Universidade Federal do Ceará – UFC
Departamento de Computação
Especialização em Tecnologias da Informação 2004

JNXD
Um Framework de Persistência Orientado a Objetos.

Ênfase no componente JNXDBuilder.

Rafael de Oliveira Lopes

Fortaleza – Ceará – 2007

Rafael de Oliveira Lopes

JNXD
Um Framework de Persistência Orientado a Objetos.

Ênfase no componente JNXDBuilder.

Monografia para conclusão do curso de Especialização em Tecnologia da Informação com Ênfase no Desenvolvimento Web (Turma 2004), oferecido pela Universidade Federal do Ceará.

Orientadora: Prof. Vânia Maria Ponte Vidal.

Fortaleza – Ceará – 2007.

JNXD
Um Framework de Persistência Orientado a Objetos.

Ênfase no componente JNXDBuilder.

Rafael de Oliveira Lopes

PARECER: _____

DATA: ___/___/____

BANCA EXAMINADORA:

_____
Vânia Maria Ponte Vidal
(Orientadora)

_____

AGRADECIMENTOS

Primeiramente a Deus por ter me dado força e determinação para concluir o curso. À minha família, que foi sempre o alicerce da minha vida e minha principal incentivadora. Aos professores do curso que puderam engrandecer o conhecimento que possuo hoje e que poderei utilizar acadêmica ou profissionalmente.

RESUMO

Este trabalho descreve o JNXD (Java Native XML Database), um framework para persistência de objetos em XML data sources, evitando o mapeamento OO-XML. O JNXD permite que aplicações interajam com bancos de dados XML nativos (*Native XML Databases*).

JNXD provê um mecanismo de serialização para conversão de documentos XML recuperados do banco de dados em POJOs (Plain Old Java Objects), assim como seu processo inverso. O JNXD vem solucionar problemas enfrentados pelo mapeamento objeto-relacional utilizado por outros frameworks de persistência e permitir um uso mais eficaz das linguagens de acesso a documentos XML definidos pelo W3C.

LISTA DE FIGURAS

Figura 3.1 – Console cliente do eXist-db.....13
Figura 3.2 – Estrutura do documento jnxd.xsd 15
Figura 3.3 – Documento de definição do Spring (jnxd-applicationContext.xml) ..16
Figura 4.1 – Arquitetura do JNXD framework. 17
Figura 5.1 – Componentes do JNXD framework.....19
Figura 5.2 – Diagrama de classes do componente JNXDBuilder.....20
Figura 6.1 – Documento nxd.xml 22
Figura 6.2 – Exemplo de documento de mapeamento de cosultas. 23
Figura 7.1 – JNXD no java.net 26

SUMÁRIO

1. Introdução 8
2. Características do JNXD framework 10
3. Ferramentas e APIs 12
3.1. eXist-db 12
3.2. XML:DB API 13
3.3. XStream API 14
3.4. JAXB API 14
3.5. Spring Framework 15
4. Arquitetura 17
5. Componentes 19
5.1. JNXD Builder 20
6. Configuração da geração automática da estrutura NXD. 21
7. JNXD no java.net 25
8. Conclusão 27
9. Referências 28
10. Código Fonte JNXD Builder 29

1. Introdução

Dentre as inúmeras tecnologias que surgem a cada dia, a plataforma JAVA desponta como uma das mais promissoras no mercado de TI, para dar suporte ao desenvolvimento de aplicações portáveis entre os milhares dispositivos móveis e desktops intercomunicáveis existentes e aos que ainda estão por vir. A subdivisão da plataforma Java em Java Standard Edition, Java Enterprise Edition e Java Micro Edition nada mais é do que uma tentativa de abranger e dividir o suporte ao desenvolvimento em dimensões diferentes de hardware e software, cujos limites tendem a se estreitar com o passar do tempo.

O crescimento vertiginoso do uso da *Extended Markup Language* (XML) para a comunicação entre sistemas heterogêneos e distribuídos evidencia necessidades de armazenamento próprias para esse fim. Por isso, os bancos de dados relacionais e objeto-relacionais tendem a disponibilizar novas capacidades de processamento e mecanismos de consultas em documentos XML. Além disso, a integração entre os sistemas reforçados pelo uso dos web services, para alavancar os negócios de empresas no mundo, põem em pauta o paradigma objeto-relacional que “resolve” o gap entre a visão do modelo de objetos e o modelo de dados relacional (físico). Sendo assim, focamos nosso trabalho em bancos estruturados para armazenamento nativo em XML, especificamente no domínio de aplicações que usam Native XML Databases.

Procurando fugir do problema clássico do mapeamento objeto-relacional em frameworks como o Hibernate, iBatis e tecnologias como JDO, EJB(Entity Beans) e outros, esse trabalho propõe o desenvolvimento de um framework capaz de persistir objetos serializados em XML, para obter todo o potencial de linguagens de consulta como XQuery, XPath e XUpdate, ou seja, não usar arquivos de mapeamento de objetos para o modelo de armazenamento físico.

Esse trablaho procura focar mais precisamente em um dos componentes pertencentes a este framework: JNXDBuilder. Mais a frente iremos ver a importância

do mesmo e sua interação entre os demais componentes presentes no JNXD (*Java Native XML Database*) framework.

2. Características do JNXD framework

O framework proposto satisfaz as seguintes características:

- Não deve existir arquivo de mapeamento entre classes e o XML de armazenamento.
- Adotar uma abordagem não intrusiva das classes a serem persistidas. Todo objeto de domínio é passível de ser armazenado.

- Todos os atributos de um objeto são armazenados, inclusive, caso existam, agregações de outros objetos.

- Deve existir uma configuração que informe quais são os locais físicos de armazenamento de cada entidade no banco. Ou seja, uma associação entre URIs e domínios. *Por exemplo:*
    - *O domínio "funcionário", representado pela classe br.com.xxx.Funcionario.java, deve ser armazenado na URI "/empresa/departamento/funcionario".*

- Depois de persistidos, os XMLs podem ser consultados usando as linguagem XQuery, XPath, XUpdate. Dependendo, do suporte a essas linguagens, que varia de NXD para NXD.

- Os resultados das consultas, que são documentos XML, devem ser deserializados para objetos. Se o XML retornado for uma "mesclagem" de XMLs de domínios diferentes, o framework deve permitir a deserialização para um Value Object genérico.

- As consultas sobre documentos XML são armazenadas em documentos XML.

- Dado um XML de configuração, a estrutura hierárquica do banco NXD pode ser gerada automaticamente.

Na figura abaixo, sugerimos a idéia básica da persistência de objetos Java/XML:

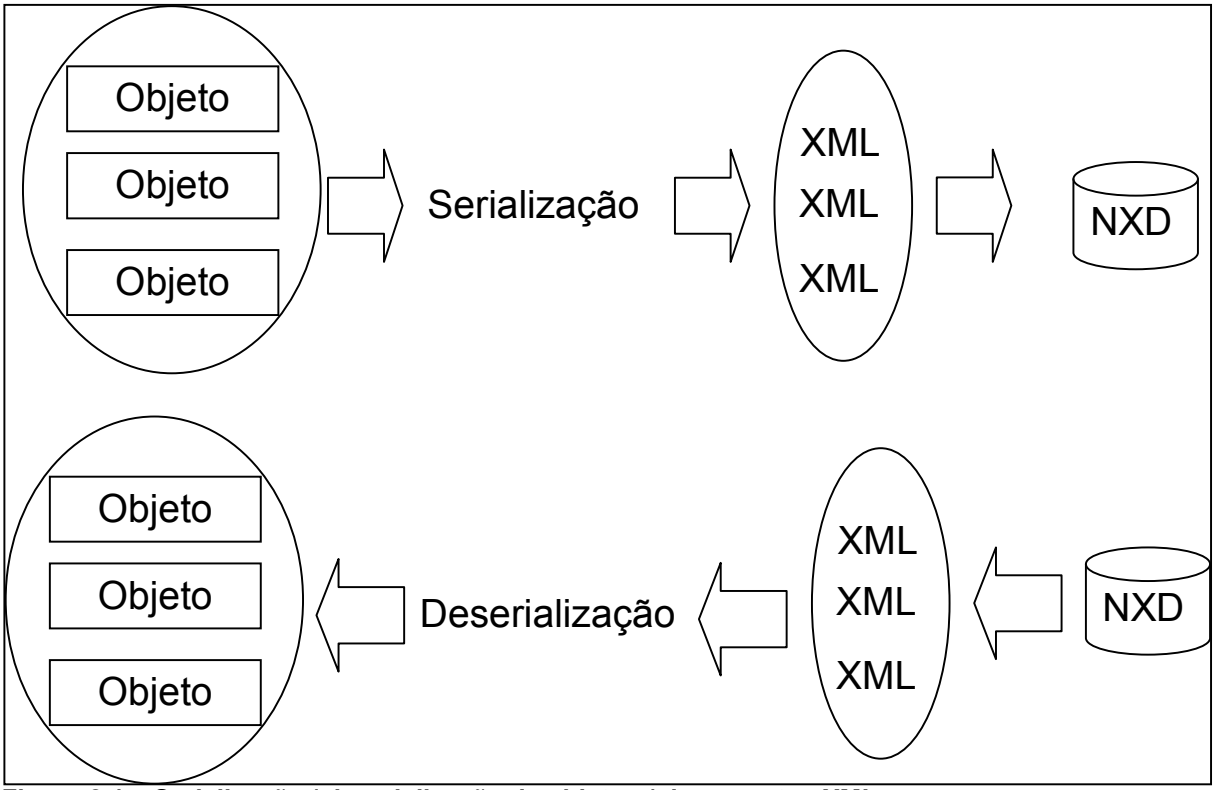

Figura 2.1 – Serialização / deserialização de objetos / documentos XML

A serialização/deserialização deve ser de forma tal que documentos XML bem-estruturados sejam gerados e consumidos pelo banco de dados NXD. Esse processo deve ser escalável para que a performance de consultas, inserções e atualizações de arquivos não seja comprometida, principalmente levando em consideração o tamanho dos documentos ou o número de documentos retornados por uma consulta.

3. Ferramentas e APIs

No momento de projeto, tivemos de eleger a tecnologia de desenvolvimento, as APIs necessárias e o banco de dados a ser utilizado em nossos testes.

A tecnologia de desenvolvimento, como já citado em tópicos anteriores, é a tecnologia Java.

O banco de dados XML nativo eleito dentre os vários que achamos disponíveis no mercado, como:

- eXist-db (http://www.exist-db.org);
- dbXML (http://www.dbxml.org);
- Ozone/XML (http://www.ozone-db.org);
- Tamino (http://www.softwareag.com);
- X-Hive (http://www.xhive.com);
- Excelon (http://www.exceloncorp.com);

Atualmente já possuímos banco de dados nativos XML mais evluídos, até mesmo os que estão listados acima, mas na época de projeto do framework escolhemos entre os citados o eXist-db, por ser o melhor banco de dados XML open source.

Selecionamos também as APIs necessárias para realizar a comunicação entre o framework e o banco de dados, e para a própria inicialização dos componentes do framework. Abaixo é feita uma breve apresentação dessas ferramentas e APIs utilizadas.

3.1. eXist-db

O eXist-db, como mencionado, é um banco de dados XML nativo criado a partir de uma iniciativa open source da organização eXist fundada por Wolfgang Méier. É um

banco de dados que pode ser facilmente integrado a aplicação que processam documentos XML se adequando a diferentes cenários. É um banco de dados totalmente desenvolvido em Java e que provê o armazenamento de documentos em uma estrutura hierárquica, dentro do que é chamado de coleção. Fazendo utilização de tecnologias de acesso a documentos XML como XPath, é possível acessar dados nos documentos XML armazenados. O eXist-db provê uma interface cliente amigável que proporciona esses tipos de consultas. Aplicações, como o framework criado, realizam a interação com o eXist através de uma *application programming interface* (API) chamada XML:DB API.

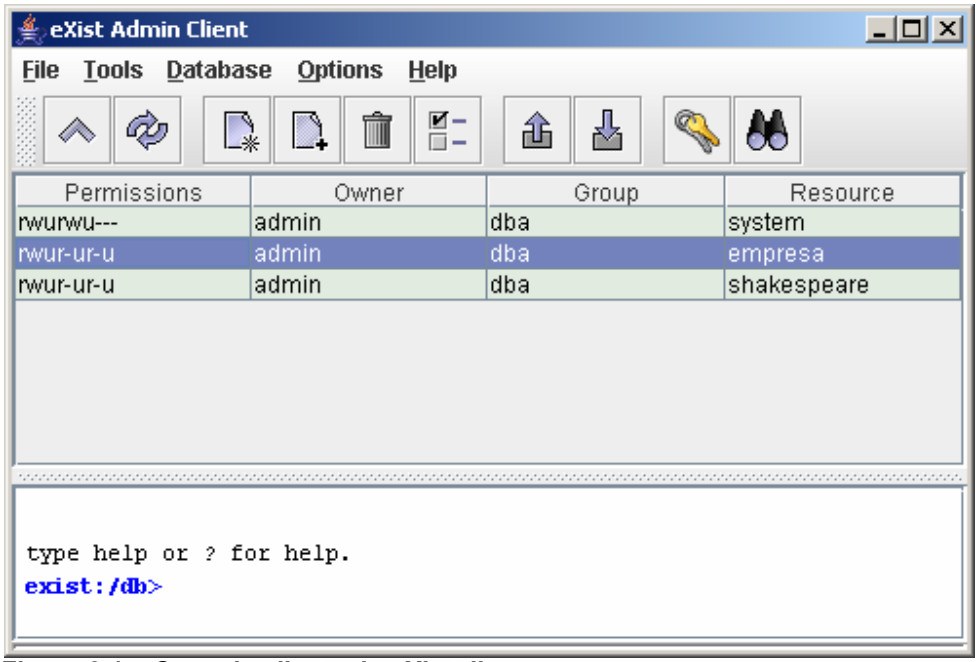

Figura 3.1 – Console cliente do eXist-db.

3.2. XML:DB API

Esta API é utilizada para acessar banco de dados XML e manipular seus dados. Surgiu a partir de uma iniciativa tomada pela organização XML:DB, formada por grupos simpatizantes da tecnologia XML. Visando a expansão dessa tecnologia resolveram criar algo que fosse difundido e ajudasse a mostrar esse conhecimento para as diversas comunidades tecnológicas existentes. É através dessa API que

nosso framework acessa o native XML database, no caso o eXist, e realiza as mais diversas operações.

3.3. XStream API

A Xstream API é formada por um conjunto de classes utilizadas para serializar e deserializar objetos em documentos XML. O framework possui um componente de serialização/deserializaçãi que se utiliza dessa API.

3.4. JAXB API

Java Architecture for XML Binding API possui um conjunto de classes que provêem formas de representar documentos XML em objetos Java. Essa API permite que, até mesmo, pessoas que não tenham conhecimento profundo sobre a tecnologia XML poderem incorporá-la a suas aplicações e manipular seus dados.

A partir de um arquivo de extensão XSD (XML Sechema Definition) é possível definir a estrutura de um documento XML. A JAXB API possui um componente para gerar as classes que irão processar os documentos XMLs que seguem a definição descrita no XSD. No caso do framework utilizamos o documento *nxd.xsd* para definir as classes que irão processar os arquivos que conterão as consultas a serem realizadas pelo framework e que são definidas pelo cliente, usuário do framework.

Abaixo temos um diagrama vizualizado pelo Eclipse (http:// www.eclipse.org) onde podemos identificar a estrutura do documento XSD e seus elementos.

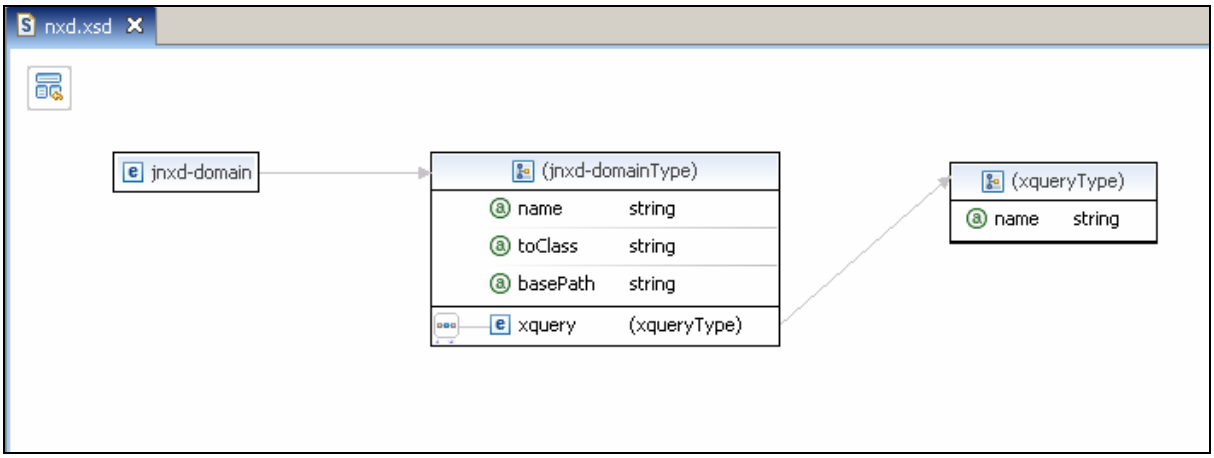

Figura 3.2 – Estrutura do documento jnxd.xsd

3.5. Spring Framework

Spring framework (http://www.springframework.org/) é mais um framework JEE que já tomou conta do mercado.

Spring provê uma maneira diferente para gerenciar objetos de negócio. É desenvolvido numa arquitetura em camadas sendo possível utilizar somente algumas de suas funcionalidades.

O JNXD Framework faz uso de algumas funcionalidades do Spring como, simplificar o uso do JDBC, configurando em seu arquivo de propriedades os dados necessários para acessar o banco de dados XML nativo que se desejar acessar.

Faz-se uso também do que chamamos de inversão de controle, que nada mais é do que a instanciação automática a partir de uma configuração simples no arquivo de propriedades do Spring.

Abaixo temos a definição do arquivo utilizado pelo NXD Framework para configuração de alguns objetos necessários:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean id="databaseFactory" class="org.nxd.config.spring.NXDDatabaseFactoryBean" singleton="true"/>
    <bean id="database" factory-bean="databaseFactory" class="org.nxd.config.NXDDatabaseConfig">
      <property name="driver">
         <value>org.exist.xmldb.DatabaseImpl</value>
      </property>
      <property name="uri">
         <value>xmldb:exist://localhost:8080/exist/xmlrpc/db</value>
      </property>
      <property name="login">
         <value>admin</value>
      </property>
      <property name="password">
         <value></value>
      </property>
    </bean>
    <bean id="xmlSerializer" class="org.nxd.serializer.DefaultXMLSerializer" lazy-init="true"/>
    <bean id="xmlBuilder" class="org.nxd.builder.NXDBuilder" lazy-init="true"/>
    <bean id="saxContentHandler" class="org.xml.sax.helpers.DefaultHandler" lazy-init="true"/>

    <bean id="configFactory" class="org.nxd.config.spring.NXDConfigFactoryBean" singleton="true"/>
    <bean id="config" factory-bean="configFactory" class="org.nxd.config.NXDConfig">
        <property name="nxds">
            <list>
            <value>Empresa.nxd.xml</value>
            <value>Funcionario.nxd.xml</value>
            </list>
        </property>
        <property name="serializer" ref="xmlSerializer"/>
        <property name="builder" ref="xmlBuilder"/>
        <property name="saxHandler" ref="saxContentHandler"></property>
    </bean>
</beans>
```

Figura 3.3 – Documento de definição do Spring (jnxd-applicationContext.xml)

4. Arquitetura

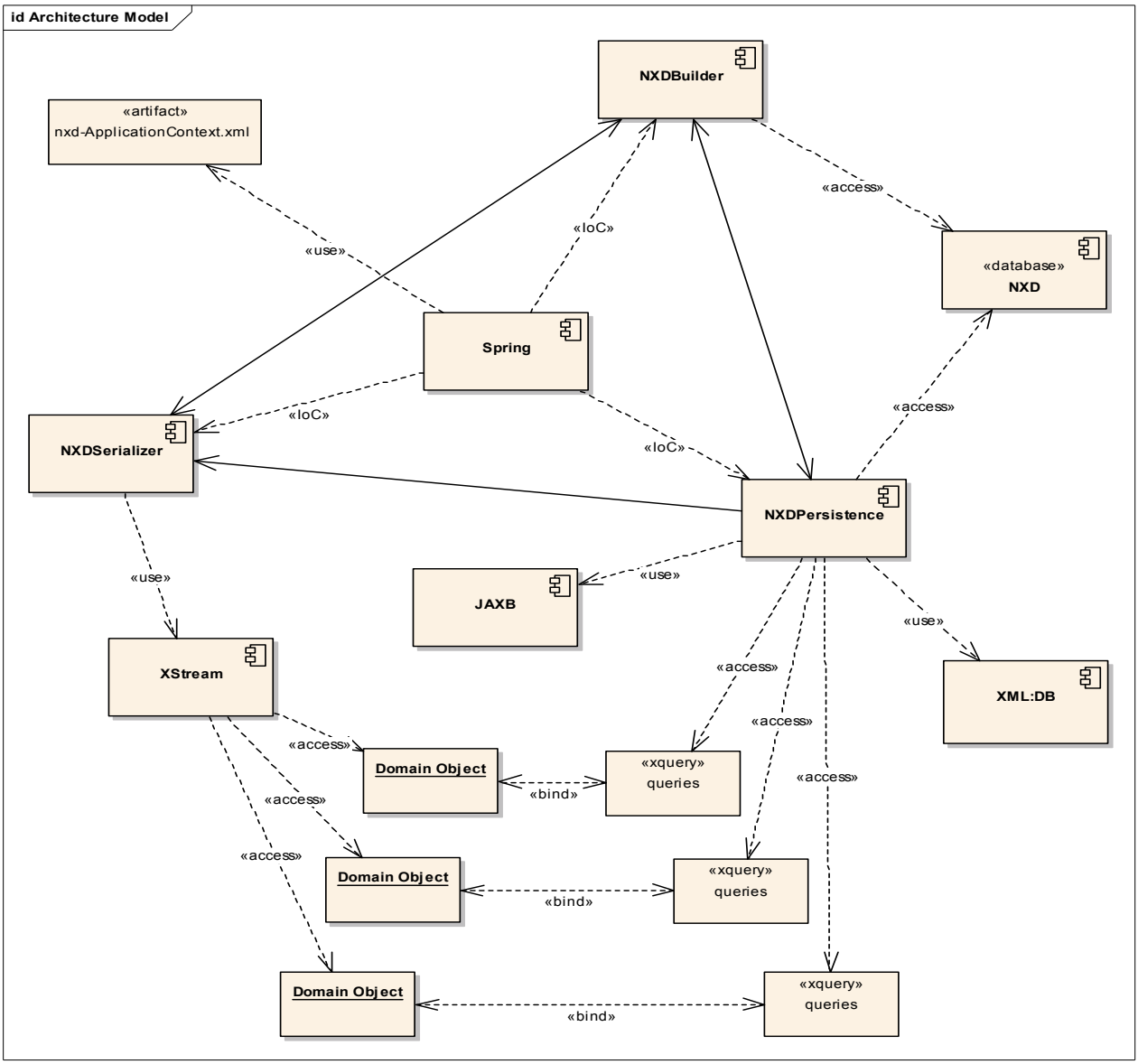

Figura 4.1 – Arquitetura do JNXD framework.

A arquitetura sugerida acima subdivide o framework em 3 componentes principais responsáveis por:

1. Serialização e deserialização de objetos Java e XML(JNXDSerializer),

2. Gerenciador da estrutura hierárquica do modelo orientado a objetos para o modelo físico no banco NXD (JNXDBuilder).
3. Gerenciador de persistência de documentos. (JNXPersistence)

5. Componentes

O NXD Framework é formado por três componentes principais, como é definido abaixo no diagrama de componentes:

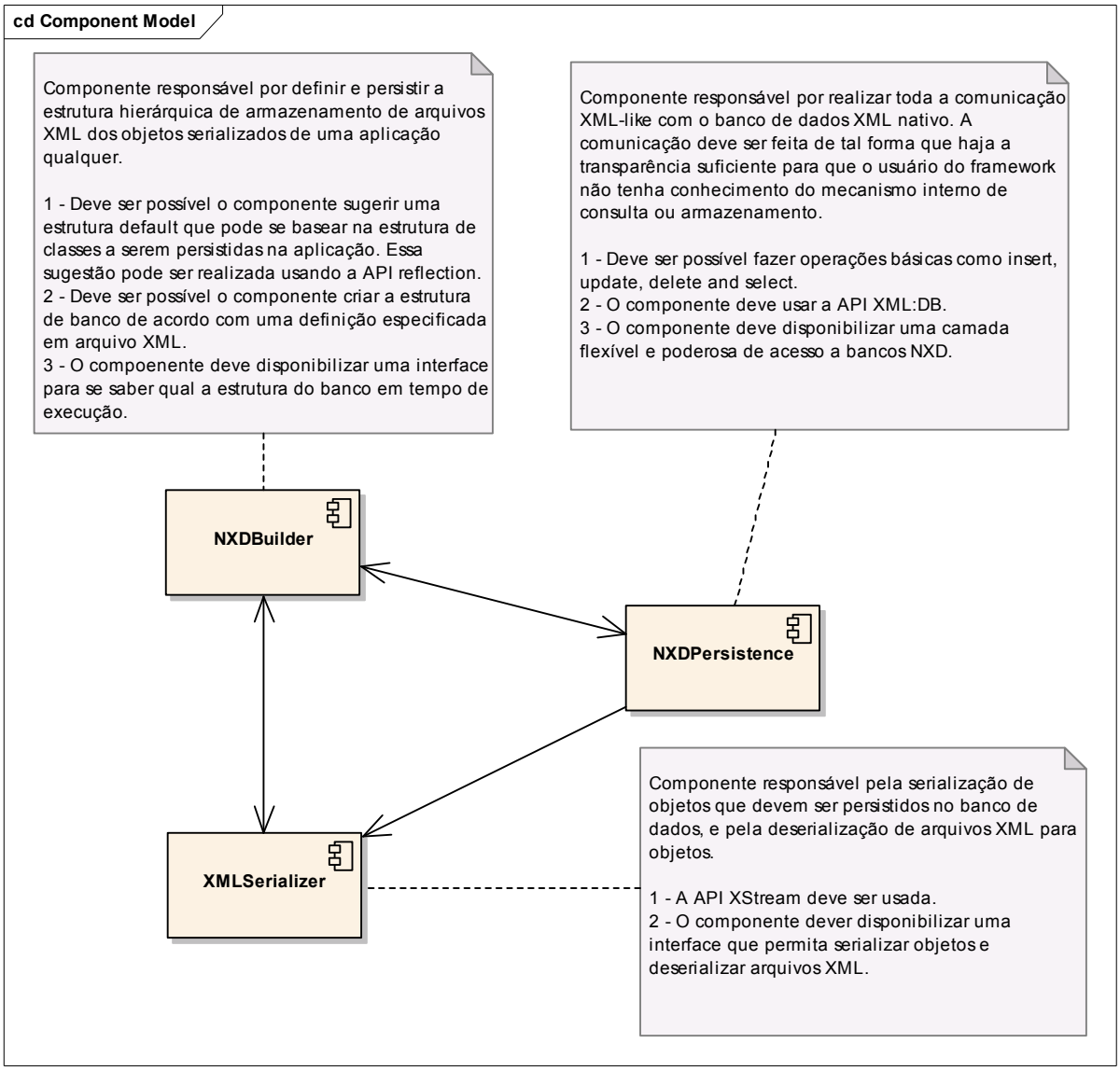

Figura 5.1 – Componentes do JNXD framework.

Como apresentado na introdução, o foco dado nesse trabalho é no terceiro componente, JNXD Builder, desenvolvido por mim.

5.1. JNXD Builder

NXD Builder é o componente responsável por informar ao framework detalhes sobre a estrutura do banco de dados XML nativo, afim de que se possam realizar as operações necessárias, como, pesquisas, inserções, alterações e remoções. Ainda há um componente que é executado no momento do carregamento do framework utilizado para criar a estrutura do banco de dados automaticamente, e será apresentado depois com mais detalhes. Segue a estrutura de classes do JNXBuilder:

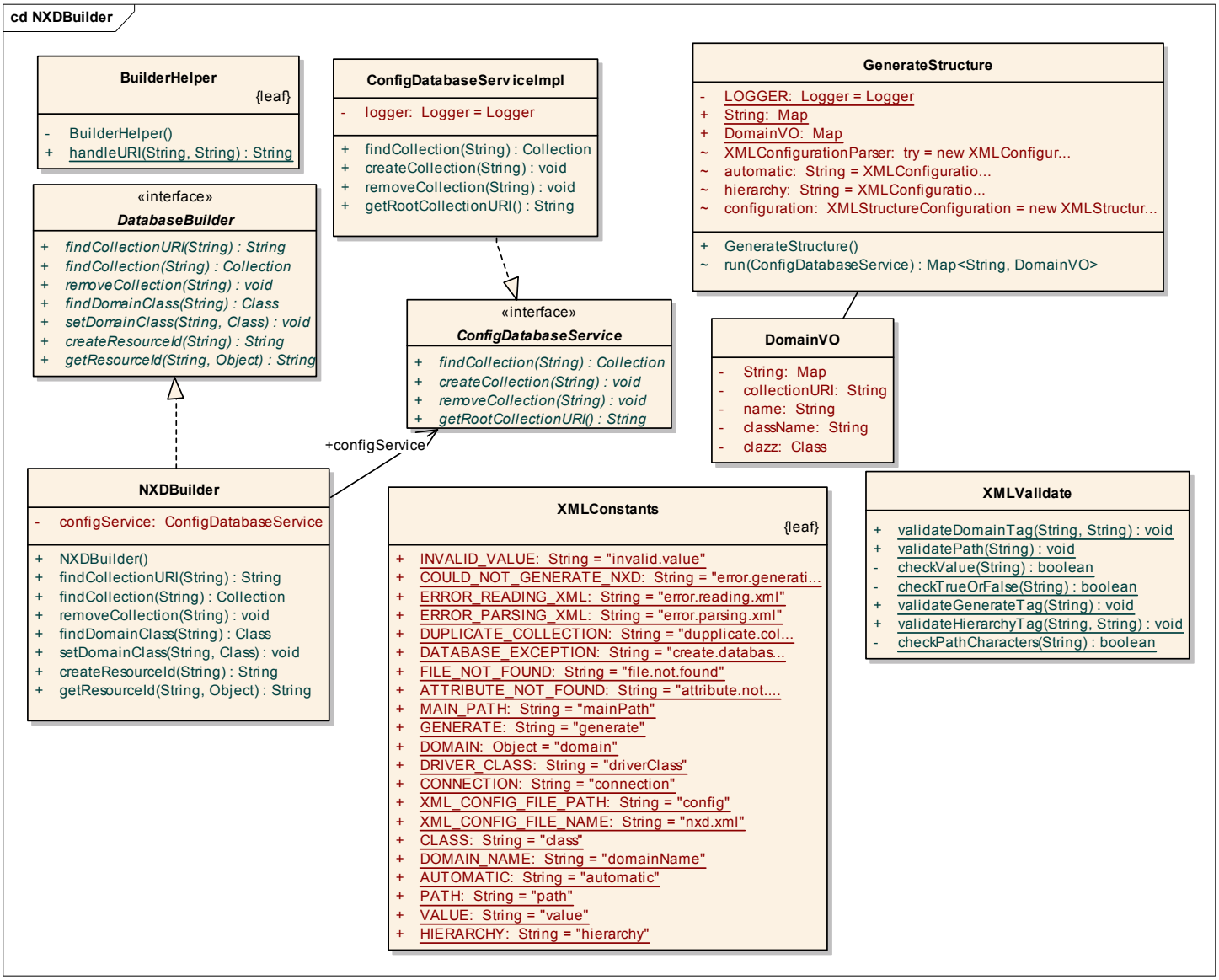

Figura 5.2 – Diagrama de classes do componente JNXDBuilder.

Esse componente faz uso de alguns padõres de projeto como:

- Singleton
- AbstractFactory
- Template Method
- Builder

A classe GenerateStructure.java é invocada somente quando a aplicação cliente é inicializada, pois ela é responsável por gerar a estrutura do banco de dados automaticamente, caso o cliente opte por deixar isso a cargo do framework. A configuração para setar essa opção é mostrada mais abaixo.

A classe NXDBuilder, hoje renomeada para JNXDBuilder, é a principal desse componente. Ela possui métodos que são invocados e utilizados para realizar operações no banco de dados, como busca de dados, inserções, alterações e remoções. É o componente responsável por invocar operações do componente JNXDPersistence que possui as operações mais diretas de acesso ao banco.

6. Configuração da geração automática da estrutura NXD.

Para a utilização da funcionalidade de geração automática da estrutura NXD a partir do JNXD, basta realizar a configuração em um documento XML: o documento *nxd.xml*.

```
nxd.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE config PUBLIC "-//NXD//DTD BEAN//EN" "NXDPersistence/java/config/nxd.dtd">
3
4 <!-- NXD Framework configuration file. -->
5
6 <config>
7     <database>
8         <generate automatic="true"/>
9         <hierarchy automatic="true"/>
10    </database>
11    <domains>
12        <domain class="org.nxd.domains.Empresa" domainName="empresa"/>
13        <domain class="org.nxd.domains.Departamento" domainName="departamento"/>
14        <domain class="org.nxd.domains.Funcionario" domainName="funcionario"/>
15        <domain class="org.nxd.domains.Dependente" domainName="dependente"/>
16    </domains>
17 </config>
```

Figura 6.1 – Documento nxd.xml

Esse documento se inicia com as tag <config> que serve para manter uma certa organização do documento. As tags <database> e <domains> são utilizadas para separar configurações para fins distintos, porém, necessárias para a geração da estrutura do banco de dados e inicialização de alguns objetos do framework.

A tag <database> é formada por mais duas outras tags que especificam se o banco deve ser ou não gerado automaticamente, ou se o banco deve ser gerado em forma hierárquica ou se deve possuir somente a coleção raiz que irá armazenar os XMLs da aplicação. Caso essa segunda opção for escolhida se faz necessário informar um atributo chamado “mainPath” informando qual o nome da coleção raiz que deverá ser gerada.

A tag <domains> agrega as tags que definem os domínios que estarão presentes na aplicação que deverá fazer parte do banco de dados XML. A tag <domain> possui atributos como:

1. *class:* Utilizado para definir a classe associada ao objeto que representará o domínio especificado.
2. *domainName:* Atributo que irá definir o nome da coleção a ser conhecida pelo framework.

3. *path*: Caso o banco de dados não seja gerado automaticamente será necessário informar o path das coleções existentes na estrutura do banco de dados já existentes.

A nova versão do framework que está sendo desenvolvida já deixa bem mais claro que esse documento XML só é necessário para a geração automática do banco de dados, pois a versão atual ainda faz uso de alguns elementos ainda aí definidos caso a estrutura automática não seja solicitada pelo cliente.

O JNXDBuilder é responsável também por informar ao componente JNXDPersistence qual as consultas a serem executadas. Essas consultas são mapeadas em documentos XML que seguem a estrutura definida pelo documento jnxd.xsd apresentado em tópicos anteriores. Esse documentos tem a estrutura seguida pelo exemplo abaixo:

Empresa.jnxd.xml

```
1<?xml version="1.0" encoding="UTF-8"?>
2<jnxd-domain name="empresa" toClass="org.nxd.domains.Empresa">
3    <xquery name="findByName">
4    //org.nxd.domains.Empresa
5    </xquery>
6</jnxd-domain>
```

Figura 6.2 – Exemplo de documento de mapeamento de cosultas.

O documento acima apresenta na sua tag raiz o nome do domínio associado ao documento. No caso, *"empresa"*, esse é o mesmo nome definido no documento *nxd.xml* que apresenta as configurações referentes aos domínios. Um atributo novo apresentado documento XSD acima é o *pathCollection*. Ele vai indicar qual a coleção raiz onde essas consultas serão realizadas. Essa modificação vem junto com a alteração que permite desacoplar o uso do XML de geração automática da estrutura do banco de dados. Essa funcionalidade também está sendo desenvolvida para a versão 2.0 do framework.

A tag <*xquey*> possui um atributo *name* que é utilizado para nome a operação a ser realizada, essa será a chave de pesquisa da consulta no documento, portanto não é possível configurar duas consultas para o mesmo nome.

Como foi mostrado, a utilização do framework é bastante simples e de fácil utilização.

Esse componente não é muito útil sem a interação entre os outros componentes definidos pelo framework. Nas referências desse projeto apontamos os trabalhos desenvolvidos pelos membros da esquipe responsáveis por esses componentes.

7. JNXD no java.net

O java.net, resumindo, é um repositório de idéias, projetos inovadores e discussões relacionados à tecnologia Java. Idústrias provedoras de softwares, uniersidades e vários grupos de engenheiros, desenolvedores e pesquisadores buscam compartilhar suas idéias e projetos a fim de apresentar soluções a problemas que outros ainda não conheçam.

No java.net qualquer um pode se registrar em qualquer projeto e fazer parte de uma equipe de desenvolvimento dando sua contribuição. O JNXD foi incluído no java.net e já possui novos colaboradores. Estamos buscando incluir novas funcionalidades e realizar melhorias nas funcionalidades já existentes, como as que já foram apresentadas em tópicos anteriores.

Outras funcionalidades a serem implementadas são:

- Suporte à API XQJ(IBM, Oracle).
- Realização de testes em outros bancos NXDs com suporte a XQJ.
- Incorporação de mecanismos de suporte a Web Services(mensagens SOAP).
- Integração com Web Frameworks(Struts, JSF RI, JSF MyFaces, Ajax, Sean).

Nesse projeto está em anexo o código fonte do componente desenvolvido, mas o código fonte de todo o projeto está incluído no site do JNXD no java.net (https://jnxd.dev.java.net/). O proejto utiliza o Subversion control (http://subversion.tigris.org/) como ferramenta de controle de versão.

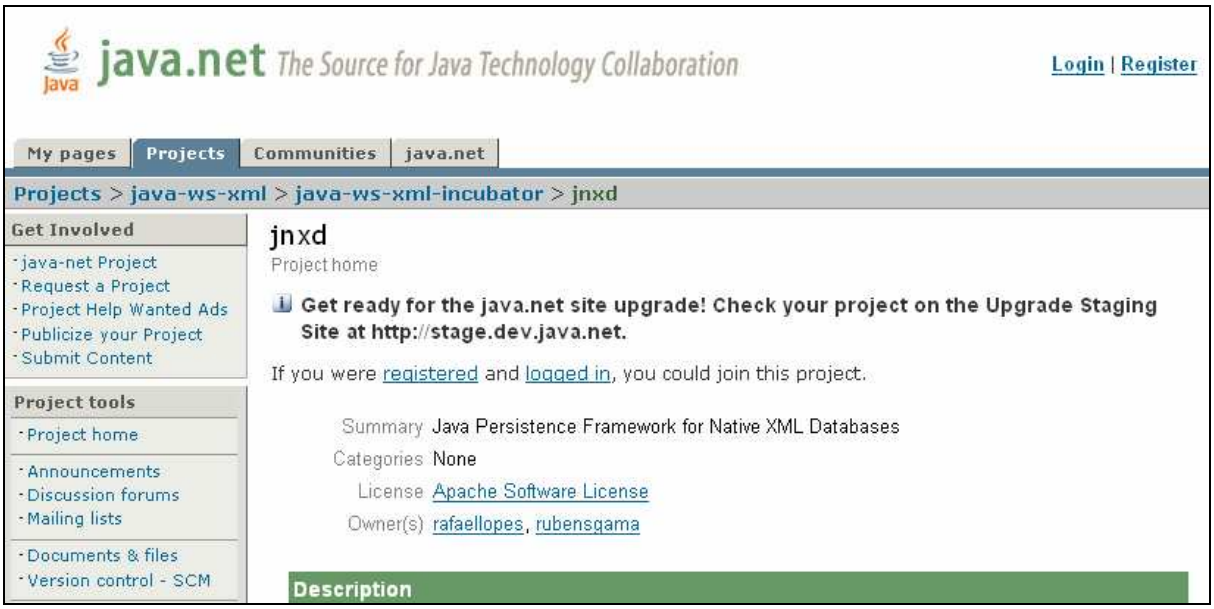

Figura 7.1 – JNXD no java.net

8. Conclusão

A tecnologia XML é uma tecnologia que já está bem difundida no mercado. Um ponto onde comprovamos isso pelo fato de encontrarmos várias APIs que fazem o tratamento de documento XML. A criação de bancos de dados XML vem crescendo na comunidade tecnológica, mas ainda não está sendo a opção número 1 para utilização dos diversos tipos de aplicações. São bancos de dados simples e de fácil utilização. Os mais robustos como o Tamino já vem aparecendo mais no mercado, fazendo com que as comunidades tecnológicas enxerguem cada vez mais o potencial dessa tecnologia e acreditem na mesma.

Outros projetos parecidos com o JNXD estão sendo desenvolvidos, como o XQuery Object Mapping (https://xqom.dev.java.net/) que procura utilizar o mapeamento entre o banco de dados relacional e seus onbjetos de forma a retornar para a aplicação cliente um documento XML estruturado, fazendo uso da API XQquery for Java.

A primeira fase do desafio já foi vencida, e daqui pra frente resta evoluir a ferramenta, para quem sabe um dia, ter um framework de fácil utilização para os propósitos definidos pelo JNXD.

9. Referências

- **eXist-db** (http://exist.sourceforge.net/)
- **Spring framework** (http://www.springframework.org/)
- **XML:DB API** (http://xmldb-org.sourceforge.net/xapi/)
- **JAXB API** (http://java.sun.com/webservices/jaxb/)
- **Anatomy of a Native XML Database** (http://www.sleepycat.com/pdfs/wp_xmldb_1104.pdf)
- **An Introduction to the XML:DB API** (http://www.xml.com/lpt/a/2002/01/09/xmldb_api.html)
- **Introduction to dbXML** (http://www.xml.com/lpt/a/2001/11/28/dbxml.html)
- **Introduction to Native XML Databases** (http://www.xml.com/lpt/a/2001/10/31/nativexmldb.html)
- **Generating XML Documents from Databases** (http://www.informit.com/articles/article.asp?p=30081&rl=1)
- **XQuery Object Mapping Framework** (https://xqom.dev.java.net)
- **XStream** (http://xstream.codehaus.org)

10. Código Fonte JNXD Builder

O código fonte apresentado refere-se somente ao componente JNXDBuilder.

BuilderHelper.java

```
package org.nxd.builder;

/**
 * Helper class whiach contains utilitary methods used by Builder component.
 * @author Rafael Lopes (rafaelblopes.br@gmail.com)
 */
public final class BuilderHelper {
    /**
     * Private constructor.
     */
    private BuilderHelper() {
    }
    /**
     * Concatenates a URI with a domain.
     * This method is related specifically to builder component.
     * @param uri URI of a collection.
     * @param domain Domain name.
     * @return URI of a domain.
     */
    public static String handleURI(final String uri, final String domain) {
        String uriUpdate = uri;
        if (!uri.endsWith("/") && !uri.endsWith("\\")) {
            uriUpdate += "/" + domain;
        } else {
            uriUpdate += domain;
        }
        return uriUpdate;
    }
}
```

DatabaseBuilder.java

```
package org.nxd.builder;

import org.nxd.exception.DomainException;
import org.nxd.exception.DomainNotFoundException;
import org.xmldb.api.base.Collection;

/**
```

```
 * An NXD has a URI which self identifies. In this case, it's root URI. For
 * example:
 *
 * -corporation -departament -emplotee -dependent This database has
 * "/corporation" as root URI. We can say that each node in hierarchy is a
 * domain where can store XML files. A domain is identified by a name. For
 * example: "corporation", or "emplyee". Below of "emplyee" domain, it can store
 * XML files which can refer to an "employee", obviously. We can say that a
 * domain has a unique URI, and a value object that represent it. For example,
 * the "emplyee" domain has the URI "/corporation/department/employee" and it
 * has "br.com.application.EmployeeTO" as a domain class.
 *
 * This service allows to get informations of collections of an NXD accordingly
 * the specified domain.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 */
public interface DatabaseBuilder {
    /**
     * Get the URI of a domain.
     *
     * @param domain
     *          Domain.
     * @return URI of collection.
     * @throws DomainNotFoundException Domain was not found.
     */
    String findCollectionURI(String domain) throws DomainNotFoundException;

    /**
     * Get the collection of domain.
     *
     * @param domain
     *          Domain.
     * @return Collection.
     * @throws DomainNotFoundException Domain was not found.
     */
    Collection findCollection(String domain) throws DomainNotFoundException;

    /**
     * Removes a collection by domain.
     *
     * @param domain
     *          Domain;
     * @throws DomainNotFoundException Domain was not found.
     */
    void removeCollection(String domain) throws DomainNotFoundException;
```

```
    /**
     * Gets the class of a domain.
     *
     * @param domain
     *          Domain.
     * @return Class of domain.
     * @throws DomainNotFoundException Domain was not found.
     */
    Class findDomainClass(String domain) throws DomainNotFoundException;

    /**
     * Configures a class for a domain.
     *
     * @param domain Domain name.
     * @param domainClass Domain class.
     * @throws DomainNotFoundException Domain was not found.
     */
    void setDomainClass(String domain, Class domainClass)
        throws DomainNotFoundException;

    /**
     * Creates a resource in a domain.
     *
     * @param domain
     *          Domain.
     * @return Unique identifier.
     * @throws DomainNotFoundException If domain was not found.
     * @throws DomainException Error creating domain identifier.
     */
    String createResourceId(String domain)
        throws DomainException, DomainNotFoundException;

    /**
     * Gets a resource identifier by domain.
     *
     * @param domain
     *          Domain name.
     * @param obj
     *          Objeto.
     * @return Unique identifier.
     * @throws DomainException Error getting domain identifier.
     * @throws DomainNotFoundException Domain was not found.
     */
    String getResourceId(String domain, Object obj)
        throws DomainNotFoundException, DomainException;
}
```

NXDBuilder.java

```
package org.nxd.builder;

import static org.nxd.config.NXDLoader.domainsMap;

import org.nxd.ConfigDatabaseService;
import org.nxd.config.NXDFactoryConfig;
import org.nxd.config.NXDLoader;
import org.nxd.exception.CollectionException;
import org.nxd.exception.CollectionNotFoundException;
import org.nxd.exception.DomainException;
import org.nxd.exception.DomainNotFoundException;
import org.nxd.exception.XQueryException;
import org.nxd.model.DomainVO;
import org.nxd.persistence.database.manager.xquery.XQueryDatabaseManager;
import org.xmldb.api.base.Collection;
import org.xmldb.api.base.Resource;
import org.xmldb.api.base.ResourceIterator;
import org.xmldb.api.base.ResourceSet;
import org.xmldb.api.base.XMLDBException;

/**
 * Creates the hierarchy necessary to support the XML store in an NXD.
 * Reflection and introspection is used to investigate a list of transfer
 * objects to suggest.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 */
public class NXDBuilder implements DatabaseBuilder {
    /**
     * Reference of a service that configures the database.
     */
    private ConfigDatabaseService configService;

    /**
     * Default constructor.
     */
    public NXDBuilder() {
        configService = NXDFactoryConfig.getConfigService();
    }

        /**
     * @see org.nxd.builder.DatabaseBuilder#findCollectionURI(java.lang.String)
     * @param domain
     *          Domain name.
```

```
 * @return URI of a collection.
 * @throws DomainNotFoundException If collection was not found.
 */
public final String findCollectionURI(final String domain)
        throws DomainNotFoundException {
    try {
        String uri = configService.getRootCollectionURI();
        DomainVO domainVO = domainsMap.get(domain);
        return BuilderHelper.handleURI(uri, domainVO.getCollectionURI());
    } catch (CollectionNotFoundException e) {
        throw new DomainNotFoundException(e);
    }
}

/**
 * @see org.nxd.builder.DatabaseBuilder#findCollection(java.lang.String)
 * @param domain
 *          Domain name.
 * @return Collection in database.
 * @throws DomainNotFoundException If domain was not found.
 */
public final Collection findCollection(final String domain)
        throws DomainNotFoundException {
    try {
        String collectionURI = findCollectionURI(domain);
        return configService.findCollection(collectionURI);
    } catch (CollectionNotFoundException e) {
        throw new DomainNotFoundException(e);
    }
}

/**
 * @see org.nxd.builder.DatabaseBuilder#removeCollection(java.lang.String)
 * @param domain
 *          Domain name.
 * @throws DomainNotFoundException If domain was not found.
 */
public final void removeCollection(final String domain)
        throws DomainNotFoundException {
    try {
        String uri = configService.getRootCollectionURI();
        DomainVO domainVO = domainsMap.get(domain);
        uri = BuilderHelper.handleURI(uri, domainVO.getCollectionURI());
        configService.removeCollection(uri);
    } catch (CollectionNotFoundException e) {
        throw new DomainNotFoundException(e);
    } catch (CollectionException e) {
```

```
        throw new DomainNotFoundException(e);
    }
}

/**
 * @see org.nxd.builder.DatabaseBuilder#findDomainClass(java.lang.String)
 * @param domain Domain name.
 * @return Class of domain.
 */
public final Class findDomainClass(final String domain) {
      DomainVO domainVO = NXDLoader.domainsMap.get(domain);
    return domainVO.getClassObject();
}

/**
 * @see org.nxd.builder.DatabaseBuilder#setDomainClass(java.lang.String,
 *      java.lang.Class)
 * @param domain Domain name.
 * @param domainClass Domain class.
 */
public final void setDomainClass(final String domain,
        final Class domainClass) {
    //empty
      if (NXDLoader.domainsMap.containsKey(domain)){
                DomainVO domainVO = domainsMap.get(domain);
                domainVO.setClassObject(domainClass);
      }
}
/**
 * @see org.nxd.builder.DatabaseBuilder#createResourceId(java.lang.String)
 * @param domain
 *          Domain name.
 * @return Resource identifier.
 * @throws DomainException Error creating resource identifier of this
 * domain.
 * @throws DomainNotFoundException If domain is not found.
 */
public final String createResourceId(final String domain)
        throws DomainException, DomainNotFoundException {
    try {
        String uri = configService.getRootCollectionURI();
        DomainVO domainVO = domainsMap.get(domain);
        uri = BuilderHelper.handleURI(uri, domainVO.getCollectionURI());
        Collection collection = configService.findCollection(uri);
        return collection.createId();
    } catch (CollectionNotFoundException e) {
        throw new DomainNotFoundException(e);
```

```
        } catch (XMLDBException e) {
            throw new DomainException(e);
        }
    }

    /**
     * @see org.nxd.builder.DatabaseBuilder#getResourceId(java.lang.String,
     *      java.lang.Object)
     * @param domain
     *          Domain name.
     * @param obj
     *          Object.
     * @return Resource identifier.
     * @throws DomainNotFoundException Collection was not found.
     * @throws DomainException XML handling error.
     */
    public final String getResourceId(final String domain, final Object obj)
            throws DomainNotFoundException, DomainException {
        try {
            String resourceId = null;
            String uri = configService.getRootCollectionURI();
            DomainVO domainVO = domainsMap.get(domain);
            uri = BuilderHelper.handleURI(uri, domainVO
                                        .getCollectionURI());
            XQueryDatabaseManager xqueryManager =
                NXDFactoryConfig.getContext().getXqueryManager();
            xqueryManager.setNamespace(domain, uri);
            ResourceSet set = xqueryManager.query("//.");
            ResourceIterator iterator = set.getIterator();
            while (iterator.hasMoreResources()) {
                                Resource resource = iterator.nextResource();
                                resourceId = resource.getId();
                                break;
                        }
            return resourceId;
        } catch (CollectionNotFoundException e) {
            throw new DomainNotFoundException(e);
        } catch (XQueryException e) {
            throw new DomainException(e);
        } catch (XMLDBException e) {
            throw new DomainException(e);
        }
    }
}
```

ConfigDatabaseService.java

```
package org.nxd;

import org.nxd.exception.CollectionException;
import org.nxd.exception.CollectionNotFoundException;
import org.nxd.exception.DuplicateCollectionException;
import org.xmldb.api.base.Collection;

/**
 * Collection is a structure of a database which can be store XML files.
 * An NXD has a root collection which contains all others child collections.
 * Each structure, or root or child collection, defines a new database. For
 * example: -corporation -department -employee - dependent. We can say that
 * this database has the URI "/corporation" as root collection. We can say too
 * that "/corporation/department/emplyee" is an URI of emplyee collection. This
 * service creates, searches and removes collections from an NXD database.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 */
public interface ConfigDatabaseService {
    /**
     * Gets a collection through a URI.
     * @param uri URI of collection.
     * @return Collection of database.
     * @throws CollectionNotFoundException If collection was not found.
     */
    Collection findCollection(String uri) throws CollectionNotFoundException;

    /**
     * Creates a collection in database using URI specified.
     *
     * @param uri URI of collection.
     * @throws DuplicateCollectionException If already exists this collection
     * in database.
     * @throws CollectionException Error acessing a collection.
     */
    void createCollection(String uri) throws DuplicateCollectionException,
        CollectionException;

    /**
     * Removes a collection of database.
     *
     * @param uri URI of collection.
     * @throws CollectionNotFoundException If collection was not found.
     * @throws CollectionException Error acessing a collection.
     */
```

```
    void removeCollection(String uri) throws CollectionNotFoundException,
        CollectionException;

    /**
     * Gets the URI of root collection.
     * @return URI of root collection.
     * @throws CollectionNotFoundException If this collection was not found.
     */
    String getRootCollectionURI() throws CollectionNotFoundException;
}
```

```
ConfigDatabaseServiceImpl.java

package org.nxd;

import org.apache.log4j.Logger;
import org.nxd.config.NXDDatabaseConfig;
import org.nxd.config.NXDFactoryConfig;
import org.nxd.config.NXDLoader;
import org.nxd.exception.CollectionException;
import org.nxd.exception.CollectionNotFoundException;
import org.nxd.exception.DuplicateCollectionException;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.base.Collection;
import org.xmldb.api.base.XMLDBException;

/**
 * Interface used to manage collections of an NXD database.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 */
public class ConfigDatabaseServiceImpl implements ConfigDatabaseService {
    /**
     * Logger.
     */
    private final Logger logger = Logger
            .getLogger(ConfigDatabaseServiceImpl.class);

    /**
     * @see org.nxd.ConfigDatabaseService#findCollection(java.lang.String)
     * @param uri
     *            Collection identifier.
     * @return Collection of an NXD database.
     * @throws CollectionNotFoundException
     *             Collection was not found.
     */
```

```
public final Collection findCollection(final String uri)
        throws CollectionNotFoundException {
    try {
        NXDDatabaseConfig databaseConfig = NXDLoader.getDatabase();
        return DatabaseManager.getCollection(uri, databaseConfig
                .getLogin(), databaseConfig.getPassword());
    } catch (XMLDBException e) {
        logger.fatal("[findCollection(String uri)]");
        throw new CollectionNotFoundException(e);
    }
}

/**
 * @see org.nxd.ConfigDatabaseService#createCollection(java.lang.String)
 * @param uri
 *          Collection identifier.
 * @throws CollectionException
 *          Error acessing collection.
 * @throws DuplicateCollectionException
 *          Collection already exists.
 */
public final void createCollection(final String uri)
        throws DuplicateCollectionException, CollectionException {
    try {
        try {
            findCollection(uri);
            throw new DuplicateCollectionException();
        } catch (CollectionNotFoundException e) {
            NXDFactoryConfig.getContext().getCollectionManager()
                    .createCollection(uri);
        }
    } catch (CollectionException e) {
        logger.fatal("[createCollection(String uri)]");
        throw new CollectionException(e);
    }
}

/**
 * @see org.nxd.ConfigDatabaseService#removeCollection(java.lang.String)
 * @param uri
 *          Collection identifier.
 * @throws CollectionException
 *          Error acessing collection.
 * @throws CollectionNotFoundException
 *          Collection was not found.
 */
public final void removeCollection(final String uri)
```

```
        throws CollectionNotFoundException, CollectionException {
    findCollection(uri);
    NXDFactoryConfig.getContext().getCollectionManager().removeCollection(
            uri);
  }

  /**
   * @see org.nxd.ConfigDatabaseService#getRootCollectionURI()
   * @return URI of root collection.
   * @throws CollectionNotFoundException If collection is not specified.
   */
  public final String getRootCollectionURI()
        throws CollectionNotFoundException {
    NXDDatabaseConfig database = NXDLoader.getDatabase();
    if (database == null) {
        throw new CollectionNotFoundException();
    }
    return database.getUri();
  }
}
```

```
XMLConstants.java

package org.nxd.util;

/**
 * This is a final class that contains the constants used in the component.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 * @since 04/04/2006
 *
 */
public final class XMLConstants {
  /**
   * <code>INVALID_VALUE</code> - Attribute must contain valid value.
   */
  public static final String INVALID_VALUE = "invalid.value";
  /**
   * <code>COULD_NOT_GENERATE_NXD</code> - Error generating NXD.
   */
  public static final String COULD_NOT_GENERATE_NXD = "error.generating.nxd";
  /**
   * <code>ERROR_READING_XML</code> - The file could not be read.
   */
  public static final String ERROR_READING_XML = "error.reading.xml";
```

```
    /**
     * <code>ERROR_PARSING_XML</code> - The XML could not be parsed
     */
    public static final String ERROR_PARSING_XML = "error.parsing.xml";
    /**
     * <code>DUPLICATE_COLLECTION</code> - When is creating a collection that
     * already exists.
     */
    public static final String DUPLICATE_COLLECTION = "dupplicate.collection";
        /**
         * <code>DATABASE_EXCEPTION</code> - Database structure could not be
         *  generated.
         */
        public static final String DATABASE_EXCEPTION =
"create.database.exception";
        /**
         * <code>FILE_NOT_FOUND</code> - Configuration file not found.
         */
        public static final String FILE_NOT_FOUND = "file.not.found";
    /**
     * <code>ATTRIBUTE_NOT_FOUND</code> - Attributes not found in XML config.
     *  file.
     */
    public static final String ATTRIBUTE_NOT_FOUND = "attribute.not.found";

    /**
     * <code>MAIN_PATH</code> - Attribute "mainPath" of the tag "hierarchy".
     */
    public static final String MAIN_PATH = "mainPath";
    /**
         * <code>GENERATE_DATABASE</code> - element "generate" of the
configuration
         *  file.
         */
        public static final String GENERATE = "generate";
        /**
         * <code>DOMAIN</code> -  element "domain" of the configuration file.
         */
        public static final Object DOMAIN = "domain";
        /**
         * <code>DRIVER_CLASS</code> - element "driverClass" of the configuration
         *  file.
         */
        public static final String DRIVER_CLASS = "driverClass";
        /**
         * <code>CONNECTION</code> - element "connection" of the configuration file.
         */
```

```
        public static final String CONNECTION = "connection";
        /**
         * <code>XML_CONFIG_FILE_PATH</code> - configuration file path.
         */
        public static final String XML_CONFIG_FILE_PATH = "config";
        /**
         * <code>XML_CONFIG_FILE_NAME</code> - configuration file name.
         */
        public static final String XML_CONFIG_FILE_NAME = "nxd.xml";
        /**
         * <code>CLASS</code> - attribute "class" of the element "domain".
         */
        public static final String CLASS = "class";
        /**
         * <code>DOMAIN_NAME</code> - attribute "domainName" of the element
"domain"
         */
        public static final String DOMAIN_NAME = "domainName";
        /**
         * <code>AUTOMATIC</code> - attribute "automatic" of the elements "generate"
    * and "hierarchy".
         */
        public static final String AUTOMATIC = "automatic";
    /**
     * <code>PATH</code> - attribute "path" of the element "domain".
     */
    public static final String PATH = "path";
    /**
     * <code>VALUE</code> - value attribute of the element driverClass.
     */
    public static final String VALUE = "value";
        /**
         * <code>HIERARCHY</code> - hierarchy attribute of the element generate.
         */
        public static final String HIERARCHY = "hierarchy";
}
```

```
XMLValidate.java

package org.nxd.util;

import org.nxd.config.XMLConfigurationParser;
import org.nxd.exception.XMLExceptionMessages;
import org.nxd.exception.XMLSyntaxException;
```

```
/**
 * This class contain methods to validate the xml file configuration data.
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 * @since 04/03/2006
 *
 */
public class XMLValidate {

    /**
     * @param clazz - Value of the attribute class
     * @param domain - Value of the attribute domainName
     * @throws XMLSyntaxException
     *                              If the attributes are not declared in the nxd.xml document.
     */
    public static void validateDomainTag(String clazz, String domain)
            throws XMLSyntaxException {

          String attributes = "";
        if (checkValue(clazz)) {
                        attributes = XMLConstants.CLASS;
        }
        if (checkValue(domain)) {
                        if (attributes.equals("")){
                                attributes = XMLConstants.DOMAIN_NAME;
                        }else{
                                attributes = attributes + ", " +
XMLConstants.DOMAIN_NAME;
                        }
        }
                if (!attributes.equals("")) {
                        throw new
XMLSyntaxException(XMLExceptionMessages.getMessage(
                                XMLConstants.ATTRIBUTE_NOT_FOUND,
                                new String[] { attributes }));
                }
    }

    /**
     * @param domainName
     *          The domainName to be found in the pathMap.
     * @throws XMLSyntaxException
     *          Se o(s) atributo(s) class e\ou domainName não forem
     *          declarados no XML de configuração.
     */
    public static void validatePath(String domainName)
                        throws XMLSyntaxException {
```

```
                if (!XMLConfigurationParser.pathMap.containsKey(domainName)) {
                        throw                                                   new
XMLSyntaxException(XMLExceptionMessages.getMessage(
                                        XMLConstants.ATTRIBUTE_NOT_FOUND,
                                        new String[] { XMLConstants.PATH }));
                } else if (checkPathCharacters(XMLConfigurationParser.pathMap
            .get(domainName))) {
         throw new XMLSyntaxException(XMLExceptionMessages.getMessage(
                XMLConstants.INVALID_VALUE,
                new String[] {XMLConfigurationParser.pathMap
                        .get(domainName), XMLConstants.MAIN_PATH }));
      }
        }

        /**
         * @param dataValue Value to be validated
         * @return The validation result
         */
        private static boolean checkValue(String dataValue) {
                if (dataValue == null || dataValue.equals("")){
                        return true;
                }
                return false;
        }

   /**
    * @param dataValue
    * @return
    */
   private static boolean checkTrueOrFalse(String dataValue) {
      if (dataValue != null &&
            (dataValue.equals("true") || dataValue.equals("false"))){
         return false;
      }
      return true;
   }

        /**
         * @param automatic Value of the attribute automatic
         * @throws XMLSyntaxException
    *                              If the attributes are not declared in the nxd.xml document.
         */
        public static void validateGenerateTag(String automatic)
                        throws XMLSyntaxException {
                if (checkValue(automatic)){
         throw new XMLSyntaxException(XMLExceptionMessages.getMessage(
                XMLConstants.ATTRIBUTE_NOT_FOUND,
```

```
                new String[] { XMLConstants.AUTOMATIC }));
                 } else if (checkTrueOrFalse(automatic)){
        throw new XMLSyntaxException(XMLExceptionMessages.getMessage(
                XMLConstants.INVALID_VALUE,
                new String[] { automatic, XMLConstants.AUTOMATIC }));
    }
      }
/**
 * @param automatic Value of the attribute automatic
 * @param mainPath Value of the attribute mainPath
 * @throws XMLSyntaxException
 *             If the attributes are not declared in the nxd.xml document.
 */
public static void validateHierarchyTag(String automatic, String mainPath)
        throws XMLSyntaxException {

    String attributes = "";
    if (checkValue(automatic)) {
        attributes = XMLConstants.AUTOMATIC;
    } else if (checkTrueOrFalse(automatic)){
        throw new XMLSyntaxException(XMLExceptionMessages.getMessage(
                XMLConstants.INVALID_VALUE,
                new String[] { automatic, XMLConstants.AUTOMATIC }));
    }
    if (automatic.equals("false") && mainPath != null) {
        if (checkPathCharacters(mainPath)) {
            throw new XMLSyntaxException(XMLExceptionMessages.getMessage(
                    XMLConstants.INVALID_VALUE, new String[] { mainPath,
                            XMLConstants.MAIN_PATH }));
        }
    }
    if (!attributes.equals("")) {
        throw new XMLSyntaxException(XMLExceptionMessages.getMessage(
                XMLConstants.ATTRIBUTE_NOT_FOUND,
                new String[] { attributes }));
    }
}

/**
 * It makes the validation of a String, checking if its characters are
 * between the following intervals: a - z, A - Z and "."
 *
 * @param path the path to be validated
 * @return boolean
 */
private static boolean checkPathCharacters(String path) {
    for (int i = 0; i < path.length(); i++) {
```

```
            int c = path.charAt(i);

            if (!((c >= 65) && (c <= 90)) && !((c >= 97) && (c <= 122))
                    && !(c == 46)) {
                return true;
            }
        }

        return false;
    }
}
```

GenerateStructure.java

```
package org.nxd.structure;

import java.util.Map;

import org.apache.log4j.Logger;
import org.nxd.ConfigDatabaseService;
import org.nxd.config.XMLConfigurationParser;
import org.nxd.config.XMLStructureConfiguration;
import org.nxd.exception.CollectionException;
import org.nxd.exception.CollectionNotFoundException;
import org.nxd.exception.DuplicateCollectionException;
import org.nxd.exception.XMLConfigurationException;
import org.nxd.exception.XMLExceptionMessages;
import org.nxd.exception.XMLFileNotFoundException;
import org.nxd.exception.XMLSyntaxException;
import org.nxd.model.DomainVO;
import org.nxd.util.XMLConstants;

/**
 * This class is the main class of the Generate Structure component.
 * It has the main method to be invoked and create the database structure as
 * informed in the nxd.xml document.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 * @since 04/04/2006
 *
 */
public class GenerateStructure {

    /**
     * <code>LOGGER</code>.
     */
```

```
private static final Logger LOGGER = Logger
        .getLogger(GenerateStructure.class);
      /**
       * The class default constructor.
       */
      public GenerateStructure() {
      }

      /**
       * The method main to be invoked and generate the database structure.
       * @param configDatabaseService Service to access the database.
       * @throws XMLFileNotFoundException -
 *             When the nxd.xml file is not found.
       * @throws XMLSyntaxException -
 *             When happens an XML sysntax error
       * @throws CollectionNotFoundException -
 *             When happens an Collection not found error
       * @throws CollectionException -
 *             When happens an collection error
       * @throws XMLConfigurationException -
 *             When there is an error related to the XML configuration
       */
      public static Map<String, DomainVO> run(
       ConfigDatabaseService configDatabaseService){

    Map<String, DomainVO> structureMap = null;

    try {
       // instantiates the main object that parser the nxd.xml document.
       XMLConfigurationParser xmlConfig = new XMLConfigurationParser();
       // parse the document
       xmlConfig.readXMLFile();
       /*
        * Retrieve the attribute that says to generate the structure
        * automatically.
        */
       String automatic = XMLConfigurationParser.parametersMap
              .get(XMLConstants.AUTOMATIC);
       String hierarchy = XMLConfigurationParser.parametersMap
              .get(XMLConstants.HIERARCHY);

       // instantiates the main object that generates the database
       // structure
       XMLStructureConfiguration configuration =
          new XMLStructureConfiguration();
       /*
        * Mount the structured map with the information set in the nxd.xml
```

```
         */
        structureMap = configuration.getStructuredMap(automatic, hierarchy);

        if (automatic.equals("true")) {
            for (DomainVO domainVO : structureMap.values()) {
                try {
                    if (hierarchy.equals("false")){
                        configDatabaseService
                        .createCollection(domainVO.getCollectionURI());
                        break;
                    }
                    configDatabaseService
                            .createCollection(domainVO.getCollectionURI());
                } catch (DuplicateCollectionException e) {
                    // Collection already exists, so continue creating the
                    //others.
                    LOGGER.debug(XMLExceptionMessages.getMessage(
                            XMLConstants.DUPLICATE_COLLECTION,
                            new String[] { domainVO.getCollectionURI() }));
                }
            }
        }
    } catch (Exception ex) {

LOGGER.error(XMLExceptionMessages.getMessage(XMLConstants.COULD_NOT_G
ENERATE_NXD), ex);
    }
                return structureMap;
      }

/*    public static void test(DomainVO domainVO) throws Exception{
      //initialize database driver
      Class cl = Class.forName("org.exist.xmldb.DatabaseImpl");
      String uri = "xmldb:exist://localhost:8080/exist/xmlrpc";
      Database database = (Database) cl.newInstance();
      DatabaseManager.registerDatabase(database);

      Collection root = DatabaseManager.getCollection(uri + "/db");
      CollectionManagementService mgtService = (CollectionManagementService)
          root.getService("CollectionManagementService", "1.0");
      mgtService.createCollection(domainVO.getCollectionURI());
   }

   public static void main(String[] args) {
      run(null);
   }*/
```

```
}
```

DomainVO.java

```
package org.nxd.model;

import java.util.Map;


/**
 * This class represents the domain.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 *
 */
public class DomainVO {

    /**
     * <code>propertyAliases</code> - Map with the alias related to the
     *  attributes of the domain Class.
     */
    private Map<String, String> propertiesAliases;
    /**
     * <code>collectionURI</code> - The base URI of the collection in database.
     */
    private String collectionURI;
        /**
         * <code>name</code> - The domain name.
         */
        private String name;
        /**
         * <code>className</code> - The domain class name
         */
    private String className;
        /**
         * <code>clazz</code> - The domain class object.
         */
        private Class clazz;
        /**
     * @return Returns the propertyAliases.
     */
    public Map<String, String> getPropertiesAliases() {
        return propertiesAliases;
    }
    /**
```

```
 * @param propertiesAliases The propertiesAliases to set.
 */
public void setPropertiesAliases(Map<String, String> propertiesAliases) {
    this.propertiesAliases = propertiesAliases;
}
/**
 * @return Returns the className.
 */
public String getClassName() {
    return className;
}
/**
 * @param className The className to set.
 */
public void setClassName(String className) {
    this.className = className;
}
/**
 * @return Returns the collectionURI.
 */
public String getCollectionURI() {
    return collectionURI;
}
/**
 * @param collectionURI The collectionURI to set.
 */
public void setCollectionURI(String collectionURI) {
    this.collectionURI = collectionURI;
}
    /**
     * @return Returns the class.
     */
    public Class getClassObject() {
            return clazz;
    }
    /**
     * @param clazz The class to set.
     */
    public void setClassObject(Class clazz) {
            this.clazz = clazz;
    }
    /**
     * @return Returns the name.
     */
    public String getName() {
            return name;
    }
```

```
	/**
	 * @param name The name to set.
	 */
	public void setName(String name) {
		this.name = name;
	}

}
```

DefaultTO.java

```
/**
 *
 */
package org.nxd.model;

import java.io.Serializable;

/**
 * @author Rafael Lopes
 *
 */
public class DefaultTO implements Serializable{

	/**
	 * <code>serialVersionUID</code> - long
	 */
	private static final long serialVersionUID = 717582585487487048L;
	/**
	 * <code>resourceId</code> - String
	 */
	private String resourceId;

	/**
	 * @return the resourceId
	 */
	public String getResourceId() {
		return resourceId;
	}

	/**
	 * @param resourceId the resourceId to set
	 */
	public void setResourceId(String resourceId) {
		this.resourceId = resourceId;
```

```
        }

}
```

XMLStructureConfiguration.java

```
package org.nxd.config;

import java.lang.reflect.Field;
import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;
import java.util.HashMap;
import java.util.Map;

import org.nxd.exception.XMLSyntaxException;
import org.nxd.model.DomainVO;
import org.nxd.util.XMLConstants;
import org.nxd.util.XMLValidate;

/**
 * This class is responsible to get the information in the nxd.xml file and
 * call the generate NXD structure.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 *
 */
public class XMLStructureConfiguration {

        private final String SERIAL_VERSION_UID = "serialVersionUID";
    /**
     * <code>structureMap</code> - Map<domainName, DomainVO> This map is filled
     * with the data set in the nxd.xml document.
     */
    private Map<String, DomainVO> structureMap =
        new HashMap<String, DomainVO>();
    /**
     * The class default constructor
     */
    public XMLStructureConfiguration() {

    }

    /**
     * Some class type come like "class br.com...." or "interface br.com....".
     * This method is used to remove the words "class " or "interface "
```

```
 * if they exists.
 *
 * @param field
 *          Field to extract the type name.
 * @return Type without words like "class " or "interface "
 */
private String getFieldTypeName(String field) {
   if (field.indexOf("class ") != -1) {
      field = field.substring(6, field.length());
   } else if (field.indexOf("interface ") != -1) {
      field = field.substring(10, field.length());
   }
   return field;
}

/**
 * Create a map with the domains and populated DomainVO objects with
 * domain name, domain path and the class related to it.
 *
 * @param automatic
 *          The parameter that specify if the database structure must be
 *          generate automatcally.
 * @param hierarchy
 *          The parameter that specify if the structure follow a hierarchy
 *          level, or if there's only one root structure for all domains.
 * @throws XMLSyntaxException
 *           If there is any error in the nxd.xml document syntax.
 */
public Map<String, DomainVO> getStructuredMap(String automatic,
      String hierarchy)
      throws XMLSyntaxException {

   String mainPath =
      XMLConfigurationParser.parametersMap.get(XMLConstants.MAIN_PATH);

   Map<String, String> propertiesAliases = new HashMap<String, String>();
     for (String className : XMLConfigurationParser.domainMap.keySet()) {
      DomainVO domainVO = null;
      boolean inStructureMap = false;
      try {
         String domainName = XMLConfigurationParser.domainMap
                                          .get(className);

         for (DomainVO vo : structureMap.values()) {
            if (vo.getClassName().equals(className)) {
               domainVO = vo;
               propertiesAliases = domainVO.getPropertiesAliases();
```

```
                inStructureMap = true;
            }
        }
        if (!inStructureMap) {
            domainVO = new DomainVO();
            domainVO.setPropertiesAliases(propertiesAliases);
            domainVO.setClassObject(Class.forName(className));
            domainVO.setClassName(className);
            domainVO.setName(domainName);
            domainVO.setCollectionURI(domainName);
            structureMap.put(domainName, domainVO);
        }

        if (automatic.equals("true")) {

            Class clazz = Class.forName(className);
            Field[] fields = clazz.getDeclaredFields();

            for (Field field : fields) {
             if (!field.getName().equalsIgnoreCase(
                                                    SERIAL_VERSION_UID)) {
                                                String                fieldType                =
getFieldTypeName(field.getType()
                                                            .toString());
                                                // if the field is not a collection
                                                if (XMLConfigurationParser.domainMap
                                                            .containsKey(fieldType)) {
                                                    for       (DomainVO       vo       :
structureMap.values()) {
                                                            if
(vo.getClassName().equals(fieldType)) {
                                                                    String    newPath    =
domainName + "/"
                                                                                +
vo.getCollectionURI();

        vo.setCollectionURI(newPath);
                                                            }
                                                    }
                                                } else {
                                                    analyzeGenericType(domainVO,
domainName, field,
                                                                    propertiesAliases);
                                                }
                                            }
            }
        } else {
```

```
                XMLValidate.validatePath(domainName);
                domainVO.setCollectionURI(
                        XMLConfigurationParser.pathMap.get(domainName));
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
    setMainPath(hierarchy, mainPath);
      return structureMap;
}

/**
 * This method set the main path for all the domains, because the database
 * structure must has only one collection.
 */
private void setMainPath(String hierarchy, String mainPath) {

    if (hierarchy.equals("false")){
        if (mainPath == null) {
            mainPath = "";
            // find and set the main path automatically.
            int dashCount = 0;
            // get the main path.
            for (DomainVO domainVO : structureMap.values()) {
                int dashCountAux = 0;
                String uri = domainVO.getCollectionURI();

                int index = uri.indexOf("/");
                if (index == -1) {
                    mainPath = domainVO.getCollectionURI();
                } else {
                    while (index != -1) {
                        dashCountAux++;
                        uri = uri.substring(index + 1);
                        index = uri.indexOf("/");
                    }
                    if ((dashCount == 0) || (dashCountAux < dashCount)) {
                        dashCount = dashCountAux;
                        mainPath = domainVO.getCollectionURI();
                    }
                }
            }
        }
        //set the main path for all elements in the map.
        for (DomainVO domainVO : structureMap.values()) {
            domainVO.setCollectionURI(mainPath);
```

```
        }
    }
}

/**
 * Analyze the generic field and set the new DomainVO reorganizing the paths
 * for the existing domains.
 * @param domainVO The DomainVO that is being created.
 * @param domainName The domainName to have its object created.
 * @param field The field to be analysed
 * @throws ClassNotFoundException
 *            Error when the field type is not recognized.
 */
private void analyzeGenericType(DomainVO domainVO, String domainName,
        Field field, Map<String, String> propertiesAliases)
        throws ClassNotFoundException {
    String fieldType;
    // if it is collection, get the Generic type.
    Type genericType = field.getGenericType();
    if (genericType instanceof ParameterizedType) {
        fieldType = getFieldTypeName(field.getType().toString());
        propertiesAliases.put(field.getName(), fieldType);
        ParameterizedType ptype = (ParameterizedType) genericType;
        Type[] genericParameters = ptype.getActualTypeArguments();
        if (genericParameters.length == 1) {
            Type genericParamType = genericParameters[0];
            fieldType = getFieldTypeName(genericParamType.toString());
            if (XMLConfigurationParser.domainMap.containsKey(fieldType)) {
                String newPath = "";
                DomainVO vo = structureMap.get(fieldType);
                if (vo != null) {
                    newPath = domainName + "/" + vo.getCollectionURI();
                    setNewPath(vo.getName(), newPath);
                }
                if (newPath.length() == 0) {
                    DomainVO newDomainVO = new DomainVO();

                    String newDomainName = XMLConfigurationParser.domainMap
                            .get(fieldType);

                    newPath = domainVO.getCollectionURI() + "/"
                            + newDomainName;
                    newDomainVO.setPropertiesAliases(new HashMap<String, String>());
                    newDomainVO.setClassObject(Class.forName(fieldType));
                    newDomainVO.setName(newDomainName);
                    newDomainVO.setClassName(fieldType);
                    newDomainVO.setCollectionURI(newPath);
```

```
                    structureMap.put(newDomainName, newDomainVO);
                }
            }
        }
    } else {
        fieldType = getFieldTypeName(field.getType().toString());
        propertiesAliases.put(field.getName(), fieldType);
    }
}

/**
 * This method reorganize the paths when generating the database structure
 * automatically.
 *
 * @param domainName
 *          The domain name of the domain to have its path reorganized.
 * @param newPath
 *          The new path to be set.
 *
 */
private void setNewPath(String domainName, String newPath) {
    for (DomainVO vo : structureMap.values()) {
        String path = vo.getCollectionURI();
        int index = path.indexOf(domainName);
        if (index != -1) {
            int indexDash = path.indexOf("/", index);
            if (index == 0) {
                if (indexDash != -1) {
                    String subPath = path.substring(indexDash + 1, path
                            .length());
                    vo.setCollectionURI(newPath + "/" + subPath);
                } else {
                    vo.setCollectionURI(newPath);
                }
            } else {
                if (indexDash != -1) {
                    String subPath = path.substring(0, indexDash);
                    vo.setCollectionURI(subPath + "/" + newPath);
                } else {
                    vo.setCollectionURI(newPath);
                }
            }
        }

    }
}
```

```
}
```

XMLConfigurationParser.java

```
package org.nxd.config;

import java.io.CharArrayWriter;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.apache.log4j.Logger;
import org.nxd.exception.XMLConfigurationException;
import org.nxd.exception.XMLExceptionMessages;
import org.nxd.exception.XMLFileNotFoundException;
import org.nxd.exception.XMLSyntaxException;
import org.nxd.util.XMLConstants;
import org.nxd.util.XMLValidate;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;


/**
 * This class read the nxd.xml configuration document and validate it.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
*/
public class XMLConfigurationParser {

    /**
     * <code>LOGGER</code>.
     */
    private static final Logger LOGGER = Logger
            .getLogger(XMLConfigurationParser.class);

            /**
```

```
         * The class default constructor
         */
        public XMLConfigurationParser() throws XMLFileNotFoundException{
      try {
                        URL url = getClass().getClassLoader().getResource(
                                    XMLConstants.XML_CONFIG_FILE_PATH + "/"
                                                +
XMLConstants.XML_CONFIG_FILE_NAME);
                        streamXmlFile = new FileInputStream(url.getPath());
                } catch (FileNotFoundException e) {
          String error = XMLExceptionMessages.getMessage(
                XMLConstants.FILE_NOT_FOUND,
                new String[] { XMLConstants.XML_CONFIG_FILE_PATH + "/"
                      + XMLConstants.XML_CONFIG_FILE_NAME });

          LOGGER.error(error, e);

                        throw new XMLFileNotFoundException(error, e);
                }
        }
        /**
         * <code>streamXmlFile</code> - FileInputStream object to represend the
         * nxd.xml document.
         */
        private FileInputStream streamXmlFile;
   /**
    * <code>contents</code> - Buffer that collect the SAX events.
    */
   private CharArrayWriter contents = new CharArrayWriter();
   /**
    * <code>domainMap</code> - Map<class, domainName> this map is filled with
    *  data specified in the nxd.xml document.
    */
   public static Map<String, String> domainMap = new HashMap<String, String>();

   /**
    * <code>pathMap</code> - Map<domainName, path> this map is filled with
    *  data specified in the nxd.xml document.
    */
   public static Map<String, String> pathMap = new HashMap<String, String>();

   /**
    *  <code>parametersMap</code>  -  Map<parameterName,  parameterValue>  this
map
    *  is filled with data specified in the nxd.xml document.
    */
   public static Map<String, String> parametersMap =
```

```
        new HashMap<String, String>();

    /**
     * Parsers the nxd.xml document.
     */
    public void readXMLFile() throws XMLConfigurationException{
        /*
         * Inner class to read the nxd.xml file.
         * By thos class we can implement anything we want to support the data
         * in the XML file.
         *
         */
        class XMLHandler extends DefaultHandler {
            public void startDocument() throws SAXException {
            }

            public void endDocument() throws SAXException {
            }

            public void startElement(String namespaceURI, String localName,
                String qName, Attributes attr) throws SAXException {
                XMLstartElement(namespaceURI, localName, qName, attr);
            }

            public void endElement(String namespaceURI, String localName,
                String qName) throws SAXException {
            }

            public void characters(char[] ch, int start, int length)
                throws SAXException {
                XMLcharacters(ch, start, length);
            }
        }

        processXML(new XMLHandler());
    }

    /**
     * Helps to parser the xml document.
     * @param handler The handler is used to help the xml parsing process.
     */
    public void processXML(DefaultHandler handler)
            throws XMLConfigurationException {
        try {
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse(streamXmlFile, handler);
```

```
    } catch (ParserConfigurationException e) {
        LOGGER.error(XMLExceptionMessages
                .getMessage(XMLConstants.ERROR_PARSING_XML), e);
        throw new XMLConfigurationException(e);
    } catch (SAXException e) {
        LOGGER.error(XMLExceptionMessages
                .getMessage(XMLConstants.ERROR_READING_XML), e);
        throw new XMLConfigurationException(e);
    } catch (IOException e) {
        LOGGER.error(XMLExceptionMessages
                .getMessage(XMLConstants.ERROR_READING_XML), e);
        throw new XMLConfigurationException(e);
    }
}

/**
 * When the parser is parsing the nxd.xml file, it calls this method that
 * do all the process to get the information from the file.
 * @param namespaceURI The namespace URI
 * @param localName The local name
 * @param qName The tag name
 * @param attr The attribute name
 * @throws SAXException Exception thrown if parsing document fails.
 */
protected void XMLstartElement(String namespaceURI, String localName,
        String qName, Attributes attr) throws SAXException {
    contents.reset();

    try {
        if (qName.equals(XMLConstants.DOMAIN)) {
            String clazz = attr.getValue(XMLConstants.CLASS);
            String domain = attr.getValue(XMLConstants.DOMAIN_NAME);
            String path = attr.getValue(XMLConstants.PATH);
            // Validate the class and domainName attributes
            XMLValidate.validateDomainTag(clazz, domain);
            domainMap.put(clazz, domain);
            // Path is an required attribute only if the automatic attribute
            // is "false"
            pathMap.put(domain, path);
        } else if (qName.equals(XMLConstants.GENERATE)) {
            String automatic = attr.getValue(XMLConstants.AUTOMATIC);
            XMLValidate.validateGenerateTag(automatic);
            // Validate attributes automatic and hierarchy.
            parametersMap.put(XMLConstants.AUTOMATIC, automatic);
        } else if (qName.equals(XMLConstants.HIERARCHY)) {
            String automatic = attr.getValue(XMLConstants.AUTOMATIC);
            String mainPath = attr.getValue(XMLConstants.MAIN_PATH);
```

```
            // Validate attributes automatic and hierarchy.
            XMLValidate.validateHierarchyTag(automatic, mainPath);
            parametersMap.put(XMLConstants.HIERARCHY, automatic);
            parametersMap.put(XMLConstants.MAIN_PATH, mainPath);
        }
    } catch (XMLSyntaxException ex) {
        throw new SAXException(ex);
    }
  }
  /**
   * When parsing the nxd.xml file, this method holds the characters read from
   * the file.
   * @param ch Character
   * @param start The start position
   * @param length The length
   * @throws SAXException Exception thrown if parsing document fails.
   */
  protected void XMLcharacters(char[] ch, int start, int length)
        throws SAXException {
      contents.write(ch, start, length);
  }
}
```

XMLConfigurationException.java

```
package org.nxd.exception;

/**
 * Exception class for XML Configuration errors.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 * @since 25/03/2006
*/
public class XMLConfigurationException extends Exception {

    /**
     * <code>serialVersionUID</code>
     */
    private static final long serialVersionUID = 6604169232665761808L;

    /**
     * Default constructor calling parent constructor.
     */
    public XMLConfigurationException() {
        super();
    }
```

```
    /**
     * @param arg0
     */
    public XMLConfigurationException(String arg0) {
        super(arg0);
    }

    /**
     * @param arg0
     * @param arg1
     */
    public XMLConfigurationException(String arg0, Throwable arg1) {
        super(arg0, arg1);
    }

    /**
     * @param arg0
     */
    public XMLConfigurationException(Throwable arg0) {
        super(arg0);
    }

}
```

XMLExceptionMessages.java

```
package org.nxd.exception;

import java.text.MessageFormat;
import java.util.ResourceBundle;


/**
 * This class is used to get exception messages from a properties file.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 * @since 03/24/2006
 *
 */
public class XMLExceptionMessages {

        /**
         * <code>bundle</code> - ResourceBundle
         */
```

```
        private static ResourceBundle bundle;

    static {
        bundle = ResourceBundle.getBundle("XMLExceptionMessages");
    }

    /**
     * Gets the value for the key passed.
     *
     * @return message message corresponding the key in properties
     * @param key
     *          key in properties. key cannot be null
     * @return message
     */
    public static String getMessage(String key) {
        String message = bundle.getString(key);
        return message;
    }

    /**
     * Gets the value for the key passed, and formt with the params.
     *
     * @return message message corresponding the key in properties
     * @param key
     *          key in properties. key cannot be null
     * @param param The array of the parameters.
     * @return Mesasge formated
     */
    public static String getMessage(String key, String[] param) {
        String message = bundle.getString(key);

        MessageFormat messageFormat = new MessageFormat(message);
        message = messageFormat.format(param);

        return message;
    }
}
```

XMLFileNotFoundException.java

```
package org.nxd.exception;

/**
 * This exception class is used when the nxd.xml document is not found.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
```

```
 * @since 03/24/2006
 *
 */
public class XMLFileNotFoundException extends Exception {

	/**
	 * <code>serialVersionUID</code> - The srial version UID.
	 */
	private static final long serialVersionUID = 5342509238230988842L;

	/**
	 * Default constructor.
	 */
	public XMLFileNotFoundException() {
		super();
	}

	/**
	 * @param arg0
	 */
	public XMLFileNotFoundException(String arg0) {
		super(arg0);
	}

	/**
	 * @param arg0
	 * @param arg1
	 */
	public XMLFileNotFoundException(String arg0, Throwable arg1) {
		super(arg0, arg1);
	}

	/**
	 * @param arg0
	 */
	public XMLFileNotFoundException(Throwable arg0) {
		super(arg0);
	}

}
```

XMLSyntaxException.java

```
package org.nxd.exception;
```

```
/**
 * This exception class is used when happens an error related to the nxd.xml
 *  document syntax.
 *
 * @author Rafael Lopes (rafael.lopes.br@gmail.com)
 * @since 03/24/2006
 *
 */
public class XMLSyntaxException extends Exception {

	/**
	 * <code>serialVersionUID</code> - The serial version UID.
	 */
	private static final long serialVersionUID = 5342509238230988842L;

	/**
	 * Default constructor.
	 */
	public XMLSyntaxException() {
		super();
	}

	/**
	 * @param arg0
	 */
	public XMLSyntaxException(String arg0) {
		super(arg0);
	}

	/**
	 * @param arg0
	 * @param arg1
	 */
	public XMLSyntaxException(String arg0, Throwable arg1) {
		super(arg0, arg1);
	}

	/**
	 * @param arg0
	 */
	public XMLSyntaxException(Throwable arg0) {
		super(arg0);
	}

}
```