Intercessão em Tempo de Implantação – uma Abordagem Reflexiva para a Plataforma J2EE™ –

Cristina Verçosa Pérez Barrios de Souza, Carlos Alberto Maziero
Programa de Pós-Graduação em Informática Aplicada – PPGIA
Pontifícia Universidade Católica do Paraná – PUCPR
Rua Imaculada Conceição, 1155 – CEP 80215-901
Phone/Fax: +55 41 330 1669
Curitiba – PR – Brazil
e-mail: {cristina, maziero}@ppgia.pucpr.br

Resumo

Esse artigo apresenta uma proposta para utilizar princípios de reflexão computacional em uma plataforma voltada para aplicações corporativas baseadas em componentes: a Plataforma Java para Corporações, ou J2EE. O principal objetivo da proposta é o de manter as vantagens da consistência de ambiente da J2EE e, ao mesmo tempo, possibilitar a alteração do comportamento de componentes servidores, mantendo intactas suas funcionalidades básicas – o que caracteriza reflexão comportamental, ou intercessão. Com isso, é adicionado um grau a mais de flexibilização na implementação de aplicações corporativas nas situações onde é necessário introduzir controle e / ou modificar a funcionalidade da aplicação como um todo. Para tanto, apenas são utilizados os padrões J2EE para construção e implantação de aplicação corporativa. Essa abordagem é então denominada MOP (Meta-Object Protocol) em Tempo de Implantação – ou mais especificamente, Intercessão em Tempo de Implantação.

Abstract

This paper presents an approach to use some features of computational reflection in the Java Platform for the Enterprise, the J2EE platform. Our main goal is to maintain the J2EE's environment consistency, and, at the same time, to enable the change of server components behavior, without modify their basic functionality – what characterizes beharioral reflection, or intercession. Consequently, it provides one more flexibility level on introducing control and / or on modifying the whole enterprise application functionality. In order to achieve such a goal, only the J2EE's application composition abilities are used. This approach is henceforth called Deploy-Time MOP (Meta-Object Protocol) – or more specifically, Deploy-Time Intercession.

1. Introdução

Uma das principais preocupações das aplicações servidoras se concentra em como prover interoperabilidade e portabilidade, a fim de fornecer uma solução duradoura e de grande alcance. Essas habilidades em particular recebem especial atenção da Plataforma Java para Corporações. Também denominada de Java 2 *Platform Enterprise Edition*, ou J2EE, ela define uma arquitetura Java unificada, centrada em serviços e baseada em componentes, que promove interoperabilidade, portabilidade e ambiente de execução consistente para aplicações corporativas.

O paradigma reflexivo, por sua vez, torna possível que as computações observem e modifiquem as propriedades de seu comportamento, obtendo autoconsciência dos seus comportamento e estado, e alterando e fazendo uso dessa informação nas decisões sobre o que fazer em seguida [13]. Permite assim que uma aplicação controle seu comportamento atuando

sobre si mesma. Para tanto, a parte funcional (nível base) de uma aplicação é separada de suas partes não funcionais (nível meta), ficando o nível base com os métodos e procedimentos da aplicação em si (sua funcionalidade básica), e o nível meta com as funções de controle e gerenciamento da aplicação. Essa abordagem é conhecida como protocolo de meta-objeto, ou MOP (*Meta-Object Protocol*), detalhado na seção 3 deste artigo.

Dessa forma, a reflexão computacional permite independência entre nível base e nível meta, fornecendo flexibilidade de implementação, uma vez que alterar o gerenciamento / controle do nível meta não implica em alterar, ou mesmo afetar, a implementação dos algoritmos da aplicação, localizados no nível base.

No que se refere à Plataforma Java, tais capacidades reflexivas integrais não estão completamente especificadas, sendo esse o tema de estudo de várias abordagens: [10], [20] e [5]. No entanto, essas propostas ou propõem uma nova perspectiva para o padrão Java, ou se baseiam em práticas nem sempre acessíveis na plataforma J2EE – o que pode vir a comprometer as vantagens corporativas e de ambiente oferecidas pela plataforma J2EE.

Contudo, a J2EE define facilidades para composição de aplicação que favorecem o desacoplamento de funcionalidade em componentes lógicos, e encorajam a reutilização de código orientado a componente. Tais características, aliadas à facilidade de alteração de suas entradas de ambiente (usadas para configuração, controle e gerenciamento), geram aberturas que permitem customizar o comportamento de componentes – ou seja, utilizar os princípios de reflexão – no momento da montagem de aplicação.

Esse trabalho objetiva, portanto, propor uma abordagem consistente para utilizar reflexão computacional na plataforma J2EE, mantendo todas as suas características de interoperabilidade, portabilidade e consistência de ambiente. Com esse propósito, é apresentada a abordagem MOP (*Meta-Object Protocol*) em Tempo de Implantação, ou mais especificamente Intercessão em Tempo de Implantação – como detalhado na seção 4.

Esta abordagem utiliza-se apenas da facilidade de composição das aplicações J2EE, visando possibilitar a alteração do comportamento de componentes de negócio de forma sistematizada e modular, baseada no paradigma reflexivo. Fornece assim um grau a mais de flexibilização de implementação nas situações onde é preciso introduzir controle e / ou alterar o comportamento da aplicação corporativa como um todo.

2. Plataforma Java

A tecnologia de componentes revolucionou a forma de desenvolver complexos sistemas de informação através da combinação e extensão de blocos reutilizáveis de software. Nessa linha, os JavaBeans (lançados pela Sun Microsystems em 1996) emergiram rapidamente como um importante padrão de componentes para aplicações cliente, com as vantagens de portabilidade e desenvolvimento através de ferramentas visuais [12]. Contudo, os JavaBeans não foram projetados para criar aplicações servidoras. A JVM (Java *Virtual Machine*) possibilita que uma aplicação execute em qualquer sistema operacional – portabilidade WORA (*"Write Once, Run Anywhere™"*) –, porém componentes servidores precisam de serviços adicionais, não providos diretamente pela JVM, mas sim por uma infra-estrutura de sistemas distribuídos [12].

2.1. Enterprise JavaBeans

A especificação da arquitetura Enterprise JavaBeans (ou EJB, lançada pela Sun Microsystems em 1998), veio suprir a demanda por componentes servidores, estendendo o modelo original dos componentes JavaBeans para suportar aplicações servidoras. Para tanto, os EJBs são preparados para suportar serviços essenciais (nome, transação, segurança, etc.) providos por diferentes infra-estruturas de sistemas distribuídos (p. ex. CORBA) [16].

Na arquitetura EJB, um componente enterprise bean é implantado dentro de um *container*, que provê um contexto de aplicação e habilita a interação entre vários componentes, podendo fornecer também gerenciamento e serviços de controle para os mesmos – através de um servidor EJB. Tal construção possibilita portabilidade, garantida por contratos (ou interfaces) entre o *container* e o EJB, e entre o cliente e o *container* (Fig. 1).

O *container* invoca as interfaces do EJB em tempo de execução. Logo, o cliente não interage diretamente com o EJB, mas sim com as interfaces, cujas classes são geradas pelo próprio *container*, em tempo de implantação. Assim, também é possível que o *container* intercepte as chamadas a métodos para inserir gerenciamento. As interfaces EJB são [21]:

- **Interface *Home***: provê o acesso aos serviços de ciclo de vida do EJB (criação/destruição de instâncias do EJB). O *container* registra essa interface para cada classe de EJB instalada no mesmo, através da API *Java Naming and Directory Interface* (JNDI), permitindo que o cliente a localize. Quando um cliente cria ou localiza um enterprise bean, o *container* retorna a sua interface *Remote*.
- **Interface *Remote***: provê o acesso aos métodos de negócio do enterprise bean, e permite que o *container* insira serviços de gerenciamento de estado, controle de transação, e serviços de segurança e de persistência.

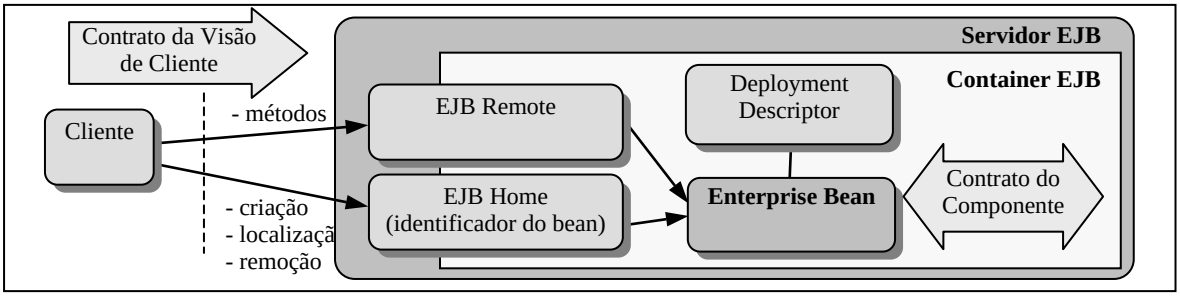

Fig. 1 – O *Container*EJB.

As regras associadas com o gerenciamento de ciclo de vida, transações, segurança e persistência de um EJB são definidas em um arquivo denominado *deployment descriptor* – descritor de implantação, escrito em XML (eXtensible Markup Language). Essas regras são declaradas em tempo de desenvolvimento. Em tempo de execução, o *container* executa serviços de acordo com os valores descritos nesse arquivo [21].

2.2. Plataforma J2EE

A contínua evolução dos padrões Java (JavaBeans e EJBs) conduziu, enfim, à especificação da Plataforma Java para Corporações (lançada pela Sun Microsystems em 1999). Também denominada de J2EE, ela define uma arquitetura Java baseada em

componentes e em serviços (nome, transação, segurança, etc.), e habilitada para aplicações multi-camada. Fornece assim o suporte de ambiente necessário aos componentes servidores EJB. Seu lançamento agrega uma série de especificações, dentre um conjunto de lançamentos [14] [17] [18] [22], cujo objetivo é o de auxiliar os desenvolvedores a alcançar o ideal WORA no lado servidor:

- **J2EE *Platform Specification***: (Especificação da Plataforma) define as APIs Enterprise Java (EJB, JNDI, JDBC, Servlets, JSP, JMS, JavaMail, …) e suas versões, que devem ser suportadas para garantir mínima qualidade de serviço, compatibilidade, portabilidade e integração**.**
- **J2EE *Application Programming Model***: (Modelo de Programação de Aplicação) que visa auxiliar o desenvolvimento de aplicações corporativas multi-camada para a plataforma J2EE. Inclui exemplos e *design patterns* bem sucedidos para corporações.
- **J2EE *Compatibility Test Suite***: (Conjunto de Testes de Compatibilidade) utilizado por fornecedores de software para verificar se sua implementação da plataforma J2EE é compatível com a especificação J2EE
- **J2EE *Reference Implementation***: (Implementação de Referência – J2EE SDK) é uma implementação da especificação da plataforma J2EE, que visa demonstrar suas capacidades, bem como prover uma definição operacional da mesma. Está disponível, juntamente com seu código fonte, para livre utilização.

A J2EE proporciona, dessa forma, um ambiente de execução integrado, consistente e atestado (Fig. 2), cujo *backbone* é constituído pelos componentes EJB. Garante assim uma determinada qualidade de serviço, assegurando também portabilidade e interoperabilidade.

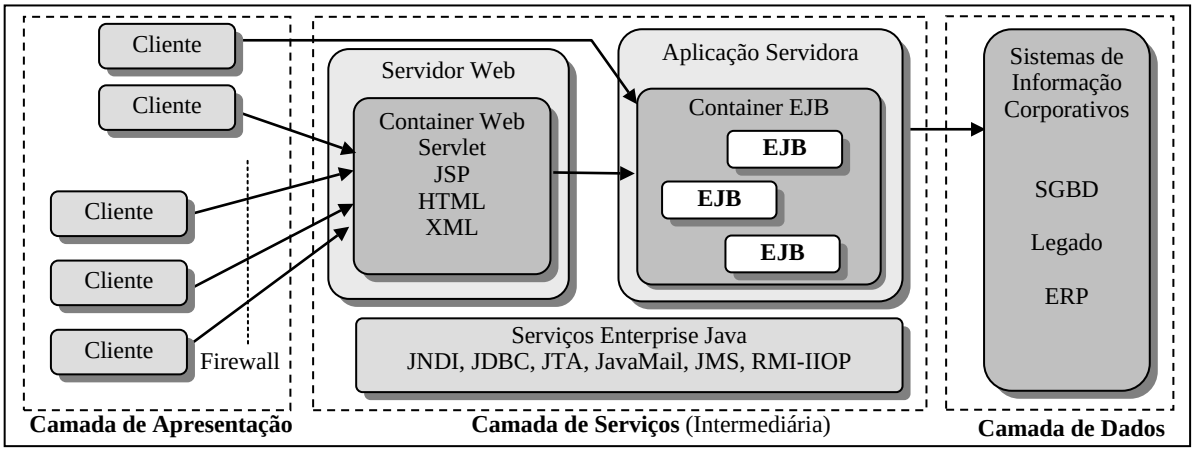

Fig.2 - Ambiente multi-camada J2EE.

2.2.1. Aplicação Corporativa

Uma aplicação corporativa J2EE é feita de um ou mais enterprise beans, de componentes Web, componentes de aplicação de cliente e de um *deployment descriptor* referente ao conjunto da aplicação – cada componente / aplicação têm seu próprio *deployment descriptor* (Fig. 3). Uma Ferramenta de Implantação de Aplicação cria automaticamente todos os *deployment descriptors* necessários [14], de acordo com os seguintes agrupamentos:

- **Uma aplicações Cliente** e seu *deployment descriptor* são empacotados em arquivos **.jar**.

- **Componentes Web** (JSP e Servlets), todos os seus arquivos relacionados (.HTM, .GIF, etc.) e seu *deployment descriptor* são empacotados em arquivos do tipo **.jar**, que nesse caso são chamados de *Web Archive* (Arquivo Web), cuja extensão é **.war**.
- **Componentes Enterprise JavaBeans**, juntamente com todos os seus arquivos de classes e seu *deployment descriptor*, são empacotados em arquivos **.jar**.
- **Uma aplicação Corporativa**, que contém os arquivos de todos os seus componentes, mais o *deployment descriptor* da própria aplicação, é empacotada em um arquivo do tipo **.jar**, nesse caso chamado de *Enterprise Archive* (Arquivo Corporativo), cuja extensão é **.ear**. Assim que um arquivo **.ear** é criado (pela *Application Deployment Tool*), ele está pronto para ser implantado em um servidor J2EE.

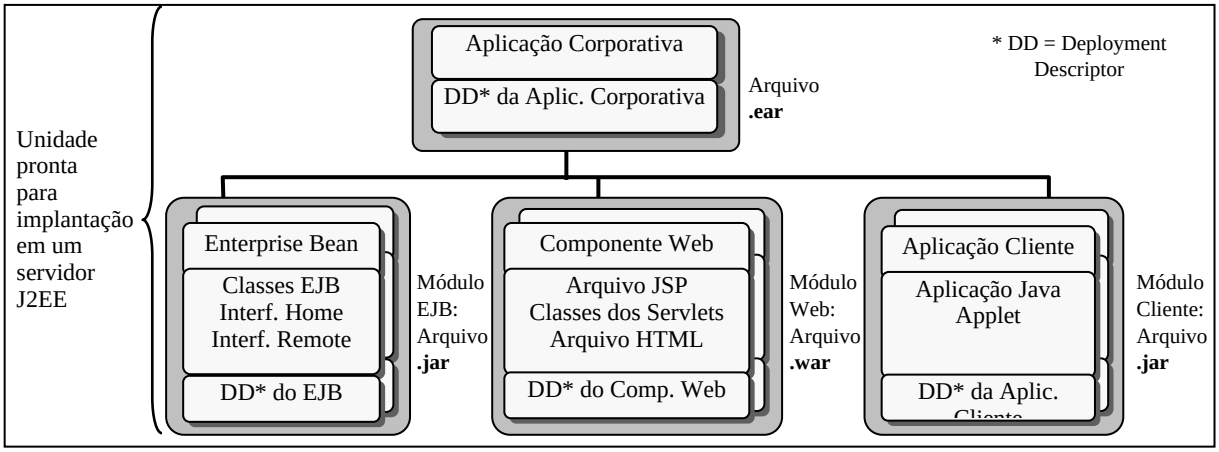

Fig. 3 - Aplicação Corporativa.

2.2.2. Modelo de Programação J2EE

O modelo de programação J2EE fundamenta-se na integração de camadas, ou montagem da aplicação. É assim que o desenvolvimento, a implantação e a reutilização de código orientado a componentes são facilitados. Para tanto, a J2EE considera vários cenários de aplicações – apesar de não haver tendências favorecendo um cenário em detrimento de outro [14]. Dentre os cenários suportados, há destaque para o cenário denominado de Modelo de Aplicação Multi-camada (Fig. 4), cuja habilidade maior está em desacoplar o acesso a dados de questões de como realizar interações com o usuários, propiciando escalabilidade.

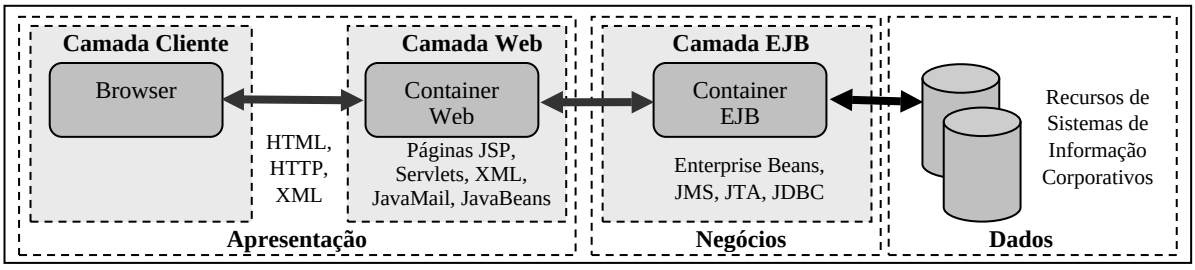

Fig. 4 - Modelo de Aplicação Multi-camada.

O modelo de programação J2EE também adota o *pattern* MVC (*Model-View-ControllerI*), que orienta o processo de decompor uma aplicação em componentes lógicos: o *Model* representa as regras de negócio e de dados; o *View* trata da apresentação; e o *Controller* gerencia a interação do usuário com o *View* e as invocações ao *Model*. No modelo

de Aplicação Multi-camada da J2EE, o MVC pode ser implementado através de páginas JPS, EJBs e componentes JavaBeans, como ilustrado na Fig. 5.

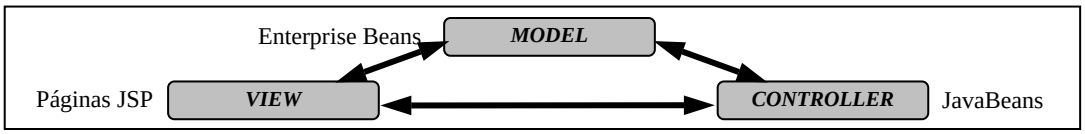

Fig. 5 - MVC em aplicações J2EE Multi-camada.

2.2.3. Considerações sobre a J2EE

As especificações e definições da plataforma J2EE preenchem diversas lacunas existentes nos atuais ambientes corporativos, principalmente no que diz respeito ao conjunto das características: portabilidade, interoperabilidade, produtividade e consistência de ambiente, dentro de um contexto que permita evolução com escalabilidade e desempenho – propriedades essenciais em aplicações servidoras. E foi esse leque de vantagens altamente promissor que contribuiu para determinar a adoção da J2EE como plataforma de sustentação da a proposta deste trabalho.

Contudo, é importante destacar que, ao se adotar essa plataforma, deve-se estar ciente de que não se obterá apenas vantagens corporativas. Também existem restrições de programação, especificamente aquelas inerentes aos componentes enterprise beans, que não recomendam a utilização de várias primitivas Java, sob pena de quebrar a portabilidade do componente EJB e gerar conflito com as atribuições do *container* EJB. Contudo, isso não constitui um problema se tais primitivas não forem o foco principal da aplicação em questão, exatamente o caso das aplicações corporativas, onde a meta principal, como já enfatizado, é produtividade com habilidades de interoperabilidade e portabilidade

3. Reflexão Computacional

A reflexão computacional é o processo em que um sistema pode analisar seu próprio comportamento e atuar sobre o mesmo. Para tanto, deve conter dados que representem os aspectos estruturais e computacionais do sistema. Ainda, tais dados devem poder ser manipulados, e devem ser conectados causalmente ao comportamento do sistema: alterações nos mesmos devem causar alterações no comportamento e vice-versa [3].

Quando a reflexão é aplicada à programação orientada a objetos, tem-se a abordagem denominada de protocolo de meta-objetos [6], ou MOP (*Meta-Object Protocol*), que estrutura os objetos em dois níveis: nível base (objeto-base) e nível meta (meta-objeto) [7]. Cada objeto-base ***x*** está associado com um meta-objeto ^***x***, que representa aspectos estruturais e computacionais de ***x***, gerenciados através de computações feitas em ^***x***. As chamadas aos métodos do objeto-base são desviadas a fim de ativar meta-métodos que permitam a modificação do comportamento do objeto-base, ou a adição de funcionalidades a seus métodos (Fig. 6). Essa abordagem possibilita a separação dos aspectos funcionais (nível base) de uma aplicação dos não funcionais (nível meta). Os métodos da aplicação ficam então no nível base, enquanto seu controle e o gerenciamento ficam no nível meta. Assim, a reflexão abre a implementação de um sistema sem revelar detalhes desnecessários, fornecendo flexibilidade de implementação: alterar o gerenciamento a nível meta não implica em alterar, ou afetar, os algoritmos da aplicação no nível base [3].

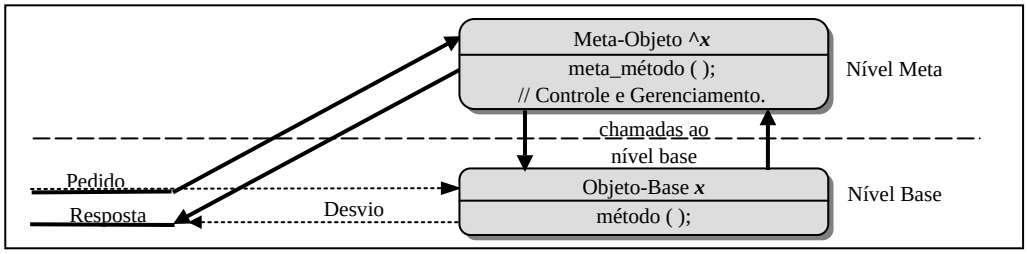

Fig. 6 – Reflexão Computacional.

3.1. Reificação

A reificação (*reification*), ou materialização, é o ato de converter algo que estava implícito, ou não expresso, em algo explicitamente formulado, então disponibilizado para manipulação conceitual, lógica ou computacional. É através do processo de reificação que o nível meta obtém as informações estruturais internas dos objetos do nível base (métodos e atributos). Porém, o comportamento do nível base, dado por interações entre objetos, não é completamente modelado apenas pela reificação estrutural dos objetos-base. É preciso intermediar as operações de nível base e transformá-las em informações manipuláveis pelo nível meta [10].

Seguindo esse raciocínio, a reflexão computacional pode ser dividida em dois tipos de funções [20][5]:

- **Introspecção** (*introspection*): também referenciada como reflexão estrutural (*structural reflection*), que refere-se ao processo de obter informação estrutural do programa e usá-la no próprio programa. Isso é possível através da representação em nível meta dessa informação, feita por um processo de reificação.
- **Intercessão** (*intercession*): também referenciada como reflexão comportamental (*behavioral reflection*), ou ainda interceptação[1], refere-se ao processo de alterar o comportamento do programa no próprio programa, através do ato de interceder, intermediar, ou interceptar [3] operações de nível base (invocação de método, ou assinalamento e obtenção de valores de atributos), que podem ser reificadas e então manipuladas pelo nível meta [10].

O programa que realiza a reflexão computacional, introspecção e/ou intercessão, é chamado de programa meta, enquanto o programa executado na computação reflexiva é chamado de programa base [20].

3.2. Considerações sobre a Reflexão Java

A reflexão computacional (comportamental e / ou estrutural) permite um alto grau de flexibilidade aos sistemas computacionais, pois possibilita a extensão de sua capacidade de gerenciamento e de sua adaptabilidade a novos requisitos – uma demanda sempre presente em sistemas de informação.

[1] De fato, o termo mais usado na literatura para designar reflexão comportamental é interceptação, porém sua semântica enfatiza bloqueio, obstáculo. Por essa razão, adotou-se o termo intercessão [5] [20], cujo significado – interceder, intermediar – é mais apropriado.

A plataforma J2EE já contém implicitamente características de introspecção (reflexão estrutural), fundamentadas sobre as APIs de Reflexão Java, que inclusive são utilizadas por ferramentas de desenvolvimento para levantar informações estruturais (métodos, atributos) dos componentes [12]. Contudo, a reflexão comportamental, que permite a intercessão de propósito geral a métodos, não é especificada diretamente pelo padrão Java. Essa é uma das principais motivações de estudos que propõem o acréscimo dessa característica ao padrão Java, como o metaXa [5], o OpenJava [20] e o Guaraná [10] – com o objetivo de permitir que sistemas baseados em Java se beneficiem de um mecanismo de reflexão unificado. Em síntese, essas arquiteturas reflexivas fundamentam-se em uma das seguintes abordagens:

- Abordagem MOP em Tempo de Execução (*Runtime* MOP), adotada pelo metaXa e pelo Guaraná, é caracterizada por um interpretador Java estendido (JVM estendida para poder suportar as características reflexivas), com habilidades de intercessão em tempo de execução; não altera a linguagem de programação Java base, nem o formato dos *bytecode*.
- Abordagem MOP em Tempo de Compilação (*Compile-Time* MOP), adotada pelo OpenJava, é caracterizada pela extensão da sintaxe da linguagem Java Padrão, a fim de prover intercessão e introspecção a uma aplicação em tempo de compilação – processo executado por um pré-processador de código fonte; apenas classes cujo código fonte está disponível podem tornar-se reflexivas.

Ainda, ao se decidir como utilizar reflexão na plataforma J2EE, deve-se ter em mente que, ao se trabalhar com componentes encapsulados, torna-se implícito que seu código fonte não está disponível. Qualquer acesso ao mesmo apenas se dá através de sua interface. Da mesma forma, extensões na JVM podem implicar em problemas de compatibilidade com a plataforma (p. ex., gerando conflitos com as atribuições do *container*). Por fim, soluções corporativas baseadas em componentes são intrinsecamente de menor granularidade que as soluções visadas pelas abordagens [5], [20] e [10]. Isto altera o enfoque objetivado pela aplicação: maior granularidade, menor modularidade, solução mais ad hoc; menor granularidade, maior modularidade, solução de maior abrangência, característica das soluções servidoras corporativas.

Ao mesmo tempo, o modelo de programação da J2EE favorece a montagem de aplicação pela combinação de blocos lógicos (seção 2.2.2), gerando a possibilidade de utilizar reflexão comportamental (intercessão) apenas nos chamados componentes de negócio – no caso, EJBs. Tal reflexão é restrita a um módulo funcional da aplicação, não estendendo-se a toda plataforma, como proposto pelas arquiteturas reflexivas metaXa, Guaraná e OpenJava – que propõem uma nova perspectiva para o padrão Java. Porém, ainda assim é possível alterar o comportamento da aplicação, uma vez que é na lógica de negócio que estão os métodos que definem seu domínio funcional. Consequentemente, mantém-se a idéia de um nível meta capaz de controlar a aplicação, com independência em relação ao nível base.

Enfim, o conjunto dessas características presumem uma alternativa para utilização de reflexão computacional na plataforma J2EE, ao menos para invocação de método de negócio (intercessão ou reflexão comportamental), abordagem denominada de **Intercessão em Tempo de Implantação**, cujo principal preceito é o de preservar a compatibilidade com o padrão J2EE. A seção 4 a seguir desenvolve essa proposta, através de um modelo de integração.

4. O Modelo de Integração

O modelo de integração proposto foi elaborado a partir da análise das arquiteturas reflexivas metaXa, Guaraná e OpenJava – cujas soluções orientaram a elaboração da abordagem MOP em Tempo de Implantação (*Deploy-time* MOP), no sentido de evidenciar quais restrições as mesmas teriam na plataforma J2EE, e quais alternativas poderiam ser consideradas. Assim, elegendo a preservação das características e vantagens do contexto J2EE como principal preceito da proposta, foi possível verificar que os princípios do OpenJava, MetaXa e Guaraná não poderiam ser adotados, uma vez que ou propõem novos padrões Java, ou se baseiam em práticas nem sempre acessíveis na plataforma J2EE. Tal situação é melhor explicada a partir dos seguintes fatores da J2EE:

(1) Aplicações J2EE trabalham com componentes de negócio que, a princípio, são encapsulados – os COTS (*Components Off-The Shelf*), que não disponibilizam seu código fonte; e

(2) Modificações nos padrões Java (p. ex. extensões na JVM) podem implicar em problemas de compatibilidade com a plataforma, gerando comprometimento de seu ambiente Java corporativo, consistente e integrado.

Por conseguinte, ao se considerar a plataforma J2EE, é possível inferir que o fator (1) impossibilita a abordagem MOP em Tempo de Compilação, e o fator (2) não recomenda a abordagem MOP em Tempo de Execução. Em resumo, os fatos e decisões que orientaram a confecção da proposta podem ser resumidos como na Tab. 1 a seguir.

CONDUÇÃO DA PROPOSTA	
Motivação	A reflexão comportamental, que permite a intercessão de propósito geral de métodos, não é ofertada diretamente pela especificação Java.
Restrição	Ao se trabalhar com componentes EJB prontos (componentes de prateleira, encapsulados), não se tem acesso ao seu código fonte. Logo, não é possível manipular ou estender código dos componentes para alterar suas funcionalidades.
Preceitos	Não alterar as características da plataforma J2EE. Apenas usar os princípios definidos por sua arquitetura, mantendo compatibilidade a linguagem Java, *bytecode*, JVM, APIs Enterprise Java e ambiente corporativo consistente
Formalização da Proposta	Proporcionar intercessão (reflexão comportamental) de propósito geral aos métodos dos componentes de negócio da Plataforma J2EE, visando flexibilidade de implementação nas situações onde é necessário introduzir controle e / ou alterar o comportamento da aplicação corporativa – através da abordagem denominada de **Intercessão em Tempo de Implantação,** ou mais genericamente, MOP em Tempo de Implantação.

Tab. 1 – Fatos e decisões que orientaram a Proposta.

4.1. Convergência de Vantagens

Para valer-se tanto das vantagens da reflexão computacional quanto da plataforma J2EE, é preciso primeiramente atender aos fatores (1) e (2), levantados no início da seção 4. O modelo de programação da plataforma J2EE responde a esses quesitos, pois, ao favorecer a

montagem de aplicação pela combinação de blocos lógicos, gera a possibilidade de utilizar reflexão comportamental (intercessão) apenas na camada de negócio, representada pelos componentes de negócio. É um mecanismo reflexivo restrito apenas a um módulo da aplicação, mas que ainda assim permite a alteração do comportamento de toda a aplicação, uma vez que é a camada de negócios que define sua funcionalidade básica.

A Tab. 2, a seguir, resume quais recursos da plataforma J2EE foram adotados para efetivar o modelo de integração, de forma a atender aos preceitos expostos.

RECURSOS J2EE QUE APOIAM A PROPOSTA	
Modelo de Programação	O Modelo de Aplicação Multi-camada, auxiliado pelo *pattern* MVC, favorece a estratificação da aplicação em camadas lógicas: a lógica de negócio (ou os aspectos funcionais da aplicação corporativa) fica isolada na Camada de Negócio, representada pelos componentes de negócio. É nessa camada lógica que a proposta se concentra.
Componentede Negócio	Os enterprise JavaBeans, ou EJBs, são o tipo de componente J2EE destinado a guardar a lógica de negócio da aplicação corporativa; ou seja, são os EJBs que definem a Camada de Negócio da aplicação J2EE. Por conseqüência, a intercessão de seus métodos proporciona a habilidade de alterar o comportamento de toda a aplicação.
Deployment Descriptors	A facilidade de composição da aplicação corporativa pela edição das suas declarações XML (existentes nos *deployment descriptors*) permite a alteração, ou redirecionamento, da referência a componentes EJB (nome JNDI) no momento da montagem, ou composição da aplicação. Isso possibilita a inserção de um nível meta de controle através de um outro componente – denominado meta-componente.

Tab. 2. Principais recursos da plataforma J2EE que sustentam a Proposta.

As decisões de projeto, exibidas nas Tabs. 1 e 2, permitem a definição do mecanismo de suporte para a proposta: o meta-componente – detalhado na Tab. 3.

META-COMPONENTE	
Princípio	Princípio MOP: provê um nível meta (meta-componente) capaz de controlar a aplicação, com independência em relação ao nível base (componente de negócio base), e que é habilitado em tempo de implantação.
Implementação	Deve ser inserido no momento de montagem da aplicação, em tempo de implantação, via redirecionamento (alteração de nome JNDI no *deployment descriptor*), e entre a chamada cliente e o componente de negócio base.
Atribuição	Prover novas funcionalidades (controle, gerência, alteração de comportamento, etc.) à aplicação corporativa J2EE, sem necessidade de acesso ou alterações no código fonte do componente de negócio base.

Tab. 3 – Atribuições e princípio de funcionamento do Meta-componente.

4.2. Detalhamento do Modelo de Integração

Este modelo de integração adota o Modelo de Aplicação Multi-camada, auxiliado pelo *pattern* MVC. A intercessão de funcionalidade é realizada como descrito a seguir.

Passos	DESCRIÇÃO
1º	As novas funcionalidades da aplicação J2EE devem ser implementadas na forma de um novo EJB, o meta-componente, que deve ter as mesmas interfaces (*Home* e *Remote*) do EJB base.
2º	O meta-componente deve ser inserido na aplicação J2EE no momento da sua montagem, sendo referenciado com o nome JNDI original do componente EJB base – este último deve adotar um novo nome JNDI dentro do contexto da aplicação (redirecionamento de componentes pela alteração de seus nomes JNDI).
3º	O meta-componente deve se encarregar de chamar, quando e se necessário, o componente EJB base – logo deve saber o novo nome JNDI do EJB base.
4º	O *namespace* (*package* Java) do componente EJB base não é alterado, pois está definido no código fonte; assim, para o correto redirecionamento dos componentes, o *namespace* do meta-componente deve ser referenciado no componente que representa o C*ontroller* no MVC do Modelo de Aplicação Multi-camada, ou seja, no JavaBean – que é quem de fato invoca os componentes de negócio (*Model* no MVC) em nome dos clientes.

Tab. 4 – Seqüência para intercessão de funcionalidade em aplicações J2EE.

Observa-se que a inserção do meta-componente na aplicação J2EE não é percebida pelo cliente, pois o redirecionamento de nomes acontece na camada de serviços da aplicação corporativa, orientada por sua vez pelo MVC: componentes Web (o *View*), componentes JavaBeans (o *Controller*), meta-componente EJB e o componente EJB base (o *Model*).

4.3. Representação Gráfica

A seqüência de passos da Tab. 4 é ilustrada pelas Fig. 7 e Fig. 8. A primeira, Fig. 7, exibe o modelo da aplicação J2EE **sem** a inserção do meta-componente. A evolução desse sistema, pela inclusão de novas funcionalidades e / ou controle através de um meta-componente, é exibida na Fig. 8.

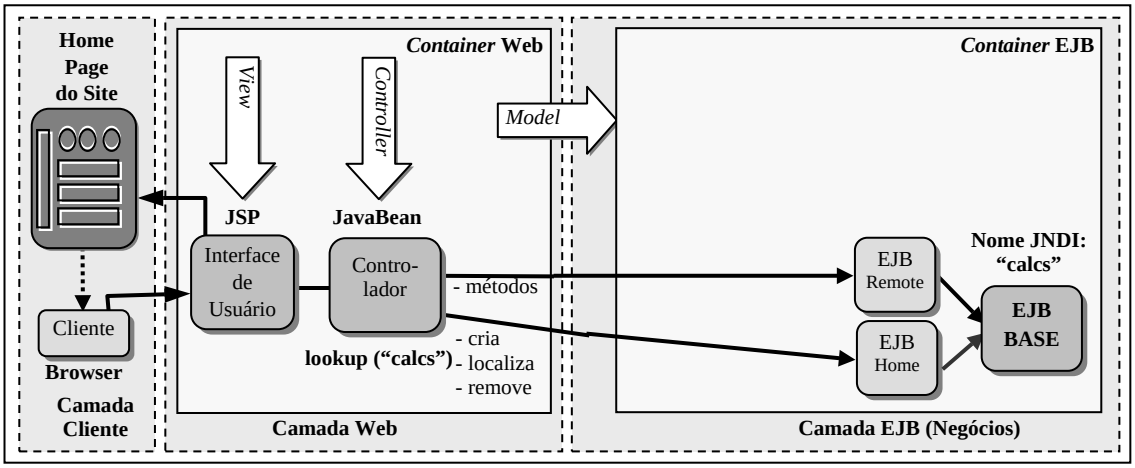

Fig. 7 – Modelo de Aplicação Multi-camada, com o *pattern* MVC.

A Fig. 8 exibe o meta-componente EJB desviando as chamadas ao componente EJB base, pois está identificado com o nome JNDI original e tem suas interfaces idênticas às do componente base – provê assim um nível meta capaz de controlar o nível base.

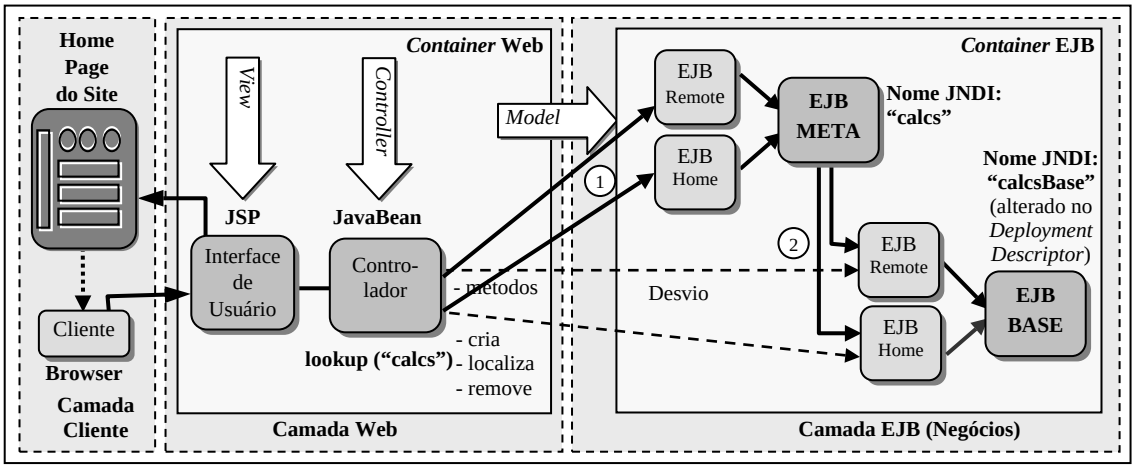

Fig. 8 – Proposta para intercessão de funcionalidade em aplicações J2EE.

4.4. Exemplo de Aplicação

A Fig. 9 apresenta um diagrama UML (*Unified Modeling Language*) de componentes de uma aplicação J2EE original, onde se quer adicionar controle para contabilização de acessos aos seus métodos de negócio – através de meta-componentes EJB. Esta situação reflete o modelo de aplicação ilustrado na Fig. 7.

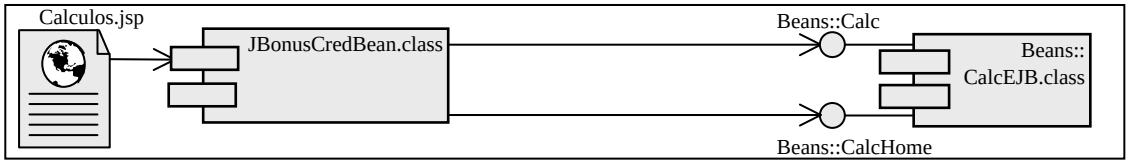

Fig. 9 – Aplicação J2EE original , sem controle de contabilização.

A Fig. 10 apresenta um diagrama UML de componentes da aplicação J2EE, já com o controle de contabilização. Nele, é enfatizada a construção física da aplicação, indicando como está definida a montagem de seu executável.

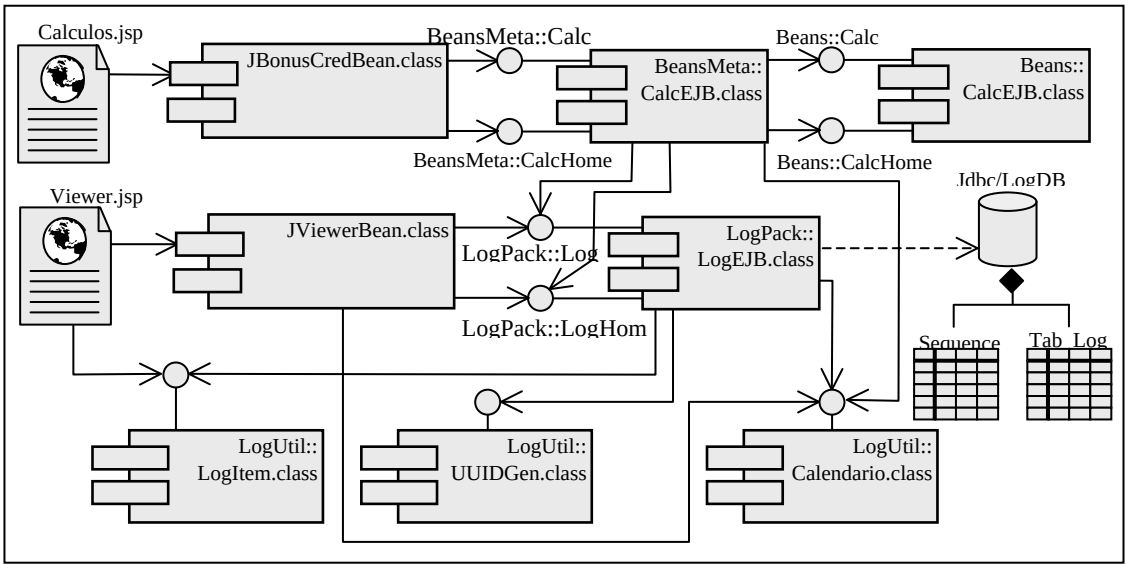

Fig. 10 – Aplicação J2EE com inserção de controle, via Intercessão em Tempo de Implantação.

É possível destacar a estratificação da aplicação da Fig.10 da seguinte maneira: funcionalidade base (pacote Java "Beans"); funcionalidade de controle (pacotes Java "BeansMeta", "LogPack" e "LogUitl"). A funcionalidade de controle, por sua vez, é inserida com a ajuda da abordagem de Intercessão em Tempo de Implantação (comparar com a Fig. 8), cujas diretrizes encontram-se na Tab. 4.

Ainda na aplicação da Fig. 10, foram utilizados pacotes Java adicionais (que também fazem parte da funcionalidade de nível meta) para manipular e armazenar as informações de contabilização de acesso aos métodos de negócio da aplicação, através de um banco de dados, representado pelo JNDI "Jdbc/LogDB". Finalmente, toda essas novas funcionalidades são adicionadas à aplicação original (Fig. 9) sem modificar, ou afetar, o componente EJB base.

A Fig. 11 apresenta como uma ferramenta de implantação (que já vem embutida no J2EE SDK) auxilia o redirecionamento de nomes JNDI. Os campos definidos no painel da ferramenta são salvos no arquivo descritor de implantação (XML) em tempo de implantação.

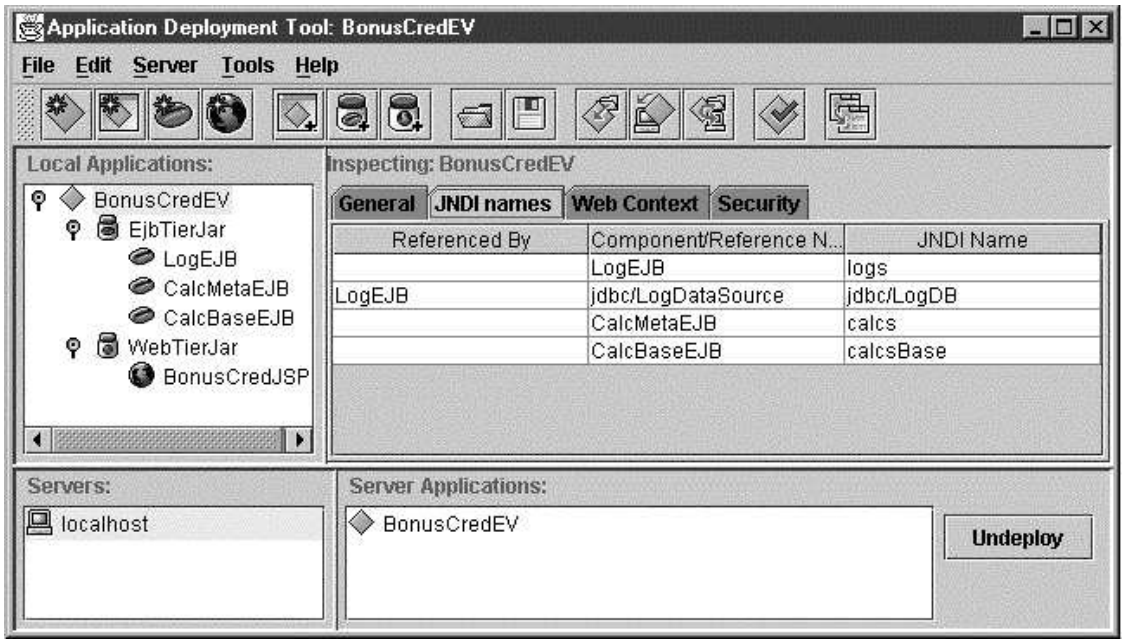

Fig. 11 – Redirecionamento de nomes JNDI na aplicação de contabilização.

Essa aplicação de contabilização é de fato bem simples, sendo sua principal pretensão a de ilustrar as perspectivas de utilização oferecidas por pela abordagem de Intercessão em Tempo de Implantação, cujas diretrizes proporcionam uma forma modular e esquematizada de modificar o comportamento de uma aplicação J2EE. Adicionalmente, também pretende-se evidenciar mais concretamente, através do exemplo implementado, as vantagens de montagem de aplicação oferecidas pela plataforma J2EE – um dos seus grandes atrativos.

5. Conclusões

Este trabalho de pesquisa apresentou um empenho no sentido de projetar e implementar uma proposta que permita a utilização de características de reflexão computacional em aplicações servidoras. Mais especificamente, foi dada ênfase a aplicações corporativas que concentram-se em interoperabilidade e portabilidade, dentro de um contexto evolutivo que admita escalabilidade e desempenho. Todas essas características são pertinentes

à plataforma J2EE, e determinaram sua escolha como plataforma base para a proposta desta abordagem.

Como conseqüência, o estudo dos requisitos para utilização de reflexão na plataforma J2EE permitiu realizar uma minuciosa avaliação da mesma, no sentido de levantar suas restrições e validar suas propriedades, a fim de utilizá-las para atingir os objetivos deste trabalho, que propõem a abordagem Intercessão em Tempo de Implantação.

5.1. Intercessão em Tempo de Implantação

O modelo de integração apresentado na seção 4 foi confeccionado com base na análise comparativa das arquiteturas reflexivas metaXa [5], Guaraná [10] e OpenJava [20]. Suas abordagens MOP em Tempo de Execução (metaXa e Guaraná) e MOP em Tempo de Compilação (OpenJava) não são recomendadas para a plataforma J2EE, justamente por propor um novo padrão ou por se basear em práticas nem sempre possíveis na plataforma. Assim, elegendo-se a preservação das características da J2EE como o principal preceito da proposta, e objetivando aliar as vantagens da reflexão com as da plataforma J2EE, foram utilizadas as facilidades para composição de aplicação, recomendadas pelo Modelo de Programação Multi-camada da J2EE. Ao favorecer o desacoplamento de funcionalidade em componentes lógicos, esse modelo possibilita a utilização de reflexão comportamental (intercessão) apenas na camada de negócio, que encerra a funcionalidade básica da aplicação, e é representada pelos componentes EJBs. Por conseguinte, interceptar os métodos dos EJBs – através da inserção (pelo redirecionamento de nomes) de um meta-componente EJB – permite a alteração do comportamento da aplicação como um todo. Ou seja, gera uma alternativa para utilizar princípios de reflexão computacional na J2EE, através da abordagem MOP em Tempo de Implantação (*Deploy-Time* MOP), ou mais especificamente Intercessão em Tempo de Implantação.

5.2. Resultados

Valendo-se das premissas expostas na seção 5.1, o modelo de integração apresentado atingiu seu objetivo, uma vez que conseguiu implementar essa característica da reflexão computacional – intercessão implementada em tempo de implantação –, verificando-a na prática. Ou seja, para validar proposta, uma aplicação J2EE foi desenvolvida (seção 4.4), de acordo com o esquema da Fig. 8 e com os procedimentos indicados na Tab. 4. Tal aplicação utiliza então a proposta Intercessão em Tempo de Implantação para realizar a contabilização dos acessos aos métodos de um componente de negócio, inserindo essa funcionalidade de controle na aplicação através de um meta-componente – sem alterar, ou mesmo afetar, os algoritmos que definem sua funcionalidade básica.

Em suma, o modelo de integração proposto pela abordagem de Intercessão em Tempo de Implantação é endereçado a aplicações que contenham componentes de negócio EJB encapsulados (COTS – *Components Off-The-Shelf*) e visa permitir, a partir de procedimentos sistematizados e auxiliados por ferramenta de implantação, que a aplicação como um todo possa ser alterada de forma mais elegante e modular, mantendo intactas suas funcionalidades básicas.

Tais facilidades, portanto, provêem como resultado maior flexibilidade de desenvolvimento e implementação nas situações onde é necessário introduzir controle e / ou modificar a funcionalidade da aplicação (alteração de comportamento ou adaptação a novos requisitos), que é precisamente a meta almejada pela proposta da abordagem.

Por fim, cabe ressaltar que a plataforma J2EE de fato comprovou ser, através deste trabalho de pesquisa, uma alternativa altamente promissora para ambientes corporativos. A organização de seu conjunto de serviços e facilidades oferece um leque de vantagens cada vez mais robusto. Consequentemente, o domínio das características e capacidades desta plataforma pode ser encarado como um diferencial positivo, visto que seus preceitos estão contribuindo ativamente para definir a evolução do futuro da computação, principalmente no que diz respeito a aplicações corporativas.

Referências Bibliográficas

[1] BOOCH, Gary, JACOBSON, Ivar, RUMBAUGH, James. **UML – Guia do Usuário**. Ed. Campos. Rio de Janeiro, 2000.

[2] BRAY, Tim, PAOLI, Jean, SPERBERG-McQUEEN, C. M. **Extensilble Markup Language (XML) 1.0**. W3c Recommendation. Fev. 1998. [online] http://www.w3.org/TR/1998/REC-xml-19980210

[3] FABRE, J., NICOMETTE, V. et. Al. **Implementing Fault Tolerant Applications Using Reflective Object-Oriented Programming**. Proceedings of th 25th IEEE International Symposium on Fault-Tolerant Computing, Pasadena, CA, EUA, Jun. 1995.

[4] GAMA, E., HELM, R. et Al. **Design Patterns – Elements of Reusable Object-Oriented Software**. .Addison-Wesley Longman Inc., EUA, 1995.

[5] GOLM, Michael, KLEINÖDER, Jürgen. **metaXa and the Future of Refletion**. OOPSLA'98 Workshop on Reflective Programming in C++ and Java, Vancouver, Canadá, Out. 1998. http://www4.informatik.uni-erlangen.de/Publications/pdf/Golm-metaXa_and_the_Future_of_Reflection.pdf

[6] KICZALES, Gregor, ASHLEY, J. Michael et. Al. **Metaobject Protocols: Why We Want Them, and What Else They Can Do**, publicado no *Object-Oriented Programming: The CLOS Prospective*, págs. 101-118, Andreas Paepcke, Ed., MIT Press, Cambridge, MA, EUA, 1993.

[7] LAU C. L. **Implementação de Técnicas de Replicação de Componentes de Software sobre Plataforma Aberta CORBA**. Dissertação submetida à UFSC para obtenção de grau de Mestre em Engenharia Elétrica. Florianópolis, Mai. 1996.

[8] MAES, Pattie. **Concepts and Experiments in Computacional Reflection**. OOPSLA'87 Proceedings, EUA, Out. 1987.

[9] MELEWS, Deborah. **CORBA and EJB Team UP in Data Center**. Application Development Trends Magazine, EUA, Jul. 2000 [online] http://www.adtmag\Pub\article.asp?ArticleID=881.

[10] OLIVA, Alexandre. **Guaraná: Uma Arquitetura de Software para Reflexão Computacional Implementada em Java™**. Dissertação submetida à UNICAMP para obtenção de grau de Mestre em Ciência da Computação. Campinas, Ago. 1998.

[11] OBJECT MANAGEMENT GROUP. **CORBA Components**. OMG Document, *orbos/99-02-05*, EUA, Fev. 1999.

[12] ORFALI, Robert, HARKEY, Dan. **Client/Server Programming with Java and CORBA - Second Edition**. John Wiley & Sons Inc. EUA, 1998.

[13] SOBEL, Jonathan, FRIEDMAN, Daniel P.. **An Introduction to Reflection-Oriented Programming**. Reflection'96, San Francisco, CA, EUA, Abr. 1996. http://www.cs.indiana.edu/hyplan/jsobel/rop.html

[14] SUN MICROSYSTEMS. **Designing Enterprise Applications with the Java™ 2 Platform, Enterprise Edition**. Sun Microsystems, Inc., EUA, Mar. 2000. http://java.sun.com/j2ee

[15] SUN MICROSYSTEMS. **Java™ Core Reflection API**, Sun Microsystems, Inc., EUA, 1997. http://java.sun.com/products/jdk/1.3/docs/guide/reflection/index.html

[16] SUN MICROSYSTEMS. **Enterprise JavaBeans™ Specification, v1.1.**, Sun Microsystems, Inc., EUA, Dez. 1999. http://java.sun.com/products/ejb

[17] SUN MICROSYSTEMS. **Java™ 2 Platform Enterprise Edition Specification, v1.2.**, Sun Microsystems, Inc., EUA, Dez. 1999. http://java.sun.com/j2ee/doc.html

[18] SUN MICROSYSTEMS. **The J2EE Application Programing Model**, Sun Microsystems, Inc., EUA, Sep. 1999. http://java.sun.com/j2ee/doc.html

[19] SZYPERSKI, Clemens. **Component Software: Beyond Object-Oriented Programming**. Addison-Wesley Pub., EUA, 1998.

[20] TATSUBORI, Michiaki. **An Extension Mechanism for the Java Language**. Dissertação de Mestrado em Engenharia, Universidade de Tsukuba, Ibaraki, Japão. Fev. 1999. http://www.hlla.is.tsukuba.ac.jp/~mich/openjava/papers/mich_thesis99.pdf

[21] THOMAS, Anne. **Enterprise JavaBeans™ Technology - Server Component Model for the Java™ Platform**. Patricia Seybold Group, EUA, Dez. 1998.

[22] THOMAS, Anne. **Java™ 2 Platform, Enterprise Edition - Ensuring Consistency, Portability, and Interoperability**. Patricia Seybold Group, EUA, Jun. 1999.