On the Role of Information Retrieval and Information Extraction in Question Answering Systems

Dan Moldovan[1] and Mihai Surdeanu[2]

[1] Human Language Technology Research Institute, University of Texas at Dallas
moldovan@seas.smu.edu
[2] Language Computer Corporation, Dallas, Texas
mihai@languagecomputer.com

Abstract. Question Answering, the process of extracting answers to natural language questions is profoundly different from Information Retrieval (IR) or Information Extraction (IE). IR systems allow us to locate relevant documents that relate to a query, but do not specify exactly where the answers are. In IR, the documents of interest are fetched by matching query keywords to the index of the document collection. By contrast, IE systems extrat the information of interest provided the domain of extraction is well defined. In IE systems, the information of interest is in the form of slot fillers of some predefined templates.
The QA technology takes both IR and IE a step further, and provides specific and brief answers to open domain questions formulated naturally. This paper presents the major modules used to build IR, IE and QA systems and Shows similarities, differences and possible trade-offs between the three technologies.

1 Information Retrieval

The three main technologies used to extract information from large collections of documents are Information Retrieval (IR), Information Extraction (IE), and Question Answering (QA). In this paper we first review briefly the state-of-the-art in each field, compare the similarities and differences between the main building modules, and then explore possible ways of combining the three technologies and performing trade-offs for various application domains.

The goal of Information Retrieval (IR) systems is to extract documents that best match a query. The two main tasks in IR are document indexing and searching. Indexing is the task of representing a document by its key features for the purpose of speeding up its finding when a query is invoked. There were many indexing schemes explored, but the most commonly used are based on word stemming and its enhancements. Term weighting is an indexing technique that gives a degree of importance to a word in a description. Word proximity, especially adjacency, is frequently used to Capture some of the linguistic relations between words. Some advanced indexing methods take into consideration compound terms and phrases, that is groups of words that collectively have a syntactic role. These are detected with shallow syntactic parsing and semantic analysis.

The goal of searching is to locate the most relevant documents and to rank them in Order of decreasing match with a query. Thus, a similarity measure needs to be introduced for this. Similarity algorithms are usually based on term weighting which is by far the most important feature that controls the IR precision. There are two types of weights, initial weights and relevance weights. The first are used when a searcher presents a query to the system, while the later is used after the documents are retrieved for the purpose of ordering them. The most common type of initial weighting is inverse document frequency (*idf*) which assigns weights to the terms in a query such that the weights are in inverse Proportion to the frequency of occurrence of these terms in the document collection, namely, low frequency terms are likely to point to a relevant document. An improved method is *tf.idf* weighting, which multiplies the term frequency weights by the collection frequency weight.

Relevance feedback is the name given to methods that try to modify queries automatically, that is to re-weight queries by adding or deleting keywords, or expanding keywords. Conceptual indexing has been studied by [Woods 1997, Mihalcea 2001]. The idea is to index documents based on concepts not words. This involves word sense disambiguation, a Problem that has not been solved yet for open text.

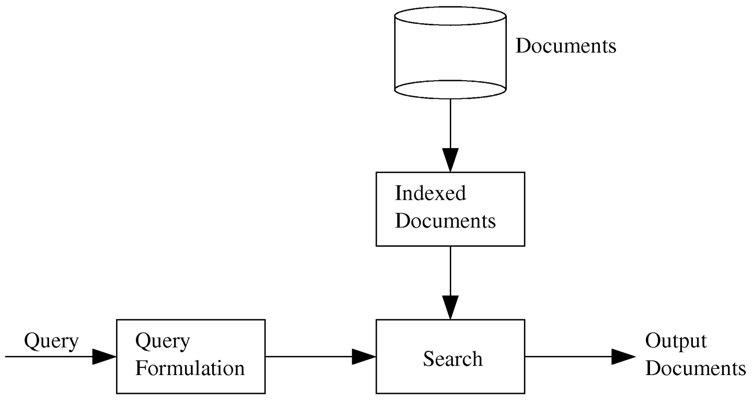

Fig. 1. A generic IR System Architecture

Figure 1 shows a high level block diagram of a generic IR architecture. Search is done by using an inverted file, which contains a dictionary file and postings file. The dictionary file contains lists of keywords, classification terms, journal titles that can be used as keywords. A count is associated with each entry in the dictionary file, it specifies how frequently a key occurs. The postings file contains a series of lists one for each of the entries in the dictionary file. Each such list contains the identifiers of all documents that contain a given term. This enables the IR system to search only those documents that contain query terms. The posting file also stores the term location which facilitates proximity search.

Inverted files provide rapid access to large document collections, but they require large storage overhead; index file may be twice as large as the document file.

Once the initial query terms are submitted they are looked up in the dictionary file to determine whether they have been used to index any of the documents in the collection. If they have, their frequency of occurrence in the database is extracted. Each entry in the dictionary file has a pointer to the corresponding list in the posting file. These lists are further processed to check operators such as AND, OR, NEAR and others. After the logic operations have been performed, a new list that contains the identifiers of all relevant documents is formed. Usually several iterations are necessary to return an appropriate size output file. The user has now access to the documents and if not satisfied another query is formulated.

2 Information Extraction

Information Extraction systems attempt to pull out information from documents by filling out predefined templates. The information typically consists of entities and relations between entities like who did what to whom, where, when and how. IE got a boost with the MUC (Message Understanding Conference) in late 1980's and early 1990's. The MUC IE systems focused on one domain at a time; for example joint ventures, terrorism in Latin America, management successions, and others. Users of such systems had to specify their information need in the form of a template with Slots that the system had to fill automatically.

Figure 2 shows templates filled by the CICERO information extraction on the MUC-6 management succession domain [Harabagiu and Surdeanu 2002]. A template can store information directly extracted from the document, e.g. organization names such as "Fox Inc." and most importantly can store information about the inter-template relations. As shown in Figure 2 each `SUCCESSION_EVENT` template stores one or more links to `IN_AND_OUT` templates, which describes the actual succession event: who seized/acquired the management position, with further links to `PERSON` and `ORGANIZATION` templates.

A typical IE system architecture is a cascade of modules, like those in Figure 3. A preferred method of implementation for these modules is finite state automata [Hobbs et al. 1996, Harabagiu and Surdeanu 2002]. The *tokenizer* is the first module of the IE system. The task of the tokenizer is to break the document into lexical entities (tokens). Generally, a token is a word or a punctuation sign, but in some cases it may be a word fragment. For example, the word "U.S.A." is broken into six tokens: "U", ".", "S", ".", "A", and ".". It is the task of the next module to identify the above sequence as a complex word representing an abbreviation of "United States of America".

The *Lexicon* module identifies words and complex words that have lexical and semantic meaning. This is done by inspecting both dictionaries and gazetteers. Dictionaries contain open-domain lexico-semantic information, e.g. "house" is an artifact, or to a lesser extent domain-specific information, e.g. "mail bomb" is a kind of bomb. Gazetteers typically store well-known entity names, such as locations, e.g. "Dallas" is a city in the state of "Texas" part of the country "United States of America".

```
<TEMPLATE-9301190125-1> :=
        DOC_NR:    "9301190125"
        CONTENT:   <SUCCESSION_EVENT-9301190125-1>
                   <SUCCESSION_EVENT-9301190125-2>
                   <SUCCESSION_EVENT-9301190125-3>
                   <SUCCESSION_EVENT-9301190125-4>
                   <SUCCESSION_EVENT-9301190125-5>
                   <SUCCESSION_EVENT-9301190125-6>
                   <SUCCESSION_EVENT-9301190125-7>
<SUCCESSION_EVENT-9301190125-1> :=
        SUCCESSION_ORG: <ORGANIZATION-9301190125-1>
        POST:           "chief executive officer"
        IN_AND_OUT:     <IN_AND_OUT-9301190125-1>
                        <IN_AND_OUT-9301190125-2>
        VACANCY_REASON: REASSIGNMENT
        COMMENT:        "Joseph M. Segel OUT, Barry Diller IN as
                         chief executive officer of QVC Network Inc."
<IN_AND_OUT-9301190125-2> :=
        IO_PERSON:      <PERSON-9301190125-1>
        NEW_STATUS:     IN
        ON_THE_JOB:     UNCLEAR
        OTHER_ORG:      <ORGANIZATION-9301190125-2>
        REL_OTHER_ORG:  OUTSIDE_ORG
        COMMENT:        "Barry Diller IN"
...
<ORGANIZATION-9301190125-2> :=
        ORG_NAME:       "Fox Inc."
        ORG_TYPE:       COMPANY
<ORGANIZATION-9301190125-1> :=
        ORG_NAME:       "QVC Network Inc."
        ORG_TYPE:       COMPANY
<PERSON-9301190125-1> :=
        PER_NAME:       "Barry Diller"
        PER_ALIAS:      "Diller"
        PER_TITLE:      "Mr."
<PERSON-9301190125-3> :=
        PER_NAME:       "Joseph M. Segel"
        PER_ALIAS:      "Segel"
        PER_TITLE:      "Mr."
...
```

Fig. 2. Sample templates for the MUC-6 management succession domain

The *Preprocessor* identifies simple lexical entries that are not stored in lexicon or gazetteers. Some of the items identified by the preprocessor are: phone numbers "1-(800) 88-7777", money "$4.75", dates "December 25", times "8:30am", measures such as "l0kg", "one hundred degrees", and others.

The *Name Entity Recognizer* is one of the most important modules in IE. It assigns lexical features to words or groups of words such as locations, organizations, persons, addresses, and others. Proper names are particularly useful for extraction systems since they point to objects about which we need to identify properties, relations, events. The technique is to use capitalization if available.

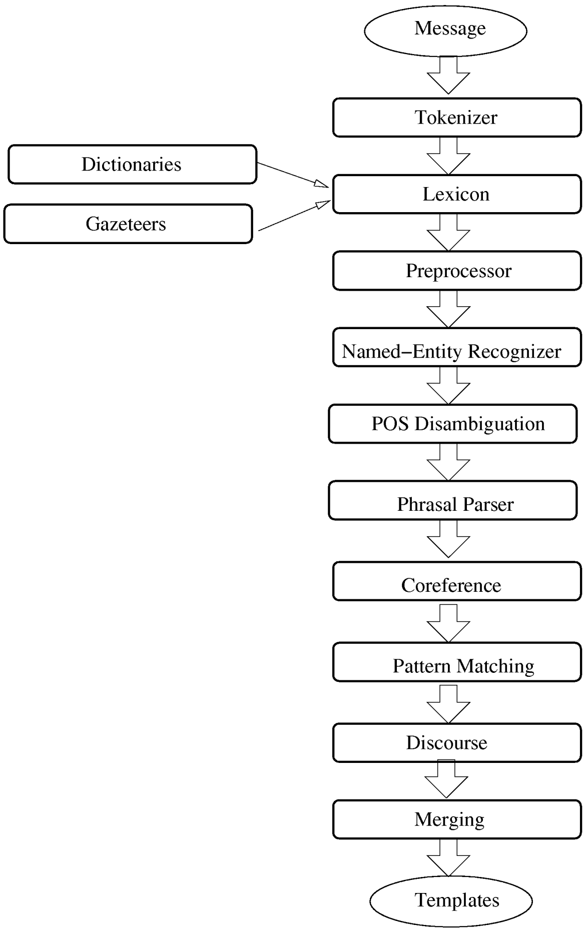

Fig. 3. An IE System Architecture

Some of the most frequently used methods are Hidden Markov Models and finite state automata patterns. With the help of dictionaries these techniques are able to recognize that "John Smith" is a proper name, and "John Hopkins" is a University; or that "Austin Ventures" is a Company, "Austin, Texas" is a City and "Austin Thomas" is a name. Machine learning methods are sometimes used to train the Name Recognizer in a new domain. NE-recognition benefits by morphological analysis by looking up in a dictionary for all morphological variations of words.

Part of Speech Tagging is useful for subsequent text analysis stages. This involves specifying the part of Speech of each word. POS taggers are rule-based or statistical and achieve an accuracy around 95%. The *Parser* identifies simple noun phrases (NP), e.g. "the fast red car", verb phrases (VP), e.g. "is being observed daily", and also particles that may be significant in subsequent text analysis. In many MUC systems only shallow parsing was used. It was more important to recognize NP or VP and less important to solve the attachment of prepositional phrases, or close subordination. The reasons for avoiding full syntactic parsing is the time taken by parsers, especially if sentences are long, and the possible errors that a parser may introduce.

Coreference Resolution is the task of determining that a noun phrase refers to the same entity as another noun phrase. This involves equating various forms of personal proper names, for example “President Bush”, “George Bush”, “the 43rd President of US”, etc. There are other more complex forms of coreference such as definite or indefinite noun phrase and pronoun coreference that some IE systems attempt to solve. There is need for temporal coreference resolution in which “today” from one document has to be related to “a week ago” in another document, but few systems have implemented this.

Pattern Matching is the basic method for extracting domain-specific information to fill the template slots. Considerable effort is put in developing domain-specific patterns that represent events specific to a domain. Typically these are subject-verb-object (SVO) patterns that are tailored to a domain by specifying semantic constraints that each component must meet. For example “Greenspan makes a recession” fits the pattern `<human, causes, entity>` while “Greenspan makes a mistake” does not fit the above pattern. This is decided by checking semantic constraints that the subject, verb, and object would have to satisfy collectively for pattern matching.

Domain Coreference is typically part of a *discourse analysis* module and attempts to fill empty template slots, which can be retrieved from the context of the pattern previously recognized. Consider the following example from the “natural disaster” domain:

> . . . flooding has become a way of life in Guerneville in the last several years. This gas station along the Russian River gets hit nearly every time. . . .

The domain pattern recognition module matches the `<disaster, destroys, artifact>` pattern over this text. Based on the semantics associated with this pattern, the following template is constructed:

```
<NATURAL_DISASTER> :=
    AMOUNT_DAMAGE: "this gas station"
    LOCATION: "Russian River"
```

Note that due to the inherent complexity of the natural language not all template slots can be covered through patterns. In this example a very important slot: the disaster type (“flooding”) is not recognized by the pattern. In such situations the domain coreference module inspects the pattern context for possible fills for the empty slots. In the above example the resulting template is:

```
<NATURAL_DISASTER> :=
    DISASTER: "flooding"
    AMOUNT_DAMAGE: "this gas station"
    LOCATION: "Russian River"
```

Merging is the task of combining and consolidating the information that refers to the same event by combining the templates that refer to the same events. Merging is a multi-step process. For example, first the templates that refer to the same events found in one sentence are merged, then the same is done at the document level, then at the collection level.

3 Question Answering

A QA system accepts questions in natural language form, searches for answers over a collection of documents and extracts and formulates concise answers. As shown in Figure 4, the three essential modules in almost all QA systems are *question processing, document retrieval*, and *answer extraction* and *formulation.*

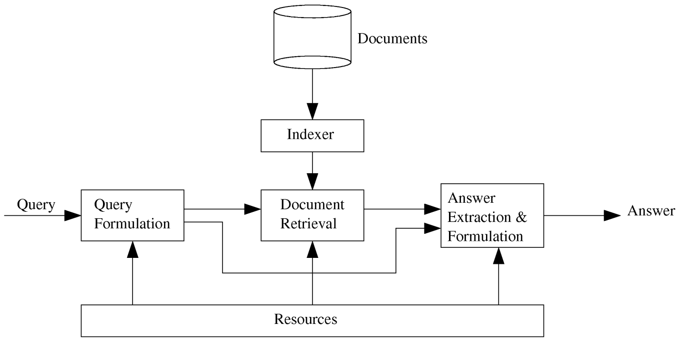

Fig. 4. A generic QA System architecture.

Just like MUC has boosted the IE technology, the Text Retrieval Conference (TREC) QA track has stimulated considerable interest and research in Question Answering in the last few years. In 2001, the US Government has initiated a new research program in QA, called Advanced Quest ion Answering for Intelligence (AQUAINT).

Since modern QA systems are open domain, the performance of a QA system is tightly coupled with the complexity of questions asked and the difficulty of answer extraction. For example, in TREC many systems were quite successful at providing correct answers to simpler, fact-seeking questions, but failed to answer questions that required reasoning or advanced linguistic analysis [Voorhees 1999]. From the combined set of 1460 evaluation questions, 70% of the participating systems answered successfully questions like Q1013: *"Where is Perth?"*, but none could find a correct answer to complex questions such as Q1165: *"What is the difference between AM radio stations and FM radio stations?"*

In order to put the QA technology into perspective, we first provide a broad taxonomy of QA systems. The taxonomy is based on several criteria that play an important role in building QA systems: (1) linguistic and knowledge resources,

(2) natura1 language processing involved, (3) document processing, (4) reasoning methods, (5) whether or not answer is explicitly stated in a document, (6) whether or not answer fusion is necessary.

Classes of Questions

Class 1. QA systems capable of processing factual questions

These systems extract answers as text snippets from one or more documents. Often the answer is found verbatim in a text or as a simple morphological variation. Typically the answers are extracted using empirical methods relying on keyword manipulations.

Class 2. QA systems enabling simple reasoning mechanisms
The characteristic of this class is that answers are found in snippets of text, but unlike in Class 1, inference is necessary to relate the question with the answer. More elaborate answer detection methods such as ontologies or codification of pragmatic knowledge are necessary. Semantic alternations, world knowledge axioms and simple reasoning methods are necessary. An example is Q198: *"How did Socrates died?"* where *die* has to be linked with *drinking poisoned wine.* WordNet and its extensions are sometimes used as sources of world knowledge.

Class 3. QA systems capable of answer fusion from different documents
In this class the partial answer information is scattered throughout several documents and answer fusion is necessary. The complexity here ranges from assembling simple lists to far more complex questions like script questions, (e.g. *"How do I assemble a bicycle?"*), or template-like questions (*"What management successions occurred at IBM in the past year?"*).

Class 4. Interactive QA systems
These systems are able to answer questions in the context of previous interactions with the user. As reported in [Harabagiu et al. 2001], processing a list of questions posed in a context involves complex reference resolution. Unlike typical reference resolution algorithms that associate anaphorae with a referent, the reference imposed by context questions requires the association of an anaphora from the current question with either one of the previous questions, answers or their anaphora.

Class 5. Speculative questions
The characteristic of these systems is their ability to answer speculative questions similar to:
"Is the Fed going to raise interests at their next meeting?";
"is the US out of recession?";
"is the airline industry in trouble?".

Since most probably answers to such questions are not explicitly stated in documents, simply because events may not have happened yet, QA systems from this class decompose the question into queries that extract pieces of evidence, after which answer is formulated using reasoning by analogy. The resources include ad-hoc knowledge bases generated from mining text documents clustered by the question topic. Associated with these knowledge sources are case-based reasoning techniques as well as methods for temporal reasoning, spatial reasoning and evidential reasoning.

Table 1. Distribution of TREC questions

Type	Number (%)
Class 1 (factual)	985 (67.5%)
Class 2 (simple-reasoning)	408 (27.9%)
Class 3 (fusion - list)	25 (1.7%)
Class 4 (interactive - context)	42 (2.9%)
Class 5 (speculative)	0 (0.0%)

Table 1 illustrates the distribution of TREC questions into the question classes. In addition to 1393 main-task questions collected from TREC-8, TREC-9 and TREC-2001, there are 25 list questions (e.g., *"Name 20 countries that produce coffee."*) and 42 context questions (e.g., *"How long was the Varyag?"; "How wide?"*).

QA System Architecture
First we identify the main modules of a QA system architecture, then present some architectural features seen in advanced QA systems. The model boundaries may change as some systems rely more on one method than another. There are ten modules presented below, the first five modules correspond to question processing, the next two modules perform document and passage processing, and the last three modules perform answer processing.

M1 The individual question words are spell-checked. Words like *Volkswangen* and *Niagra* are expanded into their spelling variants *Volkswagen* and *Niagara.* If necessary, questions such as Q885: *"Rotary engine cars were made by what Company?"* are rephrased into a normalized form where the wh-word (*what*) appears at the beginning, e.g. *"What company were rotarg engine cars made by?"*.

M2 The input question is parsed and transformed into an internal representation capturing question concepts and binary dependencies between the concepts [Harabagiu et al. 2000]. Stop words (e.g., prepositions or determiners) are identified and removed from the representation. For illustration, the representation for QO13: *"How much could you rent a Volkswagen bug for in 1966?"* captures the binary dependency between the concepts *rent* and 1966.
M3 The mapping of certain question dependencies on a WordNet-based answer type hierarchy disambiguates the semantic category of the expected answers [Pasca and Harabagiu 2001]. For example, the dependency between *How much* and *rent* for Q013 is exploited to derive the expected answer type *Money.* The answer type is passed to subsequent modules for the identification of possible answers (all monetary values).

M4 Based mainly on part of speech information, a subset of the question concepts are selected as keywords for accessing the underlying document collection. A passage retrieval engine accepts Boolean queries built from the selected keywords, e.g. *Volkswagen* AND *bug.* The retrieval engine returns passages that contain all keywords specified in the Boolean query. Therefore keyword selection is a sensitive task. If the wrong question word (e.g. *much*) is included in the Boolean query (*much* AND *Volkswagen* AND *bug*), the retrieval is unsuccessful since the passages containing the correct answers are missed.

M5 Before the construction of Boolean queries for actual retrieval, the selected keywords are expanded with morphological, lexical or semantic alternations. The alternations correspond to other forms in which the question concepts may occur in the answers. For example, *rented* is expanded into *rent.*

M6 The retrieval engine returns the documents containing all keywords specified in the Boolean queries. The documents are then further restricted to smaller text passages where all keywords are located in the proximity of one another. Each retrieved passage includes additional text (extra lines) before the earliest and after the latest keyword match. For illustration, consider QOO5: *"What is the name of the managing director of Apricot Computer?"* and the associated Boolean query *Apricot* AND *Computer* AND *director.* The relevant text fragment from the document collection is *"Dr Peter Horne, managing director of Apricot Computers"*. Unless additional text is included in the passages, the actual answer *Peter Horne* would be missed because it occurs before all matched keywords, namely *director*, *Apricot* and *Computer.*

M7 The retrieved passages are further refined for enhanced precision. Passages that do not satisfy the semantic constraints specified in the question are discarded. For example, some of the passages retrieved for Q013 do not satisfy the date constraint *1966.* Out of the 60 passages returned by the retrieval engine for QO13, only two passages are retained after passage post-filtering.

M8 The search for answers within the retrieved passages is restricted to those candidates corresponding to the expected answer type. If the expected answer type is a named entity such as MONEY, the candidates (*$1, USD 520*) are identified with a named entity recognizer. Conversely, if the answer type is a DEFINITION, e.g. Q903: *"What is autism?"*, the candidates are obtained by matching a set of answer patterns on the passages.

M9 Each candidate answer receives a relevance score according to lexical and proximity features such as distance between keywords, or the occurrence of the candidate answer within an apposition. The candidates are sorted in decreasing Order of their scores.

M10 The system selects the candidate answers with the highest relevance scores. The final answers are either fragments of text extracted from the passages around the best candidate answers, or they are internally generated.

Performance Evaluation

A QA system with this baseline linear architecture [Moldovan 2002] was tested on 1460 questions collected from TREC-8, 9 and TREC-2001. Answers were extracted from a 3 Gbyte text collection containing about 1 million documents from sources such as Los Angeles Times and Wall Street Journal. Each answer has 50 bytes.

The accuracy was measured by the Mean Reciprocal Rate (MRR) metric used by NIST in the TREC QA evaluations [Voorhees 1999]. The reciprocal ranking basically assigns a number equal to 1/R where R is the rank of the correct answer. Only the first 5 answers are considered, thus R is less or equal to 5. When the system does not return a correct answer in top 5, the precision score for that question is zero. The overall system precision is the mean of the individual scores. System answers were measured against correct answers provided by NIST.

Table 2. Distribution of errors per System module

	Module	Module definition	Errors (%)
QP	(M1)	Keyword pre-processing (split/bind/spell check)	1.9
	(M2)	Construction of internal question representation	5.2
	(M3)	Derivation of expected answer type	36.4
	(M4)	Keyword selection (incorrectly added or excluded)	8.9
	(M5)	Keyword expansion desirable but missing	25.7
DR	(M6)	Actual retrieval (limit on passage number or size)	1.6
	(M7)	Passage post-filtering (incorrectly discarded)	1.6
AP	(M8)	Identification of candidate answers	8.0
	(M9)	Answer ranking	6.3
	(M10)	Answer formulation	4.4

The inspection of internal traces, at various checkpoints inserted after each module reveals the system errors for each evaluation question. The goal in this experiment is to identify the earliest module in the chain (from left to right) that prevents the system to find the right answer, i.e.cCauses the error.

As shown in Table 2, question pre-processing is responsible for 7.1% of the errors distributed among module M1 (1.9%) and M2 (5.3%). Most errors in module M2 are due to incorrect parsing (4.5%). Two of the ten modules (M3 and M5) account for more than half of the errors. The failure of either module makes it hard (or impossible) for subsequent modules to perform their task. Whenever the derivation of the expected answer type (module M3) fails, the set of candidate answers identified in the retrieved passages is either empty in 28.2% of the cases (when the answer type is unknown) or contains the wrong entities for 8.2% (when the answer type is incorrect). If the keywords used for passage retrieval are not expanded with the semantically related forms occurring in the answers (module M5), the relevant passages are missed.

The selection of keywords from the internal question representation (module M4) coupled with the keyword expansion (module M5) generate 34.6% of the errors. Both these modules affect the output of passage retrieval, since the set of retrieved passages depends on the Boolean queries built and submitted to the retrieval engine by the QA system.

Modules M6 and M7 are responsible for the retrieval of passages where answers may actually occur. Their combined errors is 3.2%. In module M6 there are parameters to control the number of retrieved documents and passages, as well as the size of each passage.

Answer processing is done in modules M8 through M10. When the expected answer type is correctly detected, the identification of the candidate answers (module M8) produces 8.0% errors. 3.1% errors are due to named entity recognition (incomplete dictionaries) and 4.9% are due to spurious answer pattern matching. Modules M9 and M10 fail to rank the correct answer within the top 5 returned in 10.7% of the cases. Module M9 fails if the correct answer candidate is not ranked within the top 5, whereas M10 fails if the returned answer string is incomplete, namely it does not fit within 50 bytes.

A More Advanced QA System Architecture

The results presented in previous sections correspond to the serialized baseline architecture. Such an architecture is in fact a simplified version of our system which uses several feedbacks to boost the overall performance [Moldovan 2002].

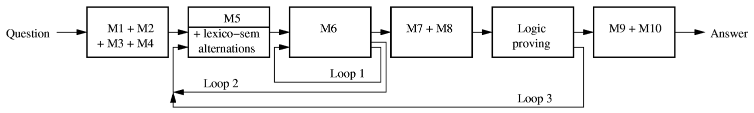

Fig. 5. Architecture with feedbacks

As shown in Figure 5, the architecture with feedbacks extends the serialized architecture in several ways. Keyword expansion (module M5) is enhanced to include lexico-semantic alternations from WordNet. A new module for logic proving and answer justification is inserted before answer ranking. In addition, three loops become an integral part of the system: the passage retrieval loop (loop 1); the lexico-semantic loop (loop 2); and the logic proving loop (loop 3).

Table 3. Impact of feedbacks on precision

Feedback added	Precision (MRR)	Incremental enhancement
none	0.421=b	0%
Passage retrieval (loop 1)	0.468=b_1	b+11%
Lexico-semantic (loop 2)	0.542=b_2	b_1+15%
Proving (loop 3)	0.572=b_3	b_2+5%

As part of loop 1, the Q/A system adjusts Boolean queries before passing them to the retrieval engine. If the output from the retrieval engine is too small, a keyword is dropped and retrieval resumed. If the output is too large, a keyword is added and a new iteration started, until the output size is neither too large, nor too small. When lexico-semantic connections from the question to the retrieved passages are not possible, loop 2 is triggered. Question keywords are replaced

Table 4. Summary of the main modules in IR, IE and QA

Subsystem	Module	IR	IE	QA
Question processing	keyword preprocessing	x		x
	question representation			x
	answer prediction			x
	keyword selection	x		x
	keyword expansion	x		x
document indexing and retrieval	document indexing	x	y	x
	document search and retrieval	x	y	x
	document ranking	x	y	x
document processing	morphological and lexical proc		x	x
	extract relevant passages		x	x
	syntactic parsing		x	x
	name entity recognition		x	x
	coreference		x	y
	discourse processing		x	x
	semantic analysis			x
use of world knowledge	WordNet		x	x
	dictionaries		x	x
use of domain knowledge	domain ontologies		x	x
	domain patterns		x	
	domain coreference		x	
	domain event merging		x	
output extracting	patterns		x	x
	complex nlp techniques		x	x
	merger		x	x
	answer ranking			x
	logic prover			x
	answer justification			x
output formatting	template filling		x	
	answer formulation		y	x

with WordNet-based alternations and retrieval is resumed. Loop 3 relies on a logic prover that verifies the unifications between the question and logic forms. When the unifications fail, the keywords are expanded with semantically related alternations and retrieval resumes.

Table 3 illustrates the impact of the retrieval loops on the answer accuracy. The knowledge brought into the question answering process by lexico-semantic alternations has the highest individual contribution, followed by the mechanism of adding/dropping keywords.

4 A Global View of IR, IE, and QA

As we have Seen, there are some modules common to more than one technology. A global view of this is shown in Table 4. We marked with `x` features that are fully supported, and with `y` features that are only partially supported. For example, advanced QA systems have answer formulation and generation features, whereas IE systems have only some limited form of text generation based on the template fills (see the `COMMENT` template fields in Figure 2).

5 Trade-Offs in QA Systems

As Table 4 indicates, Question Answering systems use modules common with IR and IE. The recent QA literature contains reports on QA systems that emphasize either IR or IE, which indicates a rich possibility of trade-offs in implementing these systems.

IR-Based QA
An IR-based QA system is built on top of an IR engine. In this case, the Question Processing is reduced to question classification. Two basic methods were used to build IR-based QA systems:

- QA systems that retrieve relevant *documents* and extract answers directly from those documents [Kwok 20001. The IR module extracts and ranks documents using: coordinate matching, stemming, synonyms, identifies important words, proximity of question words, order of question words, capitalizations and quoted query words. Answer extraction and formulation is reduced to heuristic pattern matching which identifies entities such as person, places, dates, units (length, area, time, currency, population) by using some heuristics.
- QA systems that retrieve only relevant *passages* from documents and then extract answers by further processing those passages. The advantage of passage retrieval over full document retrieval is that the amount of text that needs to be processed is significantly reduced. The systems in this category used various techniques to retrieve passages ranging from statistical methods [Ittycheriah 2001], to identifying passages based on answer category determined from the question [Clarke 2000], or conceptual indexing coupled with relaxation ranking [Woods 2000].

QA Based on IE
IE-based QA systems use IE methods, such as named entity taggers and surface patterns, to extract answers. Not having an IR engine, these systems rely on an outside source to supply relevant documents. Answers are extracted by matching the question keywords to the documents supplied from outside. The system described in [Srihari 1999] uses a preliminary ranking of documents to find the most probable answers by counting how many unique keywords are contained in a sentence. A secondary ranking of documents is done based on the order in which the keywords appear in the question. Yet, a third ranking is used to allow for matching variants of key verbs. A QA system that is based exclusively on IE patterns is described in [Soubbotin 2001].

6 Applications

Each of the three technologies has been implemented in commercial applications systems. In this section we will explore a few ways of combining two of the technologies for the purpose of building more powerful knowledge management (KM) application systems.

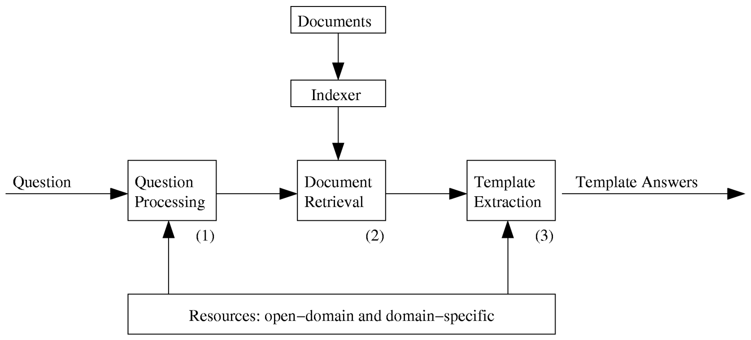

Fig. 6. IE Architecture extended with question processing and document retrieval

6.1 Domain Specific KM from Large Document Collections

This application answers domain-specific questions by finding domain specific events or entities from large document collections. From a technical point of view this system injects QA features into an IE system, such that: (a) the IE system understands natural language questions, and (b) large collections are accessed using passage retrieval techniques. For example, imagine a car insurance agency investigator who wants to find details about a certain accident:

"What accident took place in Dallas, at 3:30pm?"

The envisioned system architecture is shown in Figure 6. The system behavior is the following:

1. First the question must be processed. From the question answer type ("accident") the system identifies that the user is interested in an accident event, hence it will later enable only pattern rules relevant to this domain. The keywords are then selected just like in the QA system: "accident", "Dallas", "3:30pm".
2. The system retrieves paragraphs based on the set of keywords (similar to QA).
3. Answer processing is basically IE. Based on the question answer type, the corresponding domain (here "accident events") is enabled. Using IE techniques, the system fills in the templates extracted from the previously retrieved paragraphs, which become the answer as shown below.

ACCIDENT
    TYPE: "two cars collided"
    LOCATION: "NW Highway"
    TIME: "3:30pm"
    DAMAGE: "broken light, damaged fender"
    ...

Such a system can be applied to any domain-specific application where the answers tan be represented in a template form. There are many such applications:

a news agency agency could use the "bombing domain" to track the latest events in the Middle East, an intelligence agency could use the "people movement" domain to track the movements of important people, etc. These domain-oriented applications are very hard to be handled by open-domain QA systems, which might miss domain-specific information.

6.2 Advanced Open-Domain KM from Large Document Collections

This idea enhances the QA systems with some useful techniques implemented in IE systems. We refer mainly to the framework that identifies syntactic patterns, such as SVO (subject-verb- object) patterns or complex noun groups. This framework can be ported to a QA system to help extract answers that can be identified through syntactico-semantic patterns. Currently, some QA systems handle definition (e.g. *"What is anorexia nervosa?"*) and author (e.g. *"Who wrote the declaration of Independence?"*) questions by using patterns. But the pattern recognition mechanisms implemented are more primitive than the ones implemented in advanced IE systems. Besides these types of questions, many other questions tan be identified through patterns as demonstrated by [Soubbotin 2001].

While information extraction provides the syntactic framework for pattern identification (i.e. all the forms in which a SVO patterns can be expressed) the semantic constraints for the patterns are provided by question processing. For example, for the question:

"What is a petabyte?"

The QP module constructs the semantic constraints for the SVO pattern as follows:

`SUBJECT:` *"petabyte"*
`VERB:` any *"be"* verb
`OBJECT:` anything

The answer extraction module identifies all the matches for this pattern, and extracts the answer as the object field of the SVO pattern. Note that the advanced pattern matching techniques implemented in state-of-art IE systems allow for broad coverage of possible syntactic forms of any given pattern. The template in the above example matches not only SVO patterns in active form, e.g. *" a petabyte is ... "*, but also relative forms, e.g. *" a petabyte, which is ... "*, or gerund verb forms *"... the petabyte being ... "*.

This approach allows the transfer of the pattern matching technology developed in IE systems for more than 10 years, to open-domain QA systems, which until recently have relied only on surface-text techniques for answer detection.

6.3 QA on the Web

Until recently, QA systems have focused on extracting answers from locally-indexed collections of documents. In many real world applications, the Web

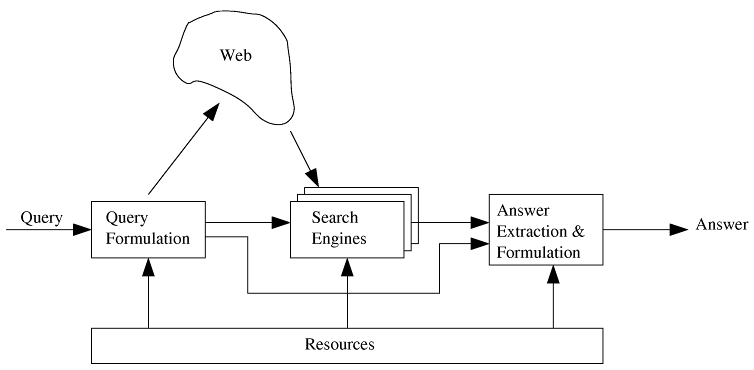

Fig. 7. Web-based Question Answering System

is a valuable source of information. With millions of anonymous contributors continuously adding new content, the Web has been growing into a huge, unstructured and diverse information resource, covering virtually every topic of interest. Search engines now offer access to over two billion Web documents [Sullivan 2000], but most of this information remains inaccessible to users, as the search engines are often incapable of pinpointing the specific information desired. The recently emerged web-based QA systems provide a natural language wrapper to search engines that: (a) provide a more intuitive interface for question formulation, and (b) extract and report natural language answers instead of a set of document URLs.

The architecture of a generic web-based QA system is shown in Figure 7. The system architecture is similar to a regular QA system, except that the static collection of documents is replaced with one or more search engines. Nevertheless, additional features have to be added to each of the three QA system modules. First, the query formulation module must understand the search engine syntax when formulating a query. For example, some search engines accept advanced boolean operators such as `OR` and `NEAR`, some do not. Second, the document retrieval module must handle the network latency when retrieving documents, and must also consider the fact that some indexed documents might not be available at all. And third, web documents are stored in various formats which must all be translated into a canonical form before being passed to the answer extraction module.

Besides these problems, the natural language wrapper around search engines offers some attractive advantages. Unlike meta-search engines, for web-based QA systems it is relatively painless to integrate multiple search engines under a single umbrella. The answer extraction module provides an elegant framework for ranking answers independently of their originating document, which means that multiple search engines can be queried in parallel and their documents merged before answer extraction and formulation.

Despite their young age and all the difficulties mentioned above, web-based QA system have approached commercial maturity [LCC].

References

[Brill 2001] E. Brill, J. Lin, M. Bnako, S. Dumais, A. Ng. Data-Intensive Question Answering. *Proceedings of the TREC-9*, 2000.

[Clarke 2000] C. L. A. Clarke, G. V. Cormack, D. I. E. Kisman, T. R. Lynam. Question Answering by Passage Selection (MultiText Experiments for TREC-9). *Proceedings of the TREC-9*, 2000.

[Harabagiu et al. 2000] S. Harabagiu, M. Pasca, and S. Maiorano. Experiments with open-domain textual question answering. *Proceedings of the 18th International Conference on Computational Linguists (COLING-2000)*, 2000.

[Harabagiu 2000] S. Harabagiu, D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Girju, V. Rus, P. Morarescu. FALCON: Boosting Knowledge for Answer Engines. *Proceedings of the TREC-9*, 2000.

[Harabagiu et al. 2001] S. Harabagiu, D. Moldovan, M. Pasca, M. Surdeanu, R. Mihalcea, R. Girju, V. Rus, F. Lacatusu, P. Morarescu, and R. Bunescu. Answering complex, list, and context questions with LCC's question answering server. *Proceedings of the 10th Text REtrievai Conference (TREC-2001)*, 2001.

[Harabagiu and Surdeanu 2002] S. Harabagiu and M. Surdeanu. Infrastructure for Open-Domain Information Extraction. *Proceedings of Human Language Technology 2002 (HLT 2002)*, 2002.

[Hobbs et al. 1996] J. Hobbs, D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel and M. Tyson. FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. Finite State Devices for Natural Language Processing, 1996.

[Hovy 2000] E. Hovy, U. Hermjakob, C-Y Lin, M. Junk, L. Gerber. The Webclopedia. *Proceedings of the TREC-9*, 2000.

[Ittycheriah 2001] A. Ittycheriah, M. Franz, S. Roukos. IBM's Statistical Question Answering System. *Proceedings of the TREC-2001.*

[Kwok 2000] K. L. Kwok, L. Grunfeld, N. Dinsti, and M. Chan. TREC-9 Cross Language, Web and Question Answering Track Experiments using PIRCS. *Proceedings of the TREC-9*, 2000.

[LCC] Language Computer Corporation web site: *languagecomputer. com*

[Mihalcea 2001] R. Mihalcea, D. Moldovan. Semantic Indexing using WordNet Senses. *Proceedings of ACL-2001 Workshop on Recent Advances in Natural Language Processing and Information Retrieval*, Hong Kong, 2000.

[Moldovan 1999] D. Moldovan, S. Harabagiu, M. Pasca, R. Mihalcea, R. Goodrum, R. Girju, V. Rus. LASSO: A Tool for Surfing the Answering Net. *Proceedings of the Eight Text REtrieval Conference (TREC 8)*, National Institute of Standards and Technology, 1999.

[Moldovan 2001] D. Moldovan, V. Rus. Logic Form Transformation of WordNet and its Applicability to Question Answering. Proceedings of the ACL, 2001.

[Moldovan 2002] D. Moldovan, M. Pasca, S. Harabagiu, M. Surdeanu. Performance Issues and Analysis in an Open-Domain Question Answering System. *Proceedings of the ACL*, 2002. Error

[Pasca and Harabagiu 2001] M. Pasca and S. Harabagiu. The informative role of WordNet in open-domain question answering. *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computationai Linguistics (NAACL-01)*, 2001.

[Soubbotin 2001] M. M. Soubbotin, S. M. Soubbotin. Patterns of Potential Answer Expressions as Clues to the Right Answers. *Proceedings of the TREC-2001.*

[Srihari 1999] R. Srihari and W. Li. Information Extraction Supported Question Answering. *Proceedings of the TREC-8*, 1999

[Sullivan 2000] D. Sullivan. Search engine sizes. searchenginewatch.com, November 2000

[Voorhees 1999] E. Voorhees The TREC-8 Question Answering track report. *Proceedings of the 8th Text REtrieval Conference (TREC-8)*, 1999.

[Voorhees 2001] E. M. Voorhees. Overview of the TREC 2001 Question Answering Track. Proceedings of the TREC-2001.

[Woods 1997] W. A. Woods. Conceptual Indexing: A Better Way to Organize Knowledge. Technical Report SMLI TR-97-61, Sun Microsystems Labs, Mountain View, CA, 1997.

[Woods 2000] W.A. Woods, S. Green, P. Martin, and A. Houston. Halfway to Question Answering. Proceedings of the TREC-9, 2000.