Padrão de Anotação Semântica de Código e sua Aplicação no Desenvolvimento de Componentes

Taluna Mendes d'Araújo Costa
Núcleo de Sistemas e Computação – UNIFACS
R. Ponciano de Oliveira, 126, 41950-275,
Salvador-BA, Brazil
Email: taluna@gmail.com

André Santanchè
Instituto de Computação – UNICAMP
Av. Albert Einstein, 1251, 13083-852,
Campinas-SP, Brazil
Email: santanche@ic.unicamp.br

***Resumo*—Anotações em código-fonte têm assumido um papel cada vez mais relevante na produção de programas. Anotações estruturadas – aquelas que seguem esquemas de construção – fornecem semântica extra ao código e são aptas para uso na automatização de tarefas complementares ao seu desenvolvimento. Iniciativas recentes têm associado estas anotações a ontologias, ampliando sua expressão e interoperabilidade. A pesquisa apresentada neste artigo contribui nesta direção com uma nova estratégia baseada em meta-anotações, para subsidiar o uso de anotações estruturadas implicitamente associadas a ontologias. Ela sistematiza e simplifica o processo de anotações semânticas, dado que programadores podem usá-las sem precisar conhecer especificidades das ontologias envolvidas. Esta estratégia foi aplicada com sucesso no desenvolvimento de componentes seguindo o modelo *Digital Content Component* (DCC). A partir das anotações, foi automatizada a construção de descrições semânticas dos componentes e suas interfaces, otimizando seu processo de desenvolvimento.**

I. Introdução

Houve um tempo em que o principal meio de documentar o código-fonte era a utilização de comentários não estruturados realizados dentro do mesmo, que explicavam o funcionamento do trecho anotado. Como estes comentários eram restritos a arquivos individuais deste código, isso dificultava a obtenção de uma perspectiva holística do sistema. Avanços nas técnicas de desenvolvimento expandiram o conceito de software. Além do código do programa, documentos relacionados e dados associados adquiriram relevância na composição do software, exercendo um papel importante no processo de engenharia de software [1]. A documentação passa a ter mais importância, pois é um fator determinante para a qualidade e produtividade no desenvolvimento e, principalmente, manutenção do software.

Dentre os desafios relacionados à documentação do software, a localização e a ligação entre as diferentes peças dentro do mesmo são consideradas essenciais para entender como ele funciona e como sua manutenção deve ser feita de forma apropriada. Usualmente, peças do mesmo software e sua respectiva documentação são desenvolvidas por pessoas diferentes, em estágios distintos e desvinculados. Isto resulta em uma natural fragmentação da documentação e, como consequência, profissionais que não participaram do processo do desenvolvimento gastam muito tempo sintetizando as informações e estabelecendo os elos entre os elementos [2].

Anotações estruturadas dentro do código-fonte representaram um grande avanço neste sentido. Por estruturada entende-se que a anotação deve seguir um esquema previamente definido, possibilitando a automatização de sua extração e interpretação. As anotações estruturadas também inauguraram um novo campo de aplicações, no qual as anotações subsidiam a automatização de outras atividades além da documentação, e.g., mapeamento de classes a esquemas relacionais, descrição de interfaces para serviços Web.

Ontologias associadas a anotações estruturadas ampliam seu poder de expressão, principalmente quando seguem padrões abertos da Web Semântica. As ontologias podem representar o domínio da aplicação e se relacionar com peças internas do software. Elas podem dar suporte à descrição de entidades de alto nível – como componentes de software – e suas funcionalidades [3].

A pesquisa apresentada neste artigo envolve nossa estratégia de associação de anotações estruturadas a ontologias. Comparada aos trabalhos relacionados nesta direção, destacamos duas contribuições da nossa proposta: (i) ela unifica duas vertentes de anotação de código – uma voltada à descrição e outra que alinha estruturas do programa com aquelas da ontologia; (ii) ela introduz o princípio de meta-anotação como ponte entre anotações estruturadas e ontologias. As meta-anotações sistematizam o processo de anotação e tornam as associações às ontologias implícitas, possibilitando que programadores as usem sem conhecer especificidades das ontologias.

Esta pesquisa foi motivada pelo processo de produção de componentes que seguem o modelo *Digital Content Component* (DCC) [4], [5]. Conforme será mostrado na Seção II-B, esse modelo exige a criação de metadados que descrevem semanticamente os componentes e suas interfaces, utilizando padrões abertos da Web Semântica. Nossa estratégia de anotação semântica foi integrada ao processo de codificação do componente, minimizando a carga extra envolvida no registro de metadados e dispensando o aprendizado de ferramentas adicionais.

Este artigo está organizado da seguinte maneira: a Seção II apresenta os conceitos básicos que subsidiaram a pesquisa; a Seção III apresenta a nossa proposta de anotação semântica de código associada a ontologias; a Seção IV descreve a aplicação prática da estratégia de anotações no contexto de

desenvolvimento baseado em componentes, no caso os *Digital Content Components*; a Seção V confronta nossa abordagem com trabalhos relacionados; a Seção VI apresenta a conclusão e trabalhos futuros.

II. Fundamentação Teórica

Esta seção apresenta uma síntese dos conceitos relacionados que envolvem tanto as anotações estruturadas e sua associação a ontologias, como o modelo DCC sobre o qual foi implementada nossa estratégia.

A. Anotações Estruturadas

Diversas linguagens de programação têm incluído mecanismos de inserção de metadados na forma de anotações em código-fonte, de modo que possam ser usados em tarefas de pós-processamento, ou recuperados durante a execução do programa. Tais como os comentários, essas anotações são estruturas declarativas inseridas no código-fonte, mas ao contrário dos comentários, anotações não são ignoradas pelo compilador; elas seguem esquemas de construção passíveis de interpretação, de modo possam ser embutidas no código compilado. Os metadados inseridos no código compilado podem ser acessados através de reflexão.

A este mecanismo daremos o nome de anotações estruturadas de código. Em linguagens de programação orientadas a objetos tem se difundido o uso de anotações estruturadas que seguem esquemas definidos por classes [6], e.g., Java *Annotations* e o .NET *Attributes*. Neste artigo nos concentraremos no mecanismo utilizado pela linguagem Java, por ser este o contexto em que se desenvolveu esta pesquisa.

Anotações na linguagem Java foram implementadas na versão 1.5 [7], através da especificação *JSR-175: A Metadata Facility for the Java Programming Language* [8]. A fim de demonstrar o uso das *Java Annotations*, será adotado um exemplo, que também será a base para a apresentação de nossa contribuição.

Considere um sistema de vendas para uma livraria on-line. Este sistema define uma classe `Book`, que dispõe de dois atributos: `title` e `price` (preço de venda do livro) – os demais atributos foram removidos para fins de simplificação. O código a seguir apresenta um recorte da classe em Java:

```
public class Book
{
  private String title;
  private float price;
  ...
}
```

Um possível cenário envolve aplicações externas clientes, que recebem tais objetos como resposta a consultas. Neste caso, a sintaxe Java está limitada a definir o preço como um valor de ponto flutuante. Dentre as opções para preencher a lacuna de semântica extra está o recurso *Java Annotations*.

O esquema de anotações é construído fazendo-se uso de um mecanismo derivado do sistema de classes e suas interfaces do Java. Anotações são objetos cuja estrutura é definida por declarações `@interface`. Por exemplo, para se criar um tipo de anotação denominado `UnitOfMoney` (unidade de dinheiro):

```
public @interface UnitOfMoney
{
  public String name();
  public String prefix();
}
```

Diversos elementos de um código Java podem ser anotados: interfaces, atributos, métodos, parâmetros, construtores, enumerações, pacotes, variáveis locais e, inclusive, a próprias anotações Java (meta-anotações). Para se anotar qualquer um dos elementos, basta precedê-lo com uma instância de anotação, caracterizada pelo símbolo de @, sucedida pelo tipo da anotação e o valor de seus campos entre parênteses. A seguir, a anotação do atributo `price`, indicando que o preço será em Real:

```
@UnitOfMoney(name="Real", prefix="R$")
private float price;
```

O tipo de anotação `Language` aplicada ao atributo `title` indica que o título está em Português:

```
@Language("portuguese")
private String title;
```

As informações extraídas destas anotações podem ser usadas para diversos fins, sendo uma das mais usuais a geração de código-fonte como parte de *frameworks*. Um exemplo disso é a especificação *JSR-181: Web Services Metadata for the Java Platform* [9]. Nela, anotações permitem o registro de dados adicionais relacionados a serviços Web e facilitam na geração automática de *Web Service Description Language – WSDL* [10] – padrão aberto para especificação de interfaces de serviços Web.

B. Digital Content Component (DCC)

A tecnologia de *Digital Content Components* foi usada como cenário prático para a aplicação da nossa proposta de anotação. Por esta razão, esta subseção apresenta os fundamentos do assunto. O DCC é uma unidade auto-descritiva que encapsula qualquer tipo de conteúdo [4], [5]. Seu modelo subjacente é uma convergência do modelo de componentes de software com o modelo de objetos digitais complexos, proveniente do domínio de bibliotecas digitais, no sentido de suportar tanto conteúdo digital quanto software executável na mesma unidade básica. A estrutura do DCC é composta de quatro partes: (1) o conteúdo propriamente dito; (2) uma estrutura de gerenciamento escrita em XML; (3) uma ou mais interfaces descritas utilizando o padrão WSDL associado ao SAWSDL ou OWL-S, para a descrição semântica dos serviços; (4) metadados em OWL usados para descrever as funcionalidades, restrições de uso e relacionamentos com outros DCCs. Para dar suporte a execução dos DCCs, tem-se a infraestrutura chamada Fluid Web [4]. Essa infraestrutura provê um *framework* de

suporte à execução, distribuição e gerenciamento dos componentes. Todo processo de comunicação entre DCCs depende fortemente da descrição de suas interfaces, principalmente quando estes interoperam em ambientes externos. Por esta razão, a descrição semântica das interfaces utilizando padrões abertos é essencial. Adicionalmente, DCCs são armazenados em repositórios que permitem o seu compartilhamento e reúso. Os metadados descritivos destes componentes são essenciais para a sua descoberta. Até o momento, a descrição semântica dos componentes e das interfaces vinha sendo realizada manualmente, após a construção dos componentes. Neste contexto, surge a necessidade de acoplar a descrição semântica dos DCCs e de suas interfaces ao processo de desenvolvimento de software, o que deu origem ao tema desta pesquisa.

C. Serviços Web Semânticos

Conforme descrito anteriormente, os DCCs utilizam a WSDL [10] – para descrever suas interfaces. Esse padrão é utilizado para descrição de serviços Web e sua estrutura tem foco na interoperabilidade sintática e estrutural fornecida pelas especificações XML, que são utilizadas na sua representação. Tal como acontece na Web Semântica, em que as linguagens RDF e OWL atuam sobre o XML, proporcionando interoperabilidade semântica, é possível estender descrições WSDL utilizando RDF/OWL. Isto torna possível o enriquecimento das descrições com termos vindos de modelos de domínio, incluindo vocabulários, taxonomias e ontologias [11].

Dentre os padrões propostos, as principais recomendações do W3C são: Ontology Web Language for Services (OWL-S) [12], Web Service Modeling Ontology (WSMO) [13] e Semantic Annotations for WSDL (SAWSDL) [14]. Enquanto a OWL-S e a WSMO definem ontologias para a descrição e composição de serviços, a ser posteriormente associadas a serviços Web – processo conhecido como *grounding* – a SAWSDL segue o sentido inverso, partindo das descrições WSDL e associando-as a ontologias através de anotações.

III. Padrão de Anotação Semântica em Código-fonte

Esta seção apresenta as principais contribuições desta pesquisa, descrevendo nossa estratégia para anotação de código implicitamente relacionada com ontologias e a sua aplicação no desenvolvimento de software.

O acréscimo de semântica no desenvolvimento de código, bem como a exploração de seus benefícios, não são um tema novo em computação. Segundo [15], em linguagens orientadas a objetos a estrutura de classes pode ser tratada sob duas perspectivas: como uma técnica arquitetural com estrutura modular e como tipos da linguagem de programação. Nesta última, o uso da classe pode conferir maior semântica ao código, independente do seu papel como estrutura modular.

Retomando o exemplo apresentado na subseção II-A, além do papel estrutural da classe `Book`, ela também agrega semântica ao objeto. Qualquer aplicação que receba o objeto pode imediatamente identificar seu papel a partir da classe. Considerando que o atributo `price` da classe `Book` seja redefinido da seguinte maneira:

```
public class Book
{
  private String title;
  private MonetaryValue price;
  ...
}
```

Anteriormente, o atributo `price` foi declarado apenas como sendo `float` e uma aplicação que usasse este objeto só seria capaz de perceber este atributo como um número real, sem nenhuma semântica adicional. Torná-lo uma instância da classe `MonetaryValue` agregou semântica ao objeto. Isto permite aprimorar a forma como as aplicações tratam este atributo. Por exemplo, o simples fato de pertencer à classe `MonetaryValue` possibilita uma checagem de tipo mais consistente por parte do cliente que recebe o objeto.

Tal solução tem um efeito colateral. Classes em linguagens de programação não foram feitas apenas para agregar semântica ao código. Seu uso implica em um *overhead* tanto de memória quanto de processamento. Outro aspecto importante consiste no fato de que as classes em linguagens de programação possuem limitações relacionadas ao seu papel nas estruturas de programação. Por exemplo, a maioria das linguagens de programação – tal como o Java – não aceita herança múltipla. Apesar disto ter uma função importante na forma como as classes reusam a estrutura e o código de outras, do ponto de vista da semântica isto limita a sua expressividade.

As anotações, introduzidas na subseção II-A, podem ser usadas para agregar semântica a partes do código. Isto evitaria o *overhead* citado anteriormente. Anotações estruturadas, da maneira em que foram realizadas no exemplo da subseção II-A, são bastante úteis para o consumo humano. Entretanto, como utilizam conteúdo tipo “string livre” para o registro dos campos das anotações – a exemplo dos campos `name` e `prefix` em `UnitOfMoney` – esta abordagem traz problemas para o processamento automatizado.

Strings livres são passíveis de erro e não estão sistematicamente sujeitas a vocabulários controlados. A ausência destes vocabulários controlados, tais como aqueles adotados por ontologias ou taxonomias, resultam em problemas de: (i) ambiguidade, (ii) duplicidade e (iii) ausência de um mecanismo pré-definido para registro de semântica dos termos. O problema (i) ocorre quando um termo pode ser interpretado de mais de uma maneira. O problema (ii) envolve mais de um termo para representar o mesmo conceito – por exemplo, “portuguese” e “português” podem representar a mesma língua. A duplicidade não necessariamente é evitada com vocabulários controlados, porém, quando associados a ontologias, termos que se referem ao mesmo conceito podem ser ligados por uma relação de equivalência. O problema (iii) se apresenta principalmente quando acontecem erros, ou se introduzem novos termos desconhecidos pelo sistema, e.g., é possível que o sistema não seja capaz de identificar a que língua remete o termo “tupi”. Termos associados a taxonomias e ontologias definem

mecanismos padronizados, e interpretáveis por computadores, para explicitar a sua semântica.

Um vocabulário e/ou taxonomia embutidos no programa, mesmo que declarados na forma de bibliotecas, de modo que possam ser compartilhados por outros programas na mesma linguagem (e.g. Java), têm sua interpretação e consequente uso restritos àquela linguagem. Quando o vocabulário é armazenado em um repositório, ele pode seguir padrões abertos; pode ser gerenciado e atualizado por aplicações especializadas; pode se relacionar com outros vocabulários/ontologias e não exige a recompilação de código para cada modificação no vocabulário. Outra vantagem diz respeito à interoperabilidade de programas que publicam serviços para terceiros. Se anotações em seus metadados e interfaces utilizam padrões abertos, programas externos são capazes de interpretar a semântica das anotações sem ter acesso a estruturas internas do sistema.

Estas observações motivaram a concepção de nossa estratégia para relacionar anotações estruturadas em Java a padrões abertos da Web Semântica. Como detalharemos adiante, ela parte de trabalhos relacionados nesta direção, mas inova em dois aspectos: unificando duas vertentes de anotação em um modelo homogêneo; sistematizando e simplificando o processo de anotação a partir do mecanismo de meta-anotações.

Há duas principais vertentes que fazem uso de associação de código a ontologias, as quais classificamos como: alinhamento e descrição. Elas variam em propósito e estratégia. Estruturas de dados utilizadas por programas de computador podem ser *alinhadas* com ontologias de tal maneira que os dados presentes nestas estruturas – em memória ou em disco – possam ser interpretados sob a ótica das ontologias. Por outro lado, o programa de computador pode ser objeto de *descrição* através de metadados associados a ontologias. A seguir apresentamos nossa estratégia unificada que abarca ambas as vertentes. Cada uma das subseções a seguir apresenta uma delas.

A. Alinhando código-fonte com ontologias

O alinhamento envolve estabelecer uma equivalência entre estruturas de dados da linguagem de programação e estruturas de uma ontologia através de anotações. Isto acontece especialmente naquelas classes que representam objetos de dados na linguagem de programação. Por exemplo, uma classe `Person` em Java pode ser alinhada com uma classe `vCard` da ontologia vCard [16], que representa os dados de uma pessoa. Tal alinhamento permite uma posterior interpretação semanticamente mais rica dos dados tratados pelo programa.

Para fazer o alinhamento entre a semântica do código Java com a Web Semântica, foi criada a anotação estruturada `@SemanticMap`:

```
public @interface SemanticMap {
  public String value();
}
```

A anotação estruturada `@SemanticMap` é composta por um campo do tipo *String*, que conterá a URI do elemento semântico associado à parte do código anotada. A solução proposta neste trabalho vai além da simples referência a elementos de uma ontologia. Foi definida uma política de associação de anotações a elementos de ontologias, que disciplina o tipo de associação a ser feita e define um mapeamento entre estruturas do código Java com elementos de uma ontologia. Tais estratégias foram desenvolvidas a partir de um estudo comparativo de trabalhos relacionados, como será apresentado em V. A seguir são detalhadas duas das principais estratégias de associação:

Classes Java são mapeadas para classes OWL

Um tipo de anotação `@SemanticMap` é associado a uma classe ou interface Java e deve relacioná-las a uma classe RDF (`rdf:Class`) ou OWL (`owl:Class`) na ontologia. Consequentemente, instâncias da classe Java serão interpretadas como instâncias da respectiva classe da ontologia (*Individuals*).

Retomando o exemplo do sistema de vendas de livros, a classe `Book` pode ser alinhada a uma classe de mesmo nome na ontologia *SUMO (Suggested Upper Merged Ontology)* [17]:

```
@SemanticMap("http://www.ontologyportal.org/
 SUMO.owl#Book")
public class Book { ... }
```

Como pode ser observado neste exemplo, URIs são de difícil leitura para o ser humano. Por esta razão, criamos uma estratégia específica para o registro de *Namespaces* [18], passível de ser usada em qualquer referência a uma URI, baseada no mesmo princípio amplamente utilizado na Web Semântica. Ela faz a associação de prefixos compactos aos prefixos da URI. Para esse propósito, foi criada um tipo de anotação especializado chamado `@Namespace`:

```
public @interface Namespace {
  String[] value();
}
```

Tal como acontece em documentos da Web Semântica, a anotação `@Namespace` é declarada no escopo mais amplo em que ela se aplica, sendo recursivamente válida em qualquer sub-escopo. No nosso caso, ela será declarada no início da classe/interface como mostra o exemplo a seguir:

```
@Namespace({"sumo:<
http://www.ontologyportal.org/SUMO.owl#>"})
@SemanticMap("sumo:Book")
public class Book { ... }
```

A declaração associou o prefixo `sumo:` ao prefixo de URI http://www.ontologyportal.org/SUMO.owl#. Todos os exemplos subsequentes adotarão *namespaces* mas, para fins de simplificação, omitiremos a declaração dos mesmos.

Atributos Java são mapeados para propriedades OWL

Atributos, campos e parâmetros de métodos Java são alinhados pelo `@SemanticMap` a propriedades de uma ontologia.

No caso do RDF, são associados ao `rdf:Property` que engloba qualquer tipo de propriedade; para o OWL, são associados ao `owl:DataProperty` ou `owl:ObjectProperty`. A seguir é apresentado tal alinhamento para atributos.

Evoluindo o anterior, foram alinhados os atributos `title` e `author` na classe `Book` com as propriedades `title` e `creator` do vocabulário *Dublin Core* [19].

As propriedades do *Dublin Core* são do tipo `rdf:Property` e não restringem o tipo de valores que aceitam através do `rdf:Range`. Isto lhes confere grande liberdade na atribuição de valores. Entretanto, usualmente a propriedade RDF ou OWL restringe seus valores. Por exemplo, o campo `price` foi alinhado à propriedade `fin:price` da ontologia *LSDIS Finance* [20], conforme o código a seguir:

```
@SemanticMap("sumo:Book")
public class Book {

  @SemanticMap("dc:title")
  private String title;

  @SemanticMap("dc:creator")
  private String author;

  @SemanticMap("fin:price")
  private MonetaryValue price;
}
```

A propriedade `fin:price` é uma `owl:ObjectProperty` e seu `rdf:range` requer objetos da classe `sumo:CurrencyMeasure`, que representam valores em dinheiro. Em se tratando de uma `owl:ObjectProperty`, como a do exemplo, o alinhamento do tipo é feito indiretamente, através do alinhamento da classe em que é declarado o atributo em Java, conforme pode ser observado no exemplo:

```
@SemanticMap("sumo:CurrencyMeasure")
public class MonetaryValue { ... }
```

O alinhamento de um atributo Java com uma `owl:ObjectProperty` requer que o tipo do respectivo atributo seja uma classe Java. No exemplo, o atributo `price` é da classe `MonetaryValue` que, por sua vez, é mapeada para a classe OWL `sumo:CurrencyMeasure`.

O mesmo mecanismo aqui ilustrado para atributos vale para campos Java e parâmetros de métodos. Em Java os campos (eventualmente também chamados de propriedades) são publicados na forma de métodos manipuladores (prefixos `get` e `set`). Por esta razão, o mapeamento é feito sempre a partir do método `get` associado ao campo, por ser este obrigatório.

B. Descrevendo código-fonte com ontologias

Em muitos casos, o objetivo final do relacionamento de anotações com ontologias não é mapear estruturas de dados do programa, mas usar elementos da ontologia para descrever aspectos, estruturas e processos do programa. A diferença essencial está no fato de que não se buscarão equivalências, já que os elementos da ontologia estão sendo usados como base de metadados que descrevem o programa em si. Tais metadados subsidiarão operações de gerenciamento de código – tal como indexação de componentes em um repositório de software – como também a construção de ferramentas capazes de explorar a semântica das descrições em operações de descoberta, recomendação e `matching` de módulos. Para fazer a *descrição* de um elemento de código Java alinhada com ontologias foi criada a anotação estruturada `@SemanticDescribe`:

```
public @interface SemanticDescribe {
  String ref() default "";
  String type() default "";
  PV[] value() default {};
}
------------------------
public @interface PV {
  String property();
  String value();
}
```

Esta anotação envolve um ou mais objetos de anotação `@PV` (sigla indicando *property-value*), que, por sua vez, é composta pelos campos: `property` e `value`. O modelo da anotação é baseado na lógica descritiva do RDF, que trabalha com a tripla (recurso, propriedade, valor), em que recurso é a entidade que está sendo descrita, através de uma propriedade com um valor específico.

Além de agregar um conjunto descritivo de propriedades/valores, cada anotação `@SemanticDescribe` pode representar a instância de uma classe RDF/OWL através de seu campo `type`. Por ser uma propriedade com valor *default*, `type` é opcional, o que permite a descrição sem a instanciação de classe.

O exemplo a seguir ilustra o `@SemanticDescribe` aplicado à descrição de um componente de software. Ele foi associado a uma instância da classe OWL `sw:Component`, dando valor às suas propriedades:

```
@SemanticDescribe(type = "sw:Component",
value = {
@PV(property ="dc:creator",
    value ="mailto:horacio@mail.com"),
@PV(property ="dc:title",
    value="Monetary Conversion") })
public class MonetaryConversion { ... }
```

Em alguns casos, a descrição completa já está disponível em um recurso externo ao código, na forma de instância de uma classe. A anotação, neste caso, pode apontar para o endereço da instância já pronta. Por exemplo, se a instância descritiva do componente `MonetaryConversion` apresentado estiver em um arquivo OWL na URI: `java:MonetaryConversion` (em que `java:` é um *namespace*). O campo `ref` tem a função de

apontar para tais recursos externos:

```
@SemanticDescribe(
    ref="java:MonetaryConversion")
public class MonetaryConversion { ... }
```

Como se pode observar, à medida em que se busca enriquecer as descrições elas se tornam mais complexas ao programador que as aplica. Há dois aspectos que contribuem para este efeito: (i) o esquema das descrições não está explícito para o programador em seu ambiente de desenvolvimento – como são esquemas definidos por ontologias, o programador teria que buscá-los externamente e ser capaz interpretá-los; (ii) ferramentas associadas ao desenvolvimento Java também não são capazes de interpretar o esquema, e não podem dar suporte ao seu uso, nem verificar sua consistência.

Adicionalmente, lidar com ontologias na Web Semântica envolve interagir com padrões e metáforas aos quais o programador usualmente não está habituado. Estas observações motivaram a formulação de uma estratégia baseada em meta-anotações, que combina a abordagem de descrições com a de alinhamentos para simplificar o processo.

C. Meta-anotações e Ontologias

Em experiências práticas no contexto de autoria de textos – detalhadas em [21] – desenvolvemos uma metodologia de anotação concomitante à criação de conteúdo. Ela baseia-se no uso de padrões de anotação que implicitamente são associados a ontologias, sem que seja necessário que o autor tenha consciência de tal associação. Esta é a essência da metodologia denominada Semântica In Loco [22] e que neste trabalho foi aplicada no contexto de programação.

Para tornar implícito o mapeamento entre as anotações do programador e as ontologias, são usadas as meta-anotações. O processo de anotação é organizado em duas etapas, executadas por atores (programadores) cumprindo papéis distintos. A primeira etapa envolve o projeto e implementação dos padrões de anotação, que tem como resultado a especificação das meta-anotações. A segunda etapa envolve o uso destas meta-anotações como ferramenta descritiva durante a construção de código. A seguir estas etapas são detalhadas:

Especificação de Meta-anotações

Os atores envolvidos nesta etapa devem ser capazes de lidar tanto com desenvolvimento em Java quanto com ontologias. Eles serão responsáveis pela seleção das ontologias envolvidas nas descrições, bem como pelo projeto e implementação das meta-classes que alinham anotações descritivas Java a elementos destas ontologias.

Ao invés de ter como alvo de alinhamento as estruturas do programa, aqui as meta-anotações alinham anotações estruturadas Java com classes RDF/OWL. Por exemplo, considere uma anotação estruturada `@Component` a ser usada na descrição de componentes. A especificação desta anotação (meta-anotação) pode ser alinhada com uma classe OWL `sw:Component` da seguinte forma:

```
@SemanticMap("sw:Component")
public @interface Component {

  @SemanticMap("dc:creator")
  String author();

  @SemanticMap("dc:title")
  String title();
}
```

Cada vez que esta anotação for usada na descrição de componentes em código Java seu conteúdo será implicitamente associado à respectiva ontologia. Este mapeamento só precisa ser definido uma única vez nas meta-anotações e usado várias vezes através das anotações.

Uso das Meta-anotações

O usuário das meta-anotações é um segundo ator, que não precisa ter conhecimento do mapeamento feito nas mesmas. Ao contrário, ele só precisa estar ciente das classes de anotação em Java e o modo de usá-las. O exemplo a seguir mostra o uso da anotação `@Component` especificada anteriormente:

```
@Component(author="mailto:horacio@mail.com",
         title="Monetary Value")
public class MonetaryConversion { ... }
```

É importante notar que o usuário da anotação não faz qualquer referência à ontologia. Toda vez que o programador utiliza a anotação `@Component` ele está implicitamente criando uma instância de `sw:Component` em OWL. Ao dar valores às propriedades `author` e `title` da anotação Java, ele está implicitamente dando valor às propriedades `dc:creator` e `dc:title` da instância OWL respectivamente.

IV. Anotando um DCC

Conforme detalhado na subseção II-B, o DCC é dividido em quatro partes: (i) conteúdo, (ii) arquivo XML contendo sua estrutura, (iii) interface em WSDL adicionado de semântica (SAWSDL ou OWL-S) e (iv) metadados OWL.

O padrão de anotação criado neste trabalho foi usado no processo de geração automática dos itens (iii) e (iv). Isso é possível através da extração, em tempo de execução, dos dados contidos nas anotações semânticas descritas anteriormente. Originalmente, a codificação de DCCs em Java exigia que, após a implementação de código, se utilizasse uma ferramenta de anotação OWL, tal como o Protégé, para descrevê-lo semanticamente. Uma das contribuições deste trabalho foi criar um procedimento automatizado de geração de tal descrição a partir de anotações de código. Neste processo são combinadas as técnicas de meta-anotação e anotação. Os metadados descritivos de um DCC são registrados a partir de uma meta- anotação definida a partir da classe de anotação @DCC:

```
@Namespace({"dcc:<
http://purl.org/dcc/dcc.owl#>",
         "dcctax:<
http://purl.org/dcc/dcctaxonomy.owl#>"})
@SemanticMap("dcc:DCC")
public @interface DCC {

  //Identificação única do componente
  @SemanticMap("dcc:oid")
  String oid();

  //Liga o DCC à sua taxonomia em OWL
  @SemanticMap("dcc:dcctype")
  String type() default "dcctax:ProcessDCC";

  //Conj. de interfaces providas pelo DCC
  @SemanticMap("dcc:provides")
  String[] provides() default {};

  //Conj. de interfaces requeridas pelo DCC
  @SemanticMap("dcc:requires")
  String[] requires() default {};
}
```

A anotação estruturada `@DCC` em Java foi associada à classe `dcc:DCC` em OWL, que é utilizada para a descrição de um DCC. Cada um dos campos da anotação Java foi associado a uma propriedade OWL.

Um DCC precisa estar associado à ontologia taxonômica de DCCs que é definida em OWL. Diversos aspectos da ação dos DCCs estão ligados a esta taxonomia. A propriedade `dcc:dcctype` realiza esta conexão com a taxonomia. Para fins de ilustração a Figura 1 apresenta um recorte da taxonomia de DCCs.

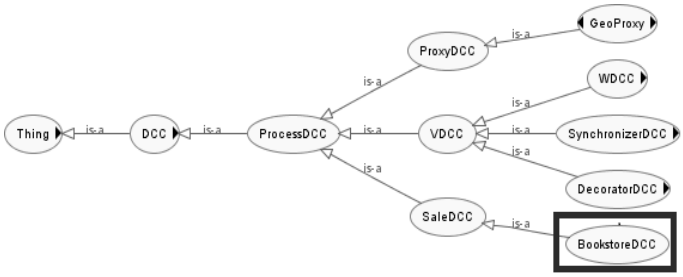

Figura 1. Recorte da Taxonomia dos DCCs.

A seguir, é apresentado o código do DCC criado para realizar a busca de livros a partir do título, autor e assunto:

```
@DCC(oid = "mydcc:examples.Bookstore",
type = "dcctax:BookstoreDCC",
provides = "mydcc:examples.IBookstore")
public class Bookstore
      implements IBookstore {
  public float searchBook(String title,
    String author){ ... }
}
```

Não é necessário adicionar a anotação estruturada `@SemanticDescribe()` no componente `Bookstore`, nem mapear semanticamente os parâmetros e retorno do método `searchBook`. O uso da anotação estruturada `@DCC` corresponde implicitamente à associação com a ontologia do DCC e o vínculo semântico para os parâmetros e retorno do método deve ser feito na interface `IBookstore` implementada pelo componente. Cada interface deve ser anotada pela meta-anotação `@DCCInterface`, apresentada a seguir:

```
@Namespace({"dcc:<
 http://purl.org/dcc/dcc.owl#>",
         "sawsdl:<
 http://www.w3.org/ns/sawsdl/>"})
@SemanticMap("dcc:Interface")
public @interface DCCInterface
{

  //Identificação única da interface
  @SemanticMap("dcc:oid")
  String oid();

  //Correspondência com o SAWSDL
  @SemanticMap("sawsdl:modelReference")
  String ref() default "";
}
```

Para este exemplo considere que `IBookstore` é uma interface para um serviço mais amplo de compra de livros usando micro-pagamentos eletrônicos (só está listada uma operação para fins de simplificação).

```
@DCCInterface(
 oid="mydcc:examples.IBookstore",
 ref="unspsc:43223210")
public interface IBookstore
    extends ISupports
{
@SemanticDescribe(ref="unspsc:43232609")
public @SemanticMap("fin:price")
    float searchBook(
      @SemanticMap("dc:title")
      String title,
      @SemanticMap("dc:creator")
      String author
);
}
```

A Figura 2 demonstra este mapeamento aplicado ao desenvolvimento de DCCs.

As informações contidas em `@DCC` e `@DCCInterface` são extraídas para geração automática dos metadados OWL. Enquanto que as descrições usando `@SemanticDescribe` e o mapeamento semântico feito via `@SemanticMap` na interface `IBookstore` serviram de base para a geração automática da interface WSDL anotada com SAWSDL. A classe `Book` foi mapeada para um tipo complexo no *XML Schema* [23] e as informações da interface foram mapeadas

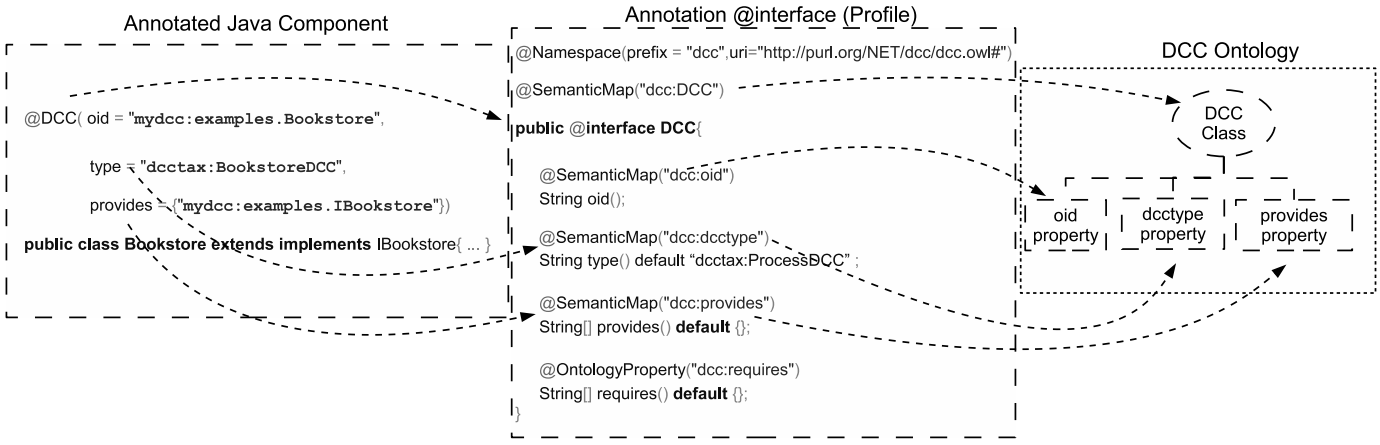

Figura 2. Meta-anotações mapeando um DCC para ontologias.

para a descrição da interface no WSDL.

Atualmente, está implementada uma ferramenta que faz a geração automática das interfaces WSDL e SAWSDL – bem como os tipos complexos XML associados – a partir das anotações. O fragmento código em SAWSDL a seguir foi gerado a partir do componente `Book` anotado:

```
...
<xsd:element name="Book" type="TBook" />
<xsd:complexType name="TBook"
sawsdl:modelReference="&cyc;Book">
<xsd:all>
<xsd:element name="author"
    type="&xsd;#string"
    sawsdl:modelReference="&dc;author"/>
<xsd:element name="title"
    type="&xsd;#string"
    sawsdl:modelReference="&dc;title"/>
<xsd:element name="subject"
    type="&xsd;#string"
    sawsdl:modelReference="&dc;subject"/>
<xsd:element name="price"
    type="&xsd;#float"
    sawsdl:modelReference=
    "&cyc;MonetaryValue"/>
</xsd:all>
</xsd:complexType>
-----------------------
<wsdl2:interface
    name="IBookstore"
    sawsdl:modelReference="unspsc:43223210">
<wsdl2:operation
    name="searchBook" ...
    sawsdl:modelReference="unspsc:43232609">
<wsdl2:input
    element="ns:searchBookRequest"
    wsaw:Action="urn:searchBook"/>
<wsdl2:output
    element="ns:searchBookResponse"
    wsaw:Action="urn:searchBookResponse"/>
</wsdl2:operation>
</wsdl2:interface>
```

Algoritmos de descoberta de DCCs, baseados na sua similaridade a partir da descrição de seus metadados e interfaces podem ser aplicados no processo de busca, conforme está descrito em [24].

V. Confrontando com Trabalhos Relacionados

[25] faz uma análise do uso de ontologias em diversas tarefas da engenharia de software e define o emprego das ontologias na etapa de implementação através de três abordagens. A primeira é relacionada à geração de código-fonte a partir de ontologias, ou seja, possibilidade da geração automática de código a partir de esquemas de ontologias, com estruturas de dados que replicam a estrutura da ontologia e módulos que permitem a serialização e deserialização dos dados, manipulados pela aplicação no formato das ontologias. A segunda abordagem diz respeito à utilização das ontologias no comportamento do software em tempo de execução. Neste contexto, estão os serviços descritos semanticamente e outras estratégias afins. A terceira abordagem envolve a utilização de das ontologias como parte da implementação lógica, através do uso de regras de inferência. Como exemplos da primeira abordagem têm-se: (i) a proposta de [26] fazendo mapeamento direto de ontologias OWL para Java; (ii) as propostas de [27] e [28] que utilizam o mapeamento objeto-relacional como ponte entre o modelo de ontologias de domínio e o código Java.

[26] faz um mapeamento da linguagem OWL para a linguagem Java, em que cada classe OWL é mapeada para uma interface Java, contendo as declarações dos métodos de acesso para as propriedades definidas pela classe. Múltiplas interfaces podem ser associadas a uma classe que as implementa, permitindo a emulação do modelo de herança múltipla aceita pelo OWL. Como propriedades Java só podem estar associadas a um tipo, enquanto propriedades OWL podem estar associadas a mais de um tipo, cada propriedade OWL multi-tipo é mapeada para uma propriedade Java do tipo `List`.

[27] e [28] utilizam as estratégias e padrões derivados do mapeamento objeto-relacional para fazer uma ponte da ontologia de domínio OWL com a linguagem Java. [27] definem uma arquitetura em três camadas: (i) modelos OWL para expressar semântica de forma interoperável; (ii) a tecnologia de componentes distribuídos *Enterprise JavaBeans (EJB)* para desenvolvimento da aplicação do usuário final e (iii) banco de dados relacionais para persistência. Tanto a parte de interface de programação EJB quanto os mapeamentos objeto-relacionais são gerados automaticamente a partir da ontologia, com uso das seguintes convenções: (1) Toda classe OWL é representada como uma entidade e atribuída a uma tabela do banco de dados. (2) O mapeamento das propriedades OWL depende de sua categoria: (a) Propriedades do tipo *DataProperty* em OWL definem valores literais, cujos tipos são definidos pelo XML Schema. Estas propriedades são mapeadas para atributos de uma tabela. (b) Propriedades do tipo *ObjectProperty* em OWL têm como valores instâncias de outras classes OWL. Estas propriedades são mapeadas para relacionamentos entre tabelas. A depender da cardinalidade das propriedades, são criadas tabelas associativas para persistência de dados.

[28] definem uma estratégia semelhante à [27] para mapeamento entre ontologias e Java. Entretanto, diferentemente de [27], a proposta de [28] trabalha com o formato de tuplas RDF. O objetivo é aproveitar padrões de projetos usados no mapeamento objeto-relacional e adaptá-los para que sejam usados para mapeamentos complexos objeto-tuplas. Soluções atuais que derivam do mapeamento objeto-relacional – *Sommer* (https://sommer.dev.java.net/), *RDFReactor* [29] e OntoJava [30] – o fazem partindo da estaca zero para resolver problemas específicos, sem aproveitar toda a experiência acumulada nos padrões objeto-relacionais. Para direcionar na tradução do Java para OWL, [28] fazem uso de *Java Annotations*. Como nem toda classe, método ou atributo Java têm que obrigatoriamente ter uma equivalência com elementos OWL, as anotações Java são usadas para anotar apenas os elementos que possuem a equivalência semântica [28].

Apesar dos estudos relatados acima servirem como base para este trabalho, há algumas diferenças significativas. A maior parte das diferenças se deve ao fato de que muitos trabalhos buscaram uma equivalência de poder de expressão entre a orientação a objetos e o OWL. Esta pesquisa vai a uma direção diversa. O programa é alinhado com o OWL justamente porque este tem maior poder de expressão semântica. Neste sentido, ao invés de equiparar semanticamente o programa e a ontologia, o programa mantém sua limitação de expressividade e aponta para ontologias mais expressivas.

Outra diferença está na abordagem unificadora aqui proposta. Enquanto algumas das abordagens se concentram no alinhamento de estruturas, outras estão voltadas à descrição do código. Nossa proposta unifica ambas as abordagens em um modelo homogêneo.

A metodologia que apresentamos utiliza o conceito de meta-anotação para realizar uma associação implícita de anotações estruturadas com ontologias. Até onde sabemos, este é um conceito original, que não encontramos em outras abordagens.

Além do uso das ontologias na etapa de implementação, outros pontos citados na análise feita por [31] são relevantes para essa pesquisa. O uso das ontologias para integração de sistemas a partir dos serviços Web semânticos e o uso de ontologias para ajudar na evolução de sistemas. Os serviços Web, que são anotados semanticamente, facilitam a automação da descoberta, composição, invocação e monitoramento dos serviços na Web. Em relação à manutenção de sistemas, quando o conhecimento adquirido ao longo do desenvolvimento é armazenado para futura pesquisa, o entendimento do mesmo por parte de pessoas que não fizeram parte do processo inicial é facilitado, pois é sabido que é gasto certa de 40% a 60% do tempo só para entender o software a ser mantido [32].

Nesta pesquisa, a semântica associada à descrição de interfaces de serviços Web foi utilizada para a descrição de interfaces de componentes. Deste modo, as anotações semânticas associadas a interfaces dos componentes puderam ser transformados em padrões abertos de descrição.

VI. Conclusão

Neste artigo, apresentamos nossa metodologia para a associação de anotações estruturadas em código a ontologias. Essa associação foi feita através da utilização do recurso *Java Annotation* através de duas abordagens: explícita e implícita. No primeiro caso, as ontologias foram associadas diretamente aos elementos do código-fonte de duas maneiras: descrição e alinhamento, e esta pesquisa contribuiu no sentido de apresentar uma abordagem que unifica ambas as vertentes em um modelo homogêneo. Já no segundo caso, a associação foi feita de forma indireta através de meta-anotações, que encapsularam a referência à ontologia.

Esta estratégia está alinhada com a metodologia denominada Semântica In Loco, cuja meta é justamente tornar o processo de anotação parte natural do processo do autor e integrado com as metáforas que este está acostumado. Neste caso, o autor é o programador e as metáforas correspondem ao ambiente de desenvolvimento de software e a técnica de anotações estruturadas, que já faz parte de sua linguagem de programação.

Ainda que a nossa proposta de meta-anotações realize uma ponte implícita entre anotações estruturadas e ontologias, ainda persistem algumas situações que exigem uma referência explícita e direta às ontologias. No exemplo da Seção IV, o autor, no papel de usuário das meta-anotações, ainda precisa registrar uma referência à taxonomia de tipos dos DCCs. Estamos trabalhando no sentido de também tornar tais referências indiretas e transparentes para este autor.

Diante disso, as principais contribuições deste trabalho são: (i) uma estratégia para associação de anotações de código Java a ontologias representadas em RDF/OWL, unificando as abordagens de alinhamento e descritiva; (ii) um mecanismo de mapeamento de estruturas de programação a elementos da ontologia, que permite automatizar tarefas; (iii) uma estratégia de anotação semântica implícita baseada em meta-anotações associadas a ontologias; esta estratégia é inovadora neste projeto e não é encontrada em nenhum trabalho relacionado; (iv)

um algoritmo de geração de interfaces em WSDL+SAWSDL e arquivo OWL aplicados a DCCs como resultado de processamento de *Java Annotations*.

Como trabalhos futuros pretendemos: aplicar o mesmo mecanismo a outras linguagens de programação; ampliar a automatização de tarefas na geração de DCCs baseada em anotações semânticas; adaptar a estratégia para outros domínios de aplicação.

Pesquisas em paralelo estão sendo desenvolvidas no sentido de aprimorar o mecanismo de descoberta e recomendação de componentes, baseada na descrição semântica das interfaces. Como o framework *Fluid Web* descreve aplicações como composições de componentes, está em desenvolvimento uma ferramenta que valida as composições pela verificação de *matching* entre interfaces. Tal ferramenta é capaz de descobrir e recomendar DCCs para atender ou adaptar especificações de interfaces. O processo explora algoritmos de similaridade baseados em ontologias.

AGRADECIMENTOS

Este trabalho foi parcialmente financiado por: CNPq, INCT em Ciência da Web (CNPq 557.128/2009-9), FAPESP e CAPES-COFECUB (projeto AMIB).

REFERÊNCIAS

[1] R. S. Pressman, *Engenharia de Software*, 5th ed. Rio de Janeiro: Mc Graw Hill, 2002.

[2] R. Witte, Y. Zhang, and J. Rilling, "Empowering software maintainers with semantic web technologies," in *ESWC 07: Proceedings of the 4th European conference on The Semantic Web*. Berlin, Heidelberg: Springer–Verlag, 2007, pp. 37–52.

[3] H.-J. Happel and S. Seedorf, "Applications of ontologies in software engineering," in *In 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), held at the 5th International Semantic Web Conference (ISWC 2006*, 2006.

[4] A. Santanchè and C. B. Medeiros, "A component model and infrastructure for a fluid web," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, pp. 324–341, 2007.

[5] A. Santanchè, C. B. Medeiros, and G. Z. Pastorello, "User–author centered multimedia building blocks," *Multimedia systems*, vol. 12, pp. 403–421, 2007.

[6] M. Eichberg, T. Schäfer, and M. Mezini, "Using Annotations to Check Structural Properties of Classes," in *Fundamental Approaches to Software Engineering*, ser. Lecture Notes in Computer Science, M. Cerioli, Ed. Springer Berlin / Heidelberg, 2005, vol. 3442, pp. 237–252. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-31984-9_18

[7] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The java language specification*. Prentice Hall, 2005.

[8] J. Bloch, "Jsr 175: A metadata facility for the java programming language," 2004. [Online]. Available: http://www.jcp.org/en/jsr/detail?id=175

[9] A. Mullendore, "Jsr 181: Web services metadata for the javatm platform," 2009. [Online]. Available: http://www.jcp.org/en/jsr/detail?id=181

[10] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana, "Web services description language (wsdl) version 2.0 part 1: Core language – w3c recommendation 26 june 2007," World Wide Web Consortium (W3C), Tech. Rep., 2007. [Online]. Available: http://www.w3.org/TR/wsdl20/

[11] K. Verma and A. Sheth, "Semantically annotating a web service," *IEEE Internet Computing*, vol. 11, pp. 83–85, 2007.

[12] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "Owl–s: Semantic markup for web services – w3c member submission 22 november 2004," 2004. [Online]. Available: http://www.w3.org/Submission/OWL--S/

[13] J. de Bruijn et al., "Web service modeling ontology (wsmo) – w3c member submission 3 june 2005," 2005. [Online]. Available: http://www.w3.org/Submission/WSMO/

[14] J. Farrell and H. Lausen, "Semantic annotations for wsdl and xml schema – w3c recommendation 28 august 2007," World Wide Web Consortium (W3C), Tech. Rep., 2007. [Online]. Available: http://www.w3.org/TR/sawsdl/

[15] B. Meyer, *Object–Oriented Software Construction*. Prentice Hall, 1997.

[16] H. Halpin, R. Iannella, B. Suda, and N. Walsh, "Representing vcard objects in rdf," W3C Member Submission 20 January 2010, 2010.

[17] I. Niles and A. Pease, "Towards a standard upper ontology," *Formal Ontology in Information Systems*, pp. 2–9, 2001.

[18] T. Bray, J. Paoli, C. M. Sperberg-Mcqueen, E. Maler, and F. Yergeau, "Extensible markup language (xml) 1.0 (fifth edition)," W3C, Tech. Rep., 2009.

[19] "Dublin core metadata element set, version 1.1," Tech. Rep. [Online]. Available: http://www.dublincore.org/documents/2008/01/14/dces/

[20] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma, "Meteor–s web service annotation framework," *International World Wide Web Conference*, pp. 553–562, 2004.

[21] A. Santanchè and L. A. M. Silva, "Document–centered learning object authoring," *Learning Technology Newsletter*, vol. 12, no. 1, pp. 58–61, Jan 2010.

[22] A. Santanchè, "Otimizando a anotação de objetos de aprendizagem através da semântica in loco," in *Anais do XVIII Simp. Brasileiro de Informática na Educação*, 2007, pp. 526–535.

[23] D. Carlson, *Modeling XML applications with UML: practical e–Business applications*. Addison–Wesley Professional, April 2001.

[24] A. Santanchè and C. B. Medeiros, "Self Describing Components: Searching for Digital Artifacts on the Web," in *Proc. of XX Brazilian Symposium on Databases*, 2005, pp. 10–24.

[25] D. Gasevic and N. Kaviani, "Ontologies and software engineering." in *Handbook on Ontologies*. Springer Berlin Heidelberg, 2009.

[26] A. Kalyanpur, D. J. Pastor, S. Battle, and J. Padget, "Automatic mapping of owl ontologies into java," 2004.

[27] I. Athanasiadis, F. Villa, and A. Rizzoli, "Ontologies, javabeans and relational databases for enabling semantic programming," *31st Annual International Computer Software and Applications Conference*, pp. 341–346, 2007.

[28] M. Quasthoff and C. Meinel, "Design pattern for object triple mapping." *IEEE international Conference on Services Computing*, pp. 443–450, 2009.

[29] M. Volkel, "Rdfreactor–from ontologies to programmatic data access," in *International Semantic Web Conference*, 2005.

[30] A. Eberhart, "Automatic generation of java/sql based inference engines from rdf schema and ruleml," in *Proceedings of the First International Semantic Web Conference*, 2002.

[31] D. Gasevic and D. Djuric, *Model driven engineering and ontology development*. Springer, 2009.

[32] S. L. Pfleeger, *Software Engineering: theory and practice*. Prentice–Hall, 1998.