Distributed NLP, Java Technologies for Distributed Computing, and ML for Question Answering DRAFT

Daniel Sonntag
DaimlerChrysler Research and Technology, RIC/AM
89013 Ulm Germany

daniel.sonntag@daimlerchrylser.com

February 11, 2004

Abstract

In this paper we propose a possible connection between Distributed NLP, Java Technologies for Distributed Computing and Machine Learning techniques for Question Answering. In a natural process, knowledge bases and processing components for natural language are loosely coupled, instead of hard-wired, to take advantage of a synergistic effect that can be learned in supervised experiments on a meta level.

We begin with our view on QA as application which determines the workflow of the answering process. Then we express the QA process by a distributed computing task and show finally, how the control of this distributed QA task can be learned automatically.

Contents

1 Introduction: QA processing steps

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 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. At least, data access, data availability and good performances of single QA components cannot be guaranteed for all possible query instances in the normal case.

One way to enhance performance and robustness is the following example. The idea in [NX03] is that depending on the complexity of the query, the processing steps can be varied, i.e. shallow and deep QA strategies can be selected corresponding to different levels of linguistic processing. We will briefly present the strength and weakness of this approach. The idea to use different processing stages for different query types is good, for example, 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 [ACS00] and in [CNPB00]. *(Here a good example for deep NLP in QA in missing!* On the other hand, robustness to accept all queries (e.g. difficult entity classes such as duration and measures, very short queries, very long queries, ungrammatical queries, unrecognised tokens, no appropriate answer in knowledge base) and scalability in terms of efficient processing cannot be achieved by this coarse-grained distinction. The major reason 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. The QA system characteristics do count.

In the case of processing resources, the QA strategy is dependent on the suitability of an atomic or composite process to handle the task which can be expressed in terms of its *apriori* probability of returning a successful and well timed answer or its *aposteriori* online performance. Especially for the latter case, we have high expectations.

In the case of information resources, the QA strategy is dependent on the representation of the knowledge, the decisive question is whether the knowledge is explicit or implicit represented. If implicit represented, knowledge inference plays a major role which assumes both the query and the possible answers to be represented in a logical form (e.g. first-order logic predicates).[1] However, we do not address the question of knowledge representation and knowledge inference here and get back to QA strategies by the following assumption: The best way to answer a query is to decide on the QA strategy not by a query/answer type dependent feature at an early stage of the QA process, but during processing in a freely coupled architecture whose stages can be defined for each instance individually. So how can this be achieved? Of course, the approach must be flexible enough to decide which kind of linguistic processing steps are required and which components are suitable for a particular instance at certain processing stage. Flexibility can also be shown by leveraging the strictness of order in which the NLP components are to be applied.

At this stage, we informally introduced requirements for a QA system and hence the software architecture. Next, we light up the underlying software principles and patterns, before we draw attention to the connection between the QA requirements and the conceptual/computational processing resources.

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. We introduce *JavaSpaces* for this task. JavaSpaces is a powerful Jini service (network technology service) from Sun Microsystems, that facilitates building distributed applications for the Internet and Intranets. 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 [EF99].[2]

Several benefits of distributed computing, such as increased computation

[1]The relationship between logic-based IR models and inference models for QA is still worth exploring, but is left undetermined in this approach sketched here

[2]A distributed network architecture requires specific knowledge of network programming and with the help of JavaSpaces this effort can be reduced to a minimum.

power with parallel CPU's can be encountered. Especially for QA, the benefits in scalability, resource sharing and availability of resources come into account. Scalability is to be understood in terms of adding new QA components (possibly lightweight annotators or databases) to match the size of the question 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. Resource sharing is a related topic. It means basically that the answer documents are collected on a complete different server as e.g. the NLP components. Availability of data/components can be guarantied by a distributed architecture, if several server are available for the same NLP task and the distributed application chooses the best available server.

The benefits don't come along without challenges. In a distributed network, the communication between network points may be too slow (latency). This can occur, when many interactions between several components is the normal case, if a special order of processing and interaction between components does occur (maybe the case for QA). This leads to the problem of synchronisation. Distributed systems run without the control, which process terminates first, before another can be invoked. If another process relies on the output of the first, they have to be synchronised, e.g. a QA step must be started on a consistent data state, and other loading or deleting actions have to be looked as long as the first process is running.[3] This topic will be addressed in section 4.4. We proceed with another problem in distributed systems: Just like a database query has to be committed (alternatively also partly if an atomic transaction action fails) to bring the database from one consistent state to the other, a distributed application must be robust if a component fails to return a (proper) result or returns it too late. In NLP, computational complexity of e.g. deep parsers constrained by HPS or LF grammars has always been an issue. Unfortunately, computational intensive components have to be part of good performing state-of-the-art QA systems. In the context of distributed application, the disadvantage of computational intensive components can be turned into a vantage, indirectly. With a suitable transaction protocol, deep NLP components can always be started in parallel, and will only be used when appropriate and delivered in time, which

[3]One can even think of a QA system with JavaSpaces as a distributed object-relational multimedia database. This adds potentials for multimedia processing in different environments at different places, but also introduces the challenges stated above.

is determined during processing. This point lefts a mark on important use cases in distributed computing architectures. We will introduce the complete set of QA relevant use cases in the next section in a formal way.

2.0.1 Use cases for distributed computing architectures

In this section we discuss use cases for distributed computing architectures in the context of QA. Many NLP problems, such as QA, can naturally and easily be expressed in a distributed architecture. Consider that a complex NLP or QA query can often be processed asynchronously and co-ordinating. (Here some examples would have been adequate)

The following use cases (cited from the more general use case desiderata in [CBTW00]) attract interest for QA in a distributed computing architecture:

- Support of Language Engineering R&D workers producing software and performing experiments.
- Localisation and internationalisation. The goal in [CBTW00] for this use case is to allow the use of the architecture in and for different languages. In our specific application context (special LRs for QA), localisation and internationalisation become much broader in scope, particularly in a distributed computing architecture in which those terms refer to basic distributed processing concepts, to run local processes over a network of geographically possibly far-off NLP servers. This use case implements recent NLP activities by implementing recent engineering requirements. *Grid technology is intended to take the concept of the [Internet] one stage further by allowing seamless access and use of distributed computing resources as well as information.*[4]

We will adopt the following terminology to refer to special types of NLP components. *Language Resources (LRs)* refer to data-only resources such as lexicons, corpora, thesauri or ontologies [CBTW00]. *Processing resources (PRs)* refer to resources whose character is principally programmatic or algorithmic, such as POS-Tagger, NERs or 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. We first focus on PR

[4]Work in progress by Ewan Klein, Miles Osborne and Lex Holt

requirements and associate them with our distributed computing architecture. Focusing on PRs, the requirements stated in [CBTW00] look like the following. We will directly relate them to the requirements of a distributed system architecture as can be achieved by the use of JavaSpaces technology. The requirements:

- **PR Management:** The developer or itself is able to choose a subset of available components and wire them together .
  In our proposed architecture, the predefined order of components to be applied is more relaxed, the only wires between components is their input and output behaviour. In section 2 we could already receive an impression which benefits and problems are to be expected with this approach.

- **Distributed Processing:** Components must be made available over a network for distributed processing. This requirement on the configuration of NLP component is the key point in the common infrastructure for both PRs and LRs. Furthermore, the idea of distributed NL processing finds as its successor the idea of distributed NL computing which forms the basis of the ML learning task described in section 3, as one of the main concerns in this paper.

- **Component Communalities:** Families of components share certain characteristics. These communalities 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 protocol of suitable components based on the component communalities is a concrete meta learning task[5]. The learning process hereby comes along with a s̈ilbing rivalryämong the components of the same class in a distributed architecture. The component communalities requirement initiates the ML task described in section 3.

[5]To avoid misunderstanding, here meta learning means to learn which components should be applied in which order, learned by the output of a single QA component on the test instances. To produce additional meta data of the components or their interaction is not compulsory, but desirable, e.g. information about the performance of a system component on a class of questions, or single question instances

The question, how to specify a component, i.e. the interface with one another for the distributed computing architecture is not discussed in this work, some very good ideas for NLP component state descriptions can be found at Evan Klein's homepage.

In the rest of this section, we sum up our ideas for connecting QA and JavaSpaces and plan how to proceed. We have heard that QA systems could be designed in a distributed computing architecture, and that QA components, just as other NLP resources may also be spread over a network. We conclude that JavaSpaces may become an important tool for realising component-based QA systems. Having discussed QA systems and JavaSpaces, we now have an idea, how they can be connected to build runnable QA applications.

In the next chapter, we will bring the JavaSpaces, QA systems and ML altogether in such a way that distributed QA systems can be learned a processing behaviour with unique and striking potentials in distributed computing environments.

3 Machine Learning and QA

Since the QA process involves a variety of single (NLP) processing components (tokeniser, morphology parser, NE recognition, 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 researcher developing state-of-the-art QA tools have subscribed to the view that optimising single components, e.g. finer-grained NE rules or probabilistic grammar rules by ML techniques can improve their systems [NS03]. For some task, like detecting the expected answer type, one can easily agree with this opinion, since this tasks can be expressed as a classical classification task that can be solved by SVMs, for example [ZL03], and the process of classifying the 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. To that effect, using machine learning to improve a single QA component (PR), does not necessarily improve the QA system. This can be explained as follows: The performance of a QA system can only be increased by ML techniques if, and only if the optimisation criterion of the ML process is equal to the optimisation criterion of the entire QA

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

process. This means informally, that the question answering process itself has to be learned, not the individual components regardless the processing level. And there is an additional aspect we show for preprocessing. It might be advisable to skip a (preprocessing) component or reverse the order in which some (preprocessing) components are applied to a query, as shown in [Cha97] for tagging and partial parsing. In [Cha97] 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 showed variation of processing component's order on a very low level. But even the more abstract QA workflow level can be varieted or partly reversed. Experiments in [CNPB00] show that even though linguistic filters have been used thus far as post-processing filters, further improvements might be made by applying the filters at the retrieval stage. With our approach, this question is answered by the QA system itself and its processing control. What can be said is, that the selection of PRs and LRs and the order in which the PRs are to be applied, is a matter of the actual words in the query on a fine-gained level or at least matter of the query type however defined on a coarse-grained level. The behaviour of the system to correctly decide which component is to applied when, is what we target to model and learn automatically. In a plan-driven architecture, that means to learn to decide which operator (component) is to be applied next in the workflow. For this task, you need to know about the qualitative result of a component. Before we concentrate on the components of the QA learning architecture, we try to bring this assumption and problem specification into formal means and computational terms. In the view of the proposed ML task, it means to overcome the inductive bias that a function for QA cannot be approximated by optimising the functions of the QA components. More precisely, we try to address a kind of language bias[7]: We do not assert that the optimisation function of any machine-learned component is not expressive enough, but the hypothesis space does not include all possible functions for the QA process, which is a combination of (function of) the QA components. This function should not only contain the input and output of a component, but also data about the quality. The function to be learned in then a meta function over all components and the component data and forms a meta learning task. For this task, you need to know about the quality of a component's result. A recent approach is called introspection in QA systems, where intermediate results are evaluated to decide on feedback loops to redo a QA step which turned

[7]The language for representing functions defines a hypothesis space that does not include all possible functions

out to be unsatisfactory. In [ea] 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. We would like to propose a methodology, in which the plan how to proceed is not predefined by a prior classification model that restricts the process pipeline. Exactly at this point aspects of meta learning occur and can be taken into account. The meta learner accounts for the question, how it can be assured that the QA components are loosely coupled together, while still preserving a control mechanism for process order and termination. Prior to this is the question

- How to represent and activate QA components? A derivate is,
- How can the components communicate?

We adress this questions in the beginning of the next section 4 .

In addition to this questions, we concern ourselves with the question how it can be assured that the QA components are loosely coupled, or in other words, how the activation, the input and output behaviour of the individual components can be controlled and how this control can be learned in order to maximise the function (objection function to be approximated) of the complete QA process.

We would like to propose a software architecture that accounts for this questions. On the implementation side, we will propose the control structure of a suitable JavaSpace implementation that focuses on the so-called Java Entries, on the ML side we will propose an idea to model feature spaces and to infer knowledge by inferring rules (modelling the component's meta knowledge by rules) over the features and propose hence a fundament for ML algorithm selection. Especially, in section 4.1 we explore, how the activation of the individual components can be controlled. In section 4.3 we explore how the control can be learned and define the feature space of the ML component.

4 QA components

It is sensible to first 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 PRs, and decomposable question types, to define the workflow for QA and the possible variations in the workflow[8]

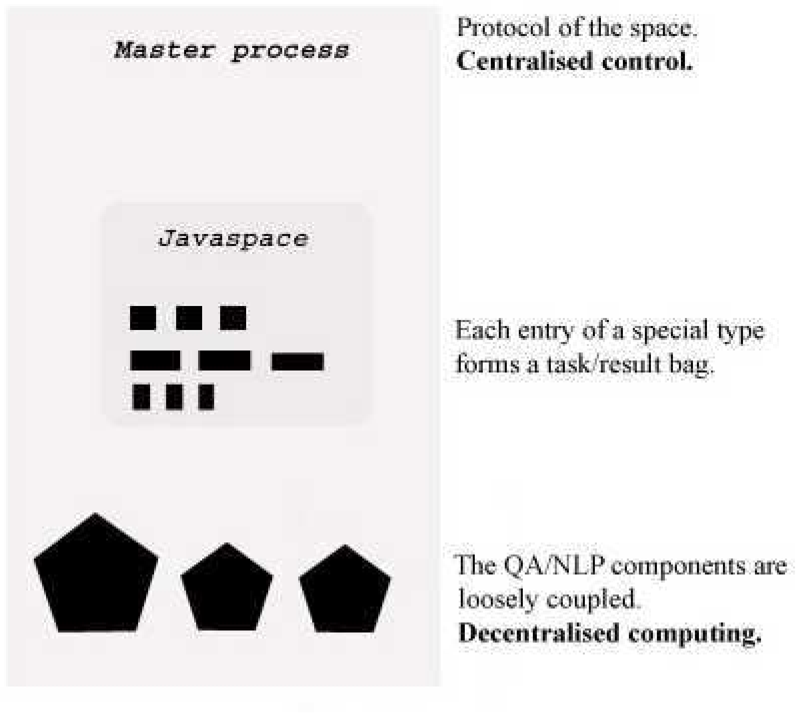
We will treat both PR components and LR components in the same way, because they exhibit the same input/output behaviour. They will be both represented as services. They will communicate by a task and result area. While the QA process is running, every component listens for a task in its dedicated task area and performs an apparent task immediately. The components thus communicate by the task and result areas. The crucial point is that the components only interact indirectly, through the data exchange areas. This uncouples them and offers the freedom to choose components (i.e. component results) out of a set of possible results on a specific task and to decide on the QA process workflow. We will now introduce the complete architecture to show how the data exchange area can be implemented and controlled.

4.1 Distributed QA architecture

In our distributed architecture, the actual QA components, as introduced in the previous section play a minor part. As desired and defined, they are quite exchangeable as far as new components implements the same interface of the special QA/NLP service.

The central piece in the (software) architecture is a JavaSpace that serves as NLP/QA data exchange area. Components can deposit data and wait for other data to process. The central issue apart from the already mentioned data communication aspect is the synchronisation of data entries that ties processes in a distributed program together. We will provide the conceptual framework for their use in the following. It turns out that one standard application scenario (discussed in [EF99]) accounts for the requirements of the QA process. Accordingly, the architecture consists of QA components as services of wide-spread network programs, a JavaSpace data exchange area and a master process that controls the results of the individual components and decides which result has to be returned. It is time to illustrate how the architecture looks like:

[8]Ewan Klein declares that an ontology of data descriptions induces an ontology of components. This ontology could serve as basis for interchangeability and exchangeability of PRs.


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.

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 (see [EF99] again).
Processing a query, several scenarios may occur. In one scenario, the query is decomposed into subtasks and the task bags are be filled accordingly. The master process then listens for the result entries from the result bags (filled by the components) and puts the results together to deliver a query result. In this case, we investigated parallel computing to deliver partly results of the query. The master process takes control, which subtasks are to be fulfilled (Note, that the order in which the results are returned is not important). Of course, this scenario only occurs, if a decomposable query was posed. On the other hand, in all query cases we come up with one or more subtasks to be fulfilled. We figure out the following subtask, the list is still to be complemented [9]:

- Answer type detection
- Answer template filling

[9]We do not assume an ontology of tasks and components so far.

- Retrieval of relevant documents, sentences or paragraphs (possibly plus text summarisation)
- Answer zooming
- Answer verification (possibly plus answer tiling)

It is to be mentioned that every subtask may consist of other subtasks. In our proposed architecture, we content ourselves with that abstract processing level. The master process takes control, whether a specific subtask was processed satisfactory. The definition of a subtask. i.e. which NLP components are to be applied in which order will be predefined and the most promising workflows will be listed. For example, for detecting the answer type, a shallow and a deep process could be started and the order of tagging and partial parsing could be changed into different workflows. The predifined workflow are expected to have high success probabilities (apriori prob.), of course. While processing, the master process supervises whether a PR or LR is available over the distributed system and whether it returns a proper result in time. The strenght of this appraoch is the control of the master process, where the ML approach begins.

In subsequent steps, this control is to be expanded towards component level, that is the control of single steps in subtasks by the master control process. So far.

4.2 Master control protocol

4.3 Learning of the master control protocol

4.4 Synchronisation

5 Conclusion and outlook

We conclude so far that the high conceptual and computational demands of QA systems can at best be solved in a distributed computing architecture, for which JavaSpace technology provides the necessary concepts.

The outlook is deferred to the possibility of recurrent user involvement in the QA process. Unstructured natural language queries are often difficult to formulate or ambiguous. Consequently, the updated or redesign of the query language should also be taken into consideration. The master control could detects e.g. poor retrieval performances of relevant documents in activated LRs. Apart from the request to the user to reformulate the query, also new query types are conceivable to meet the high demands for specific requests. Since natural language queries are more user-friendly, the alternative query types (e.g. multimodal queries, template-based queries, ...) are only invoked on system demand - under the control of a master process in a distributed system.

References

[ACS00] Steven Abney, Michael Collins, and Amit Singhal. Answer extraction. In *Applied Natural Language Processing (ANLP)*, 2000.

[CBTW00] Hamish Cunningham, Kalina Bontcheva, Valentin Tablan, and Yorick Wilks. 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.

[Cha97] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI '97)*, pages 598–603, 1997.

[CNPB00] Claire Cardie, Vincent Ng, David Pierce, and Chris Buckley. 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.

[ea] Krzysztof Czuba et al. A machine learning approach to introspection in a question answering system.

[EF99] Ken Arnold Eric Freeman, Susanne Hupfer. *JavaSpaces Principles, Patterns and Practice.* Addison Wesley, 1999.

[NS03] Günter Neumann and Bogdan Sacaleanu. A Cross-Language Question/Answering-System for German and English. Technical report, LT-Lab, DFKI, Saarbrücken, Germany, 2003.

[NX03] GÃ$\frac{1}{4}$nter Neumann and Feiyu Xu. Mining answers in german web pages. In *International Conference on Web Intelligence*, Halifax, Canada, 2003. IEEE/WIC WI-2003.

[ZL03] Dell Zhang and Wee Sun Lee. Question classification using support vector machines. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 26–32. ACM Press, 2003.