Jacek Ambroziak
ORT, IPPT PAN

SYSTEMY TABLICOWE. IMPLEMENTACJA W JĘZYKU C

1. Motywacje

W opracowaniu opisuje się założenia projektowe, rozważania implementacyjne i uwagi o zastosowaniu programu stanowiącego narzędzie przeznaczone do budowy konkretnych zastosowań systemów z bazą wiedzy. Opisywany program jest niezależny od jakiejkolwiek dziedziny potencjalnego zastosowania. Po dołączeniu doń odpowiednio skonstruowanej bazy wiedzy (w formie zbioru tekstowego) program realizuje funkcje mechanizmu wnioskującego oraz wejścia/wyjścia systemu ekspertowego, który może być zastosowany zarówno w kontekście konsultacji, jak i w systemie wizji czy inteligentnego sterowania robotem itp. Ten typ programu określa się mianem skorupy systemu ekspertowego (expert system shell). Więcej informacji na temat systemów ekspertowych zawarto w osobnych materiałach [1,2].

Program definiuje pewien język formalny służący do zapisywania baz wiedzy. Język ten, jego składnia, semantyka i pragmatyka zostaną szczegółowo opisane.

Program został napisany w języku C dla komputerów IBM PC/XT/AT, może jednak być bez trudu przeniesiony na inne, zwłaszcza większe komputery. Zagadnienia związane z wyborem języka programowania stanowią treść oddzielnej części opracowania.

Zasadniczym źródłem motywacji do napisania tego programu była potrzeba posiadania go w celu testowania różnorodnych koncepcji przetwarzania wiedzy, jak również w celu zbudowania

kilku konkretnych zastosowań. Przy okazji pisanie i uruchamianie programu tego typu pozwala dowiedzieć się więcej o problemach napotykanych przy implementacji systemów przetwarzania danych symbolicznych oraz o ich rozwiązaniach. Jest więc w tym aspekcie czysto informatycznym studium wciąż mało popularnego lecz szybko zyskującego na znaczeniu obszaru wysoko wyspecjalizowanych zastosowań informatyki: technik programowania sztucznej inteligencji (dalej: AI - od Artificial Intelligence).

Systemy o analogicznej funkcji są już dostępne na świecie od dość dawna (wczesne lata 80-te) w postaci skomercjalizowanej. Znane są jako "expert system shells". Systemy te różnią się bardzo pomiędzy sobą w wielu aspektach poczynając od ceny a kończąc na programach szkoleniowych organizowanych przez producentów. Kilka spośród nich wyposażono we wszystkie powszechnie uznane techniki AI. Do systemów takich należą m.in. ART f-my Inference, KEE f-my IntelliCorp, S.1 f-my Teknowledge, LOOPS f-my Xerox oraz Knowledge Craft f-my Carnegie Group. Systemy te mają wbudowane mechanizmy strukturalizacji wiedzy (systemy wykorzystujące ramy z różnymi typami dziedziczenia własności), kilka mechanizmów wnioskowania oraz bogate graficznie środki komunikacji z użytkownikiem. Ważne jest także zapewnienie właściwego środowiska interakcyjnego, ułatwiającego tworzenie baz wiedzy poprzez narzucanie określonej spójności metodologicznej tworzonym w bazie elementom reprezentującym wiedzę o danej dziedzinie zastosowania. Systemy te nie są jednak dla nas praktycznie dostępne. Są bardzo rozbudowane i konstruowane są z myślą o komputerach o dużej mocy obliczeniowej klasy "workstation". Wszystkie wymienione systemy mają więc swoje implementacje na komputerach takich, jak SUN, Apollo, Xerox czy Symbolics.

Istnieją też oczywiście narzędzia opisywanego rodzaju napisane dla IBM PC. Mają one jednak daleko skromniejsze możliwości, nie wystarczające do budowy prawdziwie dużych zastosowań, tj. zastosowań wymagających reprezentowania dużej ilości faktów, reguł itp., a przede wszystkim wielu różnych rodzajów wiedzy. Kilka takich systemów - EXSYS, M.1 i FLOPS - pozostaje do naszej

dyspozycji; jeden z nich (EXSYS) posłużył nawet do zbudowania niewielkiego zastosowania [3]. Dwa pierwsze spośród nich nadają się głównie do nauki posługiwania się prostymi schematami regułowymi z wbudowanym wnioskowaniem wstecz. W tej sytuacji stworzenie własnego programu tego typu, prócz korzyści doraźnych i czysto badawczych, daje szansę posiadania narzędzia bogatszego (w techniki AI) od wielu, skądinąd trudno dla nas dostępnych produktów światowych. Co najwyżej nasz program nie dorówna rywalom w zakresie jakości komunikacji z użytkownikiem oraz dokumentacji. Podstawową jego przewagą będzie jednak otwartość, tj. możliwość ustawicznego rozwijania i doskonalenia. Ponadto, co stanowi jedno z głównych założeń, program może być przystosowywany do swoich zastosowań poprzez uzupełnianie go o potrzebne w danym zastosowaniu cechy i możliwości. W ten sposób stanie się punktem wyjścia dla, być może, całej rodziny bardziej wyspecjalizowanych programów. Próba zawarcia wszystkich identyfikowanych w trakcie dalszych prac cech w jednym systemie mogłaby prowadzić do jego zbytniego i zbędnego uogólnienia, a także do zwiększenia wymiarów, co jest niepożądane na IBM PC.

Należy podkreślić, że wersja programu istniejąca w czasie pisania tego opracowania nie jest jeszcze zamknięta, nawet ze względu na cechy potencjalnie potrzebne we wszystkich zastosowaniach, tj. w części niezależnej od dziedziny. Niemniej, już teraz można zademonstrować funkcjonowanie najistotniejszych elementów architektury programu, które prawdopodobnie pozostaną niezmienione także w następnych wersjach.

2. Podstawy architektoniczne

System zaprojektowany został zgodnie z koncepcją tzw. architektury tablicowej (blackboard), która w ostatnich latach zyskuje sobie rosnącą popularność w dziedzinie zastosowań AI. Jest to koncepcja najbardziej wyrafinowana, bardzo ogólna oraz otwarta na różnego rodzaju rozszerzenia. W szczególności jest bardzo obiecująca dla badaczy wieloprocesorowych systemów AI jak

i dla tych, których interesują systemy sprzężone, symboliczno - numeryczne.

Nazwa opisywanej architektury pochodzi od pojęcia tablicy: globalnej struktury danych, na której zapisywane są elementy przetwarzanego zagadnienia oraz inne informacje. Struktura ta odpowiada pojęciu pamięci krótkoterminowej systemu produkcji, a także pamięci roboczej czy też globalnej bazie danych systemu regułowego [5]. Różnica uzasadniająca odrębność nazwy pochodzi od sposobu, w jaki tablica jest wykorzystywana, choć i on przypomina zwykłe systemy regułowe.

Treść tablicy jest badana i modyfikowana przez aktywne elementy zwane modułami wiedzy (knowledge sources). Każdy moduł ma dostęp do całej tablicy, choć w praktyce interesuje go tylko niewielki jej fragment. System ekspertowy pisany jest tak, by moduły były wyspecjalizowane w określonych poddziedzinach całego rozwiązywanego zagadnienia, reagowały na modyfikacje tablicy dotyczące tych poddziedzin i w razie potrzeby dokonywały własnych zapisów, przyczyniając się do postępu w rozwiązywaniu postawionego przed systemem zagadnienia. Nazwa "tablica" odnosi się więc do globalnej bazy danych systemu ekspertowego, na której zapisów i innych modyfikacji dokonuje zespół kooperujących "specjalistów" - modułów wiedzy wyspecjalizowanych w rozwiązywaniu fragmentów postawionego przed systemem problemu.

2.1. Sterowanie

System tablicowy jest sterowany zdarzeniami (event driven). Każda modyfikacja tablicy jest zdarzeniem, które może zainteresować (pobudzić) nastawione na jego typ moduły wiedzy. Moduły te zawierają wiedzę proceduralną i gdy są wykonywane (interpretowane) w jakiś czas po pobudzeniu, mogą dokonywać własnych modyfikacji tablicy. Modyfikacje te stają się źródłem kolejnej generacji zdarzeń i cyki pracy systemu tablicowego zamyka się.

Najistotniejszą cechą systemu tablicowego jest zasada

rządząca wyborem kolejnych modułów wiedzy do wykonania. Aby moduł mógł zostać wykonany musi być, po pierwsze, pobudzony przez jakieś, interesujące go zdarzenie. Od momentu pobudzenia do interpretacji modułu może jednak upłynąć dowolnie długi czas (mierzony ilością cykli systemu), a nawet może w ogóle nie dojść do wykonania modułu.

Do stwierdzenia potencjalnej użyteczności modułu po zaistnieniu jakichś zdarzeń, służy związany z modułem warunekpobudzenia (trigger). Moduł zostaje pobudzony gdy warunek ten jest spełniony przez któreś ze zdarzeń. Do opisania zdarzenia pobudzenia tworzy się tzw. rekord aktywacji modułu, gdzie zapisuje się który moduł został pobudzony, przez jakie zdarzenie, w jakim kontekście, w którym cyklu pracy systemu, a także inne informacje istotne dla dalszego postępowania z tym zapisem.

Skoro część wykonawcza modułu może dokonywać operacji na danych z tablicy, w swym zapisie wykorzystuje zmienne (odpowiadające parametrom formalnym procedur klasycznych języków programowania). W rekordzie aktywacji zapisuje się wiązania części tych zmiennych, dokonane w czasie sprawdzania warunku pobudzenia. Naturalnie, jeden moduł wiedzy może być w danym cyklu kilkakrotnie pobudzany przez dowolne ze zdarzeń wygenerowanych w poprzednim cyklu pracy systemu, a jedno zdarzenie może pobudzić więcej niż jeden moduł. W tych wypadkach generowana jest odpowiednia liczba rekordów aktywacji rejestrujących wszystkie, różne od siebie wzajemnie zdarzenia pobudzenia. Z uwagi na to, że każdą zmianę zaszłą na tablicy traktuje się jako zdarzenie, rozmaitość zdarzeń może być większa od rozmaitości modułów i mogą się zdarzyć cykle bez żadnych nowych pobudzeń. Wygenerowane rekordy aktywacji dodaje się do listy pobudzeń (triggered list), która jest jedną z ważnych list systemowych utrzymywanych przez interpreter systemu tablicowego.

Po uzupełnieniu listy pobudzeń przegląda się jej elementy (rekordy aktywacji) i sprawdza, czy pobudzone moduły rzeczywiście mogą być wykonane. Pozwala się o tym przekonać kolejny po warunku pobudzenia, warunek wstępny modułu. O ile warunek pobudzenia nastawiony był na zdarzenia, których kolejne generacje powstają i

są zapominane w następujących po sobie cyklach, o tyle warunek wstępny bada statyczne aspekty sytuacji na tablicy, tj. układy danych, które mogą pozostawać nie zmieniane w ciągu dowolnej ilości cykli. Należy zwrócić szczególną uwagę na to, by warunek wstępny miał w ogóle szansę spełnienia. W przeciwnym razie układ obu warunków będzie funkcjonował jak pułapka powodując nieograniczony przyrost niepotrzebnych rekordów aktywacji. Może okazać się wskazane dodanie do istniejącego schematu kolejnego warunku określającego sytuacje, w których rekord aktywacji staje się niepotrzebny i może być usunięty.

W czasie sprawdzania warunku wstępnego mogą zostać związane wyznaczone zmienne modułu. Wiązania zawierające wskaźniki do danych spełniających warunek, zapisywane są we właśnie przetwarzanym rekordzie aktywacji. Jeżeli warunek wstępny został spełniony przez aktualną zawartość tablicy, moduł wiedzy może w zasadzie od razu być zastosowany. Zamiast tego, rekord aktywacji modułu uzupełniony o dane pochodzące z badania warunku wstępnego, przenoszony jest z listy pobudzeń na listę wykonania (invocable list), kolejną listę systemową właściwą architekturze tablicowej. Lista ta - z definicji - zawiera rekordy aktywacji odpowiadające modułom, które mogą być bezpośrednio wykonane.

W systemach AI listę o podobnym charakterze określa się często mianem agendy. Gdy lista ta staje się pusta, program musi zakończyć działanie. Jeśli zawiera pojedynczy element, musi on jako jedyny kandydat zostać zinterpretowany w następnym cyklu. Najczęściej lista wykonania zawiera większą ilość rekordów opisujących możliwe do podjęcia działania. Problem wyboru działania nazywa się problemem sterowania i w systemach tablicowych jego traktowanie jest szczególnie ważnym elementem.

Rozwiązywanie problemu sterowania jest szukaniem odpowiedzi na pytanie: jaką czynność wykonać jako następną? Jeżeli w ramach danego systemu i postawionego przed nim problemu można w ogóle osiągnąć zadany cel wykonując jakieś działania, to cała inteligencja systemu realizuje się w trafnych wyborach kolejnych działań. Działania tylko w szczególnych przypadkach są od siebie niezależne; ogólnie biorąc, poprzez modyfikację środowiska (tu:

tablicy) wpływają między innymi na to, jakie działania będą dopuszczalne w kolejnych cyklach. Nieprawidłowy wybór akcji może utrudnić albo wręcz uniemożliwić osiągnięcie celu.

System tablicowy przy okazji każdego przeglądu listy wykonania upewnia się, czy zgodnie z definicją listy każdy jej element opisuje wciąż jeszcze możliwe do wykonania działanie. Gdy rekordy aktywacji oczekują na liście wykonania, sytuacja na tablicy wciąż się zmienia i jest możliwe, by warunek wstępny modułu przestał być spełniony. Gdy taka sytuacja rzeczywiście ma miejsce, rekord aktywacji przenosi się z powrotem na listę pobudzeń, zapominając wiązania zmiennych warunku wstępnego. Obie listy zawierają więc te i tylko te rekordy, które opisują zdarzenia pobudzenia dotychczas nie sfinalizowane działaniem pobudzonego modułu. Na liście pobudzeń pozostają te spośród nich, których moduły jeszcze (lub już) się nie kwalifikują do wykonania, ze względu na niespełnienie warunku wstępnego.

Kontynuując porównanie z prostszymi systemami regułowymi, warto zauważyć, że wybór jednego działania spośród wielu możliwych do wykonania odpowiada rozwiązaniu konfliktu (conflict resolution), jednemu z trzech etapów cyklu pracy klasycznego systemu regułowego. Pomiędzy oboma architekturami zachodzą jednak w tym względzie istotne różnice. Po pierwsze, w systemie regułowym zbiór konfliktowy tworzony jest w każdym cyklu od nowa, podczas gdy lista wykonania jest co najwyżej stale modyfikowana.

Po drugie, strategie rozwiązywania konfliktu, mniej lub bardziej skomplikowane, są zazwyczaj zawarte w strukturze interpretera stanowiąc jego cechę charakterystyczną, niezależną od zastosowania. W systemie tablicowym zagadnienie wyboru modułu do wykonania, przynajmniej w zasadzie, podlega ogólnym metodom rozwiązywaniu problemów.

W inteligentnym zachowaniu się człowieka wielką rolę odgrywa umiejętność myślenia refleksyjnego, a więc nie tylko działanie lecz i myślenie o działaniu, a nawet o samym myśleniu. AI ma świadomość znaczenia tego faktu od początku swego istnienia, lecz dopiero w ostatniej dekadzie pojawiły się programy jawnie eksploatujące refleksję jako składnik inteligencji [5][6]. Warto

zauważyć, że podstawowy język AI, LISP, posiadając wspólną formę dla danych i programów (jak na niższym poziomie sam komputer von Neumanna) wnosi potencjał refleksyjności. Program lispowy może bezpośrednio odczytywać, modyfikować i wykonywać inny program napisany w tym języku. Jest to przy tym elementarna cecha LISPu, poznawana już przez początkujących.

System tablicowy może być wyposażony w cechy działania refleksyjnego bez specjalnych trudności technicznych. Zauważmy, że jego elementy "systemowe" jak listy pobudzeń i wykonania, baza faktów i baza modułów wiedzy mogą być traktowane jak wszystkie inne dane, z którymi system ma do czynienia. Dla LISPu jako języka implementacji jest to oczywiste; w przypadku użycia C, realizacja tej własności także nie nastręcza trudności o ile jest planowana od początku. Zachowanie się rekordów aktywacji może być obserwowalne dla systemu, o ile zmiany w obszarze list pobudzeń i wykonania będą generowały zdarzenia, jak wszystkie inne zmiany zachodzące na tablicy.

Narzucającym się rozwiązaniem na poziomie metodologicznym jest zaliczenie do struktur tablicy obu list i zbioru modułów wiedzy [6]. Problem sterowania - wyboru najlepszego następnego działania - zostaje ujawniony i rozwiązywany jest równolegle z zagadnieniem postawionym przed systemem. Barbara Hayes-Roth dzieli tablicę na dwie części: tablicę dziedziny (domain blackboard) i tablicę sterowania (control blackboard). Z każdą z tablic związany jest odpowiedni zbiór modułów wiedzy. Moduły dziedzinowe zawierają wiedzę specyficzną dla zastosowania systemu, zaś sterujące zajmują się koordynacją działania całego systemu. Ich funkcją jest szeregowanie bieżących czynności systemu - rozwiązywanie konfliktu. Mogą w realizacji tej funkcji wykorzystywać dowolnie złożone struktury danych, a sam proces rozwiązywania traktowany jest jako wymagający (sam w sobie) wiedzy i ogólnych metod rozwiązywania problemów.

3. Język implementacji

Jako język implementacji programu wybrano język C (Aztec C

3.2d). W uzasadnieniu tego wyboru język ten należy skontrastować z dwoma alternatywnymi kandydatami, Golden Common LISPem oraz Turbo Pascalem. Turbo Prolog nie był w ogóle rozważany ze względu na brak biegłości w posługiwaniu się tym językiem u autora opracowania.

LISP jest językiem szczególnie predestynowanym do roli języka implementacji systemów z bazą wiedzy:

1. interakcyjny tryb pracy z wykorzystaniem mniej lub bardziej rozbudowanego środowiska programistycznego pozwala szybko oceniać poprawność fragmentów programu bezpośrednio po edycji i w ten sposób przyczynia się do większej produktywności programisty, mierzonej tak tempem uruchamianego programu, jak i łatwością przeprowadzania eksperymentów z różnymi sposobami zapisania algorytmicznych koncepcji przetwarzania danych symbolicznych;
2. dynamiczne struktury danych stanowią podstawę filozofii języka i wraz z programami gospodarki pamięcią (włączając w to odzyskiwanie nieużytków) są gotowe do wykorzystania;
3. struktura języka zrównująca składnię i implementację danych i programów pozwala elegancko i naturalnie wprowadzać techniki programowania przydatne w złożonych systemach przetwarzania informacji symbolicznej, wykorzystujące możliwość przetwarzania danej struktury jako danej a następnie wykonania jej jako programu. W szczególności w regułach produkcji warunki i akcje reguł mają taką właśnie dwoistą naturę: z jednej strony stanowią dane, gdy reguły są gdzieś przechowywane, przeglądane, modyfikowane itp., a z drugiej, sprawdzeni warunku lub wykonanie akcji to wykonanie odpowiadających im struktur informacyjnych jako programów. W LISPie można to osiągnąć bezpośrednio, wykorzystując jako interpreter ten sam zasadniczy interpreter, który wykonuje wszystkie nie skompilowane programy LISPowe.

Niestety LISP uzyskuje pełne walory narzędziowe dopiero na komputerach o większej mocy obliczeniowej. W przypadku niewielkiego IBM PC już samo interakcyjne środowisko języka zajmuje znaczącą część zasobów i tym mniej pozostaje pamięci i cykli dla budowanych nad tym środowiskiem zastosowań. Gdyby więc udało się zbudować nad GC LISPem narzędzie wyższego poziomu, to z

kolei zupełnie już problematyczne stałoby się wykorzystanie go do nietrywialnych (czytaj: 'pamięciochłonnych') zastosowań.

Należy wspomnieć, że GC LISP doczekał się kolejnych wersji, w szczególności wyposażonych w kompilator (którego brak w wersji dostępnej nam), a także wykorzystujących dodatkową pamięć komputera AT. Istnieje też wersja produkowana łącznie ze specjalnym rozszerzeniem sprzętowym komputera IBM PC/AT t.zw. kartą Humminga, opartą na procesorze 80386 i o pamięci do 24 MB. Implementacji tych nie posiadamy.

Innym ważnym zagadnieniem jest uniwersalność oprogramowania i jego przenośność. Chodzi tu o możliwość tworzenia zastosowań dla współpracowników i odbiorców zewnętrznych. Zastosowanie GC LISPu prowadziłoby do konieczności łączenia w jeden pakiet napisanego w LISPie narzędzia wyższego poziomu, jego bazy wiedzy i samego systemu LISPu (ze skomplikowaną procedurą jego instalacji). Z analogicznych powodów światowi producenci narzędzi AI oryginalnie napisanych w LISPie, decydują się na przeprogramowywanie sprawdzonych już produktów na język C, tak by mogły być wykorzystywane na wielu komputerach z UNIXem, a nie tylko na maszynach lispowych Lambda czy Symbolics, co ze względu na koszt tych maszyn znacząco obniża wzięcie tych produktów. Podobny efekt miałby miejsce gdybyśmy wykorzystywali najnowsze wersje GC LISPu.

Opisane powyżej względy efektywności i uniwersalności wyłączają GC LISP z rozważań. W efekcie zadanie budowy narzędzia AI komplikuje się o konieczność ustosunkowania się do wymienionych powyżej trzech zalet LISPu, które w takim wypadku muszą być od początku zaprogramowane.

Do implementacji programu mógł jeszcze posłużyć Turbo Pascal, najpopularniejsza implementacja Pascala na IBM PC. Możliwość ta została także odrzucona ze względu na przewagi języka C w wielu wymiarach, w jakich oba języki można porównywać ze względu na rodzaj zadania. Porównania te zostaną przedstawione w kolejnych paragrafach.

Każda dobra realizacja języka C jest bliska standardowi języka, co prowadzi do dużego stopnia przenośności

oprogramowania. Opisywany program będzie więc można bez zasadniczych trudności uruchomić na innym komputerze gdyby zaszła taka potrzeba i była do tego okazja. Różne implementacje różnią się przede wszystkim zawartością bibliotek, przy przenoszeniu oprogramowania może więc co najwyżej zajść potrzeba uzupełnienia biblioteki w wersji docelowej. Ponadto, większość nowoczesnych komputerów posiada system operacyjny UNIX, w którego środowisku język C znajduje się zawsze jako część oprogramowania standardowego. Kompilator Pascala musi być kupowany oddzielnie.

Porównywane języki są do siebie bardzo podobne pod względem poziomu i elementów służących strukturalizacji programu (analogiczne konstrukcje iteracyjne, wyboru, blokowe itp.) lecz różnią się w swej ideologii.

Pascal ma opinię języka klarownego, dobrze spełniającego wymogi języka edukacji informatycznej a także środka komunikacji algorytmów w literaturze. I tu jednak istnieją ważne odstępstwa wśród fundamentalnych pozycji literatury informatycznej: książka [7] wykorzystuje własny pascalo-podobny język Pidgin ALGOL, [8] postępuje podobnie idąc w kierunku matematycznej czystości języka, zaś [9] również unikając idiosynkrazji dowolnego spośród istniejących języków, tworzy dla swoich dydaktycznych celów język maszynowy abstrakcyjnego komputera MIX. Rola Pascala w dzisiejszej informatyce nie jest więc tak wielka, jak można by sądzić z popularności jego nazwy, choć właśnie ta popularność i możliwość tworzenia zastosowań w dość szerokim zakresie dzięki udanej implementacji Turbo Pascala przyczynia się do dużej atrakcyjności języka.

Język C, z drugiej strony, został zaprojektowany jako wygodny język narzędziowy, służący do pisania zastosowań często określanych mianem systemowych, a więc takich jak procesory tekstów, kompilatory i inne programy użytkowe. Jest on mniej przejrzysty od Pascala (choć o wiele czytelniejszy od abstrakcyjnego języka z [9]), jednak wiele technik programowania, zarówno nawiązujących do sprzętu jak i ogólniejszych, daje się zapisać w sposób bardziej zwięzły i naturalny. W szczególności konstrukcje odnoszące się do bajtowej organizacji pamięci oraz do

pojedynczych bitów, które zapisuje się bezpośrednio w języku z [9], a w C za pomocą znanych idiomów, w Pascalu wymagają, po pierwsze, odejścia od standardu, który nie posiada praktycznie żadnego modelu maszyny, i po drugie programowania trikowego niezgodnego z duchem języka (np. konwersje typów danych przy użyciu rekordów z wariantami).

Uruchamianie programów napisanych w C jest dość pracochłonne, może być jednak dokonywane przyrostowo poprzez rozwijanie kolejnych modułów, zapisywanych w odzielnych zbiorach i oddzielnie kompilowanych. W Pascalu natomiast, trzeba zawsze kompilować całość programu i nie jest zupełnie oczywiste, w jaki sposób program taki mógłby korzystać z modułów napisanych w innych językach, np. w asemblerze (pomijając nieelegancką konstrukcję inline).

Pascal nadaje się do tworzenia zastosowań zamkniętych, ograniczających się do konkretnego algorytmu przetwarzania danych, nie wymagającego komunikacji ze światem zewnętrznym względem programu. C dopuszcza szeroki zakres tej komunikacji, pozwalającej odnosić się tak do cech sprzętu, jak i systemu operacyjnego (np. w UNIXie uruchamianie nowych procesów itp.).

Wszystkie te argumenty przemawiają jednogłośnie za wyborem języka C, jako języka implementacji dużego, o ambicjach uniwersalności narzędzia do budowy systemów z bazą wiedzy. W dodatku, wybór ten zgodny jest z aktualnym trendem światowym.

4. Język bazy wiedzy

Program skorupy systemu ekspertowego definiuje pewien język formalny służący do zapisywania baz wiedzy. Projekt języka stanowi zasadniczy składnik pracy. Opisując język należy podać trzy jego wymiary.

Składnia języka określa jakie napisy stanowią dobrze zbudowane elementy języka. Semantyka podaje znaczenie, tj. sposób interpretacji, poszczególnych, dobrze zbudowanych konstrukcji

języka. Wreszcie, pragmatyka dotyczy celów, do jakich konstrukcje języka mogą być wykorzystywane. Język tu opisywany będzie najprawdopodobniej rozszerzany i modyfikowany w kolejnych wersjach programu.

4.1. Ogólna budowa bazy wiedzy

Elementami wierzchniego poziomu bazy wiedzy są zapisy kolejnych modułów wiedzy, zapis strategii wyboru następnego działania ("policies"), zestaw działań inicjalizujących pracę systemu oraz zestaw działań zamykających pracę systemu. Dwa ostatnie elementy są opcjonalne. Po elementach wierzchniego poziomu następuje kończąca zapis bazy wiedzy kropka. Wymienione elementy są obiektami złożonymi. Ich elementami mogą być reguły produkcji lub tzw. akcje. Moduły wiedzy. Moduły wiedzy zapisywane są w następujący sposób:

knowledge_source < ... treść modułu ...>

W zapisie tym "knowledge_source" jest słowem kluczowym języka i jak wszystkie pozostałe słowa kluczowe musi być podawane we wskazanej postaci.

Treść modułu, zawarta w nawiasach klamrowych, składa się z kilku elementów. Elementy te muszą występować w stałej kolejności. Każdy z nich zapowiadany jest odpowiednim słowem kluczowym, po którym następuje znak dwukropka i dalej sama treść składnika modułu.

name: <identyfikator>

Składnik ten nadaje modułowi wiedzy nazwę wybraną przez użytkownika. Może być pominięty i wtedy moduł uzyska nazwę postaci "Kn_Src<numer>", wygenerowaną przez system. Nazwy modułów muszą być różne.

problem_domain: <identyfikator>

Podawana jest tu identyfikacja tablicy, na której moduł wiedzy ma działać. Jeżeli moduł zajmuje się szeregowaniem

rekordów aktywacji, odpowiednim dla niego identyfikatorem jest "control". Element ten wykorzystywany jest przy wyborze modułu do wykonania.

description: "<ciąg znaków>"

Zwięzły opis modułu przeznaczony dla osób czytających tekst bazy wiedzy. Ciąg znaków nie może zawierać znaku cudzysłowu.

trigger: <ciąg <akcji>>

Warunek pobudzenia. Składa się z dowolnej ilości następujących po sobie akcji (opisanych poniżej) mających znaczenie predykatów. Spełnienie ich wszystkich dla określonego zdarzenia powoduje utworzenie odpowiedniego rekordu aktywacji.

precondition: <ciąg <akcji>>

Warunek wstępny. Funkcjonuje jak warunek pobudzenia. Gdy zostanie pominięty, traktowany jest jak tożsamościowo spełniony.

action_level: <identyfikator>

Nazwa warstwy (obszaru tablicy), dla której moduł został napisany. Element ten odgrywa rolę przy szeregowaniu zadań poprzez mechanizm skupiania uwagi (focus control).

ks_efficiency: <liczba zmiennoprzecinkowa>
ks_credibility: <liczba zmiennoprzecinkowa>
ks_importance: <liczba zmiennoprzecinkowa>

Liczbowe określenia - odpowiednio - efektywności modułu, wiarygodności zawartej w nim wiedzy oraz ważności modułu w procesie rozwiązywania problemu. Liczby te dobierane są subiektywnie przez osoby piszące bazę wiedzy i powinny być wybrane dla wszystkich modułów według jednej skali. Są wykorzystywane przez mechanizm szeregujący tak, że szansa wybrania modułu do wykonania jest rosnącą funkcją każdej z tych składowych.

Podobnych charakterystyk liczbowych mogłoby być więcej. Do konkretnego zastosowania należy odpowiednie wykorzystanie tych wartości. Podstawowym motywem wykorzystania tej trójwymiarowej charakterystyki użyteczności modułu jest możliwość dynamicznych,

tj. zachodzących w czasie działania systemu, modyfikacji aktualnej polityki wyboru modułów do wykonania. Dla przykładu, w systemie czasu rzeczywistego w sytuacji konieczności szybkiego rozwiązania problemu, wskazane wydaje się oparcie na modułach efektywnych, raczej niż wiarygodnych lecz drogich w sensie czasu wykonania. Wielowymiarowe charakterystyki liczbowe modułów pozwalają na stosowanie różnych kryteriów ich oceny. Liczby te mogłyby też być modyfikowane przez sam system w procesie uczenia się, gdyby system mógł sam ocenić wiarygodność, efektywność i ważność modułu.

code: <ciąg <akcji>>

Gdy moduł jest wykonywany, akcje tu wymienione (jeśli jakieś podano) wykonywane są bezwarunkowo, przed wykonaniem jakiejkolwiek z następujących reguł. Procedura działania modułu może być całkowicie zawarta w tym składniku. W przeciwnym razie odgrywa on rolę akcji wstępnej modułu.

rules: <ciąg <reguł>>

Ostatni ze składników modułu. Po dwukropku wymienia się kolejno wszystkie reguły produkcji modułu. Jeśli podano jakieś reguły, wykonywane one są (po akcji wstępnej) zgodnie ze strategią wnioskowania w przód.

4.2. Strategia szeregowania zadań

Tym mianem określa się zestaw kryteriów, które łącznie pozwalają posortować rekordy aktywacji i znaleźć najwyżej oceniany, który będzie wykonany w następnym cyklu. Dla ścisłości warto dodać, że wyrafinowanie systemów tablicowych może sięgać jeszcze dalej i nie jest konieczne by do wykonania kierować moduł najwyżej ocenionego rekordu aktywacji. Do ostatecznego wyboru można wykorzystywać schemat losowy, który daje większe szanse wyżej ocenionym rekordom.

W opisywanym systemie na strategię szeregowania składa się zestaw jawnie wymienionych w bazie wiedzy reguł produkcji, o

budowie takiej jak reguły modułów. Reguły te wykonuje się wszystkie po kolei dla ocenianego rekordu aktywacji. Reguły te odczytują między innymi subiektywne, liczbowe charakterystyki modułów, mogą faworyzować moduły sterujące przed dziedzinowymi, oraz ostatnio (w sensie cykli systemu) pobudzone przed pobudzonymi wcześniej i dotąd nie wykonanymi.

Odpowiedni element bazy wiedzy ma postać:

```
policies (

   rules:

      (ciąg (reguł))

)
```

Ujawnienie opisywanych reguł szeregowania zgodne jest z ogólną ideologią systemu tablicowego lecz zmniejsza efektywność systemu. Wydaje się, że w czasie przechodzenia zastosowania systemu tablicowego z fazy eksperymentalnej do fazy eksploatacji, ustalona już strategia szeregowania zadań powinna być zrealizowana przez sam interpreter, tj. na poziomie języka implementacji, nie zaś na poziomie języka bazy wiedzy.

4.3. Inicjacja tablicy

```
blackboard (

   (ciąg (akcji))

)
```

Akcje tu wymienione wykonywane są na samym początku pracy systemu, po inicjalizacji interpretera i wczytaniu bazy wiedzy. Ich efektem są pierwsze zdarzenia, które wywołują całe dalsze działanie systemu. Akcje te mogą zapisać na tablicy wstępne dane.

4.4. Epilog

```
epilog (
        <ciąg <akcji>>
)
```

Wymieniona konstrukcja zawiera akcje, które przeznaczone są do wykonania przy końcu działania systemu, po wyczerpaniu listy modułów, które możnaby uruchomić.

4.5. Reguły produkcji

Każda spośród używanych w systemie reguł, w modułach wiedzy lub poza nimi, ma następującą budowę:

```
 rule (

  <ciąg <akcji>>

==>

  <ciąg <akcji>>

)
```

Pierwsza grupa akcji, przed symbolem "==>", stanowi warunek reguły. Akcje traktowane są jako predykaty, a ich sekwencja jak koniunkcja predykatów. Spełnienie tej koniunkcji dla danych bezpośrednich lub będących aktualnymi wartościami zmiennych użytych w warunku, pociąga za sobą wykonanie drugiej grupy akcji, stanowiącej 'akcję' reguły. Pominięcie warunku oznacza jego tożsamościowe spełnienie, tj. bezwarunkowe wykonanie akcji przetwarzanej reguły.

4.6. Akcje

Akcje stanowią podstawowy budulec bazy wiedzy. Odpowiadają instrukcjom i wyrażeniom konwencjonalnych języków programowania i są bezpośrednio odpowiedzialne za modyfikacje tablicy oraz operacje wejścia/wyjścia. Zapis każdej akcji zaczyna się otwierającym nawiasem kwadratowym i kończy zamykającym nawiasem kwadratowym. Na akcję składają się trzy elementy, z których ostatni jest opcjonalny.

Pierwszym składnikiem akcji jest operator. Operator to symbol (najczęściej identyfikator), z którym związany jest rodzaj akcji. Po operatorze następuje zapis obiektu, którego dotyczy akcja. Ostatnią część stanowią, opcjonalnie, kolejne argumenty operatora akcji.

Akcje wykorzystywane są także jako elementarne warunki w regułach i modułach wiedzy. Na każdą akcję można patrzyć jako na wyrażenie, które po wykonaniu zwraca jakąś wartość. Wartość 'Nil' (jak w LISPie) została zarezerwowana dla logicznego fałszu. Gdy akcje połączone są w sekwencję koniunkcji, dla spełnienia warunku każda elementarna akcja musi zwrócić wartość rożną od 'Nil'. Gdy którakolwiek akcja zwróci wartość pustą, kolejne akcje nie są już interpretowane, a warunek nie jest spełniony.

4.6.1. Akcje elementarne

[+= <obiekt> <wyrażenie>]

<obiekt> zwiększa się o zmiennoprzecinkową wartość wyrażenia.

[print <obiekt>]

Wartość <obiektu> zostaje wyświetlona użytkownikowi w formacie odpowiednim do typu obiektu.

[set <obiekt> <wyrażenie>]

Obiekt podstawia się wartością wyrażenia. Jest to podstawowa akcja systemu.

[extend <obiekt>]

Tworzy i zwraca jako wartość nowy obiekt o typie swojego argumentu ze skopiowanym slotem 'Object'.

[assert <fakt>]

Wprowadza obiekt typu 'fakt' do tablicy wywołując proces aktualizacji przekonań. Uzasadnieniem wprowadzonego faktu stają się reguła zawierająca akcję 'assert' oraz fakt, który spełnił warunek tej reguły.

[read <obiekt>]

Czyta ze standardowego wejścia liczbę, symbol lub listę liczb/symboli/list i podstawia uzyskaną wartością <obiekt>.

[new <typ obiektu>]

Kreuje i zwraca nowy obiekt podanego typu, np. Fact, Event.

[fact <obiekt>]
[event <obiekt>]

Predykaty sprawdzające, czy podany obiekt jest typu, odpowiednio, faktu lub zdarzenia. Niespełnienie predykatu sygnalizowane jest wartością 'Nil'.

[known <obiekt>]
[unknown <obiekt>]

Predykat sprawdzający czy obiekt posiada wartość, tj. czy wartość ta jest znana, czy też nie była dotąd podstawiana, oraz predykat przeciwny.

[halt]

Akcja wstrzymująca pracę interpretera.

[* <obiekt> <obiekt>]
[/ <obiekt> <obiekt>]
[+ <obiekt> <obiekt>]
[- <obiekt> <obiekt>]

Operatory arytmetyczne. Zakłada się, że <obiekty>

reprezentują liczby zmiennoprzecinkowe. Akcje zwracają wynik wskazanego działania arytmetycznego dla podanych liczb.

4.7. Inne zagadnienia leksykalne

Baza wiedzy jest zapisywana w formacie swobodnym i, analogicznie jak to ma miejsce dla konwencjonalnych języków programowania, jak Pascal czy C, można i należy stosować znaki białe dla uzyskania formy graficznej zapisu ułatwiającej czytanie treści bazy. Wprowadzono pojedynczy znak komentarza: ~ (tylda). Wszystkie znaki występujące po tyldzie do końca linii wyłączone są z normalnej interpretacji. Za tyldą można zatem wpisywać komentarze i tworzyć wizualne ograniczniki poszczególnych elementów bazy wiedzy. Wśród znaków zawartych pomiędzy tyldą a końcem linii mogą wystąpić specjalne sekwencje stanowiące informację sterującą dla interpretera. W chwili obecnej określone są dwie takie sekwencje: L- oraz L+. Sterują one wyświetlaniem tekstu bazy wiedzy na ekranie monitora w czasie jej wczytywania. Domyślnie tekst jest wyświetlany. Włączanie i wyłączanie może być przydatne podczas uruchamiania fragmentu bazy wiedzy tak, by tylko on był wyświetlany przy kolejnych uruchomieniach systemu. Tekst bazy wiedzy należy zakończyć znakiem kropki.

LITERATURA

1. J. Ambroziak, *Systemy ekspertowe*, cz.1, Informatyka, 11-12, 1986.
2. J. Ambroziak, *Systemy ekspertowe*, cz.2, Informatyka, 1, 1987.
3. W. Sosnowski, J. Ambroziak, M. Kleiber, *EXASP - system ekspertowy do analizy płyt usztywnionych*, VIII Konferencja "Metody komputerowe w mechanice konstrukcji", Warszawa, 1987.
4. N. J. Nilsson, *Principles of Artificial Intelligence*, Springer-Verlag, Berlin, Heidelberg, New York, 1982.
5. D. Lenat, M. Prakash, M. Shepherd, *CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition*

Bottlenecks, Artificial Intelligence Magazine, 1986.

6. B. Hayes-Roth, *Blackboard architecture for control*, Journal of Artificial Intelligence 26, North-Holland, 1985, 251-321.
7. A. V. Aho, J. E. Hopcroft, J. D. Ullman, *Projektowanie i analiza algorytmów komputerowych*, Warszawa, PWN 1983.
8. E. W. Dijkstra, *Umiejętność programowania*, Warszawa, WNT 1985.
9. D. E. Knuth, *The Art of Computer Programming: Fundamental algorithms*, Addison-Wesley, 1973.