Chapter 1

Agent-Based Intelligent Information Dissemination in Dynamically Changing Environments

H. Sevay, **C. Tsatsoulis**
Department of Electrical Engineering and Computer Science
Information and Telecommunication Technology Center
University of Kansas
Lawrence, KS 66045 U.S.A.
{hsevay,tsatsoul}@ittc.ukans.edu

This chapter presents an agent-based approach to intelligent information dissemination in dynamic environments. We will describe a prototype agent-based system, called the Anticipator, which enables an information dissemination system to adapt to changing information needs of users and provide the users with critical information relevant to those dynamic changes. The Anticipator models the information needs of each user with a dynamically adaptable profile that describes the current information needs of that user and how to adapt those information needs according to the changes in the environment. The Anticipator employs two types of agents, Profile Agents and Event Monitoring Agents. Profile Agents manage runtime user profiles. Event Monitoring Agents watch for interesting changes in the environment and report such changes to the Profile Agents such that the content and the frequency of the information sent to users can be adapted dynamically.

1 Introduction

Information dissemination via the "pull" technology defines much of today's information user's activity. Gathering information using web search engines is one very common example of this approach. However, the "pull" technology assumes that the individual user knows about

what data sources are available, how frequently to search these data sources for appropriate information or potential updates as well as how to further filter the information gathered from these data sources after all the processing is done by the underlying information gathering tools. Therefore information “pull” can be considered a form of exploration through an information space, and, as such, it is not a reactive or an anticipatory mechanism for information collection. Moreover, in “pull” technology, there is a heavy burden on the user to collect the needed information. To have the most appropriate information in a dynamic setting, users need to have access to all necessary data sources to retrieve the needed information and monitor the changes in data to detect interesting events, and then link these changes to their own information needs continually. As a result of this burden on the user, not all types of information needs can be satisfactorily met via information “pull”.

The capability to anticipate the information needs of users is critical for building information dissemination systems that operate in dynamic environments. As the environment changes, the information needs of users may change dynamically. Therefore, information dissemination systems need to adapt to these changes to provide the most appropriate information to their users at any given time by reacting to interesting changes in the world and, whenever possible, by anticipating future information needs based on the current state of the world.

Contrary to information “pull”, in information “push” the burden is on the system for delivering the needed information to the user at the appropriate time. The user is assumed to be a direct consumer of information, and the information is assumed to be delivered automatically, without the user being involved in the information collection task. In this second approach, it is, therefore, the function of the system to know what information will be needed by a given user at what time, how frequently to deliver this information, and, most importantly, how to adapt to the dynamic changes in the environment as these changes lead to changing information needs. Therefore a mechanism for anticipating user information needs fits naturally into the “push” paradigm.

In this chapter, we present an agent-based approach to intelligent information dissemination in dynamic environments. Particularly we

focus on a prototype agent-based system, called the Anticipator, which enables an information dissemination system to adapt to changing information needs of users and provide users with critical information relevant to those dynamic changes. In the Anticipator, the information needs of each user are modeled using a dynamically adaptable *profile* that describes the current information needs of that user and how those information needs are to be adapted according to the changes in the environment.

The Anticipator employs two types of agents, a set of Profile Agents (PAs) and a set of Event Monitoring Agents (EMAs) to adapt both the content and frequency of the information sent to the users. An Event Monitoring Agent watches for an event that a user is interested in, and it reports the occurrence of this event to a particular Profile Agent. A Profile Agent maintains a parameterized model of the changing information needs of a specific user, and it adapts this model (i.e., profile) based on the dynamic changes detected by individual Event Monitoring Agents.

For each new user that needs to be added to the system, the Anticipator instantiates a profile that applies to that user. The primary components of a user profile are information requests that can be active or inactive at any given time, and event rules that describe how that profile needs to be adapted when interesting changes happen in the environment. A profile is a predefined generic template for a particular type of user, and its instantiation is based on the latest known state of the world. From this point on until the user leaves the system, that profile lives as a persistent object continually adapting the information sent to the user for whom it was instantiated. Each user profile changes dynamically as a result of events of interest in the environment that may warrant specific changes or an entire *reinstantiation* of that profile.

When a profile is instantiated for a user, there are two types of Event Monitoring Agents that are created. The Anticipator creates an EMA for each information request that applies to the current state of the user and the environment. It also creates EMAs for each event rule in the profile. Both the information requests and the event rules are instantiated based on the latest known state of the world.

There are two types of Event Monitoring Agents in the Anticipator: (1)

time-driven EMAs that periodically take an action such as submitting information requests on behalf of users; (2) data-driven EMAs that watch for interesting changes in the environment and subsequently notify the appropriate Profile Agents so that the user profiles being managed by those Profile Agents can be adapted to the current situation.

Our discussion in this chapter will focus on the Anticipator that operates as an intelligent component of an overall adaptive information dissemination system. We will also discuss how dynamically adaptable user profiles persistently adapt the information needs of each user as changes occur in the environment.

The remaining sections in this chapter are organized as follows: Section 2 provides an overview of the Anticipator system, and Section 3 introduces the architecture of the Anticipator. Section 4 discusses how Profile Agents operate within the Anticipator framework. Section 5 describes the profile instantiation and reinstantiation processes. In Section 6, we describe the details of the profile information request representation. Section 7 covers the operation of the Event Monitoring Agents. Section 8 presents an example that demonstrates the fundamental operations of the Anticipator. In Section 9, we discuss other research work closely related to this study. In Section 10, we briefly discuss future work. Finally, in Section 11, we present conclusions. Please note that, throughout this chapter, we have edited the format of most of the knowledge representations and written them in pseudo form to make them easier to understand.

2 Overview of the Anticipator

The problem we are addressing is delivering information to users at the *right* time and adapting the content and frequency of the information to be delivered depending on the changes in the environment, so that the user receives only pertinent information.

To accomplish this task, an information dissemination system must have access to both the state of the user and the state of the rest of the environment that pertains to that user. To be able to adapt the information sent to the user dynamically, the information dissemination system has to

monitor particular aspects of the world that are of interest to each given user. Therefore, first we need a mechanism for describing the information needs of a given user under changing conditions. Second, we need a mechanism for interacting with the data sources that can provide the user with the needed information. Third, we need an event monitoring mechanism to watch for interesting changes in the environment and alert the system of those changes, so that the information needs of the user can be adapted dynamically according to the profile.

In the Anticipator, user profiles are managed and adapted by Profile Agents that stay active in the system from the time a user is entered into the system until that user leaves the system. Each Profile Agent also has a number of Event Monitoring Agents (EMAs) associated with it. The EMAs are dynamically created based on the event descriptions in a profile. Some EMAs are responsible for periodically sending information requests to data sources so that the user receives the needed information. Some EMAs are responsible for monitoring the state of the world and alerting the Profile Agents they are linked to of these changes. The state of the world is assumed to be reflected at the data sources, and, hence, the monitoring of the state of the world happens by periodically accessing the data stored at these locations.

Because of this continual feedback that Event Monitoring Agents provide to the Profile Agents, one can view the operation of the Anticipator as a closed feedback loop. The goal of the system is to *stabilize* the output of the system as the environment changes, hence reducing the *error* between the desired output and the actual output. The output of the system is the information that is being provided to the user, and the desired output is the information that should be provided to the user under the current conditions. Therefore each Profile Agent tries to adapt the information content and the frequency with which that information is sent to the user so that user receives needed information at appropriate update intervals. Alternatively, the behavior of the Anticipator can be represented as a state machine, with the important distinction that only the current state actively runs in the system at any given time, and the remaining states lay dormant implicitly in the user profile representation without any active component in memory.

3 The Anticipator In An Information Dissemination System

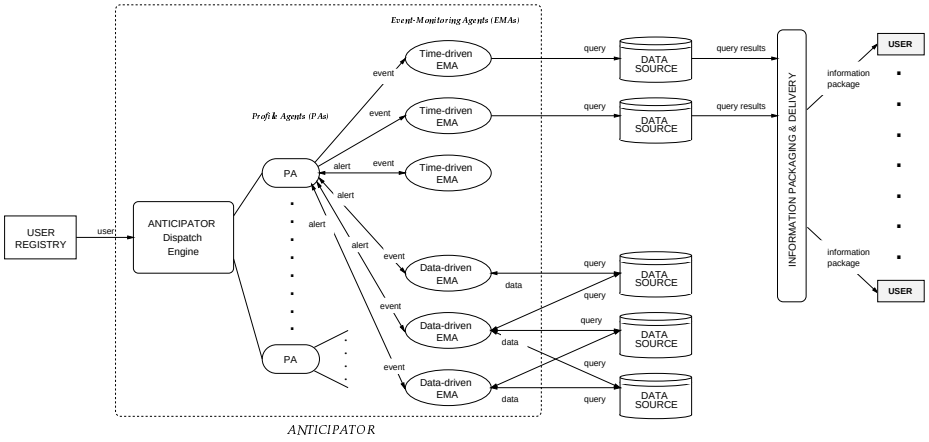

Figure 1. Anticipator Architecture as part of a larger information dissemination system

Figure 1 shows how the Anticipator fits as a component in the overall architecture of an information dissemination system. The User Registry adds and removes users from the system. The Anticipator Dispatch Engine creates Profile Agents for each new user, manages working Profile Agents, and also removes Profile Agents when users leave the system. The Data Sources provide the needed information that users require by responding to information requests that time-driven EMAs submit to them. We assume that the Data Sources continually warehouse data about the state of the world (e.g., weather conditions, medical records, locations of mobile vehicles, etc.) Data-driven EMAs watch for interesting events in the state of the world, and, when they detect such events, they immediately alert the Profile Agents they are associated with of these events. In response to these alert messages, Profile Agents dynamically modify the information content and update rate flowing to the users. As Data Sources produce data to be sent to the users, the Information Packaging and Delivery component stores the information intended for each user into an information package and sends that package to the appropriate user.

4 Profile Agents

The task of a Profile Agent is to manage the user profile of a specific user. Each user profile is initially derived from a generic profile template that is designed for a certain type of user (e.g., stock broker, military strategist, economist, pregnant woman, computer user, etc.). Upon instantiation, a user profile becomes an active object in the system. The Profile Agent with the user profile it is managing and all of the Event Monitoring Agents associated with it live as a set of distributed elements in the Anticipator but work as a collective entity.

Figure 2 shows the general format of a user profile, which is comprised of three components:

- Profile variable definitions
- Information requests
- Event Rules

Profile <profile name>

Profile variable definitions:
profile_variable$_1$ $\Leftarrow$ *instantiation_rule*$_1$
...
profile_variable$_N$ $\Leftarrow$ *instantiation_rule*$_N$

Information Requests:
information_request$_1$ ... *information_request*$_N$

Event rules:
event_rule$_1$ ... *event_rule*$_N$

Figure 2. General format of a user profile

4.1 Profile variable definitions

Profile variables are used to parameterize different aspects of a given user profile. Their values are assigned by special rules, called instantiation rules (described next), which are referenced by each profile variable definition. There can be as many profile variables as needed in different parts of a profile.

4.2 Instantiation rules

Each *instantiation rule* instantiates (or reinstantiates, as the case may be) the value of a specific profile variable. An instantiation rule decides on the value of a profile variable depending on the particular aspects of the current state of the world. Hence the value assigned to a profile variable may differ depending on the current conditions.

4.3 Information requests

Information requests describe the specific information that needs to be retrieved from available data sources on behalf of users. They are parameterized to allow dynamic adaptation of the content of the information that users will receive. For example, suppose that a user needs temperature information. An information request to ask for this information can be written such that location can be a parameter so that the temperature information request can be customized to any location.

In the Anticipator, an information request can be active or inactive at any given time. Active information requests are the ones that will ask for the currently needed information for a user. Inactive information requests are the ones that do not apply to the current state of the world; but, upon changes in the environment, inactive information requests can potentially be activated. Therefore the state of the world – including that of the user – drive which information requests must be activated and which ones must be deactivated at any given time.

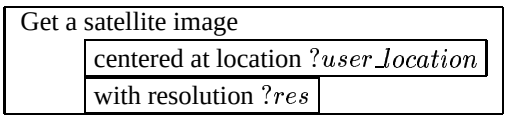

Figure 3. An example information request for obtaining the satellite image of the area centered at user's location with a certain resolution

For example, the information request given in Figure 3 obtains a satellite image of the region centered at user's current position with a certain resolution. Let us suppose that the resolution of the image is adjusted based on the instantiation rule given in Figure 4. This example rule depends on the terrain type of the current user location, and the terrain information is retrieved from the data source that stores and updates

information about the user. If, for instance, the user is in a densely forested area, the resolution of the image will be set to *high*.

```
Rule image-resolution
   if the user is currently in a desert then
      ?resolution ← low
   else if the user is currently in an open area then
      ?resolution ← medium
   else if the user is currently in a densely forested area then
      ?resolution ← high
   else
      ?resolution ← normal
```

Figure 4. Example instantiation rule that can assign different values to a profile variable based on the current state of the world

4.4 Event Rules

An *event* is a condition that may be true of a set of data at a given point in time. Each event rule describes an event that is known to be of interest to the type of the user for which the profile was designed for, and how the current state of a user profile needs to be modified when that event condition is satisfied. Specifically, each event rule specifies a set of actions to be executed when the event condition of that rule is satisfied.

For this mechanism to work, a Profile Agent creates a data-driven Event Monitoring Agent for each event rule in a profile. The EMA then starts monitoring the condition of the corresponding event rule. When an EMA detects that its event condition is satisfied, it alerts the Profile Agent, and the Profile Agent takes the actions specified in the corresponding event rule.

Figure 5 shows the general format of an event rule, which, in structure, is very similar to the event-condition-action (ECA) rules in active databases, which also provide for any number of actions that can be executed upon the satisfaction of the event condition.

5 Profile Instantiation

In the Anticipator, profile instantiation is the very first step that takes place when a new user is being added to the system. Profile instantiation

E: A logical statement
A_i: An action

if (E) **then**
  execute_action(A_1);
  execute_action(A_2);
  . . .
  execute_action(A_N);
end if

Figure 5. General format of an event rule

happens in two stages. For each profile template to become an active entity in the system, the profile variables in that template must first be instantiated by their instantiation rules. Next the active information requests in the profile are instantiated now that all of the profile variables have been assigned ground values. Then, for each new user, the Anticipator creates a Profile Agent. Given a profile template for each user, the Profile Agent starts managing that profile after instantiating the profile template and creating the Event Monitoring Agents. Algorithm 1 gives the algorithm for the operation of a Profile Agent, given a profile template and a unique user ID for which that template is to be instantiated.

To instantiate the profile variables in a user profile, a Profile Agent runs the instantiation rule corresponding to that profile variable. Each instantiation rule may internally require information about the state of the world to assign a specific value to a profile variable. To get the latest state of the world, a Profile Agent asks for the values of needed (state) variables from data sources.

Once the profile variables and information requests are fully instantiated based on the latest known state of the world, then the Anticipator creates the Profile Agent that will manage this newly created working profile. This includes the initial spawning of the Event Monitoring Agents. Moreover, all further runtime modifications to this profile will be carried out by this Profile Agent.

Depending on the profile design, a set of information requests may be initially marked active. Upon the completion of the instantiation

ALGORITHM 1 : Operate Profile Agent (P, U)

P: profile template to be used
U: unique ID of a user
V_i: a profile variable
R_i: instantiation rule associated with V_i
Q_i: an active information request in the profile
E_i: an event rule in the profile
C_i: event condition of E_i
M_i: a currently running EMA created based on E_i

```
// Instantiate all profile variables
for all V_i in P do
   V_i ← Execute rule R_i;
end for

// Instantiate all active information requests and spawn time-driven EMAs
for all Q_i in P do
   Instantiate information request Q_i;
   Spawn EMA based on Q_i;
end for

// Spawn data-driven/time-driven EMAs for all event rules
for all E_i in P do
   Spawn EMA based on E_i;
end for

// Watch for event alerts and adapt the profile
while (TRUE) do
   for all M_i do
      s ← Get the current status of EMA M_i;

      if (s is an event alert, i.e., C_i is TRUE) then
         Execute all actions in E_i;
      end if
   end for
end while
```

process, a Profile Agent will spawn a time-driven Event Monitoring Agent for each of these active information requests. Then, periodically, each of these EMAs will automatically send an information request to the appropriate data sources. Then the result of these information requests will be sent by the data sources to the user.

A Profile Agent will also spawn as many data-driven EMAs as the number of event rules in the profile it is managing. Each of these EMAs

will monitor the state of the world by periodically asking appropriate data sources for data that they need to check on their event condition.

We must note that it may be critical that the profile variables be reinstantiated every time a data-driven EMA alerts a Profile Agent of an event so that the user profile is modified according to the most current state of the world, and hence the user can receive the information that would be most valuable to have. In the Anticipator, the implementation of this behavior depends on the profile design. In addition, the definition of which events are interesting for a user and how the information content sent to a user needs to be adapted depends on the application, particularly, the event rules in a profile.

6 Parameterized Information Requests

An active information request in a profile is sent to a particular data source (or mediator), which will provide the needed information to the user. In the Anticipator, information requests are described with six attributes. These attributes are *frequency*, *language*, *description*, *status*, *cycle*, and *body*. The *frequency* attribute specifies how often an information request needs to be submitted to the data sources so that a user receives regular information updates; for example, every hour, every ten minutes, once a day, etc. The *language* attribute specifies the language that is used in implementing the actual request that will ask for the needed information. Therefore, it is possible to use multiple query languages within a profile (for example, SQL, Lorel, MSL, etc.). For user-friendliness, each information request also contains a *description* field to allow the profile designer to include an informal description of what information will be requested and/or any other information that would be helpful in indicating the purpose of an information request in a profile. The *status* attribute specifies whether an information request is active or inactive at a given time. Hence this attribute directly controls which information requests the Anticipator will send to data sources. The *cycle* attribute, on the other hand, specifies how many times an information request should be submitted to the data sources; for example, a cycle value of 2 says that a given information request must be submitted to the data sources twice with a period specified by the *frequency* attribute, and, upon the second submission, the information request must

be deactivated. The deactivation of an information request means that the Event Monitoring Agent responsible for submitting that request to the data sources will automatically remove itself from the system. For persistent information requests, the *cycle* value must be set to a special value, *infinite*. Finally the *body* attribute contains the formal description of the content of the information that will be requested from data sources.

```
Information-Request <name>
   body        ← <Formal specification of the request>
   language    ← <Language of the formal specification>
   frequency   ← <value> <unit>
   status      ← [active|inactive]
   cycle       ← [1-N|infinite]
   description ← "A textual description of the request"
```

Figure 6. General format of a profile information request

The general format of a profile information request is given in Figure 6.

```
Information-Request Temperature-in-city
   body        ← Get the temperature in city ?city
   frequency   ← ?mins minutes
   status      ← active
   cycle       ← infinite
```

Figure 7. Example information request that retrieves the temperature in a specified city every $?mins$ minutes

Figure 7 shows an example information request (written in pseudo code) that obtains the temperature in a given city. This active information request will run indefinitely since its *cycle* attribute is set to *infinite*, and it will update the temperature every `?min` minutes. Since the location is specified using a variable (`?city`), this information request can be customized to the current location of a user during the profile instantiation/reinstantiation process.

Figure 8 shows an initially inactive information request that retrieves information about gas stations within 5 miles of the current location of a vehicle. This information request could be used in providing the driver of a vehicle with a list of closest gas stations in the case the fuel level in that vehicle falls below a certain level.

Since the information sent to the user will be updated every 5 minutes in this information request, it may be necessary that the vehicle position

```
Information-Request Nearest-gas-stations
   body        ← Get the list of all gas stations within
                 5 mile radius of a vehicle's current
                 location ?loc
   frequency   ← 5 minutes
   status      ← inactive
   cycle       ← 1
```

Figure 8. Example inactive information request that retrieves information about gas stations closest to a vehicle's location

information (*?loc*) is also updated at least at the same rate. In the Anticipator, this can be done by defining an event rule in the user profile that will automatically reinstantiate this information request every so often.

7 Event Monitoring Agents

Event monitoring is a critical component of the task of anticipating user information needs. The Anticipator uses two types of EMAs. Time-driven EMAs are responsible for taking periodic actions such as submitting information requests to data sources according to the frequency parameter of each information request, and data-driven EMAs are responsible for detecting whether changes in data sources satisfy the event conditions they are checking.

7.1 Event Types

In the Anticipator representation, we divide the event types into two as time-driven events and data-driven events. Similar categorizations have been used elsewhere in the literature (Decker *et al.* 1996).

7.1.1 Time-driven events

In the Anticipator, time-driven events depend only upon the passage of time. If the required update period has elapsed since the last occurrence of the same event, then the time-driven event is said to have occurred. In the Anticipator user profile representation, the frequency attribute of an information request specifies the period of an implicit time-driven event. For example, if the frequency value of a information request is 15

seconds, then, every 15 seconds, the event condition of the time-driven event created for updating this information request will be true.

There is also an explicitly represented time-driven event type that is akin to a periodic “ping” or a (unix) signal. These events are defined to alert Profile Agents to take specific actions regularly. For example, if we are interested in the position of a moving vehicle to decide on what kind of information to send to the driver of that vehicle, then we will have to make sure that the Profile Agent associated with that moving vehicle gets periodic updates on the position of that vehicle. One method of implementing this functionality within the Anticipator architecture is to explicitly reinstantiate information requests that depend on the vehicle movement by having a time-driven EMA alert the appropriate Profile Agent to execute reinstantiation actions on the working profile of the vehicle driver.

7.1.2 Data-driven events

Data-driven events, on the other hand, are dependent not on the passage of time but on the satisfaction of a logical condition involving state data. Therefore, data-driven events can be conditioned on the current values of state variables that reflect the current state of the world, trends or averages involving those variables besides other types of measurements.

We can further divide data-driven events into two subcategories as *instantaneous* and *continuous*. Instantaneous data-driven events are those with an event condition that depends on the latest value of a set of state variables. Continuous data-driven events are then those that require storage of previous value(s) in some form. For example, an event condition that checks on a running average or a trend would be classified as continuous, since prior state information would have to be stored.

The categorization of events into time-driven (periodic) and data-driven (non-periodic) have been used exclusively in active database research (Hanson and Noronha 1999). The categorization of events into time-driven and data-driven is a useful one, especially because the handling of each one requires a totally distinct implementation, with the time-driven events being simpler to implement, since they only depend on the passage of time. Data-driven events, on the other hand, can be much more complex, since they need to ask data sources for state data to be

able to check the truth value of their event condition.

7.2 Time-driven Event Monitoring Agents

The description of a time-driven EMA can be based on an information request or event rule. Since the Anticipator creates a time-driven EMA for each active information request based on the frequency parameter in that request, the Anticipator does not have an explicit event representation for EMAs that are based on information requests. The only action that such EMAs take is to submit their information requests to data sources. However, remaining types of time-driven EMAs require explicit event descriptions to allow periodic execution of actions other than submitting information requests to data sources. For example, this type of time-driven EMAs can be used for pinging the appropriate Profile Agents for periodic reinstantiation of information requests.

ALGORITHM 2 : Time-Driven Event Monitor Operation

T: information request update period (frequency)
Q: instantiated information request to be sent
D: data source that can produce results for Q
U: unique ID of a user

while (TRUE) **do**
  Send Q to D and have the results forwarded to U;
  Wait T;
end while

A time-driven EMA has a simpler working structure than a data-driven EMA. When a time-driven EMA is spawned by a Profile Agent, the frequency of that EMA attribute has a specific numerical value that may have been assigned during profile instantiation or specified as a constant at the time of the corresponding profile template design. At regular intervals specified by this frequency value, a time-driven EMA sends its already instantiated information request to the appropriate data source. How we choose which data source to send an information request to is largely a question of mediation, and this issue of mediation is beyond the scope of this chapter. In our model, we assume that all information requests are sent to addresses designated *a priori*, and that mediators decide to which specific data sources to forward information requests. Moreover, since each request contains a description of a user's

destination address, the data source responding to that request can send its results directly to that user.

7.3 Data-driven Event Monitoring Agents

Each data-driven EMA in the Anticipator is created based on an event rule in a given profile. The condition of that event rule becomes the *event* that the EMA checks for. An event is usually made up of conditions that require that certain aspects of the current state of the world be known. Therefore, an EMA asks the appropriate data sources to retrieve

ALGORITHM 3 : Data-Driven Event Monitor Operation

T: state variable update period
C: event condition to check for
A: Profile Agent that spawned this event monitor
S_i: a state variable

while (TRUE) **do**
  for all S_i used in C **do**
    $S_i \leftarrow$ Retrieve the value of S_i from a data source;
  end for

  if (C is TRUE) **then**
    Send an *event alert* to A that C is now TRUE;
  end if

  Wait T;
end while

the values of variables, called *state variables* that are used in its event condition. This activity is performed periodically. After each update of state variables, the EMA checks whether its event condition is satisfied. If so, then it means that an event has occurred. Then the EMA automatically alerts the Profile Agent that spawned this EMA. In response to an event alert, the Profile Agent takes the actions prescribed in the action section of the event rule in the profile. If, on the other hand, the event condition is not satisfied, the cycle of information updating and condition checking continues until the EMA is removed from the system. Algorithm 3 summarizes how a data-driven EMA operates.

The actions that the Profile Agent takes in response to an event alert are used to adapt the profile to the changes that were deemed of interest

to the user. For example, these actions can direct the Profile Agent to activate currently inactive information requests, deactivate currently active information requests, or reinstantiate a set of information requests. It is also possible that some events may require that the entire profile be reinstantiated. A reinstantiation causes all of the existing EMAs to be removed from the system. Also the profile variables in the profile are reinstantiated. After the removal of existing EMAs, the Profile Agent once again creates a time-driven EMA for each active information request, and a data-driven EMA for each event rule. Algorithm 4 gives the algorithm for this profile reinstantiation process.

ALGORITHM 4 : Reinstantiate Profile (P, U)

D_i: a data-driven EMA
T_i: a time-driven EMA
U : unique ID of a user
P : profile template to be used

$S_d \leftarrow$ Set of currently active data-driven EMAs
$S_t \leftarrow$ Set of currently active time-driven EMAs

for all D_i in S_d **do**
  Remove EMA D_i;
end for

for all T_i in S_t **do**
  Remove EMA T_i;
end for

Call *Algorithm 1*(P, U); // See page 11

8 An Example

In this section, we will present an example scenario to demonstrate how the Anticipator handles dynamic changes in an environment. This example is in the domain of battlefield awareness. We will start with the profile of a military unit in active duty. The profile is given in Figure 9.

Suppose that the military unit has a unique ID of $acomp001$ and that this unit has already been added as a new user to the system. In the profile template for this user, there are three information requests and a single event rule. The first information request is the only request that is initially

```
Profile military-unit
   Profile variable definitions:
      ?lat  ⇐ Rule_A
      ?long ⇐ Rule_B
      ?res  ⇐ Rule_C

   Information Requests:
      Information-Request friendly-info
         body        ← Get a list of friendly units within
                       25 miles radius of user's current
                       location (?lat, ?long)
         frequency   ← 10 minutes
         status      ← active
         cycle       ← infinite

      Information-Request satellite-image
         body        ← Get a satellite image of the area
                       within 25 miles radius of user's current
                       location (?lat, ?long) with resolution ?res
         frequency   ← 15 minutes
         status      ← inactive
         cycle       ← infinite

      Information-Request hostile-info
         body        ← Get a list of hostile units within
                       25 miles radius of user's current
                       location (?lat, ?long)
         frequency   ← 5 minutes
         status      ← inactive
         cycle       ← infinite

   Event rules:
      Event-Rule enemy-close-by
         if there any enemy units within 25 miles then
            Reinstantiate this profile;
            Activate information request satellite-image;
            Activate information request hostile-info;
         endif
```

Figure 9. Example profile for a military unit

activated, since its status field is marked *active*. Since the remaining two requests are inactive, the Anticipator will not create any time-driven EMAs for them at the time of the first instantiation of this profile. There is only one event rule in this profile (`enemy-close-by`) that can potentially activate both of these initially inactive information requests.

Figure 10 shows an example implementation of one of the instantiation rules used in the military unit profile. Instantiation rule `Rule_A` first retrieves a few values about a unit from data sources. Then, depending on the current status of the unit, it subtracts an offset value from the unit's

```
Instantiation-Rule Rule_A
   unitID ← Get unit's unique ID;
   lat    ← Get current latitude of unit unitID;
   status ← Get current status of unit unitID;

   if unit unitID is at high alert then
      offset ← 20 miles;
   else if unit unitID is at medium alert then
      offset ← 30 miles;
   else if unit unitID is at low alert then
      offset ← 40 miles;
   else
      offset ← 50 miles;
   end if

   Instantiate value (lat - offset);
```

Figure 10. Example instantiation rule

current latitude. This value becomes the instantiated value for profile variable `?lat1`.

```
Information-Request friendly-info
   body      ← Get all friendly units within a radius of
               25 miles around unit's current
               location (?lat, ?long)
   frequency ← 10 minutes
   status    ← active
   cycle     ← infinite
```

Figure 11. Information request `friendly-info` after an instantiation

After the military unit profile is fully instantiated, the profile starts running in the system in its instantiated form, and it is managed by a unique Profile Agent created on behalf of *acomp001*. For example, the only initially active the information request, `friendly-info`, might look as shown in Figure 11 after instantiation. So when the system creates a time-driven EMA for this information request, that newly created time-driven EMAs will automatically start submitting its information request to the appropriate data source every 10 minutes. This information request will then cause that data source to send information about friendly forces in the area to user *acomp001* every 10 minutes. In addition, the system

will also create a data-driven EMA for the single event rule, `enemy-close-by`, to watch whether there are any enemy units approaching.

Let's imagine that a hostile unit moves within 25 miles of the friendly unit. When that happens, the event condition of the event rule `enemy-approaching` will be true. Then the system will execute the actions of that event rule, first reinstantiating the entire working profile of user *acomp001* and then activating previously inactive information requests `satellite-image` and `hostile-info`. Since the first action calls for a complete reinstantiation of the profile, the system will first remove all the existing time-driven and date-driven EMAs associated with user *acomp001*. Then it will generate new versions of those monitors based on the reinstantiated profile. With this, another cycle of Profile Agent operation will start.

9 Related Work

The problem of selecting what information is relevant to what users can be thought of as information filtering (Belkin and Croft 1992). It may also be referred to as selective dissemination of information (SDI) (Foltz and Dumais 1992) when users submit their profiles to a system from which they expect to receive information updates. In the SIFT system, for example, profiles are made up of keywords that users need to submit to an information dissemination system (Yan and Garcia-Molina 1995). SIFT differs from the Anticipator in how it represents, uses, and updates user profiles. First, a profile in the SIFT system is a single query that a user submits to the system to get periodic updates. SIFT supports both the specification of the update period and how many times an information request should be submitted; these actions are specified by the *frequency* and the *cycle* attributes in information requests in the Anticipator. Therefore, since users need to send queries to SIFT to identify what they are interested in, profiles in SIFT are composed of a set of static queries, which can only be changed by the user instead of a higher-level user profile construct as in the Anticipator. In addition, SIFT requires that users interact directly with the system, which the Anticipator aims to abstract away.

Information requests in the Anticipator are predefined in profiles in

parameterized form so that they can be customized to the current state of the world. Moreover, the user need not interact with the underlying information dissemination system to receive information or possess detailed knowledge about how to retrieve information from data sources.

To filter network news, a SIFT user can modify his/her profile by adjusting the values of the precision and recall parameters. The Anticipator, on the other hand, modifies its profiles automatically at runtime as a result of events reported to Profile Agents by data-driven Event Monitoring Agents.

Systems such as SIFT can work well depending on the quality of the retrieval algorithm they employ; however, they do not address the problem of adapting to changing user information needs. On the other hand, active database systems (Paton and Diaz 1999) provide event alerters (triggers) that notify applications of interesting data events (Dittrich *et al.* 1986, Hanson and Noronha 1999). However, an alert mechanism can only let a user know about relevant changes, but it does not fully support a user whose information needs change due to changes in an environment. Especially when we take into account rapidly changing decision environments such as battlefields, critical patient monitoring, or the stock market, it becomes apparent that we need a heavily user-centric approach for dynamic and intelligent information dissemination. We therefore believe that employing user profiles in the proactive manner that we described in this chapter will bring both increased expressibility and adaptability to information dissemination systems.

There is a clear difference between information systems that filter electronic mail, Usenet news, and the World Wide Web documents, and the kinds of information dissemination systems we focused on in this chapter. In information filtering and web browsing, the sources of information are documents. Existing information filtering systems (Papakonstantinou and Vassalos 1998, Armstrong *et al.* 1995, Sorensen and McElligott 1995, Lieberman 1995, Balabanovic 1997) represent documents using different models and they try to match user preferences saved in profiles to these models. Conversely, in the Anticipator, the information requests in profiles directly identify the information that needs to be sent to the users. That is, the Anticipator does not reason

about the contents of the material (text documents, maps, images, etc.) that its requests from data sources on behalf of its users.

Some of the information filtering systems use relevance feedback and information about the actions of users (browsing, deleting files or emails, etc.) to learn and adapt user profiles. For example, the Letizia system (Lieberman 1995) only uses the actions of the users for learning, since the goal of that system is to minimize its interaction with the users. Other systems such as NewT (Sheth 1994) and Amalthaea (Moukas 1998) use genetic algorithms in conjunction with relevance feedback from users to evolve their user profiles such that those profiles are adapted according to changing interests of the users over time. Relevance feedback is also used in (Edwards *et al.* 1997) to learn user profiles based on rule induction and instance-based algorithms. Genetic algorithms are used in (Sorensen and McElligott 1995) to learn user profiles, but the approach is different from that of NewT and Amalthaea.

In recent years, search engines and web spiders have become very popular tools for locating relevant documents on the Internet, but there are numerous domains where the retrieval of very specific and potentially changing information is required. Paton and Diaz classify such systems as open database applications (Paton and Diaz 1999). One important characteristic of these systems is that they operate in dynamically changing domains such as battlefields, patient monitoring, and air traffic control. However, these systems also do not seem to be organized from the perspective of individual users with potentially changing information needs. Thus the Anticipator aims to provide a methodology for an alternatively user-centric approach.

10 Future Work

In its current status, the Anticipator does not support learning, and there is no mechanism for user-level feedback. Since an information dissemination system such as the Anticipator needs to support a wide variety of users, existing profile templates may not always be sufficient. Therefore, it is conceivable to use learning methods to identify new information needs and conditions under which those information needs would be valid.

11 Conclusions

Information systems can be more beneficial to their users if they can dynamically anticipate the needs of their users. To this end, the profile reinstantiation process is critical in providing dynamic adaptability to changes in the environment at the system level. Since the main task of the information consumer will not necessarily be information collection, having systems that automatically provide needed information to users as conditions change can be of great value in our increasingly complex information society.

In this chapter, we demonstrated that knowledge-based modeling of the information needs of users in profiles provides a strong basis for adapting to changing user information needs. The meta-level representation of user information needs that the Anticipator provides enables an information dissemination system to access a wide variety of external systems and technologies from a single interface. The benefit of a single representation system that, for example, enables different query languages for requesting information to be used, is that the generation of new profiles and editing of existing profiles can become transparent to the details of the data sources being employed, especially if we assume that mediators are used to retrieve data from different types of sources.

In the Anticipator, the actions attached to the event conditions in profiles allow dynamic adaptation of working profiles so that the information needs of users are kept consistent with the changing environment, which may frequently cause new information needs to arise. In addition, the event conditions in profiles encode *expertise* about when to send information to particular users and also what information to send. Information systems need such expertise because individual users would be overwhelmed by having to constantly link changes that occur in their environment to new information they may need. Likely they also would not possess the resources required to detect the changes that are relevant to their information needs. An automated system with a more global view of the environment can provide the needed information intelligently allowing the users to focus on their immediate goals.

Acknowledgments

This work was supported in part by DARPA contract F30602-94-C-0192 and by DARPA contract F30602-97-C-0267, both monitored by Rome Laboratories. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Air Force position, policy, or decision, unless so designated by other documentation. Many ideas in this paper were shaped with the help of Jon Dukes- Schlossberg and Yongwon Lee (Lockheed-Martin), Marc Zev and Nancy Lehrer (ISX Corporation), Alok Nigam and Ray Emami (Global InfoTek), and Hector Garcia-Molina and Joachim Hammer (Stanford University).

References

Armstrong, R., Freitag D., Joachims T., and Mitchell, T. (1995), "WebWatcher: A learning apprentice for the World Wide Web," *Proceedings of the 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, pp. 6–12.

Balabanovic, M. (1997), "An adaptive web page recommendation service," *Proceedings of the First International Conference on Autonomous Agents*, pp. 378–385.

Belkin, N.J., and Croft, W.B. (1992), "Information filtering and information retrieval: Two sides of the same coin?" *Communications of the ACM*, vol. 35 (12), pp. 29–38.

Decker, K., Sycara, K., and Williamson, W. (1996), "Intelligent adaptive information agents," *Working Notes of the AAAI-96 Workshop on Intelligent Adaptive Agents, (Technical Report WS-96-04)*, pp. 117–126.

Dittrich, K. R., and Kotz, A. M., and Mülle, J. A. (1986), "An event/trigger mechanism to enforce complex consistency constraints in design databases," *SIGMOD Record*, vol. 15 (3), pp. 22–36.

Edwards, P., Green, C. L., Lockier, P. C., Lukins, T. C. (1997), "Exploiting learning technologies for world wide web agents," *IEE Colloquium on Intelligent World Wide Web Agents, Digest No: 97/118*, pp. 3/1–3/7.

Foltz, P.W., and Dumais, S.T. (1992), "Personalized information delivery: An analysis of information filtering methods," *Communications of the ACM*, vol. 35 (12), pp. 51–60.

Hanson, E.N., and Noronha, L.X. (1999), "Timer-driven database triggers and alerters: Semantics and a challenge," *SIGMOD Record*, vol. 28 (4), pp. 11–16.

Lieberman, H. (1995), "Letizia: An agent that assists web browsing," *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 924–929.

Moukas, A., and Maes, P. (1998), "Amalthaea: An evolving multi-agent information filtering and discovery system for the WWW," *Autonomous Agents and Multi-Agent Systems*, vol. 1 (1), pp. 59–88.

Papakonstantinou, Y., Vassalos, V. (1998), "Query rewriting using semistructured views," *Technical Report*, Database Group, Computer Science Department, Stanford University.

Paton, N.W., and Diaz, O. (1999), "Active database systems," *ACM Computing Surveys*, vol. 31 (1), pp. 63–103.

Sheth, B.D. (1994), "A learning approach to personalized information filtering," *Master's thesis*, MIT Media Lab.

Sorensen, H., and McElligott, M. (1995), "An online news agent," *Intelligent Agents Workshop*, British Computer Society Specialist Group on Expert Systems and Representation and Reasoning Special Interest Group, Oxford, UK.

Yan, T., and Garcia-Molina, H. (1995), "SIFT–A tool for wide-area information dissemination," *Proceedings of the USENIX Winter 1995 Technical Conference*, pp. 177–186.

■