TECNOLOGIAS DE ORGANIZAÇÃO E IMPLEMENTAÇÃO DE SISTEMAS DE INFORMAÇÃO NA WEB. *

Resumo

A preocupação com o desenvolvimento de sistemas é constante, e com o surgimento da Internet acentuou-se a concorrência entre as Instituições para disponibilizar seus produtos e serviços. Porém, a maioria das aplicações apresentam características indesejáveis que resultam em alguns problemas de manutenibilidade do sistema, a organização, descrição e recuperação de informação e a formas de evolução. Algumas tecnologias e técnicas são apresentadas por meio do desenvolvimento de um estudo de caso para minimizar ou solucionar determinados problemas encontrados nas aplicações da *Web*. Através desse estudo pode-se averiguar a viabilidade de utilização dessas tecnologias de organização e implementação de sistemas de informação para Web. Considerando que as Bibliotecas Digitais freqüentemente encontram-se na *Web*, estamos iniciando o desenvolvimento de uma biblioteca digital com o uso dessas tecnologias, objetivando a avaliação da viabilidade e da adequabilidade dessas tecnologias para esse tipo de sistema. Como o estudo de caso apresentado, de certa maneira, se assemelha as Bibliotecas Digitais podemos neste momento inferir que os mesmos benefícios obtidos nesse estudo também podem ser obtidos no desenvolvimento desse tipo de Unidade de Informação. Porém, a conclusão dessa hipótese só pode ser concretizada após o desenvolvimento completo e do estudo das formas de utilização dessa Biblioteca Digital.

Palavras chaves: Padrões de Projeto – Aspectos; *Servlets; Cookies;* Metadados; Biblioteca Digital

Abstract

The concern with the system developments is constant, and the appearing of the internet increases the concurrency between the institutions to make available its products and services. However, the most of applications show undesirable characteristics that results in some problems such as maintenability, organization, description and information recovery and its evolutions forms. Some technologies and techniques are showed through case study development for to decrease/solve some problems found in Web-based applications. Through this case study one can to check the using, organization and implementation techniques feasibility for information Web-based information systems. Take into consideration that Digital Libraries often found in the Web, we are starting a Digital Library Development with this technologies, aim the feasible and suitable evaluation of this technologies for this system types. As the case study showed assimilate as Digital Libraries, we are deduce that the same objectives obtained in this study can be obtained in the development of this type of Information Unit too. Although, the conclusion of this hypothesis only can be make after the complete development and the study of utilization forms this Digital Library.

Key-words: (Padrão de Projetos – Aspectos) Servlets; Cookies; Metadata; Digital Library

* ARAÚJO, L.S.(Mestranda em Ciência da Informação, Universidade Estadual Paulista (UNESP), Marília, SP, Brasil. E-mail: liriane@marilia.unesp.br)

VIDOTTI, S.A.B.G. (Docente do Curso de Pós-Graduação em Ciência da Informação e do Depto. De Ciência da Informação, UNESP, Marília, SP, Brasil.E-mail: vidotti@marilia.unesp.br)

1 Introdução

Esse trabalho, fruto de um estudo teórico- descritivo e exploratório visa analisar tecnologias/técnicas de informática que podem ser utilizadas no desenvolvimento de aplicações para a Web, em especial aplicações relacionadas as Bibliotecas Digitais.

A revisão de literatura apontou que alguns problemas encontrados em aplicações da World Wide Web (WWW ou Web) dizem respeito a manutenibilidade do sistema, a organização, descrição e recuperação de informação, e as formas de evolução desse.

Dentre as tecnologias disponíveis para a organização, descrição e recuperação da informação, estamos estudando os padrões de projeto (*design patterns)*, que têm auxiliado os projetistas no desenvolvimento de sistemas, e os metadados que são usados para facilitar os processos de captação, identificação, descrição e transmissão de informação.

Dentre as tecnologias de informática disponíveis para a implementação estamos enfocando as bibliotecas *servlets* - presentes na linguagem de programação Java, a programação orientada a aspectos e os *cookies*, que armazenam informações específicas do usuário.

O objetivo deste trabalho é apontar os benefícios que essas tecnologias fornecem para o desenvolvimento de sistemas de informações da Web, em especial de Bibliotecas Digitais – foco da nossa pesquisa. Segundo Camilo (2001):

> A proliferação de bibliotecas digitais causa uma transformação da forma tradicional de disponibilização de informações para um novo cenário que melhora a qualidade dos serviços prestados, e isso exige tanto uma nova postura dos profissionais, onde serão exigidos conhecimentos e habilidades no que diz respeito aos recursos eletrônicos, quanto o conhecimento de *softwares* específicos.

2-Tecnologias para implementação de aplicações da Web

A seguir são apresentadas algumas tecnologias de organização e implementação de sistemas para Web.

2.1-Padrões de Projeto

Gammma et al.(1997), definem padrões de projeto como "a descrição planejada de objetos e classes interconectados para a resolução de um problema genérico de projeto em um contexto em particular". Um padrão de projeto nomeia, identifica e abstrai os aspectos chave de uma estrutura de projeto identificando-a para criação de objetos reutilizáveis. Nesse contexto, eles são independentes da tecnologia e da arquitetura, dependendo somente do domínio do problema e do contexto em que o mesmo acontece.

Esses padrões fornecem uma linguagem comum que irá facilitar a comunicação entre desenvolvedores e projetistas, melhorando o aprendizado de jovens desenvolvedores e incrementando a padronização do desenvolvimento, permitindo assim a construção de softwares reutilizáveis que se comportam como blocos de construção para sistemas mais complexos.

Segundo Prieto (2001):

> A manutenção de sistemas é a fase do ciclo de vida que mais absorve investimentos e esforços dentro das organizações. Fatores como a falta de documentação, falta de organização do processo de desenvolvimento e a mudança constante de tecnologias, plataformas e ferramentas vêm agravar esse quadro de forma alarmante.

Grande parte dos softwares hoje em funcionamento foi desenvolvida há 10 ou 15 anos atrás, em uma época em que tamanho e espaço de armazenamento eram as principais preocupações em um projeto de software. Esses sistemas usualmente migraram de plataforma e foram adaptados às novas tecnologias sem contar com as mudanças impostas por adaptações

requisitadas por usuários e mudanças nas regras de negócios. Tudo isso resulta em sistemas com problemas de projeto, codificação, lógica e com uma documentação pouco explicativa.

Pressman(2001), afirma que:

> O trabalho de manutenção representa 70% de todo esforço despendido pelas organizações responsáveis por software. E essa taxa tende a aumentar, resultando em uma organização que gasta todo o seu tempo em manutenção de sistemas antigos e não possui disponibilidade para criação de novos softwares.

Tanto os problemas citados por Pressman(2001) quanto os citados por Prieto (2001) podem ser minimizados com a utilização das tecnologias de organização e de implementação apresentadas nesse artigo.

Com a utilização de padrões de projeto é possível a elaboração de sistemas mais flexíveis e funcionais, com maior reutilização e maior qualidade. O tempo de desenvolvimento e de manutenção e o esforço de implementação são reduzidos.

O objetivo do padrão de projeto é definir uma literatura para auxiliar desenvolvedores e projetistas de software na resolução de problemas recorrentes encontrados em qualquer desenvolvimento de software. Levando em consideração o esforço e os custos despendidos atualmente na atividade de manutenção, é grande a necessidade de mecanismos que auxiliem o engenheiro de software na elaboração de diretrizes, com técnicas atuais, que venham a minimizar a necessidade dessa atividade.

Os padrões de projeto têm auxiliado os projetistas no desenvolvimento de sistemas orientados a objetos. Yoder et.al. (1998) apresentam o padrão *Persistence Layer* como uma forma otimizada para uma implementação que utilize uma linguagem orientada a objetos e banco de dados relacional, como a maioria das aplicações existentes na Web atualmente.

O estudo de caso apresentado na seção 3 utiliza esse padrão para organizar e estruturar de forma adequada uma aplicação procurando sanar problemas de manutenção e aumentar os níveis de reuso.

O padrão de projeto *Persistence Layer* consiste em uma camada de persistência que cuida da interface entre objetos de aplicação e tabelas de banco de dados relacional.

Os padrões de projeto apesar de serem relativamente novos vêm sendo amplamente discutidos e estudados pela comunidade de engenharia de software. Eles se apresentam como uma técnica útil e prática que vêm apresentando resultados interessantes e sendo utilizados em diferentes aplicações e situações.

2.2-Servlets

Segundo Deitel et. al. (2001), " a linguagem de programação Java é orientada a objetos, portável, *multithread*, segura, fácil de aprender e permite o acesso a banco de dados via Internet.". Em 1995 a linguagem Java começou a ser utilizada e em pouco tempo foi considerada a melhor solução para aplicações Web. Esse tipo de aplicação opera em um ambiente cliente-servidor e necessita de algum mecanismo intermediário para a comunicação entre o cliente e o servidor. Um dos recursos que ela possui para tratar as solicitações/respostas dos clientes, sem comprometer a performance do servidor, é o *servlet*. A utilização dessa linguagem para aplicações desse tipo ocorreu devido à sua portabilidade e capacidade de processamento múltiplo.

Segundo Deitel et. al. (2001), "as organizações consideram a *Web* de suma importância para suas estratégias de negócio e Java possui recursos que permitem o desenvolvimento de aplicações para arquiteturas cliente-servidor, como é o caso da *Web"*. Essa arquitetura, segundo Sant'Anna (2001), é baseada na divisão de trabalho entre dois processos: o cliente e o servidor. O servidor é uma entidade passiva que nunca inicia a troca de mensagens e executa em uma máquina mais poderosa que a do cliente.

Java possui dois tipos especiais de programas. Estes são *applet* e *servlet*. As *Applets* são executadas no lado cliente (por um browser), agora elas podem ser embutidas em páginas *Web*, sendo assim, elas se tornam interativas.

Os *servlets* são executados pelo servidor, ou seja, por um servidor *Web* que é usado para criar requisições de clientes. Seu procedimento utiliza menos memória melhorando significativamente o desenvolvimento da aplicação. Os s*ervlets* possuem muitas vantagens sobre suas concorrentes e parece-nos uma solução viável para aplicações cliente-servidor.

Camargo(2001) afirma que, "(...) o interesse em desenvolver sistemas orientados a objetos que sejam utilizados via Internet é crescente. A construção de páginas HTML deve ser dinâmica pois os dados atualizados devem estar disponíveis a qualquer momento para os usuários."

Assim como citado por Camargo(2001), os *servlets* podem ser utilizados para a geração de páginas HTML de forma dinâmica. Essa característica é essencial em aplicações *Web*, em especial em Bibliotecas Digitais pois, pode-se apresentar ao usuário informações atualizadas rapidamente. O estudo de caso apresentado por este artigo, na Seção 3, utiliza *servlets* para a geração dinâmica de páginas HTML, mantendo as informações atualizadas.

2.3-Programação Orientada a Aspectos

Apesar do amadurecimento das pesquisas, a manutenção de software permanece como um problema central. Uma das abordagens proposta por Kulesza e Menezes (2000) "é tratar questões de evolução e mudança em um software e construí-lo com base em uma arquitetura e projeto concebidos de forma a serem adaptáveis".

O paradigma de orientação a objeto (OO) tem demonstrado as facilidades que ele pode trazer para o processo de manutenção de *software*, tais como, facilidade de reuso através de relações de herança e composição de classes, e encapsulamento de decisões de

implementação. Entretanto diversos problemas ainda são encontrados durante a evolução desses sistemas, entre eles, entrelaçamento de código relacionado a diferentes interesses, dificuldade no entendimento de colaborações entre classes, aumento da complexidade do sistema com a construção de extensas hierarquias de classes e uso de polimorfismo. A comunidade de engenharia de *software* vem propondo nos últimos anos diversas técnicas de projeto de *software*, tais como, padrões, programação orientada a domínio, programação adaptativa e programação orientada a aspecto.

Programação orientada a aspectos foi proposta originalmente para facilitar a separação de código relativo a requisitos funcionais do referente a requisitos não-funcionais. Os preceitos de programação orientada a aspecto podem levar a várias vantagens como separação de interesses e facilidades na reutilização. Esta programação é uma técnica que complementa o paradigma O.O. para o desenvolvimento de sistemas.

Segundo Elrad et. al (2001), "Programação Orientada a Aspecto [POA] é uma nova evolução na linha de tecnologia para separação de interesses." POA é construída sob tecnologias existentes e provê mecanismos adicionais que possibilitam afetar a implementação do sistema em um modo de *crosscuting*. Em POA, um único aspecto pode contribuir para a implementação de um interesse. Sem esta separação de preocupações, qualquer alteração no código referente a tal preocupação que atravessa o sistema requer a alteração em todas as partes atingidas. Toda vez que uma nova parte é acrescentada ao sistema, o desenvolvedor deve considerar quais preocupações se aplicam ao código em questão. Todos esses problemas levam a ter redundância, redução da legibilidade do código e, uma maior propensão a erros de inconsistência. Com modularização dos aspectos, o código referente a cada preocupação reside em apenas um lugar facilitando a manutenção desse código. Além disso, o código de cada classe fica livre do código relacionado a uma

preocupação adicional, e assim pode ser reusado em diferentes contextos, combinado com diferentes aspectos, dependendo das necessidades da aplicação.

Czarnecki e Eisenecker (2000) comentam que:

> A maioria das notações e linguagens de programação fornece construtores para organizar um sistema em composições hierárquicas de unidades modulares, porém essas unidades concentram-se na busca e composição de unidades funcionais, que são expressas como objetos, módulos e procedimentos. Mas também há interesses (concerns) de um sistema envolvidos em mais de um componente funcional como: sincronização, interação de componentes, persistência e controle de segurança.

Esses interesses são geralmente expressos por pequenos fragmentos de código espalhados pelos componentes funcionais. Eles são chamados de aspectos e alguns são dependentes de um domínio específico, como por exemplo definição de produtos financeiros, enquanto outros são mais gerais, como por exemplo sincronização e fluxo de trabalho.

Segundo Elrad et.al (2001), "muitos interesses não são claramente decompostos em apenas uma dimensão, como por exemplo, apenas objetos". Há certos interesses que cortam transversalmente várias classes e fazem com que o código que trata desse interesse se torne espalhado e entrelaçado com o código que trata de outros interesses do mesmo sistema. A programação orientada a aspectos é baseada na idéia de que sistemas são programados de forma mais adequada por meio da separação de interesses e seus relacionamentos e posteriormente compondo-se esses interesses em um único sistema. Esses interesses podem ser tanto funcionais, como características ou regras de negócio, quanto não-funcionais, como gerenciamento de transação e persistência de dados. Há também propostas de considerar determinados padrões de projeto como interesses de uma aplicação.

Tarr (2002) comenta que "o conceito da separação de interesses tem o objetivo de identificar, encapsular e manipular somente as partes de um *software* que são relevantes para um determinado conceito ou propósito particular". Diferentes espécies de interesses podem ser relevantes para diferentes desenvolvedores com diferentes papéis ou em diferentes

estágios do ciclo de desenvolvimento. Por exemplo, a espécie de interesse dominante no paradigma de objetos é a "classe" já, na modelagem orientada a características são as "características" (*features*) como impressão e persistência e, na programação orientada a aspectos são os "aspectos" como controle de concorrência e distribuição.

2.3.1-Aspect/J

Segundo Kulesza e Menezes (2000), "AspectJ é uma extensão à linguagem Java que inclui suporte de propósito geral à Programação Orientada a Aspectos, proposta pelo *Palo Alto Research Center*, tradicional centro de pesquisas da Xerox ". Ou seja, AspectJ é uma extensão orientada a aspecto para Java e permite especificar "aspectos" que afetam classes e objetos de um programa escrito em Java. AspectJ é implementado na modalidade código-aberto, sendo composto de um compilador que suporta a definição de aspectos, além da linguagem Java como se conhece. Assim como uma classe Java, um aspecto em AspectJ possui um nome e pode declarar atributos e métodos com diferentes visibilidades.

Um *aspecto* é um módulo de código-fonte AspectJ (assim como as classes são em Java) que inclui a definição de *pointcuts*, *advices* e introduções. Um aspecto, assim como uma classe, pode conter atributos e métodos.

Um Aspect pode possuir atributos e métodos e participar de uma hierarquia de aspectos por meio da definição de aspectos especializados. Os pontos de junção em Aspect/J são pontos bem definidos na execução de um programa, como por exemplo: chamada de métodos, acesso a atributos e construção de objetos. Aspect/J permite nomear um conjunto de pontos de junção e associar uma determinada implementação a eles, que pode ser executada antes, após ou durante a execução dos eventos relacionados a esses pontos.

Ao lado do desenvolvimento das linguagens de padrões, outros pesquisadores procuram olhar para a fase de projeto (*design*) e criar processos e notações para o projeto baseado em aspecto.

Chavez e Lucena (2001) apresentam uma alternativa para a carência de métodos de projeto orientados a aspectos citadas por Elrad et.al(2000). Os autores propõem um modelo de projeto para desenvolvimento de software orientado a aspectos que objetiva diminuir a distância semântica existente entre as fases de projeto e implementação. Eles propõem também um conjunto de princípios e regras para a modelagem orientada a aspectos tais como: baixo acoplamento e alta coesão. Os autores apontam que a separação entre as classes base e os aspectos em Aspect/J pode simplificar o desenvolvimento e também as regras de composição.

2.4-Metadados

Segundo Penna et. al. (1977), "os metadados são utilizados para descrever, identificar e definir um recurso eletrônico com o objetivo de modelar e filtrar o acesso, termos e condições para o uso, autenticação e avaliação, preservação e interoperabilidade."

Atualmente os recursos da Internet são considerados importantes meios de comunicação, e estão se transformando em instrumentos para a disseminação de publicações e serviços de informação. Nesse sentido, torna-se um ambiente poderoso para a implementação de procedimentos para a produção de documentos eletrônicos e a elaboração de pesquisa de informações digitais.

Dessa forma, os metadados estão sendo desenvolvidos para estruturar e descrever distintos objetos significativos de informação, armazenados em bibliotecas e centros de documentação e que, por meio de recursos tecnológicos são distribuídos e recuperados através de *site*.

2.5-Cookies

Segundo Deitel et.al.(2000) "(...) uma maneira popular de personalizar as páginas da *Web* é via *cookies"*. Os *cookies* podem armazenar informações sobre o computador do usuário para recuperação, mais tarde, na mesma seção de navegação ou em seções de navegação futuras. Quando o *servlet* recebe a próxima comunicação do cliente, o *servlet* pode examinar os *cookies* que enviou para o cliente em uma comunicação anterior, identificar as preferências dos clientes e imediatamente exibir os produtos de interesse para o cliente.

A finalidade dos *cookies* é de melhorar continuamente as relações individuais com os usuários. Ao fazê-lo, continua-se a utilizar a melhor tecnologia disponível para melhorar a experiência do usuário. Acredita-se que os usuários possam obter o máximo do *site. Cookies* são pequenos arquivos enviados por um *servlet* como parte de uma requisição enviada por um cliente e são umas das formas de identificar um único cliente anônimo e entender sua navegação durante a utilização do *site*. Uma das utilizações dos *cookies* é armazenar senhas e números de identificação de usuários para *sites* específicos. Usos comuns de *cookies* incluem: sistemas de pedido *on-line*, personalização de *site* e rastreio de *site*.

Utiliza-se da informação coletada nos *cookies* para compreender os padrões de uso, oferecer recursos personalizados e identificar problemas que usuários enfrentam à medida que navegam.

3 – Estudo de Caso

A seguir descrevemos o estudo de caso realizado com a utilização das tecnologias citadas anteriormente. O sistema proposto consiste na transação de produtos via *Web*, no qual realiza cadastros de clientes, cadastro de produtos, cadastro de pedido e cadastro de Itens de Pedidos (ARAÚJO, 2002). A figura 1 exibe o modelo de classe do sistema, no qual, contém as classes do padrão, as classes de aplicação, os *servlets* e os aspectos

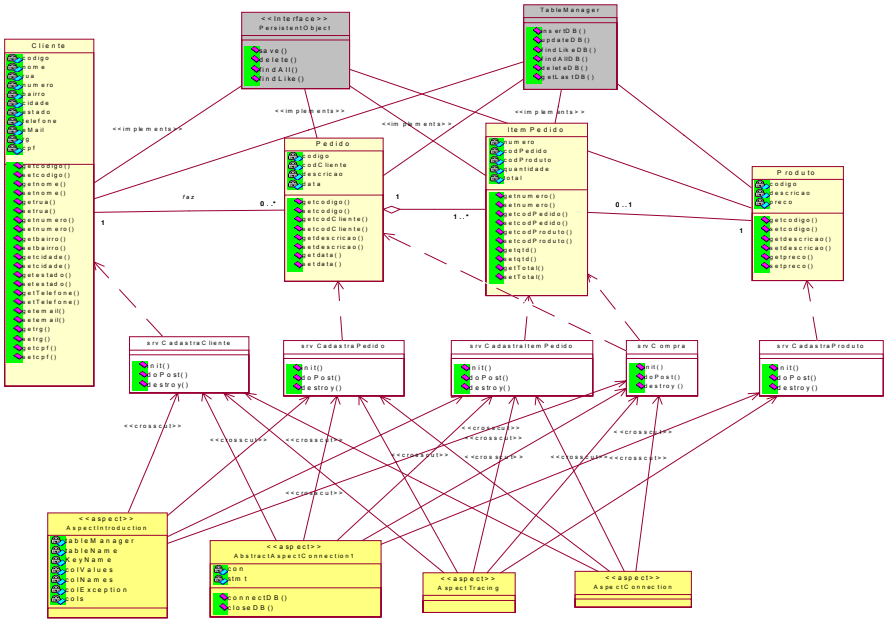

Figura 1 - Modelo de classe
Fonte: Araújo (2002, p.6)

Na Figura 1 estão apresentadas as classes do padrão de projeto *Persistence Layer*, destacadas pela cor cinza; as classes de aplicação destacadas pela cor amarelo-claro; os *servlets* pela cor branca e os aspectos pela cor amarelo.

As classes do padrão de projeto *Persistence Layer* são consideradas como uma camada de persistência. Os autores definem um conjunto de sub-padrões que podem ser utilizados para a implementação dessa camada de persistência, mas apenas alguns sub-padrões foram utilizados pelo estudo de caso:

- Camada Persistente (*Persistent Layer*): camada para salvamento e recuperação de objetos em um banco de dados relacional.
- *CRUD* (Criação (**C**reate), Leitura (**R**ead), Atualização (**U**pdate) e Remoção (**D**elete)): operações mínimas necessárias para a persistência de objetos, que são: criação, leitura, atualização e eliminação.
- Gerenciador de Tabelas (*Table Manager*): permite o mapeamento de um objeto para a sua respectiva tabela (ou tabelas) no banco de dados.
- Gerenciador de Conexão (*Connection Manager*): efetua a conexão do sistema com a base de dados e garante a sua manutenção.

As classes do padrão de projeto se relacionam com todas as classes de aplicação, pois, todas classes de aplicação possuem os métodos que estão definidos nessas, sendo assim, há uma redução de informações, pois não há a necessidade de implementar todos esses métodos nas classes de aplicação novamente, pois essas herdam os atributos e métodos necessários das classes do padrão.

Os aspectos se relacionam com todos os *servlets*, pois, todos *servlets* possuem interesses em comum, no qual os aspectos tratam. Este separa as informações em um módulo, e através dos relacionamentos de heranças os *servlets* utilizam as informações necessárias, sem a necessidade de implementar essas funções ou métodos para cada *servlet.*

As classes desenvolvidas foram: `cliente`, `produto`, `pedido` e `ItemPedido`. Os *servlets* desenvolvidos foram: `srvCadastraCliente`, `srvCadastraProduto`, `srvCadastraPedido`, `srCadastraItemPedido` e `srvCompra`.

Os aspectos desenvolvidos foram: `AspectConnection`, `AbstractAspectConnection`, `AspectIntroduction` e `AspectTracing`. Todos aspectos desenvolvidos refere a requisitos não-funcionais. Estes são apresentados a seguir:

- **AbstractAspectConnection:** este aspecto dever ser reutilizado por outras classes ou aspectos. Este aspecto foi utilizado para conectar e desconectar com o banco de dados, tendo em seu código fonte a implementação dos respectivos métodos. Este foi reutilizado pelo aspecto `AspectConnection`.

- **AspectConnection:** sua função é conectar e desconectar com o banco de dados, porém, este reutilizou ("extends") do `AbstractAspectConnection` e por isso apenas contém em seu código fonte os *JoinPoints*, ou seja, apenas informa quando entrecortar os *servlets* para conectar ou desconectar com o banco, que é no método `init()` e `destroy()`.

- **AspectIntroduction:** este aspecto introduz atributos que serão utilizados por todas classes de aplicação. Neste estudo foi utilizado este aspecto para introduzir os atributos da classe do padrão de projeto `TableManager`, pois todas classes a utilizam, sendo assim, o código fonte das classes de aplicação ficam mais legíveis, pois estes atributos estão implementados no aspecto; e a classe `TableManager` também fica "livre" da implementação dos atributos.

- **AspectTracing:** este aspecto trata das mensagens obtidas na execução do programa, ou seja, mensagens mostradas no servidor. A vantagem é que não precisa colocar mensagens por todo código fonte para verificar até onde o sistema foi executado com sucesso, basta apenas criar este aspecto e descrever as mensagens requeridas. Este entrecorta o momento determinado para aparecer as mensagens nas quais foram implementadas, por exemplo, toda vez que o método `init()` de todos *servlets* forem acessados, uma determinada mensagem é apresentada. As figuras 3 e 4 mostram a vantagem de redução de linhas de código na utilização da programação orientada a aspectos em relação a programação orientada a objeto.

-

POO	POA
public class Cliente implements PersistentObject { private int codigo; private String nome; private String rua; private int numero; private String bairro; private String cidade; private String estado; … private TableManager tablemanager; private String tableName; private String keyName; private Vector colValues; private Vector colNames; private Vector colNamesException; private int cols;	public class Cliente implements PersistentObject { private int codigo; private String nome; private String rua; private int numero; private String bairro; private String cidade; private String estado; …

Figura 3 – Vantagem do AspectIntroduction
Fonte: Araújo (2002, p.17)

As classes de aplicação do sistema foram desenvolvidas da mesma forma como se fossem desenvolvidas apenas com a programação orientada a objeto. A diferença entre ambos é que com a criação do `AspectIntroduction`, as classes não necessitam dos atributos na implementação, sendo assim, os códigos fonte ficam mais legíveis. Essa figura mostra primeiro a implementação sem a utilização de aspecto, demonstrando assim a necessidade de implementar todos esses atributos, sendo que utilizando aspecto, a classe fica desprovida desses atributos.

Outro exemplo é o Aspecto `AspectConnection` basta implementar os pontos de junção, ou seja, os `poincuts`, considerando que o `AspectConnection` *extends* do `AbstractAspectConnection`, no qual, possui a implementação dos métodos e atributos. Na classe `ConnectionManager` precisa implementar todos atributos e métodos para conexão e desconexão. Essas diferenças estão mostradas na Figura 4.

POA

```
public aspect AspectConnection extends AbstractAspectConnection
{
     pointcut doConnection(): call (void init(..));
     pointcut doCloseConnection(): call (void destroy(..));
    }
```

POO

```
public class ConnectionManager {

  static Connection con;
  static Statement stmt;

    public static void ConnectDB () {
    try {
      Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
      con = DriverManager.getConnection ("jdbc:odbc:Compra", "dba", "sql");
      stmt = con.createStatement ();    }
    catch (Exception e) {
      System.out.println ( "problemas para abrir database" );     }  }
  public static void CloseDB () {
    try {
      stmt.close ();
      con.close ();     }
    catch (Exception e) {
      System.out.println ( "problemas para fechar database" ); }}
```

Figura 4 – Vantagens do AspectConnection e AbstractAspectConnection.
Fonte: Araújo (2002, p.20)

As figuras 3 e 4 demonstram a redução de informações nas classes de aplicação e nos *servlets* utilizando os aspectos. Umas das principais vantagens dos aspectos são a fácil localização de determinado erro e posteriormente uma fácil manutenção, pois os interesses estão separados e sendo assim qualquer modificação não causa efeitos colaterais por todo o sistema.

4 – Considerações Finais

As técnicas apresentadas acima fornecem benefícios quando utilizadas no desenvolvimento de sistemas em ambiente Web. Atualmente esse ambiente carece de modelos e métodos que facilitem a organização e a recuperação de informações de determinados domínios. Tais modelos e métodos precisam ser desenvolvidos com ferramentas adequadas ao domínio que se destinam. Outra técnica que pode ser utilizada para agilizar o desenvolvimento de sistemas são os padrões de projeto, pois, essa além de agilizar, permite que muito conhecimento sobre um determinado domínio seja reutilizado.

As técnicas descritas acima podem ser agrupadas de forma a facilitar a recuperação e armazenamento de informações em um determinado domínio na Web. Os estudos mostram que o uso de técnicas é eficaz e eficiente, e que portanto os esforços devem ser concentrados nesta adaptação das técnicas utilizadas na Web, incluindo o desenvolvimento de métricas e técnicas para o gerenciamento de projeto.

Com a realização deste trabalho pôde-se observar a integração de diferentes tecnologias no desenvolvimento de sistemas que são disponibilizados na Internet e que devem ser constantemente atualizados e melhorados.

As tecnologias de organização e implementação apresentadas na seção 3, juntamente coma experiência obtida com o desenvolvimento do estudo de caso demonstram que pode-se obter alguns benefícios em aplicações Web.

Como as Bibliotecas Digitais freqüentemente encontram-se na *Web*, estamos iniciando o desenvolvimento de uma biblioteca digital com o uso dessas tecnologias, objetivando a avaliação da viabilidade e da adequabilidade dessas tecnologias para esse tipo de sistema. Como o estudo de caso, de certa maneira, se assemelha as Bibliotecas Digitais podemos neste momento inferir que os mesmos benefícios obtidos nesse estudo também podem ser obtidos

no desenvolvimento desse tipo de Unidade de Informação. Porém, a conclusão dessa suposição só pode ser concretizada após o desenvolvimento da Biblioteca Digital.

Paralelamente, julgamos necessário um trabalho para conscientizar desenvolvedores de aplicações da Web da importância do uso destas técnicas, mostrando como as mesmas podem tornar seu trabalho mais eficiente e com melhores resultados.

5-REFERÊNCIAS

ARAÚJO, L.S. **Utilização do Padrão de Projeto *Persistence Layer* e *Servlets* na Implementação de um sistema**. Monografia de graduação – Faculdade de Tecnologia de Taquaritinga, FATEC, Taquaritinga, 2002.

ARAÚJO, L.S. **Programação Orientada a Aspectos**. Trabalho de disciplina - Universidade Federal de São Carlos, 2002.

CAGNIN, M.I.; PENTEADO, R.A.D. **Relatório Técnico,** São Carlos, 1999. Dissertação de Mestrado - Universidade Federal de São Carlos, UFSCar, 1999.

CAMARGO, V.V. **Reengenharia orientada a objetos de sistemas COBOL com a utilização de padrões de projeto e servlets.** São Carlos, 2001. Dissertação de Mestrado - Universidade Federal de São Carlos, UFSCar, São Carlos, 2001.

CAMILO, O. A. **Portal de uma Biblioteca Acadêmica: Um projeto para facilitar a recuperação da informação pelo usuário**. Marilia, 2001. Monografia de graduação - Universdade Estadual Paulista, UNESP, Marília, 2001.

CHAVEZ, C.F.G., LUCENA, C.J.P.; **Design Support for Aspect-Oriented Software Development.** Doctoral Symposium at OOSPLA 2001 and Poster Session at OOSPLA 2001, Tampa Bay, Florida, USA, October 14-18, 2001.

CZARNECKI, K., EISENECKER, U. W. **Generative Programming**. Addison Wesley, 2000.

DEITEL, H.M; DEITEL; P.J.**Java como programar**, Porto Alegre: Bookman, 2001.

ELRAD, T., Aksit, M., Kiczales, G., Lieberherr, K., Ossher, H. **Discussing Aspects of AOP. Communications of the ACM**, 2001.

ELRAD, T, Filman R., Bader A. **Aspect-Oriented Programming. Communications of the ACM**, 2001.

GAMMA, E.; et al. **Design Patterns** – *e*lements of reusable object oriented programming. Finland, 1997.

KULESZA, U., MENEZES, D. Reengenharia do Projeto do Servidor Web JAWS utilizando Programação Orientada a Aspecto. In: Simpósito Brasileiro de Engenharia de Software, 2000, São Paulo. **Anais...,** São Paulo:USP, 2000. p. 53-68.

PENNA, C.V; FOSKETT D.J.; SEWELL, P.H. **Serviços de Informação e Biblioteca**. 1997

PRESSMAN, R.S. **Software engineering: a practioner's approach**. 5ed. McGraw-Hill, 2001.

PRIETO, A.G. **Utilização de Padrões de Projetos na Reengenharia de Sistemas**. Dissertação de Mestrado - Universidade Federal de São Carlos, UFSCAR, São Carlos 2001.

SANT'ANNA, A. **Servidores Web:** Modelagem e Análise de Desempenho, São Carlos: Documento de Qualificação. ICMC-USP ,março, 2001.

TARR, P., Ossher, H. Hyper/J - User and Installation Manual. Disponível em: <http://www.research.ibm.com/hyperspace>. Acesso em: 11 dez. 2002.

YODER, J. W.; JOHNSON, R. E.; WILSON, Q. D. Connection business objects to relational databases. In: **CONFERENCE ON THE PATTERN LANGUAGES OF PROGRAMS**, 5., 1998, Monticello-IL, EUA, *Anais...,* 1998.