AMBIENTES PARA SISTEMAS DE INFORMAÇÃO

Antonio L. Furtado

Departamento de Informatica
Pontificia Universidade Católica do R.J
22.453 Rio de Janeiro, R.J.

Resumo

Este trabalho introduz os objetivos e linhas de ação iniciais do projeto NICE, que visa investigar ambientes cooperativos para o uso de sistemas de informação. O prototipo Prolog atualmente em uso é descrito quanto a seus dois componentes mais importantes e são delineados os proximos passos do projeto.

1. Objetivo do projeto

Hoje se considera que um sistema de informações automatizado inclui, alem de um banco de dados constituido de fatos basicos, um banco ou base de conhecimentos constituido de fatos e regras. Sobre todos esses fatos e regras trabalha um processo capaz de:

(a) fornecer informação, em resposta a operações de consulta;
(b) mudar o estado do sistema, sob o efeito de operações de atualização.

Deseja-se que o sistema se comporte de maneira cooperativa em relação ao usuario. A resposta a uma consulta pode não corresponder literalmente ao que foi pedido, mas deverá ser a mais util possivel em vista dos objetivos e planos do usuario [AP, We]. O efeito de uma operação de atualização pode igualmente não ser exatamente o que o usuario tinha em vista: alem de levar em conta objetivos e planos do usuario, a atualização será inibida ou modificada de modo a efetuar

transições validas entre estados validos do sistema. Em ambos os casos, o sistema deve conter conhecimento sobre si proprio de modo a complementar de modo "inteligente" as solicitações dos usuarios.

O objetivo do projeto NICE ("Nice Interaction Control Expert") é investigar metodos e desenvolver prototipos de software que visem a ambientes que permitam este tipo de uso cooperativo de sistemas de informação.

2. Estrategia adotada

Decidimos optar por uma linguagem formal no nivel basico do ambiente. Os aspectos de linguagem natural são um problema em si tão vasto que absorveriam uma parte grande demais dos esforços. Tomada esta decisão, reconhecemos que o paradigma de programação em logica é especialmente atraente para problemas que envolvam bases de conhecimento. As informações que constituem seu conteudo tomam a forma de fatos e regras. O processo que preside as consultas e atualizações é então a maquina de inferencia da linguagem. O conhecimento que o sistema deve ter de si mesmo para proporcionar o comportamento cooperativo desejado é representavel de modo uniforme na linguagem, tambem como fatos e regras, a serem aplicados pela maquina de inferencia.

Por outro lado, a linguagem Prolog, que no momento é a mais divulgada dentro do paradigma de programação em logica, é compativel com o modelo relacional de bancos de dados [Ko]. Essa compatibilidade permitiu que algumas versões de Prolog oferecessem interfaces com SGBDs relacionais [Gi, Ma].

Nossa decisão foi portanto partir de uma dessas versões: o Arity Prolog [Ma], que dispõe de um pacote relacional SQL [Da]. Note-se que a linguagem SQL, por sua importancia, já é objeto de um padrão ANSI.

Prolog/SQL já constitui uma combinação poderosa, criando condições para iniciar a abordagem de bancos de dados dedutivos [Ga], com consultas recursivas, otimização semantica, visões e restrições de integridade complexas, etc. Entretanto não é ainda suficiente para proporcionar um ambiente cooperativo como o que indicamos, o qual a nosso ver deveria dotar o sistema de informações das seguintes caracteristicas principais:

(a) o sistema é "ativo" podendo por sua vez fazer perguntas ao usuario e atuar em um modo monitorado, executando consultas e atualizações por um mecanismo de gatilhos;
(b) o comportamento do sistema é governado por regras pertencentes a certas classes pre-estabelecidas;
(c) um gerador de planos é incluido, associando um plano constituido de uma sequencia de ações a um objetivo constituido de uma conjunção de fatos positivos e/ou negativos.

Uma das consequencias de se ter um sistema que toma a iniciativa de fazer perguntas é ultrapassar o pressuposto do mundo fechado [GM], permitindo adquirir conhecimento atraves da interação com os usuarios. Essas interacões devem ser elas proprias registradas. Como o contexto em que uma interação ocorreu se altera, à medida que são feitas atualizações, sua analise posterior pode requerer uma organização de bancos de dados temporais. Tais organizações alem disso permitem consultas referentes a estados passados, complementando as consultas sobre estados futuros atingiveis que se tornam

possiveis graças à geração de planos.

A atividade de planejamento é, conceitualmente, uma combinação de consultas e atualizações: objetivos são expressões de consulta que valem no estado que seria atingido aplicando planos para atualizar o estado corrente.

Concluimos ser preciso desenvolver uma serie de ferramentas complementares com, entre outras, as seguintes funções:

(1) consulta do usuario ao sistema, com modificações anteriores e posteriores à consulta;
(2) consulta controlada feita pelo sistema ao usuario;
(3) explicacões, sob a forma de rastreio ("trace") estruturado;
(4) atualização limitada por restrições e complementada por gatilhos;
(5) gerador de planos, permitindo colocar como objetivo uma conjunção de fatos positivos e/ou negativos, com possiveis interferencias mutuas;
(6) comunicação transparente com o SQL para tratamento uniforme em Prolog de fatos situados na memoria e em banco de dados;
(7) organização de bancos de dados temporais;
(8) manipulação de estruturas tais como molduras ("frames"), hierarquias semanticas, e outras, reunindo atributos de objetos possivelmente guardados em diversas estruturas de dados;
(9) manutenção de diario ("log"), para registrar solicitações feitas durante uma sessão, e de perfis de usuarios;
(10) monitoramento das sessões de modo a ativar gatilhos ou "demonios".

As funções deverão ser integradas em um arcabouço coerente. Seguindo esse arcabouço, as ferramentas que as implementam deverão juntas constituir um ambiente de utilização. O prototipo atual já contem versões preliminares de todas essas ferramentas, com exceção da mencionada no item (7) que, embora já desenvolvida [Al], não foi incorporada ao prototipo; alem disso, os perfis de usuarios mencionados em (9) ainda não foram especificados. Referimos o leitor a [Re] para detalhes sobre (2) e (3), a [Fu] para (5) e a [Fu] para (6).

A seguir, concentraremos nossa atenção nas funções (1) e (4), onde reside a motivação fundamental do projeto.

3. Os comandos de consulta e atualização

No ambiente NICE deseja-se que as consultas e as atualizações sejam cooperativas. Vamos ver agora, com um pouco mais de detalhe, o que isso significa.

Uma informação dada ao usuario, alem de correta, deve ser util em relação a seus planos e objetivos. A partir dessa consideração pragmatica, um sistema cooperativo ao qual um usuario faz uma solicitação, seja ela uma consulta ou uma atualização, pode:

- corrigir ou reformular a solicitação antes de processa-la;
- fornecer informação adicional relevante ao que o usuario pretende fazer;
- advertir o usuario de que há outros obstaculos ao que ele pretende fazer, alem daqueles sobre os quais solicitou informação; esses obstaculos são tipicamente pre-condições de operações que não valem no estado corrente;
- oferecer para avisar quando obstaculos indicados não mais ocorrerem;
- chamar atenção para o caso de obstaculos irremoviveis (restrições de integridade ou pre-condicões irrealizaveis);
- indicar elementos satisfazendo a uma pergunta se esta falhar com os elementos indicados pelo usuario, ao inves de simplesmente dar resposta negativa;
- oferecer para exibir plano alternativo, que atinja o objetivo do usuario, quando seu plano original não for exequivel;
- dar explicações nos casos de não ser possivel atender de nenhuma forma à solicitação do usuario;
- esclarecer concepções erradas detetadas pela interação com o usuario e dar informação adicional, preventivamente, para evitar provaveis mal-entendidos;
- complementar atualizações com outras, se necessario, para manter a integridade das informações mantidas;
- vedar acesso a informações, para consulta ou atualização, conforme as normas de autorização.

As duas ultimas medidas se referem ao interesse do "sistema", tendo a ver com integridade e autorização. Ao ter em vista,

quanto às outras, o interesse do usuario, surge a questão de dar ou não prioridade ao que o usuario explicitamente declara ou parece preferir, ou então tentar achar para ele uma solução melhor (ou otima). Nesse ponto é importante a distinção entre objetivo e plano: aceitando que o objetivo do usuario é proveitoso para ele, o plano que ele tem em vista para atingi-lo, mesmo se for exequivel, poderá não ser o mais conveniente. Uma sofisticação maior nessa direção (fora do escopo deste trabalho) conduziria a ambientes para apoio à tomada de decisão.

Tanto os comandos de consulta como os de atualização são dirigidos por regras, em tres fases:

- regras "pre", preliminares à execução dos comandos
- regras "s_post", posteriores à execução bem-sucedida
- regras "f_post", posteriores à execução no caso de falha

Uma regra "pre" poderá modificar um comando C transformando-o em um novo comando C1. A C1 pode ser aplicada outra regra "pre", produzindo C2, e assim sucessivamente até atingir uma forma Cn à qual nenhuma regra "pre" possa ser aplicada de modo a efetuar alguma transformação. Cabe ao projetista o cuidado de verificar que as regras que definir garantem terminação.

Após a execução de Cn, as regras "s_post" (ou "f_post") são aplicadas, cumulativamente, sobre Cn. A intenção dessa vez não é modificar e sim complementar (ou compensar a falha de) Cn.

O comando de consulta é

query(<expressao logica>,<informação adicional>)

onde <expressão logica> é uma expressao Prolog a ser invocada como "goal" (objetivo), destinando-se a obter informação. Assim, tal expressão incluirá termos correspondentes a fatos Prolog ou tuplas SQL (as quais são expressas da mesma forma que os fatos).

Ja <informação adicional> é uma lista, possivelmente vazia, cujos elementos são expressões arbitrarias. Basicamente, a resposta direta à consulta do usuario aparece nas variaveis que existirem na <expressão logica> (ou simplesmente no sucesso ou fracasso de sua execução, se não houver variavel), enquanto a contribuição do sistema estará refletida na <informação adicional>.

Lembremos que regras "pre" podem ter modificado a <expressão logica> que efetivamente é executada. Nesse caso pode ocorrer que nem todas as variaveis originais, ou até nenhuma delas, sejam instanciadas. Isso pode ser usado deliberadamente para não revelar ao usuario informação que ele não esteja autorizado a ver. As regras "pre" poderão ou não determinar o formato e mesmo parte do conteudo dos elementos na lista que constituirá <informação adicional>. A lista poderá ser completada pela aplicação de regras "s_post" ou "f_post". Diversas aplicações de uma mesma ou de varias dessas regras resultarão em que a lista terá mais de um elemento.

O comando de atualização tem a forma

apply(<operação>)

onde <operação> é um termo Prolog cuja semantica é definida por

regras (clausulas Prolog) cujos tipos principais são:

```
added(<fato>,<operacão>,<estado>) :- <antecedentes>
deleted(<fato>,<operação>,<estado>) :- <antecedentes>
precond(<operação>,<sub-objetivo>,<estado>) :- <antecedentes>
```

Os efeitos de uma operação estão indicados nas clausulas "added" e "deleted", onde <fato> é um fato Prolog ou uma tupla SQL. A mesma operação pode adicionar ou remover diferentes tipos de fatos e assim requerer varias dessas clausulas. As clausulas "precond" são as mais interessantes: elas definem, para cada operação, que fatos devem estar presentes ou ausentes para que a operação seja aplicavel.

Essa forma de especificar operações é ideal para o gerador de planos (ver função (5) da seção anterior). O algoritmo de geração de planos [Fu], que é uma versão estendida do algoritmo WARPLAN desenvolvido por D. S. D. Warren [Wa, CC, Ch], dado um objetivo (conjunção de fatos positivos e/ou negativos) obtem um plano (sequencia de ações, ou seja, de aplicações de operações) capaz de levar do estado corrente a um estado em que o objetivo seja alcançado. As pre-condições para aplicar uma operação, dadas em clausulas "precond", são tratadas pelo algoritmo como sub-objetivos que requerem que o plano atinja estados intermediarios em que estes sejam alcançados. Na terminologia de bancos de dados, isso corresponde a garantir restrições de integridade atraves da propagação dos efeitos das operações, notando apenas que a propagação é executada antes e não depois da execução da propria operação.

O gerador de planos é utilizado para verificar se a operação

indicada pode ser imediatamente aplicada no estado corrente. Caso contrario a operação falha.

Regras "pre" podem modificar um comando de atualização. Regras "s_post" e "f_post" podem executar outras operações (propagação a posteriori) ou simplesmente oferecer informações ao usuario através de mensagens.

É importante notar que nosso conceito de operação situa-se dentro da abordagem de tipos abstratos de dados [VF]. As operações tipicamente serão escolhidas dentro de cada dominio de aplicação, que no caso de um sistema de informações academico seriam algo como "ofereça curso", "matricule aluno em curso", "transfira aluno de um curso para outro", "cancele curso", etc. Decidimos tambem oferecer duas operações genericas, "add" e "del", para o caso em que não se deseje adotar essa abordagem, mas para cada tipo de fato a ser manipulado por elas devem ser explicitadas regras "added", "deleted" e "precond"; é preciso inclusive explicitar a inexistencia de pre-condições, colocando o fato trivialmente verdadeiro - "true" - em <sub-objetivo>.

A seguir veremos o que fazem, especificamente, as regras já disponiveis no prototipo para dirigir consultas e atualizações dentro do enfoque cooperativo.

4. Regras implementadas para consulta e atualização

As regras que dirigem o prototipo podem ser independentes ou dependentes do dominio da aplicação. O prototipo já inclui algumas regras da primeira categoria para consulta e

atualização, as quais passaremos a descrever, iniciando pelas de consulta.

Se a consulta envolver objetos e atributos de objetos, uma regra do tipo "pre" verifica a adequacão dos atributos aos objetos. Nada há a modificar se os atributos forem aplicaveis diretamente ou por herança através de hierarquia is-a. A inexistencia de classe de objetos, a inexistencia de atributo e a não aplicabilidade de um atributo existente a uma classe existente de objetos são acusadas; no ultimo caso é informada ao usuario a que classe de objetos o atributo se aplica. O caso mais interessante, incluido por sugestao de D. Schwabe [SM], é o de um atributo que se aplique a uma classe C' situada abaixo, na hierarquia is-a, da classe C indicada. Por exemplo, suponha que é perguntado o salario de um elemento da classe pessoa, quando na verdade salario se aplica a empregado; mas como "empregado is-a pessoa", e pode muito bem ocorrer que o elemento indicado como pessoa seja tambem um empregado, a consulta é modificada substituindo pessoa por empregado e é executada (podendo ou não ter sucesso, evidentemente), sendo avisado ao usuario que salario é atributo de empregado.

Regras dos tipos "s_post" e "f_post" são usadas quando a consulta envolve somente fatos que são pre-condicões ou estão relacionados (por regras "related_info") com alguma das operações pre-definidas. O prototipo assume que o usuario está interessado em executar a operação e que, portanto, deve ser informado tambem dos fatos relacionados que ele não tenha incluido na pergunta. Por exemplo, se for perguntado quem ensinará um determinado curso, será assumido que o usuario

pensa em se matricular nele; receberá então informação quanto ao periodo de inicio do curso e quanto à sua duração. Mas isso só será informado se a operação for aplicavel no estado corrente; se não for aplicavel, ele será apenas informado de que a operação não pode ser executada.

Apenas para o caso de falha, uma regra "f_post" tenta ampliar o foco da consulta, através do metodo de generalização mais especifica [WC], que atua no sentido inverso da unificação. Suponha que tenha sido perguntado se existe um voo de O para D pela companhia C1 e que a resposta seja negativa; se mais tarde for feita a pergunta com parametros O, D e C2 (e esta tambem falhar) será assumido que a companhia não é fundamental, e a pergunta é generalizada para O, D e X, onde X é uma variavel. A pergunta terá então sucesso se houver voo de O para D por alguma companhia.

Se uma consulta falha em ultima instancia, o comando de explicacão por rastreio estruturado (função (3) do prototipo) é invocado.

Para atualizacão, só está disponivel no momento uma regra do tipo "f_post" que verifica se as pre-condições para executar operação que falhou podem eventualmente se tornar verdadeiras. Em caso afirmativo, é perguntado ao usuario se ele deseja ser informado no momento em que isso ocorrer; se o usuario responder que deseja, é criado um gatilho que irá emitir uma mensagem ao se tornarem verdadeiras as pre-condições. A outra alternativa oferecida ao usuario é de lhe mostrar um plano - caso haja - capaz de alcancar os efeitos da operação indicada,

sendo que esse plano poderá incluir ou não essa operação. Por exemplo, a operação de matricular um aluno em um curso B pode não ser permitida por uma restrição que proibe quem esteja cursando A de cursar tambem B. Mas um plano envolvendo a transferencia de A para B pode ser viavel, sendo então mostrado se o usuario indicar seu interesse.

No caso de falha em ultima instancia, uma explicação especial para operações é invocada. A explicação analisa a falha conforme uma ou mais das causas abaixo ocorra:

1. inconsistencia - a expressão que viola alguma restrição de integridade é mostrada juntamente com a restrição;
2. operação não produtiva - fatos que a operação deveria inserir mas já valem no estado corrente, ou fatos que deveria remover mas não valem, são mostrados;
3. operação inaplicavel - a expressão de pre-condições é mostrada; se a expressão não pode, por sua vez, tornar-se verdadeira pela execução de algum plano (obstaculo irremovivel), essa circunstancia é tambem informada.

Para terminar, daremos um exemplo simples de regras da categoria dependente de dominio, para dirigir consultas. Considere uma consulta envolvendo voos entre duas cidades O e D. Uma regra "pre" pode redirecionar o voo de O para O' ou de D para D', se O ou D são cidades cujo aeroporto não esteja em funcionamento mas que sejam vizinhas de cidades O' ou D' que estejam sendo usadas em substituição. Uma regra "s_post" pode perguntar ao usuario se seu voo se iniciará realmente em O; se se iniciar em outra cidade I, é verificada a existencia de voo de I para O. Uma regra "f_post" pode perguntar sobre a preferencia quanto a escalas, por falhar a busca de voo direto entre O e D. Nos dois casos de regras invocadas após à consulta, o roteiro completo pode ser informado ao usuario, com

as possiveis modificações efetuadas pela regra "pre".

A figura 1 esboça a arquitetura do prototipo. No ambiente NICE são disponiveis ao usuario m comandos, implementados sobre n ferramentas. Seu comportamento é regido por dois conjuntos de regras - as independentes e as dependentes do dominio de aplicação - e é adaptado a diferentes usuarios conforme seus perfis. As informações são guardadas como clausulas ou tabelas. O diario registra solicitações feitas durante uma sessão.

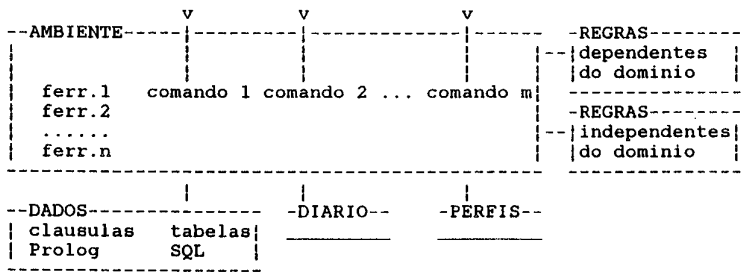

Figura 1. A arquitetura NICE

5. A continuação do projeto

Estamos principalmente interessados em ampliar a categoria de regras independentes de dominio e em oferecer ferramentas que ajudem o projetista de uma aplicação a especificar regras dependentes do dominio da aplicação.

A função de banco de dados temporais (7) deve ainda ser integrada ao prototipo. A melhor integração das demais funções e a possivel adição de outras funções são outros pontos a examinar. Por outro lado, pretendemos tratar dos aspectos de eficiencia e da aparencia da interface (janelas, etc.) somente

após os aspectos mais conceituais terem atingido um ponto relativamente estavel.

Ainda quanto à interface, devemos esclarecer que os usuarios finais não se comunicarão, em geral, através de Prolog. Prolog é a linguagem acessivel ao projetista de aplicação, sendo sua responsabilidade especificar a linguagem pseudo-natural, voltada para a aplicacao [Ne], que os usuarios irão de fato utilizar.

Finalmente, o ambiente NICE pode ser aplicado a sistemas de informação já existentes; para o caso de se pretender aplica-lo a sistema a ser desenvolvido, seria conveniente desenvolver metodos de projeto de sistemas de informação que tirassem do ambiente o maximo proveito, sendo tais metodos um topico adicional para pesquisa futura.

Agradecimentos: O presente trabalho foi parcialmente financiado pela FINEP. O texto foi preparado em equipamento do Centro Cientifico Rio da IBM Brasil.

Referencias

[Al] J. J. P. Alcazar - "Bancos de dados temporais: uma revisão" - relatorio tecnico 25/89 - PUC/RJ (1989).
[AP] J. F. Allen e C. R. Perrault - "Analyzing intentions in utterances" - Artificial Intelligence 15,3 (1980) 143-178.
[CC] H. Coelho, J. C. Cotta e L. M. Pereira - "How to solve it with Prolog" - Laboratorio Nacional de Engenharia Civil - (1982).
[Ch] D. Chapman - "Planning for conjunctive goals" - Artificial Intelligence - 32, 3 (1987) 333-377.
[Da] C. J. Date - "Database - a primer" - Addison Wesley (1983).
[Fl] A. L. Furtado - "Some extensions to Warren's plan-generation algorithm" - relatorio tecnico 20/89 - PUC/RJ

(1989).
[F2] A. L. Furtado - "Treating workspace and SQL facts uniformily in Prolog" - relatorio tecnico 21/89 - PUC/RJ (1989).
[Ga] H. Gallaire - "Bridging the gap between AI anda databases: logic approach" - in "Data and knowledge (DS2)" - R. A. Meersman e A. C. Sernadas (eds.) - North-Holland (1988) 151-172.
[Gi] M. Gillet - "VM/Programming in Logic" - Manual IBM SB11-6374-0 (1985).
[GM] H. Gallaire e J. Minker - "Logic and Databases" - Plenum (1978)
[Ko] R. Kowalski - "Logic for data description" - in "Logic and Data Bases" - H. Gallaire e J. Minker (eds.) - Plenum (1978).
[Ma] C. Marcus - "Prolog programming" - Addison-Wesley (1986).
[Ne] J. M. Neighbors - "The Draco approach to constructing software from reusable components" - IEEE Transactions on Software Engineering, 9 (1984) 564-574.
[Re] M. A. Mortari Rezende - "HIER - Highly interactive expert resolver: uma ferramenta para sistemas especialistas" - relatorio tecnico 18/89 - PUC/RJ (1989).
[SM] D. Schwabe e E. E. Mizutani - "A knowledge-based browser to access complex databases" - relatorio tecnico - PUC/RJ - em publicação.
[VF] P. A. S. Veloso e A. L. Furtado - "Towards simpler and yet complete formal specifications" - in "Information systems: theoretical and formal aspects" - A. Sernadas, J. Bubenko Jr., e A. Olive (eds.) - North-Holand (1985) 175-189.
[Wa] D. S. D. Warren - "WARPLAN - a system for generating plans" - DAI, memo 76 - University of Edinburgh (1974).
[WC] A. Walker, M. McCord, J. S. Sowa e W. G. Wilson - "Knowledge systems and Prolog" - Addison-Wesley (1987).
[We] B. L. Webber - "Questions, answers and responses: interacting with knowledge base systems" -in "On knowledge base management systems" - M. L. Brodie e J. Mylopoulos (eds.) - Springer (1986).