Um Ambiente Expert para o Apoio ao Desenvolvimento de Software

Sílvia Maria Wanderley Moraes[1]
Daltro José Nunes

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Pós-Graduação em Ciência da Computação
Caixa Postal 15.064 CEP 91591-970
Porto Alegre - RS - Brasil
E-mails: silvia@music.pucrs.br
daltro@inf.ufrgs.br

RESUMO

O emprego de assistentes inteligentes em ambientes de desenvolvimento de software é, hoje, além de uma característica desejável, uma necessidade, principalmente, em função da distância conceitual existente entre as semânticas do domínio do problema e os artefatos de software. Tais assistentes, apesar dos vários exemplos presentes na literatura referentes ao seu desenvolvimento e aplicação, não receberam em sua construção um tratamento formal.

O objetivo deste trabalho é modelar e especificar formalmente uma ferramenta *shell* para a construção de sistemas especialistas, que possa ser incorporada ao conjunto de ferramentas de um ambiente de desenvolvimento de software. O método formal escolhido para a especificação foi o algébrico e como PROSOFT é um ambiente que baseia-se nesse método, ele foi escolhido para o desenvolvimento da ferramenta.

[1] Professora da Pontifícia Universidade Católica do Rio Grande do Sul - Instituto de Informática - Av.Ipiranga, 6690 - CEP 90610-000 - Porto Alegre/RS – Brasil.

1. INTRODUÇÃO

O desenvolvimento de um software é uma atividade que exige intenso conhecimento. O engenheiro de software precisa conhecer bem o domínio da aplicação, a plataforma de implementação (software e hardware) e as técnicas atuais de desenvolvimento de software para elaborar um produto final de qualidade.

A Engenharia de Software, preocupada com a qualidade e pelo fato do desenvolvimento de software estar ligado ao conhecimento, foi buscar na Inteligência Artificial (IA) técnicas que a ajudassem na resolução de seus problemas. A área que estuda aplicação das tecnologias de IA em Engenharia de Software é conhecida como Engenharia de Software Baseada em Conhecimento (KBSE).

A KBSE tem apostado no conhecimento como um fator importante para a produção de softwares de alta qualidade. É crescente o número de pesquisadores que empregam sistemas baseados em conhecimento (SBC) para assistir os engenheiros de software em suas atividades [FAL 95].

Os SBC's aparecem, normalmente embutidos em ambiente de desenvolvimento de software, trabalhando como assistentes inteligentes ao processo de construção de um software. Esses assistentes cooperam com o engenheiro de software, ajudando de forma especializada, na solução dos problemas .

Embora esses assistentes, em sua maioria, tenham sido desenvolvidos como protótipos e existam poucas métricas para a avaliação efetiva dos resultados obtidos com a sua aplicação, a necessidade de encurtar a distância conceitual entre a semântica do domínio dos problemas e os artefatos de software faz desses assistentes uma característica desejável e necessária em qualquer ambiente de desenvolvimento [FIS 92].

Há, na literatura da área de KBSE, vários exemplos de assistentes inteligentes em ambientes de desenvolvimento de software. Entretanto, pouco tratamento formal ou quase nenhum lhes foi dado. A maioria apresenta apenas uma descrição informal do assistente.

O uso de um método formal é importante para a especificação de qualquer sistema, especialmente os mais complexos. O embasamento matemático do método torna possível garantir propriedades como completeza, corretude e ausência de ambigüidade que são muito difíceis de se obter com uma descrição informal. Além disso, uma descrição informal, como um texto em português, está suscetível a erros de interpretação e, até mesmo, à incompreensão dos procedimentos descritos.

Em função dos benefícios do uso de um método formal e da necessidade do uso de assistentes inteligentes no processo de desenvolvimento de software, o objetivo deste trabalho foi modelar e especificar formalmente uma ferramenta *shell*, para a construção de sistemas especialistas, que possa ser incorporada ao conjunto de ferramentas de um ambiente. Através dos recursos oferecidos por esta ferramenta, assistentes "inteligentes" poderão ser construídos, para os mais variados propósitos, a fim de atender as necessidades dos engenheiros de software ao longo do processo de desenvolvimento.

O método formal escolhido para a especificação foi o algébrico. O ambiente de desenvolvimento de software PROSOFT foi escolhido para o desenvolvimento dessa ferramenta *shell* por vários motivos, entre os quais, está o fato dele basear-se no método algébrico, que foi o método escolhido; por ele ser um ambiente aberto, possibilitando a inclusão da *shell* ao seu conjunto de ferramentas; e por não ter nenhum tipo de suporte inteligente.

2. O AMBIENTE PROSOFT

O PROSOFT é um Projeto de Pesquisa do Grupo de Sistemas de Informação do CPGCC/UFRGS sob a coordenação do Prof. Dr. Daltro José Nunes. O objetivo desse projeto é a construção de um ambiente de desenvolvimento de software.

O ambiente proposto prevê a definição e o desenvolvimento de um conjunto de ferramentas para apoiar o engenheiro de software da análise (definição dos requisitos) à solução de um problema (implementação) [NUN 94].

Algumas dessas ferramentas já foram definidas e implementadas. Outras, encontram-se em fase de projeto ou de implementação. Entre estas ferramentas, estão as ferramentas do Ambiente Expert, descritas neste trabalho , que em breve devem ser implementadas.

No processo de desenvolvimento de um software, a integração das ferramentas de suporte é essencial. Pois, a continuidade do processo depende da facilidade de acesso e de recuperação dos dados produzidos a cada fase do desenvolvimento.

No PROSOFT, os dados resultantes da interação ferramentas-engenheiros são armazenados em um repositório (diretório) de dados comum a todo o ambiente. Esta prática permite que qualquer ferramenta de projeto, durante o desenvolvimento, tenha acesso a dados (objetos) atualizados, conforme ilustra a Figura1.

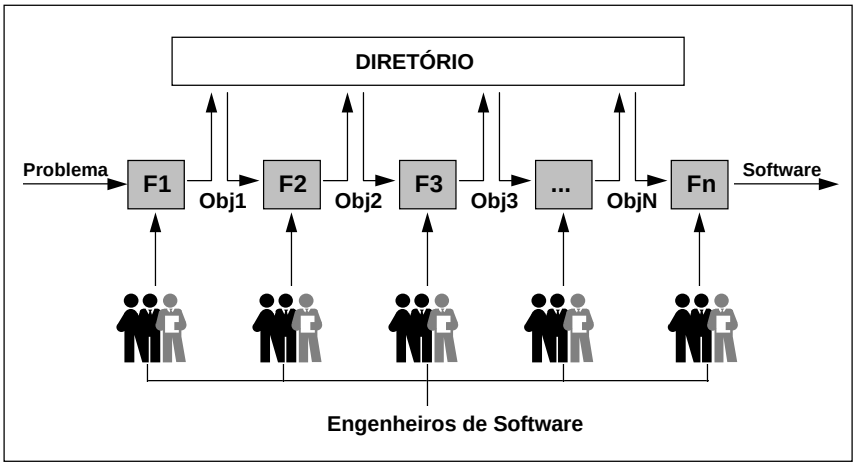


Figura1 - O processo de desenvolvimento de software no PROSOFT

O ambiente PROSOFT, em função da sua estrutura, incentiva o reuso de software, pois permite que aplicações, desenvolvidas sob seu paradigma sejam utilizadas no desenvolvimento de novas aplicações. E, além disso, devido a sua arquitetura aberta [NUN 89], admite a incorporação destas aplicações ao seu conjunto de ferramentas.

2.1 Estrutura do Ambiente PROSOFT

O ambiente PROSOFT, estruturalmente, está baseado em dois elementos: o ATO e a ICS.

↳ O ATO - *Ambiente de Tratamento de Objetos* - é o nome dado a qualquer ferramenta desenvolvida sob o paradigma PROSOFT, seja ela de suporte ou de aplicação. Um ATO PROSOFT (vide Figura 2) implementa um tipo abstrato de dados. Ele é composto, essencialmente, de uma classe e de um conjunto de operações de atuam sobre os objetos dessa classe. Cabe ressaltar, que o conceito de classe empregado no PROSOFT difere um pouco do conceito de classes em linguagens orientadas a objeto. Conforme [KOR 96], uma classe

PROSOFT é uma estrutura de dados que corresponde ao que normalmente é chamado de atributos do objeto ou variáveis de instância nas linguagens orientadas a objeto.

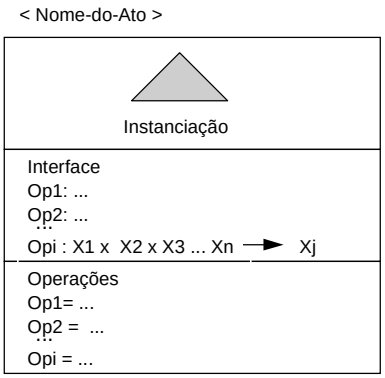


Figura 2 - Estrutura de um ATO

↳ A ICS - *Interface de Comunicação do Sistema* - estabelece o meio de comunicação das ferramentas PROSOFT. É através da ICS que um ATO pode enviar e receber mensagens de outros ATO's.

O desenvolvimento de um sistema, sob a estrutura PROSOFT, consiste na definição de um ou mais ATO's. Esses novos ATO's podem ser construídos com o auxílio do conjunto de ferramentas do ambiente (que também são ATO's). E, podem, ainda, se valerem da prática de reuso e utilizar em suas definições ATO's já existentes no ambiente.

A integração entre os ATO's que compõem o sistema assim definido é feita pela ICS, como mostra a Figura 3.

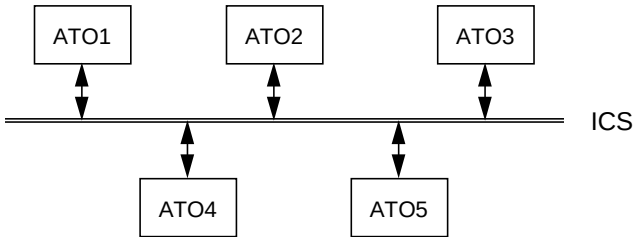


Figura 3 - Estrutura do Ambiente PROSOFT

3. PROCESSO DE DESENVOLVIMENTO DO AMBIENTE EXPERT

Em um ambiente de desenvolvimento de software, são poucas as ferramentas de suporte que desenvolvem atividades de forma completamente automática. A grande maioria interage com o engenheiro de software, auxiliando-o na execução das atividades envolvidas no processo de desenvolvimento de software. E é durante esta interação que os engenheiros podem se deparar com problemas e precisarem de ajuda mais especializada.

A função do ambiente expert é tanto possibilitar a construção de assistentes inteligentes ao processo de desenvolvimento de software quanto controlar a sua aplicação.

O engenheiro de software perante um problema, chama a ferramenta e seleciona a Base de Conhecimento adequada para o caso. A ferramenta, baseada no objeto e nas respostas do próprio engenheiro, sugere um caminho para uma provável solução ou não. A ferramenta terá condições de encontrar a resposta, se houver, dependendo evidentemente da Base de Conhecimento escolhida e das informações que lhe foram fornecidas. Na Figura 4, está ilustrada a forma de atuação desta ferramenta no processo.

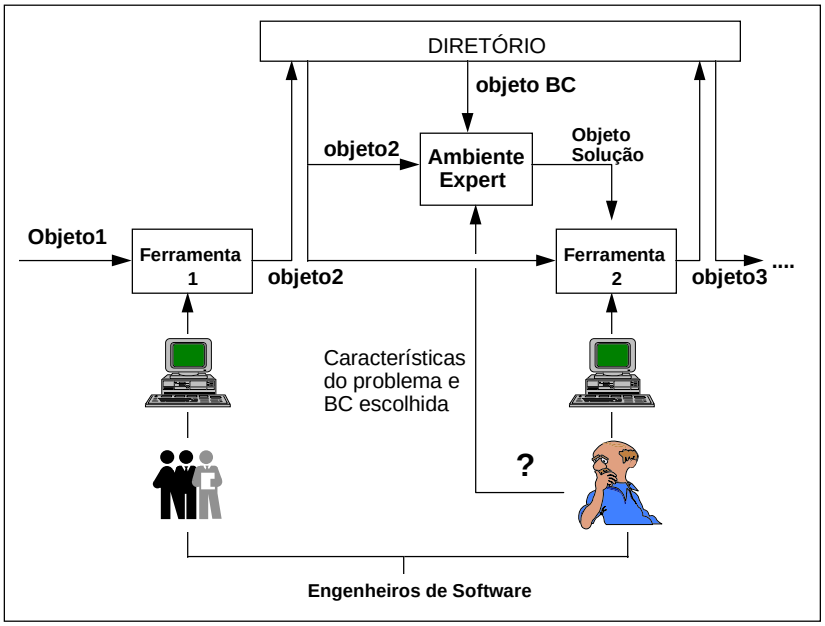


Figura 4- Auxílio prestado pelo Ambiente Expert

3.1 Descrição do Ambiente Expert

O ambiente Expert é composto por seis ferramentas PROSOFT:

- *ATO Base de Conhecimento e Informação(BCI)* - principal ferramenta do ambiente, é responsável pela construção da base de conhecimento e pela aplicação da máquina de inferência. Contém informações sobre o tipo de método de incerteza utilizado, sobre a estratégia de controle escolhida e sobre os critérios determinados para a resolução de conflitos entre regras;
- *ATO Banco de Conhecimento (Banco)* - ferramenta que permite criar uma Base de Conhecimento;
- *ATO Asserção Relacional (ASSREL)*- ferramenta integrada ao ATO BCI que permite a definição dos fatos da premissa da regra de forma relacional.
- *ATO Ação Prosoft (AÇÃO)*- ferramenta, também incorporada ao ATO BCI, que permite a definição de uma ação a ser executada pela máquina de inferência. Esta ação é uma operação de qualquer ATO Prosoft.
- *ATO Memória de Trabalho (MT)*- esta ferramenta mantém os fatos manipulados pela máquina de inferência.
- *ATO Agenda* - esta ferramenta armazena os ciclos gerados pela máquina de inferência.

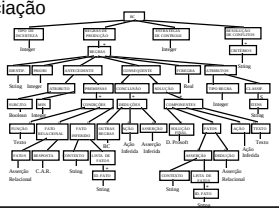
3.1.2 Descrição do ATO BCI

O ATO BCI, ilustrado pela Figura 5, é a principal ferramenta do ambiente Expert. É através das suas operações que uma base de conhecimento pode ser construída e a máquina de inferência aplicada.

O paradigma escolhido para representar o conhecimento da base foi o de regras de produção devido, principalmente, a sua simplicidade e difusão. As regras manipulam quaisquer objetos PROSOFT.

ATO BCI

Instanciação



Interface

cria_bc : -> BCI

altera_metinc(_,_,_,_): BCI, COORD,
COORD, INTEGER -> BCI

Operações

cria_bc =(MetInc 1, RegProd cria_regprod,
EstrCrl 1, Resconf cria_resconf)

altera_metinc((MetInc m,_,_,_),
coord_clid, coord_bc, mnv)=
if verifica_coord(coord_clic,
pos_metinc(coord_bc, (MetInc m, _,_,_),m))
then (MetInc mnv,_,_,_)
else (MetInc m,_,_,__

Figura 5 - ATO BCI

3.1.2.1 Instanciação do ATO BCI

O ATO BCI foi definido a partir dos tipos compostos Sets, Lists, Records e Unions [WAT 81] e [COH 86]. Além dos tipos compostos foram construídos outros ATO's para complementar a sua definição.

Os ATO's complementares criados foram:

- *Asserção Relacional*, que estabelece uma representação relacional para as asserções, representando-as na forma de tabelas;
- *Ação Prosoft*, que torna possível a definição de uma ação, uma operação de algum ATO, para posterior execução pela MI ;
- *Agenda,* que tem a função de armazenar a lista de ciclos criados pela MI
- *Memória de Trabalho*, componente da Agenda, que mantém os fatos informados e deduzidos pelo sistema durante o processamento da MI.

O ATO BCI, de acordo com a Figura 6 que ilustra a sua instanciação, é composto basicamente de 4 elementos: o Método de Incerteza, as Regras de Produção, a Estratégia de Controle e a Resolução de Conflitos.

O Método de Incerteza é o elemento da BCI que determina o tipo de raciocínio estatístico que será empregado pela MI sobre as regras para o cálculo dos fatores de certeza.

A Estratégia de controle, outro elemento da BCI, determinará o tipo de estratégia a ser aplicada pela MI: *forward chaining*, direcionando o raciocínio das premissas para a conclusão ou *backward chaining*, direcionando o raciocínio no sentido inverso.

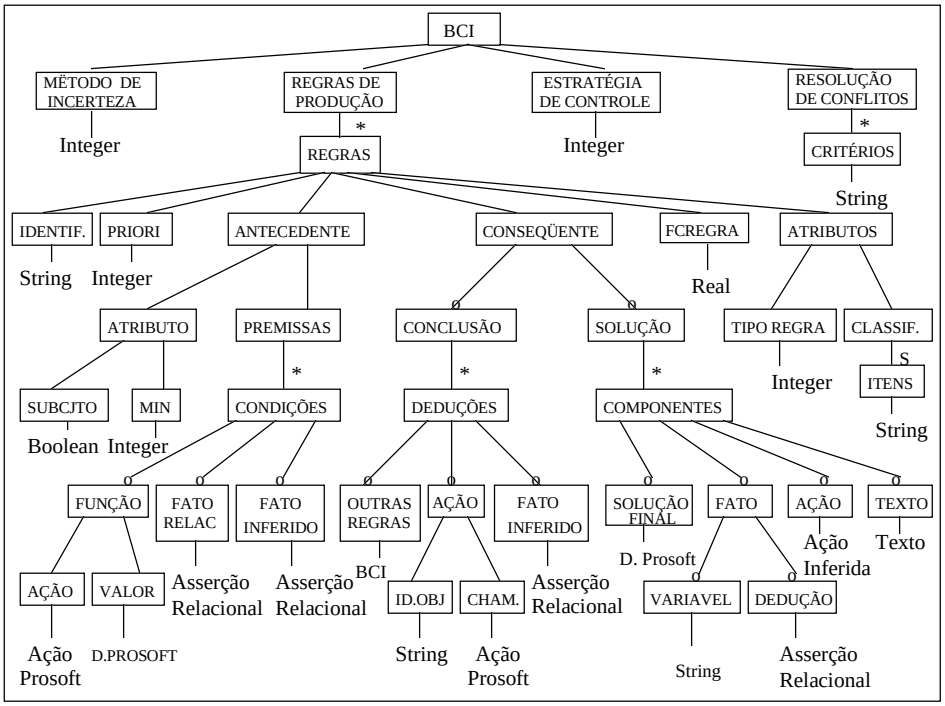


Figura 6 -Instanciação do Ato BCI

O campo Resolução de Conflitos da BC estabelece uma lista de critérios a serem seguidos pela a MI com a finalidade de solucionar uma situação de conflito onde há várias regras ativas, entretanto apenas uma deve ser disparada. É necessário para isso decidir qual regra disparar. Estes critérios, que foram expressos na forma de lista pois devem ser aplicados de acordo com uma ordem, definem os aspectos de devem ser considerados para o desempate das regras, tais como: prioridade, fator de certeza , complexidade da regra [JOHN 94], aplicar regras especiais antes de genéricas [RIC 93], limitação de contexto [WIN 93], etc.

As regras de produção, formalismo escolhido para representar o conhecimento, formam o último elemento da BCI. Elas são compostas por uma lista de regras. As regras foram definidas na forma de lista pois é necessário que haja uma ordem para a execução das mesmas, podendo esta ordem ser alterada em função da prioridade da regra.

Cada regra da BCI é formada por seis componentes :

- *Identif.:* Serve para a identificação da regra
- *Priori* : A Priori estabelece a prioridade de uma regra. A prioridade pode ser utilizada como um critério para a Resolução de Conflitos.
- *Antecedente* : O Antecedente é outro elemento constituinte de uma regra. Este componente representa o que a literatura em geral chama de parte IF da regra. O antecedente é composto basicamente de atributo e de premissas.
- *Conseqüente*: O outro componente de uma regra é o conseqüente, conhecido como parte THEN da regra. O conseqüente de uma regra pode ser de dois tipos: uma Conclusão ou uma Solução.
- *FCRegra* :O FCRegra representa o fator de certeza da regra, que mede o quanto as evidências descritas no antecedente suportam a conclusão presente no conseqüente da regra [RIC 92].
- *Atributos* : Os Atributos de uma regra especificam o seu tipo e a sua classificação. Segundo [JOHN 94], permitir que o engenheiro do conhecimento defina regras especificas para forward ou backward chaining, ou seja determine o tipo da regra, é

uma característica interessante das *shells* no que se refere às capacidades de um sistema especialista. O mesmo autor comenta ainda, que classificar as regras, através de alguns atributos, também definidos pelo engenheiro do conhecimento, agiliza a manutenção pois, no caso de expansão de uma grande BC, o engenheiro tem como localizar as regras a serem alteradas mais facilmente. Além disso, a seleção das regras aplicáveis a cada ciclo da MI pode ser tornar mais eficiente[RIC 93]. O resultado é um processamento mais rápido da MI por estar concentrada em apenas um grupo de regras e não em toda a BC. Para isso foram definidos um conjunto de itens que classificam as regras.

A Figura 7 apresenta a interface e algumas das operações do ATO BCI .

Interface

cria_bc : → BCI

cria_regra(_,_,_,_): BCI, COORD, COORD, STRING → BCI

cria_condfunc(_,_,_,_,_,_,_): BCI, COORD, COORD, STRING, STRING, {ACAOPROS ∪ D.PROSOFT}, D.PROSOFT → BCI

cria_condftrel(_,_,_,_) : BCI, COORD,COORD, STRING → BCI

Operações

Variáveis Formais coord_clic, coord_bc : COORD
idreg, n_ato, n_oper, idcontxt : STRING
selet, val : D.PROSOFT

cria_bc = (MetInc 1, Regprod *cria_regprod*, EstrCrtl 1, ResConf *cria_resconf*)

cria_regra((_, Regprod empty-list, _, _),coord_clic, coord_bc, idreg) =
if verifica_coord(coord_clic, *pos_regprod*(coord_bc, (_, Regprod empty-list, _, _),empty-list)
then (_, Regprod *ins_regra*(*inic_regra*(idreg),empty-list), _, _)
else (_, Regprod empty-list, _, _)

cria_regra((_, Regprod rp, _, _),coord_clic, coord_bc, idreg) =
if existe_idreg(idreg, rp)
then (_, Regprod rp, _, _)
else **if** verifica_coord(coord_clic, *pos_ins_regprod* (*pos_regprod*(coord_bc,
(_, Regprod rp, _, _),rp),rp))
then (_, Regprod *ins_regra*(*inic_regra*(idreg),rp), _, _)
else (_, Regprod *inclui_regra_pos*(*inic_regra*(idreg),rp,coord_clic,
pos_regprod(coord_bc, (_, Regprod rp, _, _),rp)), _, _)

cria_condfunc ((_, Regprod empty-list, _, _), coord_clic, coord_bc,n_ato,n_oper,selet,val) =
(_, Regprod empty-list, _, _)

cria_condfunc ((_, Regprod rp, _, _), coord_clic, coord_bc, n_ato,n_oper,selet, val) =
(_, Regprod *constroi_condfunc*(coord_clic, rp, n_ato,n_oper,selet, val,
pos_regprod(coord_bc, (_, Regprod rp, _, _),rp)), _, _)

cria_condftrel((_, Regprod empty-list, _, _), coord_clic, coord_bc, idcontxt) =
(_, Regprod empty-list, _, _)

cria_condftrel((_, Regprod rp, _, _), coord_clic, coord_bc, idcontxt) =
(_, Regprod *constroi_condftrel*(coord_clic,idcontxt,
pos_regprod(coord_bc, (_, Regprod rp, _, _),rp),rp), _, _)

Figura 7 - Exemplo de especificação de algumas das operações do ATO BCI

3.2.1.2 Operação Máquina de Inferência

A máquina de inferência é o mecanismo responsável pelo raciocínio do sistema especialista.

No ambiente Expert, a máquina de inferência é uma operação do ATO BCI que utiliza os ATO AGENDA e ATO MT. O ATO AGENDA armazena os ciclos reconhecimento e ação da máquina de inferência e o ATO MT é utilizado para guardar os fatos deduzidos e informados pelo sistema, sendo este último utilizado na composição do ATO AGENDA.

A operação *mi*, expressa formalmente na Figura 8, é responsável pela ativação do mecanismo de inferência. De acordo com o código da estratégia de controle é determinado o tipo de encadeamento da máquina, se for 1 é o *forward chaining* e 2 para o *backward chaining*.

Interface

mi(_): BCI $\rightarrow$ {AGENDA $\cup$ ERRO}

Operações

mi((_,RegProd empty-list,_,_)) = ERRO

mi((_,RegProd rp, EstrCrtl est,_)) =

if est = 1 **then** *mi_forward*((_,RegProd rp, EstrCrtl est,_,_), **ICS**(AGENDA, cria_agenda))

else *mi_backward*((_,RegProd rp, EstrCrtl est,_,_), **ICS**(AGENDA,cria_agenda))

Figura 8 - Especificação da operação máquina de inferência

O ambiente implementa, nesta primeira especificação, apenas o *forward chaining*, especificado pela operação externa mi_forward. A operação especifica apenas elementos fundamentais da estratégia e, por isso, não inclui retrocesso [WIN 93] e nem tratamento de incerteza.

A operação *mi_forward* (vide Figura 9), a principio, cria um objeto agenda, com o auxilio do ATO AGENDA, via ICS. Em seguida indexa as regras pelas condições presentes no seu antecedente para facilitar o *matching* e joga os fatos da base de conhecimento na agenda.

Interface

mi_forward(_,_): BCI, AGENDA $\rightarrow$ AGENDA

Operações

mi_forward(bci,agenda) = mif(bci, *indexa_regforw*(inic_mt(agenda, select-RegProd(bci)), select-RegProd(bci))

Figura 9 - Especificação da estratégia forward chaining

Através do seu conjunto de operações internas, mif e outras operações, as regras são selecionadas e aplicadas até que uma solução seja encontrada ou não existam mais regras a disparar.

O resultado da aplicação da máquina de inferência é um objeto do tipo AGENDA, onde a solução e os ciclos desenvolvidos pela máquina ficam armazenados.

CONCLUSÃO

A maior dificuldade encontrada para o desenvolvimento deste trabalho foi, sem dúvida, a definição das classes dos ATO´s PROSOFT que compõem o Ambiente Expert. A alteração das classes, por várias vezes, resultou na redefinição quase que completa das operações de manipulação desses ATO´s. Além disso, contatou-se que classes grandes e complexas dificultam o desenvolvimento das operações.

As regras definidas pelo ambiente manipulam quaisquer objetos PROSOFT já existentes. A medida que novos objetos base de conhecimentos forem definidos podem ser reutilizados na construção de novas bases.

TRABALHOS FUTUROS

Como trabalhos futuros, pretende-se tratar erros, definir formalmente um subsistema de aquisição do conhecimento e as operações de explanação: *why*, *how* e *why not*.

Além disso, pretende-se abordar outros paradigmas de representação para base de conhecimento e construir uma representação externa para a mesma.

É necessário também especificar algebricamente o tratamento de incerteza, a estratégia *backward chaining* e o *backtracking*.

REFERÊNCIA BIBLIOGRÁFICAS

[COH 86] COHEN, B.; HARWOOD,W.T; JACKSON, M.I. **The Specification of Complex Systems.** Great Britain: Addison-Wesley,1986.

[FAL 95] FALBO, Ricardo A.; TRAVASSOS, G.H. Um estudo sobre ambientes de Desenvolvimento de Software com Suporte Baseado em Conhecimento. In: SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 1995, Canela, RS. **Anais ...** Canela: SBC, 1995. 743p. p.339-350.

[FIS 92] FISCHER, Gerhard; GIRGENSOHN, Andreas; NAKAKOJI, Kumiyo; REDMILES, David. Supporting Software Designers with Integrated Domain-Oriented Design Environments. **IEEE Transations on Software Engineering**, Los Alamitos, Califórnia, v.18, n.6, p.511-522, June 1992.

[JOH 94] JOHNSON, V.M.; CARLIS,J.V. Sharing and Reusing Rules - A Feature Comparison of five Expert Systems Shells. **IEEE Expert** - Intelligent Systems & Their Applications. Los Alamitos, CA, p.3 -16, June 1994.

[KOR 96] KÖRBES, Fábio. **Implementação de um Mecanismo de Herança no Prosoft**. Porto Alegre: CPGCC da UFRGS, 1996. Projeto de Diplomação.

[WAT 81] WATT, David A. **Programming Language Syntax and Semantics**. Great Britain: Prentice Hall,1981.

[NUN 89] NUNES, Daltro J. PROSOFT - **Um Ambiente de Desenvolvimento de Software Estendível**. Porto Alegre: CPGCC da UFRGS, 1989: Publicação Interna.

[NUN 94] NUNES, Daltro J. **PROSOFT.** [s.l.: s.n], 1994. Relatório de Pesquisa interno.

[RIC 93] RICH, Elaine; KNIGNT, Kevin. **Inteligência Artificial**. São Paulo: Makron Books, 1993. 722p.

[WIN 93] WINSTON. **Artificial Intelligence**. [s.l.]: Addison-Wesley, 1993.