MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

ARQUITETURA PARA TRADUÇÃO DE MODELOS DINÂMICOS PARA MODELOS ESTÁTICOS DE CLASSE

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA

(PIBIC/CNPq/INPE)

Bolsista: Jean Novaes Santos
E-mail: jeansjcsp@gmail.com

Orientador: Eduardo Martins Guerra
E-mail: guerraem@gmail.com

Julho de 2015

São José dos Campos

1 – Introdução

Um sistema baseado na arquitetura AOM (Adaptive Object Model) é um sistema capaz de alterar seu próprio domínio (YODER e JOHNSON, 2002). Isto é possível pois o design do sistema é concebido ao nível de instâncias classe. Isto foge do modelo tradicional, baseado em classes. As mudanças ocorrem em tempo de execução do software usando metadados. A alteração dos metadados em tempo de execução vai refletir mudanças nos domínios.

A flexibilidade dos sistemas que usam design AOM necessitam de constantes mudanças, tanto em sua concepção quanto em seu uso após a implementação final. Os metadados sobre os quais se baseia a arquitetura podem ser provir de fontes externas com arquivos XML ou bases de dados. Quando os metadados são alterandos vão refletir mudanças nos respectivos domínios.

A abordagem de se utilizar metadados para representar características do domínio de sistemas trás vantagens em relação ao modelo tradicional em virtude da possibilidade de adaptação. Muitos desenvolvedores encontram dificuldades no uso da arquitetura AOM, pois esta possui vários níveis de abstração, dificultando a evolução da aplicação, pois é necessária a mudança em vários locais onde o modelo é interpretado. Isso ocorre, pois existe o benefício da adaptabilidade, porém não é possível reutilizar diretamente o modelo em vários contextos, devido ao acoplamento criado com os domínios. Este acoplamento ocorre pelo tipo da abordagem quando se implementa a arquitetura AOM na maioria dos sistemas, que seria uma abordagem bottom- up (MATSUMOTO, GUERRA 2012), sendo aplicada apenas quando necessário.

O framework Esfinge AOM Role Mapper (GUERRA, 2012) visa adaptar estruturas AOM específicas de uma determinada aplicação, que estão acopladas aos seus domínios específicos, a uma estrutura genérica, baseada no uso de metadados. O trabalho proposto visa o uso de metadados gerados pelo framework AOM Role Mapper para a geração de classes estáticas ou beans.

Este trabalho propõe o uso de metadados provenientes do framework AOM Role Mapper para a geração de classes estáticas em tempo de execução, visando o uso de frameworks genéricos no contexto das aplicações AOM.

2 – Arquitetura AOM

O modelo de arquitetura de software AOM(Adaptive Object Model) é um modelo de modelagem de software que promove a adaptação de uma aplicação a mudanças de sua arquitetura interna.

2.1 - Características

Atualmente as mudanças de requisitos das aplicações tem sido um requisito em diferentes contextos. A opção pelo ágil é constante pois neste modelo existem várias interações ou ciclos de desenvolvimento, tentando prever ou mitigar o efeitos das mudanças (MATSUMOTO e GUERRA, 2012).

Como muitas mudanças não são detectadas durante o processo de desenvolvimento existem cenários onde é aplicado a arquitetura AOM (Adaptive Object Model), um modelo de arquitetura que visa representar classes, atributos, relacionamentos e comportamentos como metadados (YODER, 2014). Dessa maneira, o modelo AOM interpreta os metadados persistidos com arquivos XML ou em bases de dados e, de acordo com mudanças nestes metados, atualiza o modelo de domínio das aplicações, principalmente em cenários onde se encontram regras de negócio, que podem mudar constantemente.

A arquitetura AOM é composta de diversos padrões de projeto, que compostos, são usados para criar aplicações seguindo esta arquitetura. A básica da arquitetura é composta pelos padrões de projeto Type Object, Property e Type formando o padrão Type Square, descrito na Figura 1.

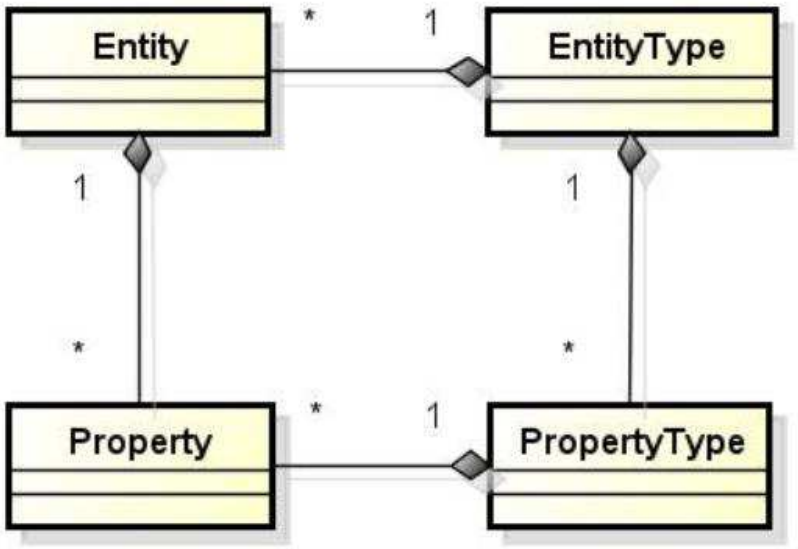

Figura 1 – Padrão Type Square(Adaptada de Guerra, 2014)

O padrão Type Object é tipicamente utilizado em situações onde o número de classes em aplicações não pode ser determinado com exatidão, e seu número pode crescer consideravelmente. Esta situação ocorre quando existem diferenças sutis entre objetos. O padrão Property se enquadra em situações em que existem inúmeras propriedades que descrevem o comportamento de objetos, mas apenas algumas são utilizadas quando uma instância deste objeto e requisitada para descrever um caso particular.

Da união destes dois padrões surge o padrão Type Square, que constitui-se como o núcleo das aplicações AOM. Na Figura 1 o padrão Type Object é usado duas vezes, sendo uma para representar entidades e tipos de entidades e outra para representar as propriedades e os tipos de propriedades. Nesta abordagem podem ser criadas diversas entidades com diferentes tipos de propriedades.

3 – Reflexão e Anotações

Existem situações em que somente a orientação a objetos não é suficiente pra resolver todos os problemas do desenvolvimento de software, mas em alguns casos pode ser utilizada a reflexão.

A reflexão e um recurso em que um software pode se observar e mudar seu comportamento, através da introspecção, onde o software obtém informações sobre seu próprio código. Este conceito também pode ser chamado de metaprogramação, onde o código manipula o próprio código. O objetivo primário é manipular instâncias de objetos através do uso de anotações. Dessa maneira pode-se interagir com instâncias de classes que não possuem estrutura conhecida.

Na linguagem JAVA utiliza-se a API Reflection, do pacote java.lang.reflect, quando se faz necessário o uso da reflexão. A alteração de classes manipuladas pela API Reflection deve ser feita com o auxílio da manipulação de bytecode de classes, pois em linguagem JAVA não é possível alterar classes que já tenham sido carregadas na máquina virtual (JVM – Java Virtual Machine).

3.1 – Geração de ByteCode

O ASM (Bruneton, 2002) é um framework de manipulação e análise de bytecode. Para analisar a geração de bytecode e geração de classes em java seu uso é necessário. Existem plug-ins que facilitam a análise pela incorporação de ferramentas em IDE's de desenvolvimento.

Na Figura 2 é apresentado um exemplo bytecode sendo gerado e apresentado pelo ByteCode OutLine Plugin, um plugin para a IDE Eclipse proveniente do framework ASM.

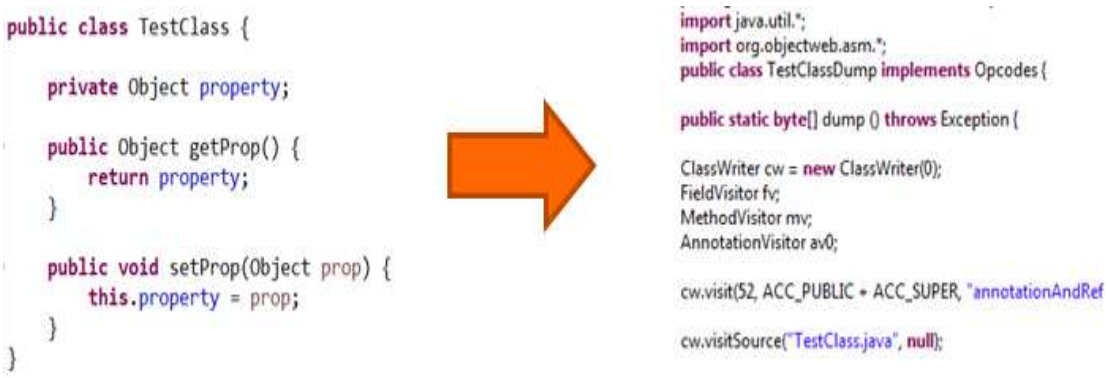

Figura 2 – Gerando ByteCode com o ByteCode Outline Plugin

4 – Medatados

Os metadados são estruturas de informação que descrevem e ajudam a recuperar ou manipular fontes de informação. Metadados são muitas vezes referidos como "dados sobre dados" (GUENTHER e RADEBAUGH, 2004).

No contexto de linguagens de programação orientadas a objetos, os metadados são comumente informações a respeito das classes. Usando uma combinação entre metadados e a reflexão consegue-se informações a respeito das classes em um software, sem necessariamente conhecer a estrutura destas classes previamente. Além dos metadados fornecidos pelas próprias classes, o que se obtém com a API Reflection na linguagem Java, existem fontes externas de metadados. Metadados podem ser marcados na estrutura das classes, através do uso de anotações. Dessa maneira, os metadados assim dizem respeito às informações sobre o código das classes.

5 – Os framerworks AOM Role Mapper e Comparison

Os frameworks AOM Role Mapper e Comparison são usados respectivamente para adaptar estruturas AOM e para comparação de instâncias. A seguir é apresentada uma breve descrição a respeito destes dois componentes do Projeto Esfinge (Guerra *et al*., 2014).

5.1 – AOM Role Mapper

O framework AOM Rolemapper, parte do projeto Esfinge, foi desenvolvido com o objetivo de evitar a adição de complexidade aos sistemas AOM. O framework adapta estruturas de aplicações com domínio específico, de uma estrutura AOM complexa, fornecendo uma estrutura AOM genérica para que outros frameworks genéricos possam ser usados em aplicações AOM.

Para integrar aplicações AOM aos frameworks genéricos, o framework AOM RoleMapper utiliza metadados provindos da aplicação, sendo estes marcados por meio de anotações. Na Figura 3 é apresentado um exemplo do uso de anotações para o mapeamento de uma estrutura AOM.

```
@Entity
public class TestClass {

    @EntityProperty
    private Integer number;
    @EntityProperty
    private Double height;

    public TestClass() {}
    public Integer getNumber() {
        return number;
    }
    public void setNumber(Integer number) {
        this.number = number;
    }
    public Double getHeight() {
        return height;
    }
    public void setHeight(Double height) {
        this.height = height;
    }
}
```

Figura 3 – AOM Role Mapper: Uso de anotações para mapear metadados.

5.1 – Comparison

O framework Comparison, outro componente do projeto Esfinge, basicamente compara

instâncias de uma mesma classe e retorna, se existirem, diferenças entre elas. As diferenças entre duas entidades podem ser usadas para alguma forma de registros de mudanças. Na Figura 4 é mostrado um exemplo do uso do Comparison.

```
public class CompareCars {

    public static void main(String[] args) throws CompareException {
        Car car1 = new Car();
        Car car2 = new Car();

        car1.setColor("black");
        car1.setYear(1975);

        car2.setColor("white");
        car2.setYear(1975);

        ComparisonComponent c  = new ComparisonComponent();
        List<Difference> difs = c.compare(car1, car2);

         for(Difference d : difs){
                System.out.println(d);
            }

    }

}
```

```
color : black / white
```

Figura 4 – Uso do Comparison e saída correspondente as diferenças.

6 – Atividades realizadas e Resultados

Segue um resumo das atividades realizadas durante o projeto e resultados alcançados.

6.1 – Atividades Realizadas

Na primeira etapa foi realizado um estudo sobre vários trabalhos envolvendo o framework Esfinge AOM RoleMapper. Neste estudo foram verificados detalhes sobre problemas encontrados nas abordagens de implementações AOM tradicionais. Foi conhecida a estrutura geral das aplicações AOM e como o framework Esfinge pode se usado para facilitar a reutilização de aplicações AOM em diferentes domínios.

Na segunda etapa foi dada continuidade ao estudo das aplicações AOM em relação a abordagens de desenvolvimento de software que são utilizadas. Foram estudados padrões de projeto de software e também o uso de Anotações e Reflexões, que são usadas para ampliar a eficiência da linguagem Java no design de sistemas onde

a orientação à objetos não é suficiente.

Após o estudo sobre Anotações e Reflexões foi reproduzido um exemplo de manipulação de bytecode usando o plugin ASM e sua extensão na IDE Eclipse.

Na terceira etapa do trabalho foi dado início a implementação de um protótipo de uma Bean Factory, ou classe auxiliar que faz a geração de beans genéricos. A implementação foi baseada em código que gera automaticamente os métodos construtores, atributos privados, getters e setters para uma entidade genérica, fornecida pelo framework AOM Role Mapper. Os metadados mapeados também devem ser fornecidos pelo framework e estes refletem o domínio das aplicações.

Na quarta e última etapa a Bean Factory foi evoluída de maneira e gerar tipos específicos e não somente atributos do tipo Object. Também foi implementado um cache para reaproveitar classes com um mesmo nome que foram recentemente geradas. Para avaliação final da implementação uma entidade foi passada a Bean Factory e os atributos de classe gerada foram recuperados. Além disso, foi utilizado o framework Comparison como validação final, pois este se presta a comparar instâncias de classes, e uma comparação efetiva de instâncias de uma classe gerada pelo protótipo caracteriza a correta geração de uma classe que atende o padrão JavaBeans.

6.2 – Resultados

O objetivo final da Bean Factory, ou classe geradora de beans, é produzir classes estáticas que possam ser usadas para adaptar estruturas AOM, permitindo seu uso com frameworks genéricos que foram projetados para serem aplicados em estruturas que seguem o padrão JavaBeans.

Para avaliar a funcionalidade da versão final do protótipo de uma Bean Factory, o framework AOM Rolemapper foi usado. Neste teste um objeto proveniente do framework do tipo IEntity foi criado e este usado em conjunto com a Factory. Na Figura 5 é apresentado um método para executar a geração de beans a partir da Bean Factory.

```java
public static void main(String[] args) throws Exception {

    IEntityType entityType = new GenericEntityType("Person");
    entityType.addPropertyType(new GenericPropertyType("number",
            Integer.class));
    entityType.addPropertyType(new GenericPropertyType("height",
            Double.class));
    entityType
            .addPropertyType(new GenericPropertyType("name", String.class));

    IEntity entity = entityType.createNewEntity();
    entity.setProperty("number", 27);
    entity.setProperty("height", 1.8);
    entity.setProperty("name", "João");

    BeanFactory gc = new BeanFactory();

    Object obj = gc.generate(entity, true);

    System.out.println("----------------------Generated");
    printClass(obj);

}
```

Figura 5 – Classe de teste para geração de Beans.

No exemplo de geração de beans citado, um objeto do tipo IEntityType, fornecido pelo AOM Role Mapper é instanciado. Este objeto trata-se de uma entidade AOM genérica, que tem seus atributos adicionados. Em seguida a Bean Factory é acionada, passando-se como parâmetro este entidade AOM, que tem seus atributos recuperados e impressos conforme a Figura 6.

```
----------------------Generated
public java.lang.String PersonAOMBeanAdapter.getName()   => João
public void PersonAOMBeanAdapter.setName(java.lang.String)
public java.lang.Integer PersonAOMBeanAdapter.getNumber()   => 27
public void PersonAOMBeanAdapter.setNumber(java.lang.Integer)
public void PersonAOMBeanAdapter.setHeight(java.lang.Double)
public java.lang.Double PersonAOMBeanAdapter.getHeight()   => 1.8
public final void java.lang.Object.wait() throws java.lang.InterruptedException
public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException
public final native void java.lang.Object.wait(long) throws java.lang.InterruptedException
public boolean java.lang.Object.equals(java.lang.Object)
public java.lang.String java.lang.Object.toString()
public native int java.lang.Object.hashCode()
public final native java.lang.Class java.lang.Object.getClass()   => class PersonAOMBeanAdapter
public final native void java.lang.Object.notify()
public final native void java.lang.Object.notifyAll()
private org.esfinge.aom.api.model.IEntity PersonAOMBeanAdapter.entity
```

Figura 6 – Impressão de atributos recuperados.

Para uma avaliação mais conclusiva foi utilizado o framework Comparison, que se presta justamente a comparar instâncias geradas de uma mesma classe, que por sua vez deve seguir o padrão JavaBeans. Se for possível que instâncias de uma classe gerada pela Bean Factory possam ser comparadas fica caracterizada a geração de beans estáticos inicialmente proposta. Na Figura 7 duas entidades com diferenças sutis são criadas. O framework Comparison tem seu componente instanciado e uma lista de diferenças entre os dois objetos é impressa.

```
public void comparingObjects() throws Exception {
    entityTypeA = new GenericEntityType("Car");
    entityTypeA.addPropertyType(new GenericPropertyType("plateNumber",
            String.class));
    entityTypeA.addPropertyType(new GenericPropertyType(
            "yearOfManufacturing", Integer.class));
    entityTypeA.addPropertyType(new GenericPropertyType("color",
            String.class));
    IEntity entityA = entityTypeA.createNewEntity();
    entityA.setProperty("plateNumber", "DGZ-1714");
    entityA.setProperty("yearOfManufacturing", 1980);
    entityA.setProperty("color", "yellow");

    BeanFactory gc = new BeanFactory();
    Object objA = gc.generate(entityA);

    entityTypeB = new GenericEntityType("Car");

    entityTypeB.addPropertyType(new GenericPropertyType("plateNumber",
            String.class));
    entityTypeB.addPropertyType(new GenericPropertyType(
            "yearOfManufacturing", Integer.class));
    entityTypeB.addPropertyType(new GenericPropertyType("color",
            String.class));

    IEntity entityB = entityTypeB.createNewEntity();
    entityB.setProperty("plateNumber", "DZZ-3421");
    entityB.setProperty("yearOfManufacturing", 1979);
    entityB.setProperty("color", "yellow");

    Object objB = gc.generate(entityB);

    ComparisonComponent c = new ComparisonComponent();
    List<Difference> difs = c.compare(objA, objB);

    System.out.println("Differences:");

    for (Difference d : difs) {
        System.out.println(d.toString());
    }
}
```

```
Differences:
yearOfManufacturing : 1980 / 1979
plateNumber : DGZ-1714 / DZZ-3421
```

Figura 7 – Validação com o framework Comparison.

Na Figura 8 é apresentada uma visão geral da solução proposta para a geração de beans estáticos. Uma aplicação, que tem acesso a uma estrutura AOM, aciona um fábrica de adaptadores, ou Bean Factory, e esta gera dinamicamente beans genéricos baseados em metadados provenientes da estrutura AOM por meio do framework AOM Role Mapper. Este bean dinamicamente gerado, que tem o papel de adaptar a estrutura, pode então ser utilizado em frameworks genéricos, que precisam desta estrutura comum para serem aplicados.

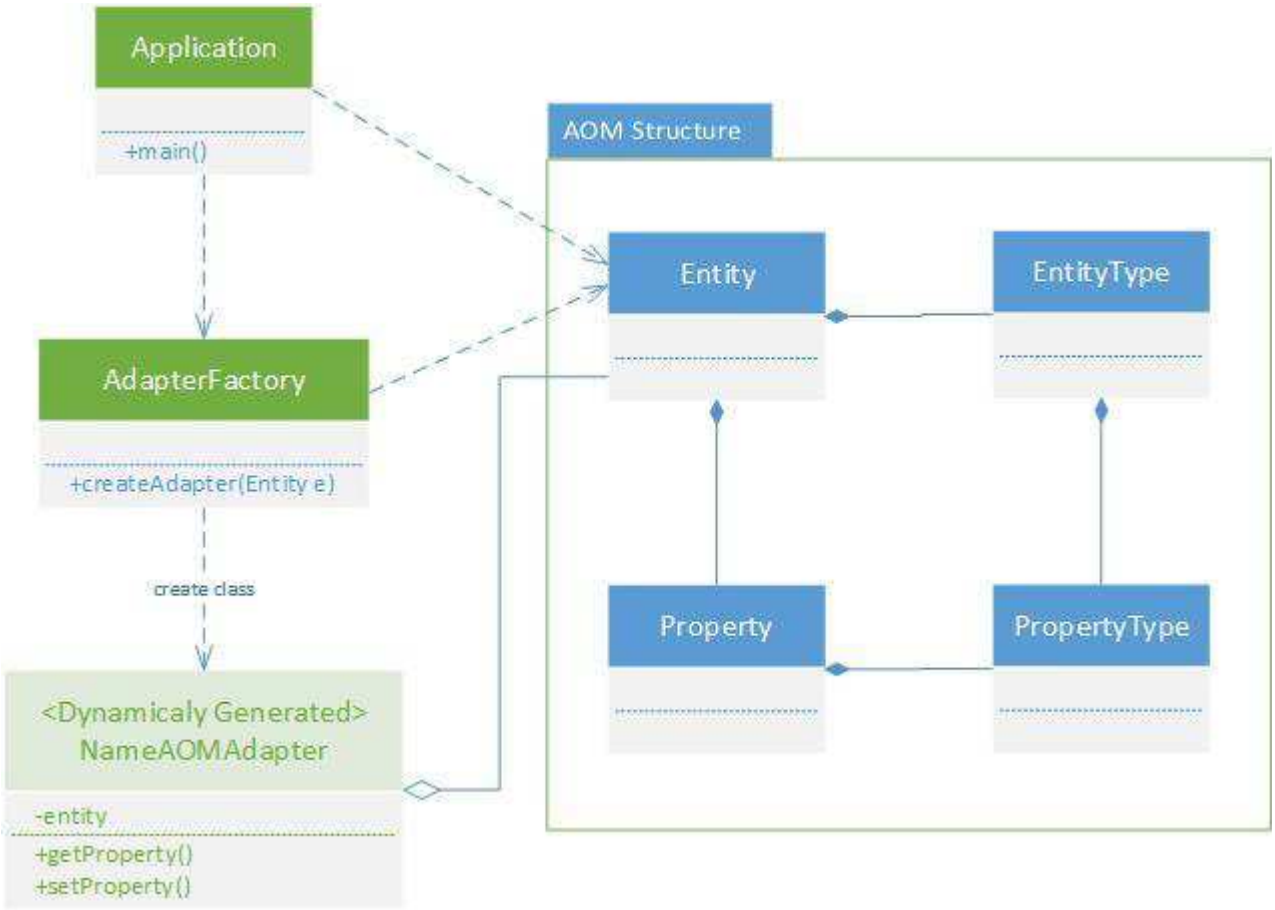

Figura 8 – Visão geral do protótipo final.

7 – Conclusão

A obtenção final de um protótipo de geração de classes estáticas foi alcançada. Ficou caracterizado segundo a avaliação final realizada que o protótipo de geração de beans, ou Bean Factory, se presta corretamente a gerar beans que podem ser utilizados para integração com frameworks genéricos externos à aplicacação AOM.

Contudo, este solução constitui-se com um primeiro passo para uma implementação mais completa, que possa prever por exemplo a geração automática de anotações que são utilizadas por frameworks comuns pra inserir funcionalidades adicionais as aplicações. Outro avanço seria o teste da solução em aplicações reais.

Este trabalho foi usado como base para o artigo “**Dynamic Generated Adapters from Adaptive Object Models to Static APIs**” que foi submetido no evento **22nd CONFERENCE ON PATTERN LANGUAGES OF PROGRAMS, 2015.**

8 – Referências

Guerra E. M. et al., **Projeto Esfinge**. Disponível em: < http://esfinge.sourceforge.net/ >, Acesso em: 05 dec. 2014.

Matsumoto, P. ; Guerra, E. M.: “**An Architectural Model for Adapting Domain-Specific AOM Applications**”. In: SBCARS- Simpósio Brasileiro de Componentes, Arquitetura e Reutilizacação de Software, 2012, Natal.

YODER, Joseph W.; **Adaptive Object Model**. Disponível em: < http://adaptiveobjectmodel.com/ >, Acesso em: 05 dec. 2014.

BRUNETON, E.; LENGLET, R.; COUPAYE, T.. **ASM - Home Page**. 2002. Disponível em: <http://asm.ow2.org/>. Acesso em: 09 fev. 2015.

GUENTHER, Rebecca; RADEBAUGH, Jaqueline. **Understanding Metadata**. 2004. Disponível em: <http://www.niso.org/publications/press/UnderstandingMetadata.pdf>. Acesso em: 06 dez. 2014.