Uma Abordagem Reflexiva para Replicação de Componentes Servidores da Plataforma Java para Corporações

Cristina Verçosa Pérez Barrios de Souza
Prog. Pós-Grad. em Informática Aplicada – PPGIA
Pont. Univ. Católica do Paraná – PUCPR
Curitiba – PR
e-mail: cristina@ppgia.pucpr.br

Carlos Alberto Maziero
Prog. Pós-Grad. em Informática Aplicada - PPGIA
Pont. Univ. Católica do Paraná – PUCPR
Curitiba – PR
e-mail: maziero@ppgia.pucpr.br

Resumo

Esse artigo objetiva apresentar a possibilidade de inclusão da reflexão computacional na plataforma J2EE, mantendo todas as suas de características de interoperabilidade, portabilidade e consistência de ambiente. Para tanto, pretende-se utilizar a habilidade de composição das aplicações J2EE, em conjunto com suas instruções de implantação (*deployment descriptors*), de forma a possibilitar a alteração do comportamento de componentes servidores, incluindo assim algum grau de flexibilização de implementação por parte do implantador da aplicação.

Uma vez estabelecida uma forma adequada de incluir reflexão na plataforma J2EE, pretende-se validar a proposta com a implementação de mecanismos de replicação em componentes servidores de negócio, visando tolerância a falhas e disponibilidade.

Palavras-chave: Plataforma J2EE, reflexão, replicação.

1 . Introdução

A influência da Internet aumentou a necessidade por uma maior e melhor interação entre aplicações distribuídas. Dentro desse contexto, para que as aplicações servidoras ofereçam uma solução distribuída de grande alcance, é preciso que possuam duas habilidades básicas: interoperabilidade e portabilidade. A sofisticação dessas habilidades pode determinar a utilidade e a longevidade das aplicações servidoras, que ganharam um grande impulso com a especificação da Plataforma Java para Corporações. Também denominada de plataforma J2EE, ela define uma arquitetura Java unificada, baseada em componentes, que promove interoperabilidade, portabilidade e ambiente de execução integrado e consistente para aplicações corporativas.

O paradigma reflexivo, por sua vez, torna possível a uma aplicação controlar seu próprio comportamento atuando sobre si mesma. Seria então desejável que as computações se beneficiassem de tais capacidades reflexivas, obtendo autoconsciência de seu comportamento e estado, alterando-os através de um exame que obtivesse e fizesse uso da informação de nível meta nas decisões sobre o que fazer em seguida [7].

No que se refere à Plataforma Java, tais capacidades reflexivas integrais não estão completamente especificadas, sendo esse o tema de estudo de várias abordagens [2][5][13]. No entanto, essas propostas implicam em um alto grau de alteração na especificação Java, o que tem como conseqüência o comprometimento das vantagens corporativas oferecidas pela plataforma J2EE. Contudo, a J2EE define facilidades para composição de aplicação, favorecendo o desacoplamento de funcionalidade em componentes lógicos e encorajando a reutilização de código orientado a componente. Tais características, aliadas à facilidade de alterar a referência a componentes oferecida pela edição das suas entradas nos *deployment descriptors* (arquivos XML - *eXtensible Markup Language* - usados para configuração de controle e gerenciamento), permitem customizar o comportamento do componente através da montagem de aplicação. Tal construção flexível de aplicações servidoras pode ser utilizada para facilitar a implementação de mecanismos de tolerância a falhas em aplicações distribuídas.

2 . Plataforma Java

A tecnologia de componentes permitiu o desenvolvimento de complexos sistemas de informação através da combinação e extensão de blocos reutilizáveis de software. Nessa linha, os JavaBeans emergiram como um importante padrão de componentes Java para aplicações cliente, com as vantagens de portabilidade e desenvolvimento em ferramentas visuais [6]. Todavia, eles 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 pela JVM, mas sim por uma infra-estrutura de objetos distribuídos [6].

2.1. Enterprise JavaBeans

A especificação da arquitetura Enterprise JavaBeans (EJB) estende o modelo de componentes JavaBeans para suportar componentes servidores, pois considera a existência de um conjunto de serviços de informação essenciais (nome, transação, segurança, etc.) providos por uma infra-estrutura de objetos distribuídos

(p. ex. CORBA). Em síntese, os EJBs definem uma arquitetura para computação distribuída, baseada em componentes servidores, que habilita o desenvolvimento e a implantação de aplicações de negócios [10].

Um componente enterprise bean é implantado em um *container*, que pode estar dentro de qualquer servidor EJB. Essa portabilidade é garantida por contratos - implementados por interfaces - entre o *container* e o enterprise bean, e entre o cliente e o *container* (Fig. 1). O *container* invoca essas interfaces *wrappers* (que "envolvem" o componente) em tempo de execução. Logo, o cliente não interage diretamente com o enterprise bean, mas sim com essas interfaces, cujas classes são geradas automaticamente pelo próprio *container*. É assim que o *container* intercepta as chamadas a métodos e insere serviços de gerenciamento. Essas interfaces são [14]:

- **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* faça gerenciamento de estado, controle de transação, serviços de segurança e de persistência.

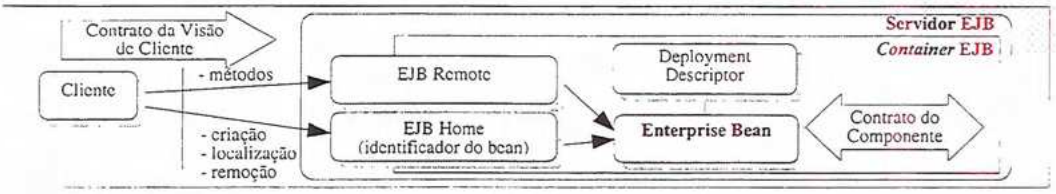

Fig. 1: O Enterprise JavaBean *Container*.

As regras associadas com o gerenciamento de ciclo de vida, transações, segurança e persistência de um EJB estão definidas em um arquivo associado denominado *deployment descriptor* (descritor de implantação, ou entrada de ambiente, escrito em XML). Essas regras são declaradas em tempo de desenvolvimento. Em tempo de execução, o *container* executa serviços de acordo com os valores descritos por esse arquivo [14].

2.2. Plataforma J2EE

A especificação da Plataforma Java para Corporações, denominada Java™ 2 *Platform, Enterprise Edition* (J2EE), define uma arquitetura Java unificada, baseada em componentes, centrada em serviços e habilitada para aplicações multi-camada, fornecendo suporte de ambiente à arquitetura EJB. Seu lançamento agrega uma série de especificações, dentre um conjunto de lançamentos [8][11][12][15]:

- **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.

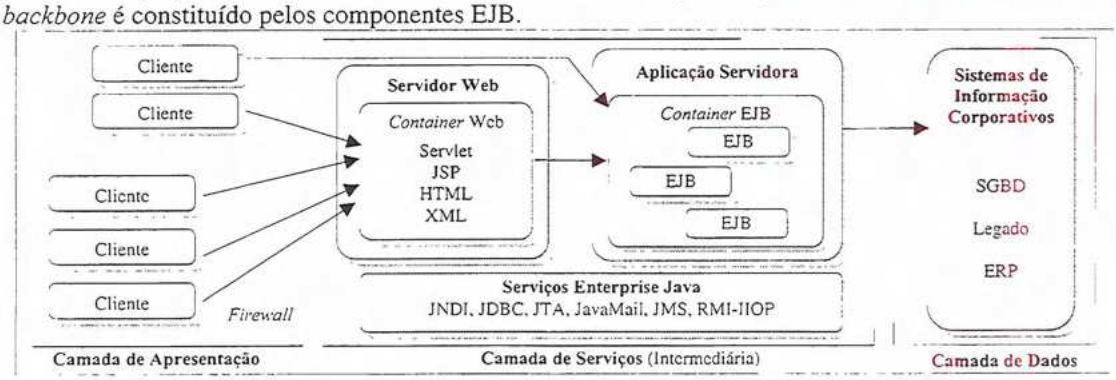

Fig. 2: Ambiente multi-camada J2EE.

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 [8], de acordo com os seguintes agrupamentos:

- **Um Enterprise Bean** é empacotado em um arquivo **.jar**, junto com suas classes e *deployment descriptor*.
- **Um Componente Web** - página JSP (Java Server Page) ou Servlet - é empacotado em um *Web Archive* (Arquivo Web), cuja extensão é **.war**, junto com seus arquivos (HTML, .GIF, etc.) e *deployment descriptor*.
- **Uma Aplicação Cliente** e seu *deployment descriptor* também são empacotados em um arquivo **.jar**.
- **A Aplicação Corporativa**, com todos os seus módulos mais o *deployment descriptor* da aplicação, é empacotada em um arquivo tipo **.jar**, *Enterprise Archive* (Arquivo Corporativo), cuja extensão é **.ear**.

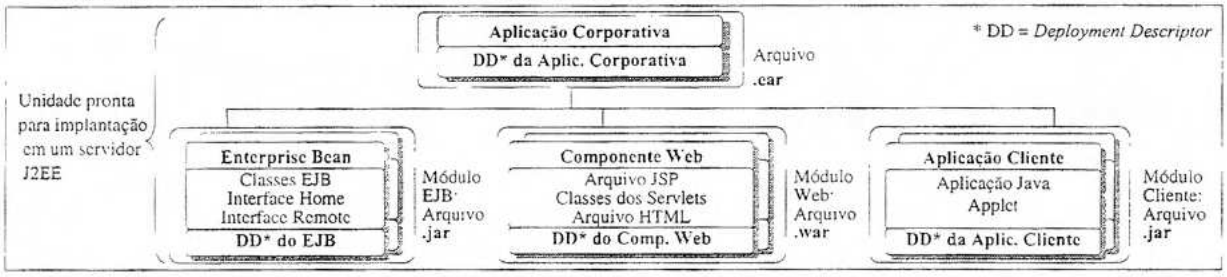

Fig. 3: Aplicação Corporativa.

A customização da implantação de uma aplicação J2EE pode ser feita pela edição dos arquivos texto XML - apesar dessa tarefa ser mais confiável se feita por ferramentas especializadas. Os *deployment descriptors* gerados automaticamente também podem ser editados para alterar as funcionalidades dos componentes [8].

2.2.2. Modelo de Programação J2EE

O modelo de programação J2EE fundamenta-se na integração de camadas - montagem da aplicação. É dessa forma 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 implícitas favorecendo um cenário em detrimento de outro [8]. Dentre os cenários suportados, há destaque para o cenário denominado de Modelo de Aplicação Multi-camada (Fig. 4), cuja habilidade maior está no fato dele desacoplar o acesso a dados de questões de como realizar interações com o usuários, propiciando escalabilidade.

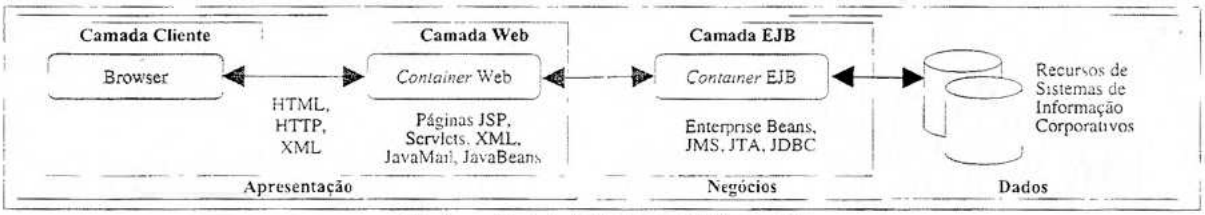

Fig. 4: Modelo de Aplicação Multi-camada.

O modelo de programação J2EE também adota o *pattern* MVC (*Model-View-Controller*), 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.

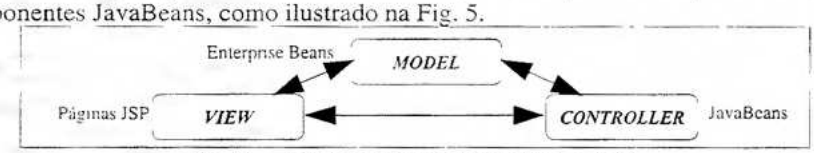

Fig. 5: MVC em aplicações J2EE Multi-camada.

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. Esses 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 [1].

Quando a reflexão é aplicada à programação orientada a objetos, tem-se a abordagem de meta-objetos [3], que estrutura os objetos em dois níveis: nível base (objeto-base) e nível meta (meta-objeto) [4]. 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 [1].

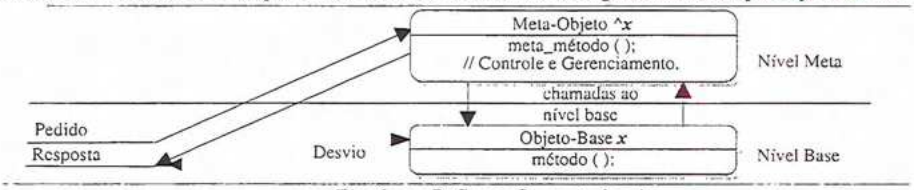

Fig. 6: Reflexão Computacional.

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 [5].

Seguindo esse raciocínio, a reflexão computacional pode ser dividida em dois tipos de funções [13][2]:

- **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*), refere-se ao processo de alterar o comportamento do programa no próprio programa, através do ato de interceder, intermediar, ou interceptar [1] 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 [5].

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 [13].

3.2. Reflexão Java - Considerações

Existem características implícitas de reflexão estrutural (introspecção) na especificação Java da Sun - fundamentadas sobre as APIs Java de Reflexão. Contudo, a reflexão comportamental, que permite a intercessão de propósito geral de métodos, não é ofertada diretamente pela especificação. Essa é uma das principais motivações de estudos que propõem o acréscimo dessa característica ao padrão Java (metaXa [2], OpenJava [13] e Guaraná [5]), com o objetivo de permitir que sistemas baseados em Java se beneficiem de um mecanismo de reflexão unificado.

A reflexão, comportamental e estrutural, como mencionado anteriormente, permite um alto grau de flexibilidade aos sistemas de aplicação. O ideal seria prover as vantagens reflexivas com uma alteração mínima na plataforma Java, garantindo compatibilidade com os sistemas de aplicação existentes.

Ao se decidir como proporcionar reflexão na plataforma J2EE, deve-se ter em mente que, ao se trabalhar com componentes, torna-se implícito que seu código fonte não está disponível (como proposto em [13]). Qualquer acesso ao mesmo apenas se dá através de sua interface. Da mesma forma, extensões na JVM (como proposto em [2] e [5]) podem implicar em problemas de compatibilidade com a plataforma J2EE, comprometendo um dos seus maiores trunfos, que é o ambiente Java corporativo, consistente e certificado.

4 . Replicação Através da Reflexão

As técnicas de replicação são uma alternativa para que os serviços em um sistema de informação continuem a ser fornecidos mesmo na ocorrência de falhas - maior confiabilidade e disponibilidade. Uma das formas de implementar técnicas de replicação é através da utilização da reflexão computacional.

Em sistemas distribuídos, as unidades de replicação - réplicas - são distribuídas em diversos pontos da rede. A coordenação da replicação define protocolos que asseguram a consistência e a transparência do conjunto, bem como o controle de concorrência e a recuperação em caso de falha. Isso habilita a noção de serviço abstrato para o cliente, pois o que é visto como um único serviço, na realidade é um grupo de serviços replicados [4].

Existem várias abordagens que são capazes de mascarar falhas individuais de réplicas membros de um conjunto de serviços replicados, como por exemplo a replicação passiva e a replicação ativa.

A validação da proposta desse artigo baseia-se na implementação do mecanismo de replicação passiva de componentes servidores da plataforma J2EE, visando tolerância a falhas. Nessa estratégia de replicação, apenas uma réplica - a primária - recebe, executa e responde às invocações dos clientes. As restantes - réplicas *backups* - têm a função de substituir a réplica primária em caso de falha. A consistência do grupo é assegurada pelo primário, que envia mensagens de *checkpoints* para as *backups* (Fig. 7). No caso de falha, é realizada uma eleição e o *backup* selecionado assume a partir do *checkpoint* mais recente.

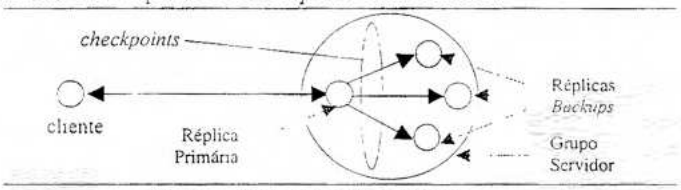

Fig. 7: Grupo de réplicas passivas.

Essa técnica de replicação pode ser implementada em um nível meta. O desenvolvedor decide quais métodos do nível base devem ser replicados - tipicamente os que modificam o estado da aplicação -, e o nível meta controla o objeto reflexivo primário, interceptando as invocações aos métodos reflexivos e manipulando, portanto, o protocolo de comunicação entre as réplicas. Assim, o nível base é idêntico no servidor primário e nas réplicas *backup*, porém as ações realizadas por ambos no nível meta são diferentes: o primário envia *checkpoints* para as réplicas *backup* após cada invocação de método reflexivo, e as mesmas processam esses *checkpoints* [1].

5 . Conclusões

Como mencionado anteriormente, a plataforma J2EE já contém características de introspecção - fundamentadas sobre as APIs Java de Reflexão [9] -, que inclusive são utilizadas por ferramentas de desenvolvimento para levantar informações estruturais (métodos, atributos) dos componentes [6]. Ao mesmo tempo, a J2EE adota um modelo de Aplicação Multi-camada que, auxiliado pelo *pattern* MVC, favorece a habilidade de desacoplar a funcionalidade da aplicação em componentes lógicos. Assim, ao se escolher esse modelo de aplicação, tem-se o isolamento da lógica de negócio (ou dos aspectos funcionais da aplicação corporativa) na Camada de Negócio, gerando a possibilidade de implementação da reflexão computacional apenas sobre os componentes de negócio - no caso, EJBs. Tal reflexão estaria restrita a um determinado módulo funcional da aplicação, não estendendo-se a toda plataforma, como proposto por outras abordagens de reflexão Java [2][5][13]. Entretanto, isso não invalida a capacidade de alteração de comportamento da aplicação, uma vez que é na lógica de negócio que estão os métodos e procedimentos que definem o domínio funcional da aplicação. Por conseguinte, mantém-se a idéia de um nível meta capaz de controlar a aplicação, com independência em relação ao nível base.

Também é importante ressaltar que o Implantador / Montador de Aplicação J2EE, ao trabalhar com componentes EJB, não tem acesso ao seu código fonte, sendo que para realizar alteração de sua a funcionalidade - visando alteração de seu comportamento ou adaptação a novos requisitos de aplicação - não é possível trabalhar com extensão de código (proposta de reflexão comportamental em [13]). Contudo, existe a facilidade de se alterar, ou redirecionar, a referência a componentes EJB (alteração de nome JNDI) no momento da montagem da aplicação (edição de entradas nos *deployment descriptors*), inserindo assim a figura de um meta componente.

Tais características em conjunto permitem realizar uma composição de aplicação que proporcione intercessão de propósito geral aos método dos componentes de negócio da J2EE, gerando abertura para implementação da reflexão comportamental. Essa é, portanto, a abordagem adotada para implementar mecanismos de replicação visando tolerância a falhas. Nela não há alterações das características da plataforma J2EE, apenas a utilização dos princípios definidos por sua arquitetura. Consequentemente, mantém-se compatibilidade com o padrão J2EE: linguagem Java padrão, *bytecode*, JVM, APIs Enterprise Java e ambiente corporativo consistente.

5.1. Estado Atual do Trabalho

O presente projeto, conforme justificado acima, adota o Modelo de Aplicação Multi-camada da J2EE e atualmente encontra-se no estágio de estudo da viabilidade da proposta de intercessão de método de negócio. Tal proposta está ilustrada na Fig. 8, onde a intercessão de funcionalidade é realizada:

- pela inclusão da figura de um meta componente, com as novas funcionalidades que se quer adicionar;
- pela inclusão de componentes *wrappers*, que se encarregam de inserir e controlar o meta componente na aplicação, e são orientados pelo MVC: componentes Web (*View*), componentes JavaBeans (*Controller*); e
- pelo redirecionamento da referência ao componente de negócio EJB (no MVC, o *Model*), feito pela edição de seu nome JNDI no *deployment descriptor* (arquivo XML).

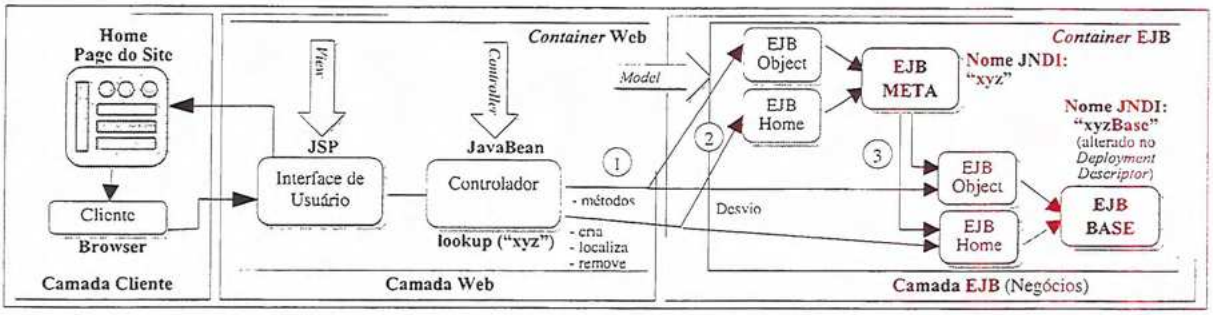

Fig. 8: Proposta para intercessão de funcionalidade na plataforma J2EE.

Quando a proposta de intercessão estiver estabilizada, será o momento de validá-la. Isso será feito através da implementação de um mecanismo de replicação passiva que atue no nível meta do modelo da Fig. 8 - como o descrito na seção 4 desse artigo -, e seja invocado por componentes *wrappers*, verificando portanto as proposições dessa abordagem.

Referências Bibliográficas

[1] 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.

[2] 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

[3] 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.

[4] 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.

[5] 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.

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

[7] 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

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

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

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

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

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

[13] 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

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

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