Ubiquitous Web Information Agents

Gerd Beuster, Bernd Thomas, Christian Wolff
{gb,bthomas,derwolff}@uni-koblenz.de

Institute of Computer Science
Universität Koblenz

Abstract. This paper gives a brief overview about the *AI* methods and techniques we have developed for building ubiquitous web information systems. These methods from areas of *machine learning, logic programming, knowledge representation* and *multi-agent systems* are discussed in the context of our prototypical information system *MIA*. *MIA* is a web information system for mobile users, who are equipped with a *PDA* (Palm Pilot), a *cellular phone* and a *GPS* device or *cellular WAP phone.* It captures the main issues of ubiquitous computing: *location awareness, anytime* information access and *PDA technology*.

1 Introduction

Nowadays, the biggest but also the most chaotic and unstructured source of information is the World-Wide-Web. Making this immense amount of information available for *ubiquitous computing* in daily life is a great challenge. Besides hardware issues for wireless ubiquitous computing, that still are to be solved (wireless communication, blue-tooth technologies, wearable computing units, integration of GPS, PDA and telecommunication devices) one major problem is that of intelligent information extraction from the WWW. Instead of overwhelming the mobile user with documents found on the web, we want to offer her the short precise piece of information she is really interested in. For example, if she is on the road, looking for a restaurant, a good information system will present the addresses of the nearby restaurants retrieved online from the web that match her preferred cuisine. Certain conditions must hold for such a system: it must be aware of the user location, it has to react directly to the user's requests, it should present information found so far whenever the user asks for it. Furthermore the information should be extracted online from the web without requiring special prepared web pages (like pages with special tagging or additional services). For these tasks, we have developed methods from various artifical intelligent (AI) areas, like *machine learning*, *logic programming*, *knowledge representation* and *multi-agent systems*.

This paper we will give a brief overview about AI methods and techniques for building ubiquitous web information systems. Prototypically we have tested these methods in our system *MIA-The Mobile Information Agent*. The main intention of this paper is: 1) to show that it is already possible to implement ubiquitous web information systems. 2) that AI techniques are very suitable to solve the emerging problems.

2 The MIA System

MIA is a multi-agent [15] based information system focusing on the retrieval of short and precise facts from the web, at anytime with fast query response times. It monitors the position of the mobile users and autonomously updates the subject of search whenever necessary. Changes may occur when the user travels to a different location, or when she changes her search interests. This section is organized as follows: Section 2.1 describes briefly the three basic components of MIA. A more detailed introduction to the *AI* methods used in the spider agent is given in Section 2.2. We conclude this section with an example *MIA* session.

2.1 Architecture

The *MIA* system is separated into three basic components (Figure 1) which are: *Mobile Agents*, *Server* and *Multi-Agent System*.

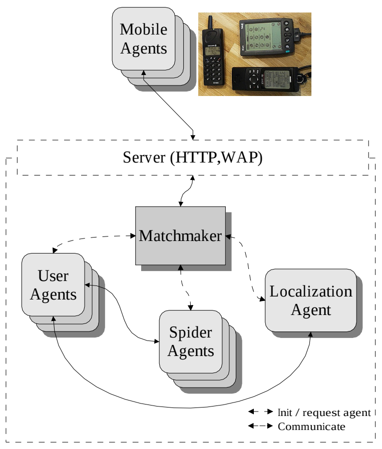


Fig. 1. system architecture

The Mobile Agent: A mobile computing device, the user is equipped with, that allows to estimate its geographical position and to communicate wirelessly with a server located on the internet. Currently we use a PDA, GPS-receiver and mobile phone. We are developing a mobile agent that uses only a mobile phone with WAP support. So far, the system knows two ways to get the information about the current position of the user: It can read out a GPS-device, or the user can add her position manually. Although we want to have an automated way to get the geographical position of the user, using GPS has some serious shortcomings: The GPS does not work within closed rooms, and it does not work too well in places with lots of buildings around. This is especially inconvenient because the user will be mainly interested in using the system in populated areas like inner cities, where we have only bad perception of GPS signals. There is an elegant way to overcome this problem. The mobile phone knows about the “cell“ it is currently connected to. We are going to use this information to determine the position of the user (Section 4).

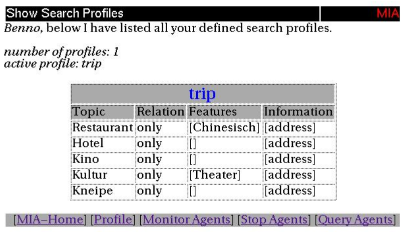
Currently the *MIA* prototype works with GPS support and manually given position information. The geographical coordinations delivered by the GPS device are resolved by the localization agent.This agent also keeps track of the user and estimates nearby cities.

The Server: The gateway to the core of the multi-agent based retrieval system. The *HTTP* protocol is used to communicate *HTML* and *WML* pages with the mobile agents.

The Agents: The core of the information system. It consists of several interacting agents. The current prototype works with three different agent types: *user agent* (user modeling and monitoring), *localization agent* and *spider agent* (intelligent web search and retrieval).

To find information relevant at the current geographical position in regard to the interests of the user we have to distinguish between two different types of information we have to search the web for: the *Topic* and the *Extract*. The *Topics* are web pages containing information related to the user’s interests and current city (e.g. Chinese restaurants in Heidelberg). The *Extract* is information about the topic itself, e.g. addresses, time-tables or descriptions. Descriptions about topics and associated extracts have to be present in each user-model, the *profile*.

Many web sites are structured hierarchically, from the generic to the specific. MIA's method to define user profiles captures these hierarchical structures. These structures play an important role in the spider algorithm, as we will see in Section 2.2. We model the topic by constrained keyword lists. That is, given a keyword, a set of constraint operators *only*, *not*, *one of*, and further constraint keywords, we define topics like: *restaurant only chinese*. The extract can be chosen from a predefined set due to the information extraction capabilities. Currently the *MIA* prototype supports the *only* operator and the *address* extract. *MIA* allows a user to have several personalized profiles, where each profile may consist of several topics. Figure 2.1 shows one profile titled *trip* consisting of five topics where each topic is assigned the extract *address*. Furthermore the user can change her profile whenever she wants to, even during information retrieval. The user agent is responsible for propagating changes in the current user profile to all agents.



Benno, below I have listed all your defined search profiles.

number of profiles: 1
active profile: trip

trip			
Topic	Relation	Features	Information
Restaurant	only	[Chinesisch]	[address]
Hotel	only	[]	[address]
Kino	only	[]	[address]
Kultur	only	[Theater]	[address]
Kneipe	only	[]	[address]

[MIA–Home] [Profile] [Monitor Agents] [Stop Agents] [Query Agents]

Fig. 2. Profile Configuration

2.2 AI Methods for Intelligent Web Information Retrieval

This section gives an overview about the artificial intelligence based techniques used by the *spider agent* to find, classify and extract information from the web.

Spidering The first step of the information retrieval process is to find web pages containing information about a special topic (Section 2). In contrast to existing search engines, our approach searches for relevant web pages online. Thus we call this approach *spidering*, instead of indexing web documents due to estimated word frequencies and building indexed databases. We determine the relevancy of a web document by its *context*, given by its relation to other pages. Our approach is as follows: We interpret the WWW as a directed graph where web pages are nodes and hyperlinks are vertices labeled with a URL and an associated text. Assume we are given some keywords and some start pages (entry points). We say that we have found a relevant web page p_x due to the keywords if, starting at one of the start pages p_0 we can find a path $P = (p_0, l_1, p_1, l_2, \ldots, l_x, p_x)$ with nodes p_i and vertices l_i labeled with the URL and associated text, such that: 1) every vertex label of this path contains at least one keyword 2) every keyword must at least occur once in one of the vertices labels. This basic idea can be improved as follows: If we are searching for a site containing information about Chinese restaurants in Heidelberg, and we follow a path where l_1 (the label of the first edge in this path) contains "heidelberg", l_2 to l_{10} contain "restaurant" and l_{11} contains "chinese", this last link is very unlikely to have still something to do with "heidelberg", although "heidelberg" occurred somewhere on this path. Therefore we introduce a parameter *MKD* the "maximum keyword distance" defining a context radius for the given keywords. In a similar way, we cut off the search paths if we depart too far from the last occurrence of a keyword: If a keyword has not occurred in the last *KN* labels in our path, we will not follow this path any further. This parameter *KN* is called the "kickout number". Hence we can define our approach as follows: Given M a set of entry points and K a set of keywords. Starting with a $p_0 \in M$ we return a link l_x iff there exists a cycle free path $P = (p_0, l_1, p_1, l_2, \ldots, l_x, p_x)$ with nodes p_i (web pages) and vertices l_j (hyperlinks connecting the pages) such that:

- $\forall l_j : 1 \leq j \leq x \quad \exists w_m \in K : w_m$ is contained in the label of l_j
- $\forall w_m \in K \quad \exists l_j : x - MKD \leq j \leq x : w_m$ is contained in the label of l_j
- $\forall w_m \in K \quad \forall l_k : KN \leq k \leq x \quad \exists l_j : k - KN \leq j \leq k : w_m$ is contained in the label of l_j

We can find such paths with a modified *best first search* strategy by using the mentioned keyword distance as the parameter for a cost function similar to standard informed search algorithms [11]. We look at the according paths of each link to decide which link of a set of links is the one nearest to a possible goal and chose the link whose path has the minimal keyword distance.

Example: A spider agents searches with the two keywords "restaurant" and "chinese" and has two links $l_{1,i}$ and $l_{2,j}$ it would follow next. Looking at the paths $P_1 = (p_{1,0}, l_{1,1}, \ldots, l_{1,i})$ and $P_2 = (p_{2,0}, l_{2,1}, \ldots, l_{2,j})$ it sees that "restaurant" occurred last time in $l_{1,i-1}$ and $l_{2,j-1}$, whereas "chinese" was in $l_{1,i-2}$ and $l_{2,j-3}$. The "keyword distances" to the first link are less and the agent will prefer to follow $l_{1,i}$ next.

Web Page Classification Adding a text classification component improves both the effectiveness and the efficiency of the system. About 96% of the web pages provided by our spider algorithm do not contain addresses. If we can filter out (parts of) these irrelevant web pages, we can increase the speed of the system significantly. The efficiency is improved, because the information extraction component gets some more information which it can facilitate in its task to extract information from a web page. So far, our information extraction agent can extract addresses from a web page. For this, it has a variety of methods available. Some methods are rather "strict" (extraction sometimes fails, although address information is available), some are rather "loose" (wrong positives on pages that do not contain addresses). With the additional information from the classifier, the information extraction component can make a better decision which method to use. If the classifier gives a high confidence in the quality of the web page, the information extraction agent will also try "loose" methods when the "strict" methods fail. If it is not likely that a web page contains addresses, the information extraction agent will not use "loose" methods but simply abandon the page.

We use Artificial Neural Networks for text classification. The network architecture we have chosen is a fully connected back-propagation feed-forward network [12] with 100 input nodes, 50 hidden nodes, and 2 output nodes. We map the web pages onto the input nodes in the following way: With each of a selection of 100 words, an input node is associated. If a web page contains one of these words, the corresponding input node has 1-activation; otherwise, it has 0-activation. The output nodes represent the classification "web page contains an address" and "web page does not contain an address". In the training phase, the appropriate output node is set to 1, and the other output node is set to 0. In the recognition phase, the document is put in the class whose associated output node has the higher activation. The neural network is trained offline on 3000 pre-classified pages. These pages are authentic web pages gathered by the spider algorithm.

The network has been trained until the error did no longer decline on an independent set of another 3000 pre-classified pages. After the training, 98% of the pages from the training set are classified correctly. This excellent classification result is partly due to the distribution of web pages: About 95% of the web pages that are gathered by the spider agent do not contain addresses. On the positive examples, the one that contain addresses, we get a recognition rate of about 60%.

The 100 words that are used to give a representation of a web page have been chosen by selecting those 100 words from all the words appearing in the web pages from the training set which contain the most information [2], i.e. the words that allow best to distinguish between pages that contain address, and pages that do not contain addresses.

Machine Learning for Information Extraction To extract information from web pages it is either necessary to *understand* the document, which would involve semantic text analyzation techniques, or to find relevant information by recognizing its underlying syntactical representation. Because linguistically motivated attempts containing semantical and syntactical parsing fail on web documents (HTML documents), it makes more sense to use the special text formatting and annotating strings (*tags*) of these documents to recognize and extract relevant information.

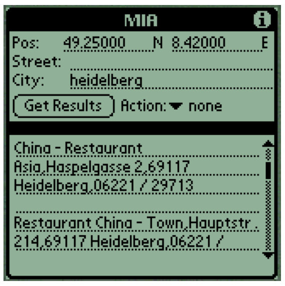
We use the syntax based approach of automatically learning extraction procedures (wrappers [16]) described in [5, 13, 14]. Similar approaches using machine learning techniques for the automatic wrapper construction are described in [1, 4, 7]. To extract information, we can assume special text parts to be delimiters marking the beginning and the end of the relevant information to be extracted. Thus the key idea of our learning algorithm is: Given a set of example extractions for a web page, e.g. some addresses from a set of addresses listed on the page, we collect the surrounding text parts (*anchors*) of each component of the example address (name, street, telefon, ...) and learn a general pattern for each of these *anchors*. Combining these learned patterns with some additional constraints results in a learned wrapper for the intended information extraction task. We can illustrate the general idea of learning wrappers for IE as follows: If a user reads a web page containing a list of addresses, she might think of a relation like: $address(Name, Street, Telefon)$. The instances of this relation are the information that can be found in the list shown on the particular web page (e.g. $address(Asia, Haspelgasse\ 2, 29713)$). Now we ask the user to determine some instances (text tuples) of this particular relation, which are contained in the list (e.g.<"Asia","Haspelgasse 2","29713">)). These instances will be our positive set of examples E^+. The learning task is to find a definition for the relation *address* such that all examples from E^+ and all other listed addresses are instances of the relation *address*. This means we have to learn an extraction procedure that is able to extract the instances in E^+ and remaining instances presented on the page. Therefore we use generalization techniques based on Anti-Unification [10, 6] in combination with methods adopted from the area of inductive logic programming [9]. For further details the reader is referred to [13, 5].

The major problem we are confronted with in the context of *MIA* is the lack of available examples. Because the classifier provides unknown web pages to the system, we cannot determine any examples, which are needed for learning. To overcome this problem we developed a learning algorithm that derives its examples by means of *knowledge* *representation* (*KR*) techniques. That is, we model our structural (syntactical) knowledge about addresses with a logical KR language and are able to query this knowledge base to derive examples that can be used as input to a modified learner. This allows us to learn wrappers even for unknown pages. We call this approach *learning from meta examples*. For the automatic construction of address wrappers, this technique shows very promising results.

2.3 Example MIA Session

Let us demonstrate the general behavior and functionality of *MIA* by a small example. Imagine you plan to take a trip, but you are still a bit unsure about the cities you will visit. During a journey you are interested in addresses of hotels, restaurants offering Chinee cuisine, cultural institutions like theaters, cinemas and pubs.At home you tell *MIA* your interests via its standard web browser interface (Figure 2.1), or with the web browser installed on your PDA.

Starting your journey, you select the mia application stored on the PDA and you activate your profile. From now on your mobile devices will contact the *MIA* server via the cellular phone and a dial-up internet connection and will transmit your current geographical position. This happens in preselected time intervals without any further user interaction. Once you initiated your profile and the server retrieves the first position updates, it tells the matchmaker about the request sent by the mobile user. The extended matchmaker decides to initiate a user agent and several spider agents to fulfill your request. If there are already agents active for you, it informs your user agent to keep track of changes of your location or of your search interests. Although search interests are usually defined before the trip, they can also be changed "on the road". Whenever you move your new location is reported to the user agent, who will communicate with the localization agent. The localization agent resolves the geographical coordinates to the name of the nearest city. Now the user agent will inform the active spider agents about the perceived changes. The spider agents will start to look for web pages about your topics of interest in your current city.



Pos: 49.25000 N 8.42000 E
Street:
City: heidelberg

China - Restaurant
Asia,Haspelgasse 2,69117
Heidelberg,06221 / 29713

Restaurant China - Town,Hauptstr.
214,69117 Heidelberg,06221 /

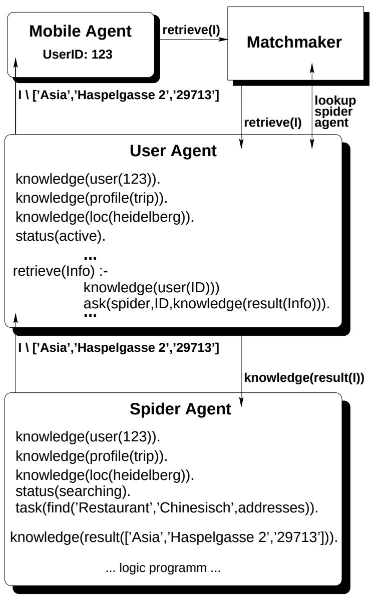
Fig. 3. PDA Results

After a while you drive through a very nice town. You decide to stay for some time. So it is time to ask *MIA* what it has found for you. All you have to do is: take your PDA, select your mia application and click on the button *get results* (Figure 3). The user agent is queried for the information it has found so far. It will ask all active spider agents for their results. Because every agent constantly listens if a request is sent to it, it immediately pauses its task and tries to answer the incoming request. The results found so far are returned directly to the user agent and finally to the mobile user's PDA.

3 Logic Programming Based Agent Technology

One key issue of *MIA*'s architecture is to use logic as programming language [8] for agents and as agent communication language. This technique allows agents to control and query other agents in a flexible and very dynamic way.

The initialization and assignment of agents is handled by the matchmaker. This is a special agent whose job is to match information searching and information providing agents. Thus it keeps track what agents are active, for which user and what their task is. We use an extended matchmaker concept that checks with simple subsumption techniques if incoming requests can be splitted and handled by already active agents. This technique is mainly used for the spider agents. It prevents the system from having more than one agent searching for the same topic.



```
knowledge(user(123)).
knowledge(profile(trip)).
knowledge(loc(heidelberg)).
status(active).
          ...
retrieve(Info) :-
          knowledge(user(ID)))
          ask(spider,ID,knowledge(result(Info))).
          ...
```

```
knowledge(user(123)).
knowledge(profile(trip)).
knowledge(loc(heidelberg)).
status(searching).
task(find('Restaurant','Chinesisch',addresses)).

knowledge(result(['Asia','Haspelgasse 2','29713'])).

          ... logic programm ...
```

Fig. 4. Agent Communication

The functionality (basic task) of each agent is described by a logic program. For example, one of these programs describe how to search the web. The knowledge acquired by the agent is added to its uniform knowledge base, the *interface base*. The interface base provides a common information exchange level for different types of agents. This is useful, because the agents do not necessarily know which rule of the other agent's program can be activated.

Whenever an agent receives a request from another agent, its actual basic task (proof) is interrupted and the incoming query is processed. Potentially, every rule of the agents program can now be *fired* by this query, but the common way is to define some uniform predicates (interface base) like *knowledge*, *status*, *task*, which describe the inner state of the agent, and information acquired. If an agent trusts a special agent, it can tell him which other services (rules of its built-in logic program) it can use in later queries. Another interesting point is to allow an agent to *add knowledge in form of rules* to an other agents logic program.

Each agent has a fixed set of *communication rules* which are similar to KQML [3], for example *ask(AgentType, UserID, Query)*, *command(AgentType, UserID, Command)*, *assert(AgentType, UserID, Info)*, etc. Executing such an communication rule results in consulting the matchmaker if an agent of type *AgentType* is active for a user *UserID*. If this is the case, the agent will communicate directly with the active agent. If not, the matchmaker will initiate an appropriate agent, and the calling agent will send initial commands to the initiated agent.

One major advantage of using interruptable logic programming based agents is the possibility to provide *anytime* information access. That is, whenever an agent ist still *working*, but already found some information, we do not have to wait until it has completed its task. Instead, we are able to interrupt its basic task and query its *interface base* to retrieve the results found so far. Afterwards, we send him back to work. This is essential for a mobile information system, namely to provide the user anytime and as fast as possible with new information. Example (Figure 4): The user *123* sends an information retrieve request to the *MIA* system. This query is send from the matchmaker to the user agent *UA* for user *123*. Because every incoming request is interpreted as an proof obligation, the *UA's* rule: *retrieve(I) :- knowledge(user(ID)), ask(spider, ID, knowledge(result(I)))* will be used to proof the query *retrieve(I)*. Applying this rule results in executing the communication rule *ask(spider, 123, knowledge(result(I)))* which results in asking the matchmaker for active agents of type spider for user *123*. In our small example, we assume that there is a spider agent active for the user *123* searching for addresses of Chinee restaurants in Heidelberg, and that its interface base consists of one address found so far: *knowledge(result(['Asia', 'Haspelgasse 2', '29713'])).* Thus the query *knowledge(result(I))* sent to it results in the computed answer: *knowledge(result(['Asia', 'Haspelgasse 2', '29713'])).* Finally the *UA* has fulfilled its proof obligation and sends the answer *retrieve(['Asia', 'Haspelgasse 2', '29713'])* to the mobile agent.

4 Conclusion & Future Work

We showed in brief detail (Section 2.2) how three major *AI* disciplines, namely *logic programming*, *machine learning* and *multi-agent systems* can be combined to build ubiquitous information systems. A general agent architecture based on interruptable logic programming techniques to implement *anytime* info access has been presented in Section 3. We gave a short introduction how to use existing hardware, namely a *Palm Pilot*, a *GPS* system and a cellular phone to build an mobile computing unit providing location awareness, anytime information access and wireless communication. We also pointed out the problems concerning the GPS position estimation and the inconvenience of the prototypical setup consisting of three separated devices. These leads to our future plans (Section 4.1).

One important point to emphasize is the capability of *MIA* to retrieve information from unmodified web pages. We are not limited to special search domains, and we can use the whole web as information database. So our system provides access to upto date online information. Though we describe an ubiquitous system here, *MIA* can also be used as an city information system for the stationary user with a PC at home.

4.1 Future Work

Currently we extend the pure keyword based web search algorithm (Section 2.2 by deductive reasoning processes that derive related terms of a keyword from an ontology. Constraint keyword lists in combination with deductive ontology bases allows us to enrich the retrieval process with domain specific knowledge. A topic like *restaurants not chinese* can then be deductively resolved to queries like *restaurants french or italian*.

So far, we have only used offline training for web page classification (Section 2.2). In the future, we will also train the networks online, creating truly *adapting agents*. For online training, the feedback of the *extraction agent* is used to adapt the network. The web page, together with the result from the extraction agent, is used as a new teaching input pattern for the neural network.

We have started to work on an elegant way to overcome the problems related with a GPS based localization (Section 2.1). The cellular phone provides information about the cell it is connected to. A cell is the area served by one of the fixed basis stations of the telephone network. But using the information about the basis station the phone is connected to, and a database about the geographical position of the basis stations, we can determine the position of the user fairly accurately. By this method, we do not get the high resolution of GPS (about 5 meters), but this does not really harm the system, because we can not use the high resolution of GPS anyway, since the resolution of reasonably priced maps is significantly lower than the resolution of GPS.

At the moment we are switching from HTML to XML. This will allow us not only to serve HTML pages, but also pages in other formats. Most importantly, we will be able to serve pages in WML, the standard document format for WAP-devices.

Acknowledgment
The work presented in this paper is funded by *Stiftung Land Rheinland-Pfalz für Innovation, Land Rheinland Pfalz, Ministerium für Wirtschaft, Verkehr, Landwirtschaft und Weinbau*, project *IWIA - Intelligent Web Information Agents*[1]

References

1. M. E. Califf. *Relational Learning Techniques for Natural Language Information Extraction*. PhD thesis, University of Texas at Austin, August 1998.
2. T. M. Cover and J. A. Thomas. *Elements of information theory*. Wiley, New York, 1991.
3. T. Finin, Y. Labrou, and J. Mayfield. *Software Agents*, chapter KQML as an agent communication language. MIT Press, Cambridge, 1997.
4. D. Freitag. *Machine Learning for Information Extraction in Informal Domains*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, November 1998.
5. G. Grieser, K. P. Jantke, S. Lange, and B. Thomas. A unifying approach to html wrapper representation and learning. Technical report, Institut für Informatik, 2000. to appear.
6. K. Knight. Unification: A multidisciplinary survey. *ACM Computing Surveys*, 21(1):93–124, March 1989.
7. N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington, 1997.
8. J. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2 edition, 1987.
9. S. Muggleton and L. D. Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 1994.
10. G. Plotkin. *Automatic Methods of Inductive Inference*. PhD thesis, University of Edinburgh, 1971.
11. D. Poole, A. Mackworth, and R. Goebel. *Computational Intelligence, A Logical Approach*. Oxford University Press, 1998.
12. D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, volume 1, Foundations. MIT Press, 1986.
13. B. Thomas. Anti-unification based learning of T-Wrappers for information extraction. In *Workshop on Machine Learning for Information Extraction*, July 1999. 16th National American Conference on Artifical Intelligence (AAAI-99).
14. B. Thomas. Token-Templates and Logic Programs for Intelligent Web Search. *Intelligent Information Systems*, 14(2/3):241–261, March-June 2000. Special Issue: Methodologies for Intelligent Information Systems.
15. G. Weiss, editor. *Multiagent Systems*. MIT Press, 1999.
16. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, March 1992.

[1] $http:\backslash\backslash www.uni\text{–}koblenz.de\backslash\sim bthomas\backslash mia.html$