

# XSpeed: Uma ferramenta para geração de aplicações distribuídas baseadas em padrões

Lincoln S. Rocha, Rute Nogueira, João Gustavo Prudêncio, Rossana Maria de Castro Andrade e Jerffeson Teixeira de Souza

Universidade Federal do Ceará, Departamento de Computação, Campus do Pici  
Bloco 910, 60455-760, Fortaleza – Ceará – Brasil.

{lincoln,rute,gustavo,rossana,jeff}@lia.ufc.br

**Resumo.** Este artigo apresenta a ferramenta, de nome XSpeed, que tem como objetivo principal promover a alta produtividade no desenvolvimento de aplicações distribuídas utilizando padrões para resolver problemas recorrentes no escopo deste domínio. XSpeed recebe como entrada um arquivo XMI contendo o modelo UML de uma aplicação e então realiza a geração automática e produtiva de código para uma plataforma específica. Também neste artigo é apresentado um estudo de caso para ilustrar a dinâmica de funcionamento da ferramenta.

**Abstract.** This paper presents XSpeed, a tool whose main objective is to promote high productivity in the development of distributed applications by using software patterns to solve recurrent problems in the scope of this domain. XSpeed receives as input an XMI file which has the UML model of an application and then performs a productive and automatic code generation for a specific platform. A case study is presented to illustrate the working dynamics of this tool.

## 1. Introdução

O processo de desenvolvimento de aplicações em ambiente distribuído unido à utilização de padrões de *software* é uma tendência natural para alcançar características desejáveis tais como reusabilidade, escalabilidade e flexibilidade [18]. Acontece que a incorporação de tais benefícios pode acarretar problemas relacionados com o aprendizado e a qualidade do código produzido.

No intuito de simplificar a tarefa de desenvolvimento de aplicações distribuídas, inúmeros *frameworks* e arquiteturas foram propostos ao longo do tempo. São exemplos de especificação para arquiteturas distribuídas: Corba [14] e J2EE [21]. Para esta última, existem *frameworks* que promovem o gerenciamento transparente de troca de mensagens e persistência de dados.

Com igual propósito, padrões de *software* vêm sendo estudados e documentados a fim de permitir a reutilização de soluções existentes, seja na forma de projetos ou idéias similares aplicáveis no processo de desenvolvimento de *software*.

Ferramentas de geração automática constituem uma outra abordagem para conferir maior rapidez no desenvolvimento de *softwares*. Os programas gerados compartilham as características do programa modelo. Assim, se este está sintaticamente correto e bem

testado, os programas gerados também estarão. Dessa forma, essa geração automática pode oferecer um programa com menos erros e maior qualidade.

Este artigo propõe uma ferramenta de nome XSpeed, que busca integrar tecnologias e facilitar a reutilização de padrões no processo de desenvolvimento de aplicações distribuídas através da geração automática e produtiva de código a partir de modelos UML [7], disponibilizados através de arquivos no formato XMI (*XML Metadata Interchange*) [15]. Desta maneira, as novas aplicações geradas seguirão um modelo de arquitetura padronizado, facilitando o entendimento e manutenibilidade do sistema como um todo.

O restante deste artigo é descrito como segue: na seção 2 encontram-se alguns padrões de *software* e ferramentas relacionadas com o desenvolvimento de aplicações distribuídas; na seção 3 é apresentada uma descrição da arquitetura do XSpeed; na seção 4 é mostrado um estudo de caso da ferramenta e na seção 5 são apresentadas algumas conclusões obtidas com a confecção deste trabalho bem como pretensões para trabalhos futuros.

## **2. Padrões de *Software***

A reutilização de padrões no desenvolvimento de um *software* tem emergido como uma das mais promissoras abordagens para a melhoria da qualidade dos artefatos de *software*, pois eles permitem que a experiência de desenvolvedores seja documentada e reutilizada, registrando-se soluções de projeto para um determinado problema em um contexto particular.

### **2.1 Ferramentas e Padrões Relacionados**

Esta subseção é dedicada à apresentação de trabalhos relacionados. Aqui são apresentados alguns padrões e ferramentas que buscam solucionar problemas envolvidos com o desenvolvimento de aplicações distribuídas.

O trabalho [8] apresenta os padrões DAP-EJB (Distributed Adapters Pattern with EJB) e PDC-EJB (Persistent Data Collections with EJB) que auxiliam na estruturação de aplicações EJB [23]. Essa estruturação pode acontecer em sistemas já existentes, sem EJB, bem como em projetos de novos sistemas, os quais obterão alguns benefícios como reusabilidade, extensibilidade, modularidade, independência de tecnologia (distribuição ou dados) e desempenho.

Também, em [17] tem-se o Padrão de DBCD (Desenvolvimento Baseado em Componentes Distribuídos), cuja intenção é a criação de componentes distribuídos reutilizáveis para diferentes domínios de aplicações. O Padrão proposto trabalha com a integração de diferentes tecnologias em uma ferramenta CASE, para apoiar o Desenvolvimento Baseado em Componentes (DBC), além de cobrir todo o ciclo de vida dos componentes distribuídos e definir mudanças no código de comunicação dos componentes.

Em [3] é apresentado um método de implementação que orienta a transformação progressiva que torna uma aplicação inicialmente centralizada em uma distribuída. O método ameniza a complexidade inerente a sistemas distribuídos e torna os testes mais efetivos. Além disso, esse método utiliza o Distributed Adapters Pattern (DAP) apresentado em [2] promovendo uma maior modularidade, reuso, extensibilidade assim como uma implementação progressiva.

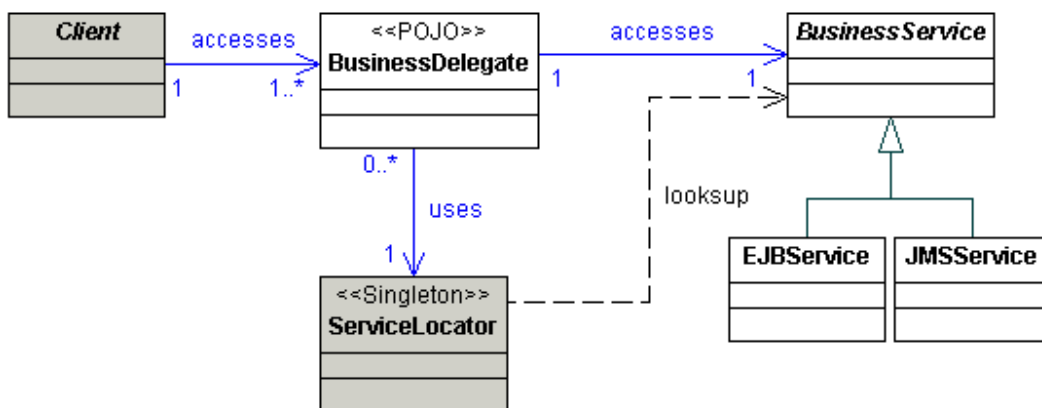
A ferramenta Cordel apresentada em [10] promove de forma flexível a geração, compilação e implantação automática de sistemas Web, com base em tecnologias de objetos distribuídos, a partir de modelos UML, seguindo especificações e tecnologias já consolidadas. Esses sistemas que adotam mecanismos que favorecem a construção de aplicações reutilizáveis, flexíveis e escaláveis trazem consigo a consequência natural de um maior esforço no processo de desenvolvimento, devido à adoção de especificações, como J2EE ou Corba que, por sua vez, implicam numa maior necessidade de aprendizado da tecnologia e uma maior quantidade do código a ser escrito.

Além da ferramenta Cordel, pode-se destacar como importante para este trabalho a ferramenta AndroMDA [4], que trabalha com a idéia de gerar código fonte a partir da especificação UML. Como consequência, o código fonte torna-se pouco relevante, pois os diagramas UML é que são essenciais para a implementação. Além disso, a ferramenta sugere o uso de diversas tecnologias para persistência dos dados como *Hibernate*, *Spring* e *SOAP*, ao mesmo tempo em que se propõe a diminuir o tempo de desenvolvimento de programas e promover o uso intensivo de padrões.

## 2.2 Padrões Utilizados

Para o desenvolvimento da ferramenta, vários padrões foram estudados com a finalidade de encontrar os que mais se adequassem à solução proposta. A seguir, os padrões escolhidos são apresentados.

O padrão BD (*Business Delegate*) [1], normalmente, é utilizado para reduzir o acoplamento entre os clientes da camada de apresentação e os serviços de negócios. O BD oculta os detalhes de implementação por trás do serviço de negócios, como forma de pesquisa e acesso em ambientes remotos. A **Figura 1** mostra o diagrama de classes do padrão BD.



**Figura 1. Diagrama de classes do BD**

Já o BO (*Business Object*) [1] é empregado para separar dados e lógica de negócio usando um modelo orientado a objeto. O BO fará, portando, o encapsulamento das regras de negócio e do gerenciamento da camada de persistência. Na **Figura 2** é exposto o diagrama de classes do BO.

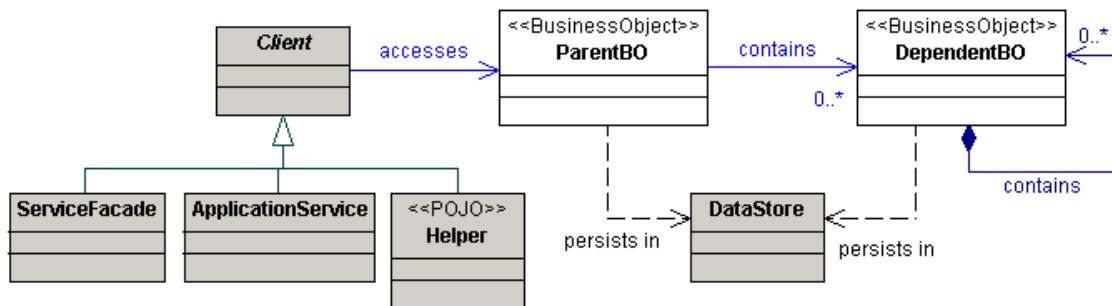


Figura 2. Diagrama de classes do BO

Segundo [1], o acesso a dados varia dependendo da sua origem. O acesso ao armazenamento persistente, como em um banco de dados, varia muito dependendo do tipo de armazenamento e da implementação do desenvolvedor. Através da utilização do padrão DAO (*Data Access Object*) pode-se extrair e encapsular todos os acessos à origem de dados em um único objeto. O DAO gerencia a conexão com a fonte de dados para obter e armazenar dados. Seu diagrama de classes pode ser observado na **Figura 3**.

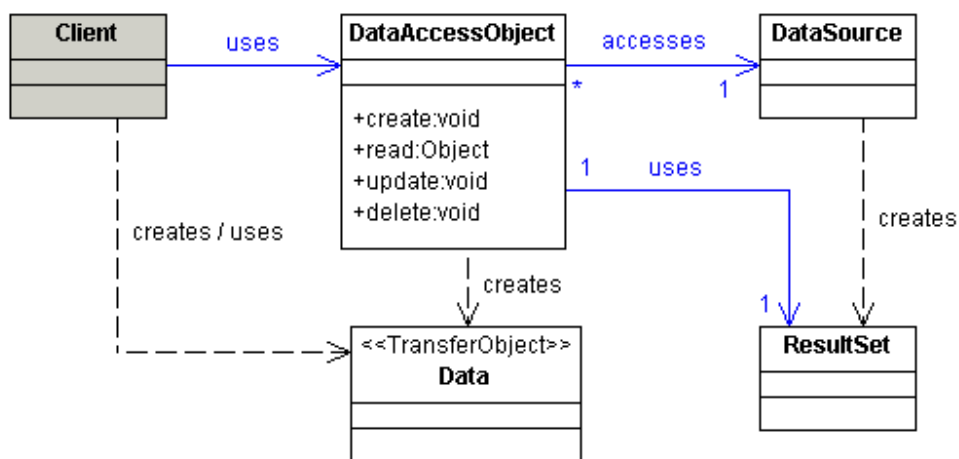


Figura 3. Diagrama de classes do DAO

### 3. XSpeed e sua Arquitetura

XSpeed é uma ferramenta que tem como objetivo principal promover a alta produtividade no desenvolvimento de aplicações distribuídas utilizando padrões para resolução de problemas recorrentes. A ferramenta recebe como entrada um arquivo no formato XMI contendo o modelo UML da aplicação a ser gerada. Em seguida, mapeia os padrões que deverão ser aplicados como estratégia de resolução de problemas e finalmente faz a geração automática do código interligando as tecnologias específicas para a resolução de cada um dos problemas inerentes ao domínio.

A arquitetura do XSpeed, demonstrada na **Figura 4**, se divide em três módulos básicos: módulo de conversão, módulo de configuração e módulo de geração.

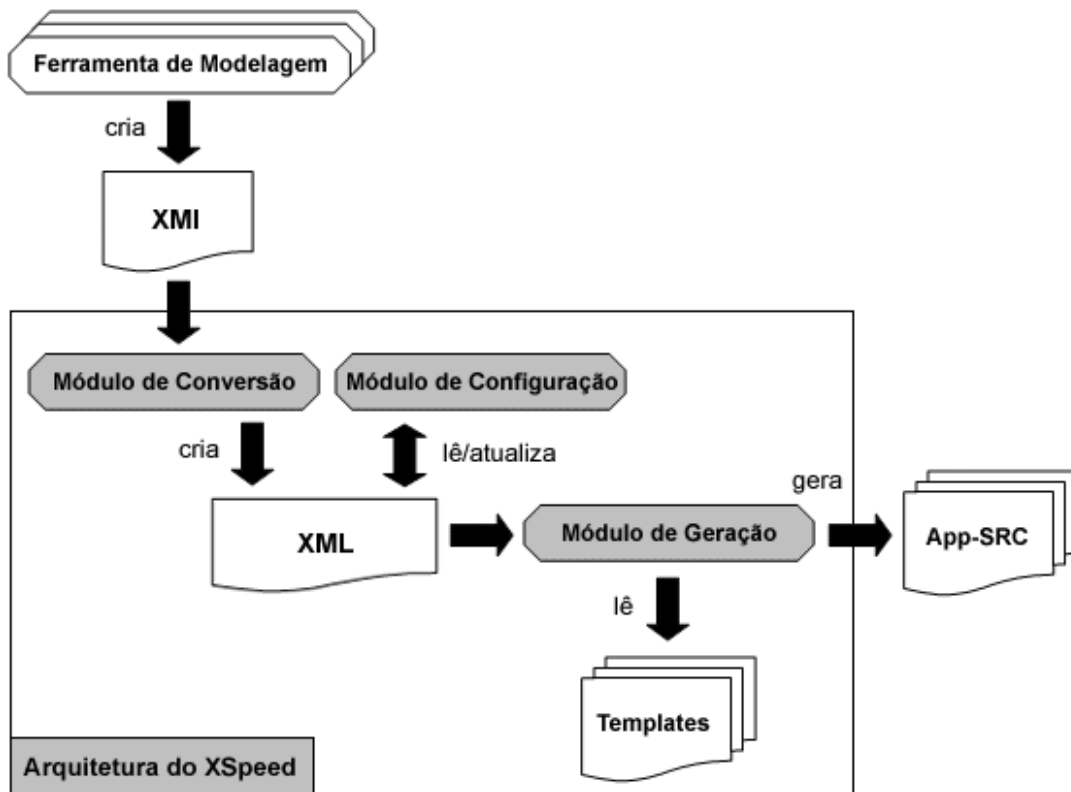


Figura 4. Arquitetura geral da ferramenta XSpeed

### 3.1. Módulo de Conversão

O módulo de conversão faz a extração das informações relacionadas a cada uma das classes do modelo UML contido no arquivo XMI, disponibilizando-as como uma instância do modelo de representação descrito na **Figura 5**. Em seguida, esta instância é armazenada em um arquivo XML intermediário que servirá como base para geração da aplicação.

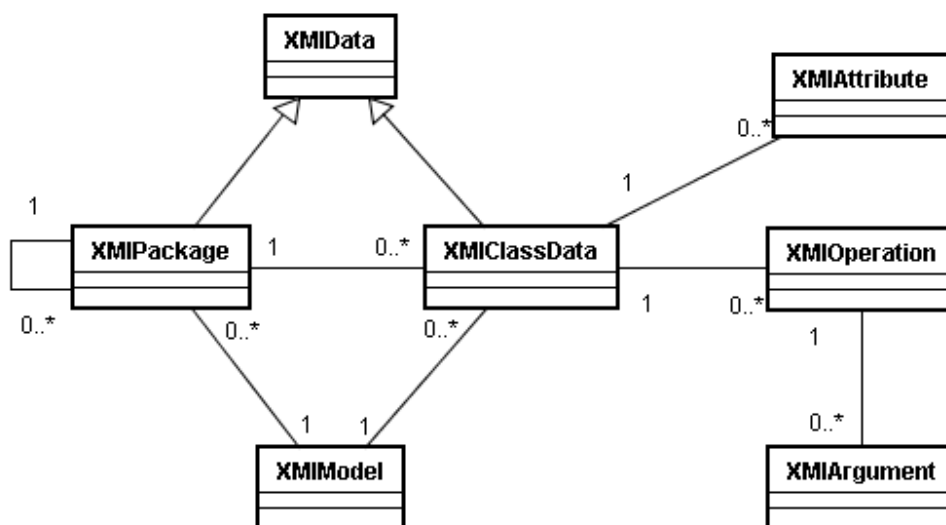


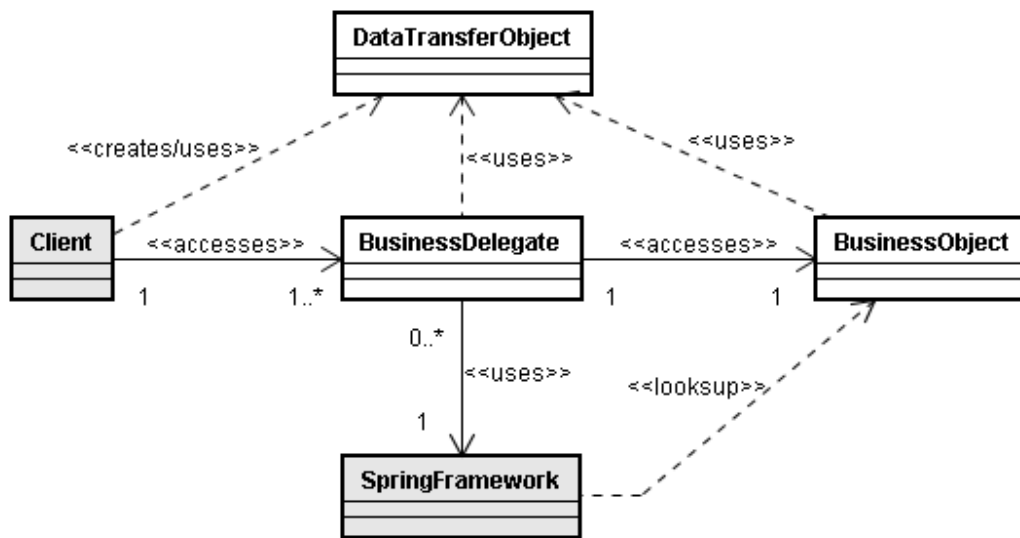
Figura 5. Modelo de mapeamento do XMI

Aqui, uma filtragem por meio de um parser é concebida sobre o arquivo XMI de entrada. Apenas informações relevantes para o XSpeed são selecionadas, tais como a estrutura e o relacionamento das classes do modelo. Desse modo, não são avaliadas as informações contidas em outros possíveis diagramas como os de casos de uso, colaboração e seqüência.

### 3.2. Módulo de Configuração

Neste módulo, é feita uma seleção manual dos padrões e das tecnologias, de persistência e acesso remoto, a serem aplicadas sobre as classes do modelo para resolução dos problemas específicos do domínio. A versão atual do XSpeed propõe um conjunto de padrões para resolução de problemas específicos de aplicações distribuídas tais como delegação de serviço, processamento de lógica de negócio e persistência.

Para resolver o problema de acesso remoto à lógica de negócio, adotou-se o padrão BD, apresentado na seção 2.2, integrado ao *framework Spring* [19] [25] que fornece uma estrutura transparente de comunicação distribuída. De acordo com o diagrama de classes observado na **Figura 6**, o código cliente fará apenas chamadas locais. Assim sendo, toda a complexidade das chamadas remotas são encapsuladas pelo BD. Além de facilitar o desenvolvimento da aplicação cliente, o seu simples uso promove a separação explícita entre a camada de apresentação e a tecnologia envolvida na camada de negócio.



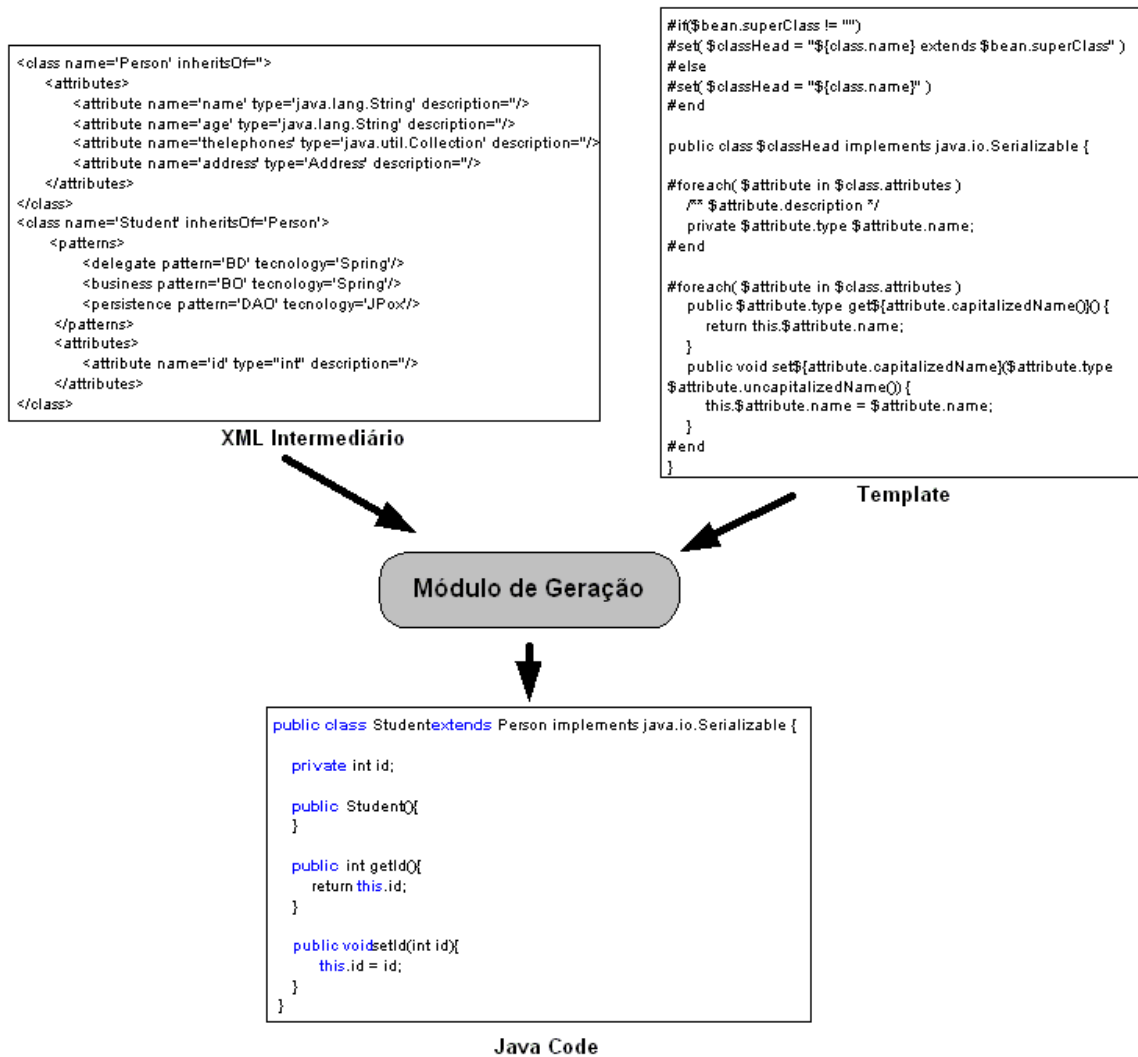
**Figura 6. Diagrama de classes do BD adaptado**

Na camada de lógica de negócio adotou-se o padrão BO, descrito na seção 2.2, que faz o encapsulamento das regras de negócio e do gerenciamento da camada de persistência. O acesso à camada de persistência pode acontecer de maneira remota (**Figura 7**), portanto, o BO também faz uso do *framework Spring*.



### 5.3. Módulo de Geração

Este módulo é responsável por fazer a geração de código de todas as classes da aplicação. XSpeed faz uma junção entre as informações contidas no arquivo XML intermediário e os templates (**Figura 9**) que definem a estrutura das classes, inclusive o formato dos padrões a serem aplicados. Só então a geração de código é realizada para uma plataforma específica. A versão atual da ferramenta possibilita a geração apenas para a plataforma Java [20].



**Figura 9. Processo de geração de código**

O motor de geração do XSpeed utiliza o Velocity Template Engine [5] para fazer a junção entre as informações contidas no arquivo XML intermediário e os templates. O Velocity possui uma linguagem de manipulação de template: a VTL (*Velocity Template Language*), que permite fazer a inserção de informações de maneira dinâmica dentro do template. O trecho de código (**Figura 10**) a seguir mostra a utilização da VTL para fazer a geração dos atributos de uma classe e seus respectivos métodos de acesso.

```

#foreach( $attribute in $bean.attributes )
  /** $attribute.description */
  private $attribute.type $attribute.name;
#end

#foreach( $attribute in $bean.attributes )
  public $attribute.type get${attribute.capitalizedName()}() {
    return this.$attribute.name;
  }
  public void set${attribute.capitalizedName()}($attribute.type $attribute.uncapitalizedName()) {
    this.$attribute.name = $attribute.name;
  }
#end

```

Figura 10. Template Velocity

4. Estudo de Caso

Nesta seção são mostradas todas as etapas para geração de uma aplicação distribuída utilizando XSpeed. A aplicação escolhida como exemplo consiste na manutenção de um cadastro simplificado de estudantes universitários. O diagrama de classes da Figura 11 representa o modelo UML da aplicação.

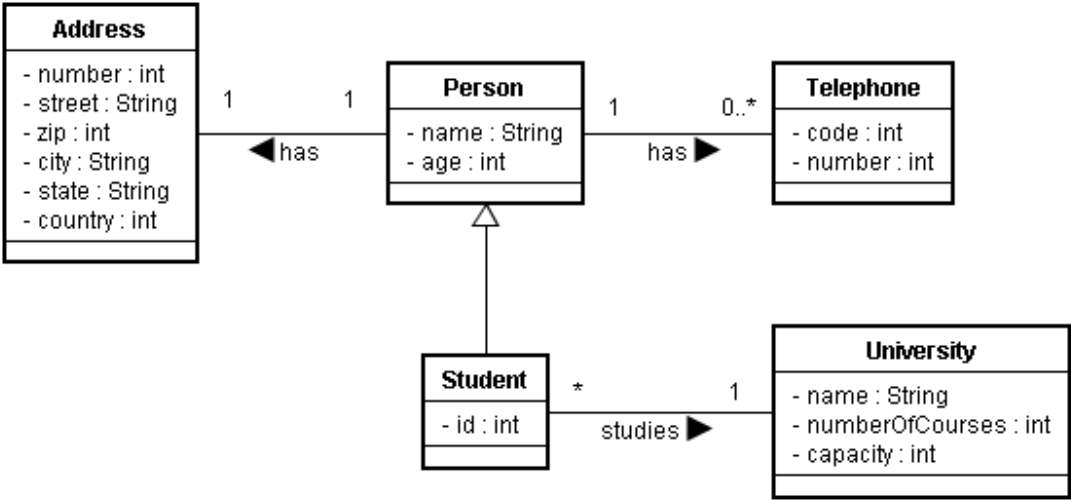


Figura 11. Diagrama de classes da aplicação

O modelo exibido na Figura 11 pode ser criado e convertido para o formato XMI por meio de uma ferramenta de modelagem UML, tais como Rational Rose [12], ArgoUML [24] e Poseidon [9]. A Figura 12 mostra a descrição das classes *Person* e *Student* no formato resultante da conversão.

```

<UML:Class xmi.id = 'I1b8378fm103281c4dc2mm7f3e' name = 'Person' visibility = 'public'
isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'
isActive = 'false'>
<UML:Classifier.feature>
  <UML:Attribute xmi.id = 'I1b8378fm103281c4dc2mm7ecc' name = 'name' visibility = 'private'
isSpecification = 'false' ownerScope = 'instance' changeability = 'changeable'>
  <UML:StructuralFeature.type>
    <UML:Class xmi.idref = 'I1b8378fm103281c4dc2mm7ebb' />
  </UML:StructuralFeature.type>
</UML:Attribute>
<UML:Attribute xmi.id = 'I1b8378fm103281c4dc2mm7eba' name = 'age' visibility = 'private'
isSpecification = 'false' ownerScope = 'instance' changeability = 'changeable'>
  <UML:StructuralFeature.type>
    <UML:DataType xmi.idref = 'I1b8378fm103281c4dc2mm7ef2' />
  </UML:StructuralFeature.type>
</UML:Attribute>
</UML:Classifier.feature>
</UML:Class>
<UML:Class xmi.id = 'I1b8378fm103281c4dc2mm7f18' name = 'Student' visibility = 'public'
isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'
isActive = 'false'>
<UML:GeneralizableElement.generalization>
  <UML:Generalization xmi.idref = 'I1b8378fm103281c4dc2mm7d7c' />
</UML:GeneralizableElement.generalization>
<UML:Classifier.feature>
  <UML:Attribute xmi.id = 'I1b8378fm103281c4dc2mm7ea9' name = 'id' visibility = 'private'
isSpecification = 'false' ownerScope = 'instance' changeability = 'changeable'>
  <UML:StructuralFeature.type>
    <UML:DataType xmi.idref = 'I1b8378fm103281c4dc2mm7ef2' />
  </UML:StructuralFeature.type>
</UML:Attribute>
</UML:Classifier.feature>
</UML:Class>

```

**Figura 12. Estrutura do arquivo XMI**

O arquivo XMI gerado é fornecido como entrada para XSpeed que, por sua vez, faz a filtragem das informações de cada uma das classes para o seu modelo de representação, descrito na seção 3.1. Em seguida, atualiza o arquivo XML intermediário incorporando as novas características da aplicação. A **Figura 13** mostra as classes *Person* e *Student* descritas no formato deste arquivo.

```

<class name='Person' inheritsOf=''>
  <attributes>
    <attribute name='name' type='java.lang.String' description=''/>
    <attribute name='age' type='java.lang.String' description=''/>
    <attribute name='telephones' type='java.util.Collection' description=''/>
    <attribute name='address' type='Address' description=''/>
  </attributes>
</class>
<class name='Student' inheritsOf='Person'>
  <attributes>
    <attribute name='id' type='int' description=''/>
  </attributes>
</class>

```

**Figura 13. Arquivo XML intermediário**

O arquivo XML intermediário sofre algumas alterações durante o processo de mapeamento dos padrões e das tecnologias que serão aplicadas sobre as classes do modelo.

No exemplo específico (**Figura 14**) foram mapeados, sobre a classe *Student*, os padrões: BD, BO e DAO e as tecnologias: *Spring* e *JPOx*.

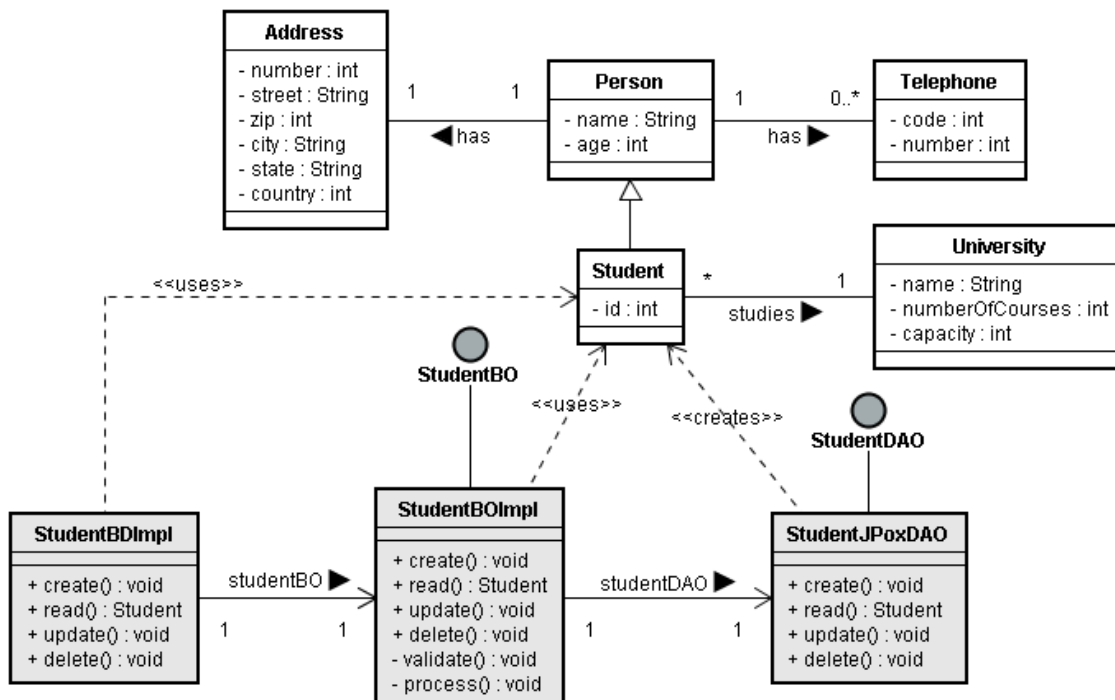
```

<class name='Person' inheritsOf=''>
  <attributes>
    <attribute name='name' type='java.lang.String' description=''/>
    <attribute name='age' type='java.lang.String' description=''/>
    <attribute name='telephones' type='java.util.Collection' description=''/>
    <attribute name='address' type='Address' description=''/>
  </attributes>
</class>
<class name='Student' inheritsOf='Person'>
  <patterns>
    <delegate pattern='BD' tecnologia='Spring'/>
    <business pattern='BO' tecnologia='Spring'/>
    <persistence pattern='DAO' tecnologia='JPOx'/>
  </patterns>
  <attributes>
    <attribute name='id' type='int' description=''/>
  </attributes>
</class>

```

**Figura 14. Arquivo XML intermediário mapeado**

Após o mapeamento dos padrões e das tecnologias, o código é gerado. O diagrama de classes da **Figura 15** mostra o novo formato da aplicação com a incorporação das classes *StudentBDImpl*, *StudentBOImpl* e *StudentJPoxDAO* e das interfaces *StudentBO* e *StudentDAO*.



**Figura 15. Diagrama de classes da aplicação após a geração**

A **Figura 16** exibe um fragmento de código da classe *StudentBDImpl* que descreve como é obtido uma instância remota de *StudentBOImpl* utilizando o *framework Spring*.

```
private StudentBO studentBO;
public StudentBD() throws Exception {
    BeanFactoryLocator beanFactoryLocator = SingletonBeanFactoryLocator.getInstance("spring/beanRefFactory.xml");
    BeanFactory beanFactory = beanFactoryLocator.useBeanFactory("br.ufc.mcc.students").getFactory();
    this.studentBO = (StudentBOImpl) beanFactory.getBean("studentBO");
}
```

**Figura 16. Obtenção de um BO**

Do mesmo modo (**Figura 17**), a classe *StudentBOImpl* obtém uma instância de *StudentJPoxDAO* para fazer o gerenciamento da camada de persistência.

```
private StudentDAO studentDAO;
public StudentBOImpl() throws Exception {
    BeanFactoryLocator beanFactoryLocator = SingletonBeanFactoryLocator.getInstance("spring/beanRefFactory.xml");
    BeanFactory beanFactory = beanFactoryLocator.useBeanFactory("br.ufc.mcc.students").getFactory();
    this.studentDAO = (StudentJPoxDAO) beanFactory.getBean("studentDAO");
}
```

**Figura 17. Obtenção de um DAO**

Por fim, a **Figura 18** exibe o formato do código da *StudentJPoxDAO* responsável por fazer a persistência transparente dos objetos da aplicação.

```
public class StudentJPoxDAO extends org.springframework.orm.jdo.support.JdoDaoSupport {

    public void create(Student student) {
        getJdoTemplate().makePersistent(student);
    }

    (...)

}
```

**Figura 18. Persistência transparente com JPox**

## 5. Conclusões e Trabalhos Futuros

Ao fim deste trabalho conclui-se que a utilização de padrões no desenvolvimento de aplicações, independentemente do domínio de atuação, é um processo que requer dos desenvolvedores um elevado grau de conhecimento específico sobre onde encontrar, como e quando utilizar padrões de *software*. Além disso, observa-se que a utilização de padrões no desenvolvimento de aplicações distribuídas implica na realização de tarefas repetitivas e enfadonhas de codificação. Nesse contexto, a ferramenta apresentada neste artigo visa facilitar e difundir a utilização de padrões com o intuito de maximizar a produtividade e a qualidade dos artefatos de *softwares* produzidos.

Como trabalhos futuros, pretende-se fazer a interligação da ferramenta com repositórios de padrões para facilitar o processo de identificação de padrões relacionados para o desenvolvimento de aplicações para um domínio específico. Neste sentido, deseja-se expandir ao máximo o escopo de atuação da ferramenta no intuito de possibilitar o desenvolvimento de aplicações para um número maior de domínios.

## Referências

- [1] Alur, D., Crupi, J., Malks, D. Core J2EE Patterns: Best Practices and Design Strategies. 2ed Edition. Prentice Hall, 2003.
- [2] Alves, V. Progressive development of distributed object-oriented applications. Master's thesis, Centro de Informatica Universidade Federal de Pernambuco, Feb, 2001.
- [3] Alves, V., Borba, P. An Implementation Method for Distributed Object-Oriented Applications. In *Second Latin American Conference on Pattern Languages of Programming*, SugarLoafPLoP'2002, pages 55-86, Itaipava, Brazil, 5th-7th August 2002.
- [4] AndroMDA Tool. <http://www.andromda.org>. Acesso em janeiro de 2005.
- [5] Apache. Velocity Template Engine (Velocity). <http://jakarta.apache.org/velocity>. Acesso em novembro de 2004.
- [6] Bauer, C. and King, G. Hibernate in Action. Softbound, 2004.
- [7] Booch, G., Rumbaugh, J. and Jacobson, I. The Unified Modeling Language User Guide. Addison-Wesley, 1999.
- [8] Dias, K., Borba, P. Padrões de projeto para estruturação de aplicações distribuídas Enterprise JavaBeans. In *Second Latin American Conference on Pattern Languages of Programming*, SugarLoafPLoP'2002, pages 55-86, Itaipava, Brazil, 5th-7th August 2002.
- [9] Gentleware. Poseidon for UML. <http://www.gentleware.com/index.php>. Acesso em janeiro de 2005.
- [10] Greve, F.G.P., Araújo, J.G.R., Andrade, S.S., Maia Filho, E.M.F., Brito, K.S., Rocha, L.A., Pinheiro, V.G. Cordel: uma Ferramenta Distribuída para a Geração de Aplicações Web. In *I2TS 3rd International Information and Telecommunication Technologies Symposium*, São Carlos, 2004. I2TS 3rd International Infor.
- [11] Hibernate. Relational Persistence For Idiomatic Java. <http://www.hibernate.org>. Acesso em janeiro de 2005.
- [12] IBM. Rational Rose. <http://www-306.ibm.com/software/rational>. Acesso em dezembro de 2004.
- [13] Jpox - JDO. Java Persistent Object (JPOX). <http://www.jpox.org/index.jsp>. Acesso em janeiro de 2005.
- [14] OMG. Corba Specification. [www.omg.org/gettingstarted/corbafaq.htm](http://www.omg.org/gettingstarted/corbafaq.htm). Acesso em julho de 2002.
- [15] OMG. XML Metadata Interchange (XMI). [www.omg.org/technology/documents/formal/xmi.htm](http://www.omg.org/technology/documents/formal/xmi.htm). Acesso em dezembro de 2003.
- [16] Reese, G. Database Programming with JDBC and Java. 2nd Edition. O'Reilly, 2000.

- [17] Santana, E., Bianchini, C.P., Prado, A.F., Trevelin, L.C. Um Padrão para o Desenvolvimento de Software Baseado em Componentes Distribuídos. In Second Latin American Conference on Pattern Languages of Programming, SugarLoafPloP'2002, pages 175-190, Itaipava, Brazil, 5th-7th August 2002.
- [18] Schmidt, D.C., Fayad, M.E., Johnson, R.E. Building Application Frameworks: Object-Oriented Foundations of Framework Design. Wiley Computing Publisher, 1999.
- [19] Spring. Java/J2EE Application Framework. <http://www.springframework.org>. Acesso em janeiro de 2005.
- [20] Sun Microsystems. Java Technology. <http://java.sun.com>, Acesso em janeiro de 2000.
- [21] Sun Microsystems. Java 2 Platform, Enterprise Edition (J2EE). [java.sun.com/j2ee](http://java.sun.com/j2ee), Acesso em dezembro de 2004.
- [22] Sun Microsystems. Java Database Connectivity (JDBC). <http://java.sun.com/products/jdbc>. Acesso em julho de 2004.
- [23] Sun Microsystems. Enterprise JavaBeans Technology (EJB). <http://java.sun.com/products/ejb>. Acesso em julho de 2004.
- [24] Tigris. Modelling tool ArgoUML. <http://argouml.tigris.org>. Acesso em janeiro de 2005.
- [25] Walls, C., Breidenbach, R. Spring in Action. Softbound, 2005.