Rozdział 6

Wnioskowanie w sieci stwierdzeń zorganizowanej jako sieć bayesowska

Sebastian RZYDZIK

6.1. Wstęp

Rozdział ten kierowany jest do autorów systemu DiaDyn zajmujących się przede wszystkim rozwojem i opracowywaniem fragmentów systemu związanych z metodami wnioskowania w sieciach stwierdzeń.

Moduł Dia_Bel jest częścią systemu DiaDyn [6.16, 6.4] i został opracowany na potrzeby realizacji procesu wnioskowania przybliżonego. Cały moduł został napisany w języku Java [6.20]. Przewiduje się, że moduł będzie rozwijany w kierunku zastosowania innych metod wnioskowania. Z tego powodu postanowiono użyć systemu wieloagentowego. Założono, że każdy rodzaj zadania, będzie obsługiwany przez ściśle wyspecjalizowanego agenta. Takie podejście pozwala na elastyczną rozbudowę systemu.

6.2. Wybór środowiska

Wybór narzędzi był podyktowany następującymi potrzebami:

- niezależność systemowa i sprzętowa,
- dostępność bibliotek programisty (API),
- stabilność działania,
- dostępność dokumentacji programisty.

Z szerokiego grona narzędzi do opracowywania i rozwijania systemów wieloagentowych, takich jak: FIPA-OS [6.21], ZEUS [6.7], BOND [6.12], czy Grasshopper [6.10], szczególną uwagę zwrócono na środowisko o nazwie JADE [6.11] (ang. Java Agent DEvelopment Framework). Jak można wnioskować z nazwy, platforma JADE została napisana w języku programowania Java [6.20]. Dzięki temu możliwe jest uruchomienie środowiska agentowego na każdym urządzeniu komputerowym, na które napisano wirtualną maszynę Java (ang. Java Virtual Machine). Najważniejszymi cechami środowiska agentowego JADE są:

- tworzenie, klonowanie i usuwanie agentów,
- swobodna migracja agentów w ramach granic środowiska,
- wbudowana usługa katalogowa przechowująca dane o uruchomionych agentach (funkcja udostępniana przez specjalnego agenta o nazwie *DF* (ang. Directory Facilitator)),

- zastosowanie standardu *ACL* wymiany komunikatów pomiędzy agentami,
- graficzny interfejs użytkownika (udostępniany przez specjalnego agenta o nazwie *RMA* (ang. Remote Management Agent)),
- wbudowane narzędzia do analizy działania agentów (Debugger) i analizy komunikatów przesyłanych pomiędzy agentami (Sniffer).

Cała platforma jest zarządzana przez specjalnego agenta o nazwie *ams* (ang. Agent Management Service), który jest rdzeniem systemu. Agent ten jest odpowiedzialny za obsługę serwisów uruchamianych w ramach działań podejmowanych przez agentów.

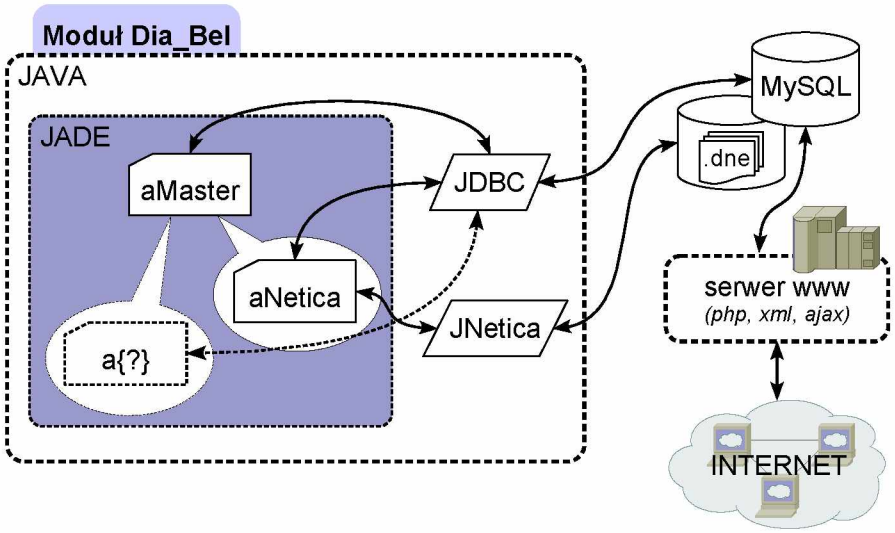
Wybór systemu agentowego podyktował jednocześnie wybór Javy jako języka programowania do tworzenia modułu Dia_Bel.

Jako narzędzie wspomagające realizację zadania wnioskowania w sieci bayesowskiej wybrano oprogramowanie Netica [6.14]. Pomimo dostępności oprogramowania dystrybuowanego na licencji GNU GPL (np. Bayesian Network tools in Java (BNJ) [6.1]) postanowiono użyć sprawdzonego i stabilnego programu jakim jest Netica, a w szczególności udostępnionej biblioteki JNetica.

6.3. Ogólna struktura modułu Dia_Bel

Na rysunku 6.1 pokazano strukturę modułu Dia_Bel. Moduł Dia_Bel jest powiązany z systemem DiaDyn za pomocą bazy danych MySQL [6.13] (z użyciem sterownika JDBC). Użytkownik nie ma bezpośredniego dostępu do tego modułu. Zlecanie zadań jest możliwe tylko z poziomu modułu Dia_Sta współpracującego z modułem Dia_Wiki [6.17, 6.6], który służy, m.in. do gromadzenia stwierdzeń i reguł. Z kolei moduły Dia_Sta i Dia_Wiki są dostępne poprzez moduł Dia_Sys [6.18] (rozdz. 3), który zarządza uprawnieniami użytkowników do wybranych elementów systemu DiaDyn. Wszystkie elementy systemu DiaDyn są zainstalowane na serwerze stron internetowych.

Jednym z ważniejszych elementów modułu Dia_Bel jest system wieloagentowy. Wybrana platforma JADE jest platformą w pełni zgodną ze specyfikacją opracowaną przez fundację FIPA (ang. The Foundation for Intelligent Physical Agents) [6.9], której celem jest opracowywanie standardów dla technologii agentowych. Od 2005 roku fundacja FIPA działa w ramach komitetu IEEE [6.19].

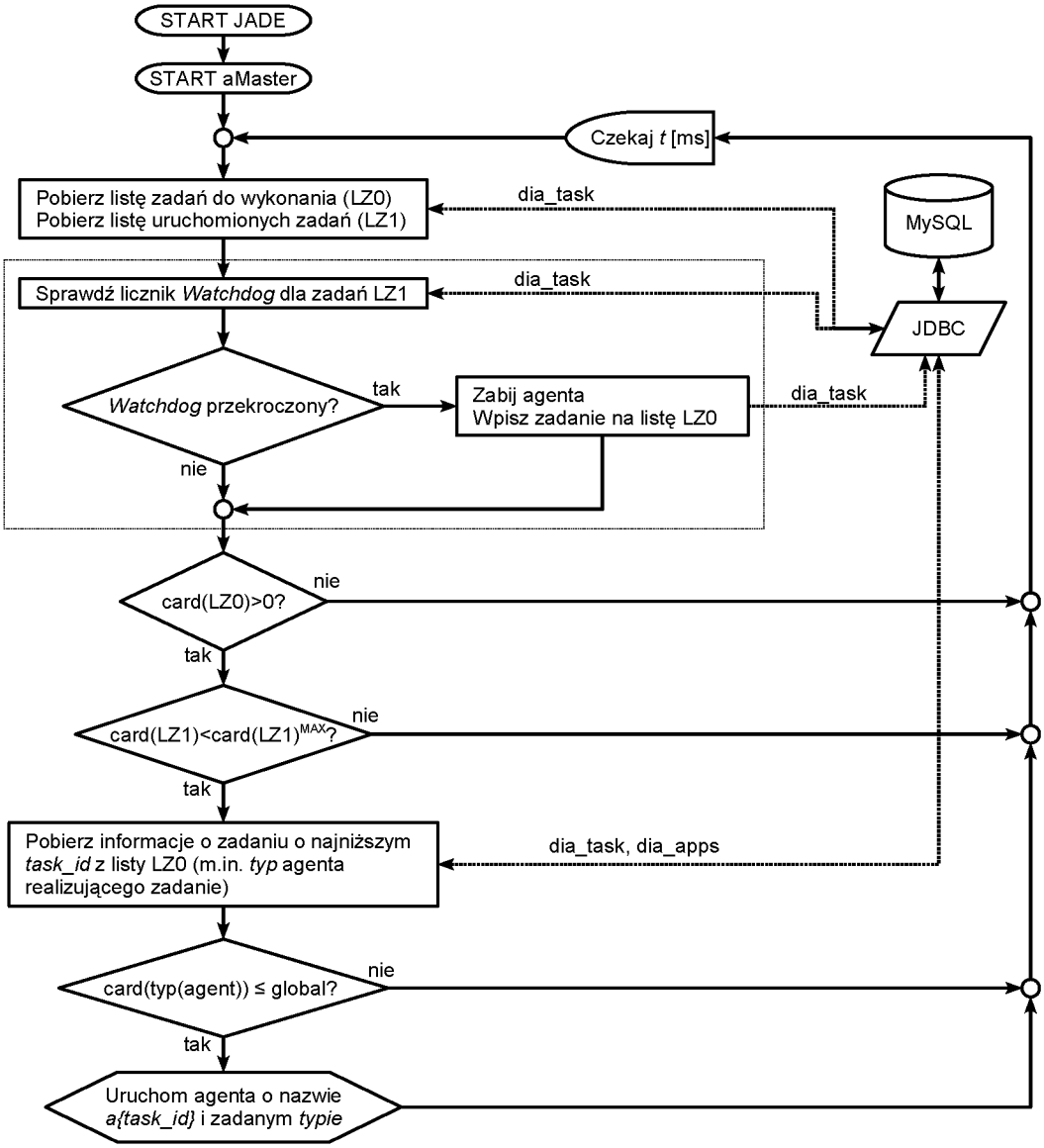


Rys. 6.1: Struktura modułu Dia_Bel

Zarządzaniem żądaniami realizacji zadań wnioskowania zajmuje się agent typu aMaster. Na podstawie danych odczytywanych z bazy danych agent aMaster tworzy agentów typu $a\{\star\}$ (gdzie $\{\star\}$ jest nazwą typu) wyspecjalizowanych do realizacji konkretnych zadań. Na dzień publikacji tych materiałów dostępny jest tylko jeden typ takiego agenta - aNetica, który jest przeznaczony do realizacji procesu wnioskowania z użyciem sieci bayesowskich. Jest to możliwe dzięki użyciu biblioteki JNetica firmy Norsys. Biblioteka ta ma dostęp do zapisanych w bazie danych zbiorów plików .dne zawierających struktury sieci przekonań. Format .dne jest formatem zapisu struktury i parametrów sieci bayesowskich stosowanym przez firmę Norsys.

6.4. Agent typu aMaster

Na rysunku 6.2 pokazano algorytm działania agenta typu aMaster.



Rys. 6.2: Algorytm działania agenta typu aMaster

6.4.1. Uruchomienie agenta

Uruchomienie agenta musi być poprzedzone uruchomieniem systemu JADE. Agent działa cyklicznie z minimalnym czasem oczekiwania równym $t[ms]$ ustalanym jako parametr. Na początku każdego cyklu pobierane są listy zadań oczekujących na wykonanie ($LZ0$) i zadań aktualnie wykonywanych ($LZ1$). Na podstawie listy $LZ1$ pobierane są wartości liczników Watchdog każdego zadania. Jeżeli dany licznik ma wartość większą od założonej wartości czasu bezczynności agenta realizującego zadany proces, to ten agent jest zabijany, a zadanie jest ponownie wpisane na listę $LZ0$. Następnie agent sprawdza czy są jakieś zadania do wykonania ($card(LZ0) > 0$). Jeżeli są żądania wykonania zadań, to agent sprawdza czy liczba uruchomionych agentów (realizowanych zadań) nie przekracza maksymalnej liczby zadań jednocześnie wykonywanych. W przypadku, gdy liczba zadań aktualnie wykonywanych pozwala na uruchomienie jeszcze jednego zadania, to agent aMaster pobiera z bazy danych szczegółowe informacje na temat zadania. Następnie na ich podstawie sprawdza czy dany agent może współdziałać z innymi agentami. Jeżeli jest to możliwe, to tworzony jest nowy agent danego typu i nazwie o postaci *aTask_id*.

6.4.2. Parametry zadania

Parametry poszczególnych zadań są zapisywane w bazie danych. Do najważniejszych parametrów, niezbędnych do rozpoczęcia realizacji zadania, należą:

- `status` - status zadania: 0 - zadanie oczekujące na wykonanie, 1 - zadanie w trakcie realizacji, 2 - zadanie wykonane pomyślnie, 3 - zadanie zakończone błędem;
- `apps_id` - identyfikator aplikacji/agenta przeznaczonego do realizacji zadania (np. agent aNetica);
- `rule_id` - identyfikator struktury i parametrów sieci przekonań (np. plik .dne);
- `start_val_id` - identyfikator początkowych wartości stwierdzeń;
- `val_id` - identyfikator nieznanych lub aktualizowanych wartości stwierdzeń.

Pozostałe parametry są parametrami mających charakter informacyjny:

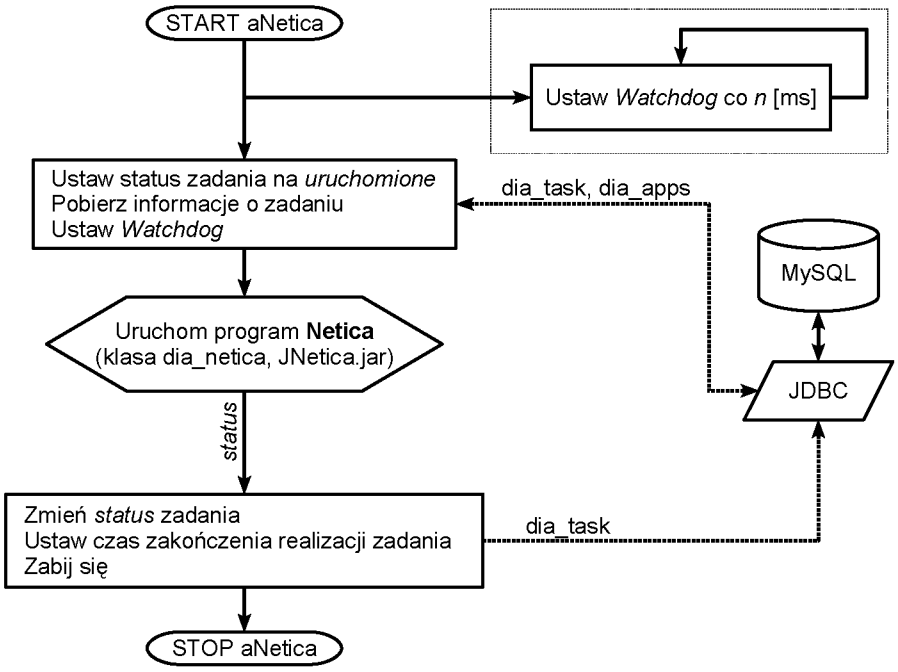
- `start_time` - znacznik czasu rozpoczęcia wykonywania zadania;
- `exec_time` - znacznik czasu aktualizowany w trakcie realizacji zadania (Watchdog);
- `end_time` - znacznik czasu zakończenia wykonywania zadania;
- `error` - opis błędu w przypadku niepomyślnego wykonania zadania (`status`=3).

6.5. Agent typu aNetica

Rysunek 6.3 przedstawia algorytm działania agenta typu aNetica. Agent aNetica realizuje zadania w dwóch wątkach. Pierwszy wątek jest odpowiedzialny za cykliczne zerowanie licznika Watchdog. Domyślnie licznik Watchdog jest zerowany co $10[s]$. Drugi wątek jest odpowiedzialny za właściwą realizację zadania.

W pierwszym kroku drugiego wątku jest zmieniany status zadania na *uruchomione* (`status`=1). Następnie, na podstawie parametrów wejściowych, agent aNetica pobiera szczegółowe informacje dotyczące realizowanego zadania, tworzy obiekt klasy *dia_netica* (zob. następny punkt) i uruchamia proces wnioskowania przybliżonego z użyciem biblioteki JNetica.

Jeżeli wartość zwrócona przez metodę obiektu klasy *dia_netica* jest równa 2, to agent aNetica zgłasza prawidłowe przeprowadzenie procesu wnioskowania. W przypadku, gdy otrzymana wartość jest równa 3, to agent zgłasza błąd. W obu przypadkach ustawiany jest czas zakończenia realizacji zadania (parametr `end_time`). Ostatecznie agent zabija się używając metody `zabijAgenta()` klasy *dia_netica*.



Rys. 6.3: Algorytm działania agenta typu aNetica

6.5.1. Realizacja procesu wnioskowania

Proces wnioskowania realizowany jest przez klasę *dia_netica* (rys. 6.4). Zadanie rozpoczyna się od odczytania parametrów uruchomieniowych, za pomocą których pobierane są nazwy stwierdzeń i ich wartości, oraz identyfikatora pliku .dne zapisanego w bazie danych. Następnie uruchamiany jest proces wnioskowania z użyciem wskazanej sieci przekonań. Po zakończeniu obliczeń, wynik zapisywany jest w bazie danych, a metoda zwraca wartość statusu, która jest odczytywana i interpretowana przez agenta typu aNetica.

6.5.2. Stosowanie sieci bayesowskich

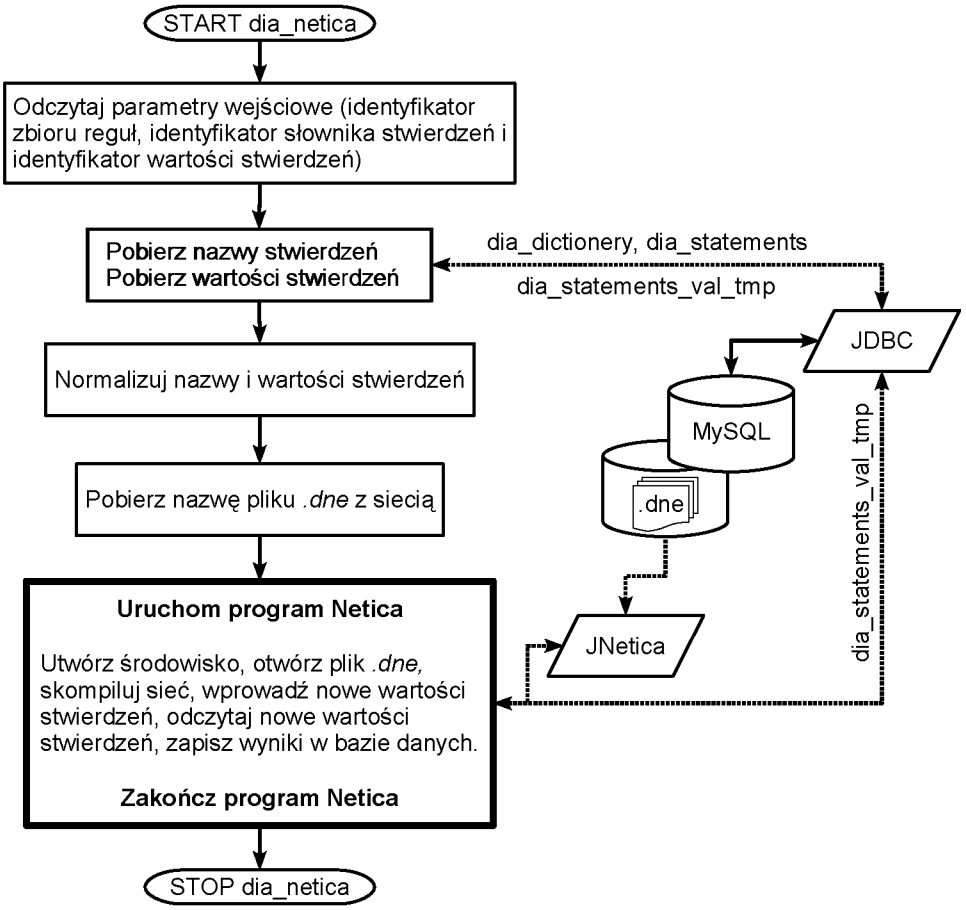
Dokładny opis budowy i uruchamiania sieci bayesowskich można znaleźć na przykład w dokumentacji dołączonej do oprogramowania Netica [6.14] lub materiałach seminaryjnych pierwszych warsztatów DiaDyn [6.3] oraz w innych publikacjach związanych z sieciami bayesowskimi: [6.8], [6.2], [6.5], [6.15].

6.6. Tworzenie własnych agentów

Przyjęto, że agent, to utworzony według przyjętego schematu program wyspecjalizowany do realizacji pewnych zadań z użyciem informacji zapisanych w bazach wiedzy i bazach danych systemu DiaDyn.

Zakłada się, że osoba chcąca utworzyć agenta realizującego określone zadania potrafi programować programy w języku Java oraz zaznajomiła się z instrukcją obsługi platformy agentowej JADE i z dokumentacją biblioteki JNetica.

Tworzenie własnego agenta należy rozpocząć od utworzenia klasy agenta. Klasa agenta określa jego typ, a dokładniej mówiąc typ zadania. W drugim kroku należy utworzyć klasę, która przeprowadza właściwy proces realizacji zadania. Opisany sposób jest tylko przykładem. Można również całą procedurę realizacji zadania zamieścić w klasie samego agenta.



Rys. 6.4: Schemat realizacji zadania z użyciem biblioteki JNetica (klasa *dia_netica*)

Na potrzeby modułu Dia_Bel napisano następujące klasy:

- `aMaster` - agent typu aMaster,
- `aNetica` - agent typu aNetica,
- `dia_czas` - zawiera metody wspomagające przetwarzanie czasu i daty z punktu widzenia systemu DiaDyn,
- `dia_jmysql` - zawiera metody wspomagające współdziałanie z bazą danych MySQL,
- `dia_netica` - klasa jest przeznaczona do wspomagania realizacji zadań wnioskowania z użyciem sieci przekonań.

Więcej informacji na temat tych klas można znaleźć w dokumentacji, dostarczonej w formacie javadoc, dołączonej do modułu.

6.6.1. Struktura przykładowego agenta

W dalszej części pokazano przykład klasy agenta aTest (wydruk 6.1). Komentarze ważniejszych fragmentów zawarto w kodzie.

Wydruk 6.1: Kod przykładowego agenta typu aTest

```
1 import jade.core.Agent;
2 import jade.wrapper.AgentContainer;
```

```
3  import jade.wrapper.AgentController;
4  import jade.core.behaviours.*;
5
6  public class aTest extends Agent {
7  private static final long serialVersionUID = 1L;
8  static String dbname = null;
9  static String dbuname = null;
10 static String dbpass = null;
11
12 //czas zerowania licznika Watchdog w [ms]
13 static long tickTIME = 10000;
14 //ustaw strefe czasu na GTM
15 dia_czas dc = new dia_czas();
16
17 protected void setup() {
18     //odczytanie parametrow bazy danych (dbname, user, pass)
19     //pierwsze trzy parametry nie moga byc zmieniane!
20     Object[] args = getArguments();
21     aTest.dbname = (String) args[0];
22     aTest.dbuname = (String) args[1];
23     aTest.dbpass = (String) args[2];
24
25     System.out.println("Dia_bel-aTest␣" +
26                         getAID().getName() +
27                         "␣jest␣uruchomiony.");
28
29     //tworz uruchamiany cyklicznie watek
30     //zerowania licznika Watchdog
31     addBehaviour(new TickerBehaviour(this, tickTIME) {
32         private static final long serialVersionUID = 1L;
33
34         //ustaw wartosc licznika Watchdog
35         protected void onTick() {
36             //w tab dia_task uaktualnic czas exec_time
37             ustawCzas(
38     getAID().getName().substring(1,getAID().getName().indexOf("@")),
39     "exec_time","");
40         }
41     });
42
43     //metoda aStart uruchamia watek realizujacy zlecone zadanie
44     aStart(getAID().getName());
45 }
46
47
48 //metoda rozpoczynajaca proces realizacji zadania
49 public void aStart (String aName){
50     dia_jmysql jmysql = new dia_jmysql(); //obiekt bazy mysql
51     //pobierz informacje o zadaniu
52     //task_id=aName.substring(1,aName.indexOf("@"))
53     String[] tabZad =
54         this.pobierzZad(aName.substring(1,aName.indexOf("@")));
55
56     //ustaw status zadania na 1
57     this.ustawStatus(tabZad[0], "1");
58     //w tab dia_task uaktualnic czas exec_time
59     this.ustawCzas(tabZad[0], "exec_time","");
60
61     //pobierz informacje o aplikacji (agencie)
62     String[] zadApps =
63         this.zadApps(Integer.valueOf(tabZad[2].trim()));
64
65     //[START] wykonaj zadanie
```

```
66      String[] args = {aTest.dbname, aTest.dbuname, aTest.dbpass};
67      //utworz obiekt reprezentujacy agenta o typie dia_zad;
68      //klasa dia_zad zostala przedstawiona w nastepnym kodzie zrodlowym
69      dia_zad diaZad = new dia_zad(args);
70      //uruchom zadanie metoda start (klasy dia_zad) i odczytaj status
71      int status = diaZad.start(Integer.valueOf(tabZad[9].trim()),
72              Integer.valueOf(tabZad[6].trim()),
73              Integer.valueOf(tabZad[7].trim()),
74              Integer.valueOf(tabZad[8].trim()));
75      //[STOP] zakoncz zadanie
76
77      //zakonczenie zadania: zmiana statusu na 2 (jezeli nie ma bledow),
78      //dodanie czasu zakonczenia obliczen w polu end_time,
79      //i zabicie agenta
80      this.ustawStatus(tabZad[0], String.valueOf("2"));
81      this.ustawCzas(
82          getAID().getName().substring(1,getAID().getName().indexOf("@")),
83          "end_time","");
84      zabijAgenta(aName.substring(1,aName.indexOf("@")));
85  }
86
87
88  //metoda pobiera informacje o zadaniach
89  public String[] pobierzZad (String task_id){
90      dia_jmysql jmysql = new dia_jmysql(); //obiekt bazy mysql
91      String[][] tabZad = null; //tablica z zadaniami do wykonania
92
93      try {
94          //[START] odczytaj informacje o zadaniach do wykonania
95          String[] kolumny = {"task_id", "status", "apps_id",
96                  "start_time", "exec_time", "end_time",
97                  "dict_id", "start_val_id", "val_id","rule_id"};
98          tabZad = jmysql.czytaj_tab(
99                  jmysql.mysql_start(dbname,dbuname,dbpass),
100                 "dia_task",kolumny,"task_id="+task_id.trim(),"");
101
102         return tabZad[0];
103         //[END] obsluga bazy danych
104         }
105     catch (Exception e) {
106         e.printStackTrace();
107         return null;
108         }
109 }
110
111
112 //metoda pobiera informacje o aplikacji realizujacej zadanie
113 public String[] zadApps (int apps_id){
114     dia_jmysql jmysql = new dia_jmysql(); //obiekt bazy mysql
115     String[][] tabApps_tmp = null; //tablica z zadaniami do wykonania
116     String[] tabApps = null; //tablica z zadaniami do wykonania
117
118     try {
119         //[START] odczytaj informacje o zadaniach do wykonania
120         String[] kolumny = {"name", "global_m", "local_m", "run"};
121         tabApps_tmp = jmysql.czytaj_tab(
122                 jmysql.mysql_start(dbname,dbuname,dbpass),
123                 "dia_apps",kolumny,"apps_id=" + apps_id,"");
124
125         tabApps = tabApps_tmp[0];
126
127         return tabApps;
128         //[END] obsluga bazy danych
```

```
129         }
130     catch (Exception e) {
131         e.printStackTrace();
132         return null;
133         }
134 }
135
136
137 //metoda wspomaga zapisywanie i uaktualnianie znacznikow czasu
138 //w polach: 'start_time', 'exec_time', 'end_time'
139 public boolean ustawCzas (String task_id, String czas, String wartosc){
140     dia_jmysql jmysql = new dia_jmysql(); //obiekt bazy mysql
141
142     try{
143         if (wartosc==""){
144             jmysql.dodaj_wiersz_sql(
145                 jmysql.mysql_start(dbname,dbuname,dbpass),
146                 "UPDATE dia_task SET " + czas + "=" +
147                 dc.teraz_dia_epoka() +
148                 " WHERE task_id=" + task_id);
149         }
150         else {
151             jmysql.dodaj_wiersz_sql(
152                 jmysql.mysql_start(dbname,dbuname,dbpass),
153                 "UPDATE dia_task SET " + czas + "=" + wartosc +
154                 " WHERE task_id=" + task_id);
155         }
156
157         return true;
158         }
159     catch (Exception e) {
160         e.printStackTrace();
161
162         return false;
163         }
164 }
165
166
167 //metoda wspomaga zapisywanie statusu zadania w polu 'status'
168 public boolean ustawStatus (String task_id, String status){
169     dia_jmysql jmysql = new dia_jmysql(); //obiekt bazy mysql
170
171     try{
172         jmysql.dodaj_wiersz_sql(
173             jmysql.mysql_start(dbname,dbuname,dbpass),
174             "UPDATE dia_task SET status=" + status +
175             " WHERE task_id=" + task_id);
176
177         return true;
178         }
179     catch (Exception e) {
180         e.printStackTrace();
181
182         return false;
183         }
184 }
185
186
187 //metoda wspomaga proces zabijania agenta
188 protected void zabijAgenta (String task_id){
189     dia_jmysql jmysql = new dia_jmysql(); //obiekt bazy mysql
190     //Object [] args = new Object[1];
191
```

```
192     //AID agentAID = new AID( "a"+task_id , AID.ISLOCALNAME );
193     AgentContainer c = getContainerController();
194     try {
195         AgentController a = c.getAgent("a"+task_id);
196         a.kill();
197         //wpis do dziennika zdarzen
198         jmysql.wstaw_log("Agent dla zadania o id=" + task_id +
199                 " zostal zabity", "dia_alog", dbname,dbuname,dbpass);
200     }
201     catch (Exception e){
202         e.printStackTrace();
203         jmysql.wstaw_log("Nie moge zabic agenta dla zadania o id=" +
204                 task_id , "dia_alog", dbname,dbuname,dbpass);
205     }
206 }
```

Przykład klasy dia_zad, która zawiera algorytm realizacji zadania, pokazano na wydruku 6.2. Realizacja zadania rozpoczyna się od uruchomienia metody `start()`.

Wydruk 6.2: Przykład algorytmu przeznaczonego do realizacji zadania wnioskowania w sieciach bayesowskich

```
public class dia_zad {
  2 static String dbname = null;
  3 static String dbuname = null;
  4 static String dbpass = null;
  5
  6 //konstruktor
  7 //podane argumenty sa przykladowe
  8 public dia_zad (String[] args) {
  9     dia_zad.dbname = args[0];
 10     dia_zad.dbuname = args[1];
 11     dia_zad.dbpass = args[2];
 12 }
 13
 14
 15 //metoda rozpoczyna realizacje zadania
 16 public int start(int rule_id, int dict_id, int start_val_id, int val_id){
 17     try {
 18         //utworz obiekt jmysql klasy dia_jmysql
 19         dia_jmysql jmysql = new dia_jmysql();
 20
 21         //odczytaj nazwy stwierdzen z bazy danych z tabeli
 22         //dia_statements i zapisz w tablicy nStw
 23         String[] nStw = null;
 24         String[] kolumny = {"file_id"};
 25         try {
 26             nStw = jmysql.array2lista(jmysql.czytaj_tab(
 27                     jmysql.mysql_start(dbname,dbuname,dbpass),
 28                     "dia_statements,dia_dictionery",kolumny,
 29                     "dict_id=" + dict_id +
 30                     " AND stmt_id=dia_statements.id",
 31                     "dia_statements.id"));
 32
 33             }
 34         catch (Exception e) {
 35             e.printStackTrace();
 36             return 3; // blad w zapytaniu nr 1
 37             }
```

```
38
39
40          //odczytaj wartosci stwierdzen poczatkowych
41          //z bazy danych z tabeli dia_statements_val_tmp
42          //start_val_id jest parametrem dostarczanym z zewnatrz
43          String[] wStwBD = null;
44          kolumny[0] = "value";
45          try {
46              wStwBD = jmysql.array2lista(jmysql.czytaj_tab(
47                      jmysql.mysql_start(dbname,dbuname,dbpass),
48                      "dia_statements_val_tmp",kolumny,
49                      "val_id="+start_val_id,"val_id"));
50
51              }
52          catch (Exception e) {
53              e.printStackTrace();
54              return 4; // blad w zapytaniu nr 2
55              }
56
57
58          //normalizacja nazw stwierdzen nStw
59          for (int i=0;i<nStw.length;i++){
60              nStw[i]=nStw[i].split("\\.")[1].trim();
61              //System.out.println (nStw[i]);
62          }
63
64
65          //normalizacja wartosci stwierdzen wStw
66          String[] wStw = new String[nStw.length];
67          String[] wStwNot = new String[nStw.length];
68          if(wStwBD.length==1){
69              int i=0;
70              StringTokenizer st =
71                      new StringTokenizer(wStwBD[0].trim(),",");
72               while (st.hasMoreTokens()) {
73                   wStw[i] = st.nextToken();
74                   wStwNot[i] = st.nextToken();
75                   i++;
76               }
77          }
78          else{
79              return 5; // blad w normalizacji danych wejsciowych
80              }
81
82          //uruchom srodowisko Netica ...
83          //(patrz dokumentacja API JNetica)
84
85          //... wczytaj siec przekonan z bazy danych (plik .dne) ...
86
87          //... wykonaj proces wnioskowania (kompilacja sieci) ...
88
89          //... odczytaj wartosci wskazanych stwierdzen ...
90
91          //... i uaktualnij pola w tabeli dia_statements_val_tmp
92
93          try {
94              jmysql.dodaj_wiersz_sql(
95                  jmysql.mysql_start(dbname,dbuname,dbpass),
96                  "UPDATE dia_statements_val_tmp SET value='" +
97                  wStwNewBD + "' WHERE val_id=" + val_id);
98              }
99          catch (Exception e) {
100             e.printStackTrace();
```

```
101                 return 6; // blad w zapytaniu nr 3
102                 }
103
104             return 1; //wartosc 1 - zadanie zakonczone poprawnie
105             }
106         catch (Exception e) {
107             e.printStackTrace();
108             return 2; //blad uruchomienia srodowiska Netica
109             }
110         }
111 }
```

6.6.2. Uruchomienie modułu Dia_Bel

Po utworzeniu własnego agenta (p. 6.6) i pomyślnym skompilowaniu jego kodu, można przejść do procesu uruchamiania modułu Dia_Bel. W pierwszym kroku należy uruchomić platformę agentową poleceniem:

```
java jade.Boot aM:aMaster(dbname dbuser dbpass
        cycleTime maxAgents maxNoAgentExecuteTime)
```

gdzie:

`jade.Boot` - polecenie uruchamiające platformę agentową JADE;

`aM:aMaster` - kolejno nazwa agenta i typ agenta;

`dbname` - nazwa bazy danych;

`dbuser` - nazwa użytkownika z uprawnieniami do wskazanej bazy danych;

`dbpass` - hasło użytkownika;

`cycleTime` - czas (w $[ms]$) pojedynczego cyklu pracy agenta aMaster;

`maxAgents` - maksymalna liczba jednocześnie uruchomionych agentów;

`maxNoAgentExecuteTime` - maksymalny czas (w $[ms]$) wykonywania pojedynczego zadania.

Należy pamiętać o wskazaniu położenia bibliotek `jade*.lib`, `mysql*.lib` oraz `NeticaJ*.lib`, np. używając zmiennej systemowej `CLASSPATH`:

```
CLASSPATH=%CLASSPATH%;.;C:\eclipse\lib\jade\lib\jade.jar;C:\eclipse\lib
\jade\lib\jadeTools.jar;C:\eclipse\lib\jade\lib\http.jar;C:\eclipse\lib
\jade\lib\iiop.jar;C:\eclipse\lib\NeticaJ.jar;C:\eclipse\lib\mysql-conn
ector-java-5.0.5-bin.jar;C:\eclipse\lib\
```

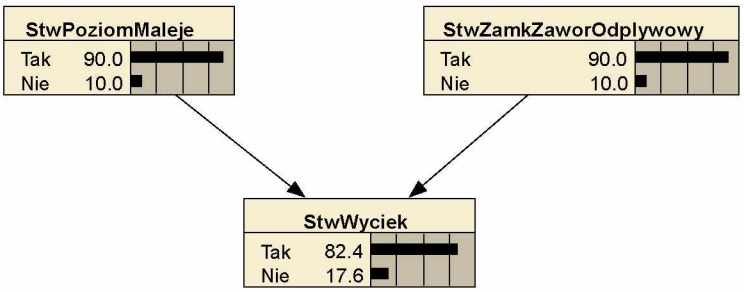
Na rysunku 6.5 pokazano informacje generowane przez platformę JADE w trakcie jej uruchamiania. Ostatnie wiersze informują o wykonaniu czterech cykli pracy platformy agentowej. W nawiasach kwadratowych przekazywane są kolejno informacje o liczbie zadań do wykonania oraz liczbie zadań aktualnie wykonywanych. Należy dodać, że ze względów ograniczeń biblioteki JNetica, można uruchomić tylko jedną kopię agenta typu aNetica i w związku z tym, zadania związane z wnioskowaniem w sieciach bayesa nie mogą być wykonywane równolegle.

6.6.3. Testowanie modułu Dia_Bel

Do testów utworzono prostą sieć bayesa zbudowaną z trzech stwierdzeń. Postać sieci pokazano na rysunku 6.6. W tabeli 6.1 pokazano wartości prawdopodobieństw warunkowych stwierdzenia `StwWyciek`.

```
C:\dia_bel>echo off
2007-10-18 11:38:45 jade.core.Runtime beginContainer
INFO: ----------------------------------
    This is JADE3.4 - revision 5874 of 2006/03/09 14:13:11
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.tilab.com/
----------------------------------------
2007-10-16 20:53:18 jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
2007-10-16 20:53:18 jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
2007-10-16 20:53:18 jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
2007-10-16 20:53:18 jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
2007-10-16 20:53:19 jade.core.AgentContainerImpl joinPlatform
INFO: --------------------------------------
Agent container Main-Container@JADE-IMTP://neo is ready.
--------------------------------------------
Dia_bel-aMaster aM@neo:1099/JADE jest uruchomiony.
0 [0], [0] Tue Oct 18 11:38:48 GMT 2007
1 [0], [0] Tue Oct 18 11:38:50 GMT 2007
2 [0], [0] Tue Oct 18 11:38:52 GMT 2007
3 [0], [0] Tue Oct 18 11:38:53 GMT 2007
```

Rys. 6.5: Moduł Dia_Bel w trakcie procesu uruchamiania



Rys. 6.6: Testowana sieć stwierdzeń

Wykaz wartości stopni prawdziwości stwierdzeń wejściowych przed procesem wnioskowania pokazano na rysunku 6.7. Dwa stwierdzenia, `StwPoziomMaleje` i `StwZamkZaworOdplywowy`, mają

Tab. 6.1: Tablica wartości prawdopodobieństw warunkowych dla węzła `StwWyciek`

StwPoziomMaleje	StwZamkZaworOdplywowy	Tak	Nie
Tak	Tak	100	0
Tak	Nie	10	90
Nie	Tak	0	100
Nie	Nie	50	50

Edycja wartosci stwierdzeń dla zadania - **Zadanie**

Nazwa zbioru wartości stwierdzeń:

dane testowe

Nazwa stwierdzenia	Wartość stwierdzenia
Maleje poziom wody	0.7
Zamkniety zawor odplywowy	0.8
Stwierdzono wyciek	N

Rys. 6.7: Wartości stopni prawdziwości stwierdzeń (moduł Dia_Sta)

Dia_Sta

Wyniki dla zadania **Zadanie**

Maleje poziom wody: 0.95
Zamkniety zawor odplywowy: 0.97
Stwierdzono wyciek: 0.93

Rozwiń wszystkie Zwiń wszystkie

Pobierz jako XML Zachowaj jako listę wartości stwierdzeń

Rys. 6.8: Wartości stwierdzeń otrzymane w module Dia_Sta

ustalone wartości stopni prawdziwości, natomiast stwierdzenie `StwWyciek` jest oznaczone jako stwierdzenie, którego wartość będziemy poszukiwać.

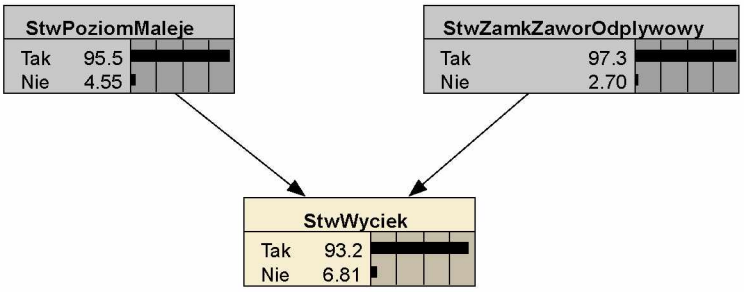
Na rysunku 6.9 pokazano komunikaty zwracane przez moduł Dia_Bel w trakcie wykonywania zadania. Pokazywane są podstawowe parametry sieci (liczba stwierdzeń i ich nazwy) oraz informacje o poszczególnych krokach realizacji zadania.

Wykaz otrzymanych wartości stwierdzeń prezentowanych w module Dia_Sta pokazano na rysunku 6.8. Dla porównania, na rysunku 6.10, pokazano wartości stwierdzeń uzyskane w programie Netica.

Więcej o procesie tworzenia zadań, edycji ich parametrów oraz ich uruchamiania opisano w rozdziale dotyczącym modułu Dia_Sta.

```
4 [0], [0] Thu Oct 18 11:38:55 GMT 2007
5 [0], [0] Thu Oct 18 11:38:56 GMT 2007
6 [1], [0] Thu Oct 18 11:38:58 GMT 2007
Dia_bel-aNetica a5@neo:1099/JADE jest uruchomiony.
Siec: wyciek
Wezly (3): [StwPoziomMaleje, StwZamkZaworOdplywowy, StwWyciek]
skompiluj siec

zakoncz Netica

strem.finalize

net.finalize

Netica zakonczona

status=ok

zakonczono zadanie

status zadania=2

zabito Agenta

7 [0], [0] Thu Oct 18 11:38:59 GMT 2007
8 [0], [0] Thu Oct 18 11:39:00 GMT 2007
```

Rys. 6.9: Moduł Dia_Bel w trakcie wykonywania zadania



Rys. 6.10: Wartości stwierdzeń otrzymane w programie Netica

Bibliografia

[6.1] Bayesian Network tools in Java (BNJ). Home page. http://www.kddresearch.org/Groups/Probabilistic-Reasoning/BNJ, 2007.

[6.2] Bednarski M. *Metody doskonalenia sieci bayesowskich stosowanych w diagnostycznych systemach doradczych*. Zeszyt 131. Katedra Podstaw Konstrukcji Maszyn Politechniki Śląskiej, Gliwice, 2006.

[6.3] Cholewa A. Przykład wnioskowania na podstawie sieci przekonań. Cholewa W., redaktor, *Warsztaty DIADYN. Materiały seminaryjne. wydanie II, poprawione i uzupełnione*, strony 22–29. Katedra PKM Politechniki Śląskiej, Ustroń, 2006-09-23.

[6.4] Cholewa W. Ogólna koncepcja systemu DIADYN. Cholewa W., redaktor, *Warsztaty DIADYN. Materiały seminaryjne. wydanie II, poprawione i uzupełnione*, strony 4–14. Katedra PKM Politechniki Śląskiej, Ustroń, 2006-09-23.

[6.5] Chow C. K., Liu C. N. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[6.6] Chrzanowski P. Przykład zastosowania modułu DIA_WIKI. Cholewa W., redaktor, *Warsztaty DIADYN. Materiały seminaryjne. wydanie II, poprawione i uzupełnione*, strony 40–43. Katedra PKM Politechniki Śląskiej, Ustroń, 2006-09-23.

[6.7] Collis J., Lee L., Thompson S. Zeus Agent Toolkit. http://labs.bt.com/projects/agents/zeus/, maj 2006.

[6.8] Cooper G. F., Herskovits E. A bayesian method for the introduction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, Boston 1992.

[6.9] FIPA - The Foundation for Intelligent Physical Agents. Home page. http://www.fipa.org, maj 2006.

[6.10] IKV++ GmBH, Informations- und Kommunikationssysteme. Grasshopper Programmer's Guide. http://www.grasshopper.de, maj 2006.

[6.11] Lab Telecom Italia. JADE - Java Agent DEvelopment Framework. http://jade.tilab.com, maj 2006.

[6.12] Marinescu D. C., Bölöni L. L. BOND. A Multi-Agent System. http://bond.cs.ucf.edu, maj 2006.

[6.13] MySQL AB. MySQL Community Server. http://www.mysql.com, marzec 2007.

[6.14] Norsys Software Corp. Netica™ Application. http://norsys.com/, marzec 2007.

[6.15] Pearl J. *Probabilistic reasoning in intelligent systems*. Morgan Kaufman, San Mateo, CA 1988.

[6.16] Projekt PBZ-KBN-105/T10/2003 koordynowany przez IMP PAN. System DiaDyn. https://kpkm.polsl.pl/diadyn/, październik 2007.

[6.17] Psiuk K. Ogólne zasady stosowania modułu DIA_WIKI. Cholewa W., redaktor, *Warsztaty DIADYN. Materiały seminaryjne. wydanie II, poprawione i uzupełnione*, strony 30–39. Katedra PKM Politechniki Śląskiej, Ustroń, 2006-09-23.

[6.18] Rzydzik S. Ogólne zasady stosowania modułu DIA_SYS. Cholewa W., redaktor, *Warsztaty DIADYN. Materiały seminaryjne. wydanie II, poprawione i uzupełnione*, strony 44–51. Katedra PKM Politechniki Śląskiej, Ustroń, 2006-09-23.

[6.19] Society IEEE Computer. Home page. http://www.computer.org/portal/site/ieeecs, maj 2006.

[6.20] Sun Microsystems, Inc. Java Platform, Standard Edition (Java SE). http://java.sun.com/javase/downloads/index.html, maj 2006.

[6.21] Treadway A., Duncan A., Newland Ch., Buckle P. FIPA-OS Agent Toolkit. http://sourceforge.net/projects/fipa-os/, maj 2006.