J.T. Yao, V.V. Raghvan, G.Y. Wang (Eds.): WSS'04, pp. 20-27, 2004.

Semi-Structured Complex List Extraction

Anders Arpteg
Department of Technology
University of Kalmar
SE-39192 Kalmar, Sweden
anders.arpteg@hik.se

Abstract

The semi-structured information available in HTML and similar documents provide valuable information that can be used for information extraction applications. This information together with other technical information about how to retrieve pages can be used to automatically extract pieces and various types of lists. The goal is to put as much intelligently as possible in the system so that as little knowledge and work as possible is required by the users, i.e. a user-driven extraction system. The advantage of a user-driven system is that the service provided by the system is available not only for experts, but for also ordinary users and thereby making the service available for a wide audience.

A problem with some lists in documents are that the structure is different for the elements in the lists, and thus it becomes more difficult to take advantage of the semi-structural information. The agent-oriented system described in this paper allows a user without expert skills to train an extraction system to extract singleton, lists, and also complex lists. The complex list type shall be able to handle these complex lists with varied structure.

The experiments conducted show that a user can train the system to extract information pieces from different sites with very little knowledge and small amount of work. However, there are still additional work needed to be able to handle more advanced extraction tasks.

1. Introduction

The information extraction (IE) concept has been given a number of definitions such as the task of semantic matching between user-defined templates and documents written in natural language text, a process that takes unseen text as input and produces fixed-format, unambiguous data as output, and extract relevant text fragments and piece them together into a coherent framework [1, 2, 3]. The preferred definition for this paper is that information extraction is the process to find relevant subsets of textual information for a given task or question and organize them into a clearly defined data structure. This is different from the area of text understanding that attempts to capture the semantics of whole documents.

Examples of applications of IE are shopping agents that locate information about products or services at different retailers and compares them to find the best retailer, event agents that collect information about events that occurs at different locations and times, and news agents that collect news articles from different sources and presents articles relevant for a specific user. Information stored in free natural text or with semi-structured format would be too difficult to handle directly without IE for these applications.

The area of information retrieval (IR) has attracted a lot of attention due to the increased popularity of the World Wide Web. Services such as Google are known by most Internet users and are an essential part of the Web today. The main difference between IR and IE is that IR returns a set of documents rather than a set of answers or phrases related to the query. Thus, the information is not translated to a defined data structure in IR. The advantage of IR is that it is possible to cover a large number of domains, whereas IE typically requires domain-dependent knowledge and is therefore limited in the number of covered domains. These two areas can be combined and complement each other to provide useful services.

The concept structured text as used in this paper refers to textual information stored in a clearly defined data model, for example in a relational database. The advantage of clearly defined structured information is that the information can be automatically analyzed and processed more effectively. Semi-structured text may not have the clear data model representation as structured text, but have more structured information than natural language text, e.g. HTML documents with presentational information combined with the content. These semi-structured documents are often less grammatically correct than natural language texts with choppy sentence fragments [4]. The natural language processing (NLP) methods designed for unstructured natural language text does usually not work as well for semi-structured information. It has also been shown that the extraction task can be performed with very high accuracy using only the semi-structured information, without use of any NLP technique [4].

The term wrapper has been given different definitions depending on the context. In the database community, it represents a software component that converts data from one data model to another. In the Web context, it represents a software component that convert information in a Web page to a structured format, e.g. into a database. The latter corresponds to the preferred definition in this paper. The term wrapper represents an IE software component that takes semi-structured textual input and generates structured text as output. Automatic wrapper generation and wrapper induction are terms that refer to the automatic construction of wrapper, for example using machine-learning techniques.

The performance of semi-structured IE system (wrappers) is often measured differently than with information retrieval systems. The precision and recall measure is typically very high and therefore not a useful measure of the system. Only systems with 100% precision and recall are of interest for sources with significant amount of semi-structured information [5]. These systems are evaluated by their expressiveness and efficiency that measures the coverage of the wrapper (percentage of sources that have 100% precision and recall) and how easily the wrapper can be adapted to new domains.

The type of information extraction system that is described in this paper works in a different way and has different types of applications. Instead of trying to identify pieces in the natural language text using linguistic techniques, the focus is to keep track of pieces in the semi-structured information. A typical application is to keep track of items in some kind of list in documents, i.e. list extraction. For example, a shopping agent needs to keep track of products available from different retailers. Information about these products can be available in some kind of list on the retailer's homepage, e.g. a table with a row for each product. Other examples are calendar of events, and ads in action sites. It is this type of semi-structured extraction tasks that is in focus for this paper.

A problem with information extraction system, as opposed to information retrieval systems, is that it is difficult to handle a large amount of domains. Search engines based on information retrieval techniques can create word-index and rank document for a huge number of domains, but it is very difficult to create a general information extraction system. Therefore, instead of creating a general information extraction system, a system is created that can easily be adapted to new domains. This is called a user-driven information extraction system [6], where non-experts shall be able to train the system to handle new domains. The user shall not need programming skills nor be required to spend a lot of time to train the system. It must be easy to adapt to new domains if a large amount of domains shall be available.

2. The ISSIE System

The ISSIE (Intelligent Semi-Structured Information Extraction) system is a user-driven information extraction system that use semi-structured information to extract content from various types of lists.

There are several possible ways to design a system for the semi-structured information extraction task described above. Since it should be user-driven, it should not require the user to have expert skills in either the knowledge domain or to have expert programming skills. It is therefore not appropriate to require the user to design an ontology for the domain or to ask for complicated regular expression rules. The basic approach taken by the ISSIE system is to monitor details of the surfing behavior of the user and try to repeat the extraction process. The user is asked to start from a given page and then to navigate to the pages that contains the extraction pieces of interest, using a traditional web client. The user shall also give a few examples of which pieces that he/she is interested in. When this "training phase" is complete, the system shall go into the "examination phase" where it tries to repeat the navigation and locate the given pieces by itself. If the examination is successful, the system can go into the third phase, the "extraction phase". In this final phase, the system can automatically extract data from the information sources.

The examples that are provided by the user can currently be of three different types: singleton, list, or complex list. The singleton examples means that a single node in the parse tree shall be extracted (see section 2.1.1 for more information about the parse tree transformation). It could for example be a title in a page, or some similar piece of information that do not have any siblings to extract. The list type shall be used when a simple list in a page shall be extracted. For example, the list could be all cells in a column of a table of products at a retailer's homepage. The user only needs to provide two examples of cells in the list, and the system will find the remaining siblings by itself. See section 2.1.2 for additional information about the sibling algorithm.

Some lists have more complicated structure, where the siblings do not share the same path in the parse tree. It is this parse tree path that is used to find the siblings, so if this path differs between the siblings, the simple list will not work. An example of such a complex list is the list of news in Google (see experiment in section 4.2). To handle such lists, the user has to provide more than two examples.

At least one example for each sibling that have different parse tree path.

The navigation performed by the user is currently monitored by the system using a proxy server. All requests and responses sent between the web client and web server are seen by the proxy and information about them is stored in a database. If the system is able to repeat the extraction process and identify the wanted extraction pieces, then the training is complete and the system is ready to extract the pieces by itself. Remember that the type of extraction tasks that is in focus here is that pieces, e.g. information about products in a retailer's homepage, shall be identified and extracted. As that information changes over time, the extraction system shall be able to extract the new or changed pieces.

The advantage of using a proxy server is that technical details of the retrieval of web pages are captured by the system. The user can use web clients as usual and the system can still retrieve important technical information. In this way, it is possible for the system to handle web sites that depend on features such as cookies, form posts, and browser dependencies. The goal is to add as much intelligence as possible to the system, to handle technical details automatically, and require as little work and expertise as possible from the user. An alternative approach is to let the system try to navigate by itself without the information received from the proxy server. This has been attempted using reinforcement learning techniques, but it is difficult to be able to extract from advanced sites using this technique [7].

The architecture of the system is shown in figure 1. The two main parts of the system are the agent-system that handles the analysis and automated extraction tasks, and the user interface that allows the user to manage and train the system to extract information. The rest of this section will give a brief overview of how these parts work and how the extraction process works.

2.1. The Agent Sub-System

The part of the system that is responsible for handling the automated extraction process is developed using the JADE platform [8]. The motivation for using an agent-oriented approach for the design and execution of the system, is mainly for software engineering reasons. The agent-oriented way to decompose, abstract, and organize relationships can be more intuitive and efficient [9]. The system consists of Surfer agents that are able to download and handle web pages on the Internet, Analyzer agents that analyze documents to find the relevant pieces of information, and a Butler agent that communicates with the user and other systems.

The communication between the agents is made using

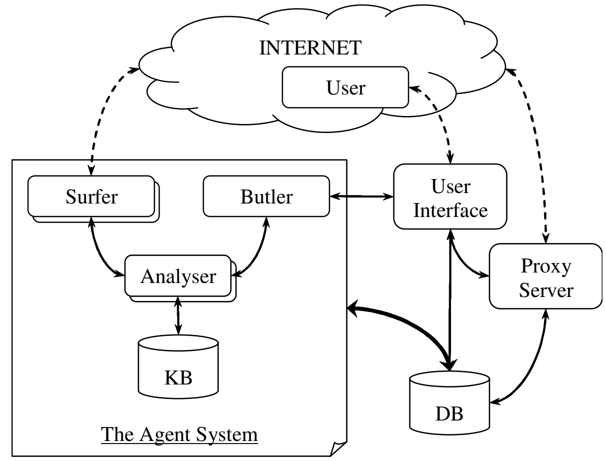

Figure 1. ISSIE Architecture

an ontology developed with Protégé-2000 [10]. The ontology designed with Protégé can be automatically used in JADE agent communication by using the Bean-generator plug-in for Protégé [11]. Also, the same ontology can be used for reasoning with the Jess logical engine in the agents. The ontology can be imported into Jess using the JessTab plug-in [12]. The integration of Protégé, JADE, and Jess provides an efficient way to communicate and reason with a high level of abstraction.

When a user wants to train the system to handle a new domain, it starts by adding a new task to the system using the user interface. When the training phase is complete and information about the navigation performed by the user and about the wanted extraction pieces is stored in the database, the agent system starts to work. The Butler agent is informed that the training phase is complete and it will send an examination request to an Analyzer agent for that training session.

The Analyzer agent will then start to examine the data from the training session. It will ask the Surfer agent to parse the requests and responses sent during the training phase. A semi-structured model of the pages are created by the Surfer, and they are they further analyzed by the Analyzer. The Analyzer works at a higher abstraction level than the Surfer. It never works with HTML or HTTP techniques, it only works with the the abstract model created by the Surfer. The advantage of this approach is that documents of other types than HTML can be analyzed by the Analyzer agent, as long as there are semi-structural information in the document.

The Analyzer agent uses the Jess logical engine and a knowledge base to handle the examination of the extraction process. The knowledge base consists of a set of rules and facts, which is used to decide which actions that the Analyzer shall take. See section 2.1.3 for more information about how this process works.

During the examination phase, when the agents shall repeat the extraction process, a set of web pages will need to be downloaded. When the Analyzer decides that a web page needs to be downloaded, it sends a download request to a Surfer agent. The Surfer agents have the abilities to communicate with web servers on the Internet and the necessary technical knowledge to create HTTP request and parse HTTP responses. It can also transform the HTML documents into a parse tree representation, i.e. the model that is later used by the analyzer.

2.1.1. The Semi-Structured Document Model

As stated earlier, the main type of information that is used by the ISSIE system in the extraction process is the semi-structured information. This is different from other types of information such as linguistic information, semantic information, and basic pattern matching. These other types of information are commonly used in other information extraction systems, e.g. named entity recognition, part of speech tagging, co-reference resolution, and use of semantical resources such as WordNet [13]. Linguistic and semantic information are currently not used by the system, but the addition of those types of techniques would improve the capabilities of the system. However, the use of linguistic information is not as appropriate for semi-structured text as for unstructured text. The text is often less grammatically correct and contains mostly choppy sentence fragments [4]. If semi-structured information exists in a document, that information can sometimes be sufficient by itself to complete an extraction task [4].

To be able to take advantage of the semi-structured information, the documents need to be transformed from the string representation to a tree representation. The system constructs a parse tree for each document where each node in the tree represents a block element[1] in the document. Also, links from one page to another page are considered to be nodes in this model. The edges between the nodes in the tree represent the parent-child relationships between the elements in the document.

The motivation for using only block-level elements is that the may represent an actual separation of text pieces, whereas in-line elements typically only represent changes in presentation. Of course, this may not always be the case and it may sometimes be preferable to also use in-line element to separated text pieces. Information about in-line element and linguistic information could be useful, but are currently not used by the system.

[1] A block element is different from an in-line element in that that typically begin on a new line and represent some block of text, e.g. DIV and P elements.

2.1.2. The Sibling Algorithm

The semi-structured information is used both for navigating through the document, locating the relevant, and to find all the siblings in a list. When a user adds a (simple) list during the training phase, he/she is supposed to provide the content for two pieces in the list. Using only the information about the content and the semi-structured information in the document, the system shall be able to locate all remaining siblings in the list.

The algorithm basically searches for the smallest possible paths to the siblings and store that path for each given sibling. These paths are then used to find all remaining siblings in the lists. The basic outline of the algorithm looks as follows for each given example:

1. Search the parse tree systematically for the first piece in the example and store that node in e.
2. Start from e, store parent node in p, and initialize up path $path_{up} \leftarrow \{p\}$.
3. Initialize list of siblings $siblings$ for all siblings (if any) for current example.
4. For each child c of p:
   (a) Add c to down path: $path_{down} \leftarrow path_{down} \cup \{c\}$.
   (b) Check content in c for sibling example match from $siblings$.
   (c) If match, store $path_{up}$ and $path_{down}$ as sibling paths for matching sibling and remove that sibling from $siblings$.
   (d) If $siblings$ is empty, terminate sibling search.
   (e) If no match, continue recursively to get children of c and go to step 4.a.
   (f) If $siblings$ is empty, terminate sibling search.
   (g) Remove c from down path: $path_{down} \leftarrow path_{down} \cap \neg\{c\}$.
   (h) Continue with next c.

2.1.3. The Examination Process

When the user has completed the training phase and thereby demonstrated to the system how to navigate and what to extract, the system shall examine the training data and try to repeat the extraction. The examination starts when the Butler agent sends an examination request to the Analyzer agent for the training session that was just completed by the user. The Analyzer now starts the examination. Here are the basic steps of the examination process:

1. Make sure the requests and responses have been parsed and that a parse tree has been built for each document. This is performed by the Surfer agent.

2. When the parse tree model has been built, the Surfer agent tries to locate the siblings for all lists that were given by the user. Each node that contains a wanted extraction piece is marked as an extraction node, including all siblings in a list, according to the sibling algorithm given above.

3. The Analyzer tries to repeat the extraction process by itself, using heuristic rules. It has three different plans to succeed with the extraction, and it starts with the simplest plan.

4. The first plan consists simply of requesting only the pages containing the extraction points. In some cases, this will work and it is therefore not necessary to request any other pages that exist in the training session. The Analyzer agent sends download requests to the Surfer agent.

5. If the first plan fails, it continues with the second plan that consists of requesting all pages containing form submittals. The motivation for such a plan is that it is common to need to authenticate in a web site before being able to obtain the wanted pieces. Also, if a search or similar type of filtering has been performed, it usually involves a form submittal.

6. If the second plan fails too, the Analyzer continues with the third plan. The third plan involves requesting all pages in chronological order from the training session, excluding duplicates.

7. If a plan succeeds and the wanted pieces are located, the Analyzer stores that plan in the database and the examination is completed. If no plan succeeds, the system responds to the user that it was unable to repeat the extraction.

2.2. The User Interface

The user interface to the ISSIE system allows the user to manage the extraction tasks. The user can train the system to handle new tasks and modify existing tasks. The interface is a traditional web interface built using .Net. It is necessary for the interface to be simple enough so that users are not required to be computer experts.

Due to the space limitations of the paper, it is not possible to include any detailed information about the interface. However, since it is a traditional web interface, there is not much relevance to give any detailed information. A small screen shot is given for the page that manages a specific

Table 1. List of experiments

Top news stories	Extract headlines of the current top news stories and the current top story from http://www.cnn.com/
Current scientific	Extract current scientific news stories headlines from http://news.google.com/
Video drivers	Extract video driver updates for a specific computer model from http://www.dell.com/

task in figure 2. From that page, it is possible for the user to give basic information about the task, to train the system, and modify other information for the task.

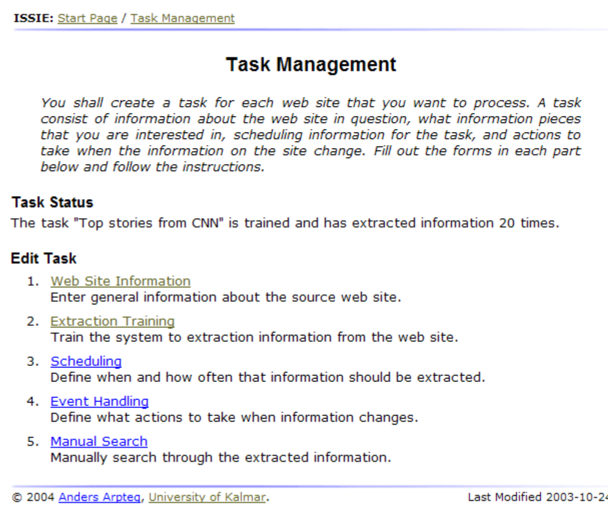

Figure 2. User Interface: Task Management

3. Experiments

To evaluate the semi-structured list extraction hypothesis, a set of experiments was conducted using the ISSIE system. These experiments consist of extraction tasks such as extract the available driver updates for a particular computer model from a manufacturer's homepage. The list of experiments conducted is shown in table 1. The motivation for choosing these extraction tasks are that they are that the tasks represent interesting and relevant tasks, regardless of how advanced the web site and the extraction is.

The basic steps in each experiment are as follows:

1. The user creates a new task in the ISSIE user interface.
2. Basic information such as name of task and start URL are given by the user.
3. The user starts the training phase by making sure that the correct proxy settings are configured in the browser and then going to the start URL.
4. The user shall now navigate from the start URL to the pages containing wanted information pieces. It is possible for the user to for example provide username and password and fill out forms to obtain the information pieces.
5. When the user arrives at a page that contains wanted information pieces, the user shall copy text from those pieces and paste them into the "training" page in the ISSIE user interface. If a list of pieces shall be extracted, only two random items in the list needs to be copied. It is possible to specify if the pieces is part of a list or if it is a singleton in the ISSIE user interface.
6. When all information pieces have been found and examples copied to the user interface, the user stops the training by clicking a "training complete" button in the ISSIE interface.
7. The agents in the ISSIE system will now analyze the training session provided by the user and try to repeat the process. Information about the status is shown to the user.
8. If the agents were able to repeat the process and find the pieces provided by the user by themselves, the training is successful and the automated extraction can start. Otherwise, the user may need to provide additional information and re-train the system.

4. Results

The system was able to find the given extraction pieces and most of the times find the additional items in lists. There were some problems to correctly find all items in a list for some tasks, since the structure for list item was not always identical.

Here is some more detailed information about each experiment:

4.1. Top News Stories

The task to extract news stories from the CNN site is a common test for information extract systems. The task was very simple, take the top stories directly from the start page, which are located in a small box in the top right part of the start page. The system should not only extract the top stories headline, but also extract the current main top story that is located in a different place in the start page.

The two pieces "CIA: Tape likely is bin Laden" and "Blair sees wider role for U.N. in Iraq" where given for the top stories list and the piece "U.S. delays troop return from Iraq" was given for the main top story singleton.

There was no problem for the system to repeat the extraction and locate the additional items in the top stories list.

4.2. Current Scientific News Stories

The task consists of finding and locating the current scientific news from the Google news site. The start page was http://news.google.com/ and the scientific news can be reached with one click from the start page. This list is an example of a complex list. Some of the articles in the list of scientific news has a picture before the headline in a separate column of and other articles have no picture and therefore are the headline placed in a different column.

The pieces "Cassini quietly awaits ride in Saturn's orbit", "New texting speed record set", and "Juniper Serious About SAML" were given as examples of headlines. The two last pieces are siblings to the first piece and covers the two different types of articles in the list, i.e. with and without picture.

The system was able to determine the paths to both siblings and able to extract all siblings in the list.

4.3. Video Drivers

The purpose for this task is to be able to extract the list of video driver updates for a specific computer model from http://www.dell.com/. This is a rather advanced task since it involves a large amount of pages to navigate through, and it also requires form posts and cookie management to work properly. It is also spread across several web servers.

The task consist of starting at the start page, navigating to the support pages, submitting the service tag number in a form to retrieve updates relevant for a specific computer, and navigating to the video drivers page. There are in total nine clicks, 34 pages[2], and one form post to reach the video drivers page.

The two pieces "Video: ATI Mobility Radeon 9000, Driver, Windows XP, Multi Language, Inspiron 8500, v.7.80.4-021206a-6945c, A00" and "Video: nVidia GeForce4 4200 Go, Driver, Windows 2000, Windows XP, Multi Language, Inspiron 8500, Latitude D800, v.6.13.10.4258, A03" were given as examples of the video

[2] The number of pages are larger than the number of clicks since there are frames, redirects, and similar requests

drivers list. The system was able to navigate and locate the given pieces and locate the additional 11 drivers in the list.

5. Related Work

There exist other systems where the semi-structured information is used, for example [14, 15, 4, 16, 5]. The main idea in the wrapper toolkit by Ashish and Knoblock is to exploit the semi-structured information to facilitate the extraction task. The construction of a wrapper starts with identifying the relevant structure of a page, building a parser based on given structure, and finally adding communication capabilities to the wrapper to be able to find different sources of information and give the result to a mediator. A set of heuristic rules are used to identify sections and subsections in the web pages. These rules are basically regular expressions that exploit HTML knowledge to find the structure. In addition, heuristics such as font size are used to determine the hierarchical level of the structure. There is no training in the system, although the user is able to correct erroneous guesses through a graphical user interface. The heuristics basically employs pattern matching rules to identify sections and subsections, with assistance of HTML knowledge. The actual structural relationships present in the source pages are not used for the identified output structure.

The Rapper system uses the same techniques as in the Ashish system [14] and adds algorithms that employ linguistic knowledge. These extensions increase the cost of adapting the system to new domains, although they increase the accuracy for implemented domains. As stated in the paper, the construction of wrappers is a non-trivial task even with these tools. A significant amount of knowledge is still required to construct a wrapper.

The WYSIWYG Wrapper Factory [15] provides a very nice graphical user interface that allows the user to add extraction rules that takes advantage of the semi-structured information. However, there is no training in the system and the user still needs to be familiar to the advanced rule language used in the system.

6. Conclusion

As previously stated, the hypothesis in the paper is that a user-driven approach to semi-structured complex list extraction is possible and that it can in the long run lead to a large amount of extractable domains. There are several possible applications for this type of information extraction. For example, the user may simply want to receive notifications when some information pieces are changed, added, or removed from a site. A more advanced long term application would be to facilitate the goal of the Semantic Web. If the information on the Web should be machine understandable and not only machine readable, then the documents need to be transformed from the presentation format of HTML to more semantically encoded formats such as RDF and OWL [17, 18]. This type of information extraction services could assist in this transformation and make the information automatically available for machines as well as for humans.

A problem with previous implementations of the ISSIE system was that it was not able to handle complex lists that had different structures. By allowing the user to provide additional examples in the lists, and adding support in the system to handle multiple paths to the siblings, these more complicated lists can also be extracted.

Another problem that is more difficult to solve is how to manage multi-page tables. It is common to split a list into several pages, to make the list more manageable. The user of the ISSIE system still expects to extract all items in the list, not only the items in the first page. A future implementation of the system could possibly be improved to have multi-page lists, in addition to complex lists, normal lists, and singletons. The user would then need to give information by example of how to navigate to other pages in the list, in addition to how to extract the items in the list.

In general, the experiments were promising and the approach of using a proxy to monitor technical details that the user is unaware of is of great help. If compared to the reinforcement learning approach that should automatically navigate given some wanted text pieces, this approach is able to handle very advanced sites easily.

The current system allows users without domain expertise knowledge and without programming knowledge to quickly create an extraction task. The output from the system provides an XML document that can be managed by other computer systems. A possible application can be to encode additional semantic knowledge into the XML document and make it public. In that way, a large amount of domain can be made machine understandable and make the information available on the Internet more valuable.

Acknowledgment

This research was funded by the Swedish Knowledge Foundation and University of Kalmar. I also wish to thank Erik Sandewall for his encouragement and guidance throughout this project.

References

[1] N. Guarino, *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, ser. Lecture Notes in Artificial Intelligence. Frascati: Springer, July 1997, vol. 1299, ch. 8. Semantic Matching: Formal On-

tological Distinctions for Information Organization, Extraction, and Integration, pp. 139–170.
[2] H. Cunningham, "Information extraction: A user guide (revised version)," Department of Computer Science, University of Sheffield, Tech. Rep. CS-99-07, May 1999.
[3] J. Cowie and W. Lehnert, "Information Extraction," *Communications of the ACM*, vol. 39, no. 1, pp. 80–91, 1996.
[4] S. Soderland, "Learning to extract text-based information from the world wide web," in *Proceedings of the 3rd International Conference on Knwoledge Discovery and Data Mining (KDD-97)*, 1997.
[5] N. Kushmerick, "Wrapper induction: Efficiency and expressiveness," *Artificial Intelligence*, vol. 118, pp. 15–68, 2000.
[6] A. Arpteg, "User-driven semi-structured information extraction," in *4th International Conference on Intelligent Systems Design and Applications*, August 2004.
[7] ——, "Adaptive semi-structured information extraction," Licentiate dissertation, Linköping university, Linköping, 2003.
[8] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE — A FIPA-compliant agent framework," in *Proceedings of the 4th International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-99)*, 1999, pp. 97–108.
[9] N. R. Jennings and M. Wooldridge, *Handbook of Agent Technology*. AAAI/MIT Press, 2000, ch. Agent-oriented Software Engineering.
[10] W. Grosso, H. Eriksson, R. Fergerson, J. Gennari, S. Tu, and M. Musen, "Knowledge modeling at the millennium – the design and evolution of protege," in *Proceedings of the 12 th International Workshop on Knowledge Acquisition, Modeling and Mangement (KAW'99)*, Banff, Canada, October 2000.
[11] C. van Aart, "Java ontology bean generator for jade 3.0," 2003, http://www.swi.psy.uva.nl/usr/aart/beangenerator/ (2004-04-19).
[12] H. Eriksson, "Using jesstab to integrate protégé and jess," in *IEEE Intelligent Systems*, vol. 18, no. 2, 2003, pp. 43–50.
[13] C. Fellbaum, Ed., *WordNet: An Electronic Lexical Database*. Cambridge: MIT Press, 1998.
[14] N. Ashish and C. Knoblock, "Wrapper generation for semi-structured internet sources," in *Proc. Workshop on Management of Semistructured Data*, Tucson, 1997.
[15] A. Sahuguet and F. Azavant, "Wysiwyg web wrapper factory," 1999.
[16] D. Mattrox, L. J. Sligman, and K. Smith, "Rapper: A wrapper generator with linguistic knowledge," in *Workshop on Web Information and Data Management*, 1999, pp. 6–11.
[17] World Wide Web Consortium, "RDF primer," 1999, http://www.w3.org/TR/REC-rdf-syntax (2004-04-19).
[18] ——, "OWL Web Ontology Language Guide," 2004, http://www.w3.org/TR/owl-guide/ (2004-04-19).