A Integração de Conhecimento em um Ambiente de Desenvolvimento de Software

Ricardo de Almeida Falbo*
Guilherme Horta Travassos

COPPE/UFRJ
Programa de Engenharia de Sistemas e Computação
Caixa Postal 68511, Cep 21945-970, Rio de Janeiro - RJ
Email: {rfalbo,ght}@cos.ufrj.br

Resumo

À medida que o processo de desenvolvimento de software torna-se mais complexo, passa a ser imprescindível que os Ambientes de Desenvolvimento de Software (ADSs) ofereçam suporte inteligente para a execução das atividades do processo. Entretanto, a maioria dos ADSs com suporte baseado em conhecimento não trata o conhecimento de maneira integrada, mas sim isoladamente em cada uma de suas ferramentas. Este texto apresenta a abordagem preliminar utilizada na Estação TABA, um ADS desenvolvido segundo o paradigma de objetos, para a integração de conhecimento descrito na forma de regras.

Abstract

As the software development process becomes more complex, Software Engineering Environments (SEEs) needs to offer intelligent support to the execution of the process activities. Nevertheless, most SEEs with knowledge-based support do not consider the knowledge as a internal integrated component, but as a internal part of each environment's tool. This paper describes the initial approach to the knowledge integration, using rules, in the TABA's Workstation, a SEE developed with the object oriented paradigm.

* Professor do Departamento de Informática da Universidade Federal do Espírito Santo em doutoramento na COPPE/UFRJ.

1. Introdução

À medida que os sistemas de software requeridos pelos usuários crescem em complexidade, o processo de desenvolvimento de software também torna-se mais complexo e passa a ser imprescindível a utilização de ferramentas automatizadas para apoiar suas tarefas. Entretanto, ferramentas isoladas oferecem apenas soluções parciais e, assim, é crescente a demanda por Ambientes de Desenvolvimento de Software (ADSs). Tais ambientes buscam combinar técnicas, métodos e ferramentas para apoiar o engenheiro de software na construção de produtos de software, abrangendo todas as atividades inerentes ao processo, tais como planejamento, gerência, desenvolvimento e controle da qualidade.

Sistemas Baseados em Conhecimento (SBCs) têm mostrado potencial para promover integração de atividades em ADSs, através da representação do conhecimento de processo de software. Além disso, SBCs podem tornar esses ambientes mais efetivos no cumprimento de seus objetos, através de suporte inteligente na execução das tarefas. Técnicas de Inteligência Artificial permitem que informações sejam inferidas em contextos inicialmente não previstos. Deste modo, interpretando o conhecimento sobre o processo, suas atividades, métodos e ferramentas, um ambiente pode oferecer assistência inteligente a gerentes e desenvolvedores, guiando, monitorando e auxiliando no uso de ferramentas durante o desenvolvimento de software.

Nesse contexto, a questão da representação de conhecimento em ADSs tem ganhado corpo. Conhecimentos de natureza diversa têm sido representados, tais como conhecimento sobre processos de desenvolvimento [Lott93], conhecimento para avaliação da qualidade [Kajihara93], conhecimento para reutilização [Ostertag92], e conhecimento sobre métodos de especificação e projeto [Rich92], entre outros. Para tal, vários métodos de representação têm sido utilizados, tais como frames, redes semânticas, regras de produção e objetos [Falbo95a]. Contudo, é possível notar que, de maneira geral, o conhecimento é embutido em ferramentas específicas e não fica disponível para o ambiente como um todo.

A maioria dos ADSs trata a integração de ferramentas como uma questão de três dimensões: controle, dados e apresentação. Entretanto, com o aumento da complexidade dos processos de software, faz-se necessário considerar informações de natureza semântica, sem as quais a integração não é plenamente obtida. Assim, é necessário considerar uma visão mais abrangente de integração, observando, além da integração de dados, controle e apresentação,

a integração de conhecimento. O conhecimento, assim como os dados, deve estar disponível e representado de forma única no ambiente, de forma a poder ser acessado por todas as ferramentas que dele necessitarem, evitando redundâncias e inconsistências.

O Projeto TABA visa a construção de uma Estação de Trabalho configurável para Engenharia de Software. Esta Estação permite que engenheiros de software configurem ADSs para diferentes aplicações e projetos. Para resolver o problema da integração, a Estação TABA considera as quatro dimensões discutidas acima. Esta filosofia de integração permite que a informação gerada por uma ferramenta possa ser compreendida por outras.

Este trabalho mostra como a Estação TABA trata a dimensão conhecimento, usando a abordagem orientada a objetos. Inicialmente, optou-se por representar o conhecimento utilizando regras de produção. Uma classe encapsula o conhecimento, que é descrito por um conjunto de regras. Uma máquina de inferência utiliza estas regras para prover a dimensão conhecimento para a Estação. Na seção 2, o ambiente e sua proposta de integração de conhecimento são suscintamente apresentados. A seção 3 discute alguns aspectos da solução adotada e de sua implementação. Finalmente, na seção 4 são apresentadas as conclusões principais do trabalho.

2. A Estação TABA

A Estação TABA [Rocha90] é um meta-ambiente de desenvolvimento de software, isto é, um ambiente capaz de gerar, por meio de configuração e instanciação, outros mais específicos. Assim, é possível prover desenvolvedores de software com ambientes de desenvolvimento que atendam às particularidades de processos de desenvolvimento, domínios de aplicação e projetos específicos.

A estrutura da Estação TABA distribui as funcionalidades necessárias em *componentes*, que conseguem interagir diretamente entre si, permitindo a obtenção de toda a funcionalidade do ambiente [Travassos94]. Dentre os componentes presentes, o *componente Conhecimento* busca incorporar mecanismos de inferência que possibilitam a integração do conhecimento armazenado no ADS. O meta-ambiente se utiliza desse componente para especificar o ADS mais adequado a um dado desenvolvimento. Os ambientes instanciados e suas ferramentas utilizam esse componente para obter informações sobre os processos em curso e o significado dos itens de software que estão sendo manuseados, respectivamente.

A Estação TABA está construída segundo o paradigma da orientação a objetos e a funcionalidade do *componente Conhecimento* é provida pelas classes *Conhecimento*, *RepresentaçãoConhecimento* e suas especializações [Falbo95b], como mostra, segundo o método de Booch [Booch94], a figura 1.

A classe *Conhecimento* provê as funcionalidades básicas para suas especializações, que por sua vez tratam de maneira modular os conhecimentos específicos de processos de desenvolvimento e suas atividades, domínios de aplicação e métodos de construção. Na figura 1, apenas uma das especializações da classe *Conhecimento* está mostrada, a saber a classe *ConhecimentoMétodo*. Esta sub-classe tem por objetivo descrever o conhecimento sobre métodos de desenvolvimento e será utilizada na próxima seção para ilustrar a abordagem de implementação.

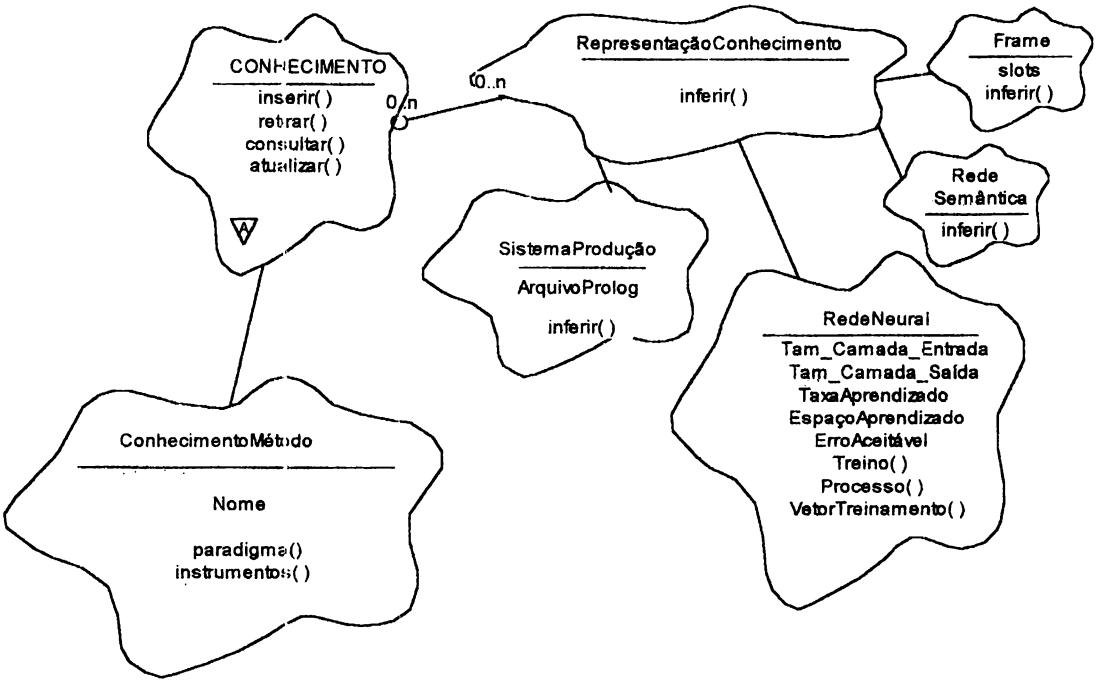

Figura 1: O Componente Conhecimento

A classe *RepresentaçãoConhecimento* e suas especializações provêem o ambiente com a funcionalidade de representar conhecimento de diferentes maneiras. Não existe a priori uma forma melhor, mais eficiente e mais fácil de representar o conhecimento, mas sim uma maior adequação ao problema que se quer resolver. Deste modo, idealmente, um ADS deve

disponibilizar diversas formas de representação. As sub-classes *SistemaProdução*, *Frame*, *RedeSemântica* e *RedeNeural*, aparecem no modelo exatamente para dar ao ambiente esta flexibilidade, permitindo representar o conhecimento através de regras, frames, redes semânticas e redes neurais, respectivamente.

Analisando-se diversos trabalhos, alguns deles brevemente discutidos em [Falbo95a], pode-se notar o intenso uso de regras para representar conhecimento em ADSs. Além disso, algumas ferramentas da própria Estação TABA, como o especificador de ambientes XAMÃ [Aguiar92], já utilizam regras para descrever seu conhecimento. Assim, para integrar esse conhecimento ao ambiente, optou-se por iniciar a implementação permitindo descrições de conhecimento em forma de regras, a partir da classe *SistemaProdução*.

3. A Integração de Conhecimento na Estação TABA

Uma vez que a Estação TABA está implementada em Eiffel [Meyer92], uma linguagem orientada a objetos, faz-se necessário, para definir regras nesse ambiente, integrar os paradigmas lógico e orientado a objetos. Linguagens orientadas a objetos e sistemas baseados em regras cresceram isolados, apesar de terem características relevantes um para o outro. Atualmente, cada vez mais são encontradas aplicações que requerem a integração dos dois paradigmas, como ocorre em ADSs.

Desde o final da década de 80, esforços têm sido feitos para combinar linguagens lógicas e orientadas a objetos. Vários trabalhos têm enfocado a construção de máquinas de inferência como forma de estender a capacidade de linguagens orientadas a objetos para suportar inferência sobre regras, tais como [Franke90] e [Czejdo93]. Entretanto, essa é uma tarefa de elevada complexidade e assim, optou-se, inicialmente, por utilizar a máquina de inferência do SWI-Prolog [Wielemaker95], encapsulada na classe *SistemaProdução*, implementada em Eiffel [Meyer92]. Deste modo, o conhecimento descrito usando regras pode ser definido e utilizado pelas ferramentas do ambiente, desenvolvidas em Eiffel, utilizando a máquina de inferência do SWI-Prolog.

Uma vez que as linguagens SWI-Prolog e Eiffel não oferecem mecanismos que permitam uma comunicação direta entre elas, mas, por outro lado, fornecem mecanismos para comunicação com código externo escrito na linguagem C, foi adotada uma abordagem de interfaceamento SWI-Prolog/C/Eiffel.

Serviços definidos em Eiffel encapsulam a máquina Prolog. As operações necessárias à comunicação entre as linguagens C e SWI-Prolog são encapsuladas em métodos da classe *SistemaProdução* não acessíveis aos demais objetos. Essas operações são definidas como funções externas e utilizam recursos para interfaceamento com a linguagem C, fornecidos pelo SWI-Prolog. Garante-se, assim, a integridade do sistema, já que apenas serviços para inferências são oferecidos aos usuários da classe *SistemaProdução*.

3.1 - Um Assistente Inteligente para o Método de Booch

Para exemplificar a solução adotada, apresentamos, a seguir, parte de uma aplicação que utiliza a abordagem anteriormente descrita: um assistente inteligente para apoiar engenheiros de software no uso do método de construção orientado a objetos proposto por Booch [Booch94].

O conhecimento sobre o método de Booch é descrito como um objeto da classe *ConhecimentoMétodo*, utilizando como forma de representação um sistema de produção, ou seja, um objeto da classe *SistemaProdução*.

A classe Eiffel *ConhecimentoMétodo*

A figura 2 mostra parte da interface da classe *ConhecimentoMétodo*. Note que um dos atributos desta classe é a sua base de conhecimento. Com esta abordagem, o conhecimento não fica intrínseco a um assistente, mas sim disponível para qualquer ferramenta do ambiente. Para utilizá-lo, basta que uma ferramenta utilize os serviços disponibilizados pela classe. Além disso, consegue-se obter modularidade, já que o conhecimento sobre o método de Booch está encapsulado em um objeto.

Quando um serviço precisa que uma inferência seja feita para obter uma resposta, ele próprio envia mensagens para os serviços de inferência da classe *SistemaProdução*. Isto ocorre, por exemplo, com o serviço *paradigma*, que consulta a base de conhecimento para retornar o paradigma do método de construção. Para tanto, tem em seu corpo uma chamada ao serviço *inferir* da classe *SistemaProdução*. Já o serviço *criar*, responsável pela criação de um objeto *ConhecimentoMétodo*, faz uma chamada ao serviço *criar* da classe *SistemaProdução*.

```
class ConhecimentoMetodo
  creation  criar
  feature
    nome: STRING;
    baseconhecimento: SistemaProducao;
    criar(met: STRING)
    paradigma(): STRING
end
```

Figura 2: Um extrato da classe *ConhecimentoMétodo*

A classe Eiffel *SistemaProdução*

A figura 3 mostra a interface da classe *SistemaProdução* que disponibiliza os serviços para comunicação com a máquina de inferência do SWI-Prolog. O serviço *criar* associa um arquivo Prolog ao atributo *arquivoProlog*. O serviço *inferir* representa o processo de inferência oferecido pela classe. O predicado, sua aridade e os argumentos são passados como parâmetros, conforme mostrado no esquema da figura 4. Os métodos *inicia_pl*, *consulta_pl*, *inferir_pl* e *finaliza_pl* são métodos externos definidos em C. É importante realçar que são métodos privados[1] da classe e, portanto, não são visíveis aos seus usuários. Eles são chamados pelo método *inferir* para ativar a máquina de inferência do SWI-Prolog, disponibilizar a base de conhecimento para consultas, efetuar o processo de inferência propriamente dito e finalizar a máquina Prolog, respectivamente.

```
class SistemaProducao
  creation criar
  feature
    arquivoProlog: STRING;
    criar(nomearq: STRING)
    inferir(pred: STRING; n: INTEGER; par: ARRAY[STRING]): STRING
  feature {NONE}
    inicia_pl(): BOOLEAN is external      "C" end;
    finaliza_pl() is external "C" end;
    consulta_pl(nomearq: ANY): BOOLEAN is external "C" end;
    inferir_pl(pred: ANY; aridade: INTEGER; par: ANY): ANY is
      external "C" end;
end
```

Figura 3: A classe *SistemaProdução*

[1]cláusula NONE de Eiffel

A Realização de uma Inferência

A figura 4 ilustra como um processo de inferência é realizado. Neste exemplo, o objeto *"booch"* da classe *ConhecimentoMétodo* utiliza a máquina de inferência do SWI-Prolog para informar seu paradigma a um objeto cliente. Para tal, quando o serviço *paradigma* é ativado, ele envia uma mensagem para seu objeto *SistemaProdução*, solicitando uma inferência.

O objeto da classe *SistemaProdução* possui uma referência ao arquivo Prolog que contém, de fato, o conhecimento. O serviço *inferir* dessa classe é então ativado e dispara o serviço privado *inferir_pl*. Esse método é, na realidade, uma função externa escrita em C, que estabelece a comunicação com a máquina Prolog, responsável pelo processo de inferência.

A função externa C mapeia as informações recebidas para o formato requerido pelo SWI-Prolog, responsável real pelo processo de inferência. Os resultados da inferência são retornados à função externa C, que por sua vez faz o mapeamento necessário para retorná-los ao mundo de objetos de Eiffel.

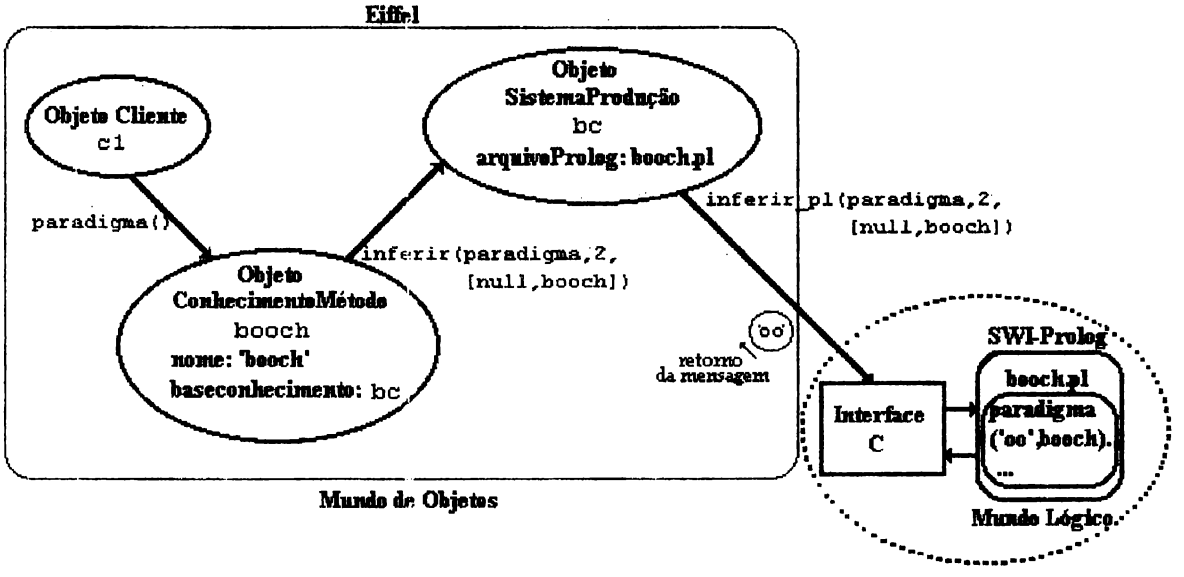

Figura 4: Mundo de Objetos x Mundo Lógico

A figura 5 apresenta um extrato do arquivo Prolog *booch.pl*, que contém a base de conhecimento sobre o método de Booch, usado no processo de inferência.

```
paradigma('orientado a objetos', booch).
instrumento('diagrama de classes', booch).
instrumento('diagrama de transicao de estados', booch).
instrumento('diagrama de categorias', booch).
instrumento('diagrama de objetos', booch).
instrumento('diagrama de modulos', booch).
instrumento('diagrama de processos', booch).
...
```

Figura 5: Um extrato do arquivo Prolog *booch.pl*

3.2 - Considerações sobre a solução adotada

É importante observar que os objetos da classe *SistemaProdução* enviam mensagens para a interface C que funciona como um objeto externo[2] que executa um método e retorna um objeto. Assim, o impacto da interface no mundo de objetos é minimizado.

Entretanto, deve-se realçar que a abordagem utilizada apresenta uma limitação: enquanto Eiffel manipula objetos, SWI-Prolog trabalha com predicados, átomos e outras estruturas de linguagens lógicas, o que impõem uma barreira conceitual entre os mundos orientado a objeto e lógico que dificulta a integração.

Idealmente, não haveria necessidade de se representar fatos como predicados, bastando representar as regras. Os fatos estariam definidos pelos estados dos objetos. Assim, em um processo de inferência, a máquina de inferência interpretaria os fatos a partir dos estados dos objetos e utilizaria as regras para compor os processos de unificação. Uma vez que SWI-Prolog não é capaz de acessar objetos, é necessário representar fatos como predicados Prolog e, para acessar essa informação no mundo de objetos, são definidos serviços que sempre resultam em um processo de inferência.

Para garantir um menor impacto no mundo de objetos, funções em C podem ser desenvolvidas para criar objetos Eiffel, de modo que o retorno das funções externas de Eiffel sejam objetos. Essa estratégia não foi implementada devido a problemas decorren-tes na identificação de métodos Eiffel a partir da linguagem C. À medida que estes problemas forem corrigidos, esta estratégia será colocada em prática.

[2]representado pela linha pontilhada na figura 4.

4. Conclusões

A integração de conhecimento é um aspecto fundamental para o sucesso da aplicação de técnicas de Inteligência Artificial em Ambientes de Desenvolvimento de Software. Deste modo, o conhecimento necessário ao processo de desenvolvimento deve ser disponibilizado para o ambiente e as ferramentas que o compõem, a partir de uma base de conhecimento modular. O conhecimento não deve ficar embutido em uma ferramenta ou assistente, mas sim estar disponível de forma global para o ambiente.

Um primeiro passo foi dado nesta direção: conhecimentos específicos são encapsulados como objetos das respectivas especializações da classe *Conhecimento*, como ilustrado na seção 3, para o conhecimento sobre métodos de construção.

Idealmente, um ADS deve disponibilizar diversas formas para representação do conhecimento. Isto está sendo considerado no âmbito do projeto TABA. No entanto, a corrente versão permite tratar de maneira integrada apenas o conhecimento descrito utilizando regras.

A solução implementada viabiliza a integração na Estação TABA do conheci-mento descrito na forma de regras. Entretanto o uso desta abordagem não permite um tratamento uniforme do mundo de objetos, isto é, ocorre uma quebra do mundo de objetos para a realização de inferências usando a máquina de inferência do SWI-Prolog. Assim, o desenvolvimento de uma máquina de inferência integrada ao sistema orientado a objetos está sendo considerada como uma próxima abordagem de implementação. Esse novo enfoque eliminará o problema de transferência, e conseqüentemente tradução, de dados relevantes entre a aplicação orientada a objetos e o sistema baseado em regras, além de permitir uma representação única e conveniente para informações factuais através de objetos.

Agradecimentos

Os autores agradecem ao CNPq e à CAPES pelo apoio financeiro que permitiu a realização deste trabalho.

Referências

[Aguiar92] Aguiar, T.C.; Um Sistema Especialista de Suporte à Decisão para Planejamento de Ambientes de Desenvolvimento de Software, Tese de Doutorado, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Março 1992.

[Booch94] Booch,G.; Object-Oriented Analysis and Design with Applications, 2nd Edition, Benjamin Cummings, 1994

[Czejdo93] Czejdo, B., Eich, C.F, Taylor,M.; "Integrating Sets, Rules and Data in an Object_Oriented Environment", IEEE Expert, Fevereiro 1993.

[Falbo95a] Falbo, R.A.; Técnicas de Inteligência Artificial aplicadas a Ambientes de Desenvolvimento de Software, Relatório Técnico ES-339/95, COPPE/UFRJ, Abril 1995.

[Falbo95b] Falbo, R.A. e Travassos, G.H.; "Um Estudo sobre Ambientes de Desenvolvimento de Software com Suporte Baseado em Conhecimento", Anais do XXI CLEI/XV Congresso da SBC, Agosto 1995.

[Franke90] Franke, D.W.; "Imbedding Rule Inferencing in Applications", IEEE Expert, Dezembro 1990.

[Kajihara93] Kajihara, J., et al.; "Learning from Bugs", IEEE Software, Setembro 1993.

[Lott93] Lott, C.M.; "Process and Measurement support in SEEs", ACM SIGSOFT Software Engineering Notes, vol. 18, n° 4, Outubro 1993.

[Meyer92] Meyer, B.; *Eiffel, The Language*, Prentice Hall Object Oriented Series, 1992.

[Ostertag92] Ostertag, E., Hendler, J., Prieto_Díaz, R. and Braun, C.; "Computing Similarity in a Reuse Library System: An AI-Based Approach", ACM Transactions on Software Engineering and Methodology, vol. 1, n° 3, Julho 1992.

[Rich92] Rich, C. and Feldman, Y.A.; "Seven Layers of Knowledge Representation and Reasoning in Support of Software Development", IEEE Transactions on Software Engineering, vol. 18, n° 6, Junho 1992.

[Rocha90] Rocha, A.R.C, Aguiar, T.C. and Souza, J.M.; "TABA: a heuristic workstation for software development", COMPEURO'90, Israel, Maio 1990.

[Travassos94] Travassos, G.H, Rocha, A.R.C.; "Um Modelo para Construção e Integração de Ferramentas", VIII Simpósio Brasileiro de Engenharia de Software, Outubro 1994.

[Wielemaker95] Wielemaker, J.; *SWI-Prolog 2.0 - Reference Manual*, Abril 1995.