Mapeando Objetos para Entidades de uma Ontologia Utilizando Metadados

Eduardo Martins Guerra, José Maria Parente de Oliveira e Clovis Torres Fernandes
Instituto Tecnológico de Aeronáutica, Pça Mal. do Ar Eduardo Gomes,50 - Vila das Acácias - São José dos Campos - SP

***Resumo* — O uso de ontologias em aplicações de comando e controle pode ser uma alternativa para isolar suas regras de negócio e torna-la mais flexível. Porém, a tradução de objetos para instâncias de ontologia é uma tarefa bastante trabalhosa, mesmo com o uso de frameworks especializados. Dessa forma, em uma aplicação que trabalhe com os dois paradigmas, essa pode ser uma tarefa pouco produtiva e sujeita a erros. Esse artigo apresenta o framework MentalLink, que propõe uma solução que utiliza metadados para mapear e traduzir objetos para instâncias de uma ontologia. Este artigo apresenta os conceitos envolvidos na concepção do framework, assim como sua estrutura e o estudo de caso que foi realizado.**

***Palavras-chaves* — Ontologia, Metadados, Orientação a Objetos, Framework, Comando e Controle.**

I. INTRODUÇÃO

As aplicações de comando e controle tendem a serem evolutivas por natureza, devido a otimizações e pequenas mudanças no processo que vão acontecendo ao longo do tempo. Isolar o conhecimento do processo e das regras de negócio através da definição de ontologias ajuda a fazer com que essas mudanças possam ser mais rápidas e mais simples de serem feitas. Isso é aplicável não somente para aplicações de comando e controle, mas também em outros tipos de aplicação cuja flexibilidade é um requisito importante.

Para o uso de ontologias por uma aplicação orientada a objetos é necessário que suas instâncias sejam traduzidas para o modelo de objetos da aplicação. Porém, essa tradução é uma tarefa trabalhosa, mesmo com o uso de frameworks especializados, como o Jena [1]. Em uma aplicação que trabalhe com os dois paradigmas, essa pode ser uma tarefa pouco produtiva e sujeita a erros.

Isso ocorre porque as APIs orientadas a objetos destinadas a manipulação de dados de uma ontologia trabalham com os conceitos da ontologia e não com os conceitos do negócio que está sendo modelado. Dessa forma, os atributos e as referências entre as instâncias da ontologia devem ser criados e ligados um por um. O código de transformação dos objetos as entidades da ontologia acabam ficando acoplados aos dois modelos, o que é prejudicial para a manutenção e flexibilidade da aplicação.

Como um alternativa de solução a esse problema, este artigo propõe a utilização de metadados na forma de anotações para o mapeamento entre os dois paradigmas. Para a validação da idéia foi criado o framework MentalLink [3], o qual fornece anotações que são colocadas nas classes que precisam ser traduzidas e uma estrutura de classes que a partir dos objetos de negócio anotados permite a tradução dos objetos para as instâncias de uma ontologia.

O artigo está organizado da forma que se segue. A seção 2 apresenta as principais diferenças entre o paradigma orientado a objetos e as ontologias. A seção 3 descreve o mapeamento baseado em metadados. A seção 4 apresenta o framework MentalLink e a seção 5 mostra as anotações de mapeamento. A seção 6 mostra os resultados obtidos com um estudo de caso e a seção 7 a comparação do que foi realizado com outros trabalhos. A seção 8 finaliza o artigo com algumas considerações finais.

II. ORIENTAÇÃO A OBJETOS E ONTOLOGIAS

O paradigma orientado a objetos é baseado em quatro conceitos básicos: classes e objetos, encapsulamento, herança e polimorfismo. Na orientação a objetos, os conceitos do software são abstraídos em classes, cujas instâncias são chamadas objetos. O encapsulamento caracteriza o desacoplamento da interface de uma classe do seu comportamento interno. A herança permite que uma classe herde as caraterísticas de outras e o polimorfismo, que uma instância possa assumir várias formas, de acordo com as classes que herda e interfaces que implementa.

Uma ontologia possui conceitos parecidos com o paradigma orientado a objetos. É possível a criação de classes para classificação dos conceitos e destas podem surgir subclasses que especializam e dividem esse conceito [5]. Dessas classes podem ser criadas instâncias. Também podem ser criadas propriedades aplicáveis a certos tipos de classes, o que seria equivalente aos atributos na orientação a objetos.

A principal diferença entre uma ontologia e uma estrutura de classes em um software orientado a objetos é o propósito com o qual elas são construídas. Enquanto a orientação a objetos tem o objetivo de estruturar um software, uma ontologia busca estruturar conhecimento. Devido ao próprio objetivo da ontologia, ela consegue ser mais expressiva que o modelo de objetos, principalmente se forem empregadas lógica de descrição e linguagens com fragmentos de 2a ordem. Dessa forma, apesar de ambas possuírem elementos similares, esta diferença conceitual pode levar a modelagens bem diferentes dentro de um mesmo domínio.

Em Ontolingua Tutorial [5], um documento que discute aspectos a respeito da modelagem de conhecimento, é feita uma pequena comparação entre a modelagem orientada a objetos e a modelagem de conhecimento de uma ontologia. Segundo é dito nesse documento, uma ontologia procura modelar o mundo e uma estrutura de classes se baseia em estrutura de dados. Em seguida é dado o exemplo de que uma classe que representasse uma elipse, em um software orientado a objetos, estenderia a classe que representa um círculo por questões de estrutura de dados. Em uma ontologia

a classificação seria o contrário, pois na realidade um círculo é uma elipse.

O exemplo dado no Ontolingua Tutorial [5] pode não ser adequado, pois em uma modelagem orientada a objetos bem feita a estrutura de herança respeita a regra "é-um". Segundo essa regra, todas as subclasses devem ser um subtipo de sua superclasse. A diferença entre uma estrutura de objetos e uma ontologia na realidade é bem sutil e como já foi dito está no propósito de cada uma.

Para exemplificar a diferença será utilizada uma ontologia em que existe uma classe chamada Pessoa, a qual possui subclasses chamadas Homem e Mulher. Em uma aplicação orientada a objetos, a não ser que existisse algum comportamento diferente para indivíduos do sexo masculino e feminino, haveria apenas a classe Pessoa com um atributo sexo para a diferenciação. Nada impede que a estrutura orientada a objetos possua subclasses Homem e Mulher e nem que a ontologia possua uma propriedade para a classe Pessoa que represente o sexo. A grande diferença foi que cada um utilizou a estrutura mais adequada ao seu propósito.

Em geral, as ontologias possuem hierarquias mais granulares e utilizam mais herança múltipla do que uma estrutura orientada a objetos. No paradigma orientado a objetos, quando não existe uma diferença de comportamento, é mais indicado o uso de atributos para diferenciar as instâncias. Por outro lado, em uma ontologia, que visa descrever um conhecimento, a criação de diversas classes e subclasses é mais adequada.

III. MAPEAMENTO BASEADO EM METADADOS

A criação de um mapeamento entre paradigmas utilizando metadados consiste em ter um conjunto de metadados que configure como as instâncias de um paradigma podem ser traduzidas para um outro paradigma. Existe também um componente tradutor que sabe como lidar com esses metadados para o processamento da transformação.

O padrão de projeto Mapeamento entre Representações [14], cuja estrutura está representada na Figura 1, apresenta uma estrutura que representa a estratégia de se utilizar metadados para mapear diferentes representações de um mesmo conceito dentro de uma aplicação. A classe chamada de InterfaceOutroParadigma representa uma API para o acesso a representações externas a aplicação, como em bases de dados ou documentos XML.

O framework Hibernate [6] foi um dos pioneiros na implementação dessa estratégia ao criar uma solução robusta para mapear o paradigma orientado a objetos para o paradigma relacional. Ele utiliza para armazenar os metadados um documento XML, o qual descreve como uma classe deve ser persistida em um banco de dados. Os conceitos de herança e associações, peculiares da orientação a objetos, são mapeados para o mundo relacional, com o suporte a três diferentes abordagens.

Quando a linguagem Java passou a suportar nativamente o uso de metadados na forma de anotações [7], a abordagem de mapeamento objeto-relacional como opção para a camada de interface de persistência foi padronizada na Java Persistence API (JPA) [8]. A maior diferença é que os metadados agora são na forma de anotações e, com a padronização, diversos fornecedores agora podem implementá-la.

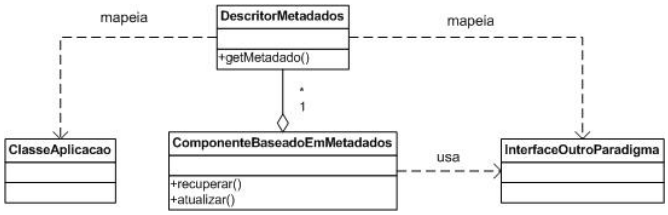

Fig 1. Estrutura do padrão Mapeamento entre Representações.

O mesmo conceito de mapeamento também é utilizado na Java Architecture for XML Binding (JAXB) [9] onde as entidades de um XML Schema são mapeadas para classes também com o uso de anotações. Nas versões iniciais dessa API utilizava convenções de nomenclatura para fazer a ligação.

O framework nacional SwingBean [10] utiliza arquivos XML para mapear como os objetos de uma classe serão exibidos e recuperados de uma interface gráfica. Apesar desse último exemplo ser diferente dos anteriores, os mesmos conceitos e a mesma estrutura é utilizada para a implementação dos conceitos..

IV. FRAMEWORK MENTALLINK

O objetivo do framework MentalLink [3] é a tradução entre as entidades de uma ontologia e as classes de uma aplicação utilizando mapeamento baseado em metadados. Dessa forma, uma aplicação pode facilmente alternar entre o paradigma orientado a objetos para a execução de comportamento e a ontologia para execução de regras de inferência. A estratégia utilizada pelo framework para o mapeamento é o uso de anotações na classe e em seus atributos ou métodos de acesso. O uso de anotações em classes é conhecido como Atributte-Oriented Programming (@OP) [15].

Na Figura 2 está representada a estrutura básica do framework, com suas principais classes. Conforme pode ser visto, a principal dependência externa é da classe Model do framework Jena. Essa classe faz o papel da interface da API para a interação com a representação do outro paradigma, no caso a ontologia.

Jena é um conjunto de ferramentas para a linguagem Java que auxilia no desenvolvimento de aplicações que utilizam conceitos de Web semântica [1]. Esse framework já possui uma grande comunidade e suas APIs já se encontram bem amadurecidas. Sua robustez já foi comprovada inclusive na sua utilização para aplicações de escala comercial [2]. Esse framework possui funcionalidades que permitem ao desenvolvedor a manipulação de modelos ontológicos, inclusive fazendo inferência com base em regras definidas.

Cada instância da classe ObjectModel, encapsula uma instância da classe Model para a manipulação das instâncias da ontologia e outra instância para a recuperação das

definições da ontologia. Essa classe recebe os objetos pelo método addObject() e realiza internamente esta tradução, adicionando as informações como instâncias da ontologia. O inverso pode ser feito pelo método getObject(), que recupera uma instância da ontologia e cria uma instância da classe correspondente, populando-a de acordo com o que foi mapeado.

O método getBaseModel() recupera o modelo base, implementação da classe Model do framework Jena, com as definições da ontologia. Da mesma forma, o método getInstanceModel() retorna o modelo com as instâncias. Para a recuperação de um modelo para a realização de inferências, existe um método chamado getInferenceModel() que realiza essa tarefa. Esse método deve ser utilizado com cuidado, pois a cada chamada ele irá gerar um novo modelo de inferência, o que pode ser um pouco demorado dependendo da ontologia.

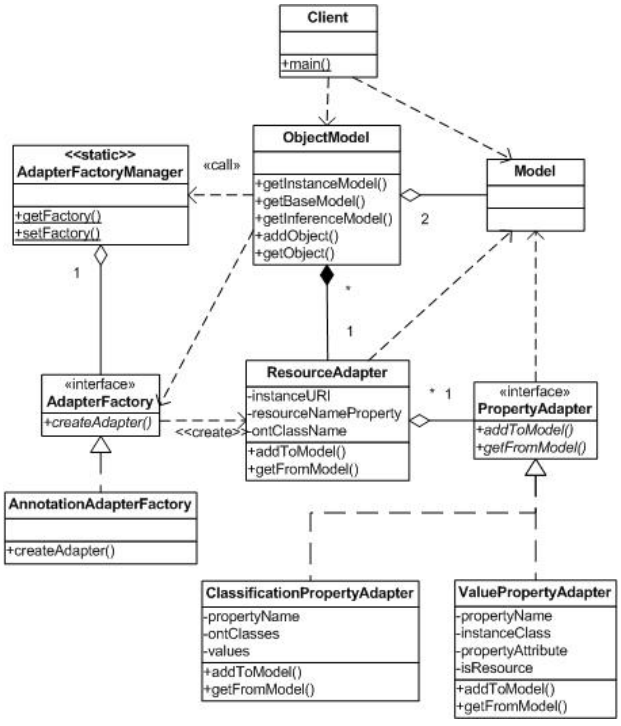

Fig 2. Estrutura básica do framework MentalLink.

O mecanismo de extração de metadados é encapsulado pela interface AdapterFactory. Até o momento a única implementação desta interface é AnnotationAdapterFactory, que cria as classes que lêem os metadados a partir de anotações na classe. Futuramente pode ser criada uma implementação para essa fábrica que leia os dados, por exemplo, em um documento XML. Os metadados serão descritos com maior detalhe na próxima seção desse artigo.

A classes ResourceAdapter e a interface PropertyAdapter representam os metadados da transformação para a inserção de entidades e propriedades. A interface PropertyAdapter possui duas implementações, ClassificationPropertyAdapter, que lida com atributos que correspondem a classes na ontologia, e ValuePropertyAdapter, que lida com atributos que são propriedades na ontologia. Todas essas classes também encapsulam a lógica para a tradução através dos métodos addToModel() e getFromModel(). Elas são criadas a partir das implementações de AdapterFactory, que é feita de forma encapsulada dentro da classe ObjectModel.

V. MAPEAMENTO DAS CLASSES PARA ONTOLOGIA

Na Seção 2 foram mostradas as principais diferenças entre o paradigma orientado a objetos e ontologias. O framework MentalLink leva em consideração essa diferença para a criação dos metadados. A Tabela 1 apresenta uma referência dos atributos dos metadados do framework que serão citados a seguir no texto.

O metadado mais básico é representado pela anotação @Resource, que se aplica a uma classe. Esse metadado relaciona uma classe Java com uma classe da ontologia, identificando dados como o nome da classe da ontologia. Toda classe mapeada deve possuir a anotação @ResourceName em algum atributo ou método de acesso. Essa anotação, que não possui atributos, identifica a propriedade que será utilizada para o nome da instância da ontologia. O nome é composto pela propriedade instanceURI de @Resource concatenada com o valor da propriedade anotada @ResourceName.

A anotação @Property é um metadado que mapeia um atributo do objeto para uma propriedade. Essa anotação pode ser aplicada a um atributo ou ao seu método de acesso (iniciado com "get") e lida com coleções de forma transparente. O valor da propriedade pode ser um literal ou uma outra instância da ontologia, nesse caso configurada pelo atributo isResource. Na chamada do método getObject() da classe ObjectModel, todo o grafo de objetos relacionado ao objeto recuperado é recuperado. O mesmo ocorre no método addObject(), o qual também cria instâncias na ontologia para todos os objetos relacionados.

TABELA I ATRIBUTOS DOS METADADOS DE MAPEAMENTO DO FRAMEWORK MENTALLINK

Metadado	Atributo	Descrição
@Resource	instanceURI	URI base para a identificação das instâncias.
	ontClass	Nome da classe correspondente na ontologia.
	baseURI	URI base para o nome da classe na ontologia.Normal
@Property	value	Nome da propriedade na ontologia.
	baseURI	URI base para o nome da propriedade na ontologia.
	isResource	Se a propriedade é uma instância ou um literal.
	instanceClass	Qual a classe Java da propriedade (usada no caso de listas)
@Classification	baseURI	URI base para o nome das classes da ontologia do atributo ontClasses.
	values	Uma lista com os valores possíveis para os atributos.
	ontClasses	Uma lista com as classes da ontologia relativas aos valores do atributo values.

A anotação @Classification mapeia um atributo do objeto que classifica a instância e que na ontologia é representada por um conjunto de classes. Essa anotação traduz uma diferença importante entre a orientação a objetos e ontologias, conforme descrito na Seção 2. O atributo values possui um array com os possíveis valores da propriedade e o atributo ontClasses um array com as classes da ontologia correspondentes.

Para ficar mais claro como o mapeamento pode ser realizado, apresenta-se um exemplo simples, mas que utiliza todas as anotações. A Figura 4 apresenta uma representação gráfica da ontologia utilizada. A principal classe é Person que possui as subclasses Man e Woman. A propriedade parentOf é aplicável a instâncias da classe Parent tendo como valor instâncias da classe Child.

```
package org.mentallink.example.ontology;

@Resource(instanceURI="http://family-instances/",
          ontClass="http://family/Person")
public class Pessoa{

    private String nome;
    private String sexo;
    private List<Pessoa> filhos =
          new ArrayList<Pessoa>();

    @Property(value="parentOf",
            isResource=true,
            baseURI="http://family/",
            instanceClass=Pessoa.class)
    public List<Pessoa> getFilhos() {
        return filhos;
    }
    public void setFilhos(List<Pessoa> filhos) {
        this.filhos = filhos;
    }
    public void addFilho(Pessoa filho){
        filhos.add(filho);
    }
    @ResourceName
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    @Classification(baseURI="http://family/",
            values={"Masculino","Feminino"},
            ontClasses={"Man","Woman"})
    public String getSexo() {
        return sexo;
    }
    public void setSexo(String sexo) {
        this.sexo = sexo;
    }
}
```

Fig 3. Exemplo de uma classe Java mapeada para uma classe de uma ontologia (importações omitidas).

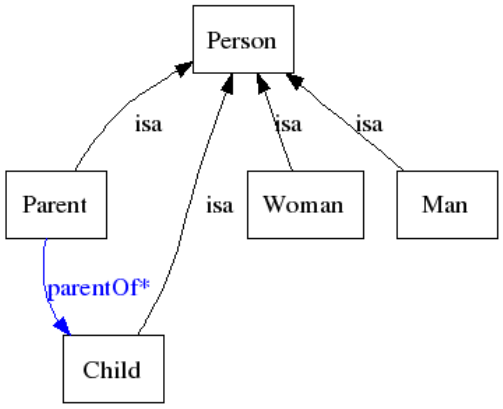

Fig 4. Representação da ontologia utilizada para o mapeamento do exemplo.

A Figura 3 apresenta como fica a classe mapeada com o framework MentalLink. A classe Pessoa recebe a anotação @Resource que a mapeia para a classe Person da ontologia. O método getNome() recebe a anotação @ResourceName, o que significa seu valor será utilizado para a criação da URI das instâncias da ontologia. A propriedade parentOf é mapeada para a lista filhos através da anotação @Property. O mapeamento para as classes Man e Woman é feito pela anotação @Classification que indica quais são os valores do atributo sexo que fazem a instância pertencer a uma das duas classes.

Como pode ser observado, não houve o mapeamento das classes Parent e Child. Apesar dessas classes não serem mapeadas explicitamente, elas podem ser facilmente inferidas com as informações mapeadas. Dessa forma, quando uma instância de Pessoa tiver alguma outra instância na lista filhos, por inferência ela também pertencerá a classe Parent. A classe ObjectModel possui o método getInferenceModel() que retorna uma implementação da classe Model do Jena com todas as inferências executadas.

As anotações já criadas no framework MentalLink ainda não conseguem representar todos os tipos de relacionamentos entre instâncias de uma ontologia. Exemplos de casos não tratados são a transitividade, simetria e reflexão. Porém ainda é possível fazer uso desses conceitos através de inferências em cima de objetos traduzidos. Em novas versões do framework, esses outros conceitos serão incorporado diretamente ao esquema de metadados.

VI. Estudo de Caso

Para validar os benefícios da utilização do framework MentalLink [3] em uma aplicação, foi desenvolvida uma aplicação acadêmica que o utilizou para a tradução entre a ontologia e o modelo de objetos. Essa aplicação se tratava do cálculo de índices ecológicos para empresas, cujas regras de combinação dos índices estavam representados em uma ontologia e os dados das empresas estavam representados em uma base de dados.

Para simplificar o entendimento da arquitetura que foi utilizada na aplicação, a Figura 5 a apresenta graficamente ressaltando os paradigmas utilizados e os frameworks utilizados na tradução das representações.

A aplicação possui um modelo de objetos que representa as principais entidades de negócio da aplicação, como as empresas e os índices. Os dados referentes as empresas são recuperados de uma base de dados utilizando o framework Hibernate [6], que faz o mapeamento objeto-relacional baseado em metadados entre as classes de domínio e as tabelas do banco de dados. Esse mesmo modelo de objetos também é mapeado para interfaces gráficas utilizando o framework SwingBean [10].

Através do framework MentalLink, as instâncias da ontologia relativas ao cálculo do índice são recuperadas e mapeadas para objetos. Com isso, consegue-se trazer para o modelo de objetos todos os relacionamentos entre as instâncias da ontologia que configuram as regras de cálculo dos índices. Com isso, conseguiu-se isolar parte da regra de negócio na definição do conhecimento utilizando uma ontologia, deixando a aplicação mais flexível.

O uso do framework MentalLink foi feito sem a necessidade de criação de novos tipos de metadados de mapeamento. Um dos benefícios obtidos com o uso do framework proposto foi o desacoplamento do modelo de objetos do modelo da ontologia. Durante o desenvolvimento a ontologia precisou ser modificada e isso teve um baixo impacto na aplicação, resultando apenas na alteração das anotações.

Outra funcionalidade que se mostrou ser bastante útil foi a recuperação automática dos relacionamentos de um objeto ao ser recuperado da ontologia. Dessa forma, todo o grafo de objetos já era construído, poupando a criação de um código complexo de ser construído, principalmente nesse caso onde existiam referências circulares.

Outro fato que vale a pena ser citado é a curta curva de aprendizado dos alunos para o uso do framework. Os alunos que implementaram a aplicação precisaram conhecer muito pouco da API do Jenna para interagirem com a ontologia. Isso tornou o desenvolvimento mais simples e produtivo, o que é o principal objetivo do framework.

VII. Trabalhos Relacionados

O framework Jena [1] é um conjunto de ferramentas que oferece uma estrutura orientada a objetos para se trabalhar com ontologias. É preciso utilizar as classes do próprio framework para representação de recursos, propriedades e classes da ontologia. O MentalLink complementa o Jena, permitindo que se trabalhe diretamente com as classes de domínio, mapeando-as através de anotações.

Dessa forma, os objetos do Jena ainda são criados, porém essa criação é feita baseada nos metadados e encapsulada dentro do framework. Com isso, a aplicação não fica acoplada com esses objetos específicos para o uso de ontologias, podendo trabalhar diretamente com suas classes de domínio.

Uma outra iniciativa de framework que utiliza a orientação a objetos para representação de conhecimento em ontologias é o O3F [11]. Porém essa iniciativa modela os tipos de conceito de uma ontologia em uma estrutura de classe, de forma até similar às classes do Jena. Esse trabalho também apresenta uma linguagem para a representação das ontologias segundo o modelo proposto. Apesar de trabalhar com orientação a objetos, o O3F define classes próprias e não trabalha diretamente com as classes de domínio da aplicação, como o MentalLink faz.

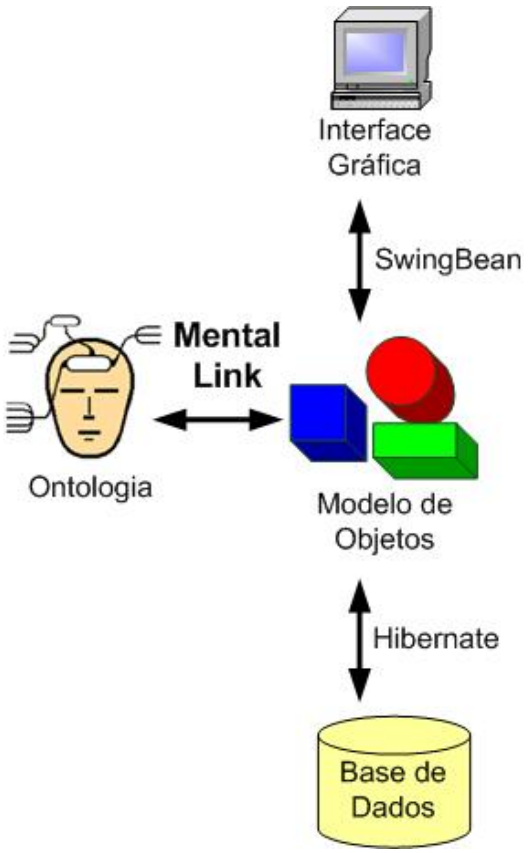

Fig 5. Representação da estrutura da aplicação utilizada no experimento.

Existem técnicas que utilizam os conceitos da orientação a objetos para a modelagem de ontologias [12] [13]. Esse estudo é complementar ao que foi apresentado neste artigo, pois quanto mais próximo de um modelo de objetos for a definição da ontologia, mais fácil e direto será o mapeamento. Recomenda-se que aplicações, que desejarem utilizar o MentalLink para mapear suas classes para uma ontologia, utilizem uma dessas técnicas para a modelagem do conhecimento.

Em relação ao mapeamento baseado em metadados, essa já é uma técnica utilizada em frameworks como o Hibernate [6] e em APIs como o JAXB [9], conforme já citado. Essa técnica foi documentada pelo mesmo autor desse artigo como um padrão de projeto chamado Mapeamento entre Representações [14] e a construção desse framework fez parte desse estudo.

VIII. Conclusão

Este artigo apresentou o framework MentalLink que introduz uma nova abordagem para o mapeamento de objetos para instâncias de uma ontologia utilizando metadados. As seguintes são consideradas as principais contribuições desse trabalho:

- Identificação das possíveis diferenças entre uma ontologia e um modelo de classes definidos para o mesmo domínio, com a definição de questões de mapeamento.
- Construção do framework MentalLink que valida a solução do mapeamento baseado em metadados para a tradução de objetos para instâncias de uma ontologia.
- Construção de um estudo de caso com o uso do framework MentalLink em uma aplicação acadêmica, na qual ele foi utilizado de forma satisfatória.

O uso do framework MentalLink traz diversas vantagens e facilidades no desenvolvimento de aplicações que precisam converter entidades de uma ontologia para objetos. A principal vantagem do uso do framework é o maior desacoplamento entre os modelos, o que traz um ganho em produtividade para a realização da conversão e flexibilidade para alteração de um dos modelos com impacto apenas nos metadados.

Na realização do experimento, os metadados do MentalLink apresentados se mostraram suficientes. Como foi citado, alguns conceitos das ontologias não foram implementados e espera-se que o uso do MentalLink em outras aplicações revele a necessidade de incorporação de novas funcionalidades e novos tipos de metadados. Porém, por mais que o esquema de metadados possa ser expandido, a sua abordagem e a sua estrutura se mostraram adequadas ao problema ao qual ele se propõe a resolver.

Um ponto que não foi levado em consideração na construção do framework foi a sua utilização em um ambiente com múltiplas threads, como uma aplicação web, por exemplo. Como o experimento foi uma aplicação para desktop, este não foi um fator limitante no uso do MentalLink Os conceitos relacionados a transações e sessões no modelo de instâncias da ontologia estão sendo estudados e já estão previstos para uma próxima versão.

IX. IMPACTO OPERACIONAL

Em aplicações de comando e controle que são evolutivas por natureza e possuem a flexibilidade como importante requisito não-funcional, o uso do framework MentalLink para se trabalhar com ontologias pode ser uma boa alternativa. Além do isolamento das regras de negócio em ontologias, o modelo de objetos e a ontologia poderiam evoluir de forma independente com o impacto apenas na definição dos metadados.

Isso possibilitaria grandes ajustes nas regras de negócio sem a necessidade de alteração da aplicação. Dessa forma, os sistemas de comando e controle poderiam incorporar novas regras e alterar as já existentes apenas com alterações na ontologia.

A criação de uma camada pouco produtiva na interface da aplicação com a ontologia poderia compensar negativamente os benefícios trazidos pelo seu uso. Nesse cenário, framework MentalLink simplificaria a implementação do mapeamento da ontologia para o modelo de objetos viabilizando o uso desse tipo de solução.

REFERÊNCIAS

[1] Carroll, J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K., *Jena: Implementing the Semantic Web Recommendations*, HP Technical Report HPL-2003-146, 2003. [http://www.www2004.org/proceedings/docs/2p74.pdf Abril 2007.]

[2] K. Portwin and P. Parvatikar. *Scaling Jena in a commercial environment: The Ingenta MetaStore project*. In Proceedings of the 2006 Jena User Conference, Bristol, UK, May 2006.

[3] MentalLink, http://sourceforge.net/projects/mentallink, Junho 2008.

[4] Davies, John, Fensel, Dieter and Harmelen, Frank *Towards The Semantic Web - Ontology-driven Knowledge Management*, John Wiley & Sons, Ltd , 2003.

[5] Farquhar, A. *Ontolingua Tutorial*. http://ksl-web.stanford.edu/people/axf/tutorial.pdf, 1997.

[6] Bauer, Christian and King, Gavin *Hibernate in Action*, Manning Publications, 2004.

[7] JSR 175: A Metadata Facility for the Java Programming Language, http://www.jcp.org/en/jsr/detail?id=175, 2003.

[8] JSR 220: Enterprise JavaBeans 3.0, http://www.jcp.org/en/jsr/detail?id=220, 2006.

[9] JSR 222: Java Architecture for XML Binding (JAXB) 2.0, http://jcp.org/en/jsr/detail?id=222, 2006.

[10] SwingBean, http://swingbean.sourceforge.net, Abril, 2007.

[11] Mota, L., L. Botelho, et al. *O_3F: an Object Oriented Ontology Framework*. 2nd International Joint Conference on Autonomous Agents and Multi Agent Systems, 2003.

[12] Denecke, Matthias *Object-oriented techniques in grammar and ontology specification*. In Proceedings of the Workshop on Multilingual Speech Communication, 2000.

[13] J. Evermann and Y. Wand, *Ontology Based Object-Oriented Domain Modelling: Fundamental Concepts*, Requirements Engineering Journal, Volume 10, Number 2, May 2005 pp. 146-160.

[14] E. Guerra, F. Pavão, C. Fernandes, Padrões de Projeto para Frameworks e Componentes Baseados em Metadados, SugarLoafPlop 2008, anais preliminares disponíveis em http://sugarloafplop.comp.uece.br/SLP2008_Preliminary_Proceedings.pdf.

[15] SCHWARZ, D. Peeking Inside the Box: Attribute-Oriented Programming with Java 1.5, In ON Java.com, O'Reilly Media, Inc., Junho 2004.