Um Sistema de Gerenciamento de Projetos de Software

Lucio Dimas dos Santos Mendes

Sul América Teleinformática
Departamento de Desenvolvimento
Av. Getulio Vargas, 3560 - Curado
50.730 - Recife - PE - Brasil

Resumo

Este trabalho apresenta o protótipo de um sistema de manutenção e monitoração de informações relativas a projetos de software. O sistema é chamado sistema SKB (Software Knowledge Base System) e o protótipo escrito em Prolog é chamado sistema SKBP ou simplesmente SKBP.

A idéia básica introduzida pelo sistema SKB é de que uma abordagem unificada usando alguns poucos conceitos matemáticos pode ser utilizada para descrever os vários elementos envolvidos em projetos de software, suas relações e as condições que devem satisfazer.

Descrevemos os sistema SKBP, incluindo suas operações mais importantes, e mostramos um exemplo simples de um problema de gerenciamento de projetos de software no qual a abordagem proposta é utilizada.

Abstract

This paper presents the prototype of a system for maintaining and monitoring information about software projects. The system is called SKB System (Software Knowledge Base System) and the prototype, written in Prolog, is called SKBP System or just SKBP.

The basic idea introduced by the SKB System is that a unified approach using a few mathematical concepts may be used to describe the various elements involved in software projects, their relationships and the conditions they must satisfy.

We describe the SKBP System, including its most important operations, and we show a simple example of a software project management problem in which the proposed approach is used.

1. Introdução

O presente trabalho descreve o protótipo de um sistema de gerenciamento de projetos de software escrito em Prolog. Esse protótipo foi desenvolvido na Universidade da Califórnia em Santa Barbara de agosto de 1985 a junho de 1986, sob orientação do professor Bertrand Meyer e resultou na tese de mestrado "A Prolog Prototype of the Software Knowledge Base System" [8].

O protótipo foi escrito em C-Prolog, um interpretador Prolog escrito em C [12], e testado em um ambiente VAX/Unix.

O que descrevemos a seguir se refere mais ao sistema proposto e menos aos detalhes de implementação do protótipo sendo que alguns aspectos apresentados (em particular a sintaxe concreta usada) são derivados diretamente do protótipo que por sua vez são baseados em Prolog.

A implementação desse protótipo teve dois objetivos principais: primeiro prover um ambiente de teste para a idéia de se usar um banco de conhecimento em gerenciamento de projetos de software e segundo, investigar a adequação do uso de uma linguagem lógica para a implementação deste sistema. Estávamos interessados particularmente em dois aspectos: primeiro, eficiência e segundo, até que ponto as funções previstas poderiam ser implementadas.

1.1 - Motivação e Objetivos

A idéia da proposta do sistema SKB está associada à crescente necessidade de ferramentas para gerenciamento de projetos de software. Essa necessidade é justificada pela dificuldade de se controlar os muitos componentes heterogêneos que interagem entre si em um grande projeto de software.

Diversas propostas têm sido feitas de sistemas para auxiliar nas diferentes etapas do desenvolvimento de software, incluindo métodos de especificação e "design", procedimentos de documentação e ferramentas para gerenciamento de configuração.

Duas importantes inovações do sistema SKB são: primeiro, não se trata de uma ferramenta de software, mas, de um ambiente no qual diferentes ferramentas podem ser construídas, e segundo, ele não impõe uma metodologia em particular mas foi concebido de forma a se adaptar às metodologias previamente adotadas por diferentes usuários.

A fim de obter tal flexibilidade, o sistema SKB permite que o usuário crie os elementos necessários à sua aplicação específica.

Dentro do ambiente SKB, um projeto de software é visto como uma coleção de objetos relacionados de diversas maneiras diferentes com outros objetos. Adicionalmente, um conjunto de condições devem ser satisfeitas para que um sistema esteja em um estado chamado "consistente".

O sistema SKB cria e mantém modelos de projetos de software chamados bancos de conhecimento de software (skbs), onde são armazenadas todas as informações relativas aos projetos. Usaremos as iniciais skb para designar um banco de conhecimento de software e "sistema SKB" quando nos referirmos ao programa.

Um skb é escrito em termos de algumas poucas entidades: átomos, atributos, relações e restrições. Cada objeto de um projeto de software é representado por um átomo. Atributos descrevem objetos implicitamente informando o que os objetos "possuem" ao invés de o que os objetos "são". Relações são usadas para descrever as interdependências entre os objetos. Atributos e relações são definidos pelo usuário. Finalmente restrições definem a semântica de projetos de software e são condições que devem ser satisfeitas pelos átomos, atributos e relações para que um skb esteja em um estado consistente.

O sistema SKB também incorpora o conceito de ação: uma sequência de comandos associada a uma restrição. Uma ação associada a uma restrição inválida deve ser executada a fim de restaurar sua validade.

Uma linguagem usada para criar e acessar os elementos de um skb é implementada pelo sistema SKB, assim como comandos para verificar consistência de skbs e executar ações.

Na seção seguinte são definidas as entidades usadas pelo sistema SKB.

1.2 - Definições

Conforme mencionado anteriormente, o sistema SKB usa alguns poucos conceitos matemáticos na descrição de projetos de software: átomos, atributos, relações e restrições.

Seguem definições desses conceitos.

Átomos:

Átomos são elementos usados para representar componentes de um projeto de software. As propriedades de um componente são descritas pelos atributos associados a seu átomo correspondente e pelas relações que conectam esse átomo a outros átomos. Tipicamente, átomos seriam usados para representar componentes tais como: programas, especificações, dados, pessoas, cronogramas, etc.

Atributos:

Atributos são os elementos que definem as propriedades de um átomo. Atributos assumem um dos seguintes tipos primitivos: inteiro, booleano, cadeia de caracteres (string) ou tempo. Atributos assumem valores ao serem associados a átomos. Por exemplo a operação:

```
assign (a, classe, 2).
```

associa o valor 2 ao atributo "classe" do átomo a. O atributo "classe" deve ter sido anteriormente declarado como sendo do tipo inteiro da seguinte forma:

```
attribute (classe, integer).
```

Relações:

As interdependências entre átomos são expressas por meio de relações binárias, o único tipo de relação aceita pelo sistema SKB. Relações são usadas para "conectar" átomos. Por exemplo a operação:

```
link (a, usa, b).
```

cria uma conexão (link) entre os átomos a e b usando a relação "usa". A relação "usa" deve ter sido anteriormente declarada da seguinte forma:

```
relation (usa).
```

Restrições:

Restrições são definidas por condições escritas como expressões booleanas envolvendo os elementos que descrevem um projeto de software. O sistema SKB implementa um grande número de operações que podem ser usadas na definição de restrições. Dois comandos podem ser usados para se verificar restrições: "check" e "correct". Ambos os comandos avaliam a expressão booleana que define a restrição. Caso o valor da expressão seja falso, o comando "check" mostra uma mensagem associada com a restrição e o comando "correct" executa a ação associada com a restrição. Por exemplo, a restrição inclusão definida como:

```
constraint (inclusão, usa subset depende,
            ['mensagem'], ação).
```

estabelece que a relação "usa" deve estar contida na relação "depende".

Ações:

Ações são sequências de comandos associados com restrições. Esses comandos podem ser comandos do sistema SKB ou comandos do sistema operacional. Quando uma restrição é inválida, a execução da sua ação associada deve restaurar sua validade.

Um exemplo de ação é mostrado na seção 4.

2 - Descrição de Projetos de Software

O sistema SKB é usado para criar modelos de projetos de software, isto é, o sistema manipula representações de objetos de um projeto e não os objetos em si. Portanto, um documento armazenado fisicamente em um arquivo em um sistema operacional poderia ser representado por um átomo tendo, por exemplo, um atributo "nome_arquivo" do tipo string cujo valor seria um string contendo o nome do arquivo no sistema operacional. Outros atributos tais como tipo do arquivo, data de criação, etc. poderiam também ser associados a esse átomo.

Essa separação entre um projeto de software e sua representação foi um decisão de projeto visando garantir a independência do sistema com relação a tipos de objetos e metodologias, bem como por simplicidade: dessa forma, o sistema não precisaria implementar comandos para manipulação de objetos reais de um projeto, tais como editores de texto ou compiladores. Objetos reais seriam manipulados por comandos do sistema operacional, acionados através de "portas" do sistema SKB. Por isso, o sistema SKB pode ser visto como um gerenciador de banco de dados usado para manter informacões sobre projetos de software. Este fato justifica a escolha de Prolog como a linguagem para implementação do sistema SKBP: Prolog tem sido usado com sucesso em implementações de sistemas de bancos de dados, como por exemplo em [2], [6] e [7].

O sistema SKB foi implementado em dois níveis: o nível de descrição e o nível de comando. O nível de descrição é usado para criar descrições de projetos de software em partes chamadas "unidades". O nível de comando implementa operações para acessar e combinar unidades já existentes para formar novas unidades, para avaliar restrições e executar ações.

A descrição de um projeto de software envolve as seguintes etapas:

- declaração dos átomos que vão representar os objetos e a associação de "propriedades" a esses átomos através de valores dados aos seus atributos.

- conexão dos átomos representando objetos usando relações previamente definidas.

- descrição da semântica do projeto por restrições que estabelecem condições a serem satisfeitas para que o projeto esteja em um estado consistente.

- definição das ações a serem executadas no caso de cada uma das restrições ser violada.

Na prática, estas etapas devem ser consideravelmente simplificadas com o uso de "macros" que encapsulam uma ou

mais operações do sistema SKB e que definem as entidades usadas em um ambiente em particular. Unidades contendo essas definições seriam combinadas e novas declarações seriam adicionadas para se criar o modelo de um projeto de software.

2.1 - O Nível de Descriçao

As operações aceitas pelo nível de descrição são usadas para criar e remover elementos de um skb, bem como para associar atributos a átomos e conectar átomos usando relações. As principais operações do nível de descrição são:

```
atom (nome_átomo).

attribute (nome_atributo, tipo_atributo).

relation (nome_relação).

constraint (nome_restrição, expressão_booleana,
            mensagem, nome_ação).

action (nome_ação, lista_de_operações).

define (nome_macro (parametros_formais),
        macro_definição).
```

Cada uma das cinco primeiras operações é usada para criar o elemento de um skb de mesmo nome. A operação "define" é usada para definir macros. Existe também uma operação de remoção para cada um dos elementos acima.

Algumas explicações:

- o nome de cada um dos elementos de um skb deve ser único.

- tipo_atributo deve ser um dos seguintes: integer, boolean, string ou time.

- expressão booleana é uma expressão booleana válida usando os operadores discutidos na seção 3.

- mensagem é uma lista de elementos que são expressões ou strings. Esta mensagem é mostrada no terminal quando o comando check (veja seção 2.2) for executado e a restrição for falsa.

- lista_de_operações de uma ação é uma lista de strings ou operações do sistema SKB. Essa lista de operações é executada quando o comando correct (veja seção 2.2) for executado e a restrição for falsa. As operações do sistema SKB são executadas e as strings são passadas como mensagens para o sistema operacional.

- parâmetros_formais é a lista de parâmetros formais de uma macro, composta de nomes começados com o símbolo "$".

- macro_definição pode ser qualquer operação ou expressão válida no sistema SKB.

O sistema SKB implementa também algumas operações encontradas em muitas linguagens de programação tais como: "if_then", "if_then_else" e "while_do", além da operação "iterate" que é uma variação do comando "for".
Essas operações são escritas como mostrado abaixo:

```
if (expressão_booleana, lista_de_operações).

if (expressão_booleana, lista_de_operações,
    lista_de_operações).

while (expressão_booleana, lista_de_operações).

iterate (lista_expressão, lista_de_operações).
```

As tres primeiras operações têm semânticas análogas às dos comandos de linguagens de programação como Pascal. A operação "iterate" executa sua lista_de_operações tantas vezes quantas forem os elementos de lista_expressão, em cada execução substituindo o símbolo especial ('$') por um elemento da lista, na mesma ordem em que os elementos aparecem na lista.
Os elementos criados durante uma sessão do nível de descrição constituem uma unidade e pode ser armazenada em um arquivo do sistema operacional ao término da sessão. A operação "use" do nível de descrição pode ser utilizada para incluir as cláusulas de um arquivo (unidade), contendo elementos de um skb, à unidade atual durante uma sessão do nível de descrição.

2.2 - O Nível de Comando

No nível de comando é definido o conceito de espaço de avaliação. Um espaço de avaliação contém as declarações de todos os elementos de um skb que serão acessados na determinação de valores de expressões que aparecem em operações do nível de comando. Espaços de avaliação são criados carregando-se unidades armazenadas em disco ou pela execução de operações do nível de descrição.
No nível de comando são implementadas operações para criar espaços de avaliação, verificar restrições, executar ações e listar o conteúdo de espaços de avaliação.
As operações usadas para criar espaços de avaliação são:

```
load (nome_unidade).

clear.
```

A operação "load" carrega todas as cláusulas contidas em uma unidade no espaço de avaliação atual. A operação "clear" remove todas as cláusulas do espaço de avaliação atual.

As operações seguintes são usadas para verificar restrições e executar ações:

```
check (nome_restrição)

checkall.

correct (nome_restrição).

correctall.

do (nome_ação).
```

A operação "check" que tem como argumento o nome de uma restrição, avalia a expressão booleana associada com essa restrição usando os elementos do espaço de avaliação atual e mostra a mensagem associada com a restrição, caso o valor da expressão seja falso. A operação "checkall" executa a operação "check" para cada uma das restrições contidas no espaço de avaliação atual.

A operação "correct" é análoga à operação "check". A expressão booleana associada com a restrição é avaliada e a ação associada com a restrição é executada se o valor da expressão for falso. A operação "correctall" executa a operação correct para cada uma das restrições contidas no espaço de avaliação atual. A operação "do" é usada para executar uma ação.

3 - Operadores e Expressões

Os tipos básicos de expressões do sistema SKB são átomo e os quatro tipos primitivos seguintes: inteiro, booleano, string e tempo. Átomo é o tipo de expressões que produzem como resultado um átomo. Outros tipos aceitos pelo sistema são obtidos a partir desses tipos básicos através da aplicação dos operadores de tipo: par, conjunto e lista.

O tipo de uma relação é conjunto de pares de átomos (escrito como conjunto(par(átomo, átomo))). Atributos são conjuntos de pares de átomo e um tipo primitivo, o tipo do atributo. Por exemplo, o tipo de um atributo inteiro é: conjunto(par(átomo, inteiro)).

Não são implementadas no sistema SKB declarações de variáveis ou constantes e a verificação da corretude de expressões com relação a tipos (type check) é feita em tempo

de execução, usando uma abordagem baseada em teoria proposta em [11].

O sistema SKB implementa diversas operações que podem ser usadas para se escrever expressões. Essas expressões podem ser dos tipos básicos ou pares de tipos básicos ou ainda conjuntos ou listas de tipos básicos ou pares de tipos básicos.

O que segue é uma longa lista de operadores aceitos pelo sistema SKB, seguido de uma breve descrição.

Operadores aritméticos:

+ :	adição aritmética
- :	subtração
* :	multiplicação
/ :	divisão
card :	cardinalidade de conjuntos
length :	comprimento de listas

Operadores booleanos e relacionais:

and :	e lógico
or :	ou lógico
not:	negação
= :	igual
/= :	diferente
> :	maior (aplicável a tipos inteiro, string e tempo)
>= :	maior ou igual
< :	menor
<= :	menor ou igual
member :	pertinência de elementos e conjuntos
subset :	inclusão de conjuntos
sublist :	inclusão de listas

Operadores tipo tempo:

cur_time :	a hora atual

Operadores tipo lista:

front :	todos menos o último elemento de uma lista não vazia
back :	todos menos o primeiro elemento de uma lista não vazia
&& :	concatenação de listas
slist:	a lista contendo todos os elementos de um conjunto em uma ordem arbitrária

Operadores tipo conjunto:

union :	união de conjuntos
inter :	intersecção de conjuntos
-- :	diferença de conjuntos
-/- :	diferença simétrica de conjuntos
domain :	domínio de relação
range :	imagem de relação
& :	composição de relações
++ :	fechamento transitivo de relações
** :	fechamento reflexivo e transitivo de relações
inv :	relação inversa
\\ :	restrição de uma relação a um conjunto (domínio)
// :	co-restrição de uma relação a um conjunto (imagem)

Outros operadores:

@ :	operador valor-do-atributo (aplicado a um átomo e um atributo produz o valor do atributo para o átomo dado)
first :	primeiro elemento de uma lista não vazia
last :	último elemento de uma lista não vazia
? :	operador escolha (aplicado a um conjunto produz um elemento arbitrário do conjunto)

além dos operadores mencionados acima, o sistema SKB aceita conjuntos e listas definidos por extensão, incluindo elementos de um mesmo tipo entre chaves e colchetes, respectivamente, além de conjuntos definidos por compreensão na forma:

```
{ x in U ! p }
```

onde:

x - é uma variável livre,
U - é um conjunto universo e
p - é uma propriedade (predicado) envolvendo a variável x.

Por exemplo o conjunto de todos os átomos no domínio da relação "usa" com classe 2 pode ser definido por compreensão como:

```
{ x in domain(usa) ! x @ classe = 2 }
```

A lista de operadores acima inclui os operadores mais importantes aceitos pelo sistema SKB. Um exemplo é mostrado a seguir incluindo alguns desses operadores, para ilustrar a utilização do sistema SKB na solução de problemas de gerenciamento de projetos de software.

4 - Utilização do Sistema SKB em Projetos de Software

Nesta seção, a título de exemplo, discutimos a solução de um problema de gerenciamento de projetos de software simplificado.

O Problema:

Desejamos saber em qualquer momento se os programas de um projeto de software foram testados com todos os seus dados de teste. Assumimos que existem somente dois tipos de objetos: programs e dados de teste. Cada programa possui um ou mais dados de teste. A condição de consistência para este projeto é de que todos os programas devem ser executados com sucesso usando como entrada todos os seus arquivos de dados de teste. Adicionalmente, programas podem ser revisados, caso em que todos os testes para este programa devem ser repetidos. Finalmente, neste exemplo, um teste produz um simples resultado: passa ou não passa.

A Solução:

Os objetos do projeto, programas e dados de teste serão diferenciados por dois atributos declarados como:

```
attribute (nome, string).
attribute (tipo, string).
```

O primeiro atributo dá o nome do arquivo contendo o objeto no sitema operacional e o segundo dá o tipo do arquivo: programa ou dado de teste.

A fim de expressar o critério de consistência, precisamos de duas relações:

```
relation (dado_de_teste_de).
relation (testado_com).
```

A relação "dado_de_teste_de" conecta cada objeto-programa a cada um dos seus objetos-dados-de-teste. A relação "testado_com", por sua vez, conecta cada objeto-programa a cada um dos objetos-dados-de-teste para os quais o programa foi testado com sucesso.

O critério de consistência é claramente satisfeito quando as duas relações forem iguais, isto é, quando cada programa tiver sido testado com sucesso com cada um de seus dados de teste. Assim podemos escrever a restrição que define esse critério de consistência da seguinte forma:

```
        constraint (todo_testado,
                    dado_de_teste_de = testado_com,
                    ['Programas não testados: ',
  range (nome \\ domain (dado_de_teste_de -- testado_com))],
                    gera_teste).
```

A condição de consistência é descrita, conforme já mencionado, pela igualdade das relações dados_de_teste_de e testado_com. A expressão entre colchetes, a mensagem associada com a restrição, é composta de um texto entre apóstrofes e da expressão

```
range (nome \\ domain (dado_de_teste_de -- testado_com))
```

que quando avaliada produz o conjunto dos nomes (strings) de todos os programas não totalmente testados. Na expressão acima pode ser observado que o atributo "nome" é tratado como uma relação.

O termo (dado_de_teste_de -- testado_com) é a diferença das duas relações que deve ser não vazia no caso de a restrição "todo_testado" estar sendo violada. O domínio dessa diferença produz o conjunto que contém somente os átomos representando programas não totalmente testados. A restrição (operador "\\") da "relação" nome à esse domínio produz nova relação contendo os pares (átomo, nome) onde átomo representa um programa não totalmente testado. Finalmente a imagem (range) dessa relação é o conjunto dos nomes (strings) dos programas não totalmente testados.

A ação "gera_teste" associada com a restrição "todo_testado" é definida da seguinte forma:

```
        action (gera_teste,
        [
  iterate (slist(domain(dado_de_teste_de -- testado_com)),
           [ testa (#) ]
        )).
```

Esta ação contém um único comando iterate que chama a macro "testa" para cada um dos átomos representando um programa não totalmente testado. Esses átomos são encontrados no conjunto domínio da diferença das relações dado_de_teste_de e testado_com. O operador "slist"

transforma esse conjunto em uma lista contendo os mesmos átomos.

A macro "testa", definida como:

```
define (testa ($p),
iterate(slist(range(dade_de_teste_de--testado_com)\\ {$p}),
    [
     $p @ nome && " < " && # @ nome,
     link ($p, testado_com #)
    ])).
```

tem como argumento um átomo ($p) e, executa o programa representado por este átomo com todos os seus dados de teste, para os quais esse programa não foi ainda testado. Para um programa não testado, representado pelo átomo $p, a lista de átomos representando dados de teste para os quais ele não foi testado é dada por:

```
slist ( range (( dado_de_teste_de -- testado_com) \\ {$p}))
```

Mais uma vez a diferença entre as duas relações aparece, desta vez como argumento do operador restrição ("\\"), que tem como segundo argumento o conjunto cujo único elemento é o átomo em questão ({$p}).

O teste propriamente dito é realizado pela execução do comanto obtido pela concatenação ("&&") do string contendo o nome do programa ($p @ nome) com o string " < " (operador redirecionamento, no Unix) concatenado ainda com o string contendo o nome do arquivo de dados de teste (# @ nome).

Após a execução do programa, o sistema SKB atualiza o modelo do projeto de software criando um "link" usando a relação "testado_com", entre os átomos representando o programa testado e seu dado de teste.

Finalmente mostramos a definição da macro "revisa" que atualiza o skb quando um programa é revisado, em cujo caso o programa deve ser testado novamente com todos os seus dados de teste.

```
define (revisa ($p),
[
    [iterate (slist(range(testado_com \\ {$p})),
     delete_link ($p, testado_com, #)
    ])
]).
```

O efeito desejado é obtido removendo todas as conexões usando a relação "testado_com" ligando o átomo representando o programa revisado aos átomos representando os seus dados de teste. O corpo da macro inclui a operação "delete_link" do nível de descrição, não discutida anteriormente, que remove uma conexão entre dois átomos por uma relação. Os demais operadores foram discutidos acima. Deixamos a cargo do leitor a interpretação da macro "revisa".

5 - Conclusão

Na tese "A Prolog Prototype of the Software Knowledge Base System", dois exemplos de aplicação do sistema SKB a problemas reais são mostrados. Essas aplicações são uma implementação do programa "Make" do Unix [4], uma ferramenta de manutenção de versões atualizadas de programas e uma implementação do programa RCS [13] de gerenciamento de configuração.

A implementação de diferentes aplicações usando o sistema SKB serviu para mostrar a flexibilidade do sistema com relação a aplicações, um dos importantes aspectos da abordagem utilizada no sistema SKB. A experiência mostrou que as operações implementadas no sistema são suficientes para criar modelos de projetos de software grandes e complexos, exatamente aqueles que mais requerem o uso de ferramentas de software para seu gerenciamento.

Conforme discutido no início deste artigo, o sistema SKB não se propõe a ser uma ferramenta de software mas um ambiente no qual diferentes ferramentas podem ser construídas. Uma consequência direta desse fato é que um certo grau de "programação" do sistema se faz necessário para adaptá-lo a uma aplicação em particular. Como pode ser observado no exemplo discutido na seção anterior, mesmo um problema simples requer um certo conhecimento matemático para criar o seu modelo no sistema SKB. Esse fato evidencia a importância do uso de macros como mecanismo de encapsulamento de conceitos matemáticos usados na descrição de projetos de software do usuário final, a quem não necessariamente interessam detalhes de implementação. Esse usuário final teria à sua disposição bibliotecas de macros projetadas para manipular os elementos de sua aplicação e que simultaneamente atualizariam o modelo de seu projeto de software no sistema SKB.

No campo das limitações, eficiência em termos tanto de velocidade como de utilização de memória, aparecem como problema crítico no protótipo implementado em Prolog. A notação utilizada, por outro lado, torna difícil a criação de modelos de grandes projetos, sugerindo a implementação de uma interface gráfica para entrada de comandos para o sistema. Finalmente a criação de bibliotecas de elementos usados em projetos de software reais se mostra extremamente importante para tornar o sistema SKB útil na prática.

6 - Referências

1. Clocksin W., Mellish C. - Programming in Prolog, Springer-Verlag Berlin Heildelberg New York (1981).

2. Dahl V. - On Databases Systems Development Through Logic, ACM Transactions on Database Systems, Vol. 7, No. 1 (March 1982), pp. 102-123.

3. Delort J. - Software Knowledge Base: The Calculus of Relations Attributes and Constraints, Specification and Algorithms, Master's Thesis - UCSB, (October 1985).

4. Feldman S. - Make - A Program for Maintaining Computer Programs, Bell Laboratories, Murray Hills - NJ (August 1978).

5. Glikson X. - Software Project and Configuration Management: The SKB Approach, Master's Thesis, UCSB, (November 1985).

6. Kowalsky R. - Logic for Data Description on Logic for Data Bases, Gallaire H., Minker J. (editors), Plenum Press NY (1978).

7. Li D. - A Prolog Database System, Research Studies Press Ltd. John Wiley, New York (1984).

8. Mendes L. - A Prolog Prototype of the Software Knowledge Base System, Mastes's Thesis, UCSB (June 1986)

9. Meyer B. - First Steps Towards A Relational Theory of Software, Department of Computer Science - UCSB (1984).

10. Meyer B. - The Software Knowledge Base, Proceedings of the IX International Conference in Software Engineering - London, (August 1985).

11. Milner R. A Theory of Polymorphism in Programming, Journal of Computer and System Sciences, Vol. 17 (1978), pp. 348-375.

12. Pereira F. (editor) - C-Prolog User's Manual - Version 1.4d.edai, SRI International, Menlo Park CA.

13. Tichy W. - Design and Implementation of a Revision Control System, Proceedings of the VI International Conference in Software Engineering, Tokio, (September 1982).