Inteligentní agenti

11. 5. 2011

Co je to inteligentní informační agent?

Inteligentní informační agent je autonomní a adaptibilní počítačový program, který operuje ve stejném programovém prostředí jako například operační systémy nebo databáze. Technologie těchto agentů kombinuje umělou inteligenci (uvažování, plánování, práce s přirozeným jazykem atd.) s technikami vývoje systémů (objektově zaměřené plánování apod.). Typické úlohy, které může agent provádět, jsou například filtrování elektronické pošty, organizování schůzek, lokalizování požadovaných informací, upozorňování na vhodnou možnost investice nebo zjištění nejvhodnějšího dopravního spojení.

Základní charakteristika inteligentních informačních agentů

Autonomní působnost = schopnost provádět uživatelem definované úlohy nezávisle na uživateli a často i bez přítomnosti nebo vedení uživatele. Uživatel jednou specifikuje co, kde a kdy má agent vykonávat, a ten provádí daný úkol pouze tehdy, když nastanou vhodné podmínky.

Přizpůsobivé chování = schopnost napodobovat jednotlivé uživatelovy kroky během provádění úlohy. Například si agent může uložit do paměti různé reakce uživatele na dané situace a podle toho potom sám provádět rozhodnutí apod.

Mobilnost = schopnost volně procházet počítačovými sítěmi a vykonávat úlohy na vzdálených místech. Agenti jsou většinou tvořeni přeložitelným scriptem, který napomáhá bezproblémovému pohybu přes různé architektury sítí. Například komunikačně orientovaný script jako Telescript může ulehčit vzájemnou komunikaci mezi jinými agenty, které jsou uloženy na odlišných procesorech.

Kooperativní chování = schopnost dvousměrné komunikace mezi agenty, které pak mohou společně provádět větší a komplexnější úlohy. Například pan X pošle panu Y elektronický dopis, který musí být neprodleně doručen. Agent nesoucí dopis od pana X se spojí s agentem pana Y a ten mu sdělí, že pan Y je na dovolené, tudíž je lepší zprávu odfaxovat sekretářce pana Y.

Proč potřebujeme inteligentní informační agenty ?

Uživatelé jsou v současné době zahlceni obrovským množstvím informací, které nabízí Internet, různé databáze apod. Potřebují rychle získat žádané informace v co nejkratší době, proto je nutno se zaměřit na nástroje (tzv.agenty), které by byly s to roztřídit a profiltrovat přicházející data do lehce manipulovatelného a přehledného množství relevantních informací, které jsou šité na míru daného uživatele.

Vzrůstá množství lidí, kteří mají zaměstnání, při kterém se nemohou neustále zdržovat na jednom místě, tudíž elektronické zprávy je třeba inteligentně směrovat a filtrovat.

Také tu máme pracovníky v oblasti managementu apod., kteří se musí rychle rozhodovat a mají velkou zodpovědnost.

V dnešní bouřlivě se rozvíjející společnosti je třeba minimalizovat čas strávený nad rutinními úkoly, aby se člověk vůbec také někdy dostal k odpočinku, zábavě, ke svým koníčkům apod.

Co je inteligentní agent trochu jinak:

Nejčastěji se objevuje definice: *inteligentní agent je software, který pomáhá uživateli a samostatně jedná podle jeho vůle.* Inteligentní agent se od jiného software odlišuje aspektem pověření agenta uživatelem k samostatnému provádění nějakého úkolu. Inteligentní agent má následující základní vlastnosti:

- **agent je autonomní** – má kontrolu nad vlastními akcemi,
- **agent je řízen cílem** – agent může být program, skript, nebo množina pravidel,
- **agent reaguje** – registruje změny prostředí a na změny odpovídá provedením akce,
- **nepřetržitý běh** – bez přítomnosti uživatele.

Mimo tyto vlastnosti společné pro všechny inteligentní agenty, může mít inteligentní agent další vlastnosti, které určují jeho speciální poslání:

- **společenskost** – agent může komunikovat s jinými agenty,
- **přizpůsobivost** – agent mění své chování na základě předchozí zkušenosti,
- **mobilita** – agent se může přemisťovat v síti z jednoho stroje na druhý.

Hlavní oblasti použití inteligentních agentů:

- hledání a filtrování informací,
- uzpůsobení informací (customizing),
- automatické reagování na signály.

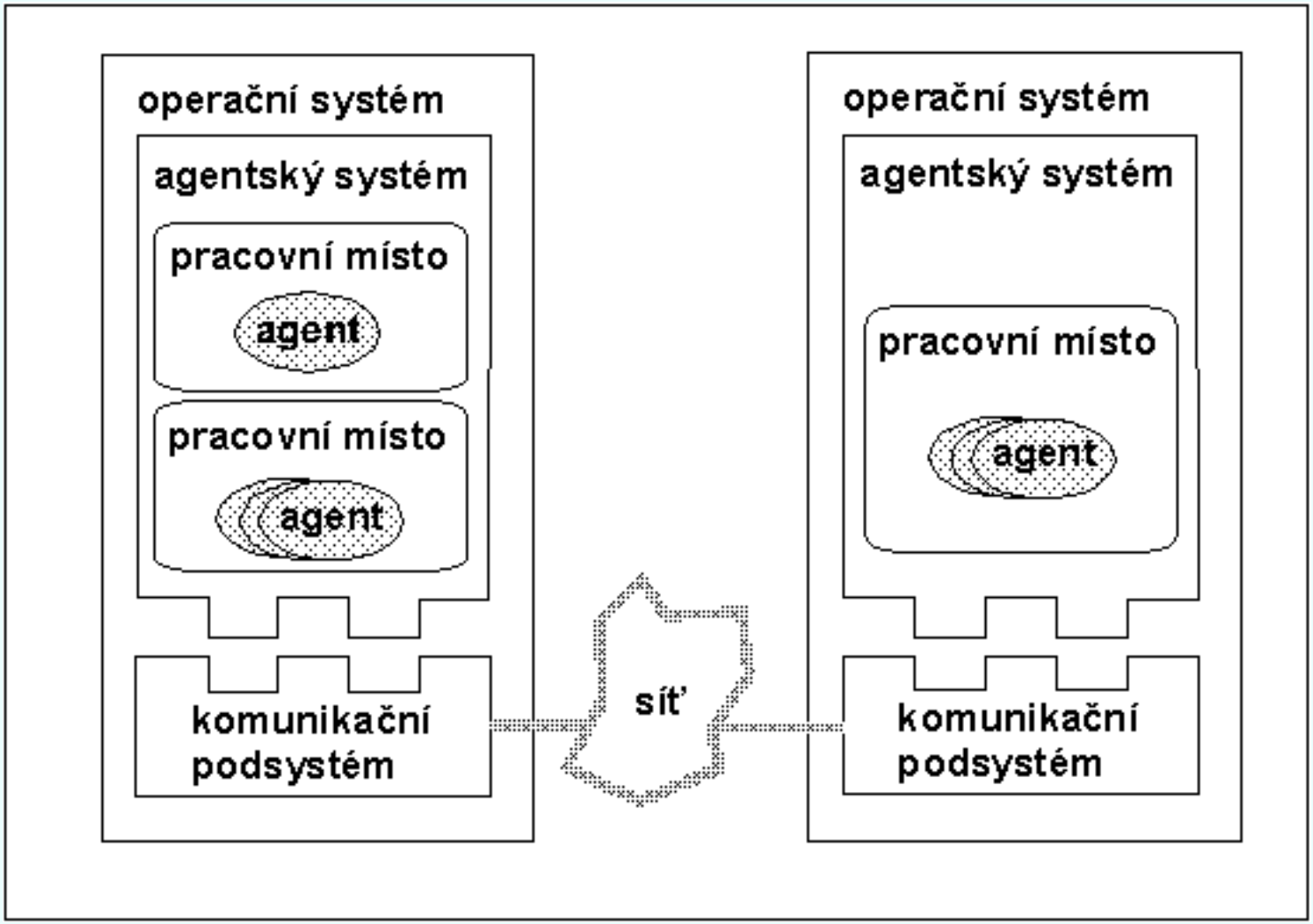
Cílem použití inteligentních agentů je

získat správné informace ve správný čas .

Struktura inteligentních agentů

- program agenta:
  - funkce zobrazující vjemy na akce
- architektura:
  - výpočetní prostředek (počítač, speciální HW)
- agent = architektura + program

Začlěnění agentského systému



Ideální racionální agent

- pro každou možnou posloupnost vjemů ideální racionální agent na základě faktů získaných posloupnosti vjemů a veškerých zabudovaných znalostí vykoná akci, u které je možné očekávat, že maximalizuje míru jeho výkonnosti

Míra výkonu (performance measure)

- objektivní kritéria pro měření úspěšnosti agenta
- výkon za dlouhou dobu (směna, životnost)

Racionalita versus vševědoucnost

- očekávaný úspěch na základě vnímaného

Příklad: Přenos agenta z jednoho agentského systému na druhý zahrnuje:

- **Inicializace přenosu**. Žádá-li agent po agentském systému přenos, musí zadat cíl přenosu, tj. cílový agentský systém. Není-li zadáno konkrétní pracovní místo, agent poběží na defaultním pracovním místě. Zdrojový agentský systém ukončí běh vlákna agenta, identifikuje a serializuje části stavu agenta, které se mají přenést, přenese agenta vhodným přenosovým protokolem.
- **Příjem agenta** na cílovém agentském systému (deserializace, obnovení stavu, spuštění agenta).
- **Přenos tříd** potřebných pro běh agenta na cílovém agentském systému. Zahrnuje nejen vlastní třídu agenta, ale všechny další třídy, které agent ke své práci potřebuje. Není však vždy nutné všechny třídy přenášet. Cílový agentský systém může mít potřebné třídy již získané z dřívější doby.

Přenos potřebných tříd má několik základních variant:

1. Automatický přenos všech potřebných tříd. V této variantě (i ve variantě 3) je nutné, aby zdrojový agentský systém byl schopen všechny potřebné třídy identifikovat. Nejčastěji ve spolupráci s linkerem.
2. Automatický přenos pouze agentské třídy, ostatní třídy na žádost cílového agentského systému v okamžiku, kdy je při běhu agenta potřebuje. Problém nastává, pokud se zdrojový agentský systém nebo vzdálený klient po přenosu agenta odpojí.
3. Přenáší se seznam všech potřebných tříd, cílový agentský systém si ihned vyžádá všechny třídy, které ještě nemá.

Problém: Bezpečnost agentských systémů

Z rozboru možných ohrožení bezpečnosti agentského systému vyplývá několik základních požadavků na bezpečnostní službu. Bezpečnostní služba musí umožňovat:

- **Autentikaci klienta pro vzdálené vytvoření agenta**. Typicky, pracuje-li uživatel v systému, který nepracuje s mobilními agenty. Autentikace může být provedena zadáním hesla, 'smart card', geometrií ruky či otiskem prstu aj.; autentikace určuje, jakou bezpečnostní politiku bude agentský systém vůči agentovi uplatňovat.
- **Vzájemnou autentikaci agentských systémů.** Rozumí se bez interakce s uživatelem. Lze realizovat udržováním důvěrné informace na agentském systému, např. privátní klíč.
- **Přístup agentského systému k výsledkům autentikace.** Před přijetím agenta musí mít cílový agentský systém přístup k informacím o zdrojovém agentském systému a agentovi, které umožní aplikovat vhodnou bezpečnostní politiku po přijetí agenta.

- **Autentikaci agenta a zjištění jeho práv.** Současně s přenosem agenta se musí přenášet jeho práva na provádění určitých akcí, aktuálně aplikovanou bezpečnostní politikou na hostitelském agentském systému se mohou práva oslabit.
- **Uplatnění bezpečnostní politiky agenta nebo agentského systému.** Agent i agentský systém typicky umožňují jiným žadatelům přístup ke svým metodám, ale musí mít nad přístupem kontrolu. Kontrolu přístupu může provádět sám agent nebo agentský systém, pak musí být schopen sám autentikovat žadatele a zjistit jeho práva, nebo agent či agentský systém může vytvořit přístupový seznam a předat jej komunikačnímu podsystému, který jej bude pri žádosti o přístup kontrolovat.
- **Volbu bezpečnosti komunikačního kanálu.** Žadatel o komunikaci si může vybrat úroveň jejího zabezpečení (odolnost proti odposlouchávání a dekódování prenášených dat, odolnost proti poškození nebo neautorizovaným změnám v datech při přenosu, odolnost proti ztrátě některých zpráv nebo jejich opakovanému doručení).

Současnými bezpečnostními metodami je neřešitelný problém autentikace agenta při vícenásobném přenosu, kde zdrojový agentský systém je jiné autority než agent. Agent sebou nemůže nosit privátní klíč (pro vytvoření ověřitelného podpisu pod svým stavem). Podpis je tedy schopen vytvořit pouze agentský systém stejné autority.

Agentský systém může v rámci aplikace bezpečnostní politiky vůči agentům omezovat jejich možnosti v následujících oblastech:

- **omezení funkcí agenta** – např. vytvářet, ukončovat běh jiných agentů, komunikace s agenty, možnosti vlastního přenosu,
- **limity zdrojů** – např. CPU, paměti, počet síťových spojení,
- **přístup k lokálním zdrojům**.

Příklad: speciální případ inteligentního agenta – Neugent

Americká firma Computer Associates v Mnichově 8. prosince ohlásila, že začíná dodávat první implementace technologie inteligentních softwarových agentů Neugents, které jsou samoučící se neuronovou síťovou technologií. Neuronová síťová technologie je softwarem, který má fungovat jako lidský mozek – má schopnost se učit, nabírat s časem znalosti, a aplikovat tyto znalosti na nové situace.

Neugents proto díky samoučícím se funkcím odstraňují nutnost ručních zápisů (skriptů) pro sledování problémových stavů a zařízení, a také umožňují zrychlit nasazování řešení správy podnikových prostředků.

Inteligentní softwarový agent Neugent „umí žít“ v nejrůznějších výpočetních prostředích, rozpoznávat různé vzory a zaznamenávat výsledné přechodové stavy. Síťové agenty Neugents umí analyzovat historická data o výkonu systému, vytvářet profil personality systému a porovnávat ji se současně sledovaným stavem. Na tomto základě mají být schopny předvídat problémy se systémem, ještě než k těmto problémům dojde. Agenty Neugents jsou tedy např. schopny předvídat, zda má v budoucnu dojít k výskytu chybového stavu.

Programování inteligentních agentů a programovací jazyky pro jejich tvorbu

Z důvodů přenositelnosti se pro programování inteligentních agentů používají pouze interpretované programovací jazyky. Nejčastěji to jsou:

- **Java** – při použití tohoto jazyka je nemožné zjistit stav agenta v okamžiku jeho žádosti o přesun, problém se obchází použitím aktualizovatelného itineráře cesty;
- **Tcl/Tk** – skriptovací jazyk implementovaný jako knihovna C procedur, Tk je nástroj podporující prvky GUI;
- **Perl** – skriptovací jazyk podobný jazyku C, resp. C-shell, rychlejší nez Tcl;
- **KQML** a **ACL** – jazyky používané pro komunikaci mezi agenty.

Poznámka: Mimo zmíněné jazyky se v různých spíše výzkumných projektech používají jejich rozšíření nebo jiné méně rozšířené či prototypové jazyky.

Příklady:

JINNI – Intelligent Mobile Agent Programming at the Intersection of Java and Prolog

JINNI (**J**ava **IN**ference engine and **N**etworked **I**nteractor), is a lightweight, multi-threaded, logic programming language, intended to be used as a flexible scripting tool for gluing together knowledge processing components and Java objects in distributed applications.

JINNI threads are coordinated through blackboards, local to each process. Associative search based on term unification (a variant of Linda) is used as the basic synchronization mechanism. Threads are controlled with tiny interpreters following a scripting language based on a subset of Prolog.

Mobile threads, implemented by capturing first order continuations in a compact data structure sent over the network, allow Jinni to interoperate with remote high performance BinProlog servers for CPU-intensive knowledge processing and with other JINNI components over the Internet.

The synergy of these features makes JINNI a convenient development platform for distributed AI, and in particular, for building intelligent autonomous agent applications. The latest version of Jinni is available from .

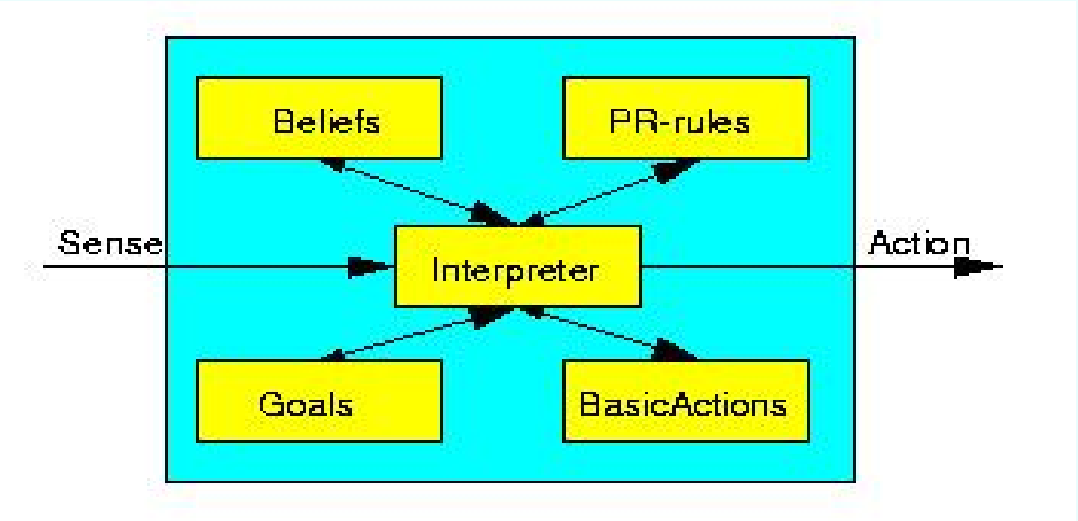
A stock market agent's buy / sell components look as follows:

```
sell(Who,Stock,AskPrice):-
  % triggers a matching buy transaction
  notify_about(offer(Who,Stock,AskPrice)).
buy(Who,Stock,SellingPrice):-
  % runs as a background thread
  % in parallel with other buy operations
  bg(try_to_buy(Who,Stock,SellingPrice)).
try_to_buy(Me,Stock,LimitPrice):-
  % this thread sets a blackboard constraint and waits
  % until the constraint is solved to true
  % by a corresponding sell transaction
  wait_for(offer(You,Stock,YourPrice),[ % server side mobile code
    lesseq(YourPrice,LimitPrice),
    local_in(has(You,Stock)),
    local_in(capital(You,YourCapital)), % server side 'local' in/1
    local_in(capital(Me,MyCapital)),    % operations
    compute('-',MyCapital,YourPrice,MyNewCapital),
    compute('+',YourCapital,YourPrice,YourNewCapital),
    local_out(capital(You,YourNewCapital)),
    local_out(capital(Me,MyNewCapital)),
    local_out(has(Me,Stock))
  ]).
```

3APL (triple-a-p-l) – an Abstract Agent Programming Language

3APL is a programming language for implementing cognitive agents. It provides programming constructs for implementing agents' beliefs, goals, basic capabilities (such as belief updates, external actions, or communication actions) and a set of practical reasoning rules through which agents' goals can be updated or revised. The 3APL programs are executed on the 3APL platform. Each 3APL program is executed by means of an interpreter that deliberates on the cognitive attitudes of that agent.

Agent structure:



BNF specification of 3APL Syntax

Examples:

Single Agent Program

```
1  PROGRAM "robot"
2  CAPABILITIES:
3      { pos(X,Y) } West() { NOT pos(X,Y) , pos(X - 1,Y) }
4  BELIEFBASE:
5      pos(100,10).
6  GOALBASE:
7      goWest()
8  RULEBASE:
9      goWest() <- pos(0,Y) | SKIP ,
10     goWest() <- pos(X,Y) AND X>0 | BEGIN West();goWest() END
```

Multiagent Program

```
1  PROGRAM "harry"
2
3  CAPABILITIES:
4
5  BELIEFBASE:
6  me(harry).
7  you(sally) .
8
9  GOALBASE:
10
11 RULEBASE:
12  <- you(You) | Send(O, You, inform, hello(You) ),
13  <- me(Me) AND received(V, You, inform, hello(Me)) AND
14     NOT sent(V,You,inform,thanks(You)) |
15              Send(O, You, inform, thanks(You) )
```

```
1  PROGRAM "sally"
2
3  CAPABILITIES:
4
5  BELIEFBASE:
6  me(sally).
7  you(harry) .
8
9  GOALBASE:
10
11 RULEBASE:
12     <- you(You) | Send(O, You, inform, hello(You) ),
13     <- me(Me) AND received(V, You, inform, hello(Me)) AND
14        NOT sent(V,You,inform,thanks(You)) |
15                Send(O, You, inform, thanks(You) )
```

Příklad programu v Javě: `GenericAgent.java`

```
import java.io.*;
import java.util.*;
import java.lang.*;
import webl.lang.*;
import webl.lang.expr.*;
import webl.util.*;

public class GenericAgent
{
   static String script = "";
   static WeblEngine webl_engine
      = new WeblEngine();
   static int number_of_patterns = 0;
   static int selected_pattern = 0;
   int search_str_set = 0;
   int current_pattern = 0;

   public GenericAgent()
   {
      executeScript("AgtFiles/Init.agt");
   }
```

```
    public void startAgain()
{
    webl_engine = new WeblEngine();
    executeScript("AgtFiles/Init.agt");
    search_str_set = 0;
}
public String executeScript(String filename)
{
    String line = null;
    String return_val = "";
    try
    {
        BufferedReader in = new BufferedReader(
            new FileReader(filename));
        while ((line = in.readLine()) != null)
        {
            script += line;
        }
        in.close();
        System.out.println("executeScript():" + filename + "\n"+ script + "\n");
        return_val = webl_engine.executeScript(script).print();
        script = "";
    }
```

```
      catch (FileNotFoundException fnf)
      {
         System.err.println("GenericAgent(): " +
            filename + " not found");
      }
      catch (IOException ex) { System.err.println(ex); }
      return return_val;
   }

   public String executeLine(String line)
   {
      Expr result;
      System.out.println("executeLine(): " + line + "\n");
      result = webl_engine.executeScript(line);
      return result.print();
   }

   public String searchWebSite(String website, String type_of_site,
      String search_string, String matching_power)
   {
      String search_str_line =
         "import Str; import Browser; var searchInput = \""
             + search_string + "\";" ;
```

```
      String type_of_site_line =
         "var searchType = \"" + type_of_site + "\";" ;
      String website_line =
         "var website = \"http://" + website + "\";" ;

      script = search_str_line +
         type_of_site_line + website_line;

      if (matching_power.length() != 0)
         executeLine("repeatThreshold = "
            + matching_power + ";");
      executeScript("AgtFiles/Search.agt");
      return "hello";
   }

   public String setSearchString(String search_str)
   {
      if (search_str_set != 0)
         return ";";
      search_str_set = 1;
      executeLine("searchString =\"" + search_str + "\";");
      return executeScript("AgtFiles/InitQuery.agt");
   }
```

```
 public void setStarttoParent()
{
   executeLine("childTag = Name(startQueryPiece);");
   executeLine("startQueryPiece = Parent(startQueryPiece);");
}

public void  executeQuery(int dir)
{
   String query = null;
   if (dir != 0) {
      String child_tag = executeLine("childTag;");
      String query_stmt = executeLine("queryStmt;");
      query = "Elem(P,\"" + child_tag + "\") directlyinside "
         + query_stmt;
   } else {
      query = executeScript("AgtFiles/Query.agt");
   }
   executeLine("records = " + query + ";");
   String ShowRec = "DummyRec = " + query
      + " contain (Pat(P,searchString)[0]);";
   executeLine(ShowRec);
   executeScript("AgtFiles/ShowRec.agt");
}
```

```
  public int getNumberOfPatterns()
{
    String result = executeLine(" Size(ToList(patterns));" );
    number_of_patterns = Integer.parseInt(result);
    return number_of_patterns;
}

public String getFirstPattern()
{
    String index = "";

    if (number_of_patterns == 0)
       return null;

    current_pattern = 0;
    index = String.valueOf(current_pattern);
    executeLine("import Str; tags = Str_Search(patterns[" +
       index + "][1], \"[<](.*?)[>]\");");
    executeScript("AgtFiles/Pattern.agt");
    showPattern();

    return "hello";
}
```

```
public String getLastPattern()
{
    String index = "";

    if (number_of_patterns == 0 )
       return null;

    current_pattern = number_of_patterns - 1;
    index = String.valueOf(current_pattern);
    executeLine("import Str; tags = Str_Search(patterns[" +
       index + "][1], \"[<](.*?)[>]\");");
    executeScript("AgtFiles/Pattern.agt");
    showPattern();

    return "hello";
}

public String getNextPattern()
{
    String index = "";

    if (number_of_patterns == 0 )
       return null;
```

```
        if (current_pattern >= number_of_patterns)
           return null;
        current_pattern ++;
        index = String.valueOf(current_pattern);
        executeLine("import Str; tags = Str_Search(patterns[" +
           index + "][1], \"[<](.*?)[>]\");");
        executeScript("AgtFiles/Pattern.agt");
        showPattern();

        return "hello";
    }

    public String getPreviousPattern()
    {
        String index = "";

        if (number_of_patterns == 0 )
           return null;
        if (current_pattern == 0)
           return null;
        current_pattern --;
        index = String.valueOf(current_pattern);
        executeLine("import Str; tags = Str_Search(patterns[" +
```

```
            index + "][1], \"[<](.*?)[>]\");");
        executeScript("AgtFiles/Pattern.agt");
        showPattern();
        return "hello";
    }

    public String showPattern()
    {
        String result = executeLine("Size(ToList(weblstmt));");
        int numberStmt = Integer.parseInt(result);
        String[] weblstmt = new String[numberStmt];
        for (int i = 0; i < numberStmt; i++)
        {
            result = executeLine("weblstmt[" + String.valueOf(i) + "];");
            weblstmt[i] = result;

            String webltext1 = "weblobj[" + String.valueOf(i) + "] := "
                + weblstmt[i] + "; weblmarkuptxt = weblmarkuptxt + Markup(weblobj[" +
                    String.valueOf(i) + "][3]);";

            String webltext2 = "if (Size(weblobj[" + String.valueOf(i) +
                "]) > 0 and minweblobj == -1) then minweblobj = " +
                    String.valueOf(i) + "; end;";
```

```
        String webltext3 = "if (Size(weblobj[" + String.valueOf(i) +
           "]) < Size(weblobj[minweblobj]) ) then minweblobj = " +
               String.valueOf(i) + "; end;minweblobj;";

        String webltext = webltext1 + webltext2 + webltext3;
        executeLine(webltext);
      }
      String browser_text1 = "import Browser;";
      String browser_text2 = "Browser_ShowPage(\"<html><body><hr>\"
                             +weblmarkuptxt+\"<hr></body></html>\");";
      String browser_text3 = "weblmarkuptxt = \"\";";
      executeLine(browser_text1 + browser_text2 +
         browser_text3);

      return "hello";
   }

   public int selectPattern()
   {
      selected_pattern = current_pattern;
      executeScript("AgtFiles/Select.agt");
      return selected_pattern;
   }
```

```
public int getNumberOfResults()
{
   String result = executeLine("Size(txt);");
   int number_of_results = Integer.parseInt(result);
   return number_of_results;
}


public String findPattern(int index, String pattern)
{
   String line1 = "import Str; currList = nil;";
   String line2 = "currList = Str_Search(txt[" +
      String.valueOf(index) + "],\"" + pattern + "\");" ;
   String line3 =
      "if ( Size(currList) != 0) then Str_Trim(currList[0][1]); else \"empty\";
         end;";
   String result = executeLine(line1 + line2 + line3);
   return result;
}
```

```
    public String executeStmt(int index, String stmt)
    {
       String line1 = "dummy =\"\";CurrentRec = NewPiece(Markup(records["
             + String.valueOf(index) +
                "]), \"text/html\");";

       String line2 = "import Str;";
       String line3 = stmt;
       executeLine(line1);
       //String CheckNilStmt = executeLine("CheckNilStmt;");
       //if (executeLine(CheckNilStmt).equals("empty"))
       ///   return "empty";
       String pre_result = executeLine(line2 + line3);
       StringTokenizer st = new StringTokenizer(pre_result);
       String result = "";
       while (st.hasMoreTokens())
          result += st.nextToken() + " ";
       return result;
    }
}
```