Ferramenta MVCase – Uma Ferramenta Integradora de Tecnologias para o Desenvolvimento de Componentes Distribuídos

Eduardo Santana de Almeida
Daniel Lucrédio
Calebe de Paula Bianchini
Antonio Francisco do Prado
Luis Carlos Trevelin
Universidade Federal de São Carlos – Departamento Computação
{ealmeida, lucredio, calebe, prado, trevelin}@dc.ufscar.br

Resumo

Este artigo apresenta a Ferramenta MVCase que integra diferentes tecnologias para orientar o engenheiro de software no Desenvolvimento de Componentes Distribuídos de um domínio do problema. As tecnologias envolvidas são: o método Catalysis, o padrão CORBA para distribuição dos componentes e frameworks de componentes.Catalysis é utilizado como método de Desenvolvimento Baseado em Componentes (DBC) para definir, especificar e projetar os componentes distribuídos, segundo a arquitetura CORBA. Frameworks de componentes são reutilizados para orientar a distribuição dos componentes desenvolvidos do domínio do problema e facilitar o acesso a banco de dados relacional.

1. Introdução

No desenvolvimento de software, a reutilização caracteriza-se pela utilização de produtos de software, em uma situação diferente daquela para qual estes produtos foram originalmente construídos [3]. O Desenvolvimento Baseado em Componentes (DBC) se preocupa com a criação de componentes que possam ser reutilizados em outras aplicações. Como solução para este problema, pesquisas [1], [3], [6] apontam, como passo fundamental, a sistematização do processo de análise e criação de componentes para um determinado domínio de aplicações.

Para que a reutilização possa ser efetiva, deve-se considerá-la em todas as fases do processo de desenvolvimento do software. Portanto, o Desenvolvimento Baseado em Componentes deve oferecer métodos, técnicas e ferramentas que suportem desde a identificação e especificação dos componentes, do domínio do problema, até o seu projeto e implementação em uma linguagem executável.

Apesar das recentes e constantes pesquisas na área de DBC, ainda há carência de métodos, técnicas e ferramentas que suportem tanto o desenvolvimento quanto a reutilização de componentes em aplicações de um determinado domínio. Embora existam diferentes tecnologias com objetivo de suportar o DBC, ainda têm-se muitas dificuldades de integrá-las para cobrir todo o processo de DBC, desde a construção dos componentes até sua reutilização pelas aplicações. Se considerarmos, ainda, o caso de componentes distribuídos, o problema é ainda maior.

Diferentes ferramentas têm sido propostas para suportar o DBC, dentre elas, destacamos as CASE [5], [11]. Entretanto, a maioria destas ferramentas, cobre parcialmente o DBC, principalmente no caso de aplicações distribuídas.

Motivado por estas idéias, este artigo apresenta a ferramenta *MVCase*, que suporta o Desenvolvimento de Componentes Distribuídos. Na sua primeira versão [9], a *MVCase* suporta a especificação dos requisitos de sistemas em alto nível de abstração, na linguagem de modelagem *Unified Modeling Language* (UML) [8]. Numa segunda versão [10], a ferramenta suporta a especificação e projeto dos componentes, baseado em *Catalysis* [6] e utiliza a tecnologia *Enterprise Java Beans* (EJB) [7], para distribuição. No estágio atual, a ferramenta *MVCase*, além de utilizar *Catalysis* como método de DBC para definir, especificar e projetar os componentes, disponibilizando

todos os seus artefatos, utiliza a arquitetura *CORBA* [4], para distribuição e *frameworks* de componentes para orientar a distribuição dos componentes desenvolvidos e facilitar o acesso a banco de dados relacional.

2. Ferramenta MVCase – Uma Ferramenta Integradora de Tecnologias para o Desenvolvimento de Componentes Distribuídos

Integrando o método *Catalysis* de desenvolvimento de software baseado em componentes, as idéias da arquitetura *CORBA* para especificação e distribuição dos componentes, os *frameworks* básicos que orientam a distribuição dos componentes e acesso a banco de dados relacional, na ferramenta *MVCase*, definiu-se uma estratégia [1], que suporta o Desenvolvimento de Componentes Distribuídos.

Os componentes do domínio do problema são construídos, em quatro passos: **Definir Problema**, **Especificar Componentes**, **Projetar Componentes** e **Implementar Componentes**. Os três primeiros passos correspondem aos três primeiros níveis de *Catalysis*. No último passo faz-se a implementação física dos componentes.

Segue-se uma apresentação mais detalhada de cada passo da estratégia, mostrando detalhes das técnicas empregadas, e os principais artefatos gerados em cada passo. Para facilitar o entendimento das entradas, saídas e controles, será utilizado um Domínio do Problema de Ordem de Serviços de uma oficina de reparos de equipamentos eletrônicos.

2.1 Definir Problema

Neste *primeiro passo* da estratégia, correspondente ao **primeiro nível** de *Catalysis*, é dada ênfase no entendimento do problema, especificando-se "o que" os componentes devem atender para solucionar o problema.

Inicialmente, são identificados os requisitos do domínio, usando técnicas como *storyboards* ou *mind-maps* [6], visando representar as diferentes situações e cenários do domínio do problema. Em seguida, os requisitos identificados são especificados em Modelos de Colaborações [6], representando a coleção de ações e os objetos participantes. Finalmente, os modelos de colaborações são refinados em Modelos de Casos de Uso [6, 8]. A Figura 1 resume este primeiro passo.

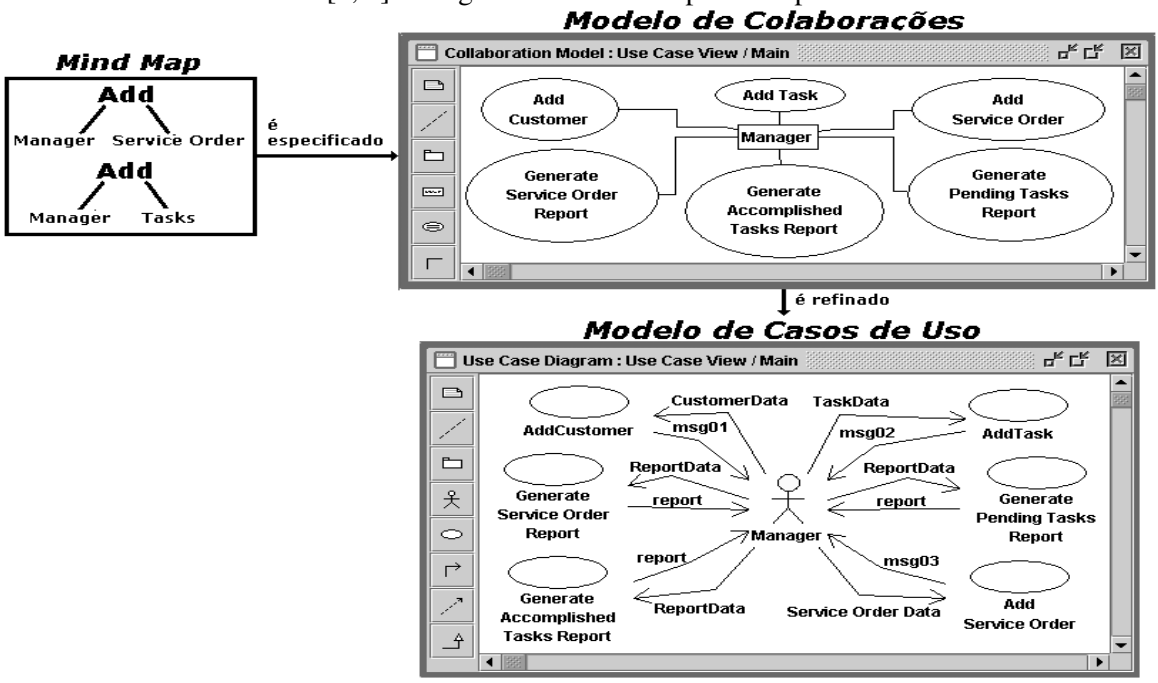

Figura 1 - Primeiro Passo da Estratégia: Definir Problema.

A Figura 1 mostra um *mind-map*, definido na identificação dos requisitos do domínio Ordem de Serviços, sendo especificado num Modelo de Colaborações, e, posteriormente, refinado e particionado em um Modelo de Casos de Uso, visando diminuir a complexidade e melhorar o entendimento do domínio do problema.

2.2 Especificar Componentes

Este passo corresponde ao **segundo nível** de *Catalysis*, onde é descrito o comportamento externo do sistema de uma forma não ambígua. Na ferramenta CASE, o engenheiro de software refina as especificações do nível anterior, visando obter as especificações dos componentes.

Esse passo tem início com o refinamento dos modelos do domínio do problema. Especifica-se o Modelo de Tipos [6], conforme mostra a Figura 2, mostrando atributos e operações dos tipos de objetos, sem se preocupar com a implementação. Ainda neste passo, pode-se utilizar o dicionário de dados para especificar cada tipo encontrado, e a *Object Constraint Language* (OCL) [8], para detalhar o comportamento dos objetos, sem ambigüidades.

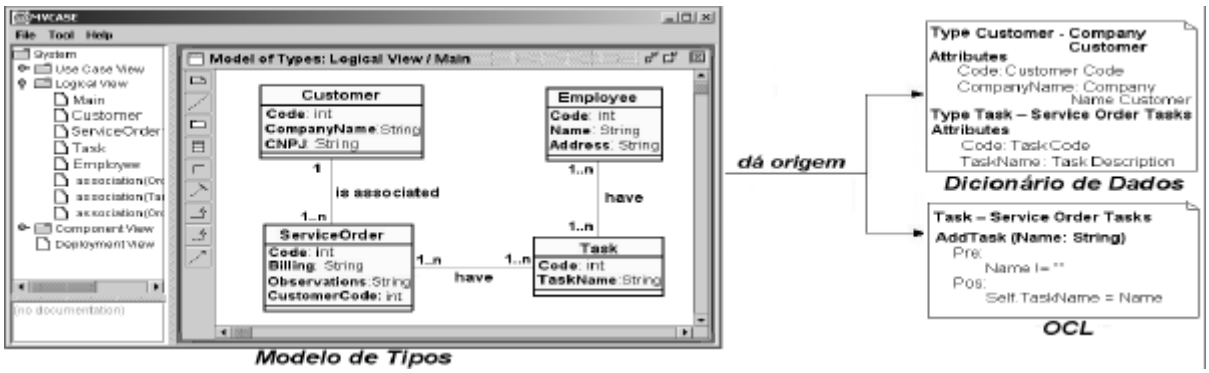

Figura 2 – Modelo de Tipos do passo Especificar Componentes.

Conforme mostra a Figura 3, o *Modelo de Tipos* é organizado em um *Framework de Modelos de Tipos* de componentes [6], com os seus atributos e relacionamentos. O *framework* é reutilizado através do *Modelo de Aplicação do Framework* [6], representando a dependência dos tipos do *framework*, com os tipos estendidos na aplicação. Os Modelos de Casos de Uso, do passo anterior, são refinados em Modelos de Interações representados pelos diagramas de seqüência [6, 8], para detalhar os cenários de utilização dos componentes nas diferentes aplicações do domínio do problema.

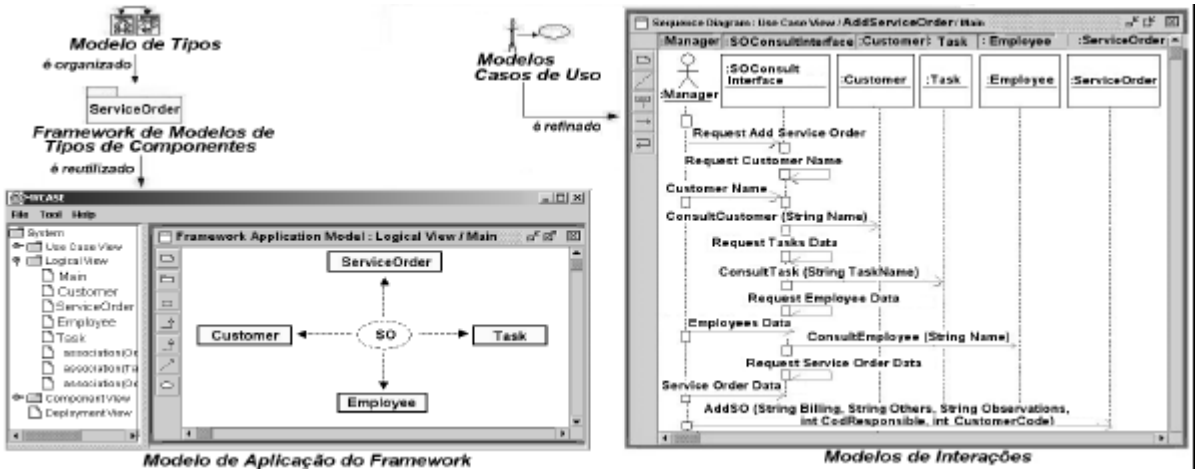

Figura 3 – Segundo Passo da Estratégia: Especificar Componentes.

Estes modelos são utilizados no próximo passo da estratégia, para obter o projeto interno dos componentes.

2.3 Projetar Componentes

Neste passo, o engenheiro de software faz o projeto interno dos componentes, conforme o **terceiro nível** de *Catalysis*. Como passo inicial, refinam-se os Modelos de Tipos em Modelos de Classes de componentes [6, 8], onde são modeladas as classes, e seus relacionamentos, preocupando-se com a definição dos componentes com suas interfaces. A Figura 4 mostra parte do Modelo de Classes do domínio OS e as especificações das interfaces em *Interface Definition Language* (IDL) [4].

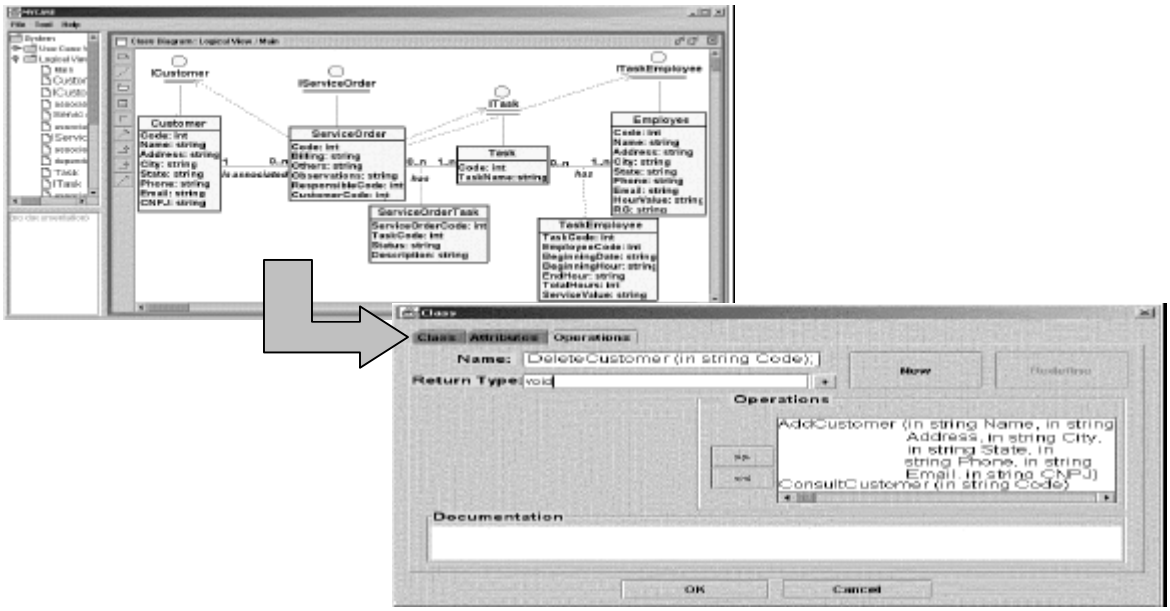

Figura 4 - Modelo de Classes obtido do Modelo de Tipos.

A partir do Modelo de Classes, pode-se construir o Modelo de Componentes [6, 8], que mostra a organização e as dependência entre os componentes. O Modelo de Componentes pode reutilizar componentes de outros *frameworks* já existentes. Assim, podem ser reutilizados componentes de domínios básicos relacionados com requisitos não funcionais, como por exemplo, para persistência em Banco de Dados [12] e distribuição de componentes [4]. A Figura 5 mostra o Modelo de Componentes obtido do Modelo de Classes da Figura 4, fazendo reutilização dos componentes dos *frameworks Persistence* e *Broker*, organizados nos pacotes *Persistence* e *Broker*, respectivamente.

2.4 Implementar Componentes

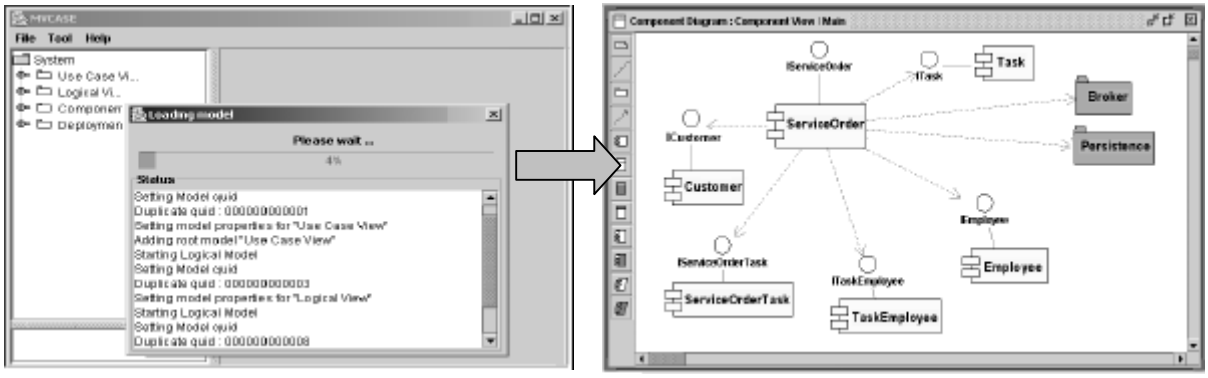

Figura 5 – Modelo de Componentes com reutilização de *frameworks*.

Neste passo, o engenheiro de software utiliza um gerador de código, da *MVCase*, para implementar os componentes projetados. Para que o componente possa ser distribuído, é gerado o código segundo o padrão *CORBA*. Para cada componente, têm-se os *stubs* e *skeletons* e as interfaces

que disponibilizam seus serviços. A Figura 6 mostra, à esquerda, o processo de geração de código dos componentes projetados, e, à direita, os códigos em *IDL* e *Java* gerados pela *MVCase*.

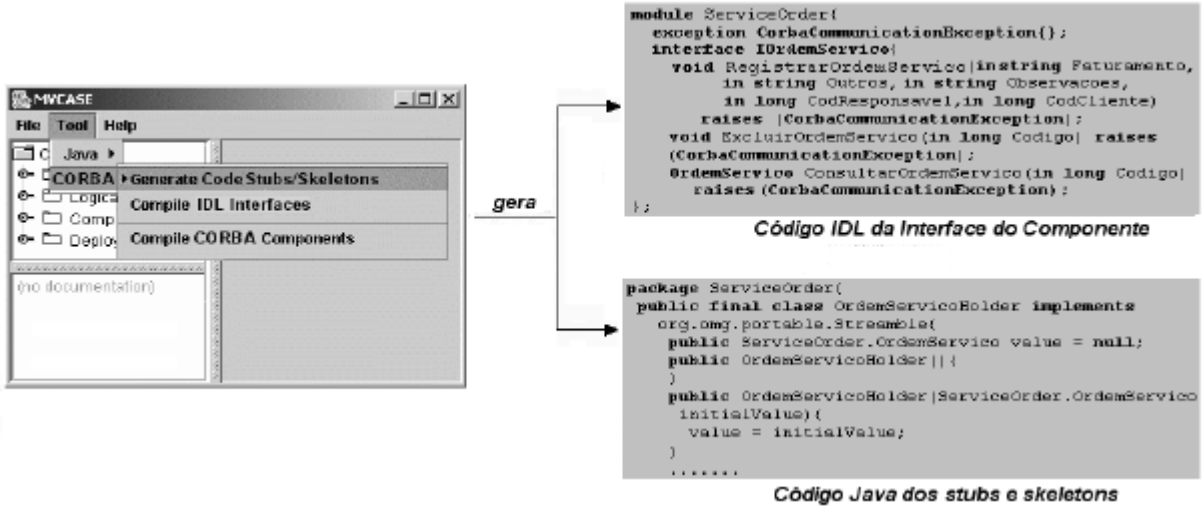

Figura 6 – Quarto passo da Estratégia: Implementar Componentes.

Uma vez construídos os componentes do domínio do problema, o engenheiro de software utiliza o editor disponível na *MVCase* para realizar o desenvolvimento das aplicações reutilizando os componentes desenvolvidos, conforme mostra a Figura 7.

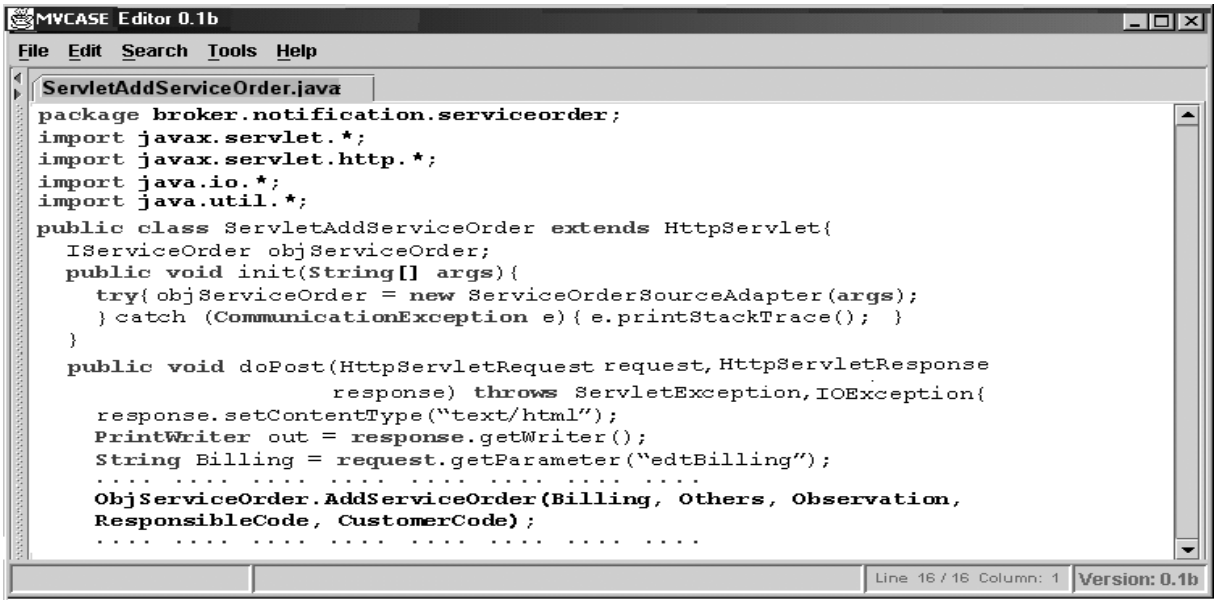

Figura 7 – Desenvolvimento de Aplicações usando a Ferramenta *MVCase*.

A Figura 8 mostra o ambiente de execução da aplicação, que registra uma ordem de serviço segundo o modelo de distribuição *Object Web* [4], utilizando *CORBA*. Na primeira camada, tem-se o cliente, que executa a aplicação. Através da comunicação via *HyperText Transfer Protocol* (HTTP), são enviadas as requisições e recebidas as respostas do *ServletAddServiceOrder*, que está disponível no Servidor Web. Na segunda camada, o servidor web comunica-se com o Servidor de Aplicação, que disponibiliza os componentes do domínio do problema. Esta comunicação é feita via *Object Request Broker* (ORB) [4]. Na terceira camada está o banco de dados com seu servidor. A comunicação do Servidor de Aplicação com o Servidor de Banco de Dados, também via *ORB*, permite o acesso aos serviços de banco de dados. Nesta arquitetura, pode-se ter os componentes e os servidores distribuídos em diferentes computadores da rede.

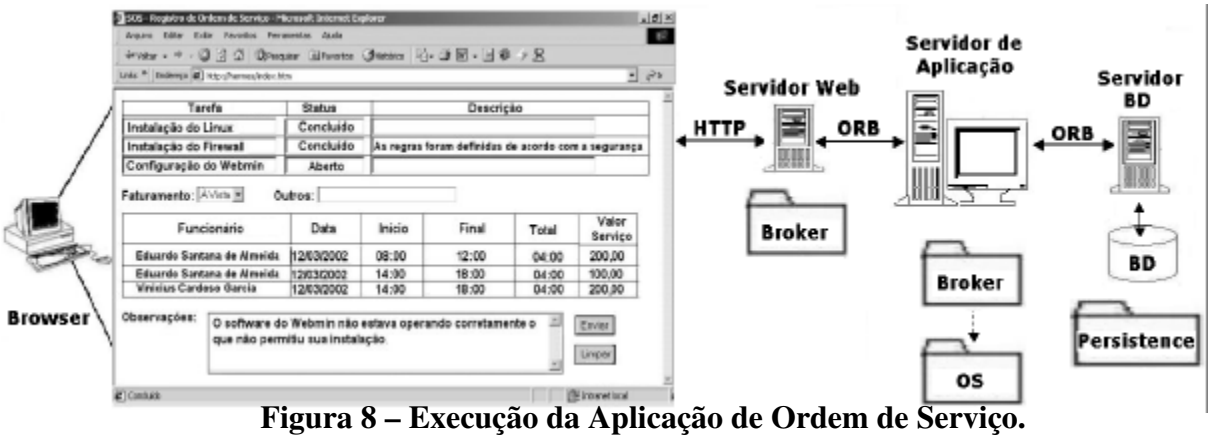

Figura 8 – Execução da Aplicação de Ordem de Serviço.

3. Conclusão e Trabalhos Futuros

Este artigo apresentou a Ferramenta *MVCase*, que consegue integrar diferentes tecnologias, orientando o Engenheiro de Software na construção dos componentes distribuídos de um domínio do problema, partindo de um alto nível de abstração até a implementação.

Estes componentes podem ser distribuídos e reutilizados, diminuindo a redundância de código e facilitando o desenvolvimento das aplicações. Outro ponto importante é a manutenção das aplicações, que fica mais simples e confiável, uma vez que os componentes reutilizados foram previamente testados.

Atualmente, a ferramenta encontra-se numa nova fase de desenvolvimento, incorporando o *Serviço de Trading* [4], para permitir a consulta dos componentes disponíveis num repositório e definindo *stereotypes* [8] para permitir a definição e geração de código utilizando *AspectJ* [2].

4. Referências

[1] Almeida, E, S., Bianchini, C, P., Prado, A, F., Trevelin, L, C. **Distributed Component-Based Software Development Strategy.** *The 12th Workshop for PhD Students In Object Oriented Systems (PhDOOS) In Conjunction With The 16th European Conference On Object Oriented Programming (ECOOP*). Lecture Notes in Computer Science (LNCS) Springer-Verlag. Málaga, Espanha. 2002.

[2] AspectJ. **AspectJ – Aspect Oriented Programming (AOP) for Java**. Disponível site AspectJ. URL: http://www.aspectj.org. Consultado em 22 de junho de 2002.

[3] Braga, R. **Busca e Recuperação de Componentes em Ambientes de Reutilização de Software**. Dissertação de Doutorado. UFRJ dezembro de 2000.

[4] Corba. **The Common Object Request Broker: Architecture and Specification**. Disponível: site OMG (1996). URL: http://www.omg.org. Consultado em 10/04/2001.

[5] Cool:Gen. **Ferramenta Cool:Gen**. Disponível: site Computer Associates. URL: http://cai.com/acq/sterling. Consultado em 10/07/2001.

[6] D'Souza, D., Wills, A. **Objects, Components and Frameworks with UML – The Catalysis Approach**. USA:Addison Wesley, 1998.

[7] Horstmann, S, C. **Core Java2 Volume2-Advanced Features**. Sun Microsystems Hal 1999.

[8] Larman, C. **Utilizando UML e Padrões**. Prentice Hall , Inc, 1999.

[9] Lucrédio, D., Prado, A, F. **MVCase - Ferramenta CASE Orientada a Objetos**, *XIV Simpósio Brasileiro de Engenharia de Software*, *Sessão de Ferramentas*. Outubro de 2000.

[10] Lucrédio, D., Prado, A, F. **Ferramenta MVCASE – Estágio Atual: Especificação, Projeto e Construção de Componentes**, *XV Simpósio Brasileiro de Engenharia de Software*, *Sessão de Ferramentas*. Outubro de 2001.

[11] Rose. **Ferramenta Rational Rose**. Disponível: site Rational the software development company. URL: http://www.rational.com. Consultado em 10/07/2001.

[12] Yoder, J.W. et al. **Connecting Business Objects to Relational Databases**. *In: Conference on the Pattern Languages of Programs*, 5, Monticello-IL, EUA. Proceedings. 1998.