6
OntCatalog: Um Protótipo de um OnOC

6.1
Motivação

Este capítulo descreve o protótipo desenvolvido como prova de conceito da metodologia e da arquitetura introduzidas nesta dissertação e serve como referencial para criação de OnOCs. As ontologias de exemplo utilizadas nos testes realizados foram construídas com base na metodologia descrita no Capítulo 3.

6.2
Descrição

O *OntCatalog* é um protótipo de OnOCs baseado na arquitetura descrita no Capítulo 4 e 5 desta dissertação.

Como um OnOC, o *OntCatalog* fornece um ambiente com serviços de consolidação para os objetos representados nas diversas fontes componentes da federação, e serviços de acesso para consulta a objetos e metadados das fontes componentes. Para isso, o *OntCatalog* mantém e gerencia as ontologias locais, representando as fontes de objetos e uma ontologia de referência, representando o esquema global da federação. Além disso, mantém os mapeamentos necessários para identificar os objetos e os metadados (classes e propriedades) equivalentes entre as fontes componentes.

O acesso aos termos das ontologias é feito através das operações implementadas no pacote *OnOCManager* que representa o módulo de interface de mesmo nome apresentado na arquitetura para OnOCs, nos Capítulos 4 e 5.

O *OntCatalog* foi desenvolvido em Java e usa a Jena API para manipular as ontologias mantidas por ele. Vale salientar que o protótipo foi implementado visando servir como referencial para instanciação de novos OnOCs, por isso, não fornece uma implementação para a interface de acesso dos clientes do catálogo. O módulo de interface *OnOCManager* fornece uma fachada de acesso às operações do sistema que pode ser acessada por classes que implementam métodos para atender requisições HTTP, CORBA, *Web Services* ou outras.

No módulo *OntRegistry* foi usado o banco de dados MySQL 4.0.18. O dicionário de dados das tabelas utilizadas pode ser visto no Anexo A.

O diagrama de pacotes da arquitetura mostrado na Figura 62 mostra o pacote do protótipo implementado, chamado *OntCatalog*. Por utilizar a API Jena para manipular as ontologias, o pacote do sistema possui dependência do pacote da API Jena, representado no diagrama pelo pacote *Jena*.

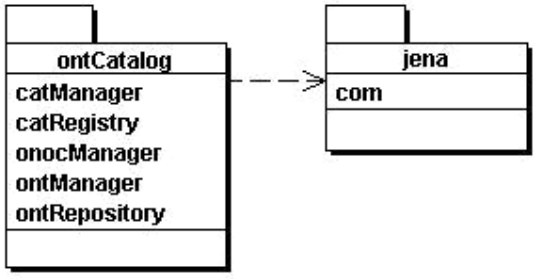

Figura 62 – Pacote *OntCatalog*.

Pela similaridade dos diagramas de classes e pacotes do protótipo implementado e da arquitetura, apresentados no Capítulo 5, estes foram omitidos deste capítulo.

As operações da interface implementadas no protótipo *OntCatalog* são as mesmas especificadas na Tabela 8 do Capítulo 4.

No protótipo apresentado neste capítulo os mapeamentos foram descritos nas ontologias locais através de cláusulas OWL, não havendo a necessidade de uso do módulo *MappingManager*. Porém, no caso do uso de regras para efetuar os mapeamentos requer que a implementação do pacote *MappingManager* seja refinada, a fim de carregar e interpretar as regras. Neste protótipo os mapeamentos são incluídos via cláusulas OWL junto com a descrição da ontologia local. Para inferir os relacionamentos foi utilizada uma máquina de inferência da linguagem OWL disponibilizada pela API Jena (Reynolds, 2005). As consultas realizadas pelas aplicações clientes fazendo uso desses mapeamentos são requisitadas através da operação *query.*

A operação *query* é usada para consultar o catálogo usando a linguagem de consulta RDQL (RDF *Data Query Language*). Incluindo o parâmetro *INF* com valor *true* a operação *query* realiza a consulta RDQL numa ontologia gerada usando inferência.

O protótipo implementado usa um repositório de ontologias baseado na API Jena. O repositório é acessado através da classe *JenaRepositoryFacade* (vide Figura 63). A classe *OntRepositoryAdapter* faz as transformações

necessárias para utilização dos métodos implementados no repositório e os métodos suportados pela arquitetura do OnOC. Para modelar este pacote foi utilizado o padrão de projeto *Adapter* (vide seção 4.4.2.3) que viabiliza a utilização de qualquer repositório de ontologias.

O repositório Jena utilizado faz o armazenamento utilizando o banco de dados MySQL. Os parâmetros de configuração para acesso ao banco de dados do repositório de ontologias podem ser acessados pela classe *Config* do pacote *OnOCManager*, que é a classe de definição das constantes do catálogo.

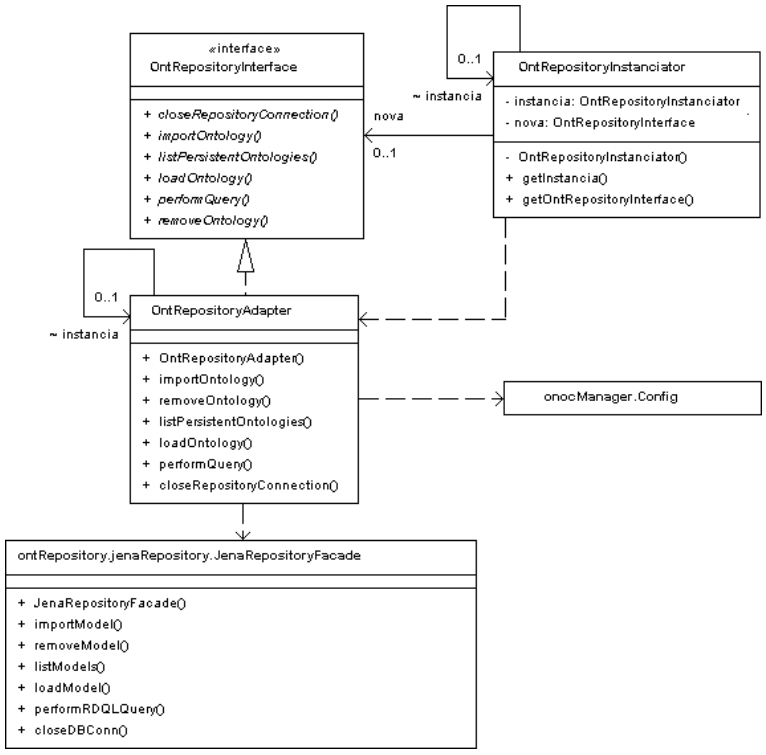

Figura 63 – Diagrama de classes do pacote *OntRepository*.

6.3
Testes

6.3.1
Critérios de teste utilizados

Foram utilizados dois critérios para teste do protótipo OntCatalog: testes unitários e testes funcionais.

Os testes unitários (teste de unidades) foram utilizados para verificar a corretude da resposta esperada dos métodos da classe *OnOCServices*. Estes

testes caracterizam-se por serem realizados do ponto de vista do desenvolvedor. Além destes, foram realizados testes funcionais, do ponto de vista de uma aplicação cliente a fim de verificar se a aplicação realiza a funcionalidade esperada.

As ontologias utilizadas nos testes foram construídas com base na metodologia descrita no Capítulo 3 e podem ser vistas no Anexo B.

6.3.2 Descrição dos casos teste

6.3.2.1 Testes unitários

Para os testes unitários, foi utilizado o JUnit[27], um *framework* que oferece suporte à criação de testes de unidades automatizados em Java. Para tanto, foi criado um caso de teste da classe *OnOCServices* para testar seus métodos.

O teste automatizado foi realizado utilizando um *script* executado pela ferramenta Ant[28]. Além disso, o *script* gera um relatório em HTML com os resultados dos testes e arquivo de *log* em XML da unidade testada (vide Anexo C).

Os casos de teste unitários foram implementados a partir da extensão da classe *junit.framework.TestCase*. Cada TestCase invoca os métodos a serem testados executando se necessário, um método de inicialização e outro de finalização. A inicialização é feita através do método chamado *setUp*. A finalização é feita através de um método chamado *tearDown*.

Para identificação do arquivo de teste foi adotada o padrão de nomenclatura do JUnit através do sufixo "Test" após o nome da classe que esta sendo testada. O caso de teste *OnOCServicesTest.java* descrito na Tabela 17 mostra os resultados esperados para cada método da classe *OnOCServices.java* do pacote *OnOCManager*.

[27] JUnit – http://www.junit.org/
[28] Apache Ant - http://jakarta.apache.org/ant/

Tabela 17: Descrição do caso de teste *OnOCServicesTest*.

CASO DE TESTE:	OnOCServicesTest.java		
PROPÓSITO:	Testar a corretude dos métodos da classe OnOCServices.java		
DESCRIÇÃO:	Este teste verificará a corretude dos resultados esperados dos métodos da classe OnOCServices.java		
DESCRIÇÃO		**RESULTADOS ESPERADOS**	**RESULTADOS OBTIDOS**
testGetOntology: dada a URI de identificação de uma ontologia local o método retorna a ontologia serializada no formato de saída especificado ou a mensagem de ERRO. O teste realizado verifica se o resultado obtido não retorna ERRO.		¬ERRO	ontologia local serializada
testGetOntology: sem o parâmetro URI o método retorna a ontologia de referência serializada no formato de saída especificado ou a mensagem de ERRO. O teste realizado verifica se o resultado obtido não retorna ERRO.		¬ERRO	ontologia de referência serializada
testGetTermDescription: dada a URI de identificação do termo o método retorna a descrição do termo serializada no formato de saída especificado ou a mensagem de ERRO. O teste realizado verifica se o resultado obtido não retorna ERRO.		¬ERRO	descrição do termo serializada
testRegisterObjectSrc: método que cadastra uma nova fonte de objetos. Retorna OK se for executado com sucesso e ERRO se houver falha.		¬ERRO	OK
testRegisterObjectSrc: método que cadastra uma nova fonte de objetos. Retorna OK se for executado com sucesso e ERRO se houver falha.		¬ERRO	OK
testRegisterNewApp: método que cadastra uma nova aplicação cliente ou participante no catálogo. Retorna OK se for executado com sucesso e ERRO se houver falha.		¬ERRO	OK
testSetAppPermission: método que define a permissão de acesso de uma aplicação a uma ontologia. Retorna OK se for executado com sucesso e ERRO se houver falha.		¬ERRO	OK
testQuery: método para realizar uma consulta RDQL sobre uma ontologia. Retorna o resultado da consulta serializado no formato de saída especificado ou a mensagem de ERRO. O teste realizado verifica se o resultado obtido não retorna ERRO.		¬ERRO	OK

6.3.2.2 Testes funcionais

Os testes funcionais foram utilizados para verificar se as operações básicas do sistema foram atendidas pela implementação. Os testes foram feitos do ponto de vista de uma aplicação cliente. São testadas as entradas referentes a algumas operações básicas do OntCatalog e verificadas as respectivas saídas.

Para execução destes casos de teste foi criada uma classe para chamada das operações da classe OnOCServices e escrita dos resultados na saída padrão. Os testes foram executados e os resultados visualizados na plataforma

Eclipse. As tabelas a seguir mostram o detalhamento das chamadas incluindo os métodos e os valores de retorno.

Tabela 18: Descrição do caso de teste da operação g*etOntology*.

OPERAÇÃO:	GetOntology (para ontologias locais)
PROPÓSITO:	Testar a corretude da operação getOntology
DESCRIÇÃO:	Este teste verificará a corretude dos resultados esperados da operação getOntology do protótipo OntCatalog.
Parâmetro	**Valor**
APPID	dani
APPPWD	senha
URI	urn:onoc:Src02#
OUTPUT	RDF/XML-ABBREV
SAÍDA ESPERADA	
Ontologia local “urn:onoc:Src02#” serializada no formato RDF/XML-ABBREV	
SAÍDA OBTIDA	

```
<rdf:RDF
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="urn:onoc:Src02#"
    xmlns:RO="urn:onoc:AeroportoRO#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <owl:Ontology rdf:about="urn:onoc:Src02"/>
  <owl:Class rdf:about="urn:onoc:Src02#Airport">
    <owl:equivalentClass>
      <rdf:Description rdf:about="RO:Aeroporto">
    </owl:equivalentClass>
  </owl:Class>
  ...
  <Airport rdf:about="urn:onoc:Src02#Airport.JFK"/>
  <Airport rdf:about="urn:onoc:Src02#Airport.ATL"/>
  <Airport rdf:about="urn:onoc:Src02#Airport.MIA"/>
  <Flight rdf:about="urn:onoc:Src02#Flight.DL1159">
    <flight.destination
           rdf:resource="urn:onoc:Src02#Airport.ATL"/>
    <flight.origin rdf:resource="urn:onoc:Src02#Airport.MIA"/>
  </Flight>
</rdf:RDF>
```

Tabela 19: Descrição do caso de teste da operação g*etOntology* (OR).

OPERAÇÃO:	GetOntology (para ontologia de referência)
PROPÓSITO:	Testar a corretude da operação getOntology
DESCRIÇÃO:	Este teste verificará a corretude dos resultados esperados da operação getOntology do protótipo OntCatalog.
Parâmetro	**Valor**
APPID	dani
APPPWD	senha
OUTPUT	RDF/XML-ABBREV
SAÍDA ESPERADA	
Ontologia de referência serializada no formato RDF/XML-ABBREV	
SAÍDA OBTIDA	

```
<rdf:RDF
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="urn:onoc:AeroportoRO#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
```

```
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <owl:Ontology rdf:about="urn:onoc:AeroportoRO"/>
  <owl:Class rdf:about="urn:onoc:AeroportoRO#Aeroporto"/>
  <owl:Class rdf:about="urn:onoc:AeroportoRO#Estado"/>
  <owl:Class rdf:about="urn:onoc:AeroportoRO#Voo"/>
  <owl:Class rdf:about="urn:onoc:AeroportoRO#Pais"/>
  <owl:DatatypeProperty
rdf:about="urn:onoc:AeroportoRO#voo.cod">
    <rdf:type rdf:resource=" owl:FunctionalProperty"/>
    <rdfs:domain rdf:resource="urn:onoc:AeroportoRO#Voo"/>
    <rdfs:range rdf:resource="xsd:string"/>
  </owl:DatatypeProperty>
  ...
  <Aeroporto rdf:about="urn:onoc:AeroportoRO#Aeroporto.POA">
      <aeroporto.cod>POA</aeroporto.cod>
     <aeroporto.estado
              rdf:resource="urn:onoc:AeroportoRO#Estado.RS"/>
      <aeroporto.nome>Salgado Filho</aeroporto.nome>
  </Aeroporto>
  <Pais rdf:about="urn:onoc:AeroportoRO#Pais.Brasil">
    <pais.cod>55</pais.cod>
    <pais.nome>Brasil</pais.nome>
  </Pais>
  <Estado rdf:about="urn:onoc:AeroportoRO#Estado.RS">
    <estado.cod>RS</estado.cod>
    <estado.nome>Rio Grande do Sul</estado.nome>
    <estado.pais
              rdf:resource="urn:onoc:AeroportoRO#Pais.Brasil"/>
  </Estado>
</rdf:RDF>
```

Tabela 20: Descrição do caso de teste da operação g*etTermDescription*.

OPERAÇÃO:	GetTermDescription
PROPÓSITO:	Testar a corretude da operação getTermDescription
DESCRIÇÃO:	Este teste verificará a corretude dos resultados esperados da operação getTermDescription do protótipo OntCatalog.
Parâmetro	**Valor**
APPID	dani
APPPWD	senha
URI	urn:onoc:AeroportoRO#Voo
OUTPUT	RDF/XML-ABBREV
SAÍDA ESPERADA	
Descrição do termo “urn:onoc:AeroportoRO#Voo” serializada no formato RDF/XML-ABBREV	
SAÍDA OBTIDA	

```
<rdf:RDF
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <owl:Class rdf:about="urn:onoc:AeroportoRO#Voo"/>
  <owl:DatatypeProperty
      rdf:about="urn:onoc:AeroportoRO#voo.cod">
    <rdfs:domain rdf:resource="urn:onoc:AeroportoRO#Voo"/>
  </owl:DatatypeProperty>
  <owl:FunctionalProperty
      rdf:about="urn:onoc:AeroportoRO#voo.origem">
    <rdfs:domain rdf:resource="urn:onoc:AeroportoRO#Voo"/>
    <rdfs:range rdf:resource="urn:onoc:AeroportoRO#Aeroporto"/>
```

```
  </owl:FunctionalProperty>
  <owl:FunctionalProperty
      rdf:about="urn:onoc:AeroportoRO#voo.destino">
    <rdfs:domain rdf:resource="urn:onoc:AeroportoRO#Voo"/>
    <rdfs:range rdf:resource="urn:onoc:AeroportoRO#Aeroporto"/>
  </owl:FunctionalProperty>
</rdf:RDF>
```

Tabela 21: Descrição do caso de teste da operação *query*.

OPERAÇÃO:	Query
PROPÓSITO:	Testar a corretude da operação query.
DESCRIÇÃO:	Este teste verificará a corretude dos resultados esperados da operação query do protótipo OntCatalog.
Parâmetro	**Valor**
APPID	dani
APPPWD	senha
URI	urn:onoc:Src02#
QUERY	SELECT ?x WHERE (?x, <rdf:type>, <urn:onoc:Src02#Airport>)
OUTPUT	RDF/XML-ABBREV
INF	false
SAÍDA ESPERADA	
Resultado da consulta serializada no formato RDF/XML-ABBREV	
SAÍDA OBTIDA	

```
<rdf:RDF
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:j.0="urn:onoc:Src02#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <j.0:Airport rdf:about="urn:onoc:Src02#Airport.ATL"/>
  <j.0:Airport rdf:about="urn:onoc:Src02#Airport.MIA"/>
  <j.0:Airport rdf:about="urn:onoc:Src02#Airport.JFK"/>
</rdf:RDF>
```

Tabela 22: Descrição do caso de teste da operação *query* (usando inferência).

OPERAÇÃO:	Query (usando inferência)
PROPÓSITO:	Testar a corretude da operação query.
DESCRIÇÃO:	Este teste verificará a corretude dos resultados esperados da operação query do protótipo OntCatalog.
Parâmetro	**Valor**
APPID	dani
APPPWD	senha
URI	urn:onoc:Src02#
QUERY	SELECT ?x WHERE (?x, <rdf:type>, <urn:onoc:Src02#Airport>)
OUTPUT	RDF/XML-ABBREV
INF	True
SAÍDA ESPERADA	
Resultado da consulta serializada no formato RDF/XML-ABBREV	
SAÍDA OBTIDA	

```
<rdf:RDF
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:j.0="urn:onoc:Src02#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <j.0:Airport rdf:about="urn:onoc:Src02#Airport.ATL"/>
  <j.0:Airport rdf:about="urn:onoc:AeroportoRO#Aeroporto.GIG"/>
  <j.0:Airport rdf:about="urn:onoc:AeroportoRO#Aeroporto.YWC"/>
  <j.0:Airport rdf:about="urn:onoc:AeroportoRO#Aeroporto.SDU"/>
  <j.0:Airport rdf:about="urn:onoc:Src02#Airport.MIA"/>
  <j.0:Airport rdf:about="urn:onoc:AeroportoRO#Aeroporto.YYC"/>
  <j.0:Airport rdf:about="urn:onoc:AeroportoRO#Aeroporto.POA"/>
  <j.0:Airport rdf:about="urn:onoc:Src02#Airport.JFK"/>
</rdf:RDF>
```