ExtrAns: Extracting Answers from Technical Texts

Diego Mollá and Rolf Schwitter, *Macquarie University*

Fabio Rinaldi, James Dowdall, and Michael Hess, *University of Zurich*

The ExtrAns answer-extraction system uses logical forms and lexical relations for semantic representation, to delve into and leverage the meaning of sentences, phrases, and words.

It's Friday at 12:20 p.m. in Nagoya, Japan, and Akira Watanabe, pilot of an Airbus passenger jet, is completing the final safety checks before requesting clearance, closing the doors, and taking off for Sapporo. However, there's a slight problem: Although the first officer's electronic centralized aircraft monitor (ECAM) is working fine, Watanabe's is completely blank. Clearly, the Airbus has an electrical problem. Watanabe radios the control tower and maintenance engineers are dispatched. They must diagnose the problem quickly, as the passengers are onboard and getting restless.

The maintenance team's ultimate knowledge source is a 15,000-page maintenance manual. Using a keyword-based information retrieval (IR) system, the team manually finds related documents to pinpoint the exact answer. However, all this takes time—and it might be just as fast in this case to open the avionics compartment and check all 24 frames of the plane's electrical system.

System users face such time and resource dilemmas daily, the world over. (We based this scenario on an actual case, changing the pilot's name and the flight details.) Fortunately, an emerging technology—*answer extraction*—uses sophisticated language technology to analyze documents and user questions and automatically extract answers accordingly. ExtrAns, an answer extraction system for technical domains, uses robust natural language processing and a semantic representation of the information's propositional content. This lets the system understand a domain's technical terminology and the relation between terms, which is vital for the answer extraction task. We've completed an ExtrAns prototype and applied it in two domains.

Finding answers: Existing options

Over the past few years, the annual Text Retrieval Conference has particularly promoted question-answering and answer extraction research through its question-answering track.[1] In TREC-QA, researchers test their systems' ability to retrieve answers to predefined questions from a sizeable document collection.

Most participating systems combine traditional IR and information extraction techniques with more sophisticated techniques, ranging from systematic pattern definitions to parser- and inference-system integration. So, a typical system uses IR techniques to preselect the documents or document fragments most likely to contain the answer. It then uses information extraction techniques, such as named-entity extraction, to select text fragments that contain strings that have a semantic type compatible with that of the expected answer. Finally, the system applies more sophisticated techniques to locate the smallest strings containing the answer and then ranks them. Many systems use the World Wide Web and lexical resources such as WordNet[2] to improve the question answering process.

Importantly, TREC-QA's competitions have shown that traditional IR techniques answer questions inefficiently. For example, when restricted to an answer with a relatively small amount of text (50 bytes), systems that relied only on IR techniques fared significantly worse than those that used some kind of language processing. Such systems account for both query and document meaning by performing syntactic and semantic analysis.

The competitions focus on *open-domain* systems—that is, systems that can (potentially) answer any generic question, from "When was Elvis Presley born?" to "What is the atomic number of the element lithium?" Given the competition's broad-based definition, participating systems must use effective, but generic, techniques, so the systems cannot properly deal with problems related to domain-specific terminology.

Additionally, TREC-QA competitions are based on relatively large text collections; TREC 9 (2000), for example, used 979,000 documents totaling 3,033 Mbytes of data.[1] Given this, the competing systems can't afford to perform resource-consuming tasks, so they must use relatively shallow text analysis. Few systems have attempted more than skimming the text's surface. A notable exception is Falcon,[3] TREC-QA 2000's best performer. It completely analyzed the pre-selected text and each query and created a logical query representation. It also used an abductive backchaining mechanism, which can provide a logical proof to justify an answer.

TREC-QA is an important catalyst in open-domain answer extraction development, however not all methods developed there are suitable for technical domains, such as that of our Airbus example. Systems working in technical domains can achieve more accurate results by exploiting a text's formatting and style conventions and domain-dependent terminology. Domain-based systems also typically work with smaller text collections, allowing more in-depth text analysis. For example, the SGML-based Airbus A320 maintenance manual is only 120 Mbytes and much smaller than TREC-QA's general corpus. So, for users such as our Airbus engineers, a more targeted answer extraction system is more effective.

ExtrAns overview

ExtrAns processes document collections offline and processes users' questions online. To obtain the semantic representations of the text sentences and the queries, ExtrAns applies the same chain of linguistic analysis modules in both the off- and online stages.

ExtrAns components

Figure 1 shows how ExtrAns' different modules interact. The syntactic analyzer and the semantic interpreter are the two central linguistic modules. Link Grammar (www.link.cs.cmu.edu/link), a freely available parser with a wide-coverage English grammar, returns the sentence's syntactic structure as a set of syntactic dependencies between the words. After traversing other linguistic modules for lemmatization, disambiguation, and anaphora resolution, ExtrAns sends the syntactic structure to a semantic interpreter that generates the logical forms.

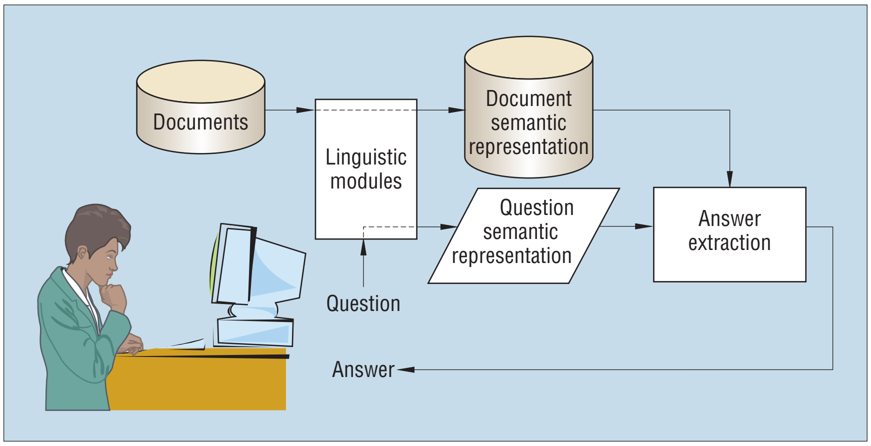

Documents
Linguistic modules
Document semantic representation
Question semantic representation
Answer extraction
Question
Answer

Figure 1. Architecture of the ExtrAns system. The syntactic analyzer and semantic interpreter modules process the documents' linguistic information (offline) and the questions (online).

When a user enters a query, ExtrAns converts the question into a logical form, then uses this form to retrieve a matching answer in the document knowledge base. Attaching pointers to the original text in the retrieved logical forms lets the system identify and highlight words most relevant to that particular answer.[4] Figures 2a and 2b show examples of ExtrAns output. When the user clicks one of the answers, ExtrAns displays the corresponding document with the corresponding answer highlighted in context.

Given possible ambiguities, a document sentence or user question can produce several logical forms. ExtrAns stores all these forms in the document knowledge base and uses all logical forms of the user's question as logical disjunctions to extract the answer. If several different interpretations of the same sentence contain the answer, the system might highlight different words in each interpretation. When this happens, ExtrAns combines all highlightings, using a graded scheme that indicates which words are used most often as answers.[4] As the screen shots in Figure 2 show, this makes ambiguities less obtrusive to users.

Applications

We built our first ExtrAns application to answer arbitrary user questions from the online Unix documentation files, or *man pages*. The system covers more than 500 unedited pages and answers questions such as "Which command copies files?" (The system is available for online testing at www.cl.unizh.ch/extrans.)

More recently, we used ExtrAns to cover the Airbus A320's *Aircraft Maintenance Manual* (*AMM*; see www.cl.unizh.ch/webextrans). The domain's highly technical nature, its use of an SGML-based format, and the manual's size—120 Mbytes compared with the 270-Kbyte Unix documentation in the first application—provide an important testbed for our system's scalability and domain independence. Assuming the manual has the answer, the system can quickly and efficiently answer questions such as "Where is the ECAM electrical contactor located?"

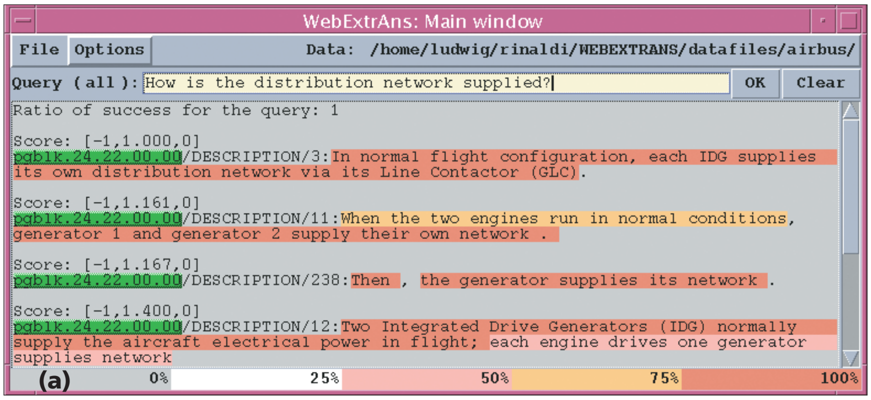
Content analysis in ExtrAns

To leverage a text's meaning and thereby extract the best answers, ExtrAns must find the most appropriate way to express a sentence's semantic and logical contents. To accomplish this, we paid special attention to the implementation of simple inferences, the development of a flat notation for logical forms, and the treatment of terminology.

Inferences

Extracting answers sometimes requires that a system make inferences. To avoid unnecessary online computational overhead, ExtrAns makes all inferences that do not require user queries offline and stores the inference results as additional information in the semantic representation. This results in a larger data store, but we found that it was more efficient to use database systems to store and access the information than to try to speed up the inference mechanism.

Currently, ExtrAns makes only a limited set of inferences. One such inference is *dis-*



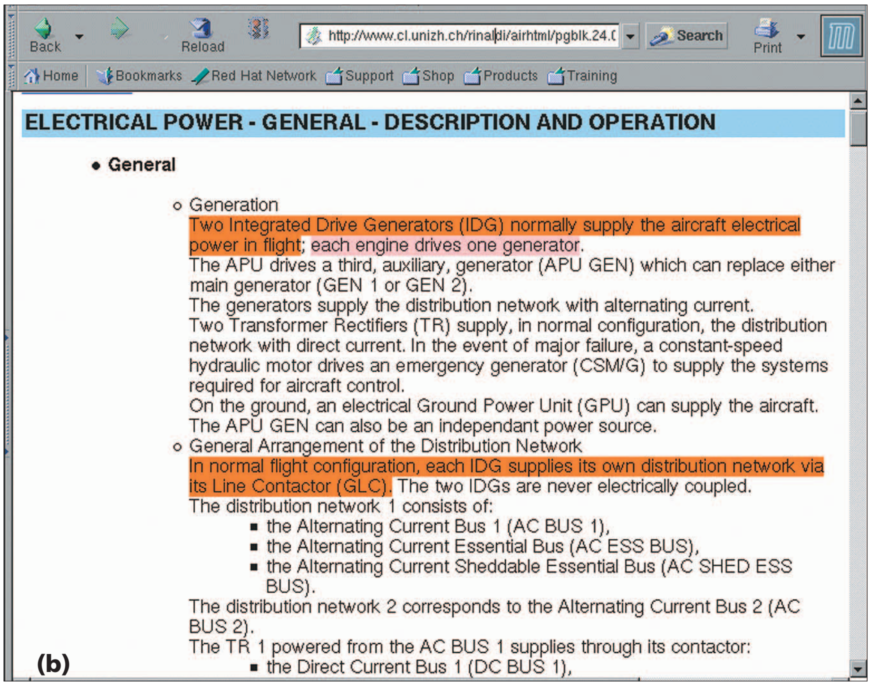
```
Data:  /home/ludwig/rinaldi/WEBEXTRANS/datafiles/airbus/
Query (all): How is the distribution network supplied?
Ratio of success for the query: 1

Score: [-1,1.000,0]
pgblk.24.22.00.00/DESCRIPTION/3:In normal flight configuration, each IDG supplies
its own distribution network via its Line Contactor (GLC).

Score: [-1,1.161,0]
pgblk.24.22.00.00/DESCRIPTION/11:When the two engines run in normal conditions,
generator 1 and generator 2 supply their own network .

Score: [-1,1.167,0]
pgblk.24.22.00.00/DESCRIPTION/238:Then , the generator supplies its network .

Score: [-1,1.400,0]
pgblk.24.22.00.00/DESCRIPTION/12:Two Integrated Drive Generators (IDG) normally
supply the aircraft electrical power in flight; each engine drives one generator
supplies network
```



ELECTRICAL POWER - GENERAL - DESCRIPTION AND OPERATION

- General
  - Generation
    Two Integrated Drive Generators (IDG) normally supply the aircraft electrical power in flight; each engine drives one generator.
    The APU drives a third, auxiliary, generator (APU GEN) which can replace either main generator (GEN 1 or GEN 2).
    The generators supply the distribution network with alternating current.
    Two Transformer Rectifiers (TR) supply, in normal configuration, the distribution network with direct current. In the event of major failure, a constant-speed hydraulic motor drives an emergency generator (CSM/G) to supply the systems required for aircraft control.
    On the ground, an electrical Ground Power Unit (GPU) can supply the aircraft.
    The APU GEN can also be an independant power source.
  - General Arrangement of the Distribution Network
    In normal flight configuration, each IDG supplies its own distribution network via its Line Contactor (GLC). The two IDGs are never electrically coupled.
    The distribution network 1 consists of:
    - the Alternating Current Bus 1 (AC BUS 1),
    - the Alternating Current Essential Bus (AC ESS BUS),
    - the Alternating Current Sheddable Essential Bus (AC SHED ESS BUS).

    The distribution network 2 corresponds to the Alternating Current Bus 2 (AC BUS 2).
    The TR 1 powered from the AC BUS 1 supplies through its contactor:
    - the Direct Current Bus 1 (DC BUS 1),

Figure 2. Example ExtrAns output: (a) Results from a user query. Answers are highlighted in colors according to pertinence. (b) When the user clicks on an answer (green highlighting) in (a), ExtrAns opens the document window.

tributivity of conjunction. For example, if the sentence says "The static inverter is activated and a beep sounds," ExtrAns' inference engine adds two additional pieces of information to the logical form, corresponding to the propositional contents of "the static inverter is activated" and "a beep sounds."

Another particularly useful type of inference is *synonymy*. Based on WordNet, we created a small thesaurus and defined synonym identifiers for all of the thesaurus's synonym sets (*synsets*). The inference engine replaces all thesaurus-defined words in the final, logical form of the query sentence with their synset identifiers. To simplify the system, we handle ambiguity in word sense trivially: If a word belongs to two or more synsets, the algorithm randomly chooses one synset. Because the words within a technical domain have limited ambiguity, word sense errors have minimal impact.

If ExtrAns can't find a direct answer to the user's question, it can relax the logical restrictions in a stepwise manner. ExtrAns first considers *hyponyms*—words that belong to semantic classes that are subsets of the original words. ExtrAns then adds the hyponyms' synsets as disjunctions in the question's logical form. Technically speaking, the resulting logical form is equivalent to the original logical form. In practice, the answer extraction module's default is to not explore hyponymy relations, so the hyponymy stage might find additional answers to the question.

In the synonym and hyponym stages, ExtrAns finds only logically correct proofs (within the semantic representation's expressive limits), making it a highly precise answer extraction system. If the hyponym stage fails, however, ExtrAns goes on to attempt *approximate matching* and returns the sentences with the highest semantic content overlap. If this method fails, the system attempts *keyword matching*, in which it abandons syntactic criteria and uses only information about word classes. This final step resembles a traditional passage retrieval that is enhanced with parts-of-speech tags.

Flat notation

Our focus on ExtrAns' logical content analysis of technical text led us to develop a methodology for creating and using logical forms. Our goal was to create a formalism that's easy to build and use, yet expressive enough for the task at hand. We also wanted a formalism that could cope with problematic sentences, including ones that were long, had spelling mistakes, or were structurally unrecognizable to the syntactic analyzer. To meet our goals, ExtrAns' logical forms use a flat notation and reification (the "Creating the Flat Logical Forms" sidebar offers further details). Take, for example, two simple sentences. First, "The ECAM contactor is located in the left frame," which is logically expressed as

```
holds(e4),
object('ecam_contactor', o2, [x2]),
evt(locate, e4, [x4, x2]),
object(anonymous_object, o4, [x4]),
object(frame, o7, [x7]),
prop(left, p2, [x7]),
prop(in, p5, [e4, x7]).
```

Next, "The static inverter is activated if the CSM/G is unavailable," which is logically expressed as

```
if(e5, p10),
prop(static, p2, [x3]),
object(inverter, o3, [x3]),
evt(activate, e5, [x5, x3]),
object(anonymous_object, o5, [x5]),
object('csm/g', o8, [x8]),
prop(unavailable, p10, [x8]).
```

The first sentence's logical form says that,

Creating the Flat Logical Forms

Typically, logical forms contain embedded expressions. We decided to use a flat notation to facilitate the derivation of partial logical forms from complex (or even ungrammatical) sentences and to enable their quick processing to find the answer to the user question. ExtrAns' semantic interpreter uses reification to build the flat logical forms.

Flattening nested expressions

Using additional arguments to flatten out a nested structure is a well-known approach. The technique is commonly used in programming when a particular programming language lacks user-defined functions that return values. In such cases, we would resort to subroutines that return the values through extra arguments. For example, instruction 1 can be substituted by sequence 2:

```
1. A := (factorial(25) – exp(12)) * 2;
2. factorial(X, 25);
   exp(Y, 12);
   A := (X – Y) * 2;
```

We can use additional arguments, X and Y, to store the results of factorial and exp, respectively. Similarly, we can flatten out the logical expression, "John ate an apple quickly," by introducing new arguments. So, we can flatten out 1 as 2:

1. $\exists a(\mathrm{quick}(\mathrm{eat}(j, a)) \wedge \mathrm{apple}(a))$
2. $\exists a, e_1(\mathrm{eat}(e_1, j, a) \wedge \mathrm{quick}(e_1) \wedge \mathrm{apple}(a))$

We use the new logical variable e_1 as a handle to refer to the modifiable event so that we can say that the eating event is quick ($\mathrm{quick}(e_1)$).

Reification

Reification is the technical term for introducing new entities that refer to abstract concepts. A classical example of reification is Davidson's introduction of entities representing events.[1] Nothing stops us from reifying every possible element in a logical form, giving way to Hobb's *ontological promiscuity*.[2] For answer extraction, we've found it more practical to reify only objects, events, and properties:

- *Objects*. A noun such as *contactor* introduces the predicate object(contactor, o1, [x1]), meaning "o1 is the concept that the object x1 is a contactor." We can use the new entity o1 in constructions with adjectives intentionally modifying nouns or in expressions of identity.
- *Events*. A verb such as *installs* introduces evt(install, e1, [x1, x2]), meaning "e1 is the concept that x1 installs x2." x1 and x2 represent the objects introduced by the verb arguments. We use event reification to modify events through adverbial and prepositional phrases.
- *Properties*. Adjectives and adverbs introduce properties. For example, an adjective such as *blue* introduces the predicate prop(blue, p1, [x1]), meaning "p1 is the concept that x1 is blue." Property reification is useful when we want to modify an adjective, as in the sentence "The house is pale blue."

References

1. D. Davidson, "The Logical Form of Action Sentences," *The Logic of Decision and Action*, N. Rescher, ed., Univ. Pittsburgh Press, 1967, pp. 81–120.
2. J.R. Hobbs, "Ontological Promiscuity," *Proc. 1985 Ann. Meeting Assoc. Computational Linguistics* (ACL), Univ. of Chicago ACL, 1985, pp. 61–69.

in the *AMM's* universe, there is an ECAM contactor (x2) and a frame (x7), a property (p2) that the frame (x7) is on the left, and the event (e4) of something locating x2 in x7. Because the sentence does not say who or what locates the contactor, ExtrAns' semantic interpreter uses a dummy anonymous object. The second sentence's logical form expresses the conditional existence of the event of something activating the static inverter (e5) if the property of the CSM/G (x8) is unavailable (p10).

We kept these examples deliberately simple to illustrate the notation; any given documentation page surely contains more complex sentences, possibly with multiple interpretations. Figure 3 shows an example of a more complex sentence and one of its interpretations. ExtrAns tried to classify all words in the text fragments that were too difficult for the parsing system as nouns, verbs, adjectives, or adverbs (marked in boldface in Figure 3). The words tagged as keywords resisted ExtrAns' classification attempts.

holds(a1), prop(in, p1, [a1, x4]),
object(configuration, a2, [x4, x3, a5]), evt(configure, x4, [x3, a5]),
object(flight, a3, [x3]), explication(x4, x6), object(each, a4, [x6]),
prop(normal, p2, [x4]), **evt(supply, e8, _)**, **object(idg, a6, [x7])**,
object(network, a7, [x12]), compound_noun(x11, x12),
object(distribution, a8, [x11, a9, a10]),
evt(distribute, x11, [a9, a10]), prop(via, p12, [x12, x16]),
object('Contactor', a11, [x6]), explication(x16, x18),
object(glc, a12, [x18]), **prop(own, p10, _)**, **keyw('Line')**.

Figure 3. Example of a flat logical form from the *Aircraft Maintenance Manual* sentence "In normal flight configuration, each IDG supplies its own distribution network via its Line Contactor (GLC)." All unknown words are marked in boldface.

ExtrAns' flat logical forms do not attempt to express a sentence's complete semantic information. For example, they discard information carried by modal verbs, tense and aspect, plural forms, and quantifiers. This is why we call them *minimal logical forms*. For ExtrAns, the only truly important information is the set of semantic relations among open words (nouns, verbs, adjectives, and adverbs) and prepositions.

Using minimal logical forms facilitates

finding answers to questions because it uses the question's logical form to generate the partial information that the answer requires. For example, the question "Where is the ECAM contactor located?" produces:

```
object('ecam_contactor', O1, Y), evt(locate, E2, [X, Y])
```

In our example, the answer's logical form does not have the holds predicate and does not express the anonymous object. ExtrAns uses unification to determine the overlap between two logical forms. Sentences containing terms that overlap are good candidates for the answer if they have compatible variables. So, the sentence "The ECAM contactor is located in the left frame" is a possible answer if the question's *O1* variable (all variables are represented in uppercase) corresponds to the answer's *o2* constant; the question's X variable corresponds to the answer's x4 constant; and so on. In other words, if we compare ExtrAns with a "bag of words" approach (such as those used in IR systems), the "words" correspond to semantic primitives and include links to other "words." We can thus view the ExtrAns answer procedure as a "bag of semantic relations" in which we account for the semantic relations between the sentence words.

As another example, given the question, "What activates the static inverter?" we might consider the sentence "The static inverter is activated if the CSM/G is unavailable" as a possible answer. However, this sentence is not a logical answer to the question because the clause "The static inverter is activated" is the consequent of a conditional. Although ExtrAns has no way of knowing whether the CSM/G is available or not, the user might know. So, ExtrAns sends the sentence to the user, letting him or her quickly determine if the sentence answers the question.

This mechanism can easily return acceptable answers to *wh-questions* (such as "What's located in the left frame?"), *yes–no-questions* (such as "Is the ECAM contactor located in the left frame?"), and some *how-questions* (such as "How is the static inverter activated?"). We are exploring mechanisms to successfully answer other question types; our focus is on integrating information extraction techniques, such as those used in systems competing in TREC-QA.

ExtrAns' semantic interpreter can produce flat logical forms with minimal domain knowledge and the administrator can easily port the system to different domains. The domain's only true impact is during the preprocessing stage of the input text and during the creation of the lexical ontology that reflects the domain's specific terms, meanings, and lexical relations.

Terminology

To produce a syntactic representation of each sentence, the system must identify terms, such as "electrical centralized aircraft monitor" or "electrical connector," as phrasal units. In the term "avionics compartment," for example, only "compartment" interacts with other sentence words, and the parser should effectively ignore "avionics." Otherwise, the system could plausibly combine "avionics" with the wrong sentence words—producing multiple possibilities without choosing the correct parse—or waste effort assigning a structure to the term itself.

The internal syntactic structure of technical terms is notoriously idiosyncratic and resists standard parsing techniques. In our Airbus domain, a technician might ask, "Where is the electronic centralized aircraft monitor?" If standardization were possible, the answer would use the same term. In reality, the answer might describe the location of the "centralized aircraft monitor," the "electronic centralized aircraft monitor," or even the "ECAM." So, essentially, a term's internal structure is indeterminable, although the term as a whole interacts in the sentence the same as an ordinary word.

To process answers efficiently, the system must preprocess the document collection to determine which terms appear in the domain and how these terms relate to each other. Unfortunately, although terminology extraction methods have matured, the process still resists full automation.[5] In ExtrAns we created an extended list of term candidates using statistical measures and methods to filter out uninteresting words.

To determine which form of a term the answer might use, we organized the extracted term list into a WordNet-like hierarchy of subtypes. This allows ExtrAns to determine which terms are specific types of other terms and how a domain's individual concepts are gathered into synsets.

Because technical terminology typically constructs more specific terms by adding words to generic terms, ExtrAns can automatically determine subtype relations across the term using a simple algorithm. This can easily determine, for example, that the "electronic centralized aircraft monitor" is a subtype of "aircraft monitor" and that "First Officer seat" is a subtype of "seat."

Finding synonymous terms requires more complex processing. To this end, we use the FASTR[6] terminology extraction tool during the document processing stage.[5] Using FASTR, we can gather the synonymous terms into synsets then place them in a hierarchy according to subtype relation. ExtrAns exploits this terminological thesaurus in subsequent processing stages.

Using rewrite rules combined with a morphological database and the hierarchy of subtypes lets ExtrAns identify linguistic variations between two terms. It can detect

- simple head inversion ("generator control unit" is a variant of "control unit for the generator")
- morphological variations ("electric contactor" is a variant of "electrical contactor")
- complex morphosyntactic variations ("electrical generation equipment" is a variant of "equipment for generating electricity")

By exploiting the synsets, ExtrAns can also detect weaker synonymy relations:

- synonymous heads ("electrical fault" and "electrical defect")
- synonymous modifiers ("upright position" and "vertical position")
- a combination of heads and modifiers ("functional test" and "operational check")

Once the candidate terms have been produced automatically, a human domain expert determines which terms on the list are actually relevant. This is time well spent: A concise and accurate domain terminology is vital if we want the system to achieve any level of content understanding. Once we've identified terms, we can improve efficiency by giving multiword terms the same status as ordinary words during syntactic analysis. In our experience, this improves parser efficiency by a significant amount (46 percent in our experiments).

Performance

To evaluate ExtrAns results, we compared it with Smart, a traditional IR system.[7] As a measure, we used TREC-QA's mean reciprocal rank,[1] which ranks results according to the placement of the first correct answer in the system's output list. A system's MRR is the mean of the rank reciprocals for all answers in a given set. Our evaluation domain was the *AMM*. We constructed 100 questions on the basis of a set of known answers in the document collection.

Although Smart found more answers than ExtrAns, most of the answers ExtrAns did find were in first position. Furthermore, in some cases ExtrAns found more than one valid answer for the same question (sometimes in the same document). ExtrAns' overall MRR was 0.63; Smart's was 0.46. As expected, ExtrAns provided far higher precision than the generic IR system, at the price of smaller recall.[8]

Answer extraction technology is now taking off. Our work on ExtrAns shows the advantages of focusing on a specific technical domain and using relatively small data sets to implement a highly precise answer extraction system. Several decisive factors contribute to ExtrAns' success:

- It explicitly handles domain-specific terminology.
- It integrates a full parser and a robust semantic interpreter that can handle complex, ungrammatical sentences.
- It uses a flat logical notation that encodes the minimal semantic information required for the current task.
- It integrates display techniques that help the user find the answer in context.

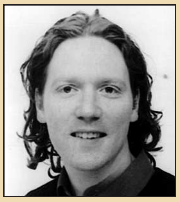
ExtrAns and other answer extraction systems work for many applications that require quick and precise answers from technical texts, including online help systems for software documentation, support systems for call centers in large organizations, Internet-based public-inquiry systems, and technical support systems. Because users can install ExtrAns on a laptop or tablet PC, they can use it onsite—and so our maintenance engineer could quickly locate and fix that faulty electrical contactor, letting the Airbus flight to Sapporo take off without significant delay. ■

The Authors

Diego Mollá is a lecturer in the Centre for Language Technology at Macquarie University in Sydney, Australia. His research focuses on bridging the gap between theoretical linguistics, especially semantics and logical forms, and practical natural language processing applications. His projects center around AnswerFinder, a question-answering system. He received an MSc in speech and language processing and PhD in the formal semantics of aspectual composition from the University of Edinburgh. He is currently secretary of the Australasian Language Technology Association. Contact him at the Division of Information and Communication Sciences, Macquarie Univ., New South Wales 2109, Australia; diego@ics.mq.edu.au; www.ics.mq.edu.au/~diego.

Fabio Rinaldi is a researcher at the University of Zurich's Institute of Computational Linguistics. His research interests include ontologies, information extraction, answer extraction, and terminology. He received an MSc in computer science from the University of Udine, Italy, and worked in various European centers (including ITC/IRST in Trento, Italy, and UMIST in Manchester, UK). He is a member of the Association for Computational Linguistics. Contact him at the Inst. of Computational Linguistics, Univ. of Zurich, Winterthurerstr. 190, CH-8057 Zurich, Switzerland; rinaldi@cl.unizh.ch; www.cl.unizh.ch/rinaldi.

Rolf Schwitter is a lecturer in the Centre for Language Technology at Macquarie University in Sydney, Australia. His research interests include answer extraction, controlled natural languages, and logic programming. He received a PhD in computational linguistics from the University of Zurich. Contact him at Macquarie Univ., Dept. of Computing, Ctr. for Language Technology, NSW, 2109 Australia; schwitt@ics.mq.edu.au; www.ics.mq.edu.au/~rolfs.



James Dowdall is a doctoral student and research assistant at the University of Zurich's Institute of Computational Linguistics. His research interests include terminology extraction, automated thesaurus construction, and efficient strategies for exploiting multiword terminology in natural language processing. He has an MPhil in theoretical linguistics from Trinity College, Dublin. Contact him at the Inst. of Computational Linguistics, Univ. of Zurich, Winterthurerstr. 190, CH-8057 Zurich, Switzerland; dowdall@cl.unizh.ch; www.cl.unizh.ch/dowdall.

Michael Hess is a professor of computational linguistics at the University of Zurich. His research interests include text-based intelligent systems (in particular, answer extraction and question answering), computational semantics, text mining, and using linguistic methods in Web-based teaching. He received a PhD in Russian linguistics and modern history from the University of Zurich. Contact him at the Inst. of Computational Linguistics, Univ. of Zurich, Winterthurerstr. 190, CH-8057 Zurich, Switzerland; hess@cl.unizh.ch; www.cl.unizh.ch/hess.

References

1. E.M. Voorhees, "The TREC Question Answering Track," *Natural Language Eng.*, vol. 7, no. 4, 2001, pp. 361–378.
2. C. Fellbaum, ed., *WordNet: An Electronic Lexical Database*, MIT Press, 1998.
3. S. Harabagiu et al., "Falcon: Boosting Knowledge for Answer Engines," *Proc. 9th Text Retrieval Conf. (TREC-9)*, Nat'l Inst. Science and Technology, 2000, pp. 479–488.
4. D. Mollá et al., "ExtrAns, An Answer Extraction System," *Traitement Automatique des Langues*, vol. 41, no. 2, 2000, pp. 495–522.
5. J. Dowdall et al., "Technical Terminology as a Critical Resource," *Proc. Int'l Conf. Language Resources and Evaluations* (LREC-2002), Euro. Language Resource Assoc., 2002, pp. 1897–1903.
6. C. Jacquemin, *Spotting and Discovering Terms through Natural Language Processing*, MIT Press, 2001.
7. G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, 1989.
8. F. Rinaldi et al., "Towards Answer Extraction: An Application to Technical Domains," *Proc. 15th Euro. Conf. Artificial Intelligence* (ECAI 2002), IOS Press, 2002, pp. 460–464.