitetura proposta. O primeiro protótipo utilizou uma linguagem dinamicamente tipada com facilidades reflexivas, a linguagem Lua, e uma extensão dessa linguagem que dá suporte a programação orientada a aspectos, AspectLua. O segundo prótotipo avaliou os impactos dessa abordagem em uma linguagem compilada, Java, utilizando padrões de projeto e uma extensão para a programação orientada a aspectos, AspectJ. A escolha por estas duas linguagens deve-se ao fato de se tratarem de paradigmas diferentes e que, portanto, podem validar a solução mais amplamente.

Referências

Bernstein, P. A. (1996) Middleware: A Model for Distributed System Services. Commun. ACM, ACM Press, v. 39, n. 2, p. 86–98. ISSN 0001-0782.

Blair, G. S. et al. (1998) An architecture for next generation middleware. In: Proceedings of the IFIP Int. Conf. on Distributed Systems Platforms and Open Distributed Processing. Springer-Verlag, 1998.

Burke B. et al. (2003) The Aspect Oriented Programming and JBoss tutorial. Disponível em: <http://www.onjava.com/pub/a/onjava/2003/05/28/>.

Cacho, N.; Batista, T. (2005) Using AOP to Customize a Reflective Middleware. DOA – Distributed Objects and Applications. Lecture Notes in Computer Science (LNCS), v. 3761.

Cacho, N., Batista, T., Fernandes, F. (2006) A Lua-based AOP Infrastructure. Journal of the Brazilian Computer Society, v. 11, p. 7-20, 2006.

Cacho, N. et al. (2006b) Improving Modularity of Reflective Middleware with Aspect-Oriented. Programming. In: 6th Workshop on Software Engineering and Middleware, 2006, Portland.

Dumant, B. et al. (1999) Jonathan: an open distributed processing environment in java. Distributed Systems Engineering, v. 6, n. 1, p. 3–12.

Godil, I.; Jacobsen, H.-A. (2005) Horizontal Decomposition of Prevayler. In: CASCON ’05: Proc. of the 2005 Conf. of the Centre for Advanced Studies on Collaborative research. IBM Press, p. 83–100.

Hanssen et al. (1999) A Framework for Policy Bindings. In: DOA ’99: Proceedings of the International Symposium on Distributed Objects and Applications. Washington USA: IEEE Computer Society p. 2.

Ierusalimschy, R.; Figueiredo, L. H. de; Filho, W. C. (1996) Lua an Extensible Extension Language. Softw. Pract. Exper., John Wiley Sons, Inc., v. 26, n. 6, p. 635–652, 1996. ISSN 0038-0644.

Kiczales, G. et al. (1997) Aspect-oriented Programming. In: IT, M. A.; MATSUOKA, S. (Ed.). Proceedings European Conference on Object-Oriented Programming. Berlin, Heidelberg, and New York: Springer-Verlag, 1997. v. 1241, p. 220–242.

Kon, F. et al. (2000) Monitoring, Security, and Dynamic Configuration with the DynamicTao Reflective Orb. In: IFIP/ACM International Conference on Distributed systems platforms. p. 121–143.

Parlavantzas, N. et al. (2000) Towards a Reflective Component Based Middleware Architecture. Disponível em: <citeseer.csail.mit.edu/331827.html>.

Pawlak, R. et al. (2004) JAC: an aspect-based distributed dynamic framework. Softw. Pract.Exper., John Wiley & Sons, NY, USA, v. 34, n. 12, p. 1119–1148.

Smith, B. C. (1982) Procedural Reflection in Programming Languages. Tese (Doutorado) — Massachusetts Institute of Technology.

Truyen, E. et al. (2001) Dynamic and selective combination of extensions in component-based applications. ICSE ’01: Proc.of the 23rd Int. Conf. on Software Engineering. IEEE p. 233–242.

Team, A. (2002) The AspectJ Programming Guide. Disponível em: <http://aspectj.org>.

Zhang, C.; Jacobsen, H.-A. (2003) Quantifying aspects in middleware platforms. In: Proceedings of the 2nd international conference on Aspect-oriented software development. ACM Press, p. 130–139. ISBN 1-58113-660-9.

Zhang, C.; Gao, D.; Jacobsen, H.-A. (2005) Generic middleware substrate through modelware. In: Middleware´05. p. 314–333.