Една реализация на интерпретатор на STRIPS планове

Милко Маринов, Светлана Стефанова

An Implementation of STRIPS Plans Interpreter: *A long standing problem in the field of automated reasoning is that of designing systems which can describe a set of actions (plan) which can be expected to allow the system to reach a desired goal. Developing a plan is defined as finding a sequence of actions to accomplish a specified goal. A plan is an organised collection of operators. A commonly used terminology throughout the AI planning systems is that of STRIPS operators. This paper presents an object-oriented organisation of STRIPS plans interpreter. The major classes and the architecture used in the system are described in details.*

Key words: *Planning techniques; domain representation; STRIPS plan operator; object-oriented implementation.*

ВЪВЕДЕНИЕ

В термините на интелигентните системи планът се дефинира като последователност от действия, необходими за постигане на определена цел. Целта на процеса на планиране е да се подредят отделните действия, така че да се достигне до предварително дефинираното крайно състояние. При решаване на задачата за планиране трябва да се отчитат следните два проблема [2,4,5,6]:

- ⇒ отделните действия имат възможността да променят състоянието на обекта. Това означава, че системата за планиране трябва да дефинира точно, какво може и какво не може да се променя в състоянието на обекта;
- ⇒ търсенето на подходяща последователност от действия се определя от зависимостите, които съществуват между отделните части на плана.

Всяка система за планиране трябва да генерира план, който представлява едно от възможните решения на дефинирания проблем. Планът може да осигури решаване на даден проблем, ако той е приложим към началното състояние на проблема и ако след неговото изпълнение се постига крайната цел.

Целта на настоящата статия е да се представи една обектноориентирана реализация на известната система за планиране STRIPS [4], като се оценят нейните предимства и недостатъци.

ОБЩА СТРУКТУРА НА ОПЕРАТОРИТЕ И ПЛАНА В СИСТЕМАТА *STRIPS*

Моделът на обекта, чийто действия ще бъдат планирани в STRIPS [1,2,3,5], се представя чрез предикатна логика от първи ред. Генерираният план се представя като последователност от оператори. Всеки проблем се характеризира с описание на началното състояние и крайната цел. Описанието на началното състояние представя на системата за планиране текущото състояние на обекта. Дефинирането на целта представя състоянието, до което трябва да достигне решението на проблема, когато планът бъде изпълнен. Последователността от оператори характеризира *действия*. Операторите описват действията като предварителни условия и очаквани резултати. Всеки оператор характеризира един *клас* от възможни действия. Той съдържа множество от променливи, които могат да бъдат заменени с константи, за да се опишат специфичните действия на конкретния оператор.

Операторите в STRIPS се описват чрез наредената тройка ***(P,D,A)***, където

- ***P*** е израз, описващ предварителните условия (***preconditions)***. Дефинират се фактите, които трябва да са налице, за да бъде възможно прилагането на оператора.
- ***D*** е списък от изрази (***delete list***), които не могат постоянно да бъдат “истина” и следователно трябва да бъдат изтрити.
- ***A*** е списък от изрази ***(add list***), които се добавят към текущото състояние на

обекта, за да се отрази резултатът от изпълнението на съответното действие.

Тези три елемента описват всеки оператор в системата за планиране и те представят текущото състояние на обекта. Ако някакъв предикат не се съдържа в "add" или "delete" списъка на даден оператор, то той не оказва влияние приприлагане на оператора.

Всеки план $\alpha=(\alpha_1, \alpha_2, ..., \alpha_N)$ дефинира последователност от междинни състояния в модела на обекта $M_0, M_1, ... M_N$, където $\mathbf{M_0}$ е началното състояние и

$$\mathbf{M}_i = (\mathbf{M}_{i-1} \setminus \mathbf{D}_{\alpha i}) \cup \mathbf{A}_{\alpha i} \qquad (i = 1, ...N).$$

Планът α може да бъде приет от системата за планиране, ако

$$\mathbf{M}_{i-1} \vdash \mathbf{P}_{\alpha i} \qquad (i = 1, ...N).$$

Системата STRIPS изпълнява следната последователност от стъпки при изграждане на плана:

1. Избира се под-цел и се прави опит тя да се постигне. Ако това е възможно, прави се преход към ***стъпка 4***, в противен случай - към ***стъпка 2***.
2. Избира се действие, чийто списък "add" съдържа клаузи, които позволяват да се продължи незавършеното доказателство на под-целта от ***стъпка 1***. Това действие се избира чрез анализ на целите, като се намалява разликата между целта и незавършеното й доказателство.
3. Предварителните условия на избраното действие формират нова под-цел. Прави се преход към ***стъпка 1*** и се прави опит да се докаже новата под-цел.
4. Ако новата под-цел съответства на крайното състояние, то интерпретаторът завършва своята работа. В противен случай, действието се изпълнява (променя се текущото състояние) като се установяват неговите предварителни условия и се прави преход към ***стъпка 1***.

Изходът от STRIPS е списък от действия, чрез които ще се постигне крайната цел. Характерно за този начин на работа на системата за планиране е, че след формиране на под-целите те се вмъкват в стек (***стъпка 3***). При системите, които използват представяне на предметната област чрез списъци, каквато е и STRIPS [1,3] се приема, че началният модел на предметната област се променя чрез процедури за добавяне и изтриване от модела на предметната област. Операторите се избират въз основа на целите, които се появяват в списъка ***"add"*** – елементите, добавени в модела на предметната област.

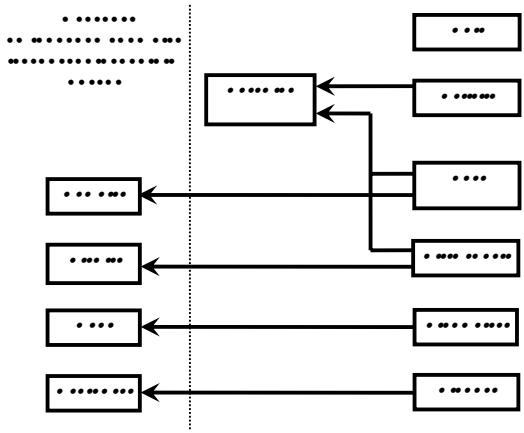
ОСНОВНИ КЛАСОВЕ, ИЗПОЛЗВАНИ ПРИ РЕАЛИЗАЦИЯ НА ИНТЕРПРЕТАТОРА

При реализация на интерпретатора е използвана библиотека, съдържаща основните алгоритми на търсене и механизма на логически извод. Йерархията на основните класове, използвани в интерпретатора е представена на фиг. 1.

Класовете ***PlanObject***, който наследява от ***Node***, и ***Planner***, произхождащ от ***DepthTree*** са основните класове в системата за планиране. Класът ***Planner*** осъществява техниките на търсене, а ***PlanObject*** представя обектите в пространството на търсене. Класът ***PlanObject*** съдържа конфигурацията на обекта, състоянието на стека, съдържащ междинната цел, и действието, което последно е било изпълнено върху обекта. Последователността от тези действия представя пътя на решение - плана.

В стека се включват обекти от различни типове, като за всеки един от тях са декларирани отделни класове: прости цели - ***Fact***, съставни цели - ***ListOfFacts*** и действия - ***Activity***. Класовете ***Fact*** и ***ListOfFacts*** се използват и при описание текущото състояние на обекта. Тъй като стекът трябва да съхранява обекти от различни класове, той е дефиниран като хетерогенен контейнерен клас. Декларираният контейнерен клас е списък, който съхранява указатели към обекти от

клас ***BasePlan***. Чрез обектите от клас ***Fact*** се дефинират предикати от вида on(a,b). Класът ***Fact*** произхожда от класа ***Complex***, който е дефиниран в библиотеката на логическите примитиви. Обектите от клас ***Fact*** са прости цели. Класът ***ListOfFacts*** дефинира сложни цели. Обектите от клас ***ListOfFacts*** са колекция от факти (например, *on(a,b)* & *on(b,c)*).



Фиг. 1 Йерархия на основните класове

Действията се дефинират чрез обекти от клас ***Activity***. Обект от клас ***Activity*** представя действието, което трябва да бъде изпълнено (например, pickup(a)). Информацията, съдържаща се в обект от клас ***Activity*** се съхранява в два различни обекта: обект от клас ***Complex***, който представя извършваното действие (например, pickup(a)), и указател към обект от клас ***Rule***.

Класът ***Rule*** представя правилата в системата за планиране. Чрез обектите от клас ***Rule*** се съхраняват името на правилото, списък на предварителните условия, списък "***add***" и списък "***delete***". Всеки един от тези списъци е обект от клас ***ListOfFacts***. Обектите от клас ***Rule*** се използват в комбинация с обектите от клас ***Actvity***. Всеки обект от клас ***Activity*** съдържа указател към обект от клас ***Rule***, така че е известно кое правило трябва да бъде приложено от системата за планиране.

Основната работа на интерпретатора се извършва от виртуалния метод ***processPlan()***, дефиниран в базовия клас ***BasePlan*** и всички класове, произхождащи от него. Тази функция извлича обект от клас ***PlanObject*** и генерира всички негови наследници. Методът

IntrList<Node> *PlanObject :: extend(int);

взема първия елемент от стека и извиква виртуалната функция ***processPlan()***, за да бъде обработен по съответен начин.

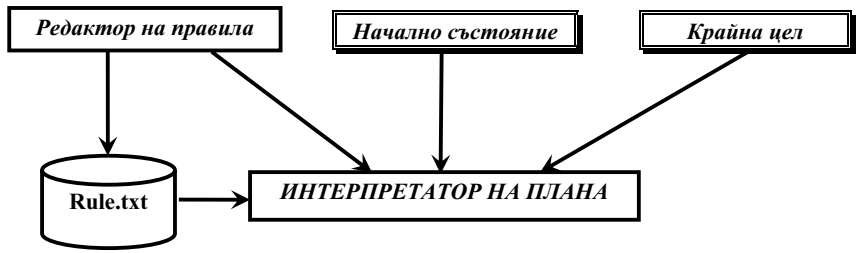
⇒ **Ако обектът е от клас *Fact* (проста цел)**. Проверява дали целта е постигната в текущото състояние на обекта. Установява се дали текущата цел съответства на състоянието или е необходима замяна. Ако елементът в текущата цел не се съдържа в крайното състояние, стекът се допълва с информацията, включена в замяната. Ако фактът не може да бъде обединен с никой от елементите на текущото състояние, откриват се действията, които ще удовлетворят този факт. Претърсват се правилата, за да се извлече необходимото действие. Ако текущата цел може да бъде обединена с някой от фактите в списъка "***add***", извлича се съответното действие. Прави се проверка дали действието вече е вмъкнато в стека. Това е необходимо, за да не се зацикли процеса на планиране.

Действието и неговите предварителни условия се добавят в стека. Предварителните условия се добавят като съставна цел (списък от факти).

⇒ **Ако обектът е от клас *ListOfFacts* (сложна цел)**. Съставната цел е постигната, ако всички нейни елементи са "истина". Ако в текущото състояние се открие под-цел, която не е "истина", тя отново се вмъква в стека. За да се определи дали съставната цел е постигната, всички под-цели трябва да бъдат обединени с фактите от текущото състояние. Всяка под-цел, за която не може да бъде намерено съответствие, се вмъква в стека. За непостигнатите под-цели се създава наследник и те се вмъкват в неговия стек. Ако всички елементи на съставната цел са постигнати, създава се нов наследник и неговият първи елемент се извлича, тъй като целта е удовлетворена.

⇒ **Ако обектът е от клас *Activity* (действие)**. Ако на върха на стека се намира обект от клас ***Activity***, то той е "истина" по дефиниция, тъй като всички негови предварителни условия вече са били извлечени. "Действието" трябва да бъде приложено към текущото състояние. Това означава, че се генерира наследник и фактите от списъка "**delete**" на ***Activity*** трябва да бъдат изтрити от състоянието на наследника, фактите от списъка "**add**" трябва да бъдат добавени. Обектът от клас ***Activity*** трябва да бъде съхранен заедно с родителския възел, тъй като той е част от плана. За да се определят действителните стойности на променливите в списъците "**add**" и "**delete**" трябва да се обедини обектът от клас ***Complex***, който е елемент на данните в класа ***Activity*** със съответния обект от класа ***Rule*** и резултатът да се приложи към елементите в списъците "**add**" и "**delete**". Трябва да се има предвид, че подреждането на елементите в полученото обединение е от съществено значение.

На фиг. 2 е представена обобщената архитектура в съчетание с технологията на използване на разработената система.



Фиг. 2 Обобщена архитектура

ПРИМЕР

При тестване на интерпретатора е използван известният класически пример [1,2], описващ преместването на блокове върху гладка плоскост от манипулатор на робот.

За да се опише състоянието на обекта са използвани следните предикати:

⇒ clear(***block***) - върху ***block*** няма нищо и той не е в манипулатора на робота;

⇒ holding(***block***) - ***block*** е в манипулатора;

⇒ arm(empty) - манипулаторът е свободен;

⇒ on(***block1, block2***) - ***block1*** се намира върху ***block2;***

⇒ ontable(***block***) - ***block*** е директно върху масата.

Използвани бяха следните правила:

⇒ *stack(x,y) -> **P:** clear(Y) & holding(X) -> **D:** clear(Y) & holding(X) -> **A:** arm(empty) & on(X,Y) & clear(X)*

⇒ *unstack(x,y)->**P:**on(X,Y)&clear(X)&arm(empty); **D:**on(X,Y)&clear(X)&arm(empty); A:holding(X)&clear(Y)*

⇒ *pickup(x)->**P:** ontable(X)&clear(X)&arm(empty); **D:**ontable(X)&clear(X)&arm(empty); **A:** holding(X)*

⇒ *putdown(x)->**P:** holding(X); **D:** holding(X); **A:** ontable(X) & arm(empty) & clear(X)*

1. Нека началното състояние на обекта да бъде описано по следния начин:
   clear(a) & clear(b) & clear(c) & ontable(a) & ontable(b) & ontable(c) & arm(empty)
   Нека крайната цел да бъде: *on(a,b) & on(c,a)*
   Планът за постигане на тази цел ще бъде:
   pickup(a) -> stack(a,b) -> pickup(c) -> stack(c,a)

2. Нека началното състояние на обекта да бъде описано по следния начин:
   clear(b) & clear(c) & ontable(a) & ontable(b) & on(c,a) & arm(empty)
   Нека крайната цел да бъде: *on(a,b) & on(b,c)*
   Планът за постигане на тази цел ще бъде:
   unstack(c,a)->putdown(c)->pickup(a)->stack(a,b)->unstack(a,b)->putdown(a)->
   pickup(b)->stack(b,c)->pickup(a)->stack(a,b)

ЗАКЛЮЧЕНИЕ

Предложената обектно-ориентирана реализация на система за планиране е осъществена на езика C++. Използвана е средата на Borland C++ Builder, чрез която е постигнат дружелюбен потребителски интерфейс. По този начин е осигурена възможност за интегриране процеса на планиране със симулиране работата на обекта и възможност за пряка намеса на потребителя в процеса на тестване и настройване на плана. Чрез редакторът на правила, който предполага и дефиниране на нови клаузи, се осигурява възможност за прилагане на системата в различни предметни области. Въпреки това е трудно да се опишат действията на по-сложни обекти само с предикатна логика от първи ред, което е един от основните недостатъци на STRIPS плановете.

ЛИТЕРАТУРА

[1] Aylett,R.S., G.J. Petley, P.W. Chung, B. Chen, D.W. Edwards, AI planning: solutions for real world problems, Knowledge-Based Systems, 2000, 13, 61–69.

[2] Cox,J.S., E. Durfee, An Efficient Algorithm for Multiagent Plan Coordination, AAMAS'05, 2005.

[3] Eiter,T., W. Faber, A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity, ACM Transac. on Computational Logic, 2004, 5(2), 206–263.

[4] Маринов М., Анализ на техниките на планиране, използвани в интелигентните системи, Автоматика и информатика, 1997, 5-6, 25-33.

[5] Marinov M., Using planning techniques in object-oriented design, in Proc. of the Int. Conference on Computer Systems and Technologies, Bulgaria, 2005, II.19.1-II.19.6.

[6] Tyugu, E., Understanding knowledge architectures, Knowledge-based systems, 2006,19, 50-56.

За контакти:

Доц. д-р Милко Маринов, Катедра "Компютърни системи и технологии", Русенски университет "Ангел Кънчев", Тел.: (082) 888 356, E-mail: MMarinov@ecs.ru.acad.bg

Гл.ас. д-р Светлана Стефанова, Катедра "Компютърни системи и технологии", Русенски университет "Ангел Кънчев", Тел.: (082) 888 356, E-mail: SStefanova@ecs.ru.acad.bg

Докладът е рецензиран.