# Agent Communication Inside an Internet Search Engine

M. López-Sánchez, F. Martín, J. García, X. Canals, X. Drudis, N. Ruiz, and A. Reyes[+]

[+] iSOCO: Intelligent Software Components S.A.
Edif. Instituto de Investigación en Inteligencia Artificial (IIIA), Consejo Superior de Investigaciones Científicas (CSIC).
Campus Universidad Autónoma de Barcelona 08193 Bellaterra Barcelona. Spain
email: {maite, martín,juli,xavix, nax, toni}@iSOCO.com

## Abstract

This paper describes some details about the architecture of a fully implemented search engine for the Internet. Its architecture is based on autonomous software agents and the paper is focused on the communication among them. Agents collaborate to gather HTML pages from de the world wide web and treat them in order to be able to retrieve those pages from subsequent users' queries. Crawling Agent collaboration is required in order to decide the URLs that should be first retrieved. Subsequent page treatment consists on first filtering the pages so that HTML format is transformed into XML and second indexing them so that information retrieval can be performed online.

**Keywords:** Search Engine, Communicating Agents, Information Retrieval, Multi-Agent Architecture.

## 1. Introduction

Internet has provided users with such an enormous amount of information that it becomes necessary to provide tools that help in discriminating the required information. For some years up to now, agents have been proposed as a way of approaching some of the problems related to dealing with information on the World Wide Web (in this sense [Etzioni 93, 97] described softbots as intelligent agents that use software tools and services on a person's behalf). This paper describes the communication among agents at i-Bot, a search engine dedicated to the web.

i-Bot is provided with an agent-based architecture, which is best explained in terms of its components (see Figure 1):

- *Crawling Agent Community*: it can be described as a group of crawling agents named bots (also known as spiders or robots) that are dedicated to download HTML (HyperText Mark-up Language) pages from the Web.
- *URL Broker Agent*: this agent manages the information about the web and provides the bots with URL's (Uniform Resource Locator) to retrieve.
- *Filtering Agent Community*: Each time a bot downloads an HTML page, a filter agent takes it and extracts its contents. The agent filters the resulting text and generates an XML (Extendable Mark-up Language) page whose structure is suitable for being indexed.
- *Indexer Agent*. Once a number of pages have been filtered, this agent groups the XML pages before indexing them. The Indexer agent ends up with an index structure suitable for retrieving purposes.

- *Query Agent Community*. Several query agents can use the index to retrieve those pages that best fit users' queries. That is, to answer them.
- *Interface Agent Community*. This module handles the user interface: it takes the user query and displays the pages that the Information Retrieval System returns when answering the users' queries.

Next sections describe the information that each one of these agents handles and how do they communicate. Nevertheless, due to space restrictions, only sections 2 and 3 describing crawler and URL broker agents have been detailed. We end the paper by comparing our agent-based approach to other search engines and by providing some conclusions.

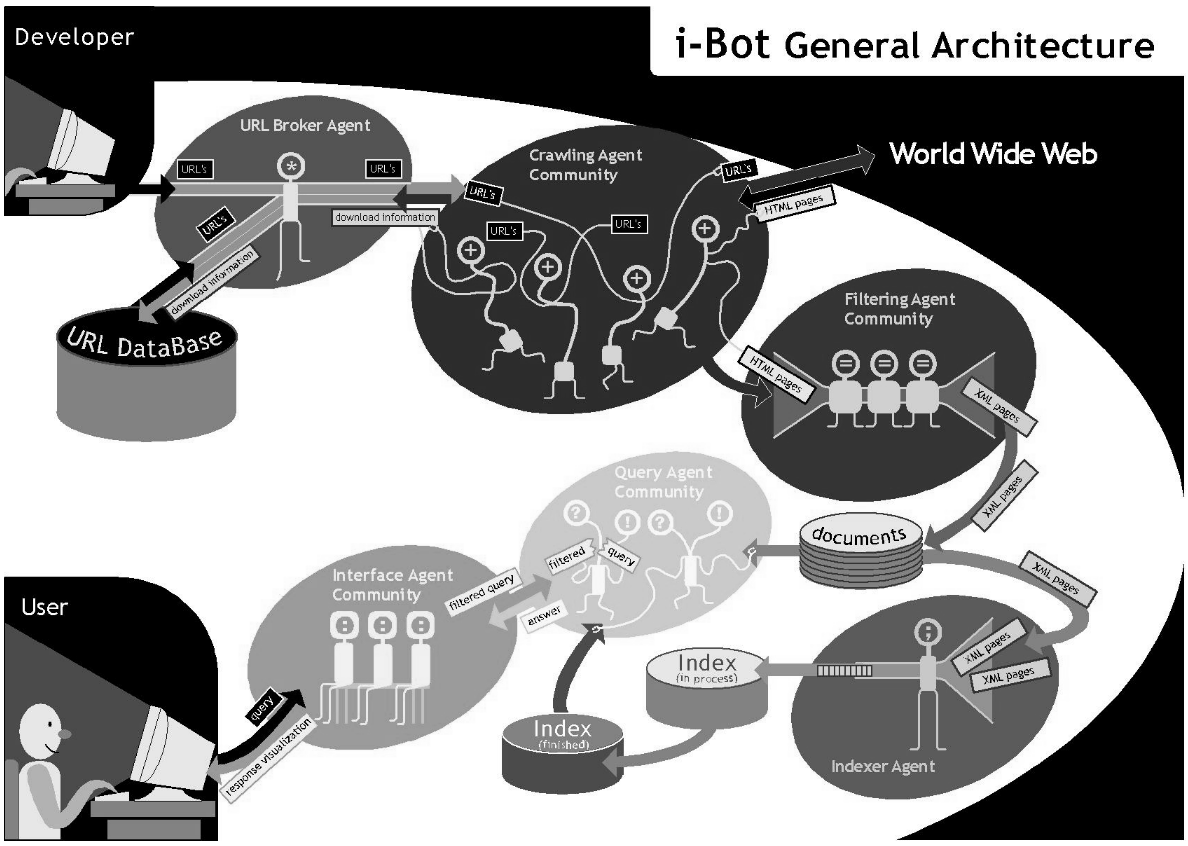

Figure 1: iBot (Web search engine) Architecture.

## 2. Crawling Agent Community

The crawling agent Community is a group of crawling agents, known as bots, that are dedicated to download HTML pages from the Web. Bots are not mobile agents (that migrate from machine to machine executing their code) but keep themselves into the community location. Their task consist on downloading (i.e., crawling) HTML pages as quick as possible and to store them in an easy-to-retrieve structure.

Downloaded HTML pages are stored with a name (which corresponds to the coding of the URL of the page: url_ID) in a tree structure of directories, which happens to be perfectly balanced (that is, the assignment of URL identifiers follows a uniform distribution). Nevertheless, not all the successfully downloaded pages are necessarily stored: the bots can decide whether a page must or must not be stored based on the language it is written on.

Each bot is an autonomous agent that decides when to ask the URL broker agent new URL's to retrieve. And once it has processed and stored the corresponding web pages, it provides the URL broker agent with the links it has extracted form the retrieved pages.

Bots are implemented in ansi C, and use the Hyper Text Transfer Protocol (HTTP) both to communicate with the URL broker agent and to request HTML pages to the Internet servers.

## 2.1 Agent Communication: bot - URL broker

The communication cycle established between each bot and the URL broker consists of three different connections:

- Bot authentication. Before any bot can send any information to the URL broker agent, it is required to be authentified. This process enables the bot to have an identifier (bot_ID) assigned so that it can use this for further communications.
- URL request. Once a bot has been authentified, it asks the URL broker agent for a number N of URL's. The URL broker uses the bot identifier to check whether the request is done by an authentified bot or not. In case the check is positive, the URL broker returns to the bot N pairs of (url_ID, URL), where url_ID are the codes for the corresponding URL's. It is possible to establish a downloading policy that avoids downloading a page if the date at the Last-Modified value of the HTTP Header is bigger than the date at which this page was previously downloaded. In this case the URL broker agent is required to include the date of the previous successful page download in the protocol.
- URL information. For each downloaded page, the robot returns information to the URL Server. This information consists of the
    - bot_ID,
    - url_ID (which acts as page identifier),
    - the HTTP response code (that is, if the page was properly downloaded, if there was a server error..),
    - the language of the page, and
    - a list of URLs that correspond to the links that appear in the downloaded page.

  If all the information is read properly, the URL Server answers an OK message.

We use the conversation protocol specification introduced by [Martin00] to describe these connections in the following subsections.

### 2.1.1 Bot authentication

Each bot asks the URL broker for an identifier (bot_ID), wich is computed from a combination of its name and its PID (process identifier). These bot_ID's are used for subsequent communications. The following figure 2 shows the communication established by the bot.

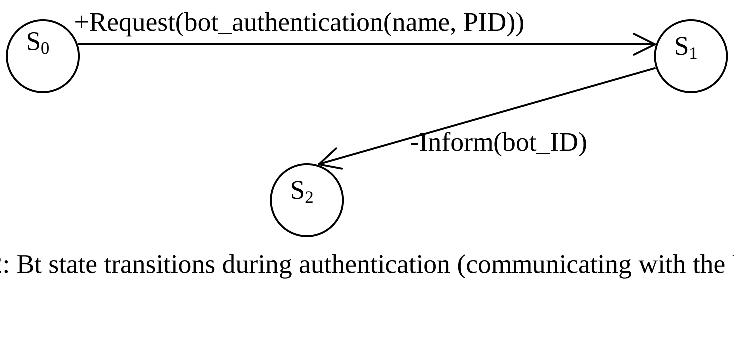

Figure 2: Bt state transitions during authentication (communicating with the URL Broker agent).

### 2.1.2 URL request

As depicted in Figure 3, an authentified bot asks the URL broker N URL's to download (the bot must include its bot_ID in its request). As response, the URL broker returns N tuples of (url_ID, URL, last_down), where url_ID is the code for the corresponding URL and last_down specifies the date this URL was last downloaded.

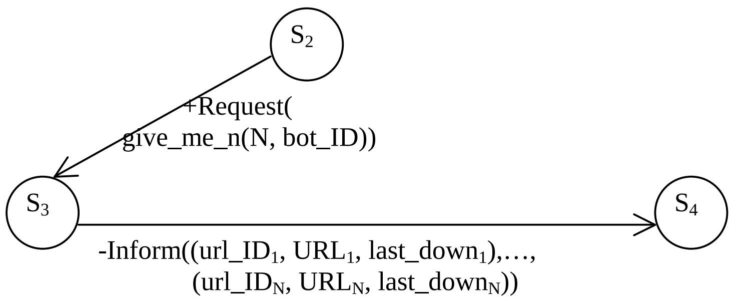
S2
+Request(
give_me_n(N, bot_ID))
S3
S4
-Inform((url_ID1, URL1, last_down1),…,
(url_IDN, URLN, last_downN))

Figure 3: Bot state transitions during URL requests (communicating with the URL Broker agent).

### 2.1.3 URL information

Bots return a tuple for each downloaded page to the URL broker (see figure 4). The tuples are of the form (bot_ID, url_ID, httpd_resp, lang, (URL1,URLm)) where m corresponds to the number of links that have been obtained from a given page i. The bot waits afterwards for the URL broker to answer a confirmation message.

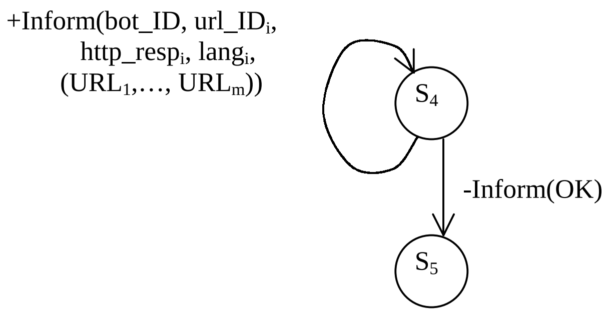

Figure 4: Bot state transitions for providing with URL information the URL Broker agent.

## 2.2 Bot-web-server communication

Bots connect by means of sockets to the web servers and use the Hyper Text Transfer Protocol (HTTP) 1.0 to request HTML pages (see figure 5). The method to perform the request is a GET method. A URL is composed by the server, a port number, and the resource (for example: if the URL is http://www.isoco.com:80/people.html, the socket is open towards the server is www.isoco.com and the port is 80. Afterwards, resource /people.html is used to ask of the specific HTML document).

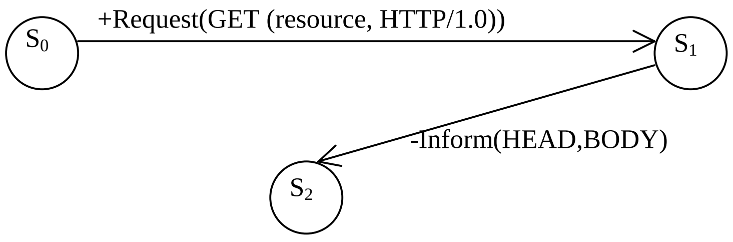

Figure 5: Bot state transitions for requesting web pages to a given web server.

## 3. URL Broker Agent

URL Broker Agent (developed in Java) manages the information about the web and communicates with the bots at the crawling community in order to keep this information updated. Policies for deciding which pages will be downloaded first are applied by the URL Broker agent. iBot is specialised on retrieving HTML pages totally or partially written in Spanish so that it gives more priority to those links extracted from pages that have been detected as Spanish by the language filter. (We have developed a filter based on the presence of a combination of words and features that are characteristic of Spanish but that in their combination exclude other languages that could contain some of these individual words).

### 3.1 URL Broker Information

In order to manage the information about the web as well as to answer bots' requests, the URL Broker agent uses a Data Base dedicated to store information about both, web pages and bots.

#### 3.1.1 URL Stored Information

- *Host*: the location of the internet server that provides the page that each URL specifies.
- *Port*: the "logical connection place" to a particular server program the http request must be bind to. As in the port standard definition, port numbers are from 0 to 65536. For the HTTP service, port 80 is defined as a default and it does not have to be specified in the URL.
- *Resource*: defines the remaining part of the URL that locates the described page.
- *URL_id*: it encodes the URL that the previous columns define. Its value has been obtained by applying the Message Digest Algorithm [Rivest92].
- *Tsstate*: As its name suggest, it is a timestamp that records the date of insert or update operations (it is automatically set to the date of the most recent operation).
- *State*: represents the http response code obtained when a robot downloads the page. This response code takes values representing whether the page has been downloaded or not; if it has been downloaded, it specifies the http response code (successful responses, redirections, errors,…). Otherwise, it specifies the origin of the URL (the code represents the kind of page the URL was extracted from).
- Bot2_id: It's the identifier of the bot that provided the URL, that is, the bot that extracted the link from a downloaded page.
- *Inlinks*: it accumulates the number of links the bots have found to be pointing to the specific page. In this manner, each time a bot downloads a page, extracts its links and sends them to the URL Broker agent, the broker treats each of these links separately: if the link does not exist, the broker adds it to the database as a new URL. Otherwise, it increments by one the number of inlinks.
- *Outlinks*: similarly to Inlinks, it stores the number of links a page contains. Its value is set at once, when a page is downloaded and its links extracted. It is worth noticing that we only consider links to HTML pages (ended in ".html",".htm",or "/").

- *Dinit*: its name stands for initial date and corresponds to the date the URL was obtained for the first time. This allows to keep track of the period of time that a page lasts on the Internet (or at least, the time that passes since its first reference was found).
- *Dlast*: similarly to Dinit, Dlast represents the date of the last time that the page was successfully downloaded. This information can be used to treat downloading errors and apply different policies of retrying their downloading. Having the time when it was successfully downloaded, it is possible to compare the time gap with the Tsstate information (which contains the last time this URL was treated). Therefore, if the difference is big, this means that the page downloading has been returning error state codes for a long time. Consequently, we could decide to delete the URL from the Data Base.
- *Language*: the language of the page is assigned when the language filter is applied on the page. This filter distinguishes between Spanish and non-Spanish pages. A page is considered to be Spanish whenever there is some Spanish on it (that is, we consider multilingual pages as Spanish whenever Spanish is one of the language).
- *Bot1_id*: the identifier of the bot to which the URL was given. This information has been included in order to check that a bot is providing the information of a page whose URL was given to it.

### 3.1.2 Bot Stored Information

- *Name*: This name is used to obtain the bot identifier, and is only provided by the bot for the first time the bot establishes contact with the URL Server. Bots' names in i-Bot are assigned by concatenating the name of the bot generation together with the PID (Process IDentifier) that each running instance of bot has got.
- *Bot_id*: Bot identifier value is obtained from applying the Message Digest Algorithm [Rivest92] over the name of the bot.
- *Dinit*: corresponds to the date the bot was authentified.
- *Dlast*: similarly to Dinit, Dlast represents the date of the last time that the bot communicated with the URL Server. This allows to keep track of evolution of the robots, and we can decide that a robot can be sent out of the system if it has being a long time without communicating with the URL Server.
- *Requests*: records the number of requests that a bot has performed. This information is stored in order to be able to monitorise bots' performance. The underlying idea is to be able to send a bot a kiss when it is performing poorly.

## 3.2 Agent Communication: URL Broker – bot

In the previous section (see 2.1), we have seen the protocol the bots use to communicate with the URL broker agent from the bots' point of view. We see now the same processes, but from the URL broker's side.

### 3.2.1 Bot Authentication

As figure 6 shows, whenever a bot asks the URL broker for an identifier, it returns a bot_id.

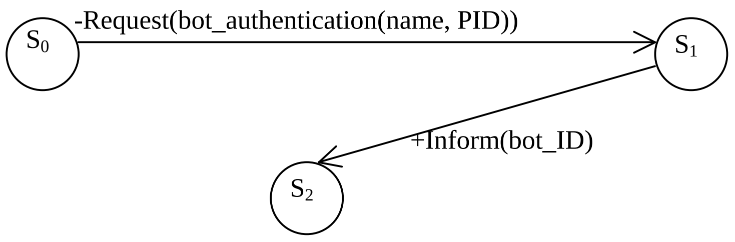

Figure 6: URL broker agent state transitions for bot authentication (communicating with a bot).

### 3.2.2 URL Response

When a bot asks the URL broker agent a number of URL's, the URL broker must check if this bot has been previously authentified. And, once this has been done, it selects from the URL Data Base the *N* URL's the bot is asking for. This process is depicted by the next figure:

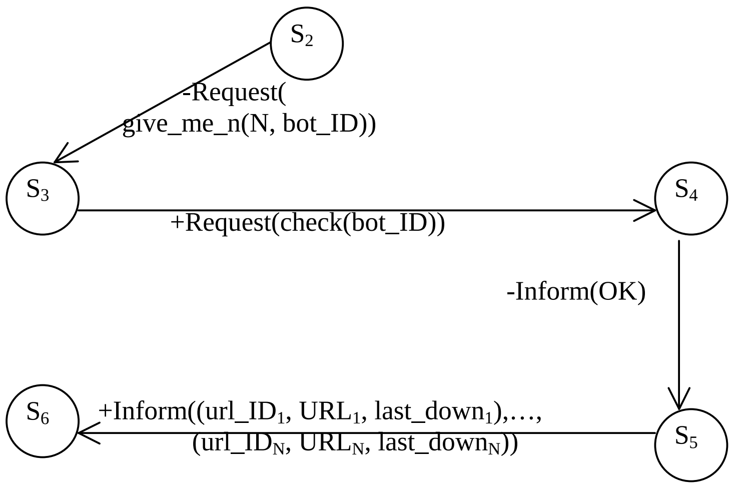

Figure 7: URL broker agent state transitions for responding URLs to a bot.

### 3.2.3 URL Information

Bots return a tuple for each downloaded page to the URL broker agent. The tuples are of the form (bot_ID, url_ID, httpd_resp, lang, ($URL_1$,$URL_m$)) where m corresponds to the number of links that have been obtained from a given page i.

Again, the URL broker agent must confirm the existence of the bot_ID and, once this has been checked, it updates the information about the affected URLs. Finally, if all the subsequent updates are successful, the URL broker agent answers the bot with a confirmation message (see Fig. 8).

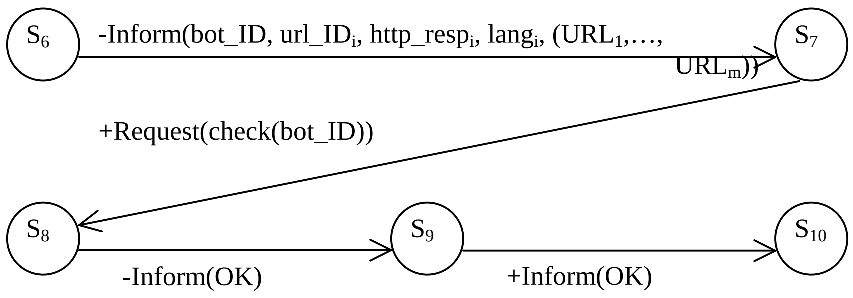

Figure 8: URL broker agent state transitions for receiving URL information from bots.

## 4. Filtering Agent Community

Each time a bot downloads an HTML page, a filter agent takes it and extracts its contents. The agent filters the resulting text and generates an XML page whose structure is suitable for being indexed. This structure contains the following information:

- URL of the page

- url_ID (the one given by the URL Server)
- Title (in ascii and in HTML)
- Keywords (in ascii and in HTML)
- Description (in ascii and in HTML)
- Text (in ascii and in HTML). Plain text is stored in two ways: keeping the HTML codification so that it can be properly visualised in a browser, and transformed into ascii (127 bits) in order to be indexed. For example, a word containing an accent 'más' is kept as 'm&aacute;s' in the HTML format and as 'mas' in ascii. In the same way, capital letters are kept in the HTML text and are stored as lowercase in the ascii text.

## 5. Information Retrieval Agent Community

Information Retrieval [Baeza99] tasks are performed by an heterogeneous community formed by an *Indexer Agent*, the *Query Agent Comm*unity and the *Interface Agent Community*.

Once a number of pages have been filtered, the *Indexer Agent* groups the XML pages before indexing them. The Indexing process [Witten99] consists on generating an index structure suitable for retrieving purposes (used to retrieve those pages that best fit users' queries). This structure contains:

- The Lexicon (a list of all the words appearing in the collection), which also contains, for each word, the number of documents in the collection that contain it, and its total number of appearances.
- The Inverted File, which stores for each word a list of <d,$f_{d,t}$> pairs, where d is the identifier of each document containing the word and $f_{d,t}$ is how many times it appeared in this text.
- A file that contains approximations of the weights that will be used when computing the relevance of documents regarding a given query.

After the building of the index structure, several *Query Agents* can use it to retrieve those pages that best fit users' queries. That is, to answer them.

Finally, the agents at the *User Interface Agent Community* handle the user interface. Each agent takes the user query, filters it and sends the question to a *Query Agent*. And, once it answers the XML pages that best fit users' queries, the *User Interface Agent* extracts the information to display and generates and HTML page that will be displayed by the browser.

## 6. Related Work and Conclusions

From all the world wide known search engines as Altavista, Yahoo or Inktomy, we distinguish Google [Brin99] and Fast [Fast98a] because of their strong academic background. The later has been distinguished because of their dedicated hardware [Fast98b], and the former because it introduced the Page Rank algorithm [Brion98]. All search engines have a similar architecture because the same basic modules are always required (i.e., crawling module, indexing, retrieval,..). Nevertheless our approach can be distinguished because of its multiagent architecture.

Regarding the specialization on Spanish web pages, there are related works in a wide range of specialization. In one side, there are general approaches describing policies of assigning priorities to certain URLs as the ones applied at webbase [Webbase99] (although, since it does not contain indexing capabilities, it is not a search engine but a web crawler). Whilst on the other side there are concrete policies, are as the one by [Chakrabarti99] that only considers pages related to specific topics. In this manner, iBot's policy is something in between: it indexes only Spanish pages but extracts the links from all pages so that the URL Broker agent is feed with all kinds of URLs and the search is kept as wide as possible.

This paper presents an approach to the communication required for agents in order to gather and retrieve HTML pages. These agents belong to iBot, a web search engine for Spanish web pages. A reduced version of iBot is publicly available at http://e-bot.isoco.com.

## References

[Baeza99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto “Moder Information Retrieval” (book) Ed. Addison Wesley. Acm press. 1999.

[Brin99] Sergey Brin and Lawrence Page “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. Research Report Computer Science Department, Stanford University. 1999.

[Brion98] Sebastien Brion, Shiv Ramamurthi and Kausal Shah “User PageRanks: Computing PageRanks based on user’s preferences”. Research Report Computer Science Department, Stanford University. December 6th 1998.

[Chakrabarti99] Soumen Chakrabarti, Martin Van den Berg, Byron Dom “Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery” The Eighth International World Wide Web Conference, May 11-14, 1999.

[Etzioni93] Orien Etzioni “Intelligence Without Robots: A Reply to Brooks”. AI Magazine 14 (4): 7-13. 1993.

[Etzioni97] Orien Etzioni “Moving Up the Information Food Chain: Deploying Softbots on the World Wide Web”. AI Magazine, 18 (2): 11-18.

[Fast98a] Fast Software Search (http://www.alltheweb.com or http://www.fast.no), white paper. Version 0.1, December 1998.

[Fast98b] Fast Pattern Matching Chip and the Fast Search Card. White Paper. December 1998.

[Martin00] Francisco J. Martín, Enric Plaza, Juan Antonio Rodriguez-Aguilar. “An infraestructure for Agent-Based Systems: An Interagent Approach”. International Journal of Intelligent Systems (John Wiley & Sons, Inc.), vol 15, pages 217-240, 2000.

[Rivest92] R. Rivest, “The MD5 Message-Digest Algorithm” Request for Comments: 1321 http://www.faqs.org/rfcs/rfc1321.html MIT Laboratory for Computer Science and RSA Data Security, Inc. April 1992.

[Webbase99] Webbase is an internet web crawler written in C publicly available at http://www.senga.org/webbase/html/ (this URL also contains a 30 pages long document that describes the system v.5.5.0 November 1999).

[Witten99] Ian H Witten, Alistair Moffat and Timothy C. Bell. “Managing Gigabytes, Compressing and Indexing Documents and Images” (book: second edition). Ed. Morgan Kaufman 1999.