Revista de Sistemas de Informação da FSMA
n. 7 (2011) pp. 22-31

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

TRILHA PRINCIPAL

Estudo comparativo da adaptação de software utilizando Chamada de Métodos Remotos e Serviços Web

Frank José AFFONSO, Evandro Luís Linhari RODRIGUES.

***Resumo* - O processo de desenvolvimento de software ao longo dos anos vem sendo direcionado a várias metodologias e, cada uma delas, com propósitos específicos para atender as necessidades emergentes. Durante esses anos, tem-se notado também que algumas delas buscam mecanismos para a reutilização de software e maior velocidade na etapa de desenvolvimento. Um fator importante neste contexto é a mutação (adaptação) que ocorre com o software ao longo de seu ciclo de vida, seja por necessidades de seus clientes ou por mudanças tecnológicas atuais. Sobre este último, tem-se observado um aumento significativo no desenvolvimento de aplicações distribuídas através da WWW, *World Wide Web*, ou remotas. Embasado na ideia de adaptação de software e na necessidade de distribuição dos sistemas, este artigo apresenta uma técnica de reconfiguração de software, capaz de atuar em vários contextos de desenvolvimento (Local, Distribuído e/ou Web). Para demonstrar sua aplicabilidade, foi elaborado um estudo de caso, cujo objetivo é mostrar a adaptação de software no desenvolvimento de uma aplicação utilizando as abordagens: orientação a serviços e chamadas remotas. Além disso, são apresentados os resultados comparativos entre as abordagens para o desenvolvimento de uma aplicação reconfigurável.**

***Index Terms* - reconfiguração de software, *Web* Service, chamada remota, reutilização.**

I. INTRODUÇÃO

UMA das características desejadas pelos engenheiros de software é a "habilidade" de adaptação do software em tempo de execução [7, 9, 18, 21, 27]. Essa habilidade está associada ao atendimento das novas necessidades dos clientes e/ou mudanças na tecnologia de software. Apoiado no primeiro fator e considerando a tendência de desenvolvimento de software distribuído, este artigo apresenta um estudo comparativo da adaptação de software utilizando uma técnica de reconfiguração capaz de atuar em sistemas desenvolvidos com Serviços Web (*Web Service* - *WS*) [25, 26] e com Chamada de Métodos Remotos (*Remote Method Invocation* - RMI) [20].

A reconfiguração de software quando realizada de maneira local exige alguns cuidados de seus gerenciadores, no entanto, em aplicações distribuídas, estes cuidados devem aumentar consideravelmente, pois vários objetos podem ser acessados por diferentes clientes ao mesmo tempo [18, 27]. Por isso, antes de realizar qualquer modificação na aplicação, seu estado deve ser salvo para que as execuções anteriores sejam preservadas [19].

Para viabilizar a reconfiguração em tempo de execução [7, 8, 9, 22], objetos tutores, por meio de um sistema supervisor de objetos, são acoplados à aplicação para realizar o monitoramento e a reconfiguração das funcionalidades. Sendo assim, este artigo tem por objetivo mostrar que a técnica de reconfiguração elaborada pode se acoplada, de maneira transparente, nas aplicações servidoras - independentemente da tecnologia de distribuição (WS ou RMI) que se pretende utilizar.

Este artigo está organizado da seguinte maneira: na seção II são apresentados os trabalhos relacionados para o desenvolvimento deste artigo; na seção III é apresentada a estrutura da aplicação; na seção IV é apresentado o mecanismo de reconfiguração de software em tempo de execução; na seção V são apresentadas as discussões dos resultados sobre a adaptação de software utilizando WS e RMI; e, finalmente, na seção VI, as considerações finais.

II. TRABALHOS RELACIONADOS

Uma alternativa ao processo de adaptação é a Reflexão

Frank José AFFONSO, Professor Assistente Doutor, Universidade Estadual Paulista - Júlio de Mesquita Filho, Departamento de Estatística, Matemática Aplicada e Computação, Avenida 24A, 1515 - 13506-900 - Rio Claro-SP/Brasil, e-mail: frank@rc.unesp.br.

Evandro Luis Linhari RODRIGUES, Professor Livre-Docente, Universidade de São Paulo – USP, Departamento de Engenharia Elétrica, Avenida Trabalhador São-carlense, 400 - 13566-590 - São Carlos-SP/Brasil, e-mail: evandro@sc.usp.br.

Computacional (RC) [7, 8, 24], que pode ser definida como qualquer atividade executada por um sistema sobre si mesmo. Seu 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 [18] *apud* [7].

A RC permite que um sistema monitore seu contexto atual e, a partir desse monitoramento, decida qual procedimento será executado. Essa característica mostra a flexibilidade de modificação de um sistema desenvolvido com reflexão computacional [7, 8, 9, 18, 22].

No modelo de computação reflexiva, Figura 1, é apresentado o fluxo de informações entre objetos e metas-objetos. Um meta-objeto, pertencente ao meta-nível, é responsável por (1) interceptar a solicitação de um objeto do nível-base e (2) encaminhar esta para outro meta-objeto do meta-nível. Este, por sua vez, (3) localizará um objeto do nível-base para que a requisição original seja atendida. O caminho inverso (4, 5 e 6) também é considerado [8].

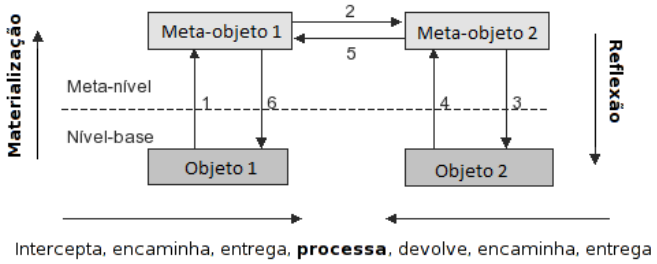

Fig. 1. Comunicação entre objetos e metas-objeto [8]

A seguir serão apresentadas algumas técnicas de reconfiguração de software encontradas na literatura que contribuíram para o desenvolvimento da proposta de reconfiguração apresentada neste artigo (Figura 5).

Em [27] a RC é utilizada na adaptação de componentes de software [6] de duas maneiras: **(1) estrutural**, que está relacionada à estrutura do componente, tais como: alteração na interface de comunicação ou no nome de uma operação; e, **(2) comportamental**, que afeta as propriedades funcionais e não funcionais [15, 16]. Essa técnica está centralizada no empacotamento do componente devido à incompatibilidade da interface original. As funcionalidades existentes são preservadas e outras são adicionadas, no formato de camadas, para atender as novas necessidades.

Para [22] o princípio fundamental da adaptação de software é a capacidade de possuir mecanismos capazes de coletar dados a respeito de seu estado atual em seu ambiente de execução, analisá-los visando identificar mudanças e alterar dinamicamente seu comportamento/estrutura sem a percepção direta de seus clientes (consumidores de funcionalidades).

Segundo [17], 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 para que atendam aos requisitos de seus clientes. Partindo desse contexto, um método de adaptação de componentes binários foi proposto pelo autor [17], cujo objetivo é controlar o tamanho dos componentes a cada adaptação realizada. Esse controle deve ser realizado devido à utilização do empacotamento de software como técnica para "adição" de novas funcionalidades, pois ele pode sofrer um aumento rápido e significativo em seu tamanho no ambiente de execução.

Segundo [14], a adaptação pode ser utilizada em componentes de software em um modelo dividido em dois enfoques: observação do comportamento (*introspection*) e observação da mudança do comportamento (*intercession*). Nesse modelo destacam-se as operações realizadas no meta-nível: *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 meta-nível.

Para [21] a reconfiguração de aplicações distribuídas desenvolvidas com a RMI [20] é limitada devido à falta de mecanismos de reflexão em objetos remotos. Sua proposta é a utilização do padrão *Proxy* [24] para que os objetos remotos se comportem de maneira transparente à aplicação cliente como se fossem locais.

A reconfiguração de componentes em ambientes distribuídos também é abordada por [5]. Sua proposta é uma extensão da RMI, XRMI - *eXtended* JAVA RMI, permitindo que uma aplicação possa monitorar e manipular invocações entre componentes durante um processo de reconfiguração dinâmica. Detalhando suas etapas, um componente pode ser carregado dinamicamente no sistema, migrado de um local para outro, descarregado do sistema, em tempo de execução, ou atualizado dinamicamente.

Para implementar a RC deve-se ter um recurso que seja capaz de reconhecer as características e funcionalidades de classes/objetos. Segundo [19] a Reflexão em JAVA pode ser utilizada para a descoberta de informações como atributos, métodos, construtores e classes carregadas pela JVM, JAVA *Virtual Machine*. O sistema de supervisão de objeto, seção IV, foi implementado em JAVA com características reflexivas.

Além dos recursos de aquisição de conhecimento sobre classes e objetos já mencionados, existe uma implementação de uma classe *proxy* dinâmica - utilizada como mecanismo de reconfiguração. Essa classe implementa um conjunto de interfaces especificadas em tempo de execução e, dependendo da solicitação, um objeto será retornado. Dessa forma, a classe *proxy* dinâmica pode ser utilizada como um tipo seguro para geração de objetos, pois sempre está relacionada a uma solicitação de objetos clientes [19].

Na literatura é possível encontrar trabalhos realizados sobre a comparação entre as tecnologias de distribuição RMI e WS. Esses trabalhos mostram características de desempenho no uso dessas tecnologias, fornecendo parâmetros significativos na escolha de uma delas no projeto de uma aplicação distribuída.

Em [13] pode-se encontrar um estudo comparativo entre o

desempenho das tecnologias de distribuição RMI e WS. Os autores realizaram testes de comunicação em um ambiente distribuído (cliente-servidor) utilizando *proxies*, *firewalls* e tunelamento de portas. Estes testes permitiram realizar uma classificação das tecnologias e, também, gerar um conjunto de diretrizes para a recomendação de qual tecnologia adotar no desenvolvimento de uma aplicação distribuída.

Segundo [10], o uso do protocolo de comunicação HTTP, *HyperText Transfer Protocol*, e a transferência de documentos XML, *eXtensible Markup Language*, aumentam a interoperabilidade das aplicações, mas, por outro lado, representam um aumento considerável no custo de execução (Memória, Largura de banda da Rede e Processamento) dos WS em relação ao RMI. O autor comenta também sobre a flexibilidade de comunicação, pelos recursos citados – HTTP e XML – das aplicações desenvolvidas com WS em relação àquelas com RMI. Outro fator citado pelo autor e importante para este artigo, é, especificamente, o gerenciamento do servidor de objetos RMI (`rmiregistry`). Ele comenta sobre a necessidade de reiniciá-lo quando a aplicação é alterada. Neste artigo é possível observar que o sistema supervisor de reconfiguração (seção IV) elimina essa necessidade e a aplicação pode ser alterada (recompilada e inserida no servidor) sem reiniciar o servidor de objetos RMI.

[12] comentam que a comunicação de aplicações desenvolvidas com RMI é menos flexível que as desenvolvidas com WS, pois a troca de mensagens ocorre através de objetos desenvolvidos, apenas, na linguagem JAVA. Para otimizar a comunicação é necessário incorporar o protocolo IIOP, *Internet Inter-Orb Protocol*, presente na arquitetura CORBA, *Common Object Request Broker Architecture*. Eles apontam essa característica como nativa nos WS e um fator diferencial na adoção desta tecnologia. Ainda neste trabalho, os autores realizaram uma avaliação de desempenho das tecnologias, considerando critérios de segurança (WS, WS-*Security*, RMI e RMI-SSL) e fazendo a distinção pelo sistema operacional (*Windows* e *Linux*).

Os resultados dos trabalhos citados sobre RMI e WS são considerados relevantes para o contexto deste artigo, porém, vale ressaltar, que os dados neles contidos não são adotados como referência pelo ano de publicação dos mesmos e, consequentemente, pela evolução destas tecnologias (RMI e WS) no decorrer desses anos.

Finalmente, [23] define: "*Web Service* é um sistema de software identificado por um URI, *Universal Request Interface*, cujas interfaces públicas e ligações são definidas e descritas usando XML. Sua definição pode ser descoberta por outros sistemas de software. Estes sistemas podem então interagir com o serviço *Web* de uma maneira prescrita pela sua definição, usando mensagens baseadas em XML, cujo transporte é feito por meio dos protocolos Internet". Os autores ainda comentam que essa tecnologia pode ser aplicada ao desenvolvimento de software orientado a serviços (SOA, *Software Oriented Architecture*) [4, 23, 25], além de ser considerada uma "evolução" do RMI e CORBA para ambientes computacionais distribuídos.

III. Estrutura da Aplicação

Para que um aplicativo seja adaptável em tempo de execução é necessário que ele tenha sido desenvolvido com base em algumas premissas fundamentais [1, 2], além de estar hospedado em um ambiente de execução capaz de reconhecer as mudanças ocorridas no meio e modificá-lo quando necessário [1, 2].

Os autores [1, 2] apresentam uma metodologia que oferece uma contribuição para o desenvolvimento de software reconfigurável e um ambiente de execução para hospedagem desse tipo de sistema.

Nesta metodologia [1, 2] a concepção das classes, componentes e serviços, chamados deste ponto em diante de "artefatos de software", é realizada de maneira "pura" - sem a adição de qualquer requisito de infraestrutura (transversal). Os autores afirmam que essa fase está diretamente relacionada ao desenvolvimento com separação de interesses em requisitos funcionais e não funcionais [15, 16]. Além disso, destacam sua importância para o desenvolvimento de software reconfigurável utilizando WS [3, 11] e RMI [20].

No ambiente de execução, eles comentam sobre a presença de repositórios de software, que são encarregados de armazenar a documentação e os artefatos de software produzidos ao longo do desenvolvimento e que servirão de fonte de informação para o processo de reconfiguração [1, 2].

Neste artigo o foco é apresentar uma técnica de adaptação de sistema capaz de atuar de maneira transparente em sistemas orientados a serviços (*Web Services*) ou desenvolvidos com chamadas de métodos remotos (RMI). A Figura 2 mostra a estrutura de comunicação para uma aplicação *Web* com WS [3, 11] e RMI [20].

Em ambos os casos, o mecanismo de comunicação entre cliente e servidor é transparente, ou seja, o cliente não tem a percepção de que a aplicação encontra-se distribuída em serviços ou objetos remotos. No lado servidor, a aplicação possui uma camada intermediária (*middleware* - *servlets*), responsável pela comunicação entre o cliente e a sua lógica (aplicação servidora). Esta última é encarregada de utilizar os objetos *Stubs*, fornecidos pela camada servidora (serviços - WS ou objetos remotos - RMI), para viabilizar a comunicação entre eles. Finalmente, na camada, *Web* Services ou RMI, são implementadas as funcionalidades utilizadas pelos clientes. O acesso pela camada de *middleware* é realizado por meio de uma WSDL, *Web Service Description Language*, ou interface remota.

A. Descrevendo a aplicação: orientação a serviços e a chamadas de métodos remotos

Para detalhar o funcionamento de uma aplicação distribuída utilizando WS e RMI, ambas com recursos de auto-reconfiguração, será necessário confeccionar uma aplicação comum com as mesmas funcionalidades para que os critérios de comparação da adaptação de software sejam os mesmos.

O sistema exemplo a ser explorado neste artigo refere-se a uma "`Calculadora`" com as quatro operações elementares:

soma, subtração, multiplicação e divisão. Basicamente, essa aplicação segue organizada nos mesmos moldes estruturais presentes na Figura 2, em que as operações elementares estão distribuídas em uma rede e estão organizadas em serviços ou em formato de métodos remotos.

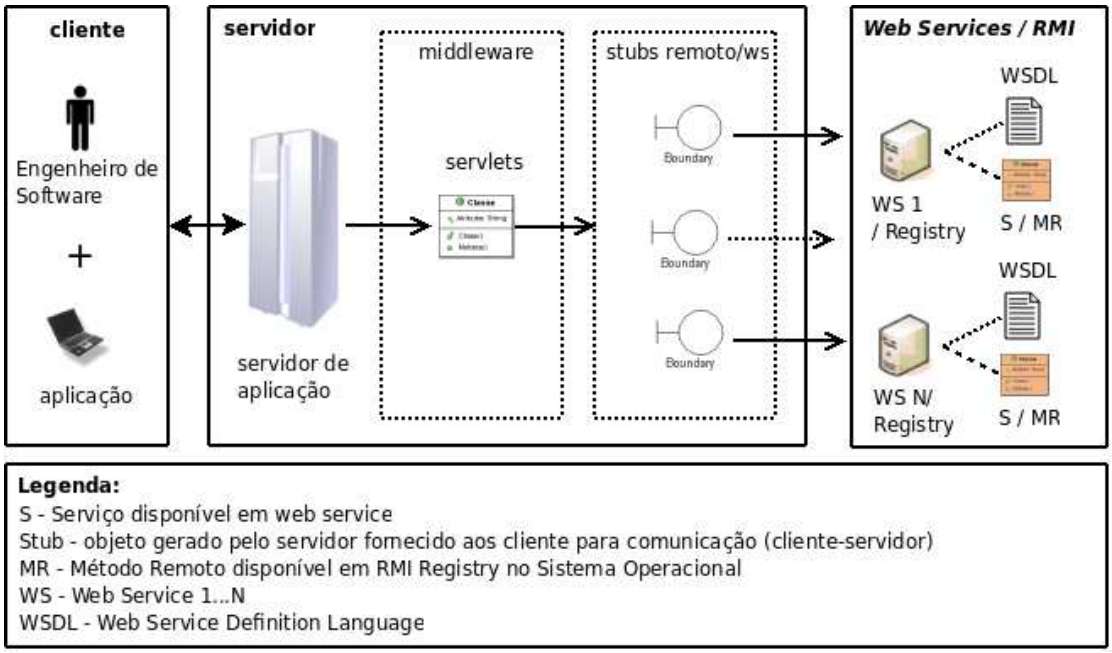

Fig. 2. Estrutura de uma aplicação distribuída

O exemplo escolhido para mostrar a proposta central deste artigo não possui grau de complexidade elevado quanto ao entendimento de sua lógica e funcionamento. No entanto, seus aspectos estruturais e comportamentais são suficientes para mostrar como serviços e chamadas remotas podem ser reconfiguradas em tempo de execução, assim como a facilidade de integração com outros sistemas (supervisão de objetos - seção IV).

Para desenvolver a "`Calculadora`" orientada a serviço ou com chamada de métodos remotos é necessário, inicialmente, definir qual(is) funcionalidade(s) será(ão) disponibilizada(s) para acesso de terceiro(s) - qualquer aplicação cliente. Em trabalhos desenvolvidos anteriormente [1, 2], um software pode ser desenvolvido em um domínio especifico, com um determinado propósito, e pode ser reconfigurado para atender uma nova necessidade em um domínio totalmente diferenciado do qual ele foi inicialmente projetado [3]. Essa situação determina, basicamente, que a funcionalidade a ser disponibilizada (serviço ou chamadas remotas) seja a mais genérica possível e independente de qualquer requisito não funcional. Sendo assim, e atendendo às premissas da metodologia definidas pelos autores [1, 2], apenas uma funcionalidade será elaborada com o nome de `operacao` e seu significado de execução será adaptado conforme as necessidades apresentadas pelas aplicações clientes.

Baseado nesses conceitos e apoiado por uma IDE, *Integrated Development Environment*, escolhida pelo desenvolvedor, inicia-se a geração do serviço *Web* para a funcionalidade `operacao`. Para este tipo de sistema, o cliente da aplicação "`Calculadora`" terá acesso apenas a uma interface de comunicação (WSDL), que descreve como utilizar este serviço num formato de uma caixa-preta, em que são definidas apenas as entradas e saídas para o cliente. Detalhes relativos à implementação e funcionamento da "`Calculadora`" não são apresentados, neste artigo, para a aplicação cliente.

A Figura 3 mostra, de maneira resumida, o arquivo XML que descreve a funcionalidade `operacao` para a aplicação "`Calculadora`".

```
1...
2 <xs:complexType name="operacao">
3   <xs:sequence>
4     <xs:element name="a" type="xs:int"/>
5     <xs:element name="b" type="xs:int"/>
6   </xs:sequence>
7 </xs:complexType>
8 <xs:complexType name="addResponse">
9    <xs:sequence>
10     <xs:element name="return" type="xs:int"/>
11   </xs:sequence>
12</xs:complexType>
13....
```

Fig. 3. Interface de comunicação XML (resumida)

Na linha 2 é possível identificar o nome do serviço disponibilizado (`operacao`), os parâmetros por ele requeridos (`a, b`), linhas 4 e 5, e, finalmente, o tipo de retorno (`int`) após sua execução, linha 10. Outras operações podem ser definidas em uma mesma aplicação, porém apenas os nomes, parâmetros e tipos de retorno são modificados.

Esta aplicação ao ser executada será hospedada em um servidor *Web*, escolhido pelo desenvolvedor (*Glassfish*), na forma de um serviço. Outro ponto importante desse tipo de aplicação é sua independência quanto à linguagem de programação dos clientes para consumo do serviço, pois fornece um mecanismo universal de comunicação baseado em XML. Sendo assim, um cliente escrito em qualquer linguagem de programação pode consumir o serviço disponibilizado pela aplicação servidora (Serviço Web).

O desenvolvimento da mesma funcionalidade (`operacao`) para a aplicação "`Calculadora`" utilizando RMI emprega os mesmos conceitos do desenvolvimento orientado a serviço quanto ao isolamento das funcionalidades (modularização), porém com procedimentos de implementação diferenciados. O desenvolvedor faz a especificação de uma interface JAVA e nela descreve a funcionalidade que deseja disponibilizar de maneira remota. Essa funcionalidade é descrita, basicamente, por meio de uma assinatura de método, que deve ser implementada por um ou mais objetos servidores. Dessa forma, para que os clientes tenham acesso a essa funcionalidade, o código fonte compilado dessa interface deve ser a eles disponibilizados, seja de maneira estática (envio manual pelos desenvolvedores) ou dinâmica (*download* dinâmico por meio de *codebase*) [20]. A Figura 4 mostra a interface remota para a aplicação "`Calculadora`".

```
1  ...
2  public interface Calculadora extends Remote {
3
4  int operacao(int a, int b)
5                          throws RemoteException;
6  // outras operações......
7  }
8 ....
```

Fig. 4. Interface Remota

Ao contrário do serviço, uma interface remota é dotada de informações relacionadas ao código fonte e não possui informações descritivas. Na linha 2 é possível identificar o nome da interface "`Calculadora`", que será fornecida à aplicação cliente. Na linha 4 é possível observar o tipo de retorno (`int`), nome da funcionalidade (`operacao`) e seus parâmetros e tipos (`int a, int b`).

Em ambos os casos, aplicação desenvolvida com serviços *Web* ou chamadas de métodos remotos, foram definidas as interfaces de comunicação da aplicação "`Calculadora`" para seus clientes. A primeira se mostrou mais flexível que a segunda quanto ao requisito interoperabilidade, pois o acesso às funcionalidades por ela disponibilizadas é transparente, podendo ser consumida por clientes implementados em qualquer linguagem de programação. Na segunda, somente clientes escritos na linguagem JAVA podem fazer uso de suas funcionalidades. Um fator considerado como negativo, encontrado no desenvolvimento da aplicação com WS, é sua dependência pela IDEs, pois o desenvolvimento é visual e apoiado por geradores de código. Em contrapartida, fator positivo a este, pode-se obter maior velocidade de desenvolvimento e minimização de erros eventualmente gerados pelos desenvolvedores [4, 10, 12, 13, 23, 25].

Sobre a interface para chamada remota de métodos e sua restrição de comunicação com aplicações clientes JAVA, pode-se minimizar esse problema utilizando o protocolo de comunicação IIOP, *Internet Inter-ORB Protocol* [20]. Dessa forma, aumentando a carga cognitiva de conhecimento do desenvolvedor para garantia de interoperabilidade desse tipo de aplicação com outros clientes - desenvolvidos em linguagens distintas. De maneira contrária a anterior, a fase de desenvolvimento dessa abordagem não está apoiada por IDEs ou qualquer mecanismos de geração de código, deixando todo trabalho de codificação para os desenvolvedores [4, 10, 13].

Na estrutura apresentada na Figura 2 pode-se observar que os clientes realizam o consumo direto dos serviços e dos métodos remotos por meio das classes `Servlets`, consideradas pelo cliente final da aplicação como servidora da aplicação. Pode-se observar que elas são servidoras para os clientes finais e consumidoras (clientes) de serviços e métodos remotos (objetos reconfiguráveis). A Figura 5 mostra a estrutura da aplicação e os aspectos comuns de comunicação e reconfiguração dos objetos.

Para utilizar este tipo de aplicação o cliente visualiza apenas as interfaces de comunicação e seus parâmetros, que são encaminhados à camada de *middleware*. Nessa camada, objetos *Stubs* são encarregados de realizar a comunicação com os objetos que realmente provém os serviços/métodos remotos. Neste momento, um ponto de comparação é abordado: o processamento distribuído ou processamento distribuído reconfigurável.

No processamento distribuído tradicional cada funcionalidade (remota ou serviço *Web*) encontra-se localizada em computadores distintos e unida por chamadas através da camada de *middleware*. Dessa forma, pode-se dizer que seu funcionamento é transparente para o cliente final, pois detalhes de distribuição da aplicação por diversos computadores são "mascarados" para uma unidade, comportando-se como uma aplicação cliente-servidor simples.

No processamento distribuído com reconfiguração existe apenas um provedor de serviços e/ou métodos remotos e um sistema supervisor, encarregado de analisar as alterações do ambiente de execução. Quando uma chamada por um serviço ou método remoto é enviada para os objetos reconfiguráveis, uma interceptação é feita por metas-objetos, que analisam aspectos técnicos (parâmetros, retornos e informações semânticas - descritoras das funcionalidades) na tentativa de

localizar no ambiente, objetos capazes de atendê-la. A requisição é processada e devolvida ao cliente, sem que ele tenha a percepção do que tenha ocorrido.

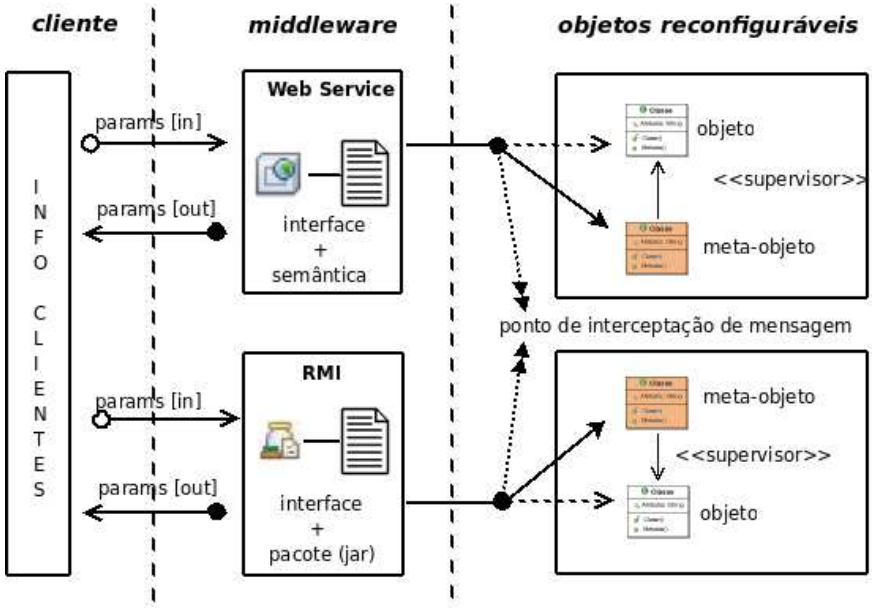

Fig. 5. Estrutura de processamento distribuído

Apesar de apresentarem o mesmo aspecto estrutural, Figura 5, a implementação que utiliza chamadas de métodos remotos requer a inserção de informação semântica. Nas chamadas orientadas a serviços estas não são necessárias e, portanto, reduzem o tempo de implementação. Outro aspecto importante a ser destacado neste artigo, seção IV, é o mecanismo de reconfiguração de objetos utilizado no desenvolvimento no sistema "`Calculadora`", tanto para a implementação orientada a serviço quanto para a orientada a chamada de métodos remotos.

A seguir, é apresentada uma breve descrição do subsistema responsável pela reconfiguração dos sistemas.

IV. Mecanismo de Reconfiguração de Software

Como pode-se observar na Figura 5 existe, em ambos os casos, um ponto de interceptação de mensagens, tanto no sistema desenvolvido com serviços (*Web* Service) quanto no sistema utilizando RMI. Este ponto refere-se a um sistema de supervisão de objetos, pois quanto uma solicitação é enviada para o objeto servidor um meta-objeto, responsável por monitorar suas ações, intercepta a ação vinda do objeto cliente e verifica a capacidade do objeto atual, por ele monitorado, em atender esta solicitação. Essa verificação/validação de funcionalidade pode ocorrer de duas formas:

1. **Quando o objeto atende a solicitação:** a execução ocorre normalmente, ou seja, o meta-objeto encaminha a solicitação para o objeto que está sendo por ele monitorado naquele momento;
2. **Quando o objeto não atende a solicitação:** a execução é interrompida momentaneamente até que um objeto possa ser localizado no ambiente de execução reconfigurável e referenciado pelo meta-objeto para atender a solicitação do cliente. A busca por objetos é realizada pelos agentes de software, dotados de informações semânticas e técnicas (parâmetros, retornos, etc), nos repositórios de informação [1, 2]. Quando nenhum objeto atender aos requisitos de solicitação do cliente, uma exceção é enviada ao cliente como resposta.

No segundo caso, quando satisfeitos os critérios de solicitação do cliente, a reconfiguração do objeto é gerenciada por um subsistema supervisor de objetos. Este é o responsável por modificar o código fonte da aplicação atual sem interromper a execução do ambiente como servidores *Web* (*Web* Service) ou remotos (RMI). A Figura 6 mostra o modelo de classes desse subsistema.

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 por uma interface `InterfaceOfReflectClass` e uma classe `ReflectClass`, que representam 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.

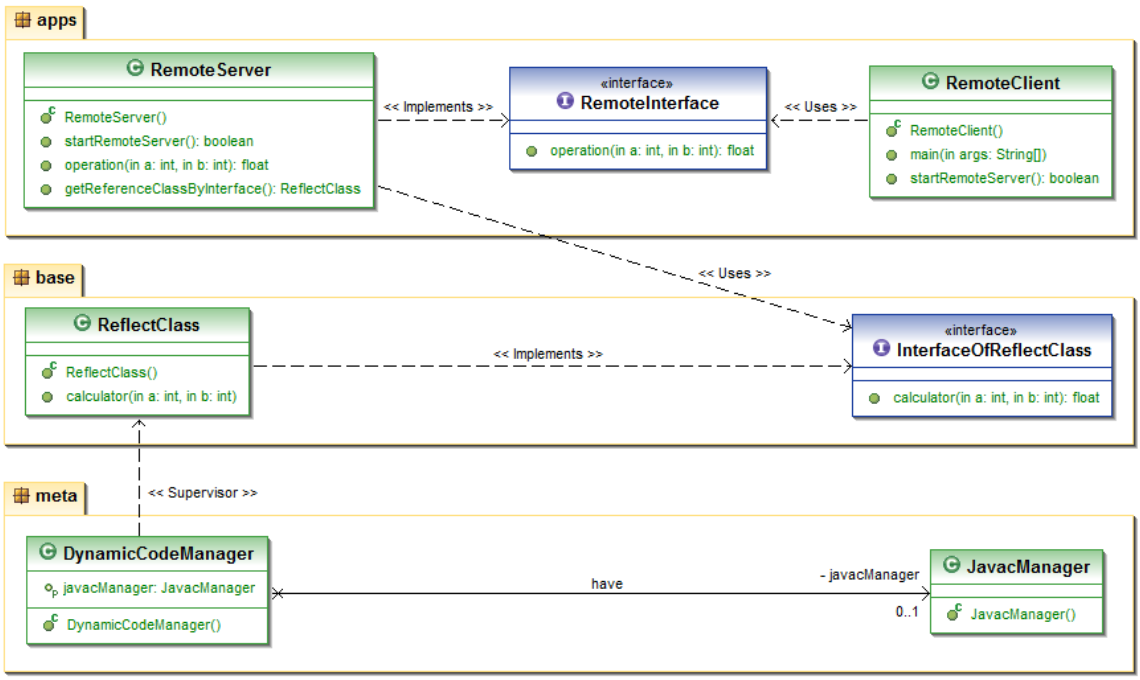

Fig. 6. Modelo de classe do subsistema de monitoramento de objetos

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 `calculator`, que está presente 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. Eles (clientes) a utilizam como se fosse local e não têm a percepção quanto à distribuição e reconfiguração.

Para melhor situar o leitor julga-se necessários alguns esclarecimentos sobre o modelo presente na Figura 6 em relação ao exemplo apresentado como estrutura de aplicação (seção III). A funcionalidade `operation` é equivalente ao conteúdo (`operacao`) apresentado na Figura 4 para RMI, por exemplo. O sistema a ser reconfigurado está representado pela classe `ReflectClass` e pela funcionalidade `calculator` por ela implementada - Figura 6. Nesta classe deve existir a implementação da lógica do sistema. Para finalizar, o nome dessa funcionalidade não possui significado para a aplicação apresentada na seção III, pois seu gerenciamento é realizado de maneira automática no ambiente de execução reconfigurável proposto pelos autores [1, 2].

Quanto aos aspectos referentes ao funcionamento da reconfiguração dos objetos, a implementação proposta prevê que os novos objetos sejam criados por mecanismos de herança com agregação de características e funcionalidades ou pela modificação e substituição dos executáveis em tempo de execução [2]. No entanto, sobre este último, por se tratar de aplicações distribuídas, ele requer maior atenção por parte do sistema supervisor, pois pode envolver objetos que estão sendo referenciados por diferentes aplicações clientes e devem ter seus estados preservados quando um novo objeto, similar a ele é criado, compilado e alocado em memória - *classload* da aplicação "`Calculadora`" - para atender uma nova execução de um novo cliente.

Ainda no contexto da reconfiguração, existem situações em que as solicitações vindas dos objetos clientes não são atendidas pelos provedores de serviços e um novo objeto necessita ser invocado. Para isso, o meta-objeto faz uma consulta nos repositórios de informação, presentes no ambiente de execução reconfigurável [1, 2], para verificar se existe algum objeto capaz de atender a solicitação requisitada pelo cliente. Caso um objeto seja encontrado, este é compilado e executado pelo sistema supervisor para atender a requisição do cliente. Caso contrário, objeto não encontrado, uma exceção é lançada e encaminhada ao cliente da solicitação [1, 2].

Retomando a ilustração na Figura 5, pode-se observar na parte superior uma área segmentada chamada de objetos reconfiguráveis. Conforme apresentado em [1, 2] existe um subsistema (Figura 6) encarregado de supervisionar o comportamento dos objetos em execução e modificá-los conforme a necessidade apresentada, semanticamente (serviços *Web* ou objetos remotos), pela aplicação cliente. Além disso, os autores comentam que qualquer sistema pode utilizar o mecanismo de reconfiguração independente do tipo de aplicação (local e/ou distribuída).

Na próxima seção são apresentados resultados obtidos com a aplicação da técnica de reconfiguração de software no desenvolvimento da aplicação "`Calculadora`" orientada a serviços e orientada a chamada de métodos remotos.

V. DISCUSSÃO DOS RESULTADOS

Com a aplicação da técnica de reconfiguração, sistema de supervisão de objetos - seção IV, no desenvolvimento da aplicação "`Calculadora`" utilizando WS e RMI, algumas características comparativas entre elas podem ser obtidas. Essas características estão embasadas nos trabalhos elaborados pelos autores [10, 12, 13], no entanto, vale ressaltar, que a avaliação de desempenho entre as tecnologias de distribuição associadas à técnica de reconfiguração (seção IV) não foi realizada neste artigo. Sendo esta uma atividade que pode ser realizada como trabalho futuro, pois necessita de sistemas de maior porte de desenvolvimento e processamento de informações.

Pode-se dizer que ambas as tecnologias oferecem interfaces de comunicação, bem definidas, com os clientes. Dessa forma, o desenvolvimento da aplicação ocorre de maneira natural e o acoplamento do sistema de supervisão de objetos é realizado de maneira transparente, sem que o cliente saiba que esteja utilizando uma aplicação reconfigurável.

Observou-se também a capacidade de ambas as tecnologias, WS e RMI, trabalharem com o sistema de supervisão de objetos como adaptadores. Apenas uma funcionalidade foi definida e implementada como interface de acesso ao sistema "`Calculadora`". O sistema de supervisão foi capaz de atuar nessa funcionalidade modificando seu conteúdo (implementado) para atender as novas necessidades das aplicações clientes.

Ainda sobre a característica citada no parágrafo anterior, existe um fator importante e diferenciador que merece ser destacado entre as tecnologias de distribuição. Esse fator é a capacidade de trabalhar com informações semânticas dos WS em relação ao RMI. Isso minimizou a integração de mais subsistema que suprisse essa necessidade no ambiente de execução reconfigurável [2].

A tecnologia RMI, por ser mais primitiva [10] necessitou de complementação do protocolo IIOP para que clientes, desenvolvidos em linguagens distintas – não JAVA, pudessem consumir as funcionalidades remotas disponibilizadas pelos objetos servidores RMI.

Outro fator de grande relevância observado na execução de ambas as aplicações (WS e RMI) foi a não necessidade de reiniciar os servidores de objetos (*Glassfish* – para WS e `rmiregistry` – para RMI). O sistema de supervisão de objetos foi capaz de identificar as modificações solicitadas pelos clientes, modificar e recompilar a aplicação em tempo de execução. Dessa forma, pode-se dizer que essa modificação ocorreu de maneira transparente, sem a percepção dos clientes do sistema.

Finalmente, outra característica relevante presente na elaboração das aplicações com o sistema de supervisão de objetos é a relação de independência de requisitos, pois a técnica apresentada, Figura 5, permite que os objetos sejam desenvolvidos num processo de linha de montagem [1, 2], conforme mostra a Figura 7.

Fig. 7. Objeto em linha de montagem [1, 2]

No processo de montagem, Figura 7, o objeto central é constituído apenas por requisitos funcionais, sendo os requisitos transversais e/ou de infraestrutura como persistência, distribuição, segurança, otimização e/ou reflexão a ele agregados, formando uma única unidade de software. Os autores [1, 2] ainda comentam que esse processo pode ser conduzido de duas maneiras: **(1) automática**, pelo sistema de supervisão de objetos no ambiente de execução; **(2) manual**, pelo engenheiro de software na etapa de desenvolvimento do software.

VI. CONSIDERAÇÕES FINAIS

Este artigo apresentou um estudo comparativo da adaptação de software utilizando WS e RMI. Notou-se, com a realização deste artigo, que essas tecnologias de distribuição se mostraram imparciais quanto ao acoplamento do sistema de supervisão de objetos. Sendo assim, elas não causam nenhum impacto negativo quando um sistema com características de reconfiguração é desenvolvido. Dessa forma, pode-se concluir que a escolha da tecnologia de distribuição fica a critério dos engenheiros de software/desenvolvedores em relação aos recursos fornecidos e a necessidade da aplicação que se pretende desenvolver.

Sobre o sistema de supervisão de objetos, existe uma característica importante a ser destacada. Segundo [10], o servidor de objetos (`rmiregistry`) deve ser reiniciado toda vez que as aplicações, nele hospedadas, são alteradas. Conforme apresentado por [2], o sistema de supervisão de objetos, apresentado na seção IV, permite que os artefatos de software, por ele monitorados, sejam reconfigurados sem a necessidade de reiniciar as atividades dos servidores (*Glassfish* – WS e `rmiregistry` – RMI) em que estão hospedados. As modificações, nas aplicações servidoras, são realizadas de maneira transparente, sem a percepção de seus clientes (consumidores de suas funcionalidades).

Ainda nesse contexto, essa característica é muito importante para os sistemas que utilizam RMI como tecnologia de distribuição, pois evita o processo de registros de nomes para os objetos servidores. Além disso, preservam os estados das

execuções vigentes de outros objetos que não estão sendo reconfigurados naquele determinado momento.

Outra característica importante que pode ser destacada e comparada com outros trabalhos realizados é a reflexão sobre objetos remotos. Em [5] é possível encontrar uma técnica de reconfiguração que atua especificamente em RMI. Sua técnica consiste, basicamente, em migrar os objetos remotos de um domínio para outro para que a adaptação seja realizada. No trabalho realizado por [2], esse "deslocamento" de objetos entre domínios é desnecessário, pois o sistema supervisor de objetos gerencia a reconfiguração e mantém o artefato (objeto remoto) no mesmo domínio para o qual ele foi desenvolvido.

A manutenção dos artefatos no domínio de atuação minimiza as incertezas na busca por novos artefatos. Quando um artefato é migrado de um domínio para outro, novas informações são a eles associadas, além da implementação que é modificada. Dessa forma, novas versões de artefatos são criadas de maneira "descontrolada", aumentando rapidamente os dados das bases de artefatos e, consequentemente, causando redundância nessas bases - presentes nos diferentes domínios. Isso pode prejudicar o sistema de reconfiguração (busca de informação) tanto nas etapas manuais quanto automáticas.

Para finalizar, nos trabalhos dos autores [10, 12 e 13] é possível encontrar um conjunto de características comparativas entre as tecnologias de distribuição WS e RMI. Os detalhes de implementação e facilidades de comunicação com os clientes também foram comprovados no desenvolvimento das aplicações propostas (seção III). O motivo da comprovação das características ou a imparcialidade (citada no primeiro parágrafo desta seção) está diretamente relacionado à maneira de como essas tecnologias foram utilizadas para caracterizar um sistema distribuído reconfigurável. Em ambos os casos (WS e RMI), a operação (`operacao`) do sistema "`Calculadora`" foi empacotada por meio de uma camada de software, responsável apenas para receber e enviar dados do/para os clientes. Dessa forma, oferecendo total independência no acoplamento do sistema supervisão de objetos, encarregado de monitorar e realizar a reconfiguração dos artefatos. Os detalhes do comportamento dessa implementação podem ser visualizados na Figura 6.

REFERÊNCIAS

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

[2] 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;

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

[4] CBDI. Site oficial do service oriented architecture practice portal. Disponível em: <http://everware-cbdi.com/cbdi-forum> Acessado em: 23 de mai. 2011;

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

[6] 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;

[7] 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;

[8] FERNANDES, A. P. Reflexão computacional. Disponível em <http://attila.urcamp.tche.br/~acauan/art_ccei_rc.html> Acessado em: 26 de fev. 2011;

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

[10] GRAY, N. A. B. Comparison of Web Services, Java-RMI, and CORBA service implementations. Fifth Australasian Workshop on Software and System Architectures, 2004;

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

[12] JURIC, B. M. et. al. Comparison of performance of Web services, WS-Security, RMI, and RMI-SSL, Journal of Systems and Software, Volume 79, Issue 5, Quality Software, Pages 689-700, ISSN 0164-1212, DOI: 10.1016/j.jss.2005.08.006, 2006;

[13] JURIC, B. M. et. al. Java RMI, RMI tunneling and Web services comparison and performance analysis, ACM SIGPLAN Notices, Volume 39, Pages 58-65. DOI=10.1145/997140.997146, 2004;

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

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

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

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

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

[19] ORACLE-REFLECT. Site oficial da oracle Disponível em: <http://download.oracle.com/javase/tutorial/reflect/index.html> Acessado em: 27 de mai. 2011;

[20] ORACLE-RMI. Site oficial da oracle. Disponível em <http://download.oracle.com/javase/tutorial/rmi/index.html>. Acessado em: 27 de mai. 2011;

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

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

[23] 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;

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

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

[26] W3C-SOA. Site oficial do w3c, world wide web consortium - web services architecture. Disponível em <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211> Acessado em: 18 de fev. 2011;

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

Frank José Affonso: possui graduação em Ciência da Computação pelo Centro Universitário Central Paulista (2000), mestrado em Ciência da

Computação pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos (05/09/2003) e doutorado em Engenharia Elétrica pelo Programa de Pós-Graduação do Departamento de Engenharia Elétrica da Escola de Engenharia de São Carlos - Universidade de São Paulo (26/05/2009). Foi celetista do Centro Universitário Central Paulista (set/2002-jan/2010) com as seguintes funções: professor; coordenador do curso de Sistemas de Informação; e coordenador dos Laboratórios de Informática do Campus. Atualmente é professor assistente doutor em Regime (RDIDP) - Efetivo da PP-QDUNESP. Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software e Programação JAVA, atuando principalmente nos seguintes temas: Reengenharia de Interface; JAVA com desenvolvimento *Web*, Desktop e Distribuído; C/C++; UML; Framework de Persistência (Hibernate) e Reconfiguração de Software em tempo de Execução.

Evandro Luís Linhari Rodrigues: possui graduação em Engenharia Elétrica pela Escola de Engenharia de Lins (1983), mestrado em Engenharia Elétrica pela Universidade de São Paulo (1992) e doutorado em Física pela Universidade de São Paulo (1998). Atualmente é professor da Universidade de São Paulo. Tem experiência na área de Engenharia Elétrica, com ênfase em Automação Eletrônica de Processos Elétricos e Industriais, atuando principalmente nos seguintes temas: processamento de imagens, microprocessadores/microcontroladores, visão computacional, análise carpal e automação.