Available online at www.sciencedirect.com
ScienceDirect
Journal of Network and Computer Applications 30 (2007) 445–465
Journal of NETWORK and COMPUTER APPLICATIONS
www.elsevier.com/locate/jnca

# The design and implementation of an agent-based framework for acceptable usage policy monitoring and enforcement

B. Stephen*, L. Petropoulakis

*Department of Electrical and Electronic Engineering, University of Strathclyde, Royal College, 204 West George Street, Glasgow G1 1XW, USA*

Received 11 November 2005; accepted 7 June 2006

**Abstract**

Reliance on the Internet in the workplace means that manually monitoring compliance with an Acceptable Usage Policy (AUP) is impractical given the volumes of data generated. Therefore, for such a system to function effectively, the processing of vast audit trails obtained must be processed by automated means. This paper introduces the incorporation of a novel user-monitoring framework into the domain of software agents for large-scale auditing of Internet use with possible extensions to general network use. It is intended that such an approach would replace current ad-hoc methods such as those based on perusing server logs with a more accurate representation of user activity. The system described herein is an experimental multi-agent one provisionally known as WebEngzilla, which actively monitors and reports on the Web browsing behaviour habits of network users unifying an ambient client monitoring system with a distributed data mining back end.


*Keywords:* Data mining; Collaborative filtering; Software agents; User profiling

## 1. Introduction

Abuse of network resources is becoming an increasingly serious threat to business. With rising numbers of employers having to discipline staff due to violations of their corporate

*Corresponding author. Tel.: +44 141 548 4840; fax: +44 141 552 2487.
*E-mail address:* bstephen@eee.strath.ac.uk (B. Stephen).

1084-8045/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2006.06.004

Acceptable Usage Policy (AUP), the costs of abuse emanate from more than just time wasting. Into every contract of employment there is an unwritten, mutual degree of trust between both parties. Implicitly this would amount to 'Do your work and I won't ask you if you're doing your work'. More formally however, most companies opt for an AUP signed by employees before they are permitted to use network resources. After all, the directors of a company are responsible for misuse of their company's resources. A recent survey by Hoffman et al. (2003) showed that in email usage alone, more than half of employees asked admitted to behaving 'immorally' in using the medium with almost a third having sent offensive material.

The majority of companies questioned in the (Hoffman et al., 2003) study monitored their employees Internet usage either constantly or constantly 'only with good reason' i.e. a particular offender. This 'good reason' was based upon allegation of misconduct derived from routine manual checks on URL requests. This fairly laborious process entailed taking the top 10 requested URLs and looking for those that served no apparent business purpose. Whether this entails manually checking the content of the offending URL or merely its presence on a blocking list, this is an operation that would have to be performed regularly by IT personnel in order to be effective. Manual monitoring in this way also incurs additional precautions to be taken, particularly with regard to privacy and data protection. Many laws on corporate monitoring both in the UK and the US advocate an automated approach to monitoring of employees behaviour and use of corporate communications devices (100 STAT, 1848; Statutory Instrument, 2000).

This more recent development calls for far greater accuracy in categorising the nature of access. If a decision to confront an employee on this issue is made on the basis of a false alarm, management may face harassment claims from the employee who triggered it. Inevitably, sites will be visited out with the control of the user, either through:

- mistyping the URL,
- 'hi-jacking' a domain name,
- pop-up advertisements,
- automated browser redirection.

To counter the shortcomings in existing approaches to network usage monitoring, this paper proposes a system that automates this task using an expandable community of software agents performing common information retrieval tasks. The approach proposed for gathering usage data is capable of providing far more detail than existing 'network edge' approaches. While this paper focuses entirely on monitoring Web browsing use of the Internet, the principles can be applied to any data source such as email or even general computer use. Commercial software exists that monitors computer use, but not in a collaborative and distributed approach like the one proposed here, one that would permit an in context representation of behaviour and content. The major problem with monitoring usage in this way is the vast quantities of data obtained and lack of a centralised data source for viewing information in context.

### *1.1. Existing approaches to monitoring*

Many tools opt for a 'network edge' approach to monitoring users. While this is good for monitoring traffic it does not give an accurate picture of user activity in terms of content or behaviour. The so-called network edge approach is usually in the form of a proxy server (Fig. 1), for HTTP specific monitoring or a packet sniffer for more general

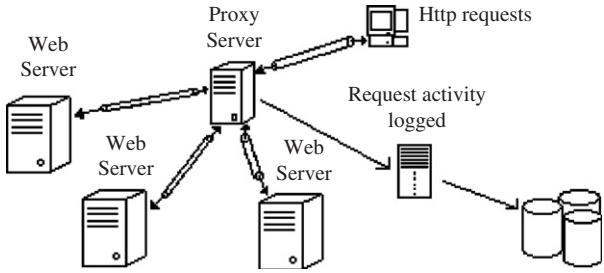


Fig. 1. Using a proxy server: multiple local machines issue requests to a proxy to make the requests to a web server on their behalf. Activity is logged at the proxy.

network traffic. In the latter case, tools such as SNORT[1] can fulfil this role as they can be augmented with rule or pattern matching scripts to decode traffic.

Analysing the logs generated by HTTP requests to a web server via a proxy may give some insight into a users browsing behaviour. Features in web server access logs include:

- user IP address,
- time of request,
- request method,
- URL of page,
- protocol used,
- return code,
- bytes transmitted.

Firstly, the remote machine and user can (usually) be identified from the request header, the header data being populated by the browser that made the request consisting of the remote machine IP address, the user name of the currently logged in user and the browser name/version/platform. Secondly, the time of requests is logged, so time between requests can be calculated, giving an approximation of the time spent viewing a page. Although ideal for their intended tasks such as blocking by keyword or volume, the data being captured will not necessarily reflect what the recipient user is doing:

- Locally cached pages do not require additional requests to be made to their site of origin. For example, the use of 'forward' and 'back' buttons on a Web browser can account for up to 80% of all page viewings.
- An accidentally selected page can be discarded instantly but cannot be differentiated from an intentionally selected one.

Limitations of existing monitoring approaches such as the one just described can be summarised as

- inadequate data acquisition,
- reliance on manual use,
- lack of suitable data analysis.

[1]http://www.snort.org.

Neither the proxy server approach nor indeed any network traffic monitoring will give a complete or accurate picture of the actual behaviour of individuals. Both suffer from inadequate data acquisition, with the source data often being ambiguous, irrelevant or absent. For example, there is no accurate way of knowing how long a user spent viewing a page or what they did with that page, going by the contents of a proxy server log. The page may have been requested by accident, the time spent viewing the page can only be approximated by considering the time of the next page request emanating from the user's machine and requests for pages are not always made every time the page is viewed (if the page is cached locally for example).

Other tools have recognised this and attempted to instrument the usage of the entire operating system. Although this makes detailed and accurate usage data available to the monitoring system it is obtained in vast quantities that are usually dealt with in lengthy reports. Additionally no account is made of content. A weakness of many commercial tools intended to block sites is their reliance on URL and keyword blacklists. Running a check through such a list may stop one known site but let potentially thousands of unknown or mirrored sites through. Correlating pages with their content would also help account for legitimate use of network resources.

Even assuming relevant data has been accurately captured, the next pitfall many approaches fall into is the processing of the data: in most cases this involves the manual generation of a lengthy report possibly listing sites visited. Regular production and inspection of a sizeable document for even a small company may be outwith resources constraints of an already overburdened IT support staff. Data mining is useful in this situation to detect and remember patterns of usage in a statistical model or profile. Sequential, co-occurrence and categorical data is obtained through the capture of behaviour, content and collaborative observations.

## 2. Software agents

There are numerous definitions of the behaviours and attributes that constitute software agents and differentiate them from conventional software. Essentially, an agent is an entity that performs a task on behalf its user, a definition so general it has led to the term being often abused—since the term first emerged in software circles more than a decade ago, its use has become increasingly widespread and its definition increasingly vague. Part of the blame for this falls upon marketing departments, believing agents to be a software trend and using it as a selling feature leading to anything from a scheduled disc defragmenter to an animated interface to a file searching facility being called an agent. However, far more rigorous criteria have since been proposed as a result of this that would set agents apart from software programmes, in particular:

- agents can communicate with other agents,
- agents communicate with one another using an agent communication language (ACL) (Genesereth94),
- agents have only minimal interfaces with other agents,
- agents are temporarily continuous (i.e. they do not do the same task all of the time and when they finish they do not halt completely),
- agents can sense the environment they are in to some extent,
- agents have their own goals possibly driven by a satisfaction or survival function,

- agents can plan,
- agents have some ability (or abilities) they can offer to perform on behalf of other agents by using the resources they possess,
- agents may have reproductive capabilities.

At a higher level and by far the most rigid definition with far more strict requirements for the attainment of agent status is that provided by Foner (1993), identifying the following key properties that define an agent:

- *Autonomy:* the prospective agent should exhibit initiative, periodic action and spontaneous execution—not just responding to the actions of a user but possibly pursue a goal independently of user intervention.
- *Personalisability:* the ability to extract and remember user attributes/preferences—often referred to as 'profiling'.
- *Discourse:* the agent and user must have common ground. A two-way feedback is required so that both parties can make their abilities and intentions known and find mutual agreement on the task required, this may require repeated interaction to achieve this iteratively.
- *Risk and trust:* agents are, by their nature, delegated tasks by other entities. Delegation requires trust that the agent can carry out the task to a reasonable standard as well as the risk of consequences resulting from an improperly executed task.
- *Domain:* domain of interest-placing the agent in an unsuitable environment, one for which it was not designed, should cause it to expire and free resources without bringing the entire system down.
- *Graceful degradation:* the agent should stand down without causing harm to others in the event of failure.
- *Anthropomorphism:* pretending to be human—there is controversy over this point as it is not really considered essential that its user perceive an agent as human.
- *Co-operation:* agents should communicate with their peers as well as the user. Collaboration with other agents as well as delegation of tasks to other agents is often required in situations where a task has multiple, highly specialised sub-tasks.
- *Expectation:* expectation should match reality—that is the agent's capabilities should match what the user or agent delegating the task expects the agent to achieve.

It should be remembered that Foner (1993) dealt mainly with user assisting agents, rather than multi-agent systems designed to automate large-scale tasks, so points such as anthropomorphism and personalisability are less relevant in this domain. More important is the conduit through which the agents exchange knowledge: the ACL.

### 2.1. Agent communication

The main purpose of an ACL is to exchange information (conceptually thought of as knowledge) between agents for the purpose of querying, asserting or updating it. Unlike remote procedure calls (RPCs), remote method invocation (RMI) and CORBA, which will perform the aforementioned, an ACL encodes additional semantic complexity through the use of speech act like representations. In doing this, individual conversational exchanges can be used not only to convey information but to also change the state or attitudes of the

recipient agent, possibly inducing further exchanges. Furthermore, the role of the ACL serves to bridge the not insignificant gap between the transport protocol and the knowledge exchange representation.

The work of Finin et al. (1994) and Genesereth and Ketchpel (1994) led to the notion of the ACL as it is today: a two-tier message structure consisting of a content language for knowledge representation (KR) enveloped by a communication language enabling transport. Although popular, well supported and capable of being used in their own rights, the resulting languages, KIF for KR and KQML for communication had various shortcomings. KQML was unrestricted by formal semantics that led to it being used in a number of ways not intended due to the ambiguous labels its performatives were assigned. And while KIF was not the only content language used with KQML, it was restricted to logical representations that are not always appropriate for exchanging knowledge pertaining to the processes being dealt with.

Although KQML claimed not to be associated with any particular agent architecture, there is a minimum level of conformance expected of KQML speaking agents at their most basic level. The three prerequisites of KQML communication are:

1. the existence of a 'router' entity,
2. the existence of a 'facilitator',
3. each agent is equipped with a KQML router interface library (KRIL).

To elaborate on the above, Finin et al. (1994) described a router as a separate process associated with a particular agent, performing asynchronous transmission and receipt of messages. Routers are designed to deal only with KQML messages and their transport, not their content, performative names or role in agent conversations.

Each KQML message is wrapped in a balanced parenthesis syntax (Fig. 2) that betrays its roots in the common LISP programming language, where, syntactically it would have been equivalent to a list of elements (it is noted in External Interfaces Working Group (1992) that the syntax is unimportant and can be replaced if need be). Parameters carried in a KQML message include (but are not limited to) the names of the sender and recipient, the ontology name and the identifier of the message (which may be used by subsequent messages in a conversation). Message parameters are identified by name and not order or position since the number of parameters present varies depending on the type of message that is sent. The first element on the list is the performative name whose presence is mandatory; all other fields may or may not be filled in since the recipient may either have no need for them or no understanding of them (Table 1).

Once transport addresses are initialised, conversations between KQML speaking agents are conducted by passing the CommonLisp-formatted messages as strings to other agents.

```
(advertise
        :sender NASDAQ-AGENT
        :recipient FTSE-AGENT#1
        :ontology NYSE-TICKS
        :language LPROLOG
        :content (monitor
                        :content (PRICE ?x ?y)))
```

Fig. 2. Typical KQML message in balanced parenthesis syntax.

Table 1
KQML performatives—these are analogous to speech acts, used to invoke changes in agent knowledge

| Achieve | Unachieve | Ask-if | Ask-about | Ask-one | Ask-all | Evaluate |
|---|---|---|---|---|---|---|
| Reply | Transport-address | Sorry | Delete-one | Delete | Insert | Delete-all |
| Tell | Unregister | Deny | Stream-about | Stream-all | Eos | Standby |
| Ready | Recommend-one | Discard | Recruit-one | Generator | Broadcast | Broker-all |
| Next | Recommend-all | Rest | Recruit-all | Subscribe | Monitor | Register |
| Untell | Forward | Pipe | Broker-one | Break | Error | Advertise |

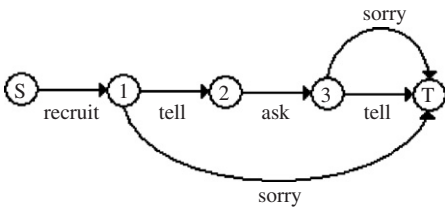


Fig. 3. A state machine representation of the KQML 'register' conversation given in External Interfaces Working Group (1992).

In Fig. 3, a simple conversation is represented as a finite state machine. The key idea is that agents are implemented to both act on message performatives according to their present knowledge and allow them to modify their knowledge. Without this 'advertise' performative, no communication between KQML agents can take place—this performative permits future requests for the performative given in the associated messages content field to be processed on behalf of other agents. Performatives such as 'ask', 'deny' and 'reply' are the staples of agent communication in KQML, used once agents are familiar with each other's capabilities and transport addresses. Typically these will be used in the interrogation of the agent knowledge base; however, automated notification of changes in agent knowledge can be achieved by means of the 'subscribe' performative.

The popularity of XML as a document encoding has led to its widespread adoption in software both as a means of formatting and storage. Intuitively, the hierarchical document structure of XML lends itself well to the persistent storage of application runtime objects in a language and platform neutral manner (Cox, 2001; Reinhold, 1999). Commonly known as seralisation, the conversion of object to document and back is at the backbone of many distributed application frameworks including CORBA and RMI, with some, such as SOAP, even using XML as their seralisation format for platform and language neutrality. One example of seralisation is given in Cox (2001) of an XML representation of an object representation of a fuzzy set. This example is illustrated in Fig. 3.3.1. In the proposed system, the fuzzy variables are parsed into their run time objects and in doing so, so are the serialized objects deeper down the hierarchy until deserialization is complete. Reinhold (1999) illustrates the present standard practice of persistently serializing runtime objects in the Java programming language, using XML, in this same way.

Extending this to KQML by treating each message as an object, it is proposed here that the common LISP syntax of KQML be replaced with a mark up language syntax. Although Labrou et al. (1999) suggested this, to the author's current knowledge such an

```
- <advertise sender="NASDAQ-AGENT" recipient="FTSE-AGENT#1">
    <ontology>NYSE-TICKS</ontology>
    <language>LPROLOG</language>
  - <content>
    - <monitor>
        <content>PRICE ?x ?y</content>
      </monitor>
    </content>
  </advertise>
```

Fig. 4. KQML message shown in Fig. 2 embedded in XML syntax.

implementation has never materialised although several attempts were made with other ACLs (Grosof and Labrou, 1999; Moore, 2000). Doing this enables additional data to be encoded in messages (through the tag attribute feature) without increasing the parsing depth of the message. XML parsers are far more commonplace than common LISP parsers therefore enabling easier implementation of ACL interpreters. From FLBC (Moore, 2000), the feature of including the sender and recipient agents in the attribute field is borrowed, making it easier for the agent to handle, not having to consult lookup tables for transport addresses (if the implementation allows it) for replies and allows easier visual inspection of agent messages. With this exception, the message is more or less hierarchically similar to its common LISP syntax counterpart, shown in Fig. 2, until the content field is dealt with. Minor enhancements included enforcing the optional message identifier field as a timestamp/agent name combination instead of the original undefined format. This again, allows better human readability of agent messages as well as easier handling on the part of the agent (Fig. 4).

More recently other ACLs such as FLBC (Moore, 2000) have also adopted an XML embedding syntax. XML permits the embedding of binary data within documents, usually to enable the integration of existing data formats such as WAV and JPEG into a new document format. Binary content can be embedded in XML as either a tag value or attribute but only if it is encoded as a base64 or hexadecimal string. Base64 encoding turns raw binary data into a string of characters that take one of 64 values (upper and lower case alphabetic, numerals, '+', '/' and '='). Note that since '<' and '>' are not included in the encoding, the parser cannot be fooled into closing a tag and corrupting the stream or malforming the tree. Side effects of this method of embedding binary data in XML are the increase in data size once it has been encoded (by 1.33 times for base 64 and doubled for hexadecimal) and the additional overhead incurred by encoding and decoding data. The coupling of recursive and binary embedding of data within messages permits far more complex exchanges between agents than the simple PROLOG and KIF strings originally envisaged.

## 3. WebEngzilla system overview

Building on the work begun in Stephen and Petropoulakis (2005), the approach taken by WebEngzilla is to plant 'sensors' on the target machine that, when activated, are loaded by the browser and effectively become part of it (Fig. 5). The sensors start automatically, have no user interface and are therefore completely invisible to the user of the target machine. During their operation, the sensors send data back to a remote host where it is logged and

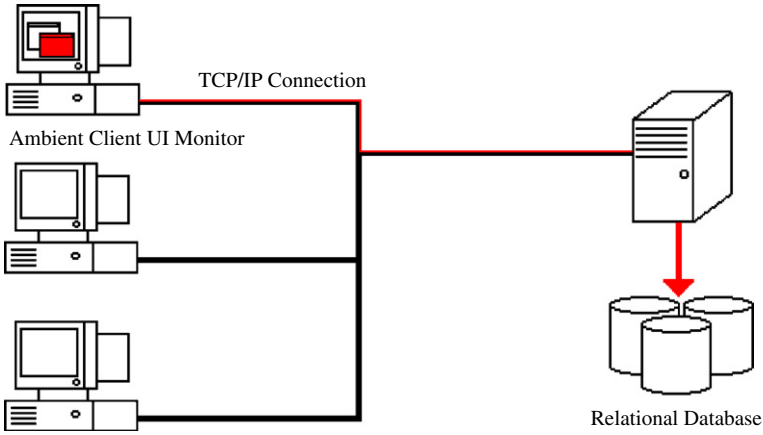


Fig. 5. The client user interface monitor referred to in the text as 'sensors'.

analysed, possibly in real-time. Embedding the sensor in the browser allows far more data to be gathered from the user than either of the aforementioned methods:

- menu activity—printing and saving pages,
- document scrolling,
- the browser is monitored, not the connection it makes with the web server, this allows the measure of:
  - o exact time spent on a given page,
  - o page revisits within the same session,
  - o how the user got to the page—hyperlink, typed, forwarded,
  - o if the browser is actually being used,
  - o when the session actually finishes (when the browser closes).

Conveniently, this level of detail permits the monitoring not just of a single user but also every browser instance that they might have open, allowing individual sessions to be independently monitored rather than as an aggregate of all user activity. The implementation details of the sensors are lengthy and outwit the scope of this paper but one of the many possible means of achieving the same functionality is detailed in Brubacher99 (Brubacher and Hunt, 1999).

Typically, the server will host a relational database for storing document and user information, agents for interfacing with sensor instances, agents for extracting document content, agent for interfacing with the relational database and most importantly a single 'agent broker'. The community of agents commits the gathered data to a relational database for storage and carries out analysis tasks on it (Fig. 6). Ideally, the sensor and the host server it connects to should be thought of as a single entity, which can be queried through an agent interface regarding the status of connected users. Exploitation of agent functionality such as subscription and delegation enables the automation of data mining tasks in accordance with the systems understanding of currently monitored user's behaviour.

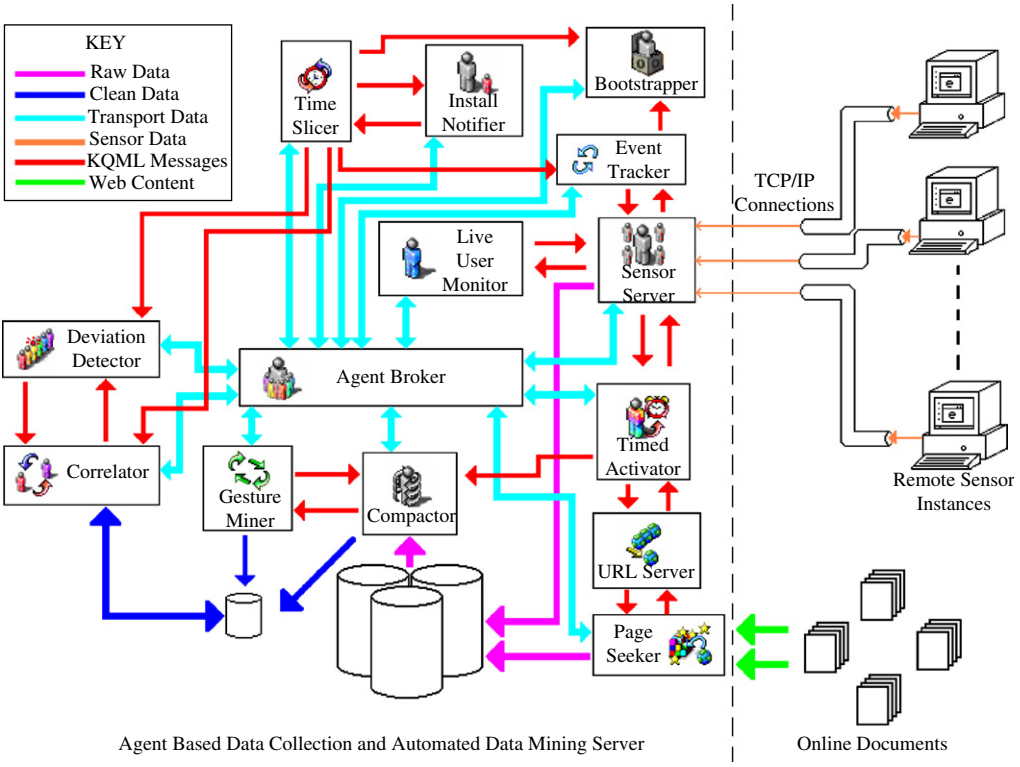


Fig. 6. The overall system, comprising remote sensors and distributed data collection and analysis server.

The 'agent broker' is the backbone of the proposed system and partly plays the role of the facilitator described by Finin et al. (1994) in the KQML specification. Although it has partial agent functionality in that it utilises an ACL to communicate, it cannot be regarded as an agent as it is purely reactive, has no identifier and has a strictly client/server relationship with the agents it serves. A sole agent broker instance exists on any given host, awaiting connections from agents wishing to be assigned a transport address and wishing to know the transport addresses of other agents. WebEngzilla agents initialise by contacting their local agent broker on a predefined transport address (sockets are used for the current implementation so this means a port and a host name), inform it of their name and in return receive an address to which they listen for incoming messages. Conceptually, this approach is not dissimilar to the federated agent architecture proposed by Genesereth and Ketchpel (1994), differing in that agents do not have to communicate through a single gateway with every message sent outwith their host/process/thread boundaries. In the system proposed here, agents make a one off communication with the agent broker at the start of their executing lives leaving the agent broker free to deal with new agents being introduced to the system and existing agents shutting down.

This is intended to alleviate a communication bottleneck when dealing with frequent agent messages. The agent broker also differs from the KQML Facilitator is in its provision for distributed processing by means of interaction with other machines hosting agent broker instances. Providing they are manually configured to be mutually aware

(for security purposes), their registered agents can set up peer-to-peer communications. This is made possible by the exchange of the agent to transport address directories held by each agent broker with additions and deletions from individual tables automatically triggering updates sent to all known agent brokers. In this situation it is possible for an otherwise idle group of agents to take on the tasks of an overloaded system. The next section elaborates upon the functionality of these agents and their relationship to the system as a whole.

## 4. WebEngzilla agents

Following the example of earlier agent toolkits such as JATLite (Jeon et al., 2000) and Jackal (Cost et al., 2000), WebEngzilla utilises a common framework upon which all agents are built, enabling consistent baseline functionality. The authors of the aforementioned toolkits recognised that by using an object-oriented (OO) language to implement agents, mechanisms such as inheritance and polymorphism could be utilised to extend agent functionality for specific applications.

Ferber (1999), Genesereth and Ketchpel (1994) and Franklin and Graesser (1996) all noted that if an entity calling itself an agent has a complete representation of the environment it operates in, then it is not really an agent. In this case it could only be described as a programme since all eventualities are explicitly catered for and data is bound to the programme. So, despite their implementation language, WebEngzilla agents are equipped with a limited degree of reflection (Watanabe and Yonezawa, 1988) allowing them self-examination abilities through a function binding interface developed by Bilas (2001) (WebEngzilla is C++ based, both Jackal and JATLite were Java based and therefore have reflection by default). Reflection is the means of enabling software to look at itself and, if realised fully, modify itself while running. Central to reflection are the concepts of meta-objects (objects that represent objects) and reification (effectively the applications data is itself). What reflection does not do is provide any of the logic for an application to actually reprogramme itself.

By binding classes and functions to their string representations allows agents to be aware of their own capabilities and advertise them. Genesereth and Ketchpel (1994) noted that the ideal ACL would be more akin to a scripting language and the approach adopted here allows messages to be treated as scripts i.e. interpreted at runtime. Even this limited implementation of reflection allows a degree of fungibility in the agent's control structure and permits agents to obtain a representation of their environment entirely through their own observations.

The 'ontology' field, whose role in KQML is noted by (Labrou et al., 1999) to be ill defined, is used as a guide to the interpretation of the 'content' field. Ontology (Gruber, 1993), in the context of agents and KR, refers to the formal specification of concepts and relations within a particular agents domain. Whatever KR is used, its ontology will, in practical terms, specify to an external entity (i.e. the KR interpreter) how to assert and query this representation. KR schemes can be roughly partitioned into the following four categories:

- logical,
- procedural,
- network,
- structured.

WebEngzilla for now utilises only a structured KR scheme analogous to the object oriented structure of the agents that use it.

In WebEngzilla, the generic agent is designed to be able to handle all performatives irrespective of Ontology. Default behaviour includes:

- *Initialisation and shutdown*—interaction with the agent broker is the absolute minimum requirement as without this an agent is incapable of finding the transport addresses of other agents to communicate with.
- *Message sending and receiving*—hiding the underlying transport medium (pipes, sockets, memory mapped file or a higher level transport medium). This listens for, receives, parses and queues incoming messages.
- *Default message handling*—preventing other agents being left by an unresponsive agent. By default all messages are replied to with a 'sorry' performative.
- The ability to broadcast messages.
- *Ontology management*—which agents are able to process which performatives.
- *Conversation handling*—In common with the way Jackal utilises the 'in-reply-to' and 'reply-with' fields to keep track of conversations, WebEngzilla indexes current active conversations are by their message id which allows an incoming message to 'wake up' the message handling thread of a suspended conversation, if the thread is already in an active state the message is added to the handlers private queue. Having conversation specific message queues means that messages in a conversation do not get handled out of their intended order.
- Subscription management—subscriptions, in the KQML context, are effectively promises to provide subscribers (other agents) with data as it changes or is received. In WebEngzilla, subscribers are stored in a table indexed according to request ontology and subscriber identifier, with the respective function being bound to a functor that notifies subscribers of any changes resulting to the ontology as a result of the action.

Extending the basic agent by OO inheritance allows these functions representing performatives to be overridden and their functionality substituted or enhanced. Capabilities such as knowledge storage, representation and acquisition are implemented by subclasses of the basic agent. From the beginning it was anticipated that a core set of agents would be required to provide the basic functionality of the system with additional agents added on the fly to perform increasing specialised tasks. So while not an essential functional requirement, a common set of default behaviours reduces the development effort involved in implementing new agents. The agent-based architecture used means that additional agents may be added 'on the fly' for more specialised monitoring, data processing tasks or scheduling activities based on time or resource usage. Additionally, agents can be situated remotely for distributed processing of tasks. In Fig. 6, the complete system is shown along with its relationship with client sensors.

### 4.1. Sensor server agent

The main component of WebEngzilla is the client/server pair of the sensor instances and the sensor server agent. This provides an agent interface to each sensor as well as being the main data collection and storage point. Other agents are able to manipulate the sensors and receive event notifications from them via communication with the sensor server agent

using the previously discussed variant of the KQML ACL. The functionality available is then represented as ontologies that the sensor server agent supports. Specifically, these pertain to:

- *Connected users*—which users (sensor instances) are connected at any given time as well as notification of the addition or removal of users from the system.
- *Connected user history*—allows the enumeration of a particular user's browsing history.
- *Connected user favourites*—allows the enumeration of 'favourite' URLs (also known as 'bookmarks') in the same way as.
- *Connected user keystrokes*—subscription or notification of all browser keystrokes may be obtained through this.
- *Connected user events*—subscription or notification of particular events may be obtained through this.
- *Connected user deactivate sensor*—allows the deactivation of a particular monitoring capability of a sensor installation.
- *Connected user activate sensor*—allows the activation of a particular monitoring capability of a sensor installation.
- *Connected user enumerate sensors states*—enumerates the status of all monitoring capabilities of a sensor installation.
- *Connected user navigate*—navigates the browser associated with a particular sensor instance to a specified page.
- *Connected user change host*—changes the sensor server a sensor installation connects to by default.
- *Connected user stand down*—halts all incoming data from a particular sensor and closes its connection.

The basic functionality of the generic agent, subsumed by the sensor server agent (by means of OO inheritance), enables the handling of subscribers and broadcasting of capabilities.

### 4.2. Notifier agent

The sensor installation procedure, which can take one of several forms, can be configured to notify a WebEngzilla server of its status. There are a number of reasons why an audit of installations would be maintained, the most prominent being to forecast traffic from sensors and route balance it among available servers accordingly. The notifier agent exists for this purpose—during installation, the sensor installer opens a two way communication with the notifier agent allowing it to receive details of the host system it is being installed upon and to furnish it with a sensor server host IP to connect to.

### 4.3. Live user monitor agent (LUMA)

Although diametrically opposed to the original objective of this system, the live user monitor permits manual control of the system via a graphical interface. The graphical interface contains an agent allowing transparent KQML communication with other parts of the system through message passing and subscription (Fig. 7).

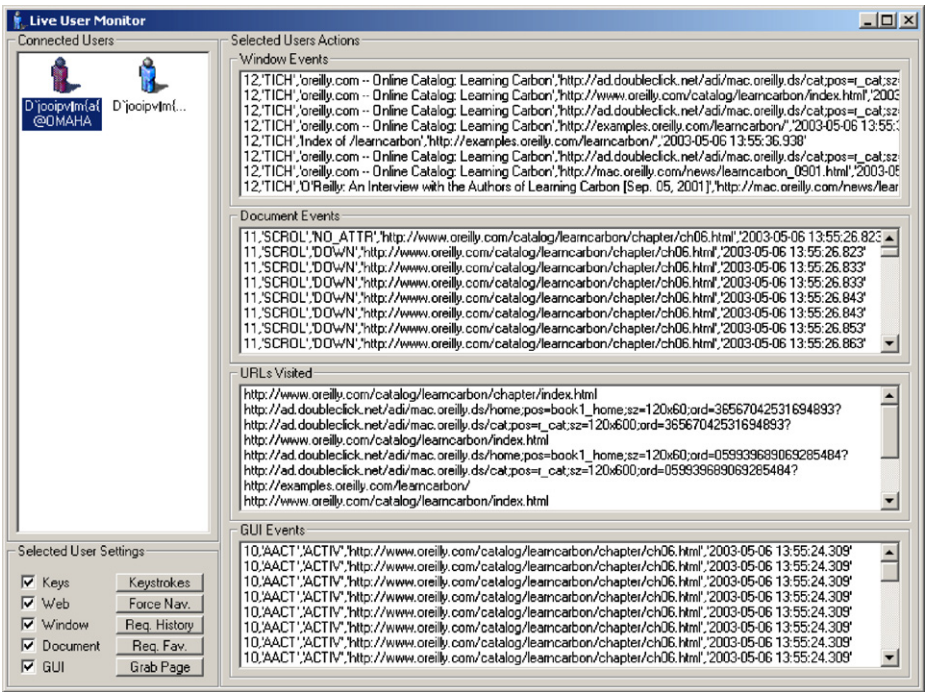


Fig. 7. Live user monitor graphical user interface showing the list of connected users (upper left with currently selected user highlighted in red), lists of the four categories of event being monitored (right) and controls for additional user information (lower left).

The live user monitor is little more than a graphical interface/manual control for the sensor server agent, permitting as it does the:

- URL redirection,
- enumeration of history and favourite URL details,
- display of visited URLs in real time,
- selective monitoring of sensor instances.

For all connected sensor instances.

To better illustrate the semantics of WebEngzilla agent communication, the following transcribes an agent conversation between a LUMA instance and a sensor server agent instance. Beginning at its initialisation, the LUMA seeks a provider for the 'connected users' ontology to populate and maintain the list of users active in the system. This is achieved by utilising the KQML 'recommend-all' performative in a message that deals with the ontology pertaining to agent ontologies—all agents are capable of handling this message and can respond to it after self examination if they are capable of supporting the named ontology or know of an agent who can. As noted in earlier in this section, broadcasting an agent message involves querying the agent broker for the transport addresses of all known agents—this means that a broadcast type message, although broadcast at the agent level, is sent to a specific recipient at the transport level.

```
<subscribe sender="LiveUserMonitor@OMAHA#0" receiver="SensorServer@OMAHA#0">
  <ontology>ConnectedUsers</ontology>
  <language>KQML</language>
</subscribe>
```

Fig. 8. Live user monitor agent subscribing to the ontology 'connected users' provided by sensor server. Any changes to this ontology, which is an access point to the users connected to the system, are posted to any and all subscribers.

Upon finding a provider of the ontology 'connected users' the LUMA makes it known to the providing agent that it would like to be updated as to the current state of the providers which it does by sending the provider a 'subscribe' message pertaining to the ontology 'connected users'. Fig. 8 shows the 'subscribe' message. Receipt of this message means that any changes in the provider's knowledge base 'connected users' are then automatically posted to the LUMA since it is a subscriber. In this situation it is simply a case of posting 'tell' and 'untell' messages to subscribers whenever a user connects or disconnects from the provider agent (in this case a sensor server agent)—messages which are handled by adding and removing user icons from the list control on the LUMA user interface.

Using the list control in the LUMA user interface, a double click on an icon selects a specific connected user to be monitored in real time, the result of this conversation being that the agent 'SensorServer@OMAHA#0' delivers events emanating from the sensor connection of selected user 'D'jooipvm{a{@OMAHA' (an obfuscation of the name 'Administrator') to the agent 'LiveUserMonitor@OMAHA#0'. This string is parsed according to the type of event it is, if for example it is a navigation event the URL associated with it will be separated from the rest of the data in the field and displayed in the respective list control. If a user session terminates, usually by the user closing their browser, any subscribers to the ontology of 'ConnectedUsers' are notified regarding the change of circumstance. Just as subscribers are told of additions to the list of users connected to the system using a KQML 'tell' performative, an 'untell' performative (shown in Fig. 10) serves the opposite purpose (Fig. 9).

So far, data being passed between agents has been restricted to simple strings, as noted earlier, more complex data can be obtained from the sensors and the following examples show how this is handled. In addition to atomic events, the sensors are also capable of retrieving contents of the 'history' and 'favourites' folders for reasons that will be elaborated upon later. These both contain a hierarchy of elements that in turn contain various data types relating to documents viewed, their URLs, the last time they were visited and what the user has labelled them as.

The message shown in Fig. 10 shows the advantage of using XML as the embedding syntax of agent messages: aggregate data structures can be inserted as the agent message body.

### 4.4. Timed activator agent

Many tasks need to be carried out at regular intervals of varying granularities. For the sake of convenience, the timed activator agent exploits scheduling functionality provided by the operating system of the host it is running on although it does allow access to query or request its services by providing an agent interface. This was a fairly important

```
<subscribe sender="LiveUserMonitor@OMAHA#0" receiver="SensorServer@OMAHA#0">
  <ontology>D`jooipv□m{a{@OMAHA?ConnectedUsersEvents</ontology>
  <language>KQML</language>
</subscribe>
```

Fig. 9. Subscription to the event stream being received by the named sensor server agent instance—any incoming events are immediately forwarded to all subscribers.

```
<untell sender="SensorServer@OMAHA#0" receiver="LiveUserMonitor@OMAHA#0">
  <ontology>ConnectedUsers</ontology>
  <language>KQML</language>
  <content content-type="ZSTRING">
    <ZSTRING>D`jooipv□m{a{@OMAHA</ZSTRING>
  </content>
</untell>
```

Fig. 10. Removal of a connected user from the system, represented by an 'untell' message relating to the ontology 'ConnectedUsers' with regard to the user named in the content field.

component in WebEngzilla as some tasks could be too resource expensive to run at high usage times. Training models of user behaviour and interest, for example, could take several hours.

### 4.5. Event tracker agent

The tracker agent, like the timed activator agent, has no functional capability of its own, its sole purpose is to direct other agents and selectively pass messages between them. Unlike the timed activator, whose actions are self instigated, the tracker acts upon certain conditions becoming true in other agents. There are several examples of this:

- sessions finishing and content being retrieved,
- content being retrieved during downtime,
- analysis being carried out during downtime.

Agent-based communication easily permits these intentions to be imparted between components of the system regardless of their physical location.

### 4.6. URL server agent

URL server agents are primarily concerned with URLs visited or cited, collecting URLs visited, and retaining records of requests for URLs and maintaining models of document citation structure. This agent can also pace the gathering of content in conjunction with the page seeker agent to avoid excessive network traffic by undertaking this task during off peak times. Earlier work by Kleinberg (1998) explored the possibilities of exploiting bibliometric document communities on the web. This work was later formalised by Cohn and Chang (2000) to have a statistical foundation, only to be extended again by Cohn and Hofmann (2001) to marry content data with the original citation data. It was noted in both cases that anchor text of links that point to a page can often be more descriptive than the

content of the page itself—in the context of this application domain, this can be useful in obtaining a description of a site with increasing accuracy. Utilising this citation information in a normal web crawler permitted (Hsu and Wu, 2005) to automatically crawl semantically related pages—a useful feature in this context as not all pages visited will give a good content representation of their topic—pages linked to it however, may provide far more descriptive content.

### 4.7. Page seeker agent

Used for gathering content of viewed pages and reducing it to document/text and document/citation co-occurrence matrices, the page seeker agent is essentially a wrapper around an HTTP client with basic functionality and a document parser. Page crawler is a typical reactive agent that is usually started on demand by another agent or agent request.

HTTP is a request/response-based protocol. There are numerous pitfalls likely to befall an HTTP client, incorporating agent behaviour has provided a solution to many of them:

- sites not responding,
- content type unknown,
- content size unknown.

Another problem with content gathering is the 'no-robots' protocol, a means of protecting site content from automated harvesting by third parties. A more pressing problem with content acquisition is the almost ubiquitous use of server-generated pages such as Active Server Pages and PHP. Most popular with e-commerce sites, these pages exist solely during the life of a transaction and cannot be obtained by request at later times (or ever by a third party). To deal with this, WebEngzilla resorts to a number of content acquisition strategies (at present limited to text):

- page acquisition via the browser,
- cache acquisition,
- crawler acquisition.

The concurrent and reflective nature of the WebEngzilla agents, allows past experience of time outs, event driven notifications and self contained knowledge to handle problematic content robustly. Given the widely varying quality of web content this is an important design feature that goes towards preserving the integrity of an important source of data for the system.

### 4.8. Gesture miner agent

The gesture miner agent employs the recognition algorithm developed by Stephen and Petropoulakis (2005) which it uses to turn streams of observed system events emanating from sensor instances into higher level predicates. As the name suggests it is used to extract recurring patterns in behaviour from prior accumulated behaviour and recognise them when they next occur. The algorithm used by the gesture miner agent is detailed in. In learning and extracting behaviour patterns, the gesture miner agent performs a task unique to WebEngzilla. Via the sensor server agent, a stream of events is received from the client;

these observations are a naïve representation of user behaviour with respect to whatever content they are currently viewing. This agent underlines the importance of the agent subscription functionality—without this the gesture miner would not be able to get real time notification of incoming events and would then not be able to recognise a repeated pattern, which it would then notify other agents of. Despite the emphasis placed on action rather than time, the source data retains all relevant temporal details regarding actions.

### 4.9. Compactor agent

Predictably, the database storing the monitored user data will grow unhindered over time. The job of the compactor agent is to periodically (if requested on behalf of a timed activator agent) or on a specific action by a user (if requested on behalf of a tracker agent) reduce the size of the stored data through summary, compression or deletion. For the most part this involves encapsulating the functionality provided by a commercial relational database management system (DBMS) in an agent wrapper; however, to retain the user profile without retaining the data requires that a model be extracted from it.

### 4.10. Correlator agent

The exploitation of machine learning techniques applied to user and document data in so called collaborative and content filters has become common place in e-commerce applications in recent years—the high level objectives of finding similar users and documents to predict future preferences are also desirable in this application domain. A correlator agent would utilise user profiles to augment its knowledge base, allowing it to be queried as to the likelihood of a user being or becoming an AUP violator. Again, because the system is agent-based/ automated, there is no requirement for a constant monitoring of the system. Despite the emphasis on automation and knowledge extraction, provision is always retained for a manual override in the form of visual inspection of the raw contents of the relational database. Data accrued by WebEngzilla is in the following co-occurrence, categorical and time series sets:

- user and document co-occurrence,
- session and document co-occurrence,
- document and content co-occurrence,
- user and content co-occurrenc,
- document and citation co-occurrence,
- content over time,
- behaviour over time.

User similarity can be based on a number of features. The simplest approach is the so called collaborative filter (Breese et al., 1998) that operates on the co-occurrence of users and objects selected by them (such as products purchased or documents selected) then deriving a similarity metric based on the most common selections between users. Godoy and Amandi (2005) based this similarity metric on a hybrid of content and collaborative observations. The level of detail provided by WebEngzilla, permits a greater degree of accuracy. An important distinction is that the user's behaviour can be taken into account. The truly unique aspect of the WebEngzilla data set is that it contains behavioural

representations with every document in a session. The session variable also, is particularly useful in that it distinguishes page revisits in a way that a user variable could not. Yamron et al. (2000) proposed models for 'topic' tracking in news broadcasts, a similar data set to the one gathered here: considering that a session is a time ordered sequence of documents that can be uniquely associated with a user and a document is a collection of words.

Low-dimensional representations such as that of Landauer and Dumais (1997) are popular in both content and collaborative filtering applications as they not only reduce computational complexity but also generalise by dealing with sparseness and in some cases postulate the existence of a hidden causal variable that may be regarded as being analogous to 'topic' in the content case and 'interest' in the collaborative case.

Within the community of agents, a pageseeker agent would be able to utilise the knowledge base of a correlator agent, recent approaches by Hsu and Wu (2005) and Godoy and Amandi (2005) demonstrate how these could be handled in a data driven manner. Rather than pursuing and analysing the contents of every page, base the analysis requirement upon document and user similarity.

### *4.11. Bootstrapper agent*

The bootstrapper agent is for the cold-start profiling scenario where a new user enters the system and in typical circumstances, no inference can be made about this user until sufficient behaviour and content preference has been observed. This agent addresses this issue by building an approximate profile of an individual user based on their browser favourites and/or history, obtainable via the sensor instances monitoring that user. Collaboration with a sensor server agent is required along with both a URL server agent, a page seeker agent and either a tracker agent or a timed activator (all described previously) to obtain a list of URLs, fetch the pages and extract their content. Internally, the bootstrapper maintains a list of users that is kept updated by subscribing to a provider of the ontology 'connected users'.

## 5. Conclusions

This paper has described a novel approach to using software agents for monitoring the acceptable use of network resources. This system utilises agents to marry three features essential to addressing effective monitoring of AUP compliance:

- detailed data capture,
- automated processing,
- data mining for in depth analysis.

A client/server framework for monitoring user behaviour and associating it with content was introduced in order to provide the level of detail required. The volume of data being handled both from the gesture recogniser and the content retrieval tools used, necessitated the use of data mining tools to build representations of user content preferences. In the development of this completely automated system, a generic structure for implementing agents was required. Agents provide a convenient model of abstraction that encapsulates distributed processing, resource sharing, task delegation and automation. Extending an

established model of agent communication, specifically the one associated with the KQML ACL, providing the following benefits:

- *Improved message parsing*—many commercial XML parsers are readily available, different parsing approaches may be adopted (stream-based SAX or in-memory-based DOM) according to implementation requirements.
- *Opaque data storage*—KQML is intended to have no knowledge of the value of the content field in an agent message. The adoption of XML enforces this and even allows the transport of binary data.
- Arbitrarily complex aggregate data structures can be passed between agents as message content.

At the time of implementation KQML was used, while the predominant ACL is now FIPA ACL, the communication principles, based largely on speech act theory, remain largely the same permitting migration with little effort.

While this system represents just a first step, it has underlined the reasons (Wooldridge and Jennings, 1998) provided as to why it is desirable to distribute such a system in this manner:

- *Physical distribution of resources*—for whatever practical reasons (software licenses, hardware limitations) it may not be possible to maintain every resource in the same physical location.
- *Complexity of problem dictates decomposition*—in a physical world problem space it is highly unlikely that there would only be one expert handling all subtasks. Expert systems are not as intelligent or adaptable as their human counterparts so it cannot be expected that they handle diverse tasks with any great degree of involvement.
- *Scalability*—entities need not exist on the same host, allowing tasks to be distributed.
- *Adaptability*—modular design permits the addition and removal of components being updated or used for specialised tasks.

The use of agents in this situation can be justified on the following grounds: automation of collection and processing of large volumes of data from multiple sources, in multiple formats, is a core activity in addressing the limitations of current user monitoring methods. While it could be argued that such functionality could be achieved using existing tools and underlying operating system services, it is the means of unification that is missing—agents provide a means of achieving this. Although it could then be further argued that an object-oriented design methodology could be used to amalgamate the key behaviour providing components, no provision is given in the OO programming paradigm for either distributed programming of components across a system, autonomy or KR. Using an agent-based architecture not only wraps specialised functionality in a combined concurrent processing, error handling, self aware structure, but also permits external queries as to the contents of its knowledge base whether locally or over a common transport medium, thereby allowing a number of very distinct components with diverse functionality to interoperate.

## References

100 STAT. 1848. Public law 99–508, 21 October 1986. Electronic Communication Privacy Act of 1986.
Bilas S. FuBi: automatic function exporting for scripting and networking. In: Proceedings of the 2001 GDC, 2001.

Breese JS, Heckermann D, Kadie C. Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of the 14th conference on uncertainty in artificial intelligence. Los Altos, CA: Morgan Kaufmann; 1998.

Brubacher D, Hunt G. Detours: Binary interception of Win32 functions. In: Proceedings of the 3rd USENIX Windows NT symposium, USENIX, 1999. p. 135–43.

Cohn D, Chang H. Learning to probabilistically identify authoritative documents. In: Proceedings of the 17th international conference on machine learning. Los Altos, CA: Morgan Kaufmann; 2000. p. 167–74.

Cohn D, Hofmann T. The missing link—a probabilistic model of document content and hypertext connectivity. Advances in neural information processing systems, vol. 13. Cambridge MA: MIT Press; 2001.

Cost RS, Chen Y, Finin T, Labrou Y, Peng Y. Using coloured petri nets for conversation modelling. issues in agent communication. Berlin: Springer; 2000.

Cox E. XML and distributed business-to-business intelligence: fusing XML and Java based expert applications. PCAI Magazine, March/April 2001. p. 16–20.

External Interfaces Working Group. ARPA knowledge sharing initiative. KQML agent communication language specification, 1992.

Ferber J. Multi agent systems: an introduction to distributed artificial intelligence. Reading, MA: Addison Wesley; 1999.

Finin T, Fritzson R, McEntire R, McKay D. KQML as an agent communication language. In: Proceedings of the 3rd International conference on information and knowledge management, 1994.

Foner L. What's an agent, anyway? A sociological case study. Technical report. MIT Media Lab.; 1993.

Franklin S, Graesser A. Is it an agent, or just a program? In: Proceedings of the 3rd international workshop on agent theories, architectures, and languages (ATAL-96). Berlin: Springer; 1996.

Genesereth MR, Ketchpel SP. Software agents. Commun ACM, 1994; 37(7): 48–53.

Godoy D, Amandi A. Modelling user interests by conceptual clustering. Elsevier Information Systems 2006;31(4–5):247–65.

Grosof BN, Labrou Y. An approach to using XML and a rule based content language with an agent communication language. IBM Research report, 1999.

Gruber T. A translation approach to portable ontology specifications. Technical report KSL 92-71. Knowledge Systems Laboratory, Stanford University, April 1993.

Hoffman WM, Hartman LP, Rowe M. You've got mail … and the boss knows: a survey by the center for business ethics of companies' email and internet monitoring. Business and society review, vol. 108(3). Oxford: Blackwell; September 2003. p. 285–307.

Hsu CC, Wu F. Topic specific crawling on the Web with the measurements of the relevancy context graph. Elsevier Information Systems 2006;31(4–5):232–46.

Jeon H, Petrie C, Cutkosky MR. JATLite: A Java agent infrastructure with message routing. IEEE internet computing, March/April 2000.

Kleinberg JM. Authoritative sources in a hyperlinked environment. In: Proceedings of the ACM-SIAM symposium on discrete algorithms, 1998.

Labrou Y, Finin T, Peng Y. Agent communication languages: the current landscape. IEEE intelligent systems, May 1999.

Landauer TK, Dumais ST. A solution to Plato's problem: the latent semantic analysis theory of the acquisition, induction, and representation of knowledge. Psychol Rev 1997;104(2):211–40.

Moore SA. KQML and FLBC: contrasting agent communication languages. University of Michigan Business School; 2000.

Reinhold M. An XML data binding facility for the Java platform. Core Java Platform Group technical note, July 1999.

Statutory Instrument 2000 no. 2699. The telecommunications (lawful business practice) (interception of communications) regulations 2000, ISBN 0 11 099984 3.

Stephen B, Petropoulakis L. An ambient software monitoring system for unsupervised user modelling. Expert systems with applications, vol. 28(3). Amsterdam: Elsevier; 2005. p. 557–67.

Watanabe T, Yonezawa A. Reflection in an object-oriented concurrent language. In: Proceedings of object-oriented programming systems, languages and applications, September 1988. p. 306–15.

Wooldridge M, Jennings NR. Pitfalls of agent-oriented development. In: Proceedings of the 2nd international conference on autonomous agents. ACM Press; 1998.

Yamron J, Knecht S, Gillick L, Lowe S, Mulbregt P. Statistical models for tracking and detection. In: Working notes of the DARPA TDT-3 workshop, 2000.