Web: www.witpress.com Email witpress@witpress.com
Paper from: *Management Information Systems 2002*, CA Brebbia and P Pascolo (Editors).
ISBN 1-85312-907-0

ALII: An information integration environment based on the active logic framework

A. A. Barfourosh[1], H.R. Mothary Nezhad[1], M. O'Donovan-Anderson[2] & D. Perlis[2,3]
[1] *Intelligent Systems Lab., Computer Engineering Dept., Amir Kabir University of Technology, Iran.*
[2] *Institute for Advanced Computer Studies, University of Maryland, College Park, MD USA*
[3] *Department of Computer Science, University of Maryland, College Park, MD USA*

Abstract

There is growing interest in accessing, relating, and combining data from multiple sources on the Web. Today, web users access the web through two dominant interfaces: clicking on hyperlinks (browsing) and searching via keyword queries (crawling). This process is often time-consuming and unsatisfactory because of the many inaccurate and irrelevant results that are returned. Better support is needed for expressing one's information requirements and dealing in more structured and systematic ways with one's search results. This paper presents ALII, our proposal for providing this support.

ALII is an Active Logic based framework for Information Integration, which can utilize a compact context representation and constructs a hierarchy model of query and web pages. The ALII project aims developing methods and tools for a model based, semantic integration of information sources on web pages.

1 Introduction

Today there is growing interest in accessing, relating, and combining data from multiple sources. Enormous amounts of heterogeneous information have been accumulated within corporations, government organization and universities. Such information continues to

Paper from: *Management Information Systems 2002*, CA Brebbia and P Pascolo (Editors).
ISBN 1-85312-907-0

grow at an ever-increasing rate. This information comes from different subject areas, and comes in different formats: bitmap, plain text, binary, etc.

The World Wide Web is one such information resource, consisting of 1.5 billion pages covering different subjects and presented in different formats. As more information becomes available on the World Wide Web, it becomes more difficult to provide effective tools for accessing this information. Today, web users access the web through two dominant interfaces: clicking on hyperlinks (browsing) and searching via keyword queries (crawling). This process is often time-consuming and unsatisfactory because of the many inaccurate and irrelevant results that are returned. Better support is needed for expressing one's information requirements and dealing in more structured and systematic ways with one's search results.

Users have two main tools to help them locate relevant resources on the Web: Catalogs and Search Engines. Human experts construct catalogs. They tend to be highly accurate but can be difficult to maintain as the web grows. To keep up with this growth search engines were designed to eliminate human effort in cataloging web sites. A search engine consists of a mechanism that "crawls" the web looking for new or changed pages, an indexing mechanism and a query interface. Users generally query against the system index, although many contemporary search engines now also use link analysis to some degree. However, link analysis only helps to identify the most popular pages, and popularity may or may not correlate to relevance for a particular query.

Let us use an example to illustrate some of the problems with web search. A reasonable search term might be "Artificial Intelligence Courses in Computer Science Departments in the USA". As we can see, the result from the search engine is huge. Finding the proper set of results from this query from within the returned set is very time consuming. Some of the pages are not relevant and some others, which may be relevant, can only be found by following links several levels deep.

It seems clear, then, that an automated system that could reason about the relevance of pages on the web to a given query would represent a significant advance in the accessibility and usability of the web. We propose Active Logic as a candidate for the reasoning engine of such a system. Active Logic is a formal architecture that is more closely tied to implementational constraints than is usual for formalisms, and which has been used to solve a number of commonsense problems in a unified manner. In particular, Active Logic seeks to apply theoretically justifiable, principled (logic-based) methods of reasoning to dynamic, uncertain—and to this extent real world—contexts. [2,3] Active Logic works by combining inference rules with a constantly evolving measure of time (a "Now") that can itself be referenced in those rules. As an example, from Now (t)—the time is now "t"—one infers Now (t+1), for the fact of an inference implies that time (at least one 'time-step') has passed. All the inference rules in Active Logic work temporally in this way: at each time-step all possible one step inferences are made, and only propositions derived at time t are available for inferences at time t+1.

In following sections, we study the existing problems in information integration and will propose an Active Logic based environment for Information Integration (ALII). In section 2 we describe the current research projects on information integration. In section 3 we review the use of focused crawling to improve the efficiency of information integration systems. Section 4 consists of a brief introduction to active logic and introduces an Information Integration Environment based on Active Logic. Our conclusions are presented in section 5.

Paper from: *Management Information Systems 2002*, CA Brebbia and P Pascolo (Editors).
ISBN 1-85312-907-0

2 Current Projects on Information Integration

There have been numerous attempts to improve the search engine. One of the major efforts in this regard is in improving Web Crawlers, also known as robots, spiders, worms, walkers, and wanderers. A crawler generally starts with some default web addresses, and downloads the specified document. For each page, it pulls out all of the addresses (links) contained within the page–it will search in breath-first-search manner later. It then indexes all of the words in that page, storing every word and phrase in that page in a database so the system can later search that database for a phrase that might exist in this page. Some other information is also stored about the page, e.g. the time when it was last downloaded, time when it was last updated, summary words, the title of the page, etc. Every word in that page is searchable by a user once it is saved to the database; this enables one to search for phrases or keywords in any document on the Internet.

Note that the crawler pulls out all of the links within that page for future reference; this is where the "crawler" concept comes in. Theoretically, a web crawler could start with one page, grab all of the links from that page, search those pages in turn (thereby grabbing more links) and continue until it has searched all pages on the Internet. The problem with this strategy is clear: you can't get to all parts of the Internet from a single point. Some pages are entirely unreferenced, and there are pools of pages, which, although internally linked, are as a group not referenced elsewhere in the net. Further, the crawler sometimes isn't terribly smart about which "path" it takes in crawling the Internet. Web crawlers are typically automatic, with only a bit of human maintenance. As a result, the information is stored away as a bunch of keywords associated with the document, but no human summary or classification is available. This makes these types of search engines excellent for finding easy to specify or unusual information, but not nearly as efficient for common information; if you type "AI" into a crawler-based search engine, it will return hundreds of thousands of results.

There are several hundred commercial web crawlers; some of popular web crawlers and their abstract descriptions have been introduced in [4] Two of the important issues in web crawler design are Scalability and Extensibility. By *scalable*, we mean that a system must scale up to the entire web, considering its growth. By *extensible*, we mean that it must be designed in a modular way, with the expectation that third parties can add new functionality. Building a scalable crawler is a non-trivial endeavor because the data manipulated by the crawler is too big to fit entirely in memory, so there are performance issues relating to how to balance the use of disk and memory. Several techniques have been used to improve the functionality of web crawlers such as: *Mining the link structure of web* and using it to bring order in URL crawling, *automatic classification of web pages* and *using machine learning methods* and *domain-specific crawling*. [4]

Crawling the web in general involves gathering up the links on each visited page and putting them into a queue of links to be followed. Focused crawling (e.g., [5,6,7,8]) reorders the links in the queue as to their predicted likelihood to lead to pages that are relevant to a particular topic. By following high likelihood links first, pages that are relevant to a particular topic are found more quickly. Furthermore, links leading to irrelevant pages tend to fall to the bottom of the queue, and can be ignored once the crawler has determined that the rate of finding new relevant pages from following links has fallen below some given threshold.

Paper from: *Management Information Systems 2002*, CA Brebbia and P Pascolo (Editors).
ISBN 1-85312-907-0

The key to creating a topic specific search engine is to extract values for specific fields from the web pages, storing the values in a database so that structured queries can be performed over the extracted information. To aid in this endeavor, much research has been done in the area of text classification (e.g., [9,10,11]). Text classification is used in topic specific search engines in at least two areas. First, it is used during crawling to classify web pages as to whether they are relevant to the given topic. Second, it can be used to classify relevant web pages or content extracted from web pages into a hierarchy. The general idea is to first convert the text to a multidimensional vector representation, where dimensions correspond to words in the text, then to use machine learning or statistical techniques (e.g., decision trees, naive Bayes) to classify the vectors in the multidimensional space.

The hypertext structure of the Web provides an additional advantage for web page classification that is not available for classifying text in general: One can use the text in links leading to the page and also the classification of nearby pages to make classification of a particular web page more accurate than if classification were performed on the text of the page alone. In comparison to extracting information from flat text files, it is possible to get increased accuracy when extracting information from web page by taking advantage of the HTML tag structure. One common approach to extracting information from web pages is to write site specific "wrappers" that extract information based upon regularities of the HTML tag structure that is typically present in a single web site. Another method for extracting information from the Web is to use a bootstrapping approach. In this approach one begins with a small relation and searches the Web for additional tuples to add to the relation. New tuples are added to the relation using a bootstrapping technique, where new tuples are added if they appear on web pages in the same contexts as existing tuples in the relation [12].

3 What can focused crawlers do?

Focused crawlers can be divided into two categories: those focused on specific *topic* and those focused on specific *document type*. If you want to crawl all pages related to "object-oriented frameworks", that is the task of topic crawlers [4]. But if you want to fetch all course pages in a university or over the world, then this is the task of focused crawling on document type. This kind of crawler would also crawl on other specific types, such as resumes, homepages, calls-for-papers, FAQs, movie listings, technical papers, etc. Focused crawling concentrates on the quality of information and the ease of navigation as against the sheer quantity of the content on the Web. A focused crawler [5] seeks, acquires, indexes, and maintains pages on a specific set of topics or document types. Topics are specified to the focusing system using exemplary documents and pages, instead of keywords. Rather than collecting and indexing all accessible Web documents to be able to answer all possible ad-hoc queries, a focused crawler analyzes its crawl boundary to find the links that are likely to be most relevant for the crawl, and avoids irrelevant regions of the Web. In essence, this process will result in personalized sub-webs within the World Wide Web. This method results in significant savings in hardware and network resources, and yet achieves respectable coverage at a rapid rate, simply because there is relatively little to do. Thus, a distributed team of focused crawlers, each specializing in one or a few topics, can theoretically manage the entire content of the Web.

Paper from: *Management Information Systems 2002*, CA Brebbia and P Pascolo (Editors).
ISBN 1-85312-907-0

The ideal focused crawler retrieves the maximal set of relevant pages while simultaneously traversing the minimal number of irrelevant documents on the web. Each focused crawler is far more nimble in detecting changes to pages within its focus than a crawler that crawls the entire Web. Focused crawlers therefore offer a potential solution to the currency problem by allowing for standard exhaustive crawls to be supplemented by focused crawls for categories where content changes quickly.

Here is what we found when we used focused crawling for many varied topics at different levels of specificity.

- Focused crawling acquires relevant pages steadily while standard crawling (like the ones used in first-generation search engines) quickly loses its way, even though they start from the same root set.
- Focused crawling is robust against large perturbations in the starting set of URLs. It discovers largely overlapping sets of resources in spite of these perturbations.
- It can discover valuable resources that are dozens of links away from the start set, and at the same time carefully prune the millions of pages that may lie within this same radius. The result is a very effective solution for building high-quality collections of Web documents on specific topics, using modest desktop hardware.
- Focused crawlers impose sufficient topical structure on the Web. As a result, apart from the naïve topical search, powerful semi-structured query, analysis, and discovery are also enabled.
- Getting isolated pages, rather than comprehensive sites, is a common problem with Web search. With focused crawlers, you can order sites according to the density of relevant pages found there. For example, you can find the top five sites specializing in Computer science research activities.
- A focused crawler also detects cases of competition. For instance, it will take into account that the homepage of a particular auto-manufacturing company like Honda is unlikely to contain a link to the homepage of its competitor, say, Toyota.
- Focused crawlers also identify regions of the Web that grow or change dramatically as against those that are relatively stable.

The ability of focused crawlers to focus on a topical sub-graph of the Web and to browse communities within that sub-graph will lead to significantly improved Web resource discovery [5,13]

4 What is Active Logic?

Active Logic is a kind of "step logic," which was developed in [2] as formal mechanism for modeling the ongoing process of reasoning. Unlike traditional logical formalisms, a step-logic does not calculate a final set of conclusions which can be drawn from an initial set of facts, but rather monitors the ever-changing set of conclusions as time goes on. There are special persistence rules so that every theorem α present at time t implies itself at time t+1; likewise there are special rules so that if the knowledge base contains both a theorem α and its negation $-\alpha$, these theorems and their consequences are "distrusted" so they are neither carried forward themselves nor used in further inference. An active logic, then, consists of a formal language (typically first-order) and inference

Paper from: *Management Information Systems 2002*, CA Brebbia and P Pascolo (Editors).
ISBN 1-85312-907-0

rules, such that the application of a rule depends not only on what formulas have (or have not) been proven so far (this is also true of static logics) but also on what formulas are in the "current" belief set. In general the current beliefs are only a subset of all formulas proven so far: each is believed when first proven but some may subsequently have been rejected. Active Logics have the following characteristics: they are *situated in time, maintain a history, tolerate contradictions, and allow meta-reasoning to be done.*

Based on the above definition, active logics are a family of inference engines that incorporate a history of their own reasoning as they run. Thus at any time ***t***, an active logic has a record of its reasoning at all times prior to ***t***, and it also knows that the current time is ***t***. As it continues to reason from time ***t***, that reasoning is also recorded in the history, marked at time *t+1* as having occurred at time ***t***. Thus an active logic records the passage of time in discrete steps, and the "current" time slides forward as the system runs. It is convenient to regard its current inferences as occurring in a working memory, that is then transferred to the history (or long-term memory) in the next time-step. Thus, an active logic has time-sensitive inference rules and consequently time-sensitive inferences. In active logics the current time is itself noted in the working memory—Now (t)—and this changes to Now (t+1) one step later. (A time-step should be thought of as very fast, perhaps 0.1 sec in correspondence with performance of elementary cognitive tasks by humans). Thus active logics "ground" Now in terms of real time-passage during reasoning.

These characteristics make active logics suitable for use in various domains including time situated planning and execution [15]; reasoning about other agents [16]; reasoning about dialog [17,18], including updating and using discourse context [19]; and autonomous agency [20].

4.1 Rules of Active Logic

There are many examples of active logics in various papers. We present here a couple of simple rules.

1) Time step update rule: t: Now(t), then : t+1: Now(t+1)

is a rule that says: if at the current step, Now has the value t, then, at the next step, let Now have the value (t + 1). This enables the active logic to keep track of step numbers and therefore of time. This is a basic rule and is included in all active logics.

2) Another example is the contradiction rule:

t: P, not(P), then: t+1: contra(P, not(P))

If at a step, we have both P and not(P) present in the database, at the next step, we add contra(P, not (P)) to the database to indicate the contradiction. There will be other rules that will cause the consequences of P and not(P) not to be derived in later steps, and rules that will attempt to resolve the contradiction and reinstate either P or not(P) to the database at a later time.

3) We can also have modus ponens: t: P, $P \rightarrow Q$, then conclude: t+1: Q.

This says: if at time t, the database contains P and ($P \rightarrow Q$), then in the next time step, conclude Q. Note that if the database contains P, ($P \rightarrow Q$) and ($Q \rightarrow R$), we do not get R immediately, but only after 2 steps. First, we use P and ($P \rightarrow Q$) to obtain Q, then in the second step, we use this together with ($Q \rightarrow R$) to derive R.

4) The inheritance rule keeps formulas in the database unless there is a contradiction:

Paper from: *Management Information Systems 2002*, CA Brebbia and P Pascolo (Editors).
ISBN 1-85312-907-0

t: P , not_know(not(P)), \+ P = Now(t), then conclude: t + 1: P.
not_know(P) is true iff P is not in the current database. Since the database is finite, this poses no computational problems. "\+ P = Now(t)" verifies that P is not of the form Now(t) and prevents time from being inherited. This rule also prevents the lemmas of a contradiction from being inherited.
5) Let the sentences initially present in the database be: Now (0), Bird (tweety), Bird (x) & not_know (not (fly (x))) → fly (x). With the above rules of inference, this is what the database looks like at consecutive steps:
At step 0: Now(0), Bird(tweety), Bird(x) & not_know(not(fly(x))) → fly(x)
At step 1: Now(1), Bird(tweety), Bird(x) & not_know(not(fly(x))) → fly(x), fly(tweety)
since "not(fly(tweety))" is not present in the database at step 0.
The database will not change thereafter.

4.2 Why Active Logic for Focused Crawling?

The focused crawler [5] has three main components: a **classifier** which makes relevance judgments on pages crawled to decide on link expansion, a **distiller** which determines a measure of centrality of crawled pages to determine visit priorities, and a **crawler** with dynamically re-configurable priority controls which is governed by the classifier and distiller. Detailed description of these components can be found in [4].

Two critical factors for the design, implementation and maintenance of a Focused Crawler are: conceptual modeling of the domain, and reasoning support over the conceptual representation. Knowledge representation and reasoning techniques play an important role for both of these factors. By using an Active Logic based framework in Focused Crawler architecture we will be able to create an engine to implement these roles well.

A major problem faced by existing crawlers is that it is frequently difficult to learn that some sets of off-topic documents often lead reliably to highly relevant documents. This deficiency causes problems in traversing the hierarchical page layouts that commonly occur on the web. Consider for example a researcher looking for papers on Natural Language processing. A large number of these papers are found on the home pages of researchers at computer science departments at universities. When a crawler finds the home page of a university, a good strategy would be to follow the path to the computer science (CS) department, then to the researchers' pages, even though the university and CS department pages in general would have low relevancy scores. An adaptive focused crawler based on active logic could in principle learn this strategy by building a tree from query and expanding this tree while the system is in process. It is doubtful that the crawler would ever explore such a path in the first place, especially as the length of the path to be traversed increases.

To address this problem, Rennie and McCallum [6] used reinforcement learning to train a crawler on specified example web sites containing target documents. The web site or server on which the document appears is repeatedly crawled to learn how to construct optimized paths to the target documents. However, this approach places a burden on the user to specify representative web sites. Initialization can be slow since the search could result in the crawling of a substantial fraction of the host web site. Furthermore, this approach could face difficulty when a hierarchy is distributed across a number of sites.

Paper from: *Management Information Systems 2002*, CA Brebbia and P Pascolo (Editors).
ISBN 1-85312-907-0

An additional difficulty faced by existing crawlers is that links on the web are unidirectional, which effectively restricts searching to top-down traversal, a process that we call "forward crawling" (Obtaining pages that link to a particular document is referred to as "backward crawling"). Since web sites frequently have large components that are organized as trees, entering a web site at a leaf can result in a serious barrier to finding closely related pages. For example, when a researcher's home page is entered, say via a link from a list of papers at a conference site, a good strategy would be for the crawler to find the department member list, and then search the pages of other researchers in the department. However, unless an explicit link exists from the researcher's page to the CS department member list, existing focused crawlers cannot move up the hierarchy to the CS department home page.

ALII utilizes a compact context representation and construct a hierarchy model of query and web pages. The crawler based on active logic also utilizes the limited backward crawling possible using general search engine indices to efficiently focus crawl the web. Unlike Rennie and McCallum's approach [6], this approach does not learn the context within which target documents are located from a small set of web sites, but in principle can back crawl a significant fraction of the whole web starting at each seed or on-topic document. Furthermore, the approach is more efficient in initialization, since the tree is constructed by directly branching out from the good set of documents to model the parents, siblings and children of the seed set.

This focused crawler uses the limited capability of search engines like AltaVista or Google to allow users to query for pages on HTML format linking to a specified document. This data can be used to construct a representation of pages as a tree that occur within a certain link distance (defined as the minimum number of link traversals necessary to move from one page to another) of the target documents regarding to query tree, which improved during the process. This representation is used to train a set of classifiers, which are optimized to detect and assign documents to different categories based on the expected link distance from the document to the target document. During the crawling stage the classifiers are used to predict how many steps away from a target document the current retrieved document is likely to be. This information is then used to optimize the query and search.

There are three distinct stages to using the active logic when performing a focused Crawl session:

1. An initialization phase when a query present to a search engine. In this phase the initial tree will be constructed.
2. A crawling phase that extract the pages from web sites. In this stage the associated tree are constructed for each of the seed pages.
3. A process phase that evaluate each page tree with query tree. In this phase query tree will improved from time (t) to time (t+1).

ALII outputs from the process phase can be used as knowledge input to a Classifier component. ALII, as presented, is a representation environment and logical reasoning tool with a formal foundation in Active Logic. The ALII Project aims at developing methods and tools for a Model-based, Semantic Integration of information sources on Web pages. Current work on the project is centered on a demonstrator application for information services. ALII can search (through the links), acquire, index and maintain pages that are relevant to a predefined set of topics and effectively build high quality collections of Web documents.

Paper from: *Management Information Systems 2002*, CA Brebbia and P Pascolo (Editors).
ISBN 1-85312-907-0

5 Conclusions

In this paper we introduced a brief picture of different issues and problems in information integration that can be tackled by using active logic. We have made some detailed suggestions regarding an important application area for active logic: focused web crawling. We believe that such a reasoning component is required for information integration to achieve an intelligent mechanism with the ability to behave effectively over considerably longer time periods and range of circumstances.

ALII, Active Logic based framework for Information Integration, will utilize a compact context representation and construct a hierarchy model of query and web pages. The crawler based on ALII will also utilize the limited backward crawling possible using general search engine indices to efficiently focus-crawl the web. ALII consists of an inference engine with inference rules which: are situated in time, can maintain a history, tolerate contradictions and do meta reasoning. Information Integration systems based on ALII can search (through the links), acquire, index, and maintain pages that are relevant to a user query.

References

[1] Soumen Chakrabarti, Martin H. van den Berg, Byron E. Dom, Distributed Hypertext Resource Discovery Through Examples, Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[2] J. Elgot-Drapkin. Step-logic: Reasoning Situated in Time. *Ph.D. Thesis, Department of Computer Science, University of Maryland, College Park*, 1988.

[3] J. Elgot-Drapkin, S. Kraus, M. Miller, M. Nirkhe and D. Perlis; A Unified Formal Approach to Episodic Reasoning. *Technical report*, University of Maryland, 1988.

[4] A. A. Barfourosh, H. R. Motahary Nezhad, M. O'Donovan-Anderson, D. Perlis. Information Integration in World Wide Web and Active Logic: A survey and problem definition Technical Report, Computer Science Department, University of Maryland, 2001.

[5] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Proc. of the 8th International Worldwide Web Conference (WWW8)*, 1999.

[6] J. Rennie and A. McCallum, "Using reinforcement learning to spider the web efficiently," in *Proc. Inter-national Conference on Machine Learning (ICML)*, 1999.

[7] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, Marco Gori. *Focused Crawling using Context Graphs*, 26th International Conference on Very Large Databases, VLDB 2000, Cairo, Egypt, pp. 527–534, 2000.

Paper from: *Management Information Systems 2002*, CA Brebbia and P Pascolo (Editors).
ISBN 1-85312-907-0

[8] J. Cho, H. Garcia-Molina, Lawrence Page. Efficient Crawling through URL ordering. Seventh International Web Conference (WWW). Brisbane, Australia, April 14-18, 1998.

[9] S. Chakrabarti, M. van den Berg, B. Dom. Distributed Hypertext Resource Discovery Through Examples, Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[10] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabelled doc-uments using EM. *Machine Learning*, 1999

[11] John. M. Pierre, On Automated Classification of Web sites, Linkoping University Electronic Press, Sweden, http://xxx.lanl.gov/abs/cs.IR/0102002 1 Feb 2001.

[12] W. Cohen, A. McCallum, D. Quass. Learning to understand the web. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2000.

[13] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghvan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. VLDB Journal, Aug. 1998.

[14] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, Marco Gori. *Focused Crawling using Context Graphs*, 26th International Conference on Very Large Databases, VLDB 2000, Cairo, Egypt, pp. 527–534, 2000.

[15] K. Purang, D. Purushothaman, D. Traum, C. Andersen, D. Traum and D. Perlis. Practical Reasoning and Plan Execution with Active Logic. *IJCAI'99 Workshop on Practical Reasoning and Rationality.* 1999.

[16] *Assessing others' knowledge and ignorance.* Kraus, S. and Perlis, D.. Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems. 1989 . 220-225.

[17] D. Perlis and K. Purang and C. Andersen. "Conversational Adequacy: Mistakes are the essence." International Journal of Human Computer Studies. 553--575. 1998.

[18] David Traum, Carl Andersen, Yuan Chong, Darsana Josyula, Michael O'Donovan-Anderson, Yoshi Okamoto, Khemdut Purang and Don Perlis. Representations of Dialogue State for Domain and Task Independent Meta-Dialogue. *Electronic Transactions on AI*, forthcoming

[19] J. Gurney, D. Perlis and K. Purang, Interpreting Presuppositions Using Active Logic: From Context to Utterances. *Computational Intelligence.* 1997.

[20] W. Chong, M. O'Donovan-Anderson, Y. Okamoto and D. Perlis. Seven Days in the Life of a Robotic Agent. GSFC/JPL Workshop on Radical Agent Concepts, January 2002, NASA Goddard Space Flight Center, Greenbelt, MD, USA