Distributed NLP and Machine Learning for Question Answering Grid

Daniel Sonntag

DaimlerChrysler Research and Technology RIC/AM, 89013 Ulm Germany
daniel.sonntag@daimlerchrysler.com

Abstract. We regard question answering as Semantic Grid application in which the answering process is best expressed as a distributed computing task and show, how the workflow control of this distributed QA task can be learned automatically. Since the control protocol contains information about each resource and service, this information can be mined to reveal semantics about the components. We address the question, which question answering components should be applied for a specific query. The inferred knowledge is machine-processable and is part of the workflow control of the distributed application. We thereby address semantics in the Grid by planning techniques for Grid Computing. For the learning task, we suggest the usage of association rule mining algorithms.

1 Introduction

Large-scale natural language processing (NLP) problems require to use resources from many isolated individual group of researchers. This point is a clear framework for distributed resource sharing which leads, on a more abstract level, to carry out NLP processes as a community. In technical terms, it leads to using a distributed Grid computing architecture. In this paper we propose a possible connection between Distributed NLP, Java Technologies for Distributed Computing and Machine Learning (ML) techniques for Question Answering (QA). In a natural process, knowledge bases and processing components for natural language are loosely coupled, instead of hard-wired. We will discuss the learning of a distributed QA system, more precisely, the mining of the master control protocol that logs the order in which the processing and language resources (confer Computational-Grid vs. Data-Grid) are to be applied during the learning phase. We then argue in favour of such an architecture by explaining how to benefit from the inferred meta data of Grid components. In this question answering process, the question is which components are to be applied in which order. Expressing the QA process as a distributed computing task, we allow for parallel processing and do not restrict the order by a predefined stream of QA components. After applying data mining methods to the control protocol, we can decide on the most suitable components for answering a specific query.

1.1 Survey on Question Answering

Question Answering is the task of finding answers to natural language questions by searching large document collections. Unlike information retrieval systems, question answering systems do not retrieve documents, but instead provide short, relevant answers located in small fragments of text.
In NLP and QA, computational complexity of sentence parsing constrained by complex linguistic grammars has always been an issue. Unfortunately, computational intensive components are part of most good performing state-of-the-art QA systems. On that account question answering is a good application for our approach, not to forget that QA incorporates a document retrieval stage, which is in functionality very similar to the Data-Grid. Modern QA systems define different processing steps for different QA types (e.g. fact-based questions, template-based questions, thematic-oriented questions). Because of the variety of processing steps and components involved (see section 2.1) and the different possibilities for answering a query, QA can be a complex, composite process, for which the best way of selecting and applying single components is not obvious and for which no closed approach is known. In addition, data access, data availability and good performances of single QA components cannot be guaranteed for all accepted natural language questions, which make the QA workflow and best solution rather undeterministic.

The central idea to enhance performance and robustness is the following described in [1]. The idea is that processing steps can be varied depending on the complexity of the query, i.e. shallow and deep QA strategies can be selected corresponding to different levels of linguistic processing. Using different processing stages for different query types is good to answer simple fact-based question for which shallow QA strategies work quite well. This was shown for Person, Location, Date and Quantity questions in [2] and in [3]. On the other hand, robustness to accept more difficult queries (e.g. queries for entity classes such as duration and measures, very short queries, very long queries, ungrammatical queries, queries containing unrecognised tokens, queries where no appropriate answer is in the knowledge base) and scalability in terms of efficient processing cannot be achieved by this coarse-grained distinction. The major reason and central idea of this paper is, that the computational and conceptual difficulty to answer a certain query is more a question of the availability of the processing and information resources to answer the question, and less a question of the type of the posed query or similar characteristics based on the query. In our distributed QA architecture, we address exactly this question. We argue, that the best way to answer a query is to decide on the QA strategy not only by a query/answer type dependent feature at an early stage of the QA process, but during processing in a freely coupled architecture which takes the (current) availability of resources into account and treats every question instance individually.

Next, we give an account of the underlying software principles and patterns, before we draw attention to the connection between the QA requirements and the conceptual and computational processing resources.

1.2 Distributed Computing and JavaSpaces

We define Distributed Computing as co-operation of several computers working together on a particularly processing-intensive problem. A single computer accounts for local processing needs and is linked toward other computers by a communication network. The object-oriented view of distributed computing is that several component objects (e.g. Java objects) work on the same problem which, in turn, is implemented as an exchange object. We propose *JavaSpaces* [4] for this task. The JavaSpaces model involves persistent object exchange areas in which remote processes can co-ordinate their actions and exchange data, thus providing a necessary ubiquitous, cross-platform framework for distributed computing. The benefits of distributed computing, such as increased computation power with parallel CPU's can be encountered, too. Especially for QA, the benefits in scalability, resource sharing and availability of resources come into account. Scalability is here to be understood in terms of adding new QA components (possibly lightweight annotators or databases) to match the size of a single question processing problem or parallel questions processing problem and especially the answer retrieval problem: If the amount of data exceeds the processing power or the document server, we can simply replace a component or add a new server without changing the client application and without changing the other servers. This addresses the question how to manage resources possibly distributed over many sides in a generic Grid infrastructure [5], but does not address the computational resource problem properly. Although different QA processes can be started in parallel[1], it is a very valuable information to know, which process workflows perform best for a given query instance. This might be unknown in advance, or unexpected because of e.g. unexpected availability problems of resources or unexpected problems of coverage. We will address these questions in Sect. 3.

2 QA Components

We introduce the QA components, how they are modelled and which kind of input and output behaviour they show. Declarations on this questions are vital, we need a classification of interchangeable, exchangeable, and decomposable processing components to define the workflow for QA and the possible variations in the workflow. We begin by a definition what exactly is meant by QA processing components. In our Grid architecture the components will have the following

[1] Deep NLP components could be started in parallel and will only be used when appropriate and delivered in time.

form: We will adopt the following terminology [6] to refer to special types of NLP components. *Language Resources (LRs)* refer to data-only resources such as lexicons, corpora, thesauri or ontologies. *Processing resources (PRs)* refer to resources whose character is principally programmatic or algorithmic, such as part-of-speech tagger, named-entity-recognisers, or sentence parsers. PRs typically include LRs such as a lexicon. For the JavaSpace QA architecture, both LRs and PRs are to be considered as possible QA components.
Focusing on PRs, the requirements stated in [6] look like the following - we will directly relate them to the requirements of a distributed system architecture: Families of components share certain characteristics which should be modelled. In our architecture, this modelling plays the central role and is directly related to the PR Management: If a component falls into a certain family of components (the equivalence classes of PRs/LRs), it can be replaced by an equivalent component in the sense of its input/output behaviour. On the other hand, the equivalent component might be more suitable for the specific query instance. The idea we propose in this paper is to mine the workflow protocol of suitable components based on the component communalities and individual component properties. We want to decide on suitable individual components and component classes and try to learn, which components should be applied for a query instance, learned by the output of the QA components (and the overall result) on the test instances. Additional meta data of the components or their interaction are not compulsory, but desirable, e.g. information about the performance of a component or component class on a certain class of questions, such as Person-Question. The component communalities described in the next section plus the meta data they induce during processing initiate the ML task we identify and present in Sect. 3.

2.1 Distributed Question Answering Grid Architecture

The central piece of the software architecture is a JavaSpace [4] that serves as NLP/QA data exchange area. Components can deposit data and wait for other data to process. The central issues are data communication and data synchronisation of data entries which tie together processes in a distributed program. We will provide the conceptual framework of their usage in the following: It turns out that one standard application scenario discussed in [4] accounts for the requirements of the QA process. Accordingly, the architecture consists of 1.) QA components as services of wide-spread network programs 2.) a JavaSpace data exchange area[2] and 3.) a master process that controls the application of individual components and decides when the answering process terminates and which result is to be returned. The concept can be seen in Fig. 1. While the QA process is running, every NLP/QA component listens for a task in its dedicated task area and performs an apparent task immediately (if allowed by the master

[2] The Javaspace itself is represented as data objects that can be independently accessed and altered by the QA components in a concurrent manner. The provided JavaSpace implementation takes care for transaction security and state consistency.

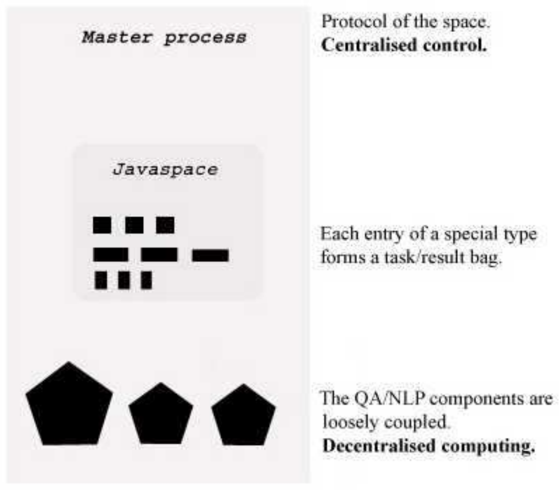
Master process
Protocol of the space.
Centralised control.
Javaspace
Each entry of a special type forms a task/result bag.
The QA/NLP components are loosely coupled.
Decentralised computing.

Fig. 1. JavaSpace Concept for Question Answering

control). The components thus communicate by the task and result areas within the JavaSpace. The crucial point is that the components only interact indirectly, through the space as data exchange area. This uncouples the data (i.e. partly results of components) in the space as well as the components themselves. This offers the additional freedom to choose components out of a set of equivalent components on a specific task to freely decide on the QA process workflow. The architecture proposed resembles the implementation of a classical blackboard pattern [7], which was already successfully applied to speech recognition problems [8]. With the help of JavaSpaces, the implementation is straightforward, and let us focus on the specific workflow control needed for QA.
Each of the processing components, which are connected to the JavaSpace, has to be started only at certain processing stages. This stages must be fulfilled chronologically, but can freely combine several PRs and LRs. We identify the following *QA processing stages*.

- Answer Type Detection
- Answer Template Filling
- Answer Document, Sentence, or Paragraph Retrieval[3]
- Answer Zooming[4]
- Answer Verification

Our concrete architecture consists of the following *top-level processing components* to implement the processing stages: A *Question Analyser QR*, a *Query*

[3] The retrieval stage often follows a text summarisation step.
[4] May include answer tiling.

Expansion component QE, a *Document Retrieval* Component DR, a *Passage Retrieval* Component PR, an *Answer Extractor* AE, and an *Answer Verifier* AV. To that effect the basic QA process workflow looks like this: All top-level processing components are connected to the PRs and LRs they require, which is illustrated in Fig. 2.

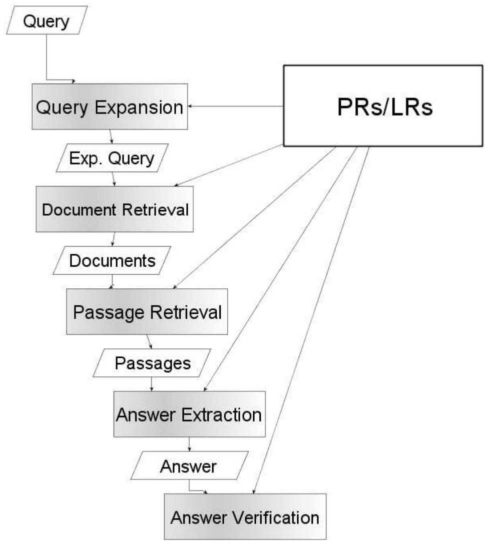

Fig. 2. Basic Question Answering Process Workflow

In addition, Fig. 2 shows, that the top-level processing components are not directly connected, but indirectly by a data input. The angular boxes represent the actual textual data input, e.g. the question string or the retrieved documents. These data items are represented as entries in the space. We will not address this issue here and focus again on the QA components. Every top-level component $p \in P = \{QE, DR, PR, AE, AV\}$ has a set of associated PRs and LRs, for example $associates(QE, \{PR1, LR1\})$, whereby $PR1$ could be a morphological analyser and $LR1$ an ontology. A very important point is, that every top-level component can have more than one associated set of PRs and LRs. It is not clear, which PRs or LRs should be applied at a certain stage, because they modify the input document and add possibly wrong or misleading information. It is e.g. not clear, whether document similarity should be calculated on filtered or unfiltered documents. We try to find out, which sets of associated PRs and LRs work best for specific queries, taking into account the associated sets of other top-level components. The set elements of different stages declare the valid processing streams of a QA process.

We will treat both PRs and LRs in the same way, because they exhibit the same input/output behaviour. Now think of a PR or LR component as an item set PR, and LR respectively, which we define recursively. Each PR may contain several other PRs and LRs. To keep thinks simple, each LR is atomic and defined by $\{LR\}$. Then PRs might have the form $\{PR, PR\}$, $\{PR, LR\}$, or $\{PR, \{PR, PR\}\}$ for example. Each embraced PR or LR defines a subtask and different top-level components may have the same subtasks associated. We then ascertain two problems which count as optimisation criteria.

Problem 1. Decide on the best performing set of associated PRs and LRs for each top-level component given a specific query.[5]

Problem 2. In addition, we have to decide, if the result of an executed PR can be reused at a later processing stage, like in dynamic programming, or if a PR has to be started again.

We prove the adequacy of the problems/optimisation criteria for QA: In a QA process, it might be advisable to skip a (preprocessing) PR or reverse the order in which some (preprocessing) PRs are applied to a query, as shown in [9] for tagging and partial parsing. In [9] a statistical sentence parser worked best if the tagging was postponed, when some parse trees delivered by a sentence tree parser already existed. This example shows variation of PRs order on a very low level. But even the more abstract QA workflow level can be varied or partly reversed. Experiments in [3] show that even though linguistic filters have been used thus far as post-processing filters, further improvements can be made by applying the filters at the document retrieval stage. In order to test on different QA streams, recent QA systems implement answer selection as a multi-stream [10] system. This poses another difficult question, how to select among the different answers returned by different streams. In our approach, we circumvent this problem by only allowing for one final answer, though it is not clear, whether this produces better results.

Consider the basic workflow in Fig. 2 again. While processing, the master process supervises whether a PR or LR is available over the distributed system, whether it should be applied at the current stage of processing (the PR/LR is element of the set associated with the current top-level component), and whether it returns a proper result in time. Additionally, the master process may take up control, whether a specific unsatisfactory subtask must be processed again. This introduces a third optimisation criterion for our system. Regarding top-level components we are interested in loops that may be reasonable. We then ascertain the following decision problem to be solved:

Problem 3. A decision must be made, if the process pipeline should be processed further, or returned to an earlier stage, i.e. a top-level component out of L, whereby L is the set of all top-level components, and PRs and LRs consulted for the specific question so far.

[5] This set can also be empty. This means, that we skip a processing stage. For example, in a simple fact-book lookup, query expansion is not appropriate.

This can best be explained in the case of the *Answer Verifier*. When the master control decides that a proper result cannot be returned yet, we must return to a former processing step, e.g. the document retrieval stage in order to enhance the recall (see e.g. [11]) of relevant documents. Consider that Problem 3 is similar to Problem 2 but concerns a different processing level, and is completely independent from Problem 2. Having defined the occurring problems and corresponding three optimisation criteria, we go on to look for how they can be expressed in the distributed question answering process. Consider again Fig. 1 and the last and most important part for discussion, the process control. The master process listens for the result entries from the result bags being filled by the components, decides on the next processing step(s) and delivers a final query result. What we aim to do is to exploit the protocols of successfully processed QA queries, that means all protocols derived from QA processes that produce the correct answer according to a supervised test set. For the rest of this paper, we concentrate on mining the protocol for the selection of appropriate sets associated with top-level components (Problem 1). The decision on a best performing set of PRs/LRs cannot be made reliably with our learning step. But in our case, an unreliable decision suffices, because the sets only define the plan to subsequent workflow steps in part and do not present a deterministic solution. The freely coupled architecture is still preserved and needs a criterion for termination (Problem 2), which we do not address in this paper.

3 Machine Learning and the QA Grid

Since the QA process involves a variety of single NLP processing resources such as tokeniser, morphology parser, NE recogniser, and chunk parser and since many of these have many adjustable parameters (e.g. HMM-based POS-Tagger), machine learning algorithms[6] are principally suitable to adjust these parameters. The majority of researchers developing state-of-the-art QA tools have focused on optimising single components, e.g. finer-grained NE rules or probabilistic grammar rules by ML techniques to improve their systems [12]. For some tasks, like detecting the expected answer type, one can easily agree with this option, since these tasks can be expressed as classical classification tasks that can be solved by SVMs, for example [13]. And the process of classifying a query is rather independent from the answer process, once the answer type is detected.
However, it cannot be assumed in principle that improving a single component also improves the overall performance as shown for passage retrieval [14]. To that effect, using machine learning to improve a single QA PR does not necessarily improve the QA system.

In our approach, we improve the QA system by mining the process control of the QA process, which reveals meta data about the input, the applied components (e.g. availability and performance), and the overall result. First, we recapitulate,

[6] Most of the methods are referred to as statistical NLP in the linguistic literature.

what we target to model and learn automatically, is the behaviour of the system to correctly decide which components are to be applied for specific query instances. In a plan-driven architecture, that means to learn to decide which PRs and LRs is to be applied at a certain workflow stage. Consider, that the application of LRs and PRs at a certain workflow stage is restricted by the LRs and PRs that suit this workflow stage. This must be specified in advance. For example, during query expansion, all components working on answer verification are rather useless. Now let, for example, $s(QE)$ be the set of suitable PRs/LRs for query extraction. Then $s(QE) \times s(DR) \times s(PR) \times s(AE) \times s(AV)$ defines the solution space.[7] The process protocol is mined for patterns that allow to minimise $s(QE) \cup s(DR) \cup s(PR) \cup s(AE) \cup s(AV)$ for a particular query instance (the particular words play a major role) or query class (the question type or semantic type of question headwords play a major role).
To accomplish this task, we need to know about the qualitative result of a specific workflow on a particular question. This qualitative result is basically represented by the correct or incorrect final answer. In addition, the qualitative result of each involved component can be measured. A recent approach in this direction is called introspection in QA systems, where intermediate results are evaluated to decide on feedback loops to redo a QA step which turned out to be unsatisfactory. In [15] this is used to decide whether the hit list of an underlying search engine contains an appropriate answer passage or not. Then it can be decided, if a new search (e.g. with the prior use of a query expansion operator) should be started or the question should be rejected. However, in our methodology the answering process is not predefined by a prior classification model that restricts the process pipeline. All we constrain is, that we have to follow the workflow illustrated in Fig. 2, but at each stage, all suitable PRs and LRs are considered in principle. During application, many different workflows may occur. We are interested in the set of involved components that constitute a successful workflow, which results in a correct answer. Then the set of possible LRs and PRs to be applied at a certain QA stage can be restricted to the most convenient subsets, which can all be started in parallel without computational explosion.

3.1 Master Control Protocol

The content of the master control protocol we propose is best explained by example. Imagine we pose the simple fact-based question *When was Frank Sinatra born?* and also know the answer *December 12, 1915*. Thus the control protocol might consist of the following information (Actually the log entry for one question would be much longer.)

1. (When) (was) (Frank) (Sinatra) (born)
2. {Date} ... {Person}
3. input#PR1#output#quality ...
   input#{PR2,{PR3, LR1}}#output#quality

[7] For simplicity, we do not regard loops here.

4. overall result quality

The entry log contains the question tokens and meta information about the expected answer type {Date} and that information about a person {Person} is being asked. In addition, the application of each PR and LR is traced. It is important to mention, that the input and output to and from each component is also logged. This information can later be explored to get quality information of each component in addition to the overall result quality. Next, we briefly sketch a suitable class of machine learning algorithms for that purpose, in which in contrary to classification and clustering, an association between features is sought.

Association Rule Learning. Association rule learning is a typical data mining technique, established methods can be found in [16], [17], [18]. Association rules are expressions like $X \Rightarrow Y$, where X and Y are disjoint sets of items. A transaction T is a set of items, given a database D of all transactions $T \in D$. We are primarily interested in the conditional probability $p(Y \subseteq T \mid X \subseteq T)$ which expresses the probability, that whenever a transaction contains X, it also contains Y. For mining the master control protocol, we declare that the log of a single answering process is an abstract transaction. After filtering out features in which we are not interested, we extract and load a (virtual) database Da of question answering transactions.

Learning of the Master Control Protocol. We try to bring forward association rule learning to the problem of mining interesting patterns from the QA master control protocol. Learning the master control protocol is the idea of modelling process control from the features space obtained from Da and to infer knowledge by inferring rules over the transactions $Ta \in Da$.
A second constraint we define concerns the process properties. For simplicity, we concentrate on a special type of patterns we are likely to find, patterns of flat simple items. For this purpose, we abstract from the process order which is logged in the control protocol, as well as from the inherent syntactic structure of PRs. For example, the processing component $PR1 = \{PR2, \{PR3, LR1\}\}$ is reduced to the process properties set {`PR1, PR2, PR3, LR1`}. We basically filter statistically relevant implications and association rules of the special sort:

```
[instance properties..., PRs..., LRs...] -> [successful answer]
```

We want to find out which input words, classes of input words like {Date} and {Person}, and PRs and LRs result in a successful answer process. Likewise, we are interested in implications, which additional PRs and LRs should be invoked in a consecutive fashion:

```
[PRs..., LRs...] -> [PRs..., LRs..., successful answer]
```

We briefly discuss the input words and input class properties, too. As such we take all query information available, that is the actual input word strings and all meta data available. We believe that some very interesting rules can be inferred with actual input words, at least question words and abstract terms. While running the association rule algorithm, we adopt a trick to filter out uninteresting patterns. We prune every rule in which not all the processes came up with qualitative results. Having produced a set of association rules, we directly relate them to the semantics of the actual Grid architecture. For first, the association rules are meant to reveal patterns about the strength of applying specific workflows and distributed components. For second, reconsider that every component stems from an equivalence class of components. By comparing the rules for each PR and LR in the same equivalence class, we get meta information about the exchangeability of components in the Grid. For last, the inferred meta data about the distributed components can be directly used by the master process to decide on the workflow of new incoming questions - consider a simple rule that gives direction to take by the system at an early processing stage `Person -> PR4 (Lookup in fact book)` that supposes a simple lookup in a fact book as data retrieval task.

In our opinion, the applicability of association rules is appropriate for this task. Although they are unfocused, may easily return noise and may need a lot of input to be useful, they can supply interesting patterns from the input space. We are furthermore only interested in association rules with few items, and by restricting rule generation to rules, which result in a correct answer, the final rules are much more focussed as in unsupervised experiments with association rules. In related tasks, even more complicated sequence patterns have proved to correctly predict failures in plan executions [19]. One point is left, to speculate on further developments in Grid semantics based on our QA system requirements. The last point is deferred to the outlook.

4 Conclusion and Outlook

We conclude that the high conceptual and computational demands of QA systems can be solved in a distributed computing architecture, for which JavaSpace technology provides the necessary concepts. We implement an opportunistic problem solving strategy for semantic Grid application, expressed as blackboard pattern. The Grid serves for specialised QA components to assemble their knowledge. The learned control protocol of the specialised subsystems, i.e. top-level components, PRs, and LRs, deliver the semantics for Grid Computing in Question Answering and show a methodology to assemble special purpose Grids via an automatic training approach. Results of the evaluation will be presented as soon as possible.

Finally, we give an outlook to further QA Grid potentials by the usage of taxonomic association rules: These rules account for taxonomies which embrace items hierarchically to item sets. Recall the definition of the PRs, which we defined accordingly. With the help of generalised association rules [20], rule items can have taxonomic structure and comprise items from different levels of abstraction, such as complex PRs. It is then possible to infer rules about more complex processing workflows. This knowledge allows for planning more than one subsequent step of processing, and hence allows for even better PR and LR co-ordination.

Acknowledgements. This work is partly supported by the EU IST FP6 004475 research project: Data Mining tools and services for Grid Computing Environments (DataMiningGrid). Thanks to Markus Ackermann, Christina Koepf and the reviewers for useful comments.

References

1. Neumann, G., Xu, F.: Mining answers in german web pages. In: Proceedings of International Conference on Web Intelligence, Halifax, Canada, IEEE/WIC WI-2003 (2003)
2. Abney, S., Collins, M., Singhal, A.: Answer extraction. In: Proceedings of Applied Natural Language Processing (ANLP) Conference. (2000)
3. Cardie, C., Ng, V., Pierce, D., Buckley, C.: Examining the role of statistical and linguistic knowledge sources in a general-knowledge question-answering system. In: Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000). Volume 180-187,ACL/Morgan Kaufmann. (2000)
4. Eric Freeman, Susanne Hupfer, K.A.: JavaSpaces Principles, Patterns and Practice. Addison Wesley (1999)
5. Foster, I., Kesselman, C.: The GRID. Morgan Kaufmann Publishers, Inc., San Francisco (1999)
6. Cunningham, H., Bontcheva, K., Tablan, V., Wilks, Y.: Software infrastructure for language resources: a taxonomy of previous work and a requirements analysis. Technical report, Department of Computer Science and Institute for Language, Speech and Hearing, University of Sheffield, UK (2000)
7. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture. A System of Patterns. John Wiley and Sons, West Sussex (1996)
8. Erman, L.D., Hayes-Roth, F., Lesser, V.R., Reddy, R.: The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. In: Proceedings of ACM Comput. Surv. 12(2). (1980)
9. Charniak, E.: Statistical parsing with a context-free grammar and word statistics. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI '97). (1997) 598–603
10. Jijkoun, V., de Rijke, M.: Answer selection in a multi-stream open domain question answering system. In: Proceedings of ECIR 2004, pp 99-111. (2004)
11. Lancaster, F.W.: Information retrieval systems: characteristics, testing and evaluation, 2 nd Ed. John Wiley and Sons, New York (1979)
12. Neumann, G., Sacaleanu, B.: A Cross-Language Question/Answering-System for German and English. Technical report, LT-Lab, DFKI, Saarbrücken, Germany (2003)

13. Zhang, D., Lee, W.S.: Question classification using support vector machines. In: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, ACM Press (2003) 26–32
14. Monz, C.: Document retrieval in the context of question answering. In: Proceedings of the 25th European Conference on Information Retrieval Research (ECIR-03), Springer (2003)
15. Czuba, K.: A machine learning approach to introspection in a question answering system (2002)
16. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Advances in knowledge discovery and data mining. In: Fast discovery of association rules. AAAI/MIT Press (1996) 307–328
17. Brin, S., Motwani, R., Silverstein, C.: Beyond market baskets: generalizing association rules to correlations. In: Proceedings of ACM SIGMOID. (1997) 265–276
18. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In Chen, W., Naughton, J., Bernstein, P.A., eds.: Proceedings of ACM SIGMOD Intl. Conference on Management of Data, ACM Press (2000) 1–12
19. Zaki, M.J., Lesh, N., Ogihara, M.: Planmine: Sequence mining for plan failures. In Agrawal, R., Stolorz, P., Piatetsky-Shapiro, G., eds.: Proceedings of Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98), New York, NY, ACM Press (1998) 369–373. A more detailed version appears in Artificial Intelligence Review, special issue on the Application of Data Mining, 1999
20. Srikant, R., Agrawal, R.: Mining generalized association rules. In: Proceedings of the 21st International Conference on Very Large Databases, Zurich, Switzerland (1995)