# ReflectTools: Uma ferramenta de apoio ao desenvolvimento de software reconfigurável

Frank José Affonso[1]

Evandro Luís Linhari Rodrigues[2]

**Resumo:** O desenvolvimento de software apoiado por ferramentas foi um grande marco na área de engenharia de software. Nos últimos anos, tem-se observado um aumento significativo no uso de ferramentas nos processos de desenvolvimento, devido aos interesses por automatização das atividades e otimização em prazos e custos. Outro fator relevante nessa área é a modificação dos requisitos de um sistema (evolução de hardware, software, redes de comunicação ou necessidades dos clientes) e, consequentemente, a adaptação que deve ocorrer com esses sistemas para que essas modificações sejam atendidas. Embasados nesses conceitos, este artigo apresenta a ReflectTools, uma ferramenta de apoio ao desenvolvimento de software reconfigurável. Esta ferramenta foi projetada para automatizar os passos previstos na Metodologia e no Ambiente de Execução Reconfigurável (definido nesta metodologia). Para proporcionar aos sistemas a capacidade de reconfiguração, esta ferramenta implementa um subsistema supervisor de objetos, que é acoplado aos sistemas na etapa de desenvolvimento/ reconfiguração. Para apresentar a ferramenta e suas funcionalidades foi elaborado um estudo de caso utilizando RMI, Remote Method Invocation.

**Palavras-chave: Ferramenta ReflectTools. Metodologia. Reconfiguração de Software.**

**Abstract:** This paper presents a tool, named ReflectTools, to Reconfigurable Software Development. To understand the peculiarities of this kind of software, a Methodology and a Reconfigurable Execution Environment are presented in this paper. Thus, it is possible to demonstrate how this kind of software is developed, as well as the environment features, whose activities (subsystems object supervisor and retrieval information) are automated by the proposed tool. In order to present ReflectTools, an application using remote method invocation was developed as a case study. Thereby, it is possible to evaluate the behavior of these subsystems in the software reconfiguration step.

**Keywords:** *ReflectTools. Methodology. Software Reconfiguration.*

## 1 Introdução

A necessidade de sistemas computacionais dotados de características de reconfiguração em tempo de execução é um desejo antigo por parte dos engenheiros de software para melhor incorporar as necessidades emergentes de seus clientes. Para atender a esse desejo tem-se a reflexão computacional [1], [2] e [3] como um mecanismo muito eficaz/eficiente na realização da adaptação/reconfiguração do software em tempo de execução. Este mecanismo pode ser aplicado em sistemas desenvolvidos em várias abordagens de desenvolvimento, tais

[1] Curso de Ciência da Computação, UNESP, Campus Rio Claro - Avenida 24A, 1515, Bela Vista - Rio Claro (SP) - Brasil
{affonso.frank@gmail.com, frank@rc.unesp.br}
[2] Curso de Engenharia Elétrica, USP, Campus São Carlos - Avenida Trabalhador São-carlense, 400, Arnold Schimidt - São Carlos (SP) - Brasil
{evandro@sc.usp.br}

**http://dx.doi.org/10.5335/rbca.2011.1804**

como orientados a objetos [1], orientado a componentes [4], orientados a aspectos [5], orientados a serviços [6], orientados a chamada de métodos remotos [7] e a combinação de ambos.

Ainda sobre o contexto da reconfiguração de sistemas, tem-se observado que os trabalhos desenvolvidos correspondem a projetos isolados, sem a adoção de ferramentas e processos de engenharia. Sobre estes últimos pode-se dizer que foram idealizados para atuar desde a captação dos requisitos até geração automática de código fonte. No entanto, alguns autores, [8] e [9], comentam sobre a carência dessas ferramentas em atender plenamente às fases de um projeto (ciclo de vida de software) e, quando atendem, tem-se um problema em relação ao investimento, custo elevado, na aquisição e manutenção [10].

Sobre a carência de ferramentas em relação ao atendimento das fases de um processo de software, outros autores, [8], [11] e [12], apontam a particularidade de atuação como um fator negativo, pois foram elaboradas para sanar problemas específicos e devem atuar de maneira cooperativa com outras ferramentas para complementar a atividade de desenvolvimento. Os autores mencionam que essa cooperação nem sempre ocorre de maneira natural, devido às dificuldades de integração das informações geradas (modelos) no processo de desenvolvimento. Por fim, mencionam que a falta de planejamento de seus idealizadores é o motivo do insucesso, pois, quando foram projetadas/construídas, a possibilidade de integração com outras ferramentas não foi abordada [11] e [12].

Diante do contexto apresentado, este artigo apresenta a ferramenta ReflectTools [13], que oferece apoio ao desenvolvimento de software reconfigurável. Esta ferramenta foi desenvolvida com finalidade específica e oferece suporte aos passos estabelecidos na Metodologia de Desenvolvimento de Software Reconfigurável (MDSR). Apesar das características específicas, em seu projeto foi estabelecida a capacidade de integração com as ferramentas (workbenchs) de desenvolvimento, tais como: Eclipse [14] e Netbeans [15]. Essas ferramentas são utilizadas na fase inicial da MDSR para modelagem e, posteriormente, desenvolvimento do sistema. Depois de realizadas essas etapas, os projetos são transferidos para a ferramenta ReflectTools e o engenheiro de software/desenvolvedor passa a utilizá-la diretamente no AER na automatização de suas tarefas [13].

Ainda sobre a MDSR e o AER, a utilização de um processo automatizado (ReflectTools) é um fator significativo na redução da participação dos desenvolvedores na etapa de concepção e codificação do software, resultando na minimização das incertezas na etapa de reconfiguração. Para realizar a reconfiguração, a ferramenta ReflectTools possui uma técnica de reconfiguração implementada no formato de um subsistema, que atua no AER no monitoramento e na modificação dos aspectos, classes, componentes e serviços (desenvolvidos na linguagem Java [16]) - chamados deste ponto em diante de artefatos de software [13] e [17].

Sobre a reconfiguração de software, pode-se dizer que, ao longo dos anos, existiu um interesse na comunidade acadêmica e corporativa de associar ao software alguma “capacidade de raciocínio”, de modo que possa ser capaz de coletar as informações do meio no qual está inserido, processá-las e tomar alguma decisão para que sua execução não seja interrompida. Dessa forma, o software poderia melhor atender às novas necessidades dos clientes e dos avanços tecnológicos (hardware, software e redes de comunicação) atuais, que apontam para as novas necessidades do software: agilidade, mobilidade e adaptabilidade [18].

Este artigo está organizado da seguinte maneira: na seção 2 são apresentados os conceitos, definições e trabalhos relacionados; na seção 3, uma breve descrição da MDSR e do Ambiente de Execução Reconfigurável (AER); na seção 4, a ferramenta ReflectTools e seus subsistemas (supervisão de objetos e repositório de métodos); na seção 5, um estudo de caso, que mostra a utilização da ferramenta em uma aplicação utilizando RMI[19]; e, finalmente, na seção 6, as considerações finais.

## 2 Conceitos, Definições e Trabalhos Relacionados

Esta seção apresenta os conceitos, definições e os trabalhos relacionados que ofereceram contribuição para a elaboração deste artigo. Para melhor apresentar os assuntos envolvidos, esta seção foi organizada da seguinte maneira: na seção 2.1 são apresentados os conceitos e definições sobre software reconfigurável e as técnicas de reconfiguração e, na seção 2.2, os trabalhos relacionados sobre as ferramentas de engenharia de software e processos automatizados.

## 2.1 Conceitos e Definições

A reflexão computacional (RC), [1], [2], [19], [21], [22] e [23], pode ser definida como qualquer atividade executada por um sistema sobre si mesmo. O principal objetivo é obter informações sobre suas próprias atividades, tendo como objetivo melhorar seu desempenho, introduzir novas capacidades (características ou funcionalidades), ou, ainda, resolver seus problemas escolhendo o melhor procedimento [24] *apud* [19].

No modelo de computação reflexiva (Figura 1) é apresentado como ocorre a comunicação entre os objetos (supervisionados) e os metaobjetos (supervisores). Neste modelo, um metaobjeto, pertencente ao metanível, é responsável por (1) interceptar a solicitação de um objeto do nível-base e (2) encaminhar esta solicitação para outro metaobjeto do metanível, que a (3) entregará a um objeto do nível-base para que a requisição original seja atendida. Quando nenhum objeto for encontrado, uma mensagem de retorno é exibida ao objeto solicitante. O caminho inverso, devolução da execução (4, 5 e 6), também é considerado.

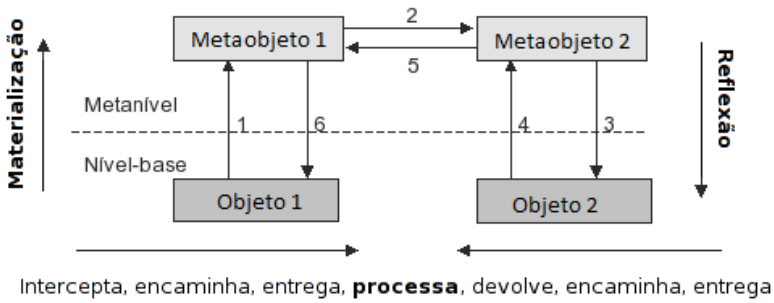


**Figura 1.** Comunicação entre objetos e metasobjetos ([1], [2], [21] e [22])

Em Maes [1] e Lisbôa [24] existe um relato de que a RC permite alterar os aspectos estruturais e comportamentais de um sistema através dos seguintes mecanismos: **(a) Introspecção**, que é encarregada das informações da estrutura do sistema (atributos e métodos); **(b) Reflexão estrutural**, responsável por obter e alterar o estado do sistema para o qual ele foi projetado, inserindo/retirando atributos e métodos, e; **(c) Reflexão comportamental**, que é responsável por obter e realizar modificações no nível base (sistema). Esses conceitos apresentados sobre RC foram utilizados na implementação de um subsistema supervisor de objetos presente na ferramenta ReflectTools, que é encarregada de monitorar e realizar as modificações nos artefatos de software em tempo de execução.

Para realizar a reconfiguração de software várias técnicas foram encontradas na literatura especializada. Essas técnicas atuam em diferentes contextos da computação e trabalham em diferentes níveis de implementação, porém estão sempre associadas a outros recursos de programação, como por exemplo, a Programação Orientada a Aspectos (POA) ([25] e [26]) e a Programação Orientada a Componentes ([27] e [28]). A seguir são apresentadas algumas dessas técnicas que contribuíram na elaboração do subsistema supervisor de objetos implementado na ferramenta ReflectTools.

Em Tanter [5], a POA é utilizada na adaptação estrutural e comportamental do sistema. Os aspectos são utilizados para entrelaçar os requisitos não funcionais (transversais) aos objetos do sistema. O entrelaçamento de requisitos (funcionais e não funcionais) é realizado por meio da ferramenta Reflex.

Janik e Zielinski [32] apresentam um sistema completo com reconfiguração baseada em aspectos. A estratégia utilizada pelos autores é a adaptação dos requisitos não funcionais; o nível funcional permanece intacto. Dessa forma, os autores destacam as mudanças ocorridas no meio de infraestrutura, tais como segurança e distribuição. Para realizar este tipo de adaptação, os autores propuseram uma extensão da POA, a AAOP, Adaptable Aspect-Oriented Programming, que representa um conjunto de aspectos adaptáveis em tempo de execução que são integrados aos artefatos de software.

Segundo Borde [29], Gomaa e Hussein [30] e Weiss [31], a RC é utilizada para a adaptação de componentes de software [27] de duas maneiras: **(1) estrutural**, que envolve a estrutura do componente como a alteração na interface ou no nome de uma operação; e **(2) comportamental**, que afeta as propriedades funcionais e não funcionais [25] e [26]. Essa técnica de adaptação está centralizada no empacotamento do componente devido às incompatibilidades da interface original. As funcionalidades existentes são preservadas e outras, referentes às novas necessidades, são adicionadas constituindo um novo componente de software.

Kim [33] relata que o desenvolvimento de software baseado em componentes tem sido empregado pelos melhores métodos de engenharia de software na atividade de desenvolvimento. Dessa forma, recursos tecnológicos modernos têm sido utilizados no desenvolvimento de componentes e no registro (deploy) por terceiros, ou adaptados [34], para que atendam aos requisitos de seus clientes. Assim, um método de adaptação de componentes binários é proposto pelo autor [33], cujo objetivo é controlar o tamanho dos componentes a cada adaptação realizada, pois a técnica empregada é o empacotamento de novas funcionalidades e, com isso, podem aumentar de tamanho rapidamente. Nesse método, os componentes são utilizados como plugins por uma ou mais aplicações [4], [29] e [31].

Segundo Maes [1], Forman e Forman [2] e Kasten [35], a adaptação pode ser utilizada em componentes de software em um modelo dividido em dois enfoques: (1) observação do comportamento (introspection) e (2) observação da mudança do comportamento (intercession). Nesse modelo se destacam as operações realizadas no metanível: refractions, que fornece uma visão limitada sobre o nível-base do componente; e transmutations, que representa a capacidade de modificar a funcionalidade de um componente no metanível.

Para Richmond e Noble [36], a reconfiguração de aplicações distribuídas desenvolvidas com a RMI [19] é limitada devido à falta de mecanismos de reflexão em objetos remotos. Sua proposta é a utilização do padrão Proxy [23] para que os objetos remotos se comportem de maneira transparente à aplicação como se fossem locais. Dessa forma, pode-se dizer que as modificações são aplicadas aos objetos remotos sem a percepção de seus clientes.

A reconfiguração de componentes em ambientes distribuídos também é abordada por Chen [7]. Sua proposta é uma extensão da RMI, XRMI - eXtended Java RMI, que permite que uma aplicação possa monitorar e manipular invocações entre componentes durante um processo de reconfiguração dinâmica. Detalhando suas etapas, (1) um componente pode ser carregado dinamicamente no sistema, (2) migrado de um local para outro no ambiente distribuído, (3) descarregado do ambiente em tempo de execução, ou (4) atualizado dinamicamente (modificação de sua estrutura interna - atributos e/ou métodos).

Para implementar a RC, a linguagem de programação deve ter um recurso computacional que seja capaz de reconhecer as características e funcionalidades de classes/objetos em tempo de execução. Segundo Oracle-Reflect [37], a reflexão em Java é realizada por meio da API, Application Programming Interface, Java Reflect [37]. Esta API permite realizar a descoberta/modificação de informações como atributos, tipos, métodos, construtores, tipos de retorno, parâmetros de métodos, classes carregadas pela JVM, Java Virtual Machine, entre outras. O subsistema supervisor de objetos (seção 4) presente na ferramenta ReflectTools foi desenvolvido utilizando somente os recursos da API Java Reflect.

### 2.2 Trabalhos Relacionados

Segundo Whitehead [8], Nakagawa [9] e Pressman [38], a engenharia de software vem enfrentando problemas quanto à existência de métodos e ferramentas que auxiliem os engenheiros de software em todas as etapas de desenvolvimento. Atualmente é grande o número de ferramentas de apoio direcionadas à solução desses problemas. No entanto, tem-se observado que poucas atendem plenamente às necessidades dos desenvolvedores, devido às mutações existentes nas tendências de desenvolvimento de sistemas. Quando atendem, existe o problema relacionado ao custo elevado, que inviabiliza sua aquisição por empresas de pequeno e médio porte.

De acordo com Gray[11], as ferramentas reduzem de maneira significativa o tempo e o custo de desenvolvimento em um projeto de software. O autor menciona a existência de várias ferramentas para propósitos específicos e a cooperação que deve existir entre elas na execução de um projeto. Em seu trabalho é possível encontrar uma breve classificação das ferramentas: aquelas direcionadas aos problemas específicos e as mais abrangentes, que atuam em várias fases do ciclo de vida de um software.

Dond [39] comenta sobre a falta de ferramentas CASE, Computer Aided Software Engineering, e ISEE, Integrated Software Engineer Environment, para gerenciar o desenvolvimento de software em linha de produto (LPS - Linha de Produto de Software) [40]. O autor relata que uma das principais razões para a falta deste tipo de ferramenta é flexibilização da arquitetura de componentes, que possa ser adaptada e reutilizada em outros domínios de sistemas.

Para Spanjers [12], as dificuldades de realizar o desenvolvimento de software distribuído estão

relacionadas às localizações geográficas e à falta de ferramentas capazes de realizar o gerenciamento deste tipo desenvolvimento. Um dos fatores apontados como insucesso num projeto de software distribuído é a falta de comunicação entre os membros das equipes ou entre as equipes. O autor comenta ainda que o uso de diferentes ferramentas de engenharia de software no processo de desenvolvimento contribui de maneira negativa para o sucesso do projeto, pois existem dificuldades na integração das informações geradas nas diferentes fases do projeto, que acabam acarretando em dificuldades na execução do projeto como um todo.

Toth [10] aborda a utilização de ferramentas de engenharia de software no processo de desenvolvimento, especificamente aquelas de código aberto e sem custo de licenças. O autor cita o mesmo problema sobre a falta de integração das ferramentas existentes e necessidade de combinar ferramentas para o desenvolvimento de um projeto [12]. Em seu trabalho é possível encontrar um conjunto de vantagens sobre a elaboração de software utilizando ferramentas open source. Dentre elas, destacam-se: (1) a união entre os desenvolvedores dessas ferramentas (time), caracterizando o desenvolvimento de software distribuído; (2) o custo de aquisição das ferramentas proprietárias, as quais nem sempre atendem às necessidades de um projeto.

Em Henttonen e Matinlassi [41] é possível encontrar uma avaliação e classificação das ferramentas em relação ao reúso e compartilhamento das informações em um ambiente distribuído. Além disso, critérios como comunicação entre os desenvolvedores são apontados como fator de sucesso na elaboração de um projeto de software orientado ao reúso. Outro fator relacionado ao sucesso é a maturidade da ferramenta em relação ao atendimento das fases do ciclo de vida de software. Os autores discutem sobre a definição da arquitetura de software como um fator importante para aumentar o reúso de software. Para isso, comentam sobre o uso de repositórios como mecanismo de armazenamento e reutilização (recuperação) da informação.

Estublier [42] apresenta uma proposta de um ambiente de engenharia de software que permite realizar as seguintes atividades: (1) definição do domínio do problema; (2) confeccionar a abstração do problema em modelos; (3) implementar a separação dos interesses; (4) realizar a adaptação entre os domínios; (5) fornecer mecanismos de implementação (persistência, controle de versão, etc); (6) realizar a interoperabilidade com outros sistemas. Para isso, o autor faz uso de uma arquitetura de referência e de geradores de código para criação de código fonte de maneira automática e padronizada.

Em Shi [43], a RC é utilizada na adaptação dos componentes de software em um processo automatizado. O objetivo da proposta do autor é o aumento do reúso de software em termos de uma arquitetura de componentes reflexiva, fazendo com que esta se torne mais flexível e possa ser adaptada e reutilizada em vários outros domínios/sistemas.

## 3 A Metodologia e o Ambiente de Execução Reconfigurável

Antes de iniciar a apresentação da ferramenta ReflectTools e de suas funcionalidades, julga-se necessário mostrar uma breve descrição dos passos da MDSR, para a qual esta ferramenta foi elaborada. Conforme apresentado na seção 2.1, o software reconfigurável (adaptável em tempo de execução) possui algumas características específicas. Dentre essas se destacam: (1) sua capacidade de autoanálise e modificação de suas características e funcionalidades; (2) necessidade de isolamento das funcionalidades a serem modificadas em tempo de execução; (3) recurso computacional (API Java Reflect [37], citado na seção 2.1) capaz de permitir que reconfiguração seja executada. Assim, baseado nessas características e na necessidade de criar um sistema de reconfiguração automática, foram elaborados uma metodologia, um ambiente de execução e uma ferramenta que pudessem auxiliar no desenvolvimento de software dessa natureza [13] e [17].

Ainda sobre a atividade de reconfiguração de software, em trabalhos anteriores, Affonso [13] e Affonso e Rodrigues [17] definem que esta atividade ocorre com maior facilidade em artefatos desenvolvidos com o mínimo de requisitos não-funcionais (infraestrutura e/ou transversais) e com a menor participação humana etapa de codificação. A minimização da participação da participação humana em processo de engenharia de software é possível quando algumas atividades (total ou parcial) são automatizadas [11], [12], [38] e [44]. A Figura 2 mostra a MDSR para o desenvolvimento de artefatos de software e sua inserção nos respectivos repositórios.

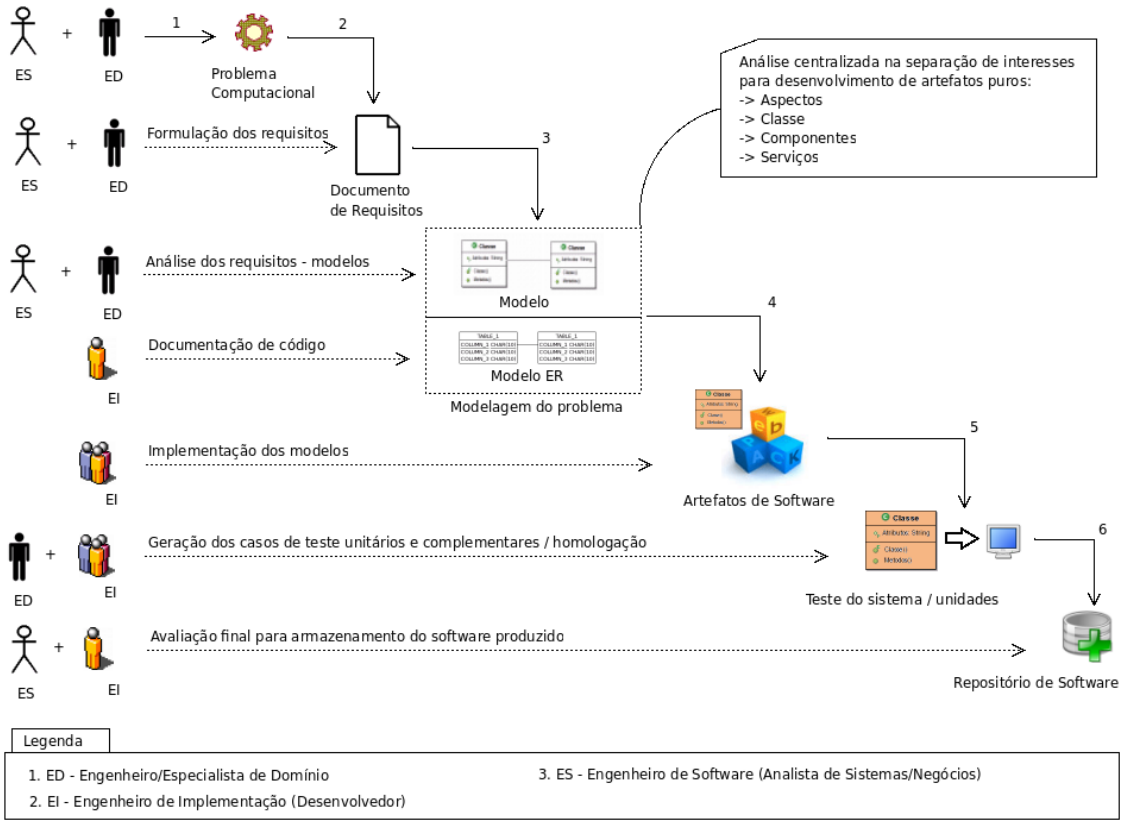


**Figura 2.** Metodologia de Desenvolvimento de Software Reconfigurável - parte 1 [13] e [17]

Em Affonso [13] e Affonso e Rodrigues [17] é possível encontrar uma descrição detalhada de todos os passos da MDSR, sendo apresentadas neste artigo, por motivos de escopo, apenas uma breve descrição de cada um deles. Como etapa preparatória para o desenvolvimento, o engenheiro de software deve preparar uma das ferramentas de desenvolvimento (Eclipse [14] ou Netbeans [15]). Essa preparação envolve a instalação de plugins para modelagem da lógica (Diagrama de Classes, Componentes e/ou outros), dos dados (Modelo Entidade-Relacionamento) e apoio ao desenvolvimento (código fonte). Sobre esses plugins, pode-se dizer que não existem restrições na escolha, pois o código fonte em Java e os scripts DDL (Data Definition Language), gerados por esses plugins, são importados para a ferramenta ReflectTools.

Inicialmente, o engenheiro de software, baseado nos requisitos, redige o documento de requisitos (**passos 1 e 2 da Figura 2**). Com base nesse documento, são confeccionados os modelos (Diagrama de Classes e Modelo Entidade-Relacionamento) do sistema (**passo 3 da Figura 2**). Os modelos gerados no passo 3 não são limitados apenas aos diagramas de classes; outros diagramas podem ser confeccionados, conforme abordagem de desenvolvimento adotada pelo engenheiro de software (Desenvolvimento de Software Orientado a Objetos, Orientado a Aspectos, Orientado a Componentes e/ou Orientado a Serviços). A próxima etapa (**passo 4 da Figura 2**) corresponde à geração automática de código fonte dos artefatos a partir dos modelos confeccionados no passo 3. Nesta etapa destaca-se a importância da preparação das ferramentas (*Eclipse* [14] ou Netbeans [15]) e dos plugins escolhidos (**etapa preparatória**). Depois de gerados os códigos fontes dos artefatos é iniciada a etapa de implementação do sistema (**passo 5 da Figura 2**). Nesta etapa, recomenda-se a inserção das informações semânticas no código fonte, que descrevem as funcionalidades dos artefatos de software e são utilizadas na seleção dos artefatos na etapa de recuperação da informação. Nota-se que essa etapa tem o apoio de um especialista de domínio para a geração e controle de vocabulário dessas informações. Ainda no passo 5, após a etapa de desenvolvimento, ou de maneira concomitante, são realizados os testes unitários por meio do framework JUnit [45], por exemplo. Finalmente (**passo 6 da Figura 2**), o projeto (sistema) é importado para a ferramenta ReflectTools. Em seguida, o engenheiro de software faz o armazenamento do projeto (sistema) nos respectivos repositórios de informações para serem utilizados no desenvolvimento de novos artefatos (reúso) ou reconfiguração automática no ambiente de execução.

A etapa de desenvolvimento de novos artefatos (Figura 3) pode ser realizada com a reutilização de

artefatos já elaborados e testados (garantia de funcionalidade). Para isso, consultas aos repositórios de informações são realizadas na busca do melhor artefato que atende às necessidades dos engenheiros de software. Ao realizar uma consulta, as informações de busca são enviadas ao ambiente para a localização do melhor artefato (**passos 1 e 2 da Figura 3**). Em seguida, cabe ao engenheiro de software avaliar se a solução apresentada (**passo 3 da Figura 3**) satisfaz aos critérios iniciais de busca para que esse artefato seja acoplado ao sistema. Em caso negativo, uma nova pesquisa pode ser realizada, modificando os critérios de busca, ou assume-se essa atividade como encerrada e um novo artefato será desenvolvido. Em caso positivo, o artefato é associado ao sistema por meio da combinação de aspectos ou injeção de dependência como um requisito não funcional (**passo 4 da Figura 3**). Finalmente, o artefato está pronto para ser utilizado (**passo 5 da Figura 3**). Esse processo pode ser repetido e novos artefatos podem ser selecionados no ambiente novamente (**passo 6 da Figura 3**), caracterizando um processo de desenvolvimento iterativo.

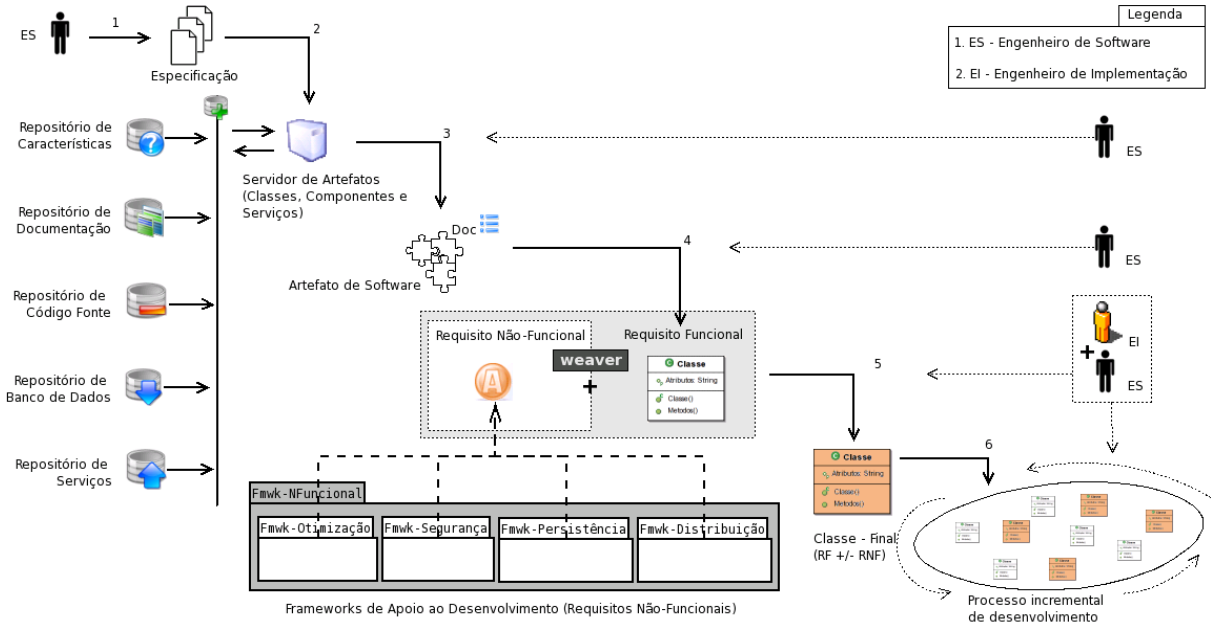


**Figura 3.** Metodologia de Desenvolvimento de Software Reconfigurável - parte 2 [13] e [17]

Os passos sobre a MDSR apresentados nesta seção correspondem, basicamente, aos passos apresentados (telas) na seção 5, que descreve o estudo de caso. Dessa forma, pode-se mostrar que a ferramenta ReflectTools atua desde o recebimento de um projeto; armazenamento de artefatos selecionado nos repositórios e na recuperação e instanciação de um ambiente de execução utilizando RMI [19]. O subsistema de supervisão dos objetos (seção 5) é instanciado automaticamente para o monitoramento de um artefato quando um artefato é inserido no ambiente de execução.

A Figura 4 mostra o AER. Na parte central é possível observar um servidor que interliga os servidores de objetos presentes em cada domínio (1 a 4). Cada servidor de objetos possui um conjunto de bases de dados (repositórios), conforme apresentado na Figura 3, cuja responsabilidade é armazenar os artefatos de software desenvolvidos em cada domínio. Nota-se também em cada domínio a presença de engenheiros de software (ES), engenheiros de implementação (desenvolvedores) (EI), especialistas de domínio (ED) e agentes de software (AS). O ES, o EI e ED atuam na concepção, elaboração e reconfiguração de artefatos de software. Finalmente, os AS's são encarregados de realizar buscas por artefatos nos núcleos, local ou distribuído, conforme critérios iniciais de pesquisa estabelecidos. Seguindo a orientação da consulta, os agentes podem retornar com o artefato desejado, parcialmente desejado (adaptável com pequenas modificações), ou uma mensagem relatando que os critérios de pesquisa devem ser modificados (artefato não encontrado) [13] e [17].

Ainda sobre o retorno da pesquisa por artefatos no AER, os ESs e os EIs são os responsáveis pelo desenvolvimento de novos artefatos ou pela reutilização em projetos distintos para os quais foram concebidos. A reutilização de software pode ocorrer de duas maneiras: **(1) integral**, quando o artefato não sofre nenhuma alteração; **(2) parcial**, quando alguns mecanismos de modificação de estrutura ou comportamento são aplicados para que um novo artefato seja produzido [13].

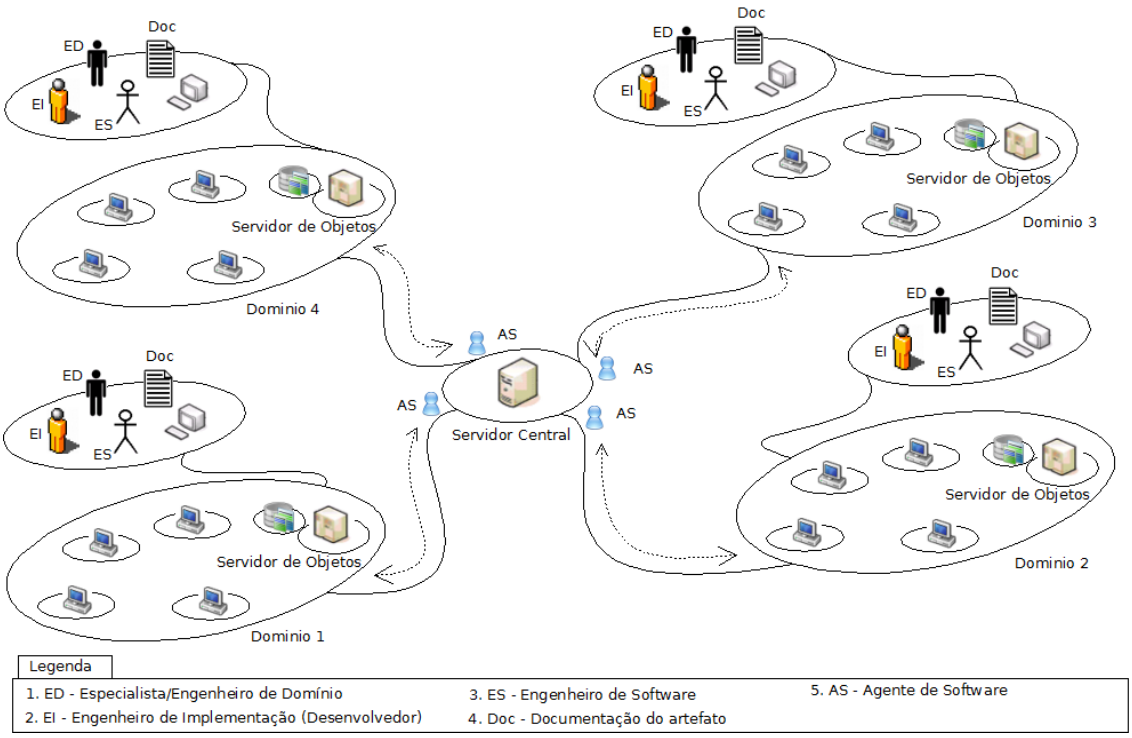


**Figura 4.** Ambiente de Execução Reconfigurável [13] e [17]

A seguir é apresentada a descrição da ferramenta ReflectTools e dos subsistemas de supervisão de objetos (reconfiguração) e consulta (repositórios de métodos remotos) como mecanismos de automatização dos passos da MDSR - apresentada nesta seção.

## 4 A Ferramenta ReflectTools e os Subsistemas

Incialmente pode-se dizer que a ferramenta ReflectTools foi projetada para automatizar os passos previstos na MDSR (especificamente o passo 6 da Figura 2 e todos os passos da Figura 3) e todas as atividades existentes no AER. Os passos 1 a 5 da Figura 2 podem ser executados nas ferramentas Eclipse [14] ou Netbeans *[15]*, conforme a familiaridade e experiência do engenheiro de software na configuração do ambiente - utilização de plugins para apoio à modelagem e ao desenvolvimento dos artefatos de software. Além disso, conforme apresentado nos trabalhos dos autores [8], [10], [11], [12] e [41], as ferramentas de engenharia de software aumentam a produtividade e minimizam as incertezas geradas pelos seres humanos (desenvolvedores). Todos esses conceitos e particularidades foram aplicados neste artigo para o contexto de desenvolvimento de software reconfigurável apoiado por ferramentas de automatização [13] e [17].

Para automatizar as atividades previstas na MDSR e no AER, a ferramenta ReflectTools implementa vários subsistemas. Neste artigo serão apresentados apenas dois por motivos de escopo - utilizados na seção estudo de caso para apresentação da ferramenta. O primeiro atua na reconfiguração de software (supervisor de objetos) e o segundo, no armazenamento e recuperação de informações de métodos remotos (repositório de métodos remotos), utilizadas no processo de construção de um novo artefato ou na reconfiguração de um já existente [17].

A reconfiguração de software é realizada, neste trabalho, por meio de uma técnica intrusiva, que tem a capacidade de analisar e modificar o conteúdo (código fonte) de um artefato de software. Esta técnica foi implementa em um subsistema supervisor de objetos, cujas responsabilidades podem ser resumidas da seguinte maneira: (1) monitorar as solicitações dos objetos clientes para os servidores; (2) interceptar as requisições e localizar no AER (repositórios de informação) um artefato que possa atender à solicitação do cliente; (3) realizar a reconfiguração do artefato localizado em tempo de execução [13] e [17]. A Figura 5 mostra uma visão geral desse subsistema organizado em pacotes.

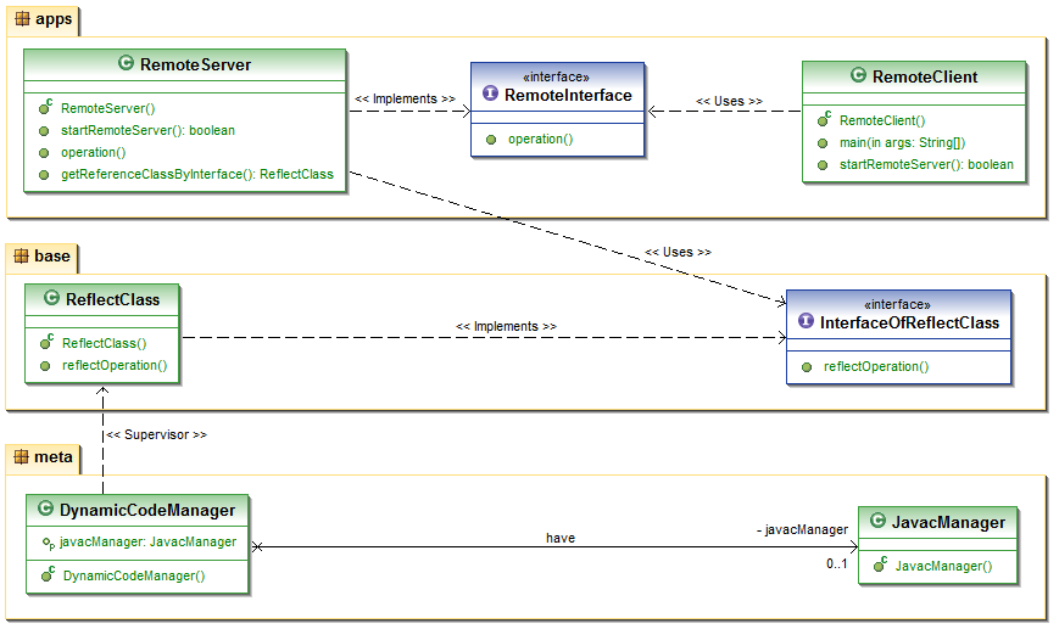


**Figura 5.** Modelo de classe do subsistema supervisor de objetos [13] e [17]

O pacote `meta` possui duas classes essenciais para a reconfiguração dos objetos, `DynamicCodeManager` e `JavacManager`, que são responsáveis por gerenciar a reconfiguração e a recompilação dos objetos em tempo de execução, respectivamente.

O pacote `base` é composto pela interface `InterfaceOfReflectClass` e pela classe `ReflectClass`, que representam, respectivamente, o "ponto" de acesso ao sistema pelos clientes e a aplicação a ser reconfigurada. Pode-se notar também a relação supervisora que a classe `DynamicCodeManager` exerce sobre a classe `ReflectClass` para a realização da reconfiguração da aplicação.

O pacote `apps` representa, em linhas gerais, as aplicações que podem consumir uma funcionalidade reconfigurável. A classe `RemoteServer` possui uma referência para a interface `InterfaceOfReflectClass` com a finalidade de utilizar a funcionalidade `reflectOperation`, que está implementada na classe `ReflectClass`. Dessa forma, pode-se dizer que a classe `RemoteServer` encapsula essa funcionalidade e disponibiliza para seus clientes na forma de uma operação `operation` transparente. Os clientes `RemoteClient` a utilizam como se fosse local e não têm a percepção quanto à distribuição e reconfiguração.

Ainda sobre o modelo apresentado na Figura 5, duas considerações devem ser destacadas. A primeira refere-se a como uma funcionalidade reconfigurável é criada. Nesta etapa o engenheiro de software faz a seleção da funcionalidade desejada, que, automaticamente, recebe um nome (`reflectOperation`) da ferramenta ReflectTools. As funcionalidades são criadas contendo parâmetros de retorno, nome do método e parâmetros de entrada conforme suas características específicas. A mesma regra é aplicada para a criação da funcionalidade `operation` na classe `RemoteServer`. A segunda consideração refere-se à administração de nomes das classes e interfaces presentes nos pacotes `apps` e `base`. Quando os artefatos são gerados pela ferramenta ReflectTools, é associado a esses nomes um número (identificador), que tem por objetivo oferecer garantia de unicidade de nomes na etapa de registro no servidor remoto (`rmiregistry`). Na seção 5, Figuras 10 e 12, é possível observar os nomes dos artefatos gerados.

A documentação e recuperação das informações para o desenvolvimento e/ou reconfiguração de artefatos de software em tempo de execução utilizam um conjunto de subsistemas, sendo cada um direcionado a uma abordagem de desenvolvimento. Neste artigo será detalhado o funcionamento de um subsistema para artefatos que utilizam como princípio de distribuição a chamada de métodos remotos [19]. A Figura 6 mostra o modelo de classe para subsistema de repositório de métodos remotos.

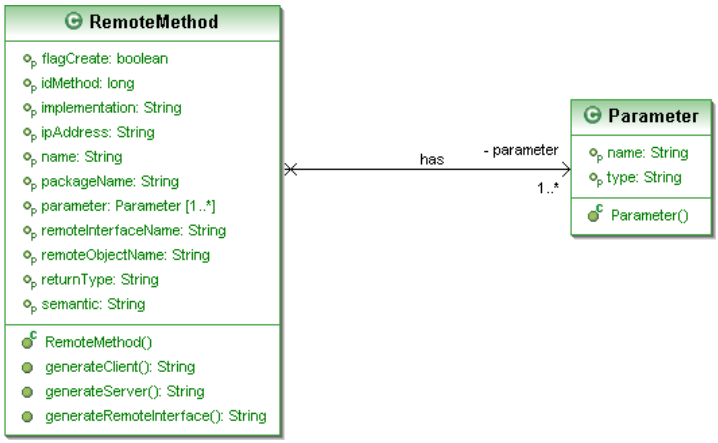


**Figura 6.** Modelo de classe para o subsistema de repositório de métodos remotos

Como se pode observar no modelo apresentado na Figura 6, um método remoto (classe `RemoteMethod`) é formado por muitos parâmetros (classe `Parameter`). As informações contidas nessas classes descrevem, basicamente, como um método remoto é implementado. Além disso, são utilizadas na restauração automática desse método no ambiente de execução da seguinte maneira: (1) o objeto é criado e salvo em local definido pela ferramenta ReflectTools; (2) compilado pelo subsistema supervisor de objetos; e (3) inserido/registrado no servidor de objetos (`rmiregistry`) com um nome de identificação que possui garantia de unicidade.

Ainda sobre o modelo apresentado na Figura 6, destaca-se o atributo (`semantic`), que é utilizado como fonte de informações descritivas para qualificar a finalidade de execução desse método remoto no subsistema de consulta. Sobre a recuperação das informações, vale mencionar que os critérios de pesquisa são considerados "exatos", ou seja, as informações fornecidas pelos engenheiros de software/especialista de domínio ou agentes de software na atividade de reconfiguração manual/automática são comparadas sem um critério analítico. O resultado da consulta é puramente binário, satisfazendo ou não dos dados de pesquisa.

Em trabalhos anteriores dos autores [13], [17] e [46] é possível observar que a justificativa para a implementação deste subsistema está apoiada na necessidade de avaliar o mecanismo de reconfiguração de artefatos remotos (métodos remotos - Figura 6). Os autores comentam ainda que outro fator que motivou esta implementação é a complexidade de projeto e implementação de um subsistema de consulta capaz de qualificar em escala a satisfação dos artefatos encontrados em relação aos critérios de iniciais estabelecidos.

Os detalhes de funcionamento dos subsistemas de supervisão de objetos e consulta de métodos remotos, apresentados nesta seção, são abordados no desenvolvimento do estudo de caso (seção 5), assim como as funcionalidades da ferramenta ReflectTools, que faz uso desses subsistemas. Sobre a ferramenta, nem todas as funcionalidades são apresentadas neste artigo, sendo destacadas apenas aquelas que contribuem para a demonstração desses subsistemas no estudo de caso escolhido - reconfiguração de artefatos utilizando chamada de métodos remotos. Detalhes mais refinados podem ser encontrados em trabalhos anteriores dos autores [13], [17] e [46].

A seguir são apresentadas, de maneira descritiva, as principais funcionalidades da ferramenta ReflectTools [13]. São elas:

- Capacidade de integração somente com as ferramentas de engenharia de software (Eclipse [14] e Netbeans [15]) para importação/integração de projetos de Código fonte Java; Scripts SQL; e, Documentação JAVADOC (HTML - HyperText Markup Language).
- Capacidade de conectividade com banco de dados (JDBC - Java Database Connectivity):
    - Execução de scripts SQL DDL para criação das bases de dados dos artefatos: Local - no mesmo domínio; e Remota - em domínios distintos;
    - Gerenciamento e administração de repositórios (bases de dados internas dos subsistemas nela implementados);
    - Os scripts suportados (testados) pela ferramenta ReflectTools são para os bancos de dados (bases)

MySQL [47] e PostgreSQL [48], pois apenas os drivers para essas bases foram implementados na ferramenta. Para utilização de outras bases, o engenheiro de software deve selecionar um plugin de modelagem e geração de código para a base desejada, além de fornecer o driver correto de conexão JDBC.

- Administração de projetos:
  - Armazenamento e recuperação de informações dos projetos nos repositórios de informações (importação de projetos e organização nos repositórios específicos);
  - Armazenamento e recuperação de funcionalidades através de parâmetros: Tipo de retorno, Nome da funcionalidade e Parâmetros de Entrada e Saída dos métodos (Subsistema de repositório de métodos remotos e Outros);
  - Para armazenar e recuperar projetos e artefatos na ferramenta ReflectTools foram utilizando indexadores em autoincremento, associados com nomes, e/ou datas, e/ou informação semântica. Destaca-se que um processo mais sofisticado de armazenamento e recuperação de informação está em fase de estudo e implementação pelo autor (trabalho futuro) [13].
- Gerenciamento da reconfiguração:
  - Subsistema supervisor de objetos;
  - Criação, Compilação e Implantação de artefatos de software remotos (métodos remotos) em servidores de objetos remotos (`rmiregistry`) [19];
  - Criação, Compilação e Implantação de Serviços no servidor Web. Esta etapa é apoiada pela ferramenta Netbeans [15], porém, o acoplamento do sistema supervisor de objetos é realizado pela ferramenta ReflectTools;

A ferramenta ReflectTools e, consequentemente, a MDSR e o AER possuem uma limitação quanto à linguagem de programação. Apesar de oferecerem suporte a várias abordagens de desenvolvimento (objetos, componentes, aspectos e/ou serviços), apenas a linguagem de programação Java [16] pode ser utilizada no desenvolvimento e, exclusivamente, na reconfiguração dos artefatos.

A seguir é apresentado um estudo de caso para mostrar a utilização da ferramenta ReflectTools na localização e utilização de uma funcionalidade no formato de um método remoto.

## 5 Estudo de Caso

O sistema considerado como estudo de caso pode ser categorizado como um sistema de informação para uma loja virtual. Os passos de modelagem e implementação dos artefatos, ferramentas (Eclipse [14] e/ou Netbeans [15]), são omitidos nesta seção, pois demandaria uma grande quantidade de espaço (telas) para sua apresentação. Assim, pode-se dizer que os passos 1 a 5 da Figura 2 foram executados e o projeto do sistema pode ser importado para a ferramenta ReflectTools (passo 6 da Figura 2). Os passos 1 a 6 da Figura 3 são apresentados nesta seção. Este estudo de caso tem por objetivos mostrar:

- Como uma funcionalidade, com potencialidade para reconfiguração, pode ser identificada, inserida nos repositórios de informação e recuperada para ser utilizada como uma funcionalidade remota reconfigurável. Além disso, mostrar seu acoplamento e utilização através da ferramenta (avaliação dos subsistemas supervisão de objetos e repositório de métodos remotos);
- A execução da ferramenta ReflectTools, se ela cumpre a finalidade de apoiar o desenvolvimento de software reconfigurável;

A lógica do sistema "loja virtual" é composta por 18 classes lógicas. Essa informação é obtida visualmente na ferramenta ReflectTools na árvore de projeto. O passo seguinte é realizar, pelo engenheiro de software, uma varredura nas classes do sistema para que funcionalidades, com potencialidade de reconfiguração, sejam identificadas. Essa atividade é feita de maneira manual e contínua, ou seja, desde a concepção do sistema, o engenheiro de software é responsável por projetar artefatos com capacidade de reutilização e reconfiguração - conforme diretrizes estabelecidas na MDSR.

Para um sistema de uma loja virtual é necessário que algumas informações sejam validadas desde a entrada de dados ou até mesmo na lógica da aplicação - para comprovar que determinada informação seja fidedigna. Adotando esse contexto como uma situação real dos sistemas, o engenheiro de software identifica na classe `Cliente` o método `validarCpf`. Esse método pode ser considerado como uma funcionalidade "genérica" com a capacidade de reutilização em outros sistemas. Além disso, possui potencialidade para reconfiguração, pois neste sistema outras informações podem ser validadas, como, por exemplo, o CNPJ (Cadastro Nacional da Pessoa Jurídica) de um fornecedor de material para a loja virtual. A funcionalidade remota, quando reconfigurada, continua fazendo validação, porém de outro tipo de informação.

Na ferramenta ReflectTools, o engenheiro de software deve selecionar o arquivo (`Cliente.java`) na árvore do projeto para acessar a funcionalidade (método `validarCpf`). A partir deste arquivo, as informações são selecionadas com a finalidade de serem enviadas para os repositórios (Send Method to Repository), conforme mostra a Figura 7.

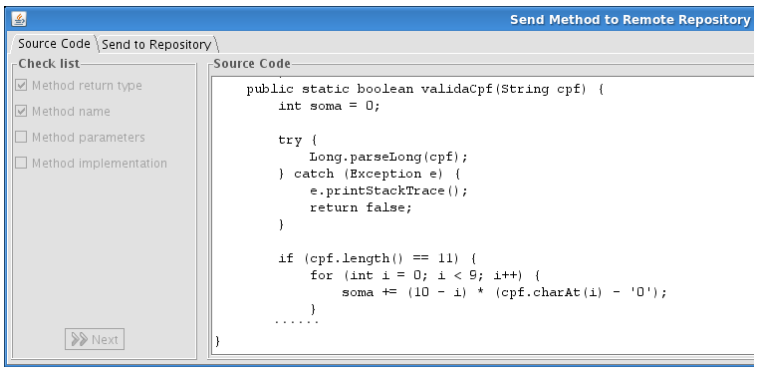


**Figura 7.** Enviar método para repositório (Coletando informações)

Ainda sobre a Figura 7, no canto superior esquerdo é possível observar duas abas (Source Code e Send to Repository). A primeira representa todo processo de extração de informação, em que o engenheiro de software deve preencher as informações técnicas (Method return type, Method name, Method parameters e Method implementation), que descrevem o método remoto - conforme modelo apresentado na Figura 6. Esse processo é realizado através da seleção direta do código fonte, manipulação de menus flutuantes e operações de interface (opções disponíveis no Check list).

Ao preencher todas as informações técnicas (Source Code) é possível inserir as informações semânticas, que descrevem a finalidade de execução dessa funcionalidade (método `validarCpf`). Essa atividade é realizada, de maneira cooperativa, pelo especialista de domínio e pelo engenheiro de software, conforme previsto nos passos da MDSR. Em seguida, a funcionalidade pode ser armazenada nos repositórios de informações (Send to Repository) para ser reutilizada no desenvolvimento ou reconfiguração dos artefatos de software (novos ou existentes). A Figura 8 mostra a interface para criação do método remoto no repositório.

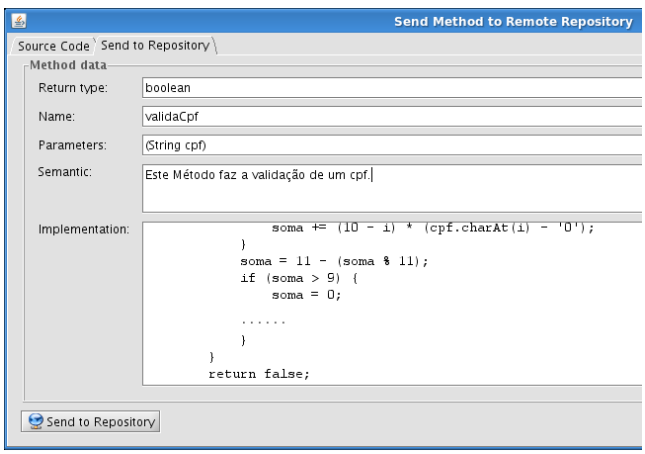


**Figura 8.** Enviar método para repositório (Inserindo informações nos repositórios remotos)

A atividade descrita (Figura 8) instancia o modelo do subsistema de repositório de métodos remotos, apresentado na Seção 4 - Figura 6. A ferramenta ReflectTools faz uma análise local para verificar se as informações técnicas (Method return type, Method name, Method parameters e Method implementation) e semânticas foram preenchidas corretamente antes de instanciar esse modelo. As demais informações (indexadores para garantia de unicidade e outras) presentes no modelo são preenchidas automaticamente e gerenciadas pelo subsistema, onde a aplicação se encontra hospedada.

As informações geradas nesta etapa, envio de funcionalidades comuns (reutilizáveis) para os repositórios, constituem uma base de conhecimento ampla que pode ser utilizada no desenvolvimento de novos artefatos e na reconfiguração dos existentes. A seguir são apresentados os aspectos relacionados às consultas e à geração automática de artefato (objetos remotos) em tempo de execução.

O procedimento de consulta nos repositórios por métodos remotos segue uma orientação de equivalência "exata". Este tipo de consulta não prevê a utilização de sinônimos ou palavras correlatas que representam um significado semelhante. Dessa forma, julga-se necessária a presença do especialista de domínio e do engenheiro de software para (1) especificar as características dos artefatos a serem pesquisados e (2) avaliar os resultados obtidos com a execução da consulta. Para realizar uma consulta, a ferramenta disponibiliza um wizard (Search for new software artefact), como mostra a Figura 9.

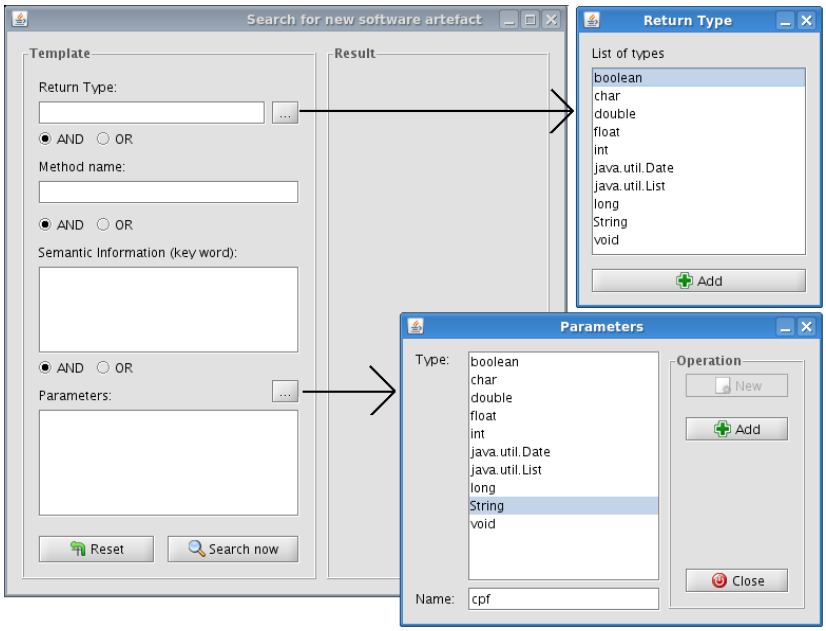


**Figura 9.** Wizard de consulta de artefatos remotos

Neste wizard, Figura 9, o engenheiro de software e o especialista de domínio devem preencher as informações (área Template) para o artefato que desejam localizar, tais como tipo de retorno, parâmetros e informações semânticas. Em seguida, solicitam que a ferramenta realize a pesquisa (Search Now) nos repositórios locais e distribuídos, conforme apresentado na Figura 4. Caso nenhum artefato seja encontrado, uma mensagem de aviso é exibida, solicitando a modificação dos critérios de busca (informações técnicas ou semânticas); em caso contrário, um ou mais artefatos são exibidos na área (Result), como mostra a Figura 10.

Na área *Result*, Figura 10, é possível observar a funcionalidade remota localizada pela interface (`RemoteInterface_9`). O nome deste arquivo é formado pelo nome das classes/interfaces definidas no modelo apresentado na Figura 5, seguido de um indexador que garante a unicidade daquela funcionalidade no AER. Caso mais de uma funcionalidade seja localizada (enquadrada nos critérios de pesquisa inicialmente estabelecidos), as setas navegacionais podem ser utilizadas para visualização do código fonte e avaliação da funcionalidade. Através deste wizard é possível salvar os arquivos clientes (local) e servidores (servidor remoto) antes de adicioná-los ao sistema que está sendo desenvolvido ou que se pretende reconfigurar. A Figura 11 mostra a estrutura de comunicação entre os clientes (ferramenta ReflectTools) e subsistemas de gerenciamento de métodos remotos - hospedados nos diferentes domínios.

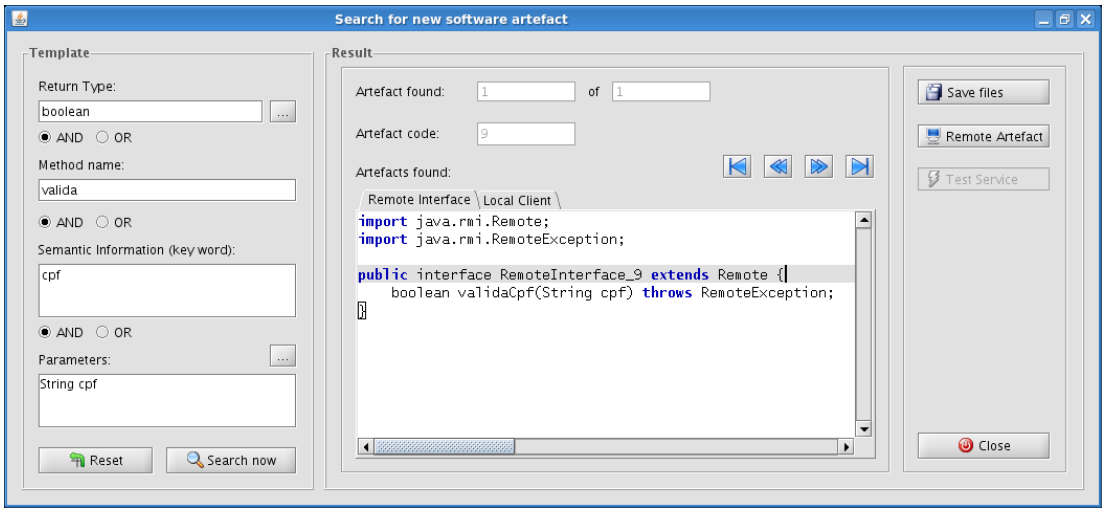


**Figura 10.** Resultado da pesquisa no repositório de método

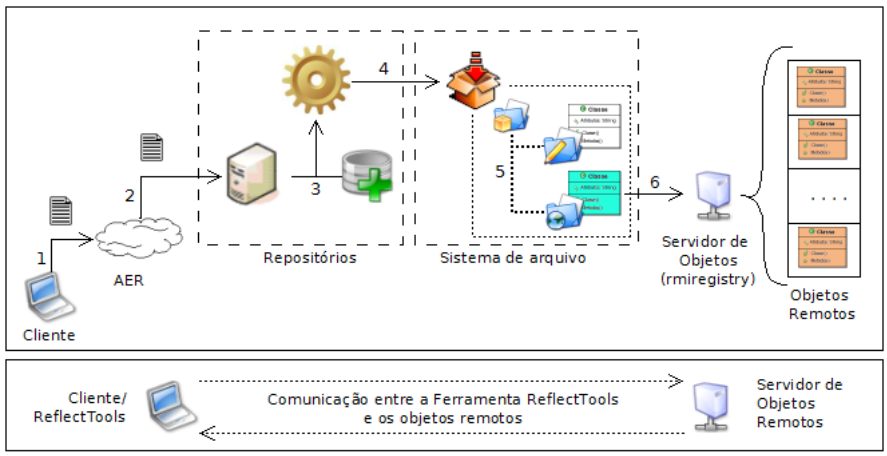


**Figura 11.** Estrutura de comunicação entre ferramenta ReflectTools e o Servidor de Objetos Remotos

Após realizar a consulta (**passos 1, 2 e 3 da Figura 11**), a ferramenta exibe os resultados - conforme apresentado na Figura 10. Nesse momento ocorre a comunicação entre os clientes (ferramenta ReflectTools) e os servidores de objetos remotos, que é realizada em duas etapas.

Na primeira, o engenheiro de software solicita a opção (Save files - Figura 10) para salvar os arquivos da interface remota e da aplicação cliente do objeto remoto. Os arquivos referentes aos artefatos (interface e aplicação cliente remotos) são criados e salvos em diretórios definidos na ferramenta ReflectTools (lado cliente).

Na segunda, o engenheiro de software solicita a opção (Remote Artefact - Figura 10) para salvar os arquivos servidores (interface remota e aplicação servidora de objetos), também em diretórios definidos no Servidor de Objetos Remotos. Em seguida, automaticamente, os arquivos são compilados, o servidor de objetos remoto (`rmiregistry`) é inicializado e o objeto remoto (`RemoteServer`) é nele registrado com um nome de identificação (**passos 4, 5 e 6 da Figura 11**). Os nomes gerados automaticamente pela ferramenta ReflectTools ou pelos Servidores de Objetos Remotos no AER seguem a mesma regra de implementação apresentada na Figura 10 para a interface (`RemoteInterface_9`).

Após a realização dessas duas etapas, preparação dos artefatos (aplicação cliente e objeto servidor) - Figura 10, a opção Test Service é habilitada. Ao selecionar esta opção, o engenheiro de software irá visualizar um wizard, Figura 12, para testar a comunicação entre o objeto remoto, presente no Servidor de Objetos Remotos, e a aplicação cliente, presente na ferramenta ReflectTools.

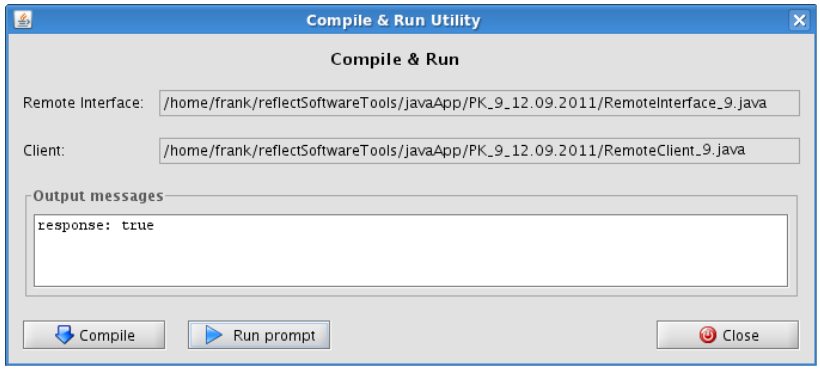


**Figura 12.** Resultado de execução

Selecionando a opção Compile e, em seguida, Run prompt, o engenheiro de software compila e executa a aplicação cliente, criando o processo de comunicação entre a ferramenta ReflectTools e o Servidor de Objetos Remotos (parte inferior da Figura 11).

Para finalizar a descrição deste estudo de caso são apresentadas algumas considerações sobre a utilização do subsistema supervisor de objetos (apresentado na Seção 4 - Figura 5) e, principalmente, como a aplicação é reconfigurada. Ao criar a aplicação servidora, conforme apresentado nesta seção, este subsistema é automaticamente acoplado à aplicação servidora (`RemoteServer`) pela ferramenta ReflectTools; a partir desse momento, todas as solicitações de execução (clientes) passam a ser monitoradas por esse subsistema.

Quando o engenheiro de software realizar uma nova pesquisa de artefatos, a ferramenta ReflectTools analisará o servidor de objetos remotos (`rmiregistry`) para verificar se existe algum objeto em execução que pode atender à solicitação (critérios de pesquisa). Seguindo os passos apresentados nesta seção, o engenheiro de software pode optar pela reconfiguração do artefato existente ou pelo desenvolvimento de um novo. Optando pela primeira, a ferramenta faz, inicialmente, uma verificação para certificar-se de que este artefato não está sendo referenciado por outra aplicação. Neste momento duas situações podem ser destacadas:

1. Não havendo referência para o artefato, seu corpo de método é modificado pelo engenheiro de software com a utilização da ferramenta ReflectTools (edição do código fonte). Desse ponto em diante a ferramenta se encarrega de compilar e substituir o artefato em memória (acesso direto ao classload da JVM). Os nomes são mantidos e as informações que o descrevem são modificadas no AER;
2. Havendo a referência por outras aplicações, a ferramenta preservará a execução vigente e um novo artefato será criado com conteúdo e nomes distintos.

Sobre a segunda situação destaca-se ainda a solução oferecida pela ferramenta ReflectTools quando artefatos mais complexos são reconfigurados. Por exemplo, havendo a referência e a necessidade de adição de atributos ou métodos aos artefatos, a ferramenta implementa a técnica de criação de novos artefatos por herança, ou seja, as modificações desejadas são adicionadas sempre na classe filha. Para preservar os artefatos criados, tanto o artefato pai quanto o filho são armazenados como um novo artefato, pois existem clientes com necessidades distintas e que são consumidores das funcionalidades que ambos disponibilizam. Maiores detalhes sobre a técnica de reconfiguração podem ser obtidos em trabalhos anteriores do autor [13], [17] e [46].

## 6 Considerações Finais

Este artigo apresentou uma metodologia, um ambiente e, em destaque, a ferramenta ReflectTools [13] para apoio ao desenvolvimento de software reconfigurável. Baseado na classificação e nos conceitos apresentados pelos autores [8], [9], [10], [11], [12], [38], [39] e [44] em seus trabalhos sobre ferramentas de engenharia de software, pode-se dizer que a ferramenta ReflectTools é factível no contexto independência, pois recebe os artefatos de software de outras ferramentas como *Eclipse* [14] e Netbeans [15] e cumpre seu propósito em: (1) automatizar um conjunto de atividades previstos na MDSR, minimizando a participação humana nas etapas de desenvolvimento e reconfiguração; e (2) oferecer um ambiente de execução reconfigurável, não necessitando de outras ferramentas complementares para isso.

Ainda, sobre o contexto independência de uso, vale destacar os subsistemas nela implementados. Neste artigo foram enfatizados os subsistemas supervisor de objetos e repositório de métodos remotos, assim como o processo de comunicação entre o ambiente de execução reconfigurável e a ferramenta ReflectTools. Os subsistemas atuam de maneira transparente para engenheiros de software e desenvolvedores, minimizando sua participação na codificação para características reconfiguráveis de software. Pode-se também notar a presença do engenheiro de software e do especialista de domínio apenas na interpretação final dos resultados: (1) quando um novo artefato de software está sendo desenvolvido; (2) quando existe a necessidade de escolha de um artefato, conforme apresentado na seção 5.

No contexto desenvolvimento, destaca-se também a padronização que deve ser adotada nas fases de concepção e elaboração dos sistemas reconfiguráveis (MDSR). A ferramenta ReflectTools permite que funcionalidades com potencialidades de reutilização em outros sistemas sejam identificadas e armazenadas em repositórios. Em seguida, podem ser reutilizadas como métodos remotos ou serviços (Web Services) no projeto de outros sistemas, aumentando a velocidade de desenvolvimento de um novo artefato ou contribuindo, de maneira significativa, como base de informação para o processo de reconfiguração manual ou automática [13] e [46].

Ainda sobre o contexto desenvolvimento, segundo Spanjers [12], existe uma grande dificuldade em realizar o desenvolvimento de software distribuído devido às localizações geográficas. Conforme apresentado na Seção 3, o AER é formado por vários domínios de atuação (desenvolvimento) e sua proposta é mostrar que uma organização (privada ou pública) pode utilizar a estrutura de desenvolvimento de software reconfigurável (MDSR, o AER e a ferramenta ReflectTools) e alcançar seus benefícios, utilizando-os em: (1) projetos locais como uma intranet; (2) projetos distribuídos em localidades mais abrangentes (rede privada - intranet e/ou internet). A principal justificativa para que esses benefícios sejam alcançados é a padronização de desenvolvimento imposta pela MDSR nas etapas de desenvolvimento e pela automatização dessas atividades do processo existente na MDSR, no AER e na ferramenta ReflectTools.

Outro fator relevante ao contexto desenvolvimento e discutido por alguns autores [8], [9], [39] e [40] é a definição da arquitetura de software para o aumento da capacidade de reúso de software. Para isso, comentam sobre o uso de repositórios como mecanismo de armazenamento e reutilização da informação. Neste artigo, pode-se observar, claramente, a utilização dos conceitos de repositórios de software para o armazenamento/recuperação de métodos remotos. Além disso, em trabalhos anteriores dos autores da MDSR [13] e [17], os repositórios são utilizados no desenvolvimento/reutilização de software em várias abordagens: orientada a objetos[38] e [44]; orientada a componentes [27] e [44]; orientada a aspectos [25], [26] e [44] e orientada a serviços [44], [49] e [50]. Em Affonso & Rodrigues [46], por exemplo, é possível encontrar reconfiguração de software aplicada em sistemas orientados a serviços web e orientados a chamada de métodos remotos.

Para finalizar, pode-se dizer que a combinação entre as ferramentas (Eclipse [14] e Netbeans [15]) e a ferramenta ReflectTools [13] oferece suporte ao desenvolvimento de software reconfigurável em todas as fases do ciclo de vida de software. É essa uma das maiores preocupações de alguns autores ([8], [9], [10] [11] e [41]) e apontadas como fator negativo quando as ferramentas não cumprem para o sucesso de um projeto de software.

## Referências


[1] MAES, P. Concepts and experiments in computational reflection. ACM SIGPLAN Notices, Orlando: ISSN:0362-1340, 1987;

[2] FORMAN, I. R. & FORMAN, N. Java reflection in action. ISBN: 1932394184. 1 ed. United State: Manning Publication, 2004;

[3] COULSON, G. et. al. On the performance of reflective systems software, IEEE International Conference on Performance, Computing, and Communications, doi: 10.1109/PCCC.2004.1395177, pp. 763- 769, 2004;

[4] LEE, Y. F. & CHANG, R. C. Java-based component framework for dynamic reconfiguration, Software IEEE Procedings, ISSN: 1462-5970, 2005;

[5] TANTER, E. et. al. Flexible metaprogramming and aop in java. Science of Computer Programming, pp: 22-30, 2008;

[6] HMIDA, M.M.B. et. al. Applying aop concepts to increase web services flexibility. In International Conference on Next Generation Web Services Practices, 2005;

[7] CHEN, X. Extending rmi to support dynamic reconfiguration of distributed systems. In Distributed Computing Systems, 2002;

[8] WHITEHEAD, J. Collaboration in software engineering: a roadmap. FOSE '07 - Future of Software Engineering, 2007, pp.214-225, doi: 10.1109/FOSE.2007.4, 2007;

[9] NAKAGAWA, E. Y, Uma contribuição ao projeto arquitetural de ambientes de engenharia de software, Tese de Doutorado ICMC-USP, 2006;

[10] TOTH, K. Experiences with Open Source Software Engineering Tools. IEEE Software, 2006, vol. 23, no. 6, pp.44-52, doi: 10.1109/MS.2006.158, 2006;

[11] GRAY, J. Software engineering tools. Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 2000, pp. 3300- 3301, doi: 10.1109/HICSS.2000.927015, 2000;

[12] SPANJERS, H. Tool Support for Distributed Software Engineering, ICGSE '06. International Conference on Global Software Engineering, 2006, pp.187-198, doi: 10.1109/ICGSE.2006.261232, 2006;

[13] AFFONSO, F. J. Metodologia para desenvolvimento de software reconfigurável apoiada por ferramentas de implementação: uma aplicação em ambiente de execução distribuído e reconfigurável, Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2009;

[14] ECLIPSE. Site oficial do projeto eclipse. Disponível em: <http://www.eclipse.org>. Acessado em: 29 de jun de 2011;

[15] NETBEANS. Site oficial do projeto netbeans. Disponível em: <http://netbeans.org>. Acessado em: 09 de jun de 2011;

[16] ORACLE-JAVA. Site oficial da oracle. Disponível em: <http://www.oracle.com/technetwork/java/javase/overview/index.html> Acessado em: 14 de set. 2011;

[17] AFFONSO, F. J. & RODRIGUES, E. L. L. Metodologia para Desenvolvimento de Software para um Ambiente Reconfigurável. Revista Multiciência, Volume 7, 2007;

[18] AKKAWI, F. et. al. Software Adaptation: A Conscious Design for Oblivious Programmers, IEEE Aerospace Conference, 2007, pp.1-12, doi: 10.1109/AERO.2007.352967, 2007;

[19] ORACLE-RMI. Site oficial da oracle. Disponível em: <http://download.oracle.com/javase/tutorial/rmi/index.html>. Acessado em: 29 de jun. 2011;

[20] FERNANDES, A. P. & LISBÔA, M. L. B. Reflective Implementation of an object recovery design pattern. VII Congresso Argentino de Ciências de la Computación. El Calafate, República Argentina, 2001;

[21] FERNANDES, A. P. Reflexão computacional. Disponível em: <http://attila.urcamp.tche.br/~acauan/art_ccei_rc.html> Acessado em: 29 de jun. 2011;

[22] SILVA, F. J. S. Adaptação dinâmica de sistemas distribuídos. Tese de Doutorado. IME-USP, São Paulo-SP, 2003;

[23] STELTING, S. & MAASSEN, O., Applied java patterns. ISBN: 0-13-093538-7. 1 ed. California: Sun Microsystems Press, 2002;

[24] LISBÔA, M. L. B. Reflexão computacional no modelo de objetos. In Tutorial II SBLP, Campinas, São Paulo, Setembro, 1997;

[25] KICZALES, G., et. al. (1997), Aspect-oriented programming, 15th European Conference on Object Oriented Programming (ECOOP). Springer. June 1997;

[26] KICZALES, G., et. al. (2001), An overview of aspectj. 15th European Conference on Object Oriented Programming (ECOOP). Springer. June 2001;

[27] D'SOUZA, D. F., WILLS, A. C. Objects, components, and frameworks with UML: the catalysis(sm) approach. 1 ed. ed. Addison-Wesley Object Technology Series. 1998;

[28] HEINEMAN, G. T. & COUNCILL, W. T. Component-Based Software Engineering: Putting the Pieces Together, Addison-Wesley Professional, 1a ed., ISBN-10: 076868207X, 2001;

[29] BORDE, E. et al. Mode-based reconfiguration of critical software component architectures, in Europe Conference & Exhibition Design, Automation & Test, 2009;

[30] GOMAA, H. & HUSSEIN, M., Software reconfiguration patterns for dynamic evolution of software architectures, in Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture, 2004;

[31] WEISS, G. M. Adaptação de Componentes de Software para o Desenvolvimento de Sistemas Confiáveis. Dissertação de Mestrado. IC-UNICAMP, Campinas-SP 2001;

[32] JANIK, A. & ZIELINSKI, K. , AAOP-based dynamically reconfigurable monitoring system, in Information and Software Technology, 2010;

[33] KIM, J. A. et. al. Component adaptation using pattern components. In Systems, Man, and Cybernetics, Tucson, AZ, USA, 2001;

[34] BASTIDE, G. et. al. Dynamic Adaptation of Software Component Structures, IEEE International Conference on Information Reuse and Integration, 2006;

[35] KASTEN, E. P. et. al. Separating introspection and intercession to support metamorphic distributed systems. In Distributed Computing Systems Workshops, 2002;

[36] RICHMOND, M., NOBLE, J. Reflections on remote reflection. In Computer Science Conference, ACSC, 2001;

[37] ORACLE-REFLECT. Site oficial da oracle Disponível em: <http://download.oracle.com/javase/tutorial/reflect/index.html> Acessado em: 29 de jun. 2011;

[38] PRESSMAN, R., Engenharia de Software, 6ª edição, ISBN: 8586804576, Editora Mc Graw Hill, 2006;

[39] DONG, J. et. al. The Research of Software Product Line Engineering Process and Its Integrated Development Environment Model. ISCSCT '08 - International Symposium on Computer Science and Computational Technology, 2008. vol.1, pp.66-71, doi: 10.1109/ISCSCT.2008.100, 2008;

[40] HALLSTEINSEN, S. et. al. Dynamic Software Product Line, Computer, ISSN: 0018-9162, 2008;

[41] HENTTONEN, K. & MATINLASSI, M. Open source based tools for sharing and reuse of software architectural knowledge. Joint Working IEEE/IFIP Conference on Software Architecture, 2009 & European Conference on Software Architecture, 2009, pp.41-50, doi: 10.1109/WICSA.2009.5290790, 2009;

[42] ESTUBLIER, J. et. al. Domain Specific Engineering Environments, Software Engineering Conference, doi: 10.1109/APSEC.2008.16, pp.553-560, December, 2008;

[43] SHI, Y. et. al. A Reflection Mechanism for Reusing Software Architecture, Sixth International Conference on Quality Software, doi: 10.1109/QSIC.2006.5, pp.235-243, October, 2006;

[44] SOMMERVILLE, I. Engenharia de Software, 9ª edição, ISBN: 978-85-7936-108-1, Editora Pearson, 2011;

[45] JUNIT.ORG. Site oficial do framework de teste unitário junit. Disponível em: <http://www.junit.org/> Acessado em: 14 de set. 2011;

[46] AFFONSO, F. J. & RODRIGUES, E. L. L. Estudo comparativo da adaptação de software utilizando Chamada de Métodos Remotos e Serviços Web. Revista de Sistemas de Informação da FSMA. ISSN: 1983-5604, Volume 7, pp 22-31 2011;

[47] MYSQL. Site oficial do banco dados mysql. Disponível em: <http://www.mysql.com/products/community/> Acessado em: 15 de set. 2011;

[48] POSTGRESQL. Site oficial do banco dados postgresql. Disponível em: <http://www.postgresql.org/> Acessado em: 15 de set. 2011;

[49] SINGH, I. et al. Projetando web services com a plataforma j2ee 1.4. tecnologias jax-rpc, soap e xml. Ed. Ciência Moderna, Rio de Janeiro, 2006;

[50] THOMAS, E. Service-oriented architecture. 1 ed. ed. Prentice Hall. 2004;