Modelagem de Conhecimento integrando Regras de Produção e Ontologias

Article

3 authors, including:

Pinheiro Vladia
Universidade de Fortaleza
44 PUBLICATIONS **157** CITATIONS

SEE PROFILE

Vasco Furtado
Universidade de Fortaleza
208 PUBLICATIONS **1,383** CITATIONS

SEE PROFILE

Modelagem de Conhecimento integrando Regras de Produção e Ontologias

Tiago Cordeiro, Vládia Pinheiro e Vasco Furtado
UNIFOR – Universidade de Fortaleza

1. Introdução

O conhecimento das organizações precisa cada vez mais ser retido e disseminado. Uma das atividades realizadas para esse objetivo é o desenvolvimento de Sistemas Baseados em Conhecimento (SBC). Estes sistemas buscam resolver tarefas que envolvem o uso de conhecimento, para as quais não existe um algoritmo determinístico que possa resolvê-las. As bases de conhecimento desses sistemas são construídas a partir da experiência e *know-how* de especialistas.

A principal atividade do desenvolvimento de SBCs é a aquisição e modelagem do conhecimento de como realizar a tarefa e de conceitos sobre o domínio, para que possam ser representados em formalismos, como regras de produção e ontologias.

Este trabalho tem como objetivo propor uma abordagem para a modelagem do conhecimento de SBCs que integra duas ferramentas: Protégé, um editor de ontologias, e JEOPS, um motor de inferência sobre regras de produção desenvolvido em Java. Com esta integração pretende-se contribuir para a facilidade e praticidade do desenvolvimento de SBCs.

2. Aquisição e Modelagem de Conhecimento

O processo de aquisição de conhecimento consiste em se obter conhecimento de uma fonte e transformá-lo em uma representação explícita, formando uma base de conhecimento. Esta fonte pode ser um especialista, documentos sobre o domínio ou um banco de dados (Vasco, 1993). Este processo é responsável pela completeza e certeza do conhecimento representado na base de conhecimento de um SBC.

A evolução do paradigma de aquisição de conhecimento se deu de uma atividade de extração, que caracterizava os primeiros sistemas especialistas, para uma atividade de modelagem. Este último foi primeiro aplicado no projeto KADS (Wielinga et al, 1992a), no qual a modelagem da expertise e a implementação do SBC são atividades separadas.

Ao final da aquisição de conhecimento temos que transformar o conhecimento, da forma como ele está disponível no mundo, para um formalismo que possa ser acessado e usado por um SBC. Existe uma série de formalismos que podem ser utilizados para representar conhecimento: sistemas de produção, raciocínio baseado em casos, redes neurais, redes probabilísticas, ontologias. Especificamente, neste trabalho, utilizamos regras de produção e ontologias, as quais são definidas a seguir:

- Sistema de produção pode ser definido como um conjunto de produções (ou regras de produção), uma memória de trabalho onde são armazenados os fatos, e um algoritmo conhecido por encadeamento progressivo (ou forward-chaining), que produz novos fatos a partir de fatos antigos.
- Ontologia é uma explícita especificação de uma ”conceitualização” (Gruber,1995), isto é, consiste de uma especificação de objetos, conceitos e outras entidades que são assumidas como existentes, além de relações entre conceitos e restrições expressas através de axiomas (Zlot et al., 2002). O uso de ontologia para representar os conceitos do domínio permite que estes conceitos possam ser reusados em várias aplicações.

3. Integração de Regras de Produção e Ontologia

A abordagem proposta prevê a integração de duas ferramentas: Protégé e JEOPS. Nesta seção apresentamos como ambas são usadas no desenvolvimento de SBCs e como uma abordagem de desenvolvimento de SBCs que integre as duas representa um ganho de produtividade em relação ao uso individual de cada uma.

3.1. Protégé

Protégé (Eriksson et al, 1999) é um software amplamente utilizado por desenvolvedores e especialistas para a construção de SBCs. Ele integra, em uma interface gráfica, ferramentas para modelagem e aquisição de conhecimento. A ferramenta de modelagem é genérica e serve ao propósito de modelar quaisquer conceitos e relações entre eles. A ferramenta de aquisição de conhecimento, gerada automaticamente a partir dos conceitos, é específica para o domínio modelado permitindo que o especialista possa inserir conhecimento declarativo sobre o domínio. A base de conhecimento resultante pode, então, ser usada por SBCs para resolver problemas referentes ao domínio.

Modelar ontologias no Protégé é uma tarefa simples. A hierarquia dos conceitos é visualizada e alterada através da representação gráfica de uma árvore. Quando um conceito é selecionado na árvore suas propriedades (*slots*) podem ser criadas ou alteradas. O Protégé oferece suporte para definição dos slots, incluindo restrições de domínio, de cardinalidade e valor padrão (*default*).

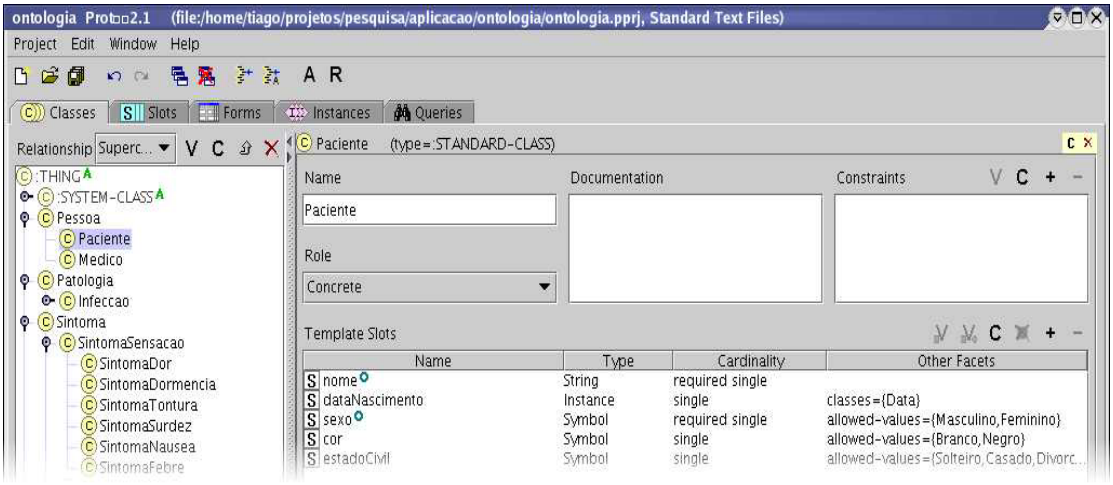

Figura 1. Exemplo de uma ontologia definida na interface gráfica do Protégé

Na figura 1, temos um exemplo de uma ontologia definida para o domínio de diagnóstico clínico. Do lado direito, temos a hierarquia de classes definidas e, do lado esquerdo, os atributos da classe selecionada (**Paciente**). A classe **Paciente** é sub-classe da classe **Pessoa** e tem os atributos **Nome**, **Documentação**, **Slots**, etc. As propriedades(*slots*) definidas para esta classe são **nome**, **dataNascimento**, **sexo**, **cor e estadoCivil**. Cada *slot* definido deve ser de um tipo primitivo, instância ou outra classe.

3.2. JEOPS

Para prover capacidade de raciocínio a uma aplicação, utilizamos o JEOPS (Java Embedded Object-Oriented Production Systems), um motor de inferência de primeira ordem com encadeamento progressivo integrado à linguagem Java (Figueira e Ramalho, 2000). JEOPS faz parte da classe de sistemas EOOPS (Embedded Object-Oriented Production Systems). Essa abordagem representa os fatos na memória de trabalho sob a forma de objetos e traz vantagens, tanto ligadas a conceitos de qualidade apresentados pela Engenharia de Software (reusabilidade, modularidade, legibilidade, manutenibilidade), como relativas à engenharia de conhecimento (representação natural de relações ontológicas).

Uma regra JEOPS é, basicamente, composta por três seções: declarações, condições e ações. A seção de declarações especifica os objetos que serão utilizados nas condições e/ou ações. As condições são qualquer expressão Java que retorne um valor *boolean*. Na seção de ações podem ser inseridos quaisquer comandos Java, no entanto estes serão executados somente se todas as condições forem verdadeiras.

O JEOPS funciona como um pré-compilador, que traduz um arquivo de regras em uma classe Java que implementa o motor de inferência de acordo com o arquivo original. Como as regras utilizam objetos, suas classes já devem estar definidas no momento da definição das regras, e devem ser implementadas como classe Java para a execução das mesmas. No Quadro 1, temos um exemplo de uma regra que imprime, na tela, o nome de todos os ancestrais de uma determinada pessoa. Esta regra será disparada para todo par de objetos **o** e **p** das classes **Objetivo** e **Pessoa**, respectivamente, que tornem as duas expressões (**o.getAlvo() == p** e **o.estaAtivo()**) verdadeiras.

```
ruleBase Familia {
  rule encontraAncestrais {
    declarations
      Pessoa p;
      Objetivo o;
    localdecl
      Pessoa pai = p.getPai();
      Pessoa mae = p.getMae();
    conditions
      p == o.getAlvo(); o.estaAtivo();
    actions
      o.desativa();
      System.out.println(pai.getNome() + " e " +
                mae.getNome() + " são ancestrais");
      insert(new Objetivo(pai));
      insert(new Objetivo(mae));
  }
}
```

Quadro 1. Base de regras que contém a regra para encontrar ancestrais

No exemplo do Quadro 1, o arquivo JEOPS só poderá ser pré-compilado se as classes **Objetivo** e **Pessoa** já estiverem implementadas, com suas propriedades e métodos, em classes Java.

No desenvolvimento de sistemas com JEOPS ocorre uma separação entre a definição (modelagem) da ontologia e a criação das regras de produção. A metodologia de desenvolvimento de SBCs usando JEOPS pode ser representada pela seqüência de passos no Quadro 2:

1. Definir os conceitos (modelar a ontologia) do domínio;
2. Implementar a ontologia em classes Java;
3. Criar as regras para resolver o problema;
4. Verificar se os conceitos definidos no passo 1 satisfazem aos requisitos das regras criadas, se não voltar ao passo 1;
5. Criar instancias das classes definidas no passo 2, inserindo-as na base de fatos do motor de inferência;
6. Executar o motor de inferência, e analisar os resultados;
7. Retornar ao passo 1, melhorando a solução.

Quadro 2. Metodologia de desenvolvimento de SBCs com JEOPS

Seguindo esta metodologia, percebemos que o tempo gasto em se implementar a ontologia do domínio em classes Java (passo 2), bem como, a interface para aquisição de instâncias é maior que o gasto na modelagem da ontologia e na definição das próprias regras de produção. Essa necessidade de "reescrever" a ontologia oferece um terreno propício para a inserção de erros e divergências em relação à modelagem original. Além disso, a ontologia do domínio precisa ser constantemente evoluída e ficamos dependentes do engenheiro de conhecimento e de um programador Java.

3.3. Integrando Protégé e JEOPS

Tendo em mente a proposta de maior rapidez e facilidade no desenvolvimento de SBCs, buscamos uma forma de fazer o JEOPS utilizar diretamente as instâncias das classes definidas na ontologia do domínio, bem como proporcionar ao engenheiro de conhecimento e especialistas uma interface para modelagem e inserção do conhecimento através do uso do Protégé.

Utilizando o Protégé como ferramenta de modelagem e aquisição do conhecimento podemos, a partir das regras JEOPS, acessar diretamente as instâncias definidas na base de conhecimento desse editor de ontologias. Para obtermos esse nível de acesso utilizamos a API (Application Program Interface) do Protégé. Desta forma, evitamos todo o retrabalho de implementação da ontologia pelo programador Java, tornando mais rápido e menos sujeito a erros o desenvolvimento das regras de produção e do SBC como um todo.

Portanto, a metodologia de desenvolvimento de SBCs foi otimizada, pois não há mais necessidade de implementação, em classes Java, dos conceitos já definidos no Protégé. As instâncias do Protégé alimentam diretamente a base de fatos do JEOPS reduzindo a metodologia aos passos descritos no Quadro 3:

1. Definir os conceitos (modelar a ontologia) do domínio no Protégé;
2. Criar as regras para resolver o problema;
3. Verificar se os conceitos definidos no passo 1 satisfazem os requisitos das regras criadas, se não voltar ao passo 1;
4. Criar instâncias da ontologia no Protégé, inserindo-as na base de fatos do motor de inferência;
5. Executar o motor de inferência, e analisar os resultados;
6. Retornar ao passo 1, melhorando a solução.

Quadro 3. Metodologia de desenvolvimento de SBCs com JEOPS e Protégé

A integração propriamente dita encontra-se na utilização, através da API do Protégé, das instâncias da base de conhecimento do Protégé diretamente na memória de trabalho do JEOPS. Para representarmos essas instâncias na base de objetos do JEOPS, implementamos a classe **Instância** que além de simplificar a tarefa de recuperação dos valores das propriedades, possui um

atributo identificador que nos ajuda a diferenciar as instâncias na memória de trabalho. Essa classe pode ser vista como uma camada para acessarmos, de forma mais fácil, os métodos da classe *Instance* que é a representação interna de uma instância no Protégé. A classe Instancia pode ser reusada em qualquer aplicação.

4. Aplicação

A fim de avaliarmos a eficácia da nossa proposta, construímos um SBC para diagnóstico clínico de doenças infecciosas. Nessa aplicação, o usuário (médico) insere, através do Protégé, informações relativas ao paciente e seus sintomas. Essas informações são inseridas na memória de trabalho onde o motor de inferência, baseado nas regras definidas, busca reconhecer padrões de sintomas e assim identificar a provável doença do paciente.

A ontologia utilizada é a mesma da Figura 1, nela modelamos os conceitos **Paciente**, **Sintoma**, **HistoriaClinica** e etc. HistoriaClinica é a classe cujas instâncias contém os dados obtidos durante a consulta, tais como, dados do paciente consultado, queixa principal, outros sintomas, etc. A regra apresentada no Quadro 4 alimenta a memória de trabalho do motor de inferência. Para obter um diagnóstico, o usuário insere na base de objetos do JEOPS uma instância da classe HistoriaClinica. A regra inicilizar utiliza essa instância para recuperar os valores das propriedades paciente e queixaPrincipal. Também são inseridos os sintomas que compõem o valor da propriedade sintomasAssociados da queixaPrincipal.

```
rule inicializar {
  declarations
    Instancia histClinc;
  conditions
    histClinc.getDirectTypeName().equals(“HistoriaClinica”);
  actions
    Instancia paciente = new Instancia(histClinc.getSlotValue(“paciente”));
    paciente.setIdentificador(“paciente”);
    insert(paciente);

    Instancia qxPrin = new
          Instancia(histClinc.getSlotValue(“queixaPrincipal”));
    qxPrin.setIdentificador(“queixaPrincipal”);
    insert(qxPrin);

    Collection lista = qxPrin.getSlotValues(“sintomasAssociados”);
    for (int i=0; i<lista.size(); ++i) {
       Instancia elem = new Instancia(lista.get(i));
       elem.setIdentificador(“sintomaAssociado”);
       insert(elem);
    }
}
```

Quadro 4. Regra que alimenta a memória de trabalho do motor de inferência

A regra mostrada Quadro 5 é responsável por reconhecer a combinação de sintomas que caracterizam um tipo de infecção chamado Hanseníase cujo principal sintoma é uma mancha dormente na pele. O que ela faz é verificar se a queixa principal é uma instância da classe SintomaManchaNaPele e se, associado a esse, há uma instancia do SintomaDormencia. Caso os objetos na base de fatos satisfaçam essas condições uma mensagem será impressa na tela informando que o paciente pode estar com hanseníase.

```
rule ehHanseniase {
  declarations
    Instancia paciente;
    Instancia mancha;
    Instancia dormencia;
  conditions
    paciente.getIdentificador().equals(“paciente”);
    mancha.getIdentificador().equals(“queixaPrincipal”);
    mancha.getDirectTypeName().equals(“SintomaManchaNaPele”);
    dormencia.getIdentificador().equals(“sintomaAssociado”);
    dormencia.getDirectTypeName().equals(“SintomaDormencia”);
  actions
    System.out.println(“O paciente “+paciente.getSlotValue(“nome”));
    System.out.print(“ provavelmente está com hanseníase”);
  }
```

Quadro 5. Regra que identifica o padrão de sintomas provocados pela Hanseníase

Neste exemplo, percebe-se que o uso da abordagem proposta no desenvolvimento deste SBC evitou a implementação em Java das classes Paciente, Sintoma e HistoriaClinica, pois as instâncias dessas classes, criadas no Protégé, alimentam diretamente a base de fatos JEOPS.

5. Conclusão

A integração das duas ferramentas aqui apresentadas como uma abordagem de desenvolvimento de SBCs mostrou que podemos, de forma simples e escalável, desenvolver sistemas baseados em regras de produção e ontologia. A simplicidade é obtida pelo uso do Protégé como ferramenta para modelagem e aquisição de conhecimento, o qual possui uma interface fácil de usar, e que proporciona uma base de conhecimento persistente. A integração do JEOPS com o Protégé elimina a necessidade de se implementar classes Java que representem a ontologia e permite uma independência entre a definição das regras e a evolução da ontologia. Desta forma, a ontologia, além de ser reutilizável, está sempre aberta a modificações e melhoramentos favorecendo a escalabilidade do sistema.

Referências Bibliográficas

Eriksson H, Fergerson R.W., Shahar Y, Musen M.A. *Automatic generation of ontology editors*. In Proceedings of the 12th Banff Knowledge Acquisition for Knowledge-based Systems Workshop. Banff, Alberta, Canada. 1999.

Figueira, C, Ramalho, G. *JEOPS – The Java Embedded Object Production System*. In Springer Verlag's Lecture Notes in Artificial Intelligence, v. 1952, pp. 52-61. 2000.

Gruber, T.R. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*. International Journal of Human-Computer Studies (IJHCS),43(5/6),907-928,1995.

Vasco, J.J.F. *A4 – Um Ambiente de apoio à Aquisição Automática de Conhecimento*. Dissertação de Mestrado, UFPB, 1993.

Wielinga, B.J., Schreiber, A.T., Breuker, J. *KADS: A Modeling Approach to Knowledge Engineering*, Knowledge Acquisition, 4(10), pp.5-53, 1992a.

Zlot, F., Oliveira, K.M., Rocha., A.R. *Modeling Task Knowledge to Support Software Development*. SEKE’02, Itália, 2002.