# Desenvolvimento de Sistemas de Informação com Suporte à Composição Dinâmica

**Mário Hozano**[1], **Hyggo Almeida**[2] *, **Luiz Tenório**[1], **Evandro Costa**[1], **Angelo Perkusich**[2]

[1] Departamento de Tecnologia da Informação, Universidade Federal de Alagoas
Campus A. C. Simões, BR 104 - Norte, Km 97.
Tabuleiro dos Martins – Maceió/AL. CEP: 57072-970

[2]Departamento de Engenharia Elétrica, Universidade Federal de Campina Grande
Av. Aprigio Veloso, 882 - Bodocongó, 58109-970 - Campina Grande - PB - Brasil

{mhls,evandro}@tci.ufal.br,{hyggo,perkusic}@dee.ufcg.edu.br

***Resumo.*** *Nos últimos anos a Web tem se firmado como principal meio na divulgação de informações e no oferecimento de serviços através de inúmeros sistemas de informação. Em alguns cenários, tais serviços devem estar disponíveis em tempo integral, requerendo uma gerência dinâmica da manutenção e evolução dos sistemas de informação. Neste artigo propõe-se uma abordagem para o desenvolvimento de sistemas de informação com suporte à composição dinâmica. Tal abordagem utiliza o modelo de componentes COMPOR integrado a uma biblioteca de* tags*, garantindo a manutenção e evolução de sistemas de informação em tempo de execução.*

***Abstract.*** *In the last years the Web has become the main mechanism to deliver information and services through many information systems. In some scenarios, these services must be available fulltime, requiring a dynamic management of the evolution and maintenance of information systems. This paper proposes an approach for developing information systems that support dynamic composition. Such approach is based on the integration of the COMPOR component model and a tag library, providing mechanisms for runtime maintenance and evolution of information systems.*

## 1. Introdução

A popularização da Internet nos últimos anos aliada à facilidade na criação e no acesso às páginas Web vem proporcionando um crescimento gradativo na construção de aplicações corporativas na rede. Ao oferecer uma grande variedade de serviços, a Web vem se tornando cada vez mais forte na medida em que simples *Web sites*, com conteúdo estático, dão lugar a complexas aplicações, com conteúdo dinâmico.

Com o uso de diversas tecnologias e abordagens a evolução destas aplicações exige flexibilização na arquitetura do software, a fim de diminuir o impacto das inevitáveis mudanças nos seus requisitos. Em alguns cenários tal evolução deve ser gerenciada dinamicamente. Em sistemas críticos de empresas que negociam seus produtos através da Web, por exemplo, a composição dinâmica é primordial. Um bom exemplo pode ser observado com um grande site de leilões que, por problemas de infra-estrutura, ficou com seu sistema 22 horas "fora do ar", o que representou um prejuízo de cerca de US$ 3 milhões, segundo [Taurion 2002].

*Bolsista CNPq no Programa de Doutorado do Departamento de Engenharia Elétrica - COPELE/DEE

Em sistemas de informação dispostos na Web a situação não é diferente. Na maioria dos casos, a consulta à qualquer tipo de informação é impossibilitada em períodos de manutenção. Os usuários (e outras aplicações) que necessitam de tais informações não são previamente informados destas operações rotineiras do sistema, bem como da previsão de retorno de seu pleno funcionamento.

Uma possível solução para este problema é definir uma infra-estrutura que permita modificar o software, sua arquitetura ou funcionamento, sem interromper a execução da aplicação, com mínimo impacto sobre os componentes que estão em operação. Neste contexto, a especificação do modelo de componentes COMPOR (*Component Model Specification* - CMS) [Almeida et al. 2004a] provê mecanismos para a composição dinâmica em aplicações, permitindo que componentes da aplicação possam ser removidos, alterados e novos componentes possam ser adicionados em tempo de execução.

Neste artigo propõe-se uma integração entre a CMS e uma biblioteca de *tags* baseada na tecnologia *JavaServer Pages* (JSP) [JSP 2005], permitindo o desenvolvimento de sistemas de informação com suporte à composição dinâmica. Invocando os serviços dos componentes da CMS à partir de uma interface gráfica Web, torna-se possível adicionar e modificar serviços e funcionalidades sem a interrupção do sistema. Para ilustrar a relevância do funcionamento desta abordagem, uma aplicação no contexto de comunidades virtuais será apresentada como estudo de caso exemplificando seu funcionamento.

Na Seção 2., descreve-se a especificação de componentes COMPOR; na Seção 3., é apresentada a estrutura e o funcionamento da integração proposta para composição dinâmica em sistemas de informação; o funcionamento desta integração é descrito como um estudo de caso na Seção 4.; trabalhos relacionados são discutidos na Seção 5.; e, na Seção 6., são apresentadas as considerações finais.

## 2. A especificação de componentes COMPOR (CMS)

As principais entidades definidas na CMS são **contêineres** e **componentes funcionais**. Os componentes funcionais implementam as funcionalidades do sistema, disponibilizando-as na forma de serviços. Os componentes funcionais não se decompõem, ou seja, não possuem "componentes-filhos". Os contêineres, por sua vez, não implementam funcionalidades, apenas gerenciam o acesso aos serviços dos seus "componentes-filhos". Ou seja, os contêineres servem como "portas de acesso" às funcionalidades dos componentes neles contidos [Almeida et al. 2004a].

### 2.1. Modelo de interação baseado em serviços

Os componentes funcionais são disponibilizados através da inserção nos contêineres. Nesse caso, é necessário que haja ao menos um contêiner (contêiner-raiz) para que seja possível adicionar os componentes funcionais. Cada contêiner possui uma lista dos serviços providos e eventos de interesse de cada um dos seus "componentes-filhos". Após a inserção de um componente, a lista de serviços e eventos de cada contêiner até a raiz da hierarquia é atualizada. Isto é necessário para que os serviços providos pelo componente recém-adicionado estejam disponíveis para qualquer outro componente funcional do sistema. Além disso, permite-se que o componente adicionado seja notificado, caso algum evento de seu interesse seja disparado por outro componente. Após esse processo, os serviços de um componente "X" podem ser acessados a partir de qualquer componente da hierarquia, sem referenciar "X" explicitamente.

Sendo assim, supondo a existência do serviço "salvar" implementado pelo componente K, pode-se solicitar a execução deste serviço a partir de um componente X, sem fazer referência a K. Este processo é apresentado na Figura 1, onde pode-se verificar que

não há referência alguma entre o componente solicitante do serviço (“X”) e o componente provedor do mesmo (“K”). Desta forma, é possível alterar o componente que provê o serviço “salvar” sem modificar o restante da estrutura.

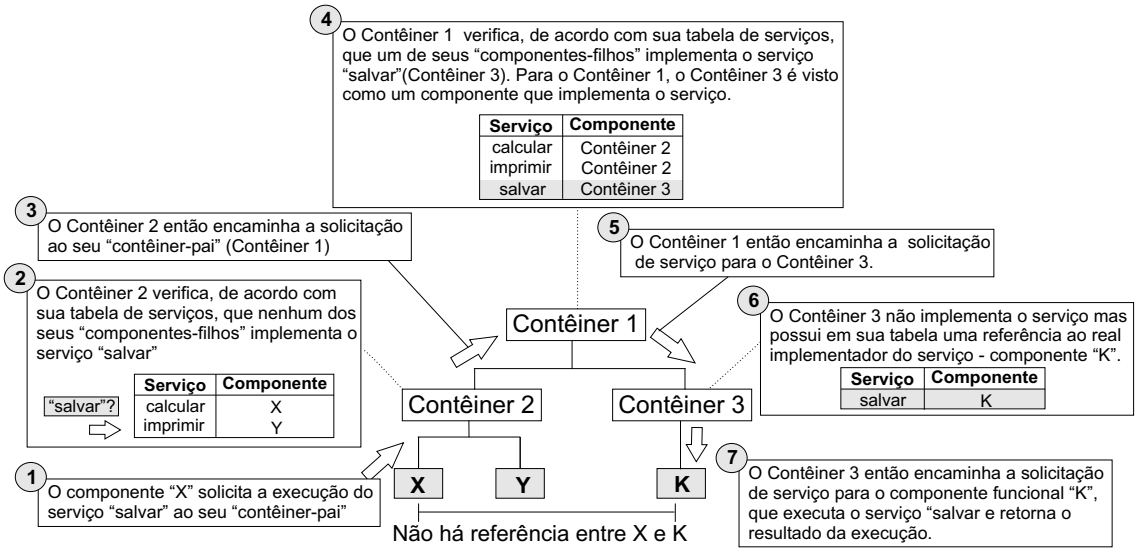


**Figura 1. Interação baseada em serviços: localização e execução sem referência entre componentes funcionais [Almeida et al. 2004a]**

A única entidade do modelo para a qual um componente funcional possui referência é o seu “contêiner-pai”. No caso de dois componentes possuírem serviços com mesmo nome, o conceito de *alias* é utilizado, sendo possível registrar um *apelido* para cada serviço. A especificação completa de interação baseada em serviços e eventos é apresentada em [Almeida et al. 2004a].

## 2.2. Implementação da CMS: o arcabouço JCF

O arcabouço JCF (*Java Component Framework*) implementa a especificação do modelo de componentes COMPOR (CMS). JCF tem seu projeto baseado no padrão *Composite* [Gamma et al. 1995]. Este padrão de projeto é aplicável a arquiteturas hierárquicas nas quais as entidades “compostas” contém tanto outras entidades “compostas” quanto “entidades-folha”. Através da utilização de uma interface única de acesso a estas entidades, garante-se uma composição recursiva.

O JCF é composto por duas classes principais, *FunctionalComponent* e *Container*, que são instanciadas, respectivamente, em componentes funcionais e contêineres, os quais são especificados na CMS. Uma classe abstrata, *AbstractComponent*, garante a composição recursiva, permitindo que os contêineres “desconheçam” a implementação dos seus “componentes-filhos”, que podem ser tanto componentes funcionais quanto outros contêineres.

A interação baseada em serviços é implementada através de invocações sucessivas do método *doIt*, “de baixo pra cima” na hierarquia de componentes até que o contêiner correto seja encontrado. Quando isto acontece, o método *receiveRequest* é invocado “de cima pra baixo” na hierarquia até que o componente funcional que implementa o serviço seja encontrado. Os parâmetros de *doIt* e *receiveRequest* são baseados em *String* para especificar o nome ou apelido do serviço. Por exemplo:

```
doIt(new ServiceRequest("serviceName", Object["param1","SBSI"]))
```

## 3. Composição dinâmica em Sistemas de Informação baseados Web

Para o desenvolvimento de sistemas de informação baseados na Web, propõe-se um arcabouço que utiliza a infra-estrutura de composição dinâmica provida pelo JCF, acoplando módulos de suporte à construção de aplicações Web. Este suporte é implementado como uma biblioteca de *tags* pré-definida, utilizando um recurso da tecnologia JSP. Através de instruções definidas diretamente na interface de apresentação baseada na web, é possível definir que os componentes visuais da interface gráfica do usuário utilizem o modelo da aplicação desenvolvida sobre o JCF, constituindo um arcabouço para composição dinâmica em sistemas de informação baseados na Web.

### 3.1. Arquitetura do arcabouço

A arquitetura do arcabouço é composta por três partes distintas: um processador de *tags*, um formatador de resultados e o modelo da aplicação implementada usando o JCF, como ilustrado na Figura 2. Os elementos da arquitetura são descritos a seguir.

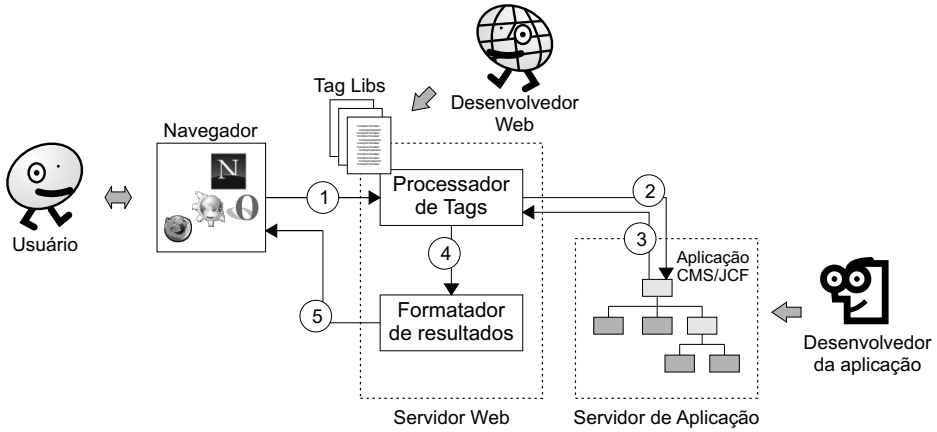


**Figura 2. Arquitetura do arcabouço**

- *Processador de tags:* responsável por verificar as *tags* descritas pelo *Desenvolvedor Web* na página requisitada ao navegador (*browser*), validá-las e, posteriormente, fazer as invocações aos serviços e eventos à aplicação JCF, fazendo a verificação de resultados e exceções. O processador de *tags* representa o ponto de integração entre a arquitetura Web e o JCF. Uma vez que as invocações de serviços são feitas ao contêiner raiz da hierarquia de componentes, todos os serviços da aplicação estão disponíveis. A execução de serviços é realizada através da invocação do método *doIt*, tendo como nome de serviço e parâmetros os valores definidos nas *tags*.

- *Aplicação JCF/CMS:* aplicação baseada em componentes de acordo com a CMS, implementada pelo JCF. Como apresentado anteriormente, a CMS permite a inserção, alteração e remoção de componentes em tempo de execução. Esta característica de composição dinâmica é refletida no sistema Web como um todo - uma vez que é possível adicionar novos componentes e os seus serviços são acessados através dos identificadores deste serviço, torna-se possível evoluir dinamicamente a aplicação. A aplicação JCF/CMS, realizada pelo *Desenvolvedor da aplicação*, implementa as funcionalidades e regras de negócio do sistema sendo desenvolvido.

- *Formatador de resultados:* modela os resultados obtidos na invocação de serviços e eventos à aplicação JCF/CMS, de acordo com um padrão definido pelo *Desenvolvedor Web*, flexibilizando a forma de apresentação. Este módulo é primordial para a apresentação de resultados com tipos de dados mais complexos, tais como entidades do modelo de negócio da aplicação. A formatação padrão é baseada na versão *String* do resultado a ser exibido, utilizando o método padrão *toString()*.

### 3.2. Biblioteca de *tags*

Para permitir a independência entre o módulo Web e o módulo da aplicação JCF, foi desenvolvido um conjunto de *tags* JSP. A tecnologia JSP possui um mecanismo para a definição de extensões da linguagem, possibilitando a criação de novas *tags*, gerando conteúdo dinâmico de acordo com a aplicação sendo desenvolvida.

Na Figura 3, ilustra-se o uso da *tag* `compor:invokeService` para invocação do serviço `findDocByTitle`. O resultado desta invocação será armazenado em uma variável chamada `docsJava`. A *tag* `compor:param` é utilizada para a passagem de parâmetro para este serviço com valor e tipo indicados.

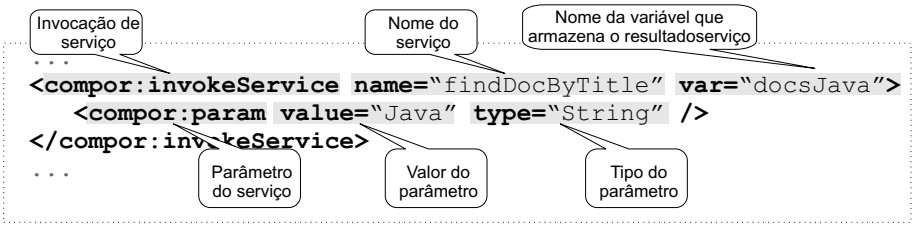


**Figura 3. Invocação do serviço `findDocByTitle` com passagem de parâmetros**

A formatação, validação e conversão dos resultados obtidos com estas interações também podem ser feitas através de *tags* da mesma biblioteca. Na Figura 4, apresenta-se uma das formas como o resultado do serviço anteriormente invocado pode ser exibido na página JSP, através da *tag* `compor:result`. Neste exemplo, foi utilizado o formatador padrão de resultados (*toString*). Existem ainda *tags* que proporcionam verificação e tratamento de exceções sobre execuções de serviços.

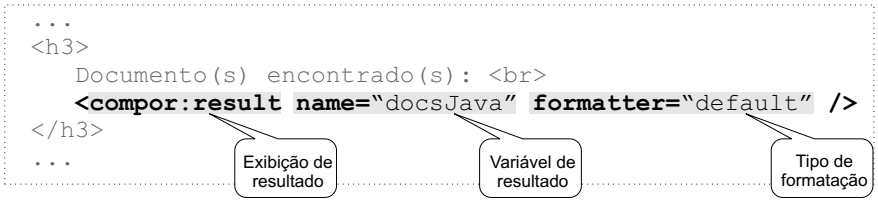


**Figura 4. Exibição do resultado obtido na invocação do serviço `findDocByTitle`**

### 3.3. Descrição funcional

O funcionamento do arcabouço para composição dinâmica em sistemas de informação baseados na Web é promovido com a comunicação entre a interface gráfica do sistema e o modelo da aplicação implementado com o *Java Component Framework*. Através desta comunicação, é possível executar serviços e eventos do modelo de maneira simples e transparente, utilizando as *tags* da biblioteca descritas anteriormente.

Na Figura 2 ilustra-se o cenário de funcionamento do arcabouço a partir da exibição de uma página JSP contendo chamadas às *tags* da biblioteca. Os passos relacionados ao cenário são descritos a seguir, considerando uma requisição realizada pelo usuário a uma página Web:

1. As *tags* da página JSP são processadas pelo *processador de tags* que extrai o nome dos serviços a serem solicitados à aplicação JCF;
2. O *processador de tags* solicita a execução de cada serviço ao contêiner raiz da aplicação JCF;
3. Após a execução do serviço na aplicação JCF, o processador de *tags* obtém os resultados. Os resultados são recebidos na forma de objetos do domínio e devem ser formatados para exibição;
4. Sendo assim, o processador de *tags* repassa os resultados para o *formatador de resultados* que os prepara para a exibição na página JSP;
5. O cenário de funcionamento do arcabouço termina com os dados já formatados enviados novamente para o navegador.

Através da estrutura do arcabouço descrita na Seção 3.1. e de seu funcionamento é possível alterar o modelo da aplicação de maneira transparente e dinâmica sem interromper o sistema, devido às características observadas no modelo de componentes implementado com o JCF.

## 4. Estudo de caso

Para ilustrar o funcionamento do arcabouço descrito neste artigo, será apresentada uma aplicação no contexto de sistemas de informação baseados em comunidades virtuais.

O arcabouço de comunidades virtuais (`ArCo`) [Almeida et al. 2004b], utilizado sobre uma licença de software livre, encapsulam os serviços de comunidades virtuais conhecidos oferecendo ferramentas de bate-papo, fórum, enquete, e-mail, agenda, biblioteca digital etc., permitindo uma maior interação entre os usuários do sistema.

Para permitir um maior controle no fluxo de informações e permissões aos serviços do sistema, o `ArCo` também conta com um componente de segurança que provê autenticação e autorização nas relações da aplicação com o usuário.

Com este componente implementado utilizando o JCF, a autenticação é realizada uma única vez através de uma tela de acesso onde o usuário informa nome e senha. Submetidas as informações necessárias através do botão “Enviar” o serviço de autenticação implementado no componente de segurança é invocado. Caso as informações sejam compatíveis com a base de dados do sistema, este serviço retornará a resposta de sucesso permitindo ao usuário o acesso as diversas ferramentas e comunidades do `ArCo`.

Com o surgimento de novas tecnologias no contexto de segurança (autenticação) para aplicações Web, a mudança do componente de segurança que desempenhe o mesmo papel pode ser possível, à medida que se observe uma abordagem mais confiável no mesmo. Para realizar esta modificação na estrutura do sistema, não será necessário interromper a execução do `ArCo`. Esta modificação pode ser feita trocando apenas o componente funcional responsável por oferecer o serviço, por outro que desempenhe a mesma funcionalidade. Assim, a arquitetura da aplicação é modificada dinamicamente sem a interrupção do funcionamento do sistema durante a manutenção.

Uma vez que não há ligação direta entre o módulo Web, representado pelas *tags*, e os componentes que implementam os serviços da aplicação construída usando o JCF, torna-se possível alterar os componentes do sistema em tempo de execução, assim como, redefinir páginas apenas alterando suas *tags* de invocação aos serviços.

## 5. Trabalhos relacionados

Algumas tecnologias conhecidas relacionadas à plataforma J2EE, tais como *Servlet* [Ser 2005] e a própria JSP, não têm como foco permitir a alteração da aplicação em tempo de execução. Apesar da possibilidade de implementar composição dinâmica sobre J2EE, utilizando mediadores, por exemplo, a ausência de uma infra-estrutura para composição dinâmica torna mais difícil o desenvolvimento considerando esse requisito. Por outro lado, aplicações desenvolvidas com *JavaBeans* [Microsystems 1997], *Enterprise JavaBeans* [Roman et al. 2001] e *CORBA Component Model* [OMG 2003], ainda que sejam baseadas em componentes, não possuem uma infra-estrutura de desenvolvimento para acesso através da Web.

Dentro do contexto de composição dinâmica, dois trabalhos correlatos são HADAS [Ben-Shaul et al. 2001] e a abordagem proposta em [Kim and Bae 2001], que é uma evolução do HADAS. HADAS é baseado em modelos distribuídos de objetos e tem foco na interoperabilidade entre componentes distribuídos. Contudo, nestas abordagens também não há suporte para o desenvolvimento de aplicações Web.

Em relação a infra-estruturas que facilitam modificações na estrutura de sistemas de informação baseados na Web, algumas abordagens podem ser observadas. Em [Gaedke and Graf 2001] utiliza-se a WCML *Web Composition Markup Language*) [Gaedke et al. 2000] como linguagem intermediária no desenvolvimento das páginas do sistema, que são geradas através de um compilador próprio. Já em [Hansen 2002] e [Dijkstra 2004], a estrutura e o gerenciamento do sistema em módulos separados, permite uma engenharia de software que facilita o entendimento do sistema facilitando a mudança de requisitos.

Apesar de facilitar o desenvolvimento e o entendimento da estrutura dos sistemas de informação baseados na Web, estas abordagens não permitem que os serviços do sistema sejam modificados dinamicamente, sendo necessária a interrupção da aplicação durante a manutenção.

## 6. Considerações finais

Neste artigo apresentou-se uma abordagem para o desenvolvimento de sistemas de informação com suporte à composição dinâmica baseado em uma biblioteca de *tags* JSP integrada ao modelo de componentes COMPOR.

Através desta integração torna-se possível alterar, remover e/ou inserir funcionalidades em sistemas de informação baseados na Web sem a necessidade de interromper o seu funcionamento.

Para trabalhos futuros, espera-se definir um conjunto de *tags* para o desenvolvimento de software usando *Java Server Faces* [JSF 2005], que provê componentes de mais alto nível para a construção de aplicações Web. Atualmente, o arcabouço está sendo utilizado no desenvolvimento de um sistema de informação para gerência de documentos científicos.

## Referências

(2005). Java Servlet Technology. http://java.sun.com/products/servlet/. Acessado em 01/07/2005.

(2005). JavaServer Faces Technology. http://java.sun.com/j2ee/javaserverfaces/. Acessado em 01/07/2005.

(2005). JavaServer Pages Technology. http://java.sun.com/products/jsp/. Acessado em 01/07/2005.

Almeida, H. O., Perkusich, A., Paes, R. B., and Costa, E. B. (2004a). Composição Dinâmica de Componentes para Aplicações com Mudanças Freqüentes de Requisitos. In *Anais do 4o Workshop de Desenvolvimento Baseado em Componentes*, pages 9–14, João Pessoa, Paraíba, Brasil.

Almeida, H. O., Tenorio, L. E. F., Costa, E. B., Barbosa, N. M., Bublitz, F. M., and Barbosa, A. A. (2004b). Um Arcabouço de Software Livre baseado em Componentes para a Construção de Ambientes de Comunidades Virtuais de Aprendizagem na Web. In *Anais do XV Simpósio Brasileiro de Informática na Educação*, pages 188–196, Manaus, Amazonas, Brasil.

Ben-Shaul, I., Holder, O., and Lavva, B. (2001). Dynamic Adaptation and Deployment of Distributed Components In Hadas. *IEEE Trans. Softw. Eng.*, 27(9):769–787.

Dijkstra, M. (2004). A model for the management of dynamic web applications. In *ICEC '04: Proceedings of the 6th international conference on Electronic commerce*, pages 402–408, New York, NY, USA. ACM Press.

Gaedke, M. and Graf, G. (2001). Development and evolution of web-applications using the webcomposition process model. In *Web Engineering, Software Engineering and Web Application Development*, pages 58–76, London, UK. Springer-Verlag.

Gaedke, M., Segor, C., and Gellersen, H.-W. (2000). WCML: paving the way for reuse in object-oriented Web engineering. In *SAC '00: Proceedings of the 2000 ACM symposium on Applied computing*, pages 748–755, New York, NY, USA. ACM Press.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley.

Hansen, S. (2002). Web information systems: the changing landscape of management models and web applications. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 747–753, New York, NY, USA. ACM Press.

Kim, I. and Bae, D. (2001). A Dynamic Composition Model for Addressing Constrained Environments. In *OOPSLA Workshop on Reuse in Constrained Environments*.

Microsystems, S. (1997). Java Beans Specification. http://java.sun.com/products/javabeans/docs/spec.html. Acessado em 01/07/2005.

OMG (2003). CORBA Component Model Specification. http://ditec.um.es/∼dsevilla/ccm/. Acessado em 01/07/2005.

Roman, E., Ambler, S. W., and Jewell, T. (2001). *Mastering Enterprise JavaBeans*. John Wiley and Sons.

Taurion, C. (2002). Revista TI - 24 Horas no Ar. http://www.timaster.com.br/revista/artigos/main_artigo.asp?codigo=530.