A PROPOSAL FOR SLS EVALUATION

Sean Boisen
Lance Ramshaw
Damaris Ayuso
Madeleine Bates

BBN Systems and Technologies Corporation
Cambridge, MA 02138

ABSTRACT

This paper proposes an automatic, essentially domain-independent means of evaluating Spoken Language Systems (SLS) which combines software we have developed for that purpose (the "Comparator") and a set of specifications for answer expressions (the "Common Answer Specification", or CAS). The Comparator checks whether the answer provided by a SLS accords with a canonical answer, returning either true or false. The Common Answer Specification determines the syntax of answer expressions, the minimal content that must be included in them, the data to be included in and excluded from test corpora, and the procedures used by the Comparator. Though some details of the CAS are particular to individual domains, the Comparator software is domain-independent, as is the CAS approach.

1 INTRODUCTION

The DARPA community has recently moved forward in beginning to define methods for common evaluation of spoken language systems. We consider the existing consensus to include at least the following points:

- Common evaluation involves working on a common domain (or domains). A common corpus of development queries (in both spoken and transcribed form), and answers to those queries in some canonical format, are therefore required.
- One basis for system evaluation will be answers to queries from a common database, perhaps in addition to other measures.
- Automatic evaluation methods should be used whenever they are feasible.
- System output will be scored by NIST, though all sites will be able to use the evaluation program internally.
- Development and test corpora should be subdivided into several categories to support different kinds of evaluation (particularly concerning discourse phenomena).

An implicit assumption here is that we are considering database query systems, rather than any of the various other natural language processing domains (message understanding, command and control, etc.). Evaluating systems for these other domains will naturally require other evaluation procedures.

Building on the points of consensus listed above, this proposal presents an evaluation procedure for the DARPA Common Task which is essentially domain-independent. The key component is a program, designated the Comparator, for comparing canonical answers to the answers supplied by a Spoken Language System. A specification for such answers, which incorporates the requirements of the Comparator, is presented in Section 2. This specification, called the Common Answer Specification (CAS), *is not intended to be suitable for interactive systems*: rather, it is designed to facilitate automatic evaluation. While we have attempted to cover as broad a range of queries and phenomena as possible, data which fall outside the scope of the CAS can simply be left out of test corpora for now.

Section 3 presents some of the justification supporting the proposal in Section 2, as well as amplifying several points. Details on the Comparator are given in Section 4.

Section 5 concludes with a discussion of corpus development, looking at what kind of data should be collected and how corpora should be annotated to facilitate testing various types of natural language.

2 THE PROPOSAL

Here we present the basic substance of our proposal for a Common Answer Specification (CAS), deferring some details and elaboration to Section 3. The CAS, which is designed to support system evaluation using the Comparator, covers four basic areas:

1. The notation used for answer expressions.
2. The minimal content of canonical answers.
3. The material included in test corpora.
4. The procedure used by the Comparator for comparing canonical answers to system output.

We assume an evaluation architecture like that in Figure 1. Everything on the right hand side of the figure is the developer's responsibility[1]. Items on the left side of the diagram will be provided as part of the evaluation process.

The Common Answer Specification was devised to enable common, automatic evaluation: it is expressly *not* designed to meet the needs of human users, and should be considered *external*. This means that developers are free to implement any output format they find convenient for their own use: we only propose to dictate the form of what is supplied as input to the Comparator. It is assumed that some simple post-processing of system output by the developer will be required to conform to the CAS.

While the central points of the CAS are domain-independent, certain details relating to content can only be determined relative to a specific domain. These portions of the proposal are specified for the personnel domain and SLS Personnel Database (Boisen, 1989), and all examples are taken from that domain as well.

2.1 Notation

S–1 *The basic CAS expression will be a relation, that is, a set of tuples. The types of the elements of tuples will be one of the following: Boolean, number, string, or date. Relations consisting of a single tuple with only one element can alternatively be represented as a scalar.*

S–2 *CAS expressions will be expressed as a Lisp-style parenthesized list of non-empty lists. Scalar answers may alternatively be represented as atomic expressions. The two Boolean values are* `true` *and* `false`*. Numeric answers can be either integers or real numbers. Dates will be an 9-character string like* `"01-JAN-80"`*. The number and types of the elements of tuples must be the same across tuples. An empty relation will be represented by the empty list. Alphabetic case, and white-space between elements, will be disregarded, except in strings.*

A BNF specification for the syntax of the Common Answer Specification is found in Appendix A. Here are some examples of answers in the CAS format:

```
((false))
FALSE
2.9999999999
((   3   ))
"04-JUL-89"
((2341 "SMITH") (5573 "JONES"))
()
```

2.2 Minimal Content

Certain queries only require scalar answers, among them yes/no questions, imperatives like "Count" and "Sum", questions like "How much/many ____ ?", "How long ago ____ ?", and "When ____ ?". Other queries may require a relation for an answer: for these cases, the CAS must specify (in linguistic terms) which of the entities referred to in the English expression are required to be included in the answer. For required entities, it is also necessary to specify the database fields that identify them.

S–3 *For WH-questions, the required entity is the syntactic head of the WH noun phrase. For imperatives, the required NP is the head of the object NP. The nouns "list", "table", and "display" will not be considered "true" heads.*

> **Examples:** For the query "Which chief scientists in department 45 make more than $70000?", the required entity is the scientists, not the department or their salaries.
>
> In the case of "Give me every employee's phone number", the only required entity is the phone number.

[1] BBN has offered their ERL interpreter (Ramshaw, 1989) as a backend database interface for those who desire one: use of the ERL interpreter is explicitly *not* required for common evaluation, however, and developers are free to use whatever database interface they find suitable.

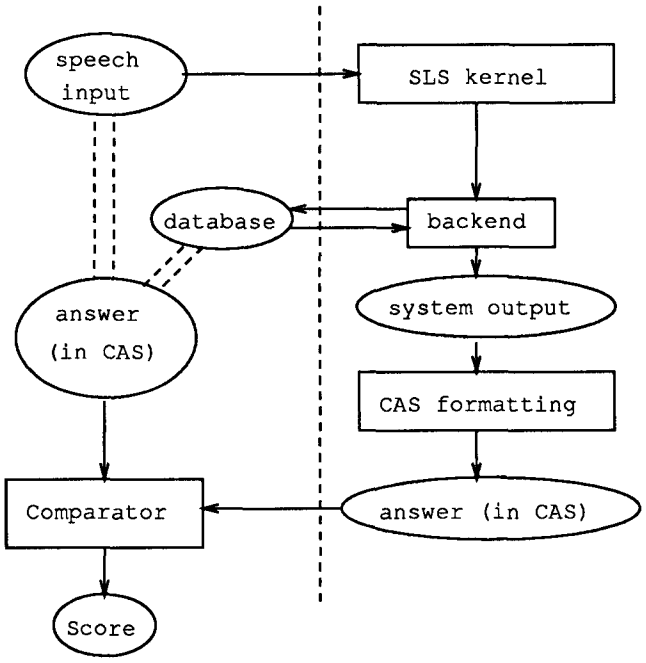

Figure 1: The evaluation process

For "Show me a list of all the departments", the required entity is the departments, not a list.

For the query "Count the people in department 43", only the scalar value is required.

Entities often can be identified by more than one database field: for example, a person could be represented by their employee identification number, their first and last names, their Social Security Number, etc. For any given domain, therefore, the fields that identify entities must be determined. If only one database identifier is available (i.e., only the field **`salary`** can represent a person's salary), the choice is clear: in other cases, it must be stipulated.

S–4 (Personnel) *In any evaluation domain, canonical database identifiers for entities in the domain must be determined. For the personnel domain, Table 1 will specify the database fields that are canonical identifiers for entities with multiple database representations.*

Example: For the query "List department 44 employees", the required database field is **`employee-id`**, so a suitable answer would be

```
((4322) (5267) ...)
```

where 4322, 5267, etc. are employee identification numbers.

Certain English expressions in the personnel domain are vague, in that they provide insufficient information to determine their interpretation. We therefore stipulate how such expressions are to be construed.

S–5 (Personnel) *In any evaluation domain, the interpretation of vague references must be determined. For the personnel domain, Table 2 will designate the referents for several vague nominal expressions.*

Example: For the query "What is Paul Tai's phone number?", the expression will be interpreted as a request for Tai's work phone number, not his home phone.

entity	database field
employees	`employee-id`
countries	`country-code`
degrees	`degree`, `employee-id`, `school-code`
departments	`department`
divisions	`division`
majors	`major-code`
schools	`school-code`
states	`state-code`

Table 1: Canonical field identifiers for domain entities with multiple database representations.

expression	database field
telephone	`work-phone`
name (of a person)	`last-name`
address	`street`

Table 2: Database fields for vague nominal expressions.

Another case involves the specificity of job titles. In the Common Personnel Database, there are several job titles that are related to a specific type of profession: for example, "STAFF SCIENTIST", "SCIENTIST", "CHIEF SCIENTIST", etc. We propose that all these be considered scientists in the generic sense for queries like "Is Mary Graham a scientist?" and "How many scientists are there in department 45?".

S–6 (Personnel) *Someone will be considered a scientist if and only if that person's* `job-title` *contains the string "SCIENTIST". The same is true for the following generic profession titles: engineer, programmer, clerk, and accountant. The terms manager and supervisor will be treated interchangeably, and someone will be considered a manager or supervisor only if that person is the supervisor of some other person.*

2.3 Test Corpora

S–7 *The primary corpus will be composed of simple pairs of queries and their answers. In addition, a distinct corpus will be used for evaluating the simple discourse capability of reference to an immediately preceding query. This corpus will contain triplets of* $\langle query_1, query_2, answer_2 \rangle$*, where* $query_1$ *contains the referent of some portion of* $query_2$*. The canonical answer will then be compared to* $answer_2$*.*

> **Example:** One entry in the discourse corpus might be the following triplet:
>
> $query_1$: "What is Mary Smith's job title?"
>
> $query_2$: "What is her salary?"
>
> $answer_2$: `((52000))`

S–8 *For the time being, we propose to exclude from the test corpora queries that:*

- *are ambiguous*
- *are overly vague (i.e., "Tell me about John Smith")*
- *don't have an answer because no data is available*
- *involve presupposition failure*
- *require sorting*
- *require "meta-information" (e.g., "List the ethnic groups you know about")*
- *or otherwise lack a generally accepted answer.*

2.4 The Comparator

S–9 *The Comparator will use an "epsilon" measure for comparing real numbers. The number in the canonical answer will be multiplied by the epsilon figure to determine the allowable deviation : numbers that differ by more than this amount will be scored as incorrect. The value of the epsilon figure will be initially set at 0.0001.*

> **Example:** If the canonical answer is 53200.0, the maximum deviation allowed will be (53200.0 × 0.0001), or approximately 5.32. Thus a system answer of 53198.8 would score as correct, but 53190.9 would be incorrect.

S–10 *Extra fields may be present in an answer relation, and will be ignored by the Comparator. The order of fields is also not specified: the mapping from fields in a system answer to fields in the canonical answer will be determined by the Comparator.*

> **Example:** For the query "Show Paul Tai's name and employee id", with the canonical answer
>
> ```
> ((4456 "TAI"))
> ```
>
> any of the following would be an acceptable answer:
>
> ```
> ((4456 "TAI" "PAUL")))
> (("TAI" 4456)))
> ```

S–11 *The only output of the Comparator will be "correct" or "incorrect". Capturing more subtle degrees of correctness will be accomplished by the quantity and variety of the test data.*

3 DISCUSSION

This section presents some of the justification supporting the proposal in Section 2, as well as amplifying several points and discussing some possible shortcomings and extensions. It may be usefully omitted by those who are not interested in these details. The organization follows the four areas of the proposal: notation, minimal content, test corpora, and the Comparator procedures.

3.1 Notation

The proposal in S–2 allows scalar values to be represented either as relations or as atomic expressions, i.e., either of

```
false
((false))
```

Treating the answer to questions like "Is Paul Tai a clerk?" as a relation seems somewhat unfortunate. This allows for a completely homogeneous representation of answers, however, and is permitted for this reason.

The range of types in S–1, while adequate for the personnel domain, may need to be enlarged for other domains. One obvious omission is a type for *amounts*: units are not specified for numbers. In the personnel domain, there are only two amounts, years and dollars, neither of which is likely to require expression in other units (though one could conceivably ask "How many days has Paul Tai worked for the company?"). Other domains would not be similarly restricted, and might require unit labels for amounts of length, speed, volume, etc. Of course, it is always possible to simply specify canonical units for the domain and require all numbers to be in those units: this would increase the effort required to conform to the CAS, however.

Answers to requests for percentages should express the ratio directly, not multiplying by 100: so if 45 out of 423 employees have PhD's, the answer to "What percentage of employees have PhD's?" should be

```
0.1064
```

3.2 Minimal Content

Under section S–3 of the proposal, one should note the possibility of a required NP being conjoined or modified by a "conjoining" PP modifier, as in the following examples:

> *List the clerks and their salaries.*
> *List the clerks with their salaries.*

Clearly in both of these cases the salaries as well as the employee IDs should be required in the answer.

One possible objection to the approach proposed in S–3 is that it ignores some well-documented concerns about the "informativeness" or pragmatic capabilities of question-answering systems. For example, in a normal context, a suitably informative answer for

> *List the salary of each employee.*

would provide not just a list of salaries but some identification of the employees as well, under the assumption that this is what the user *really* wants. Since the purpose of the CAS is automatic evaluation rather than user convenience, this objection seems irrelevant, at least until a metric for measuring pragmatic capabilities can be defined. Note that S–10 means developers are free to include extra fields for pragmatic purposes: such fields are simply not required for correctness. A similar point can be made concerning vague expressions (S–5). Only the `street` field is explicitly required for references to an address, since that should be sufficient to determine correctness, but developers may also include `city` and `state` if they wish.

One might argue that the proposed treatment of manager/supervisor in S–6 is inconsistent with the approach taken for scientists, engineers, programmers, etc. Our decision is essentially to consider manager and supervisor as technical terms which indicate a supervisory relation to another employee, rather than generic descriptions of a profession. This has the possibly unfortunate consequence that employees in the Common Personnel Database with job titles like "SUPERVISOR" and "PROJECT MANAGER" may not be considered supervisors or managers in this technical sense. Nevertheless, given that these titles seem less like descriptions of a profession, the approach is not inconsistent.

There are probably other vague expressions which will have to be dealt with on a case-by-case basis, either by agreeing on a referent or eliminating them from the corpora.

3.3 Test Corpora

Some comments are in order about various items excluded by section S–8 of the proposal. For example, ambiguity (as opposed to vagueness) is barred at present, primarily to simplify the Common Answer Specification. It would not be difficult to enhance the canonical answer to include several alternatives for queries which were genuinely ambiguous: then the Comparator could see if a system answer matched any of them, counting at least one match as correct. A more challenging test would be to only score answers as correct that matched *all* reasonable answers. This would obviously require substantial consensus on which queries were ambiguous and what the possible readings were.

Presupposition is another area where one could imagine extensions to the proposal: for queries like

> *List the female managers who make more than $50000.*

there is an implicit assumption that there are, in fact, female managers. If no managers are female, the answer would presumably be an empty relation. The *reason* the answer set would be empty, however, would have nothing to do with the data failing to meet some set of restrictions, as in ordinary cases. Rather, the object NP would have no referent: this is a different kind of failure, and systems that can detect it have achieved a greater degree of sophistication, which presumably ought to be reflected in their evaluation. Such failure to refer can be subdivided into two cases:

necessary failure: cases which are impossible by definition. For example, the set of men and the set of women are disjoint: therefore "the female men" necessarily fails to refer to any entity.

contingent failure: cases which happen to fail because of the state of the world (as in the example above).

One modification to the CAS that would include such cases would be to extend S–1 to include types for failure, with several possible values indicating the failure encountered. Then the canonical answer to the example above might be the special token

```
contingent-failure
```

rather than simply

```
()
```

Until a broader consensus on this issue is achieved, however, we consider the best approach to be the elimination of such cases from the test corpora.

Section S–8 excludes queries whose answer is not available due to missing data. As an example, in the SLS Personnel Database the home phone number for Richard Young, an attorney, is missing, and is therefore entered as `NIL`. We therefore propose to exclude queries like "What is Richard Young's home phone?" from test corpora, since no data is available to answer the question. On the other hand, the query "List the home phone numbers of all the attorneys" would not be excluded. The answer here would be the set of phone numbers, including Richard Young's:

```
(("214-545-0306") (NIL)
 ("214-665-5043") ...)
```

Queries involving sorting are currently omitted pending resolution of several technical problems:

- Given that extra fields are allowed, the primary sort keys would have to be specified by the CAS.
- Different sites might have different and incompatible approaches to sub-sorts, if the primary keys are not unique.
- Since relations are assumed to not be ordered, a different notation for sorted answers would be needed.

In addition to these problems, evaluating queries that require sorting would seem to contribute little to understanding the key technological capabilities of an SLS, and is therefore at best a marginal issue. In light of these points, we consider it expedient to simply omit such cases for the present.

3.4 Comparator

The epsilon measure proposed in S–9 assumes that, if an SLS does any rounding of real numbers, it will not exceed the epsilon figure. Two particular cases where this might be problematic are percentages and years: repeating an earlier example, the correct answer to a query like "What percentage of employees have PhDs?" might be

```
0.1064
```

and rounding this to `0.11` would score as incorrect. Similarly, for a query like "How many years has Sharon Lease worked here?", the years must not be treated as whole units: an answer like

```
36.87
```

would score as correct, but `37` would be incorrect.

One consequence of S–10 is the small possibility of a incorrect system answer being spuriously scored as correct. This is especially likely when there are only a few tuples and elements, and the range of possible values is small. Yes/no questions are an extreme example: an unscrupulous developer could *always* get such queries correct by simply answering

```
((true false))
```

since the Comparator would generously choose the right case. Eliminating such aberrations would require distinguishing queries whose answers are relations from those that produce scalars, and imposing this distinction in the CAS. We therefore assume for the time being that developers will pursue more principled approaches, and we rely on the variety of test corpora to de-emphasize the significance of these marginal cases.

4 THE COMPARATOR

In this section we describe the Comparator, a Common Lisp program for comparing system output (conforming to the CAS) with canonical answers. We have chosen this name to reflect an important but subtle point: evaluation requires human judgement, and therefore the best we can expect from a program is comparison, not evaluation. Since the degree to which system output reflects system capabilities is always imperfect, we view the results of the Comparator as only one facet of the entire effort of evaluating Spoken Language Systems. The Comparator software is available without charge from BBN Systems and Technologies Corporation, which reserves all rights to the software. To obtain a copy, contact `sboisen@bbn.com`.

The Comparator takes two inputs: the answer from a particular SLS, and the canonical answer. The output is a Boolean value indicating whether the system-supplied answer matched the canonical one. To make that judgement, the Comparator needs to perform type-appropriate comparisons on the individual data items, and to handle correctly system answers that contain extra values.

As described in S–9, real numbers are compared using an epsilon test that compares only a fixed number of the most significant digits of the two answers. The number of digits compared is intended to generously reflect the accuracy range of the least accurate systems involved. Note that there is still some danger of numeric imprecision causing an answer to be counted wrong if the test set includes certain pathological types of queries, like those asking for the difference between two very similar real numbers.

The other, more major issue for the Comparator concerns the fact that table answers are allowed to include extra columns of information, as long as they also include the minimal information required by the canonical answer(S–10). Note that these additional columns can mean that the system answer will also include extra tuples not present in the canonical answer. For example, if Smith and Jones both make exactly $40,000, they would contribute only one tuple to a simple list of salaries, but if a column of last names were included in the answer table, there would be two separate tuples.

What the Comparator does with table answers is to explore each possible mapping from the required columns found in the canonical answer to the actual columns found in the system-supplied answer. (Naturally, there must be at least as many columns as in the canonical answer, or the system answer is clearly incorrect.) Applying each mapping in turn to the provided answer, the Comparator builds a reduced answer containing only

those columns indicated by the mapping, with any duplicate tuples in the reduced answer eliminated. It is this reduced answer that is compared with the canonical answer, in terms of set equivalence.

Finally, it should be stressed that the Comparator works within the context of relational database principles. It treats answer tables as sets of tuples, rather than lists. This means first that order of tuples is irrelevant. It also means that duplicate tuples are given no semantic weight; any duplicates in a provided answer will be removed by the Comparator before the comparison is made.

5 CORPUS DEVELOPMENT AND TAGGING

Any corpus which is collected for SLS development and testing will be more useful if it is easily sub-divided into easier and harder cases. Different systems have different capabilities, particularly with respect to handling discourse phenomena: the ideal corpus will therefore include both the most basic case (i.e., no discourse phenomena) and enough difficult cases to drive more advanced research.

We propose the tagging of corpora using a hierarchical categorization that reflects the richness of context required to handle queries. These categories primarily distinguish levels of effort for Spoken Language Systems, such that the lowest category should be attempted by every site, and the highest category attempted only by the most ambitious. Two other criteria for the categorization are the following:

- Categories should be maximally distinctive: there is no need for fine distinctions that in practice only separate a few cases.
- Categories should be easily distinguishable by those who will do the tagging. That is, they should be objective and clearly defined rather than relying on sophisticated linguistic judgements.

Here is a candidate categorization, where the category number increases as the context required becomes successively richer:

Category 0: no extra-sentential context is required (i.e., "0" context): the sentence can be understood in isolation. This is the default case.

Category 1: "local" extra-sentential reference, excluding reference to answers: that is, the sentence can be understood if the text of the previous question is available. One is *not* allowed to go back more than one question, or look at the answer, to find the referent.

Category 2: ellipsis cases, such as the following sequence:

"What's Sharon Lease's salary?"
"How about Paul Tai?"

Category 3: "non-local" reference. The referent is in the answer to the previous query, or in the text of a query earlier the previous one. This probably includes several other kinds of phenomena that would be usefully separated out at a later date.

Category X: all cases excluded from corpora, both for discourse and other reasons (see S–8).

We propose two initial uses of this categorization for SLS evaluation: creating basic test corpora of Category 0 queries, and designing simple discourse corpora that include Category 1 queries (see S–7). The other categories would enable developers either to focus on or to eliminate from consideration more difficult kinds of discourse phenomena.

There may be other categories which are of interest to particular developers: for such cases, it is suggested that the developer do their own site-specific tagging, using those features which seem reasonable to them. This scheme is offered solely to expedite community-wide evaluation: there are many possible categorizations which might be useful for other purposes, but they are independent of the considerations of common evaluation.

6 CORPUS COLLECTION

The evaluation scheme described in the previous sections assumes the existence of realistic test corpora: it says nothing, however, about how such corpora will be collected. This section describes our approach to corpus collection, which uses a human simulation of a Spoken Language System. Since the test and training corpora will be shared, they need only be collected once, and then distributed to SLS developers. The setup described

here has been delivered to Texas Instruments for use in their effort to collect a common corpus for the SLS community.

Our approach to corpus collection is to simulate the behavior of a spoken language system (SLS) for database access which is beyond the state of the art, but within about 5 years of effort. We are concerned with obtaining a large corpus of spontaneous task-oriented utterances, without being restricted by current capabilities. Such a corpus would help in prioritizing research for the next few years, while providing a challenging base of examples for system evaluation.

Our methodology has been guided by the following primary goals:

- The subject should be as unbiased as possible in terms of the kinds of language which are possible.
- The subject should be performing a task, in order to get realistic input exhibiting context-dependent phenomena.
- The simulation mechanism must be fast enough to maintain the interaction in a natural fashion.
- The simulation must be accurate enough that the subject can rely on it: otherwise interactions tend to degenerate into consistency checking by the subject.
- The system must behave cooperatively, especially when it is unable to answer a query, so as not to frustrate the subject.
- Minimal training should be required so that relatively naive subjects can be utilized.

We accomplish these goals by providing a human simulator (here referred to as the *Wizard*) of a speech recognition system, using a "PNAMBC" setup ("Pay No Attention to the Man Behind the Curtain"). The Wizard is provided with tools to obtain and display answers to user queries, and to provide appropriate diagnostic messages when the query cannot be satisfied.

6.1 The Setup

In a data collection session, the subject is first provided with the following:

- general information describing the session
- a description of the database
- a list of possible tasks for the subject to pursue (including specific details, but *not* examples of English queries)
- pen and paper

The subject and the Wizard are kept in separate rooms, in order to remove any influence of the Wizard's actions on the subject. The Wizard is only allowed to communicate with the subject by displaying information on the subject's console, either as an answer to a request, or by using one of a fixed set of canned messages. A "push to talk" microphone is used by the subject when directing requests to the Wizard, and his utterances are recorded. A diagram of the setup is in Figure 2.

There is a cyclic interaction which begins by the subject verbalizing a query, which the Wizard transcribes and sends as text to the subject's screen. The Wizard then submits the text to a natural language (NL) tool: we have used Parlance™, BBN's commercial NL interface, for this component. If the query is correctly processed, the Wizard echoes the response to the subject's screen. If the NL tool fails, the Wizard may revise the request and try again until it is understood, or until a predetermined time limit expires (usually around a minute). All revision is "behind the curtain", and unseen by the subject. The speed of the NL tool (a few seconds per query for Parlance) allows for several revisions of a request if necessary, while still keeping a fast turnaround time. If the Wizard is unable to obtain a correct answer, or the request is outside the scope of the simulation, the Wizard sends one of a predefined set of canned messages to the user (see the next section for examples). All input and output to the Wizard's window is recorded in a log file.

With this setup we are able to collect approximately 50 queries per hour, using one subject per hour.

6.2 The Wizard

Beyond the simple transcription of the spoken query, the role of the Wizard is to extend the linguistic coverage of the NL tool (by revising queries when necessary), while keeping the simulation within the bounds of a system that is realistically attainable in a few years. In our preliminary experiments, the Wizard's involvement increased the coverage of requests that were within the scope of the simulated system from 67% to 94%.

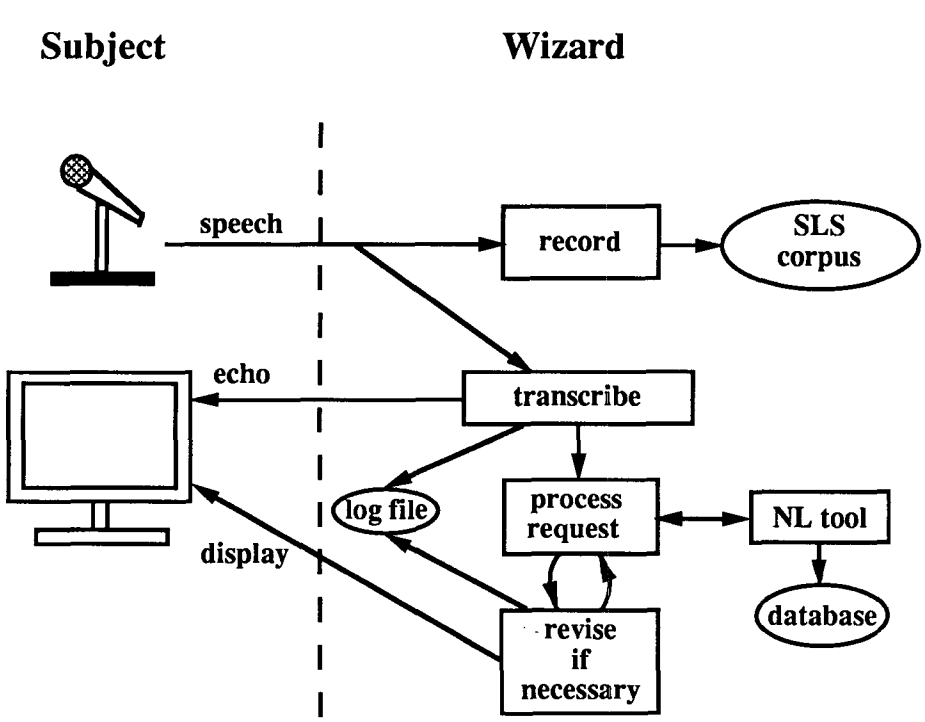

Figure 2: Data Collection Setup

Another important aspect of the Wizard's job is to decide when a query is beyond the simulated system's scope and give an appropriate reply. Any request that can be understood by the NL tool is considered safely within the scope of our simulated system. However, other requests may be problematic, for example, queries that require meta-level knowledge like "Are you able to compute percentages?". When a query cannot be answered within the bounds of the simulation, or the Wizard is unable to obtain an answer due to an NL tool limitation (even though the query is considered reasonable), the Wizard selects an appropriate canned message and dispatches it to the subject. Currently there are 11 such messages, including the following:

- "No information is available on _____ ": the information requested is not in the database—the Wizard fills in the blank.
- "No information about the information itself is available", when the subject's request assumes meta-level knowledge.
- "Sorry, the system is unable to answer your query", when the NL tool fails to process a request that *is* within the scope of the simulated system.

A successful Wizard needs to know the domain, though "expert" status is not necessary. In addition, the Wizard must be familiar with the database and with the extent of coverage provided by the particular Parlance configuration. Some linguistic knowledge is also useful to quickly diagnose likely problems with complex linguistic constructions. We've found that a large set of sample sentences, and few days experience with the NL tool, is sufficient to train Wizard apprentices.

6.3 The NL Tool

The NL tool clearly performs a crucial role in the Oz simulations. In principle, this tool could be just a low-level database query language interface. This would sacrifice turnaround speed, however, since the Wizard would have to compose and enter long, complex database queries.

Using Parlance as a NL tool provides a very fast and reliable system for processing the natural language requests. In our initial experiments simulating an SLS front end to a personnel database, 67% of the requests that were within the bounds of the simulated system were handled by Parlance without revision. Parlance also provides us with the Learner™, its knowledge acquisition tool. With the Learner, Parlance can be quickly brought to high coverage on a new database, allowing for corpus collection in various domains if so desired.

A A BNF SPECIFICATION FOR THE CAS NOTATION

The following BNF describes the syntax of the Common Answer Specification:

answer ⟶ *scalar-value* | *relation*

boolean-value ⟶ `true` | `false`

date ⟶ " *digit digit* - *month* - *digit digit* "

digit ⟶ `0` – `9`

month ⟶ `JAN` ... `DEC`

number-value ⟶ *integer* | *real-number*

relation ⟶ (*tuple**)

scalar-value ⟶ *boolean-value* | *date* | *number-value* | *string*

tuple ⟶ (*value*+)

value ⟶ *scalar-value* | `NIL`

We assume as primitives the values *integer*, *real-number*, and *string*. Integers and reals are not distinguished. Note that, unlike the special tokens `true`, `false` and `nil`, strings are case-sensitive, and in the personnel domain should usually be upper-case (since that's the form used in the SLS Personnel Database). Only non-exponential real numbers are allowed.

Answer relations must be derived from the existing relations in the database, either by subsetting and combining relations or by operations like averaging, summation, etc. For the SLS Personnel Database, this implies allowing NIL as a special case for any value.

Empty tuples are not allowed (but empty relations are). Reiterating S–2, all the tuples in a relation must have the same number of values, those values must be of the same respective types (boolean, string, date, or number), and the types in the answer must be the same as the types in the database (i.e., database values like "1355" cannot be converted from strings to numbers in answer expressions).

B A SAMPLE TEST CORPUS

Based on the Common Personnel Database, we have prepared a small corpus of queries and their answers in the Common Answer Specification. These provide concrete examples of the answer expressions that a Spoken Language System is expected to match, within the leeway allowed by the Comparator (Section 4). The intent here is not to provide a wide range of natural language contructions, but rather to provide a check on the clarity and completeness of the CAS. Note that discourse capabilities are not exercised in this corpus.

"How many employees are there in department 45?"
```
48
```

"When did Sharon Lease join the company?"
```
"06-OCT-52"
```

"Show me a list of the Hispanic employees."
```
((1468) (4688) (6213))
```

"Are there any clerks who make over $50000?"
```
false
```

"What's the number of female scientists in the company?"
```
20
```

"How long ago did Sharon Lease join the company?"
```
36.87
```

"How many years has Sharon Lease worked here?"
```
36.87
```

"Does Sharon Lease work in department 10?"

```
true
```

"Are Sharon Lease and Paul Hayes both in department 10?"

```
false
```

"Sum the salaries of the managers."

```
2331300
```

"Give me the average salary of managers who graduated from Harvard."

```
72400.0
```

"What are the ethnic groups of the members of department 11?"

```
(("BLACK") ("HISPANIC")
("WHITE"))
```

"Show me every manager's last name and the number of people they supervise."

```
((71 "GRAY") (1 "BRADY")
(40 "JACOBSON") (4 "BEEK")
(3 "EASTON") (1 "BOFF")
(8 "BANDANZA") (20 "BURRUSS")
(96 "MCGLEW") (12 "FRANKLIN")
(24 "PAYNE") (5 "BUONO")
(28 "MACLEAN") (4 "LANDER")
(29 "FAY") (47 "SEABORN")
(3 "LI") (26 "DI"))
```

"List the degrees of Sharon Lease, Jose Vasquez, and Martha Nash."

```
(("BA" 1028 3761) ("MA" 1458
3514) ("BA" 1458 3434) ("PHD"
2099 3434) ("MA" 2099 3434)
("BA" 2099 2074))
```

"Tell me the names of all senior scientists."

```
(("BURRUSS") ("SPATOLA")
("SYKLEY") ("CAVANAUGH")
("KANYUCK") ("SEABORN")
("KINGSLEY") ("MESSENGER")
("ZINKY"))
```

"What is Sharon Lease's phone number?"

```
"6218"
```

"What's the average salary for level 31 employees?"

```
((270275.0))
```

"List the home phones of people living in Irving."

```
((NIL) ("214-545-0306")
("214-665-5043")
("214-492-3575")
("214-696-8397")
("214-625-4682")
("214-884-1708")
("214-484-3185")
("214-646-4192"))
```

"List all the programmers in department 20."

```
()
```

"What percentage of employees have PhDs?"

```
0.1064
```

"Give me a list of the addresses of all the accountants."

```
(("9 LAKEVIEW TERR")
("35 MYOPIA HILL RD")
("58 HIGH ST")
("30 DELANO PARK")
("74 GODEN ST")
("49 LAWMARISSA RD")
("240 BRATTLE ST")
("380 WALTHAM ST")
("223 BROADWAY")
("33 SUMMIT AVE") ("269 RICE ST")
("21 HAWTHORNE ST") ("76 WEBSTER ST")
("55 OAK ST") ("147 GLOUCESTER ST"))
```

Acknowledgements

The work reported here was supported by the Advanced Research Projects Agency and was monitored by the Office of Naval Research under Contract No. 00014–89–C–0008. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States Government.

References

S. Boisen. *The SLS Personnel Database (Release 1).* SLS Notes No. 2, BBN Systems and Technologies Corporation, 1989.

L. Ramshaw. *Manual for SLS ERL (Release 1).* SLS Notes No. 3, BBN Systems and Technologies Corporation, 1989.