MEMOIR

similar paths and therefore the trails they leave can be compared. By matching trails, we find similar users.

1.1 Links

In advanced hypermedia architectures, information about the links between multimedia documents is stored and managed separately from the documents themselves, which remain in their native formats [15]. The links are objects in their own right; the link data consisting of a set of associated *anchors* or *endpoints* (e.g. locations in documents) and some related information. These links are kept in so-called *linkbases*, the corresponding servers are named *linkservers*. On the Web a link consists of one embedded URL; the 'first class' links we describe here can be thought of instead as collections of URLs (e.g. one associated with the source anchor and the others with the destination) together with associated information.

1.2 Trails

Trails [13, 20, 37] can be seen as threads of different user activities. Opening and viewing a document e.g. following a link to a given URL might be the most common type of activity. However, there are many more interesting activities users perform: printing a document could indicate a special interest or a high priority; mailing a document to someone else might have similar indications. We distinguish two basic entities: trail marks and trails, where a trail is composed of a number of trail marks and the user decides what should be contained in a trail.

1.3 DIM agents

Adopting separate links has important advantages for authors and readers alike (see [8]), especially when working with a complex, distributed information system. Meanwhile, the trail information provides a means of answering questions such as "who else has seen this document" or "what else should I read?".

However, systems that work with "first class" links and trails are more complex to engineer. We have found that the agent paradigm is well-suited to working with these first class links and trails.

Defining the term agent has been — and still is — a controversial and unfinished task [18, 26, 27, 28, 32]. For the purposes of this paper, we define an agent to be a software entity which responds to a request made by a user or by another agent. The problem domain that we address is **d**istributed **i**nformation **m**anagement, hence we call these agents *DIM agents*. In fact, we tackle DIM applications from a weak agents perspective [36]: our agents are autonomous, reactive, often proactive and they have social ability; where appropriate they are mobile and/or intelligent. In addition to resource discovery, we emphasise the role of DIM agents in resource maintenance and integration [10].

This paper is structured as follows. Section 2 characterises the agents in MEMOIR and describes the architecture of the agent system as well as different layers of agents. We present the underlying algorithms as well as agent communication in Section 3. Section 4 deals with related work. We finish with a summary and conclusions in Section 5.

fi The MEMOIR agent system

In this section we will describe the agents system within the MEMOIR framework.

In MEMOIR, agents can provide single or multiple responses to a user's request. As an example consider the case where a user wants to know who else has seen a particular URL. A single reply is returned to the user by a reactive agent which mines the persistent repository of users' trails (the trailbase) in order to satisfy the request. On the other hand, an agent which monitors the status of a specific URL returns a response every time the URL changes status. Agents which are capable of answering in multiple stages are called persistent agents. Following this approach, an agent can answer a user's request by providing an increasing degree of refinement of results at each stage. Persistent agents may call non-persistent ones whenever this is necessary. In MEMOIR, persistent agents have been adopted for building user profiles, monitoring URLs, etc. Tasks including simple database queries, keyword extraction from documents etc., are accomplished by non-persistent agents.

It turns out that in certain cases a user might want to interact with an agent in order to provide it with further input and/or corrections. Thus, both persistent and non persistent agents may possess a certain degree of user interaction.

The agents primarily mine the trailbase and the linkbase in order to accomplish user initiated tasks [4, 6, 24, 33]. Therefore, they can either issue simple or complex database queries to the trailbase/linkbase or extract user/document related parameters from the users' trails and links. Keyword extraction can therefore be considered as an operation which acts on URLs [30], aiming to enhance their content with the addition of keywords. Similarly, building user profiles can be viewed as an operation which extracts user model parameters from users' trails and links [7, 23]. Additionally, agents may check the integrity of data, e.g. a link integrity agent checks whether format and structure of URLs are correct and the documents are present; it can also attempt to repair links when documents move. Some agents may also accept input from the user upon initiation or while their work is in process mode, depending on the designed agent functionality.

2.1 The agent architecture

In order to understand the embedding of the agent system into the MEMOIR framework, we will give a brief overview of the different components and their interaction (see Figure 1). The key component is the *message router*, which acts as a hub with which other system services register. Any component makes its services available by registering details of its services with the message router which in turn can route particular requests of other components to the newly registered one. Messages for querying all currently available services are part of the set of message router services. The model of servicing a component allows the system to be dynamically tailored to specific needs. New services can be easily added, and multiple providers for the same service are possible. We have adopted this particular solution to a service broker because it is simple, flexible, dynamically extensible and uses standard tools and infrastructure; these were the requirements for our target installations.

Standard Java enabled Web browsers serve as user interface to the MEMOIR system in order to minimize installation and maintenance overhead. The only configuration necessary is to set the so-called *interface manager* as a proxy server and to connect to

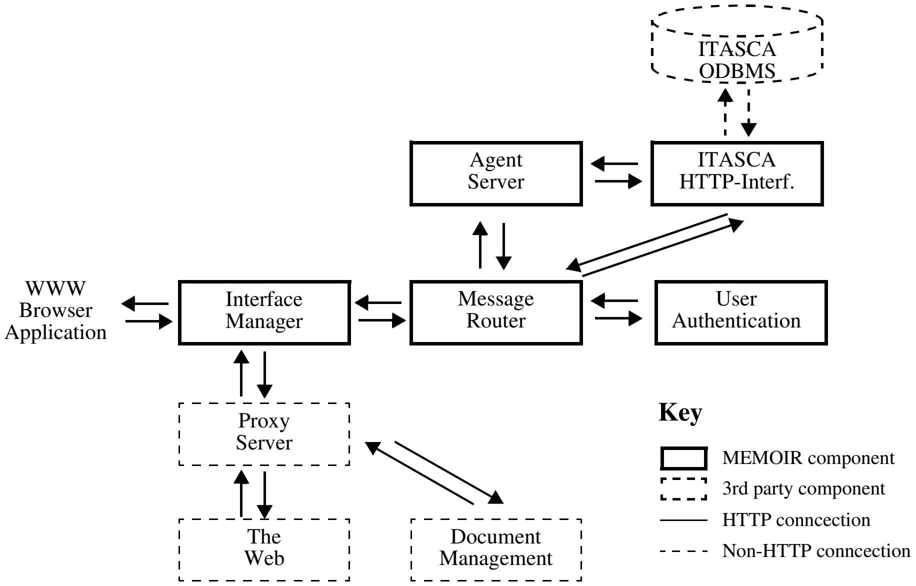


Figure 1: *The components of the MEMOIR framework.*

it once, in order to log in. Figure 2 shows a sample session with the basic MEMOIR browsing panel and an additional browsing window. The interface manager *behaves* as a proxy server and by that is able to listen to a user's requests and commands. It can be configured to use a (caching) proxy on its own. It is via the interface manager that users view the Web, document management systems and also linkbases. Thus, existing data can be re-used and is not stored within the system (though the meta data such as a document's keywords are extracted and stored in the data repository, ITASCA).

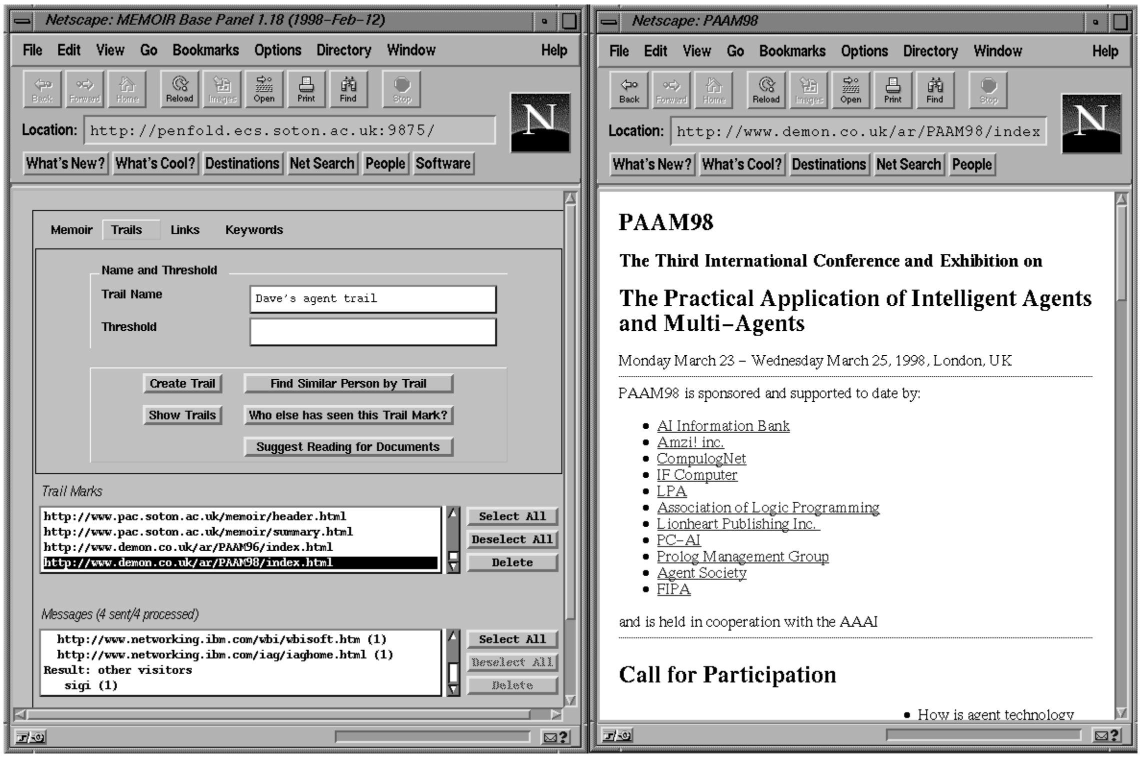
An *authentication component* is responsible for managing user authentication. As far as possible this component relies on existing user management systems.

The ITASCA distributed object oriented database system [19] is used as an underlying *data repository*. It provides the framework for the agents that are implemented directly in LISP, the data manipulation language of ITASCA. As HTTP is used as communication protocol an interface component to ITASCA is provided allowing the communication with the database via HTTP (see Figure 1). This interface component is a server process that provides access to the database by 'translating' HTTP commands to and from LISP function calls.

2.2 The agent server

The current agent system architecture conforms with the principles of modularity, distribution and openness that also characterise the MEMOIR project as a whole. As a result, apart from the set of agents it was considered necessary to add a component named the *agent server* which should accomplish the following tasks:

- Act as an agent service broker which provides provides a list of the available agent services to the rest of MEMOIR.
- Hide agent implementation details from the other MEMOIR components.



Netscape: MEMOIR Base Panel 1.18 (1998–Feb–12)

Location: http://penfold.ecs.soton.ac.uk:9875/

Name and Threshold

Trail Name: Dave's agent trail

Threshold:

Trail Marks

```
http://www.pac.soton.ac.uk/memoir/header.html
http://www.pac.soton.ac.uk/memoir/summary.html
http://www.demon.co.uk/ar/PAAM96/index.html
http://www.demon.co.uk/ar/PAAM98/index.html
```

Messages (4 sent/4 processed)

```
  http://www.networking.ibm.com/wbi/wbisoft.htm (1)
  http://www.networking.ibm.com/iag/iaghome.html (1)
Result: other visitors
   sigi (1)
```

Netscape: PAAM98

Location: http://www.demon.co.uk/ar/PAAM98/index

PAAM98

The Third International Conference and Exhibition on

The Practical Application of Intelligent Agents and Multi–Agents

Monday March 23 – Wednesday March 25, 1998, London, UK

PAAM98 is sponsored and supported to date by:

- AI Information Bank
- Amzi! inc.
- CompulogNet
- IF Computer
- LPA
- Association of Logic Programming
- Lionheart Publishing Inc.
- PC–AI
- Prolog Management Group
- Agent Society
- FIPA

and is held in cooperation with the AAAI

Call for Participation

- How is agent technology

Figure 2: *The MEMOIR user interface.*

- Control flow of messages among agents whenever agent-to-agent communication is needed.
- Control flow of messages between agents and other MEMOIR components, such as the message router, the interface manager, etc.

The agent server is a multi-threaded server entirely implemented in Java that talks HTTP. We chose Java as an implementation language mainly due to its portability features.

There can exist multiple agent servers. In such a case different agent servers are assigned possibly overlapping sets of agents. Each agent server registers its set of services with the message router. Thus, it is the message router which decides where to redirect a user's request for an agent related task to be accomplished (see Figure 3).

2.3 The community of agents

2.3.1 The first layer

We have designed and implemented a hierarchical two layer architecture for the MEMOIR agent community. The first layer consists of non-intelligent, non-persistent, reactive agents performing simple tasks such as database queries of relatively low complexity. As an example consider a "search agent" which receives as input a list of URLs and looks for published trails which are similar to the given input list. In order to compute a similarity score for each trail, the agent counts how many URLs the trail has in common with the input list. The agent receives the input either by the user or by another agent.

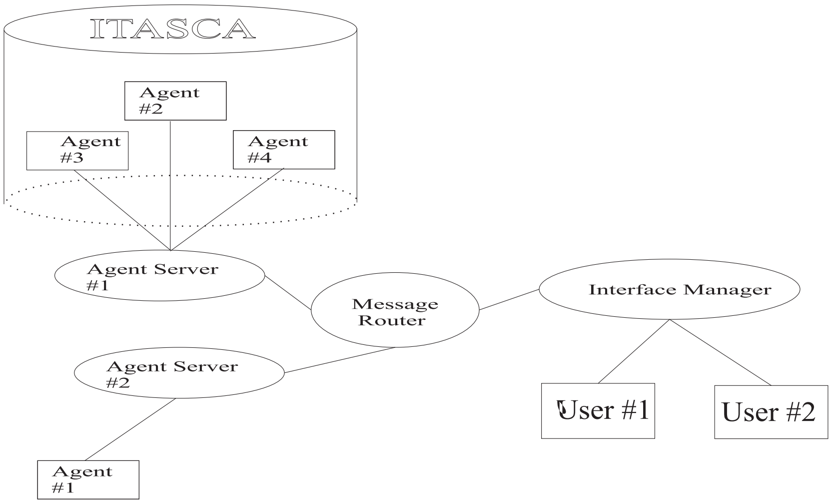


Figure 3: *Set-up and usage of multiple agent servers. Agent server #1 has been assigned a set consisting of agent #1 only, while agent server #2 is aware of a set of three agents. In this case the two sets of agents are non-overlapping.*

In the former case the user explicitly triggers the agent and receives the results, while in the latter case the results are subsequently processed by the agent which provided the input.

As a preliminary result of end-user evaluation the first layer consists of an expanding set of agents. Since we have followed a modular approach new agents can be added without redesigning the existing ones. It is crucial though, that each time a new agent is added, the agent servers become aware of the addition. At the current phase of the project the agents do not automatically register themselves with the agent server upon startup. The registration is performed by human intervention. At a next phase this restriction could be removed with the use of a KQML-like [14] approach.

2.3.2 The second layer

The second layer includes intelligent, complex and possibly persistent agents which can be considered as information brokers. They can suggest documents for further reading, find users with similar interests, etc. Such agents can call first layer agents retrieve the returned results and combine them in such a way so as to give intelligent answers (see also Figure 4 as well as Section 3.2).

3 Agent algorithms

This section gives an overview of the most representative algorithms that have been adopted for the design of the agents of the first and second layer. Due to the end-user requirement of real-time response times the computational complexity of the algorithms had to be kept as low as possible, without reducing the usefulness of the returned results.

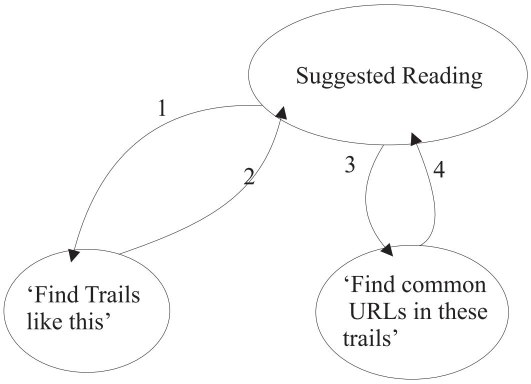


Figure 4: *Example of a second layer agent which recommends further reading given a set of URLs as input. The numbers on the arcs demonstrate the sequence with which this complex agent calls two agents of the first layer.*

fi.fi Algorithms for the first layer agents

This section introduces three first layer "search" agents which can find trails which are similar to a given set of URLs, people whose trails are similar to a set of URLs, and people whose trails contain a given set of keywords, respectively.

The first of the above agents receives a list of URLs as input and provides a set of other URLs that are similar to the given input. The agent compares the input list to each of the published trails. For each comparison a matching score is computed. The matching score is equal to the number of URLs that the input has in common with the trail against which it is compared. Trails, resulting in matching costs which do not exceed a pre-defined threshold, are rejected as irrelevant. Similar trails are ordered according to the matching cost returned.

The second of the above agents employs a similar technique as described above in order to mine the trailbase for other users with similar trails. The input is again a list of URLs. This list is representative of the interest of the user which compiled it. For each user the input list is compared with each one of his/hers trails. From each comparison a matching cost is generated. The sum of the above matching costs provides a similarity measure between the user which compiled the input list and the user whose trails were examined. The similar users are ordered according to the similarity measure calculated above. As with the previous agent, a thresholding mechanism is employed here as well, in order to shorten the length of the returned list of users.

The third agent retrieves user trails based on keywords that are provided as input by the users. In the MEMOIR trailbase, keywords are stored as metadata for each URL that is part of a published trail. For each user, it calculates a matching cost which is the number of occurrences of the input keywords in the published trails of the user. The resulting users are ordered according to the previously calculated matching score. Thresholding is employed by this agent, too.

3.2 Algorithms for the agents of the second layer

The second layer agents make use of the previously described agent functionality to do some intelligent processing with these results. This section describes two such agents which can recommend further reading and assist the user in order to find colleagues who might be experts on a particular domain of interest.

The first of the above two agents is a complex agent that is directly invoked by the user. The input is a list of URLs which are representative of a user's interest. As a first step this agent calls a previously described first layer agent in order to find trails which are similar to the given input. The returned set of similar trails is in turn given as input to another "search" agent, which returns an ordered list of the most frequently appearing URLs in the above set of trails. It is possible that the ordered list contains some of the URLs that the user gave as input to the suggested reading agent. These URLs are filtered out. The remaining ones comprise the set of suggestions to be shown to the user. Additionally, by using Web browsers as the user interface, we can launch another window or even replace the current top window's content with the a new document.

The second complex agent is intended to retrieve experts on a certain domain. Experts on a domain are characterized by people owning trails with domain relevant keywords. The user describes, via a set of keywords, a topic on which he/she wishes to find an expert. As a first step, the agent examines a noticeboard where such queries are kept, in order to determine whether similar queries were asked by other users in the past. We have made the assumption that the user has the option to make his/her queries publicly available. If the agent detects similar queries, it returns the names of the people who made the queries. So, the user has the opportunity to contact these people in order to ask them if they know who the expert is. Secondly, the agent examines the home pages of the users so as to find out whose home page contains the keywords that describe the topic. It then examines which users' trails contain the specified keywords, and at final stage, whose links contain the specified keywords. Results from each stage may be returned separately to the user, or the agent might perform some filtering operation in order to combine the results of various stages.

3.3 Agent to agent communication

In the MEMOIR framework "peer-to-peer" agent communication is a desired feature. This means that bi-directional communication should be feasible for any two agents.

We have implemented agents in LISP, as well as in Java. LISP has been chosen for some of the agents because it is the data manipulation language of ITASCA and we wanted to reduce communication overheads. Java has been mainly chosen for reasons of platform independence.

Communication between agents implemented in the same language is done using function calls or method invocation. Communication between for example LISP and Java agents is done using the MEMOIR message format which is tunneled in HTTP. A sample request for suggested reading could look as follows:

```
GET /memoir_msg?
    \MessageClass+SendMessage
    \User+sigi\Threshold+1
    \url1+http://www.pac.soton.ac.uk/memoir/
    \noURLs+1
    \QueryID+J/4711
    \RID+
    \Asynchronous+Yes
    \ProcessHost+memoir.ecs.soton.ac.uk\ProcessPort+3222
```

```
    \ProcessCaption+MR/Prototype
    \Services+SuggestReadingByTrail
HTTP /1.0
```

Figure 5 demonstrates the routing of this message. The message would be sent to the message router. The message router in turn would "know" which component would actually be able to deal with this message, in this case the agent server. The agent server would assign an agent. The agent would then query the database, and perhaps also the Web, do some intelligent weighing of results and finally return an answer that could look like the following:

```
HTTP /1.0 200 Document follows
Server:  MemoirAgentServer
        \Success 1\Source SuggestReadingByTrail
        \QueryId A/0033\RID J/4711
        \Asynchronous No
        \Diagnostic 2 suggested documents
        \Multi-1 SugDoc
        \SugDoc-1 http://www.pac.soton.ac.uk/memoir/content.html
        \SugDoc-2 http://www.pac.soton.ac.uk/memoir/intro.html
```

The MEMOIR message format is based on the open, extensible structure used in an earlier open hypermedia project (Microcosm, [9]). This format uses simple tag/value pairs and although there is a required set of tags for normal operation of the system, additional tags can easily be added to extend the information carried to suit new functions. The current implementation of the format, with the data carried in the URL (like form data), only supports textual information; it can be readily extended to accommodate binary data if necessary, by encoding the message in the body of another method. The openness of the framework allows straightforward integration of any existing information infrastructure a corporation may already have, such as document management systems or databases. It is however, necessary that all newly added components provide an HTTP interface.

HTTP is a stateless protocol. In order to be able to deal with stateful information and allow the user to work in an asynchronous fashion, we allow agents to report results back, even if a user has quit a session. To do this, we extended the MEMOIR set of messages with additional tags. These mainly include a query ID, a request ID (RID)as well as a flag expressing asynchronous behaviour. A query ID is composed of a component ID and a simple string or number that is managed by each component in order to keep a record of sent and received messages. This allows, for example, the user interface to group a user's requests and answers together, identifying those that refer to the same ID. This is also exactly the task of the request ID: each component that answers a request makes the incoming message ID the request ID and sends a reply message with a new message ID. In case a result is delivered while the user is not logged on to the system the result is stored in a message box within the ITASCA database.

4 Related work

In the area of agent systems there are many systems and prototypes around that can be considered relevant. In this section we describe only those systems and approaches that

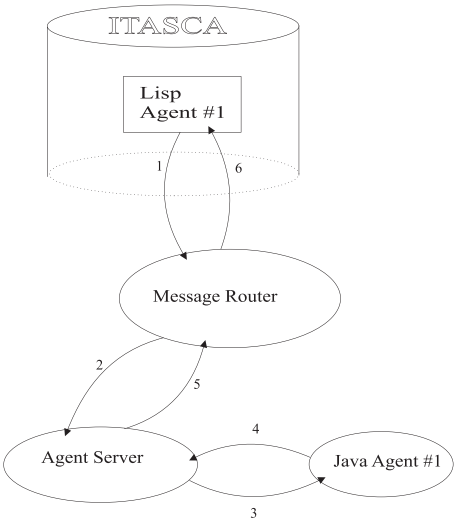


Figure 5: *LISP agent to Java agent communication: the numbers on the arcs demonstrate the sequence with which communication events take place.*

also address in some respect the idea of supporting user collaboration on the Internet. We have grouped the various systems into the following categories:

- Information filtering systems
- Systems based on the users' browsing behaviours
- Recommender systems

Note that some systems might fall into more than one category, so these categories are overlapping. However, we describe the systems within the category that we believe it fits into most.

4.1 Information filtering systems

Information filtering systems apply different algorithms such as pattern matching with contextual analysis in order to provide the users with information that they have specified in a textual way. Mostly news services are used as input. Users define their preferences by giving keywords or voting on pages. This category of systems includes Agentware i3 [1], Atlas Server [2], Beehive [17] and Yenta [12].

MEMOIR mainly differs from these systems by a higher degree of automation in that users express their interests simply by the documents they are viewing and storing as trails. Agents then process these data in order to extract similarity scores, and the more.

4.2 Systems based on the users' browsing behaviours

This category of systems provides tour guide agents that accompany users from page to page providing assistance in browsing / navigating. The agent's knowledge is based on learning from previous tours and hypertext structure. Systems within this category are WebWatcher [21], Vrisko [3], Letizia [25], the Do-I-Care collaborative agent [34] and some systems of the group of recommender systems [29].

Within the MEMOIR project we have done experiments using the trail information for generating guided tours as well as providing the users with some interface of creating and using their own guided tours. The main difference of the approach taken in MEMOIR is that none of the systems mentioned implements the notion of trails as first class objects.

4.3 Recommender systems

Recommender systems [29] stand exemplary for a category of systems that are based on the idea that users often face the problem of having to make choices without sufficient experience and therefore rely on other people's recommendations. Recommender systems assist this process. Rather than using user trails as a main source for providing assistance recommender systems have intelligent agents for weighing user votes, for instances of usenet articles in general [11, 22, 35], for music albums and artists [31] or based on e-mail [17]. *Fab* uses both user votes and content based filtering techniques [5].

MEMOIR could be described as a recommender system. It is however, based on using implicit information or automatic extraction of user profiles. As already mentioned, trails together with links serve as source for the agents. It also has to be mentioned that MEMOIR has a different focus than recommender systems: MEMOIR is about finding people with similar interests by matching trails whereas recommender systems deal with the issue of giving the users suggestions based on other users' experience.

In summarising, it might be obvious simply from the number of related systems that there is quite an interest — including commercial interest — in exploiting the idea of supporting collaboration between people by analysing the information people are looking at. And, although there may be similar ideas behind some systems none of the systems, actually fully exploits the idea of trails. Some re-use paths followed by other users [21, 29] or do intelligent analysis based on the user's browsing behaviour [25, 34], however they do not explicitly implement the notion of user trails.

Yenta [12], is probably closest to MEMOIR in that it aims at finding similar people. However, it also does not follow the idea of user trails and as a systems does not support the openness and flexibility of MEMOIR. This idea of an open flexible framework that can be easily extended both towards agent technology and input data is supported by few systems. Notable exceptions are [1, 35].

5 Summary and conclusions

In this paper we have described the agent architecture in MEMOIR. We have defined how agents inter-operate within the MEMOIR framework and we have explained how agents communicate and which algorithms they are base upon.

The MEMOIR prototype is currently being evaluated by the two corporate end-users participating in the project, namely Glaxo Wellcome and Unichema. The sub-groups to do the evaluation within these organisations include researchers that typically need to browse and search large numbers of documents prior to selecting those they wish to read. Also, technical support people who need to access the similar information, but typically in a way which leads as rapidly as possible to particular information sources, will participate in the evaluation.

The evaluation process has been split into three phases:

- Phase 1: Installation and primary testing;
- Phase 2: Intensive user-centred evaluation; and
- Phase 3: Company-wide availability.

The evaluation is currently starting phase two; yet there are already several conclusions of the work undertaken. Firstly, the open and extensible architecture of MEMOIR in general has proven very successful. Additional or modified agent functionality can easily be introduced into the system by simply registering a new service with the message router. Also, using a proxy based approach for tracing user data has shown to be very successful both because of the minimal installation overhead and because of the easiness of introducing new functionality to mine the data.

Secondly, treating trails as well as links as first class objects allows us to find similar users based on the users' actual browsing behaviour; additionally, links as first class objects being held in separate linkbases and not being embedded in the documents, not only allows users to have their own structure on the documents they are using but also to share these structures with other users.

We also believe that the approach taken by the "expert finder", i.e. to find similar users by matching user queries, is both innovative and promising.

Lastly, using an agreed set of textual messages for communication of agents and other system components allowed us to implement agent functionality in different languages appropriate to the application domain. Agents that mainly mine the ITASCA database are implemented in LISP; agents which "crawl" on the Internet are implemented in Java. This confirms our "open" approach.

6 Acknowledgements

The editors would like to thank the members of the MEMOIR project, all of whom have contributed to this paper. In particular we would like to thank Andrzej Glowinski, Alastair Hotchkiss and Roger James of Glaxo Wellcome; Steve Loades and Richard Miller of Unichema; Roger Rowe from Multicosm; Eric Gerelle, Christophe Giusti and Jacques Halé of Ibex; Gerard Hutchings and Chris Scott from the Parallel Applications Centre.

We would also like to acknowledge the support of the European Union's ESPRIT programme which funds the MEMOIR project under No. 22153.

The work of Sigi Reich has been supported by the Austrian Fonds zur Förderung der wissenschaftlichen Forschung (FWF) under grant No. J1507-INF.

We would also like to acknowledge the support by the EPSRC project No. GR/K73060.

References

[1] Autonomy. Tech. rep., Autonomy Inc., 301 University Avenue, Suite 200 Palo Alto, CA 94301. Available as `http://www.agentware.com/`, 1997.

[2] Tympani Atlas Server. Tech. rep., Tympani Development Inc., 1307 South Mary Ave., Suite 102 Sunnyvale, CA 94087. Available as `http://www.tympani.com/products/ATLAS/AtlasBrochure-.pdf`, 1997.

[3] Vrisko — personal knowledge manager. Tech. rep., Zuno Limited, 300 Third Avenue, Waltham, MA 02154. `http://www.dlib.com/Products/vrisko.html`, 1997.

[4] Atkins, D., Ball, T., Baran, T., Benedikt, M., Cox, K., Ladd, D., Mataga, P., Puchol, C., Ramming, J., Rehor, K., and Tuckey, C. Integrated web and telephone service creation. *Bell Labs Technical Journal (Software) 2*, 1 (1997).

[5] Balabanović, M., and Shoham, Y. Fab: Content-based, collaborative recommendation. *Communications of the ACM 40*, 3 (1997), 66–72.

[6] Benford, S., Snowdon, D., Brown, C., Reynard, G., and Ingram, R. Visualising and populating the web: Collaborative virtual environments for browsing, searching and inhabiting webspace. In *Proceedings of the 8th Joint European Networking Conference, Edinburgh, Scotland, May 12-15* (May 1997).

[7] Billsus, D., and Pazzani, M. Learning probabilistic user models. In *Sixth International Conference on User Modeling, Chia Laguna, Sardinia, 2-5 June* (June 1997).

[8] Carr, L., De Roure, D., Hall, W., and Hill, G. The distributed link service: A tool for publishers, authors and readers. In *Proceedings of the Fourth International World Wide Web Conference: The Web Revolution* (Boston, Massachusetts, USA, Dec. 1995).

[9] Davis, H. C., Hall, W., Heath, I., Hill, G., and Wilkins, R. Towards an integrated information environment with open hypermedia systems. In *European Conference on Hypertext Technology (ECHT) '92, Milano, Italy* (1992), pp. 181–190.

[10] De Roure, D., Hall, W., Davis, H., and Dale, J. Agents for distributed multimedia information management. In *First International Conference on the Practical Application of Intellgient Agents and Multi-Agent Technology* (London, UK, Apr. 1996), pp. 91–102.

[11] Falk, A., and Jonsson, I.-M. PAWS: an agent for WWW-retrieval and filtering. In *Practical Applications of Intelligent Agents and Multi-Agents (PAAM) '96, London* (Apr. 1996), pp. 169–179.

[12] Foner, L. N. A multi-agent referral system for matchmaking. In *Practical Applications of Intelligent Agents and Multi-Agents (PAAM) '96, London* (Apr. 1996), pp. 245–261.

[13] Furuta, R., Shipman, F. M., Marshall, C. C., Brenner, D., and Hsieh, H. W. Hypertext paths and the world-wide web: Experiences with walden's paths. In *Proceedings of Hypertext '97, Southampton, U.K.* (1997), pp. 167–176.

[14] Genesereth, M. R., and Ketchpel, S. P. Software agents. *Communications of the ACM 37* (1994), 48–53.

[15] Hall, W. Ending the tyranny of the button. *IEEE Multimedia 1*, 1 (1994), 60–69.

[16] Hill, G., Hutchings, G., James, R., Loades, S., Halé, J., and Hatzopulous, M. Exploiting serendipity amongst users to provide support for hypertext navigation. In *Proceedings of Hypertext '97, Southampton, U.K.* (1997), pp. 212–213.

[17] Huberman, B. A., and Kaminsky, M. Beehive: A system for cooperative filtering and sharing of information. Tech. rep., Dynamics of Computation Group, Xerox Palo Alto Research Center, Aug. 1996. Available as `http://www.parc.xerox.com/spl/groups/dynamics/www/beehive.html`.

[18] Huhns, M. N., and Singh, M. P. Agents on the web. *IEEE Internet Computing 1*, 5 (Sept. 1997), 78–79.

[19] IBEX Corporation SA. *ITASCA Distributed Object Database Management System. Release 2.3.5, Technical Summary.* F-74160 Archamps, France, 1995.

[20] Irish, P. M., and Trigg, R. H. *Supporting Collaboration in Hypermedia: Issues and Experiences.* The Society of Text. Hypertext, Hypermedia, and the Social Construction of Information. The MIT-Press, 1989, pp. 90–106.

[21] Joachims, T., Freitag, D., and Mitchell, T. Webwatcher: A tour guide for the world wide web. In *International Joint Conference on Artificial Intelligence (IJCAI-97), Nagoya, Aichi, Japan* (Aug. 1997).

[22] Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM 40*, 3 (1997), 77–87.

[23] Krulwich, B., and Burkey, C. The infofinder agent: Learning user interests through heuristic phrase extraction. *IEEE Expert Intelligent Systems & their applications* (Sept. 1997).

[24] La Macchia, B. *Internet Fish.* PhD thesis, MIT Artificial Intelligence Laboratory. Report No. 1579, Cambridge, MA, 1995.

[25] Lieberman, H. Letizia: An agent that assists web browsing. In *International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada* (Aug. 1995).

[26] Maes, P. Interview on software agents: Humanizing the global computer. *IEEE Internet Computing 1*, 4 (July 1997).

[27] Nwana, H. S., and Ndumu, D. T. An introduction to agent technology. *BT Technology Journal 14*, 4 (Oct. 1996), 55–67.

[28] Petrie, C. J. What's an agent... and what's so intelligent about it? *IEEE Internet Computing 1*, 4 (July 1997).

[29] Resnick, P., and Varian, H. R. Recommender systems. *Communications of the ACM 40*, 3 (1997), 56–58.

[30] Scott, M. Wordsmith tools lexical analysis software for data driven learning & research. Tech. rep., The University of Liverpool, 1997. Available as `http://www.liv.ac.uk/~ms2928/wordsmit-.htm`.

[31] Shardanand, U., and Maes, P. Social information filtering: Algorithms for automating "word of mouth". In *Conference on Human Factors in Computing Systems (CHI95). May 7 - 11, 1995 Denver, Colorado, USA* (1995), pp. 210–217.

[32] Shneiderman, B., and Maes, P. Direct manipulations vs. interface agents. *Interactions 4*, 6 (Nov. 1997).

[33] Sidler, G., Scott, A., and Wolf, H. Collaborative browsing in the world wide web. In *Proceedings of the 8th Joint European Networking Conference, Edinburgh, Scotland, May 12-15* (May 1997).

[34] Starr, B., Ackerman, M. S., and Pazzani, M. Do-I-Care: A collaborative web agent. In *Conference on Human Factors in Computing Systems (CHI95). April 13 - 18, 1996, Vancouver, Canada, USA* (1996).

[35] Terveen, L., Hill, W., Amento, B., McDonald, D., and Creter, J. Phoaks: A system for sharing recommendations. *Communications of the ACM 40*, 3 (1997), 59–62.

[36] Wooldridge, M., and Jennings, N. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 2 (June 1995).

[37] Zellweger, P. T. Scripted documents: A hypermedia path mechanism. In *Proceedings of Hypertext '89, Pittsburgh, Pennsylvania* (1989), pp. 1–14.