วิธีการแบบไฮบริดในการพัฒนาระบบไดอารีบนเว็บ

Hybrid Approach for Web-Based Diary Information System Development

วราภรณ์ จิรชีพพัฒนา*
อาจิน จิรชีพพัฒนา
เอมฤดี สุขพัน**
เดือนเพ็ญ สันติธรรม***
Waraporn Jirachiefpattana, Ph.D
Ajin Jirachiefpattana, Ph.D.
Amrudee Sukpan
Duenpen Santhititham

บทคัดย่อ

Web-based เทคโนโลยีบนเครือข่ายนับเป็นประเด็นที่ท้าทายอย่างมากต่อองค์กรทางธุรกิจในปัจจุบัน เนื่องจากความซับซ้อน การเปลี่ยนแปลงอย่างรวดเร็ว และความสำคัญที่มีต่อการอยู่รอดขององค์กร นอกจากนี้ เทคโนโลยีดังกล่าวยังมีผลกระทบอย่างใหญ่หลวงต่อการแบ่งปันและแลกเปลี่ยนข้อมูลของผู้คนและองค์กร ในปัจจุบันนี้ การติดต่อสื่อสารกับพนักงานขององค์กรในพื้นที่ห่างไกล หรือคู่ธุรกิจรวมถึงลูกค้าที่อยู่กระจายทั่วโลก สามารถทำได้โดยง่ายและไม่สิ้นเปลือง Web ยังช่วยให้เกิดความร่วมมือกันระหว่างกลุ่มคนต่างๆที่อยู่กระจัดกระจาย อย่างไรก็ตามการที่จะพัฒนาระบบสารสนเทศบน web โดยใช้เพียงเทคโนโลยีเดียวนั้นไม่เพียงพอ บทความนี้เสนอแนวทางผสมโดยการใช้เทคโนโลยี 3 ชนิด คือ Web Support, Logic Programming และ Agent ซึ่งยังไม่เคยมีการนำมาใช้ร่วมกันมาก่อนในการพัฒนาระบบสารสนเทศบน Webโดยเสนอการพัฒนาระบบจัดตารางนัดหมายเป็นตัวอย่าง

*School of Applied Statistics, National Institute of Development Administration (NIDA)

**Department of Information Technology, Faculty of Science and Technology, Walailuck University, Nakornsrithammarat, THAILAND

***Department of Mathematics, Faculty of Science, Thaksin University, Songkhla, THAILAND

Abstract

Web-based technologies pose the most significant challenges for business organizations at the end of the 20th Century due to their complexity, their pace of change, and their importance to the future viability of any company milieu. They have significantly changed the way people and organization share and exchange information. Now, more than at any other point in history, it is both easy and affordable to communicate with local or remote employees, business partners, and customers throughout the world. Consequently, the Web could also be used to support collaboration among dispersed groups of people. To develop an information system on the Web, however, only one technology is not sufficient. Thus, this paper presents a hybrid approach integrating three technologies which have not previously been utilized together in any information systems: *Web support*, *logic programming* and *agents*. This approach has been applied to a specific form of groupware: *diary scheduling* to demonstrate its usefulness.

1. Introduction

We are in the midst of a paradigm shift, where the tenets of commercial behavior and rules of engagement are being swept away by the upheavals embodied in the Internet and the Web. Fresh modes of commerce must be invented to address the societal computing models being developed in research labs and software houses. The pervasiveness of the Web will only deepen, for it seems likely that computing will become increasingly network-based. Already the major small business-oriented operating system, *Microsoft Windows*, is attempting to integrate the commonplace desktop (our local working environment) with the Web (the networked marketplace), making them *synonymous*.

In addition to Web-enabled technologies, groupware applications have rapidly become a major player in the software industry, developed in response to cries from the oft-bewildered business community faced with electronic data management, office productivity problems, and team co-ordination requirements (Schlosberg, 1995).

This paper, therefore, is concerned with a specific type of groupware: *diary scheduling*, and concentrates on aspects of societal computing which are sadly absent from most existing products. Our prototype system is

an attempt to address, and reconcile, three strands of the multifaceted needs of diary systems in the next century.

Firstly, diary scheduling is becoming intensely knowledge-based, where stored information contains vastly richer interconnections between its parts, and the queries upon that information require much more complex kinds of manipulation. It seems inevitable that the current reliance on traditional relational databases and query languages like SQL (which originated in the 1970's) will be inadequate for the job. Our system utilizes a more powerful data formalism, called the *deductive database*, which can hold facts and rules about knowledge entities and, as importantly, can be queried using advanced techniques such as unification, logical inference, and the backtracking search mechanism. We utilise the logic programming language Prolog to support these features (Sterling and Shapiro, 1986).

The needs and advantages of using deductive database and logic programming, over relational database and SQL are that *we want to write down diary information as rules and facts*, which is much easier in logic programming. SQL is a query language, not a full programming language. Consequently relational databases cannot contain rules. In fact, all programming languages are *equivalent* in a mathematical sense, so any language would be *as good as* Prolog for this diary application. However, the effort required in coding up facts and rules in, for example, Visual Basic, is a lot more work than using Prolog.

The second thread of our work is the Web. Bearing in mind its elemental importance to our future, many diary scheduling applications are still not Web-based (Blue Cannon Software, 1996; EuroSoft, 1997; Sheridan Software, 1997). Some Web-centric tools are belatedly coming available, including Lotus Organizer Web Calendar (Lotus Corp., 1997) and Lotus Internotes which is a Web-publishing package with access to Lotus Notes. Unfortunately, these tools still rely on conventional database structuring and search.

It bears repeating that embracing the Web offers substantial benefits: it supports a full range of multimedia formats such as text, graphics, sound and animations; it can be accessible to the widest spectrum of people; it is not proprietary; it is virtually universal; it can be made secure (an important aspect for diary systems); and its familiar graphical interface makes it user-friendly enough even for the most virulent computer-phobe.

The third strand of our system acknowledges that the future of business contains human participants standing side-by-side with computer agents possessing machine-intelligence. This sounds somewhat fanciful today, but agents are even now starting to assist in corporate data mining and on-line financial co-ordination. Their importance will only grow, and our system investigates their usefulness in the diary

scheduling domain. We outline two different types of agents used in our system; however, they share a common implementation strata of logic programming. This permits them to utilise the same knowledge representation and analysis techniques inherent in our diary system.

As far as we know, our work is the only one to explicitly represent monitoring agent behavior as logic programming facts and rules, although IntelliDiary (Wada et al., 1996) is implemented in April, a language with several features borrowed from logic programming (McCabe, 1996).

2. Web-Based Logical Diaries

2.1 The Diary System

Figure 1 gives a simplified functional overview of our diary system:

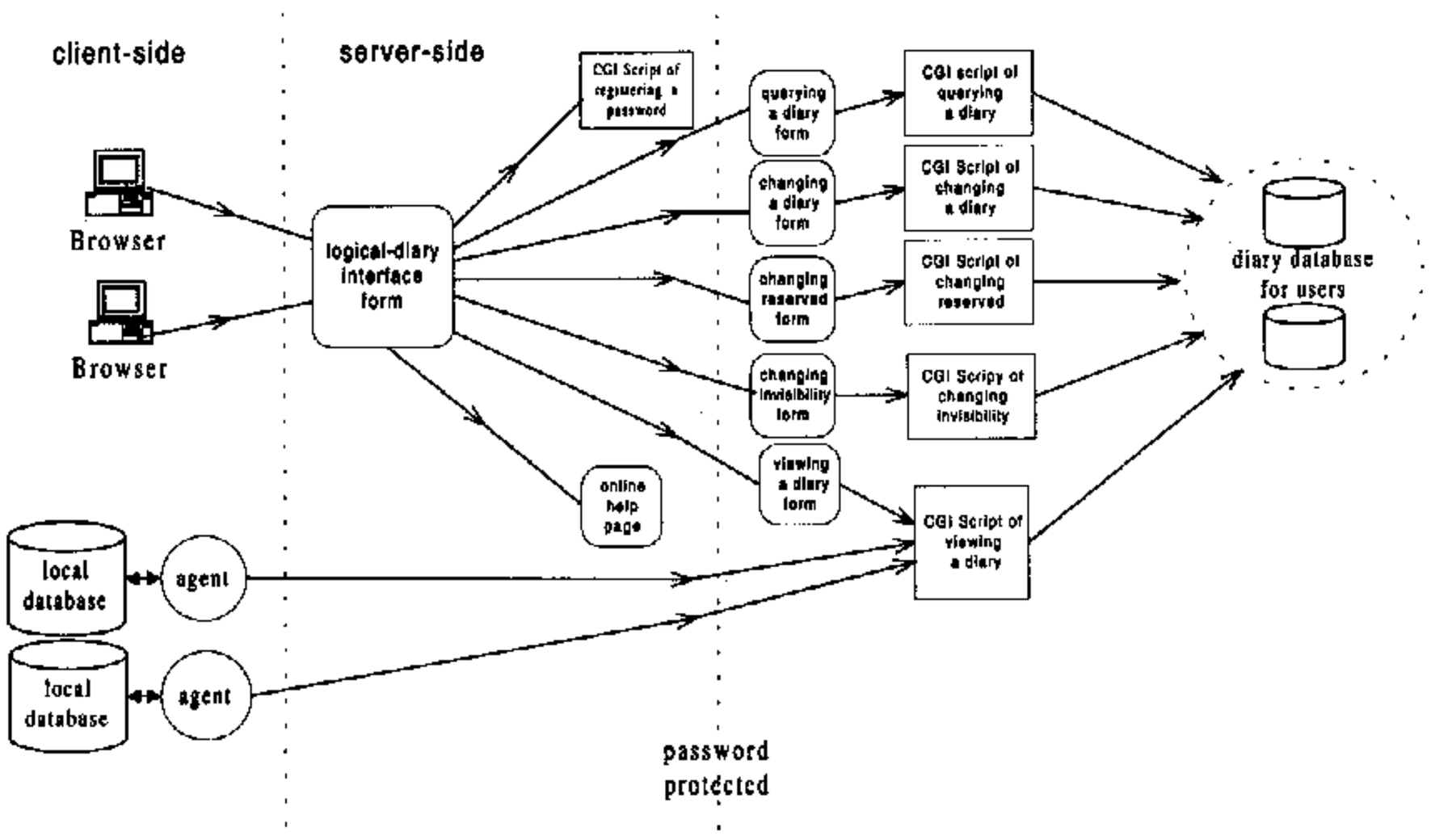

Figure 1: The Diary System.

The server side system consists of CGI scripts coded in BinProlog (Tarau, 1997), a popular, fast, and free variant of Prolog. Web pages containing forms are used to interact with the various scripts, which return answers as Web pages. We use the logic programming PiLLow Web library to process forms input and generate answer pages (Cabeza et al., 1997).

There is a diary database for each user, with access controlled by the password mechanism built into the Apache Web server (Apache Group, 1998). A new user must register a login ID and password, which will subsequently permit him/her to access other parts of the system. The registration will also initialize the person's diary database.

2.2 The Diary Database Data Structure

Each user has their own diary database, which typically consists of three predicates: diary/3, invisible/2, and reserved/1.

2.2.1 diary/3

The diary/3 predicate hold the diary information for the user. It contains facts of the form:

```
diary(Name, TimePeriod, Item).
```

Name is the name of the person/diary who the diary owner is meeting. TimePeriod has the form:

```
time(StartSlot, EndSlot, date(Day, Month, Year))
```

StartSlot and EndSlot are integers in the range 1, 2, ..., 24. For a start slot, the integer denotes the start of the hour (1 stands for midnight, 2 for 1am, and so on). For an end slot, the integer denotes the end of the hour (1 stands for 0:59am, 2 for 1:59am, and so on).

Item has the format:

```
item(topic(TopicTitle, Comments), Where, Importance)
```

TopicTitle, Comment, and Where are strings. TopicTitle holds the meeting title, Comment holds extra details related to the meeting, and Where gives the location of the meeting. Importance is an integer in the range 1,2, ...,10 used to rate the significance of the meeting: a higher value denotes higher importance.

Two example diary/3 facts:

```
diary('Board of directors', time(9,10,date(10,8,1998)),
      item(topic('ordinary meeting',
      'I must report on the progressional MML project'),
       'M.125 ',8)).
diary('#Jenny Jack',time(15,16,date(6,8,1998)),
       item(topic('MML Project',
       'Jenny and Jack will report on the progressional MML project'),
      'L.141 Laboratory',7)).
```

The first fact records a meeting with the Board of directors on August 10th between 8.00am and 9:59 am. A comment reminds the author to report on the progressional MML project. The meeting is in room M.125, and is high priority.

The second fact records a meeting with Jenny and Jack on August 8th between 2.00 pm. and 3.59 pm. to discuss the MML project. The meeting is in lab room L.141, and has a lower priority than the first appointment.

2.2.2 invisible/2

The invisible/2 predicate is used to hide parts of the owner's diary from other users. In particular, this information is utilised to filter the diary/3 facts before they are displayed by the "view a diary" CGI script (see section 3.6). invisible/2 clauses have the form:

```
invisible(UserID, TimePeriod) :- /* extra conditions */.
```

UserID is the ID of the person whose viewing is being restricted, or can be the word "all" to mean all users (except the diary owner). The user ID of the diary owner is known to the system when a user logs on. TimePeriod has the same format as in diary/3. The extra conditions usually relate to values inside the TimePeriod term, in order to generalise the invisibility range. Two examples:

```
invisible('Jack', time(9,10,date(10,8,1998))).
invisible(all, time(1, 24, date(X, 8, 1998))) :-
                    X >= 5, X =< 12.
```

The first fact states that user Jack cannot see this diary's details for meetings between 8:00 am and 9:59 am on August 10th. The rule states that everyone (except the diary owner) is prevented from seeing diary information concerning August 5th to 12th 1998.

A more advanced use of invisible/2 is to link invisibility to the presence (or absence) of diary/3 and reserved/1 clauses in the database. For instance:

```
invisible('Jack', TimePeriod) :-
        diary(Name, TimePeriod, _), Name \== 'Jack'.
```

This prevents Jack from seeing diary information for users other than himself.

2.2.3 reserved/1

The reserved/1 predicate reserves time slots in the owner's diary so that no other users can make appointments. reserved/1 clauses have the form:

```
reserved(TimePeriod) :- /* extra conditions */ .
```

As with invisible/2, the extra conditions allow the time period to be generalised or made dependent on diary/3 or invisible/2 clauses. For example:

```
reserved(time(16, 16, date(7, 2, 1998))).
reserved(time(13, 13, _)).
reserved(time(1, 24, Date)) :-
  date2day(Date, Day), Day \== "Sat", Day \== "Sun".
```

The first fact states that the 3pm to 3:59 pm slot on February 7th is reserved. The second fact uses an uninstantiated date variable to reserve every day between noon and 12:59 pm (for lunch presumably). The final clause uses the built-in predicate date2day/2 to reserve every Saturday and Sunday.

2.2 Querying a Diary

The user can query a diary by submitting a HTML form, such as the one shown in Figure 2. Our form interface was designed for people with little or no experience of Prolog. However, knowledge of Prolog is required for more advanced queries. The details of how to make conditional queries and to update the diary can be found in (Sukpan et al., 1998).

Figure 3 gives the results after submitting Figure 2 to the system (by pressing the "Execute" button). The resultant table has two columns : one for the names of the diary variables and the other for their values. The first table entry states that John has a metting with Jenny on the 15th at 19.00-20.59 for dinner. The next entry has John meeting with Jenny on the 22nd at 19.00-21.59 for a movie.

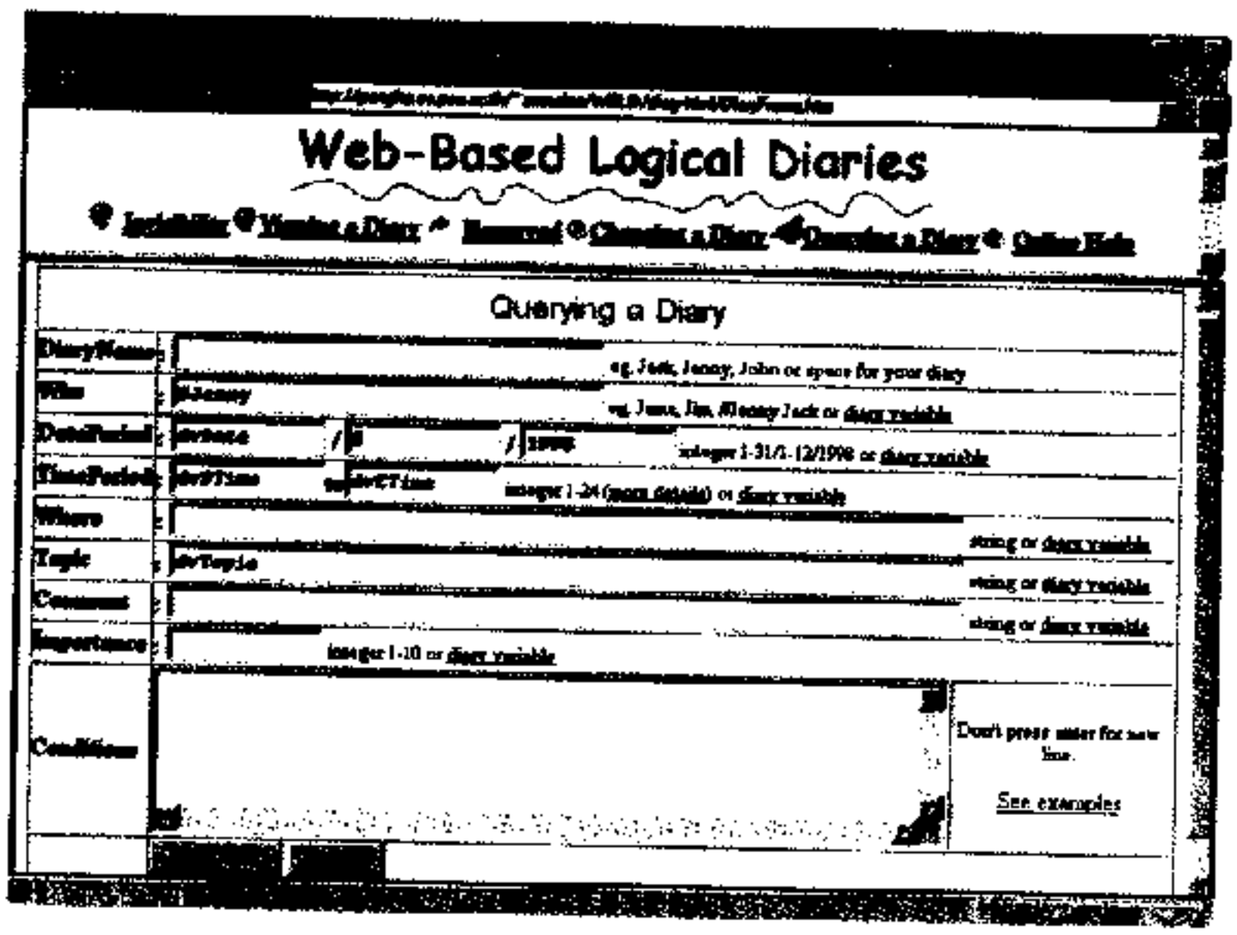

Figure 2 : Interface for the first query.

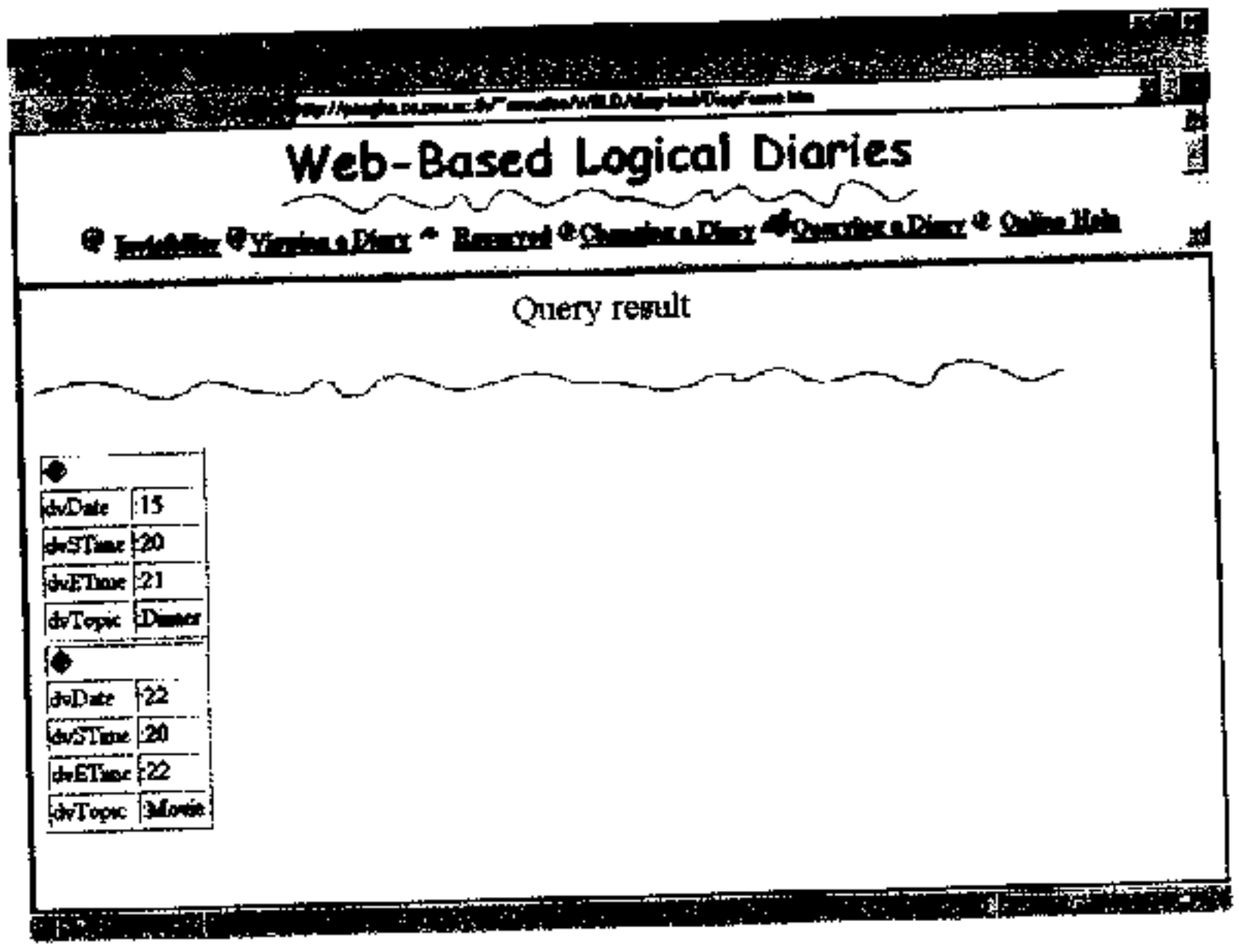

Figure 3: Results for the first query given in Figure 2.

3. Agent Helpers

The agent helpers (Jirachiefpattana et al., 1999) shown in Figure 1 illustrate how monitoring is integrated into the system. Each agent examines a diary by periodically downloading its 'visible' details by accessing the CGI script for viewing a diary. An agent does this by sending a POST query to the script (Berners-Lee et al., 1996), and by supplying a user's login ID and password to access the system.

When an agent gets back a Web page of details, it extracts the diary/3 facts, which are included in the page but commented out so that browsers do not display them. The extracted facts are stored *locally* by the agent, as are later diary downloads (of the same diary or of other diaries). The resulting collection of databases is analysed by the agents using logic programming techniques.

Emphatically our logic agent and its local database are on the client side which will make the system less portable than other web-based server-side only system, and then extra storage and computational power are needed on the client side. In fact, there are advantages and disadvantages to client-side and server-side computation. The decision about which to use has to be based on the particular problem. In this paper, however, our reason for the agents having their local database on the client side is that the agents are meant to be working for the clients, not part of the server; placing the agents on the server-side means they are no longer "owned" by the clients – they become another part of the server. That means the clients cannot change them, and will have limited ability to configure them, depending on the server.

Our agents are coded using BinProlog, although the page downloading sub-system is written in C. We plan to reimplement this in Prolog. We are investigating two types of agent helper: agents which monitor a single diary, looking for additions, deletions or updates, and agents which monitor several diaries. In the second category, we have developed an agent that can examine several diaries for free times which are 'near' to each other. Nearness is defined in terms of user-defined rules. The analysis often leads to a choice of free times, and the agent employs user-specified 'preference' rules to rank them.

We chose to implement a free-time agent since its assistance is extremely useful when someone is trying to organise a meeting involving several people. Both types of agent notify their users by sending e-mail to specified addresses.

3.1. The "diary changes" Agent

This agent monitors a specified diary for any changes by regularly downloading the visible diary/3 facts from the specified diary and storing them in a database. This database is compared with the last download of the same database using comparison rules written in Prolog by the agent's user. The agent (called dca) takes at most eight arguments, given as:

```
dca [-e examination_period][-s stop_time][-d diary_name]
    [-l login_ID][-p password][-a e-mail_address]
    [-m month/year][-h]
```

Fortunately, dca has default settings for all of these arguments which are therefore optional (denoted by [...]). The examination period is the number of minutes between downloads (the default is 15 minutes). The stop time is the number of hours from now when the agent will terminate (the default is 24). The diary name is the diary to be monitored (the default is the user's diary, whose name is assumed to be the same as the user's $USER environment variable under UNIX). The login ID is required to access the diary system, and is also assumed to be $USER by default. A password is also required to enter the diary system, which also defaults to $USER. The e-mail address is the address where dca should post change notifications. By default, the address is built from $USER and the machine's host name. The month/year argument states which month in the diary should be examined. Presently, only a single month at a time can be observed, but we hope to extend this. The default setting is the current month. The -h option prints help on calling dca. A simple dca invocation by user *duenpen* would be:

```
dca -s 48 -d joe -p 216wZp
```

dca will examine the current month of *joe*' s diary every 15 minutes for 48hours. The diary system is accessed with login ID "*duenpen*" and password "216wZp". E-mail reports will be sent to "*duenpen*" on the

host machine (if the login-id of user on the diary system is the same login-id on the host machine). dca is a fairly simple C program which uses a timer to regularly execute the agent's analysis code written in BinProlog. A typical e-mail message sent by dca is shown in Figure 4 and 5.

```
Date: Wed, 2 Sep 1998 15:20:12 -0700
From: duenpen (Duenpen Santhititham)
To: duenpen@pangha.cs.psu.ac.th
Subject: No Change

On Date 2/9/98 at 15:20:12
********* No change to joe's diary **********
```

Figure 4 : No changes to Joe's diary.

It reports that Joe's diary is unchanged at 15:20:12 on September 2nd. The next time, dca sends:

```
Date: Wed, 2 Sep 1998 15:25:12 -0700
From: duenpen (Duenpen Santhititham)
To: duenpen@pangha.cs.psu.ac.th
Subject:<Topic & Location changed>  <Appointment deleted>

Current Appointment :-
 Name : #Tom Eddy
 Date : 21/9/1998
 Time : 13.00-14.59
 Topic: Gift shop sale
 Comment: no
 Where: Central
 Important: 1.
Topic changed from Shop sale on air to Gift shop sale.
Location changed from Diana Department Store to Central.

Deleted Appointment :-
 Name : Koko
 Date : 21/9/1998
 Time : 9.00-9.59
 Topic: Got salary
 Comment: no
 Where: Bangkok Bank
 Important: 1.
```

Figure 5 : Changes to Joe's diary.

Figure 5 reports two changes to Joe's diary: an existing appointment time slot has a new topic and location, and an appointment has been deleted.

The core of the agent's behavior is based on comparing the previous version of the database (called $D_{previous}$) with the current one (called $D_{current}$). The deleted facts are identified as:

$$D_{deleted} = D_{previous} - (D_{current} \cap D_{previous})$$

The added facts are identified as:

$$D_{added} = D_{current} - (D_{current} \cap D_{previous})$$

The user can specify actions to carry out when certain facts are deleted and/or added by defining changed/3 clauses of the form:

```
changed(<deleted diary/3 fact>, <added diary/3 fact>,
                              "Subject line for the e-mail") :-
        /* extra conditions */ .
```

Two examples:

```
changed( diary(Name, time(_, _, date(X, 9, 1998)), Item),
         diary(Name, time(_, _, date(Y, 9, 1998)), Item), Subject)
:-
          X > 11, Y <= 11,
          sappend(Name, " appointment moved", Subject).
changed( diary(Name, Time, item(Topic, OldWhere, _)),
         diary(Name, Time, item(Topic, NewWhere, _)),
                                        "Location Moved") :-
         OldWhere \== NewWhere.
```

The first rule causes the user to be alerted when an appointment with Name about Item scheduled after September 11th is moved to the 11th or before. The second rule triggers a message if the location of an appointment is changed. Note the use of sappend/3 to create the e-mail subject line in the first clause. This string becomes the subject header of the e-mail sent to the user.

changed/3 clauses can be used to detect deletions by leaving the second argument unbound. For example:

```
        changed( diary(Name,time(14, 15, date(_, 9, 1998)),_), _, Subject)
:-
        sappend(Name, " appointment deleted", Subject).
```

This will cause an e-mail message to be sent when any diary entry between 1pm and 2:59pm in September is deleted.

In a similar way, changed/3 can be employed to monitor only additions by leaving the first argument unbound. For example:

```
changed(_, diary(_, _, item(topic(Title,_), _, _)), Subject) :-
        contains("salary", Title),
        sappend(Title, "appointment added", Subject).
```

This rule will trigger a message if the diary owner makes an appointment which involves the string "salary" in its title.

The series of high-level changed/3 clauses are collectively applied to the database changes using a more complex version of the following findall/3 query:

```
?- findall( change(Dd, Da, Subj),
              ( member(Dd, Ddeleted), member(Da, Dadded),
                changed(Dd, Da, Subj) ),
            Changes).
```

The list of changes collected in Changes are subsequently reformatted and sent as an e-mail message to the specified address.

3.2. The "free time" Agent

The "free time" agent monitors a series of specified databases, looking for free time common to all the diaries. The rules to decide if free times in different diaries are "near" to each other are coded by the user with logic programming clauses.

The diaries are regularly downloaded by accessing the "view a diary" CGI script for the diaries under study. The agent removes the diary facts from the retrieved pages and places them in local databases. These extracted facts have already been filtered on the server-side so that facts not visible to the agents have been removed. However, reserved/1 clauses are not currently downloaded, which means that the agent cannot discern which free times are actually reserved. The agent (called fta) is specified in a similar way to the diary changes agent:

```
fta [-e examination_period][-s stop_time]
    [-d diary_name]*|[-n local_name]
    [-l login_ID][-p password][-a e-mail_address][-r range]
    [-m file_name][-h]
```

The first difference is that zero or more diary names ,if there is a -d diary_name then there cannot be a -n local_name, these can be supplied on the command line, which list the diaries to be monitored for collective free time. The user's diary is always monitored, and so does not need to appear as an argument.

The free time agent can also be used to watch for changes to appointments with a specific person the user's diary by means of the -n option. The default behavior is for the agent to watch for any kind of free time in the user's diary only (i.e unrestricted to a specific user).

The range argument constrains the agent's examination to a fixed period. If this restriction was not applied, then the agent could end up examining a very large time span for free times. Range has the format :

```
start_period - end_period : start_day - end_day /month/year
```

For example:

```
13-15:20-21/9/1998
```

This informs the agent to monitor the time period 13-15 (noon to 2.59pm) on the 20th and 21st September 1998. Currently, fta only permits ranges that span a single month.

The file_name option gives a filename holding appointment details. This signals to fta that it should try to make the specified appointment if it finds a single free time that matches the user's 'nearness' rules (see below). The file may contain details concerning the appointment's topic, location, priority, and general comments. A simple fta call by user duenpen would be:

```
fta -d tom -d eddy -d joe -p 216wZp -r 13-15:20-21/9/1998
```

This tells the free time agent to watch for common free time in the diaries of "duenpen" (user-id was registered in diary system is duenpen too.), tom, eddy, and Joe, occurring between noon and 2.59pm on 20-21st September. No appointment is set. When free time is found, an e-mail message is sent to duenpen, as shown in Figure 6.

```
Date: Wed, 2 Sep 1998 16:35:33 -0700
From: duenpen (Duenpen Santhititham)
To: duenpen@pangha.cs.psu.ac.th
Subject:  <near time>

Free time of duenpen
and,tom,eddy,joe
during date: 20/9/1998 to 21/9/1998 ,
That is date 20/9/1998. Time: 12.00-14.59

Free time of duenpen
and,tom,eddy,joe
during date: 20/9/1998 to 21/9/1998 ,
That is date 21/9/1998. Time: 12.00-14.59

Best time is Date: 21/9/1998. Time: 12.00-12.59
Appointment being made with tom, eddy, joe
on Date: 21/9/1998. Time: 12.00-12.59
```

Figure 6: Free time between Duenpen, Tom, Eddy and Joe.

```
Date: Wed, 2 Sep 1998 16:40:48 -0700
From: duenpen (Duenpen Santhititham)
To: duenpen@pangha.cs.psu.ac.th
Subject: No free time

 Free time of duenpen
 and,tom,eddy,joe
 during date: 20/9/1998 to 21/9/1998. Time : 12.00-14.59
 ******* No matching free time *******
```

Figure 7: No free time between Duenpen, Tom, Eddy, and Joe.

Figure 6 shows the e-mail message from fta which reports that it has found two free time slots suitable for Duenpen, Tom, Eddy, and Joe. It utilises its "preference" rules (see below) to suggest a single best time. Figure 7 is the output from fta after Duenpen had filled her free time slots between noon and 2.59pm on the 20th and 21st. This meant that fta could not find a free time common to all the participants, and so reported failure.

The free time for N diaries is obtained by finding the common free time incrementally between two, three, four, etc diaries. The free time between any two diaries (called D1 and D2) is determined by user-supplied near/2 clauses of the form:

```
near( <a time period in a D1 diary/3 fact>,
      <a time period in a D2 diary/3 fact>) :-
  /* extra conditions */ .
```

The intention of near/2 is to define the meaning of "near" free times between the two diaries. When two free times in the diaries are near to each other (e.g. when both diaries have the same time period free), then that is reported to the user. Two examples:

```
near(TimePeriod, TimePeriod) :-
  TimePeriod = time(_, EndSlot, _), EndSlot <= 11.
near(time(SS1, ES1, Date), time(SS2, ES2, Date)) :-
  (SS1 <= SS2, SS2 <= ES1) ; (SS2 <= SS1, SS1 <= ES2).
```

The first rule specifies that a "near" free time occurs in both diaries when they contain the same free time period (TimePeriod), and that time slot ends before or at 10:59 am. The second rule defines a 'near' free time as being when two time periods in the diaries overlap. There are two possible cases, which are shown in Figure 8.

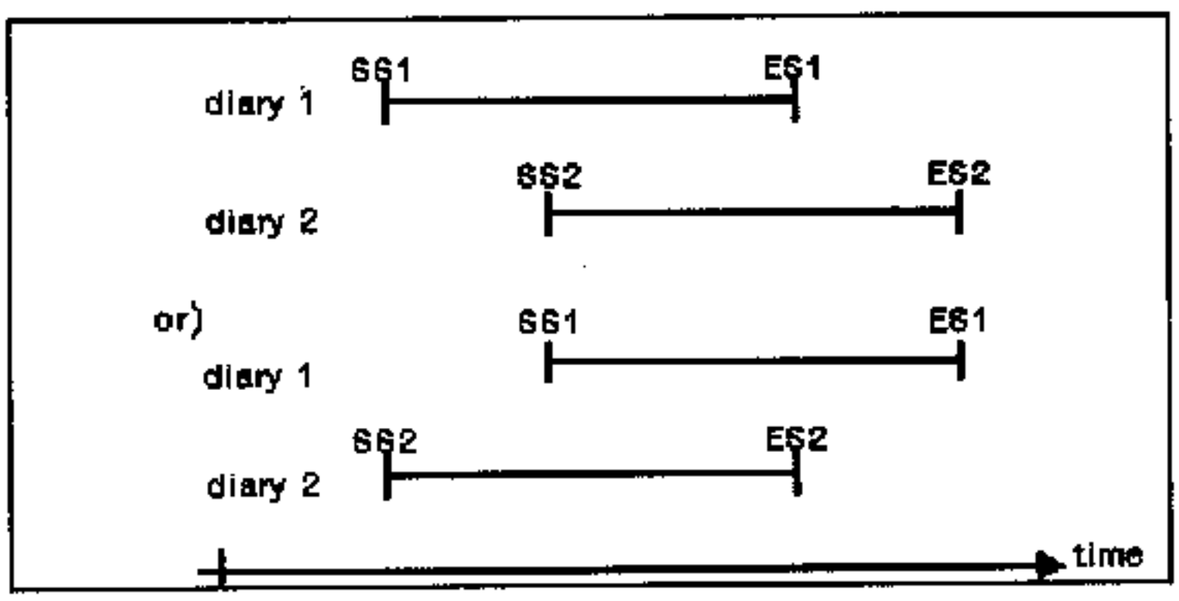

Figure 8: Overlapping free times in two diaries.

The order of the near/2 facts is important, due to the evaluation strategy of Prolog. More precise notions of nearness should be placed before more vague ones so that the system will report those times first.

The incremental construction of free times for multiple diaries frequently returns a number of choices. For example, the four individuals in the earlier call to fta (duenpen, tom, eddy, and joe) shown in Figure 6 had two possible free time slots. An ordering is placed on these by the user writing prefer/2 facts, which have the form:

```
prefer( <time period 1>, <time period 2>) :-
                    /* extra conditions */
```

A prefer/2 clause defines an ordering which prefers the first time period over the second. Four examples of prefer/2 clauses:

```
prefer(time(_, _, date(Day1, M, Y)), time(_, _, date(Day2, M, Y)))
:-
  Day1 =< Day2.
prefer(time(BT1, ET1, _), time(BT2, ET2, _)) :-
  (ET1 - BT1) >= (ET2 - BT2).
prefer(time(_, _, Date), _) :-
  day_of_week(Date, DayName),
  (DayName == saturday ; DayName == sunday).
prefer(time(BT, ET, _), _) :-
  BT >= 17, ET =< 20.
```

The first rule states that earlier days in the same month are better. The second rule prefers longer free time periods. The third rule prefers Saturdays or Sundays, and the fourth rule prefers free times between 5pm and 7.59pm. day_of_week/2 is a built-in for converting a date into its corresponding day name.

A score is kept of the number of prefer/2 rules which apply to each free time. The agent can then e-mail the series of free times to the user and suggest a "best" one, as was done in the e-mail in Figure 6. If the agent finds only one free time, it can make the appointment itself. An example of this is shown in Figure 9.

```
Date: Wed, 2 Sep 1998 16:35:33 -0700
From: duenpen (Duenpen Santhititham)
To: duenpen@pangha.cs.psu.ac.th
Subject:  <near time>

Free time of duenpen
and ,tom,eddy,joe
during date: 20/9/1998 to 21/9/1998 ,
That is date 21/9/1998. Time: 12.00-14.59

Best time is Date: 21/9/1998. Time: 12.00-12.59
Appointment being made with tom, eddy, joe
on Date: 21/9/1998. Time: 12.00-12.59
```

Figure 9: Making an appointment.

The users are notified of the appointment addition by the diary system itself, and so fta does not need to send its own e-mail messages when it makes an appointment.

4. Conclusions

We began this paper with a safe-bet prediction: that the future will be very different from the present. However, the seismic nature of the changes ahead are only dimly starting to be understood. The Web is the beginning, the first seedlings of a new model of computing where business will be seemlessly networked across the globe, where data will be inherently more sophisticated and intertwined, and where our activities will be augmented (and frequently supplanted) by our agent helpers. We have investigated a small corner of the groupware revolution to come.

Our Web-based diary system is a test-bed for the use of logic programming in three related areas: the representation of highly structured server-side information as deductive databases, the encoding of complex client-side queries over those databases using logic programming techniques such as backtracking and unification, and the formulation of agent heuristics ("rules of thumb") as logical relations. Our system, although still at a prototype stage, highlights the benefits of using logic programming in these ways.

The databases, as represented with diary/3, reserved/1, and invisible/2, are concise while retaining expressiveness. diary/3 contains the diary meeting and appointment details, reserved/1 extends the diary with the notion of "reserved" times, while invisible/2 controls the user's access to a diary.

Very complex queries can be readily devised, and have proven to be useful for searching for data, restricting the user's view of the database, and for deleting ranges of information. An oft-stated problem is that such queries are too complex for a novice, or even average, user to write. We have addressed this with a web forms-based interface, and the automatic generation of queries.

The heuristics utilized by the agents, which are encoded with the changed/2, near/2 and prefer/2 predicates, give the user a great deal of leverage for encoding monitoring behavior. However, the clauses themselves are easy to write and understand. This is partly due to the availability of pattern matching and inference inherent in the logic programming paradigm, but is also helped by the design of the agents which hide the full complexity of the queries posed against the databases.

Acknowledgements

The authors are very grateful to Dr. Andrew Davison for his technical suggestions and support, and for carefully reading this paper and making corrections.

References

Apache Group. *Apache Web Server*,http://www.apache.org/_ABOUT_APACHE.html (Retrieved September 1, 1998), 1998.

Berners-Lee, T., Fielding, R., and Frystyk, H. *HyperText Transfer Protocol (HTTP/1.0) Specification*, RFC 1945, 1996.

Blue Cannon Software. *Calendar Wise*, http://www.softwarelabs.com/business/_business126.htm (Retrieved September 1, 1998), 1996.

Bratko, I. *Prolog Programming for Artificial Intelligence*, 2nd ed., Addison-Wesley, 1990.

Cabeza, D., Hermenegildo, M., and Varma, S. *The PiLLoW/CIAO Library for Internet/ WWW Programming*, http://www.clip.dia.fi.upm.es/ miscdocs/pillow/ pillow.html (Retrieved September 1, 1998), 1997.

EuroSoft. *Calendar Magic*, http://www.focusmm.co.uk/products/productivity/ calendar.html (Retrieved September 1, 1998), 1997.

Jirachiefpattana, A., Santhititham, D., Davison, A., and Jirachiefpattana, W. "Logic Agents for Deductive Diary Database on the Web," *Proceedings of the 1999 National Computer Science and Engineering Conference (NCSEC'99)*, Bangkok, Thailand, 1999, pp. 17-23.

Lotus Corp. *Lotus Organizer Web Calendar*, http://www.lotus.com/home.nsf/tabs/ calendar (Retrieved September 1, 1998), 1997.

McCabe, F.G. "April: An Agent Programming Language for the Internet," *Tutorial at the Symposium on Industrial Applications of Prolog (INAP'96)*, Tokyo, Japan, October 1996, pp. 25-34.

Reynolds, M.C., and Wooldridge, A. *Special Edition using JavaScript*, QUE, 1996.

Schlosberg, J. "Where is the Elusive Groupware Payoff ?," *Open Computing*, Volume 12, Number 9, 1995, pp. 31-36.

Sheridan Software. *Calendar Widgets*, http://www.shersoft.com/products/calendar/ calgen.htm (Retrieved September 1, 1998), 1997.

Sterling, L., and Shapiro, E. *The Art of Prolog*, 2nd ed., MIT Publishing, 1986.

Sukpan, A., Santhititham, D., Jirachiefpattana, A., and Davison, A. "A New Time Scheduling Application on the Web Using Logic Programming and Agents," *Proceedings of INET'98 (CD-ROM version)*, Geneva, Switzerland, 1998.

Tarau, P. *BinProlog 5.75*, http://clement.info.umoncton.ca/~tarau (Retrieved September 1, 1998), 1997.

Wada, Y., Kawamura, A., McCabe, F.G., Shiouchi, M., Teramoto, Y., and Takada, Y. "An Agent Oriented Schedule Management System: IntelliDiary," *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, London, UK, 1996, pp. 655-667.