Aspectos para Construção de Aplicações Distribuídas

Cristiano Amaral Maffort
Orientador: Marco Túlio de Oliveira Valente

Instituto de Informática
Pontifícia Universidade Católica de Minas Gerais

maffort@varginha.cefetmg.br, mtov@pucminas.br

Resumo. *Neste resumo estendido, são apresentados os principais resultados de uma dissertação de mestrado na qual foi projetado, implementado e validado um sistema de apoio à programação distribuída que permite isolar a funcionalidade de distribuição da lógica de negócio de aplicações implementadas em Java. Para alcançar esse objetivo, o sistema proposto se beneficiou da sinergia gerada pela combinação de três tecnologias: aspectos, linguagens de domínio específico e geração e transformação de código. A expressividade e flexibilidade do sistema projetado na dissertação foi comprovada por meio de sua utilização em três sistemas distribuídos de médio porte.*

1. Introdução

Plataformas de *middleware* são empregadas por engenheiros de software para simplificar e tornar mais produtivo o desenvolvimento de aplicações distribuídas. Basicamente, estes sistemas encapsulam diversos detalhes inerentes a programação distribuída, incluindo protocolos de comunicação, heterogeneidade de arquiteturas, serialização de dados, sincronização, localização de serviços etc. No entanto, sistemas de *middleware* são, em geral, invasivos, pervasivos e transversais ao código de negócio de aplicações distribuídas [7, 3, 2]. Como conseqüência, aplicações distribuídas implementadas com o apoio de tais sistemas não apresentam graus esperados de reusabilidade, separação de interesses e modularidade, o que dificulta o entendimento, manutenção e evolução das mesmas.

Neste resumo estendido, descrevem-se os principais resultados de uma dissertação de mestrado na qual foi projetado, implementado e validado um sistema de apoio à programação distribuída, chamado DAJ (*Distribution Aspects in Java*) [5, 6][1]. DAJ permite isolar a funcionalidade de distribuição da lógica de negócio de aplicações implementadas em Java. Para alcançar esse objetivo, o projeto de DAJ se beneficiou da sinergia gerada pela combinação de três tecnologias: aspectos, linguagens de domínio específico e geração e transformação de código. No sistema proposto, aspectos são usados para encapsular diversos detalhes de programação requeridos por plataformas de *middleware*; linguagens de domínio específico são usadas para elevar o grau de abstração quando da definição de parâmetros de distribuição de uma determinada aplicação. Por fim, DAJ inclui uma ferramenta de geração de código, responsável por gerar aspectos que introduzem não-invasivamente código de distribuição usualmente requerido por plataformas de *middleware*. A versão atual do sistema gera aspectos para duas plataformas de *middleware* nativas de ambientes de desenvolvimento Java: Java RMI e Java IDL. Java RMI é um *middleware* utilizado com freqüência por aplicações distribuídas cujos módulos

[1] O texto completo da dissertação está disponível em: www.inf.pucminas.br/prof/mtov/maffort.pdf.

são integralmente implementados em Java. A plataforma Java IDL, por sua vez, é uma implementação do padrão CORBA e, portanto, permite que aplicações distribuídas implementadas em Java acessem sistemas desenvolvidos em outras linguagens de programação.

Este resumido estendido está estruturado conforme descrito a seguir. A Seção 2 descreve a interface de programação de DAJ. A Seção 3 apresenta uma visão geral da arquitetura usada na implementação do sistema. A Seção 4 resume os estudos de casos realizados. A Seção 5 avalia o projeto de DAJ; a Seção 6 compara DAJ com trabalhos relacionados e a Seção 7 apresenta as conclusões.

2. Interface de Programação

Em aplicações distribuídas implementadas com o apoio do sistema DAJ, classes de negócio não precisam seguir quaisquer convenções de programação, isto é, não precisam estender classes da API de uma determinada plataforma de *middleware*, não precisam implementar interfaces ou métodos `get` e `set`, não precisam ativar ou tratar exceções remotas etc. Em vez disso, no sistema proposto, descritores de distribuição são usados para descrever o papel desempenhado por tais classes em um sistema de objetos distribuídos baseado em uma determinada tecnologia de *middleware*. A partir das informações contidas em um descritor de distribuição, a ferramenta de geração de código integrante do sistema DAJ se encarrega de gerar automaticamente aspectos e classes que modularizam o código de distribuição requerido pela aplicação base. Os aspectos gerados por DAJ são implementados em AspectJ.

A Figura 1 resume o processo de desenvolvimento de aplicações distribuídas usando DAJ. A fim de melhor descrever esse processo, são descritos a seguir os principais componentes de uma aplicação distribuída construída com apoio de DAJ.

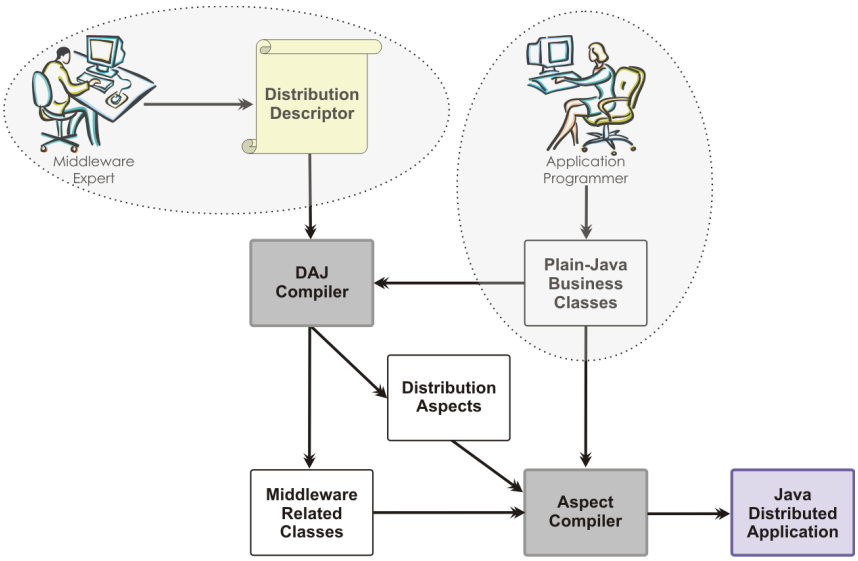

Figura 1. Desenvolvimento de aplicações distribuídas usando DAJ

Descritores de Distribuição: Por meio de tais componentes, organizados na forma de documentos XML, são configurados os objetos remotos que compõem a aplicação, ou seja, os objetos que recebem chamadas remotas de métodos. Além disso, são definidos

os objetos que, em chamadas remotas de métodos, são passados por serialização e por referência remota. Em DAJ, objetos remotos que são registrados em um servidor de nomes são chamados de servidores, e são especificados por meio de um nodo `server`, no qual definem-se um identificador para o servidor, o nome de sua interface, o nome da classe que implementa essa interface, o protocolo de comunicação utilizado (Java RMI ou Java IDL) e o nome e a porta do servidor de nomes onde o servidor será registrado. No caso de tipos passados por referência remota (nodo `remote`), deve-se informar a interface e a classe desse tipo. No caso de tipos passados por serialização (nodo `serializable`), deve-se informar a classe desse tipo.

Mostra-se a seguir um exemplo de descritor de distribuição para um sistema simples de controle de ações. Por meio deste descritor, configura-se uma implantação do sistema com dois servidores: o primeiro deles utilizará Java IDL para comunicação e será registrado com o nome `StockMarketA` (linhas 1 a 6); o segundo utilizará Java RMI para comunicação e será registrado com o nome `StockMarketB` (linhas 7 a 12). Além disso, define-se que, em chamadas remotas de métodos desses servidores, objetos do tipo `StockListenerImpl` serão passados por referência remota (linhas 13 a 16) e que objetos do tipo `StockInfo` serão passados por serialização (linhas 17 a 19).

```
 1:  <server id="StockMarketA">
 2:    <interface>stockMarket.StockMarket</interface>
 3:    <class>stockMarket.StockMarketImpl</class>
 4:    <protocol>javaidl</protocol>
 5:    <nameserver>skank.pucminas.br</nameserver>
 6:  </server>
 7:  <server id="StockMarketB">
 8:    <interface>stockMarket.StockMarket</interface>
 9:    <class>stockMarket.StockMarketImpl</class>
10:    <protocol>javarmi</protocol>
11:    <nameserver>patofu.pucminas.br:1530</nameserver>
12:  </server>
13:  <remote>
14:    <interface>stockMarket.StockListener</interface>
15:    <class>stockMarket.StockListenerImpl</class>
16:  </remote>
17:  <serializable>
18:    <class>stockMarket.StockInfo</class>
19:  </serializable>
```

Clientes: Em sistemas de objetos distribuídos, clientes obtêm referências para objetos remotos e então chamam métodos dos mesmos. Em DAJ, um cliente deve utilizar o método `getReference` para obter uma referência para um dos servidores configurados no descritor de distribuição. A partir da referência obtida, o cliente pode chamar métodos remotos desse servidor, passando parâmetros tanto por serialização quanto por referência remota, sem precisar estar ciente do *middleware* que suporta esta comunicação. Exemplifica-se a seguir a obtenção de referências para os servidores definidos no descritor de distribuição mostrado anteriormente.

```
1: StockMarket s1,s2;
2: s1=(StockMarket)ServiceLocator.getReference("StockMarketA");
```

```
 3: s2=(StockMarket)ServiceLocator.getReference("StockMarketB");
 4:   ....
 5: StockInfo info;
 6: info= new StockInfo("petr3", 124.60, "10/04/2006 18:21");
 7: s1.update(info);
 8: .......
 9: StockListener listener= new StockListenerImpl();
10: s1.subscribe("vale5", listener);
11: s2.subscribe("petr4", listener);
```

Nas linhas de 1 a 3, por meio do método `getReference`, este cliente obtém referências para os servidores de nome `StockMarketA` e `StockMarketB`, definidos no descritor de distribuição mostrado. Em seguida, o cliente chama o método `update` do primeiro servidor passando um objeto do tipo `StockInfo` por valor (linhas 5 a 7). Por fim, o cliente manifesta seu interesse em receber notificações sobre alterações nos preços de duas ações (linhas 9 a 11). Veja que em ambas chamadas do método `subscribe`, o cliente informa o mesmo objeto do tipo `StockListener`. Este objeto, passado sempre por referência remota, receberá notificações do primeiro servidor via Java IDL e do segundo servidor via Java RMI.

Servidores: DAJ gera uma classe de ativação para cada servidor configurado no descritor de distribuição, a qual possui um método `main` contendo código para instanciar, ativar e registrar o respectivo objeto remoto.

3. Arquitetura Interna

Descreve-se resumidamente nesta seção a arquitetura interna de DAJ. Basicamente, o sistema se encarrega de gerar interfaces remotas, transformar classes de negócio em classes remotas, obter referências para objetos remotos, ativar servidores e tratar exceções ativadas remotamente.

Interfaces Remotas: Interfaces remotas são interfaces com código de distribuição entrelaçado. Essas interfaces são automaticamente geradas por DAJ a partir das informações contidas nos descritores de distribuição. Tanto em Java RMI quanto em Java IDL, as interfaces geradas são adaptadas segundo as especificidades destas plataformas de *middleware* e de acordo com o cenário de implantação configurado nos descritores de distribuição.

Classes Remotas: Uma classe remota é aquela resultante da introdução em uma classe de negócio de código de distribuição requerido por uma determinada plataforma de *middleware*. Em DAJ existem dois tipos de classes remotas:

- Classes remotas usadas para instanciação de servidores: Como qualquer classe remota, estas classes devem implementar uma interface remota gerada por DAJ. Além disso, se houver necessidade de converter parâmetros de tipos de negócio em tipos remotos, essa conversão é automaticamente realizada antes da chamada efetiva do método remoto, por meio de aspectos.

- Classes remotas usadas para instanciação de objetos remotos: Nesse caso, são realizadas as mesmas adaptações aplicadas em classes usadas para instanciação de servidores. Além disso, introduz-se na classe um método responsável por realizar a ativação de seus objetos, a fim de que eles possam receber chamadas remotas.

Obtenção de Referências Remotas: Em DAJ, clientes conseguem referências para servidores remotos chamando o método `getReference`. Devido a incompatibilidades de tipo entre a interface remota e a interface de negócio, o método `getReference` retorna um *proxy* que encapsula uma referência para o objeto remoto. Este *proxy* possui duas funções: tratar exceções remotas lançadas pela plataforma de *middleware* em caso de falhas de comunicação e ativar o objeto remoto, quando o mesmo é utilizado como argumento de uma chamada remota de método, cuja semântica seja por referência remota.

Tratamento de Exceções: Plataformas de *middleware* são responsáveis por transmitir exceções ativadas no espaço de endereçamento do servidor para o espaço de endereçamento do cliente. Em Java IDL, um aspecto captura a ocorrência de uma exceção e realiza as transformações necessárias para que esta exceção seja enviada para o espaço de endereçamento do cliente. Já em Java RMI, exceções de negócio ativadas por métodos remotos são simplesmente propagadas para o espaço de endereçamento do cliente, usando o mecanismo de serialização nativo de Java.

4. Estudos de Casos

DAJ foi utilizado para modularizar código de distribuição presente em três aplicações:

- *HealthWatcher*: um sistema para gerenciamento de reclamações realizadas por cidadãos sobre as condições sanitárias de estabelecimentos comerciais da área de alimentação. Uma primeira experiência usando AspectJ para modularizar código de distribuição Java RMI presente nesse sistema foi relatada em [7]. Essa experiência foi repetida nesta dissertação, porém usando DAJ.

- *Network Pricing System* (NPS): um sistema financeiro para controle de ações negociadas em uma bolsa de valores [4]. Como NPS foi originalmente implementado usando Java IDL, sua inclusão como um dos sistemas avaliados na dissertação teve como objetivo analisar a aplicabilidade de DAJ em sistemas desenvolvidos originalmente nesta plataforma de *middleware*.

- *Library*: um sistema Java RMI para controle de uma biblioteca. Esse sistema foi anteriormente usado para avaliar um *framework* para recuperação de diagramas de seqüência UML a partir do código fonte de sistemas Java [1].

Inicialmente, o código de distribuição espalhado e entrelaçado pelas classes de negócio destes três sistemas foi integralmente removido. Em seguida, foram definidos descritores de distribuição para os mesmos. A ferramenta de geração de código de DAJ foi então usada para gerar código de distribuição para essas aplicações, de acordo com as especificações dos descritores de distribuição. Como resultado, DAJ foi capaz de modularizar o código de distribuição entrelaçado ao código de negócio destes três sistemas. Também foi possível disponibilizar novas versões destes sistemas usando uma plataforma de *middleware* diferente daquela na qual o sistema foi originalmente implementado.

A Tabela 1 apresenta o número de linhas de código das versões original e refatorada dos sistemas avaliados. Como pode ser observado, o número de linhas da versão refatorada do sistema HealthWatcher foi reduzido em cerca de 11%. O número de linhas de código do sistema NPS foi reduzido em 75%, já que sua versão original utiliza diversas classes geradas pela plataforma Java IDL. Já o núcleo do sistema Library sofreu

um aumento mínimo. O motivo é que classes da API de Java utilizadas no sistema como parâmetro ou retorno de métodos remotos tiveram que ser reimplementadas, já que as mesmas não são permitidas na interface remota de sistemas Java IDL (pois sendo uma implementação de CORBA, Java IDL tem como objetivo prover independência de linguagem de programação).

	HealthWatcher	**NPS**	**Library**
Versão Original	5129	4417	4997
Versão DAJ	4566 (-10.98%)	1104 (-75.01%)	5011 (+0.28%)

Tabela 1. LOC das versões original e baseada em DAJ dos sistemas avaliados

5. Avaliação

Com base na experiência adquirida por meio dos estudos de caso realizados, apresenta-se a seguir uma avaliação do projeto de DAJ, de acordo com os seguintes critérios: modularização, usabilidade e flexibilidade, desempenho e portabilidade. Discutem-se também tecnologias alternativas que poderiam ter sido empregadas no projeto do sistema.

Modularização: Usando DAJ foi possível sintetizar aspectos e classes que modularizaram o requisito de distribuição nos três sistemas avaliados. Como resultado, o núcleo de tais sistemas tornou-se independente de qualquer código de distribuição e, portanto, mais simples de entender, testar e evoluir. Particularmente, testes puderam ser realizados sem considerar aspectos de distribuição, o que é uma característica relevante no caso de desenvolvimento orientado por testes.

Usabilidade e Flexibilidade: O uso de descritores de distribuição tornou bastante flexível a configuração do cenário de distribuição dos sistemas avaliados. Através desses descritores foi possível, por exemplo, modificar parâmetros de configuração, tais como a plataforma de *middleware* subjacente, a localização do servidor de nomes, os nomes dos objetos remotos etc. Além disso, também foi possível reconfigurar a arquitetura de distribuição dos sistemas avaliados (por exemplo, adicionando novos servidores).

Desempenho: Para avaliar o desempenho de DAJ, foi medido o tempo para executar métodos remotos do sistema de controle de ações descrito na Seção 2. Foram executadas vinte séries de cada método remoto deste sistema. Em cada série foram realizadas dez mil chamadas remotas. Então, para cada método, foi calculado o tempo médio de execução das vinte séries. Além disso, foram avaliadas quatro versões do sistema: duas baseadas em Java RMI (com e sem o uso de DAJ) e duas baseadas em Java IDL (com e sem o uso de DAJ). Os resultados, apresentados na Tabela 2, demonstram que DAJ não impacta significativamente o desempenho de uma aplicação distribuída, em relação à sua versão original, orientada por objetos

Portabilidade: Utilizando DAJ, é possível gerar versões de uma aplicação distribuída para duas plataformas de *middleware* (Java RMI e Java IDL). Permite-se ainda que um determinado objeto receba chamadas remotas utilizando simultaneamente estas duas plataformas de *middleware* (conforme ilustrado no exemplo da Seção 2). A definição da plataforma de *middleware* a ser utilizada requer somente a configuração adequada do descritor de distribuição.

Método remoto	RMI			IDL		
	DAJ	OO	%	DAJ	OO	%
`update`	2767	2698	2.56	6473	6450	0.36
`subscribe`	5901	5803	1.69	13809	13634	1.28
`unsubscribe`	3010	2989	0.70	6428	6289	2.21
`getStock` retornando `StockInfo`	2631	2648	-0.64	6326	6332	-0.09
`getStock` retornando uma exceção	5089	5045	0.87	6401	5893	8.62

Tabela 2. Tempo médio (em ms) para executar dez mil chamadas de métodos remotos.

Tecnologias Alternativas: Pelo menos duas outras tecnologias podem ser usadas para automatizar a geração de código de distribuição requerido por plataformas de *middleware*:

- Ferramentas para manipulação de *bytecodes*: No entanto, linguagens orientadas por aspectos, como AspectJ, oferecem abstrações de mais alto nível para realizar estas mesmas manipulações. Estas abstrações contribuíram para simplificar e tornar mais produtivo o projeto e a implementação de DAJ.

- Geração de esqueletos de classes: Esta abordagem possui duas deficiências: (i) o código requerido pelo *middleware*, apesar de não ser implementado manualmente, continua invasivo e espalhado pelo sistema, prejudicando seu entendimento e evolução; (ii) uma vez implementado em uma determinada plataforma de *middleware*, dificulta-se a migração de um sistema distribuído para uma outra plataforma, já que seu código de negócio seria inserido diretamente no esqueleto das classes geradas para o primeiro *middleware* escolhido.

6. Trabalhos Relacionados

Soares, Borba e Laureano descrevem uma experiência de uso de AspectJ para criar uma versão orientada por aspectos do sistema HealthWatcher [7]. Em relação a esse trabalho, DAJ apresenta pelo menos três contribuições: DAJ oferece uma solução para modularização de código de distribuição requerido tanto por Java RMI como por Java IDL; DAJ permite passagem de parâmetros por serialização e por referência remota e DAJ permite a geração de aspectos de distribuição independentemente da arquitetura de *software* adotada no projeto da aplicação alvo.

Ceccato e Tonella descrevem um *framework* orientado por aspectos para transformar uma aplicação centralizada em uma aplicação distribuída [2]. De forma semelhante ao que ocorre com DAJ, o núcleo da aplicação permanece livre do requisito de distribuição e todos os aspectos necessários para introdução desse requisito são gerados automaticamente. Entretanto, a solução proposta é restrita a Java RMI. Além disso, objetos podem ser passados como parâmetros de métodos remotos apenas com semântica de referência remota.

Ghosh e colegas descrevem uma metodologia para implementar interesses de negócio e interesses de distribuição providos por plataformas de *middleware* [3]. De forma semelhante a DAJ, eles propõem o uso de aspectos para separar os interesses de negócio das funcionalidades específicas de plataformas de *middleware*. Além disso, eles descrevem

uma metodologia orientada por modelos para desenvolvimento de sistemas distribuídos. Assim, consideramos que DAJ é um sistema que implementa e coloca em prática vários dos princípios e conceitos propostos por Ghosh e colegas.

7. Conclusões

DAJ proporciona pelo menos dois benefícios a um engenheiro de sistemas distribuídos. Primeiro, e mais importante, desenvolvedores podem se concentrar exclusivamente no desenvolvimento do código funcional de sua aplicação, já que DAJ produzirá automaticamente aspectos e classes que modularizam o código de distribuição requerido pela aplicação. Segundo, este desenvolvedor não precisa dominar detalhes e convenções de codificação requeridos por plataformas de *middleware*.

Do ponto de vista científico, o trabalho realizado apresenta as seguintes contribuições: a solução proposta inclui uma ferramenta para geração automática de aspectos de distribuição; a solução proposta cobre duas plataformas de *middleware*; a solução proposta permite passagem de parâmetros por serialização e por referência remota; a solução proposta é flexível para atender diversas arquiteturas e configurações de sistemas distribuídos; a solução proposta foi validada por meio de três estudos de caso de média complexidade.

Os primeiros resultados do trabalho foram apresentados em um artigo no XX Simpósio Brasileiro de Engenharia de Software, o qual foi classificado como quinto melhor artigo do evento [5]. Resultados consolidados do trabalho, incluindo os três estudos de caso realizados, foram publicados em um artigo no *Journal of the Brazilian Computer Society* [6].

Referências

[1] Lionel C. Briand, Yvan Labiche, and Johanne Leduc. Toward the reverse engineering of UML sequence diagrams for distributed Java software. *IEEE Transactions on Software Engineering*, 32(9):642–663, 2006.

[2] Mariano Ceccato and Paolo Tonella. Adding distribution to existing applications by means of aspect oriented programming. In *4th IEEE International Workshop on Source Code Analysis and Manipulation*, pages 107–116. IEEE Computer Society, 2004.

[3] S. Ghosh, R. B. France, A. Bare, B. Kamalalar, R. P. Shankar, D. M. Simmonds, G. Tandon, P. Vile, and S. Yin. A middleware transparent approach to developing distributed applications. *Software Practice and Experience*, 35(12):1131–1154, October 2005.

[4] Geoffrey Lewis, Steven Barber, and Ellen Siegel. *Programming with Java IDL*. John Wiley & Sons, 1997.

[5] Cristiano Amaral Maffort and Marco Túlio Oliveira Valente. Aspectos para construção de aplicações distribuídas. In *XX Simpósio Brasileiro de Engenharia de Software*, October 2006.

[6] Cristiano Amaral Maffort and Marco Túlio Oliveira Valente. Modularizing communication middleware concerns using aspects. *Journal of the Brazilian Computer Society*, 13(4):81–95, 2007.

[7] Sergio Soares, Paulo Borba, and Eduardo Laureano. Distribution and persistence as aspects. *Software Practice and Experience*, 36(7):711–759, 2006.