KITSS: A Knowledge-Based Translation System for Test Scenarios

Van E. Kelly and **Mark A. Jones**
AT&T Bell Laboratories
Murray Hill, NJ 07974-0636
vek@research.att.com, jones@research.att.com

Abstract

Machine-assisted language translation systems for technical documents, guide humans through a process of selecting and composing variant partial translations. The constrained nature of technical sublanguages makes language processing aids cost-effective to build and use.

Analogously, we have developed KITSS, a *knowledge-based translation* system for converting informal English scenarios of the desired behavior of complex reactive systems into formal, executable test scripts. A trainable parser and reference resolver capture domain-specific linguistic knowledge. A logic analyzer establishes coherence in the translation process in a role comparable to a "story understander". It checks the consistency of each step of a translated test script using a theorem prover, a planner, and logic-encoded background knowledge about the system under test. This helps correct common but serious specification errors, including underspecificity, omitted steps, and even some outright mis-statements.

To evaluate how well such technology can scale, we have exercised our technology progressively on a graduated corpus of 100 behavior scenarios spanning 7 advanced calling features for a private telephone switch (PBX), successfully translating 70% into test scripts without any manual post-hoc editing. Our experience with KITSS has enabled us to identify many of the tradeoffs in accommodating informality in specification, versus demanding formality from a human agent.

Introduction

Problem Statement

Engineers who specify complex reactive systems, such as telephone switches, are often unfamiliar with expressive specification formalisms like temporal logics. Instead, they state functional requirements in thousands of stylized English prose scenarios describing linear traces of specific desired behaviors, such as Figure 1. These scenarios form an informal contract with other engineers who will eventually validate the final product. For an initial release of new functionality, it makes economic sense to execute each required scenario manually once or twice, but for later repetitive regression testing, a manual approach is not cost-effective. Informal scenarios must be converted into formal, executable test scripts and maintained in this form. This conversion, in practice, is a laborious and costly process for test engineers, averaging several hours per scenario initially, plus revisions for subsequent product releases. When product features and versions proliferate, the cost of generating and maintaining a library of thousands of scripts only gets worse.

Action: `(1.1) Place calls to stations B3 and D1 and make them busy. (1.2) Station B1 calls station B2. (1.3) B2 does not answer.`

Verify: `(1.4) Ringback to station B1 is replaced with Call Coverage tone.`

Action: `(1.5) Wait for CRI timeout at B1.`

Verify: `(1.6) Station B1 receives ringback again. (1.7) The call keeps ringing at station B2 (the call will not go to coverage since all the covering users are busy).`

Figure 1: Sample Behavior Scenario

Solution Approach

Because the language used in informal scenarios is not really standard English but an extremely stylized technical subdialect, it seemed promising to try automated natural-language processing to speed the conversion and maintenance processes. In translation of specialized technical documents between two natural languages, say from French to Swedish, customizable interactive translator systems have recently logged clear productivity benefits over manual approaches, especially for restricted technical dialects. These tools do not replace human translators, but recast their roles as primarily one of selecting and composing partial translations suggested by the tool. We adopted this approach to structuring human-computer cooperation

as one of the major paradigms for our own solution approach. As in most machine translation applications, we began with a large extant corpus which had to be substantially parsed "as we found it" in order to cost-justify our approach. It was not sufficient merely to design a suitably habitable dialect for the expression of future test scenarios.

In machine translation from one natural language to another, a direct translation of individual sentences, or even just of individual phrases and clauses, usually provides at least a marginally intelligible result, but this is not so for our task. First of all, an wide abstraction level gap must be bridged between the goal-oriented intensionality of scenarios and the procedural extensionality required in executable scripts. Oblique descriptions of actions and objects in scenarios must be identified with concrete operations and devices in a telephone switch test lab. For example, an action such as "Place a call to station B1" involves choosing a station to place the call, going off-hook, mapping "station B1" to its extension, and choosing a method for dialing (dialing digits, speed dialing, last number redial, etc.). The problems of "extensionalizing" nominal descriptions have been partially addressed by natural language database front-ends, where noun phrases are resolved using known objects and attributes in a database schema (*e.g.,* [Ballard and Stumberger 1986]), although such front-ends do not generally jump nearly as wide an abstraction gap as KITSS must. Nor have database front-ends had to deal with narrative text forms, nor have they faced head-on the history-sensitive scoping of references within narrative discourse structures.

Beyond the problems of bridging abstraction gaps on the level of individual phrases and statements, there are also "discourse level" semantic hazards in informal scenarios:

- entire steps can be left out,
- important timeouts and other "null actions" go unmentioned (*e.g.,* Figure 1, sentences 1.3, 1.7),
- actions may create "hidden" linguistic referents, such as when explicitly dialing a string of digits implicitly creates a new telephone call,
- actions and observations can be underspecified, such as in sentence 1.1 where there are several non-equivalent ways to force phones to become "busy",
- essential initialization steps and boundary conditions remain implicit, as in sentence 1.7 where the "covering users" (*i.e.,* telephone call screeners, probably secretaries) of B2 are mentioned but there were no explicit administrative steps to make B3 and D1 function in that capacity.

Sometimes, there are also just plain mistakes. A human concentrating on the purely linguistic aspects of the translation task is likely to miss at least some of these problems. An automated capability within the scenario translation process must flag such problems and guide humans in correcting them; a *story understanding* functionality is needed in addition to linguistic expertise. Although automated story understanding is very difficult in general, a telephony testing domain is, fortunately, much simpler to axiomatize than the "real" world.

An Implementation

KITSS (a.k.a. the Knowledge-based Interactive Test Scripting System) is our prototype system for formalizing and analyzing scenarios of telephone switch behavior for medium-sized private-network switches (PBX's). It was deliberately designed to integrate smoothly with an existing test process, partly automating one manual task without otherwise impacting a test engineer's job. Its translation subsystem guides the conversion of each sentence of an English scenario into an equivalent statement in WIL, a logic-based interlingua previously used in the WATSON system ([Kelly and Nonnenmann 1991]). The new natural-language technology we developed for this task includes an extremely high-performance adaptive statistical chart-parser [Jones and Eisner 1992] and a rule-based phrase converter which performs nominal reference resolution and case-frame normalization. English phrases converted into WIL can be paraphrased back into a stylized pseudo-English. Users can also directly input the pseudo-English forms as an alternative to the full English interface. No human ability to read or write WIL directly is required.

Scenario understanding and analysis is provided by a heavily instrumented "interpreter" which analyzes the translated WIL statements using a coarse black-box simulation of a telephone switch, prototyped using a theorem prover, a telephony domain theory expressed in logic, and a library of stereotypical plans for telephone usage. Whenever a scenario leads to an impossible state, such as trying to answer a telephone call which is not locally present, the interpreter summarizes the anomaly and, where possible, suggests plausible corrections, (such as answering the call from another station where it is present). Plausible elaborations of underspecified activities during the scenario, such as making a station become busy, are calculated using a planner and the plan library. Implicit initialization steps and test lab configuration conditions for test execution are also made explicit. As during translation, the analyzer keeps the human user "in the loop" for all modifications and elaborations of the WIL script, and all user interactions use pseudo-English paraphrases of WIL constructs.

A more extensive description of our system architecture, knowledge bases, user interface, and early experimental results was published in [Nonnenmann and Eddy 1992]. That paper also discusses our approach to solving the script maintenance problem for scenarios that have previously been analyzed. In the current pa-

per we focus on the natural language translation technology, its interface with the scenario analyzer, and the lessons learned during the latter half of the project about building and maintaining a large hybrid reasoning system. In the remainder of this paper, we first describe each of the two major functions of KITSS: translating English into WIL and analyzing the translated scenario for anomalies.

Translating English into WIL

The translation of English scenarios into WIL occurs in two stages. Initially, a statistical, trainable chart parser converts English sentences into a case frame structure similar to that of a Lexical-Functional Grammar (LFG) [Kaplan and Bresnan 1982]. The case frame is then translated by a rule-driven semantic resolver into WIL. The semantic resolver performs several tasks during its translation step, including canonicalizing the case frame, handling conjunctions, and resolving definite and indefinite noun phrase references.

In many cases, there may be more than one possible WIL reading of an English sentence. This may be due to the inherent ambiguity of natural languages, or to imprecision in the parsing and reference resolution steps. The statistical chart parser computes probabilities which aid in ranking the alternatives that arise as a result of grammatical (but not referential) ambiguity. The translator, following the highly interactive design philosophy of KITSS, also includes a simple rule-driven facility which "paraphrases" WIL back into English. The choices are ranked by the probabilities from the parser. The human may select one of these translations, or else create a novel one by cut-and-pasting WIL fragments from several of the "near-misses".

Note that we are not claiming to have successfully automated the translation of English into WIL, even for the restricted English subset we use. The conditions of use that make our translation process practical are the very same ones that differentiate past research failures in *automatic* language translation from recent commercial successes in *machine assisted* translation:

- It is not necessary to produce an automatic, correct translation of every sentence.
- Human users get machine help for salvaging something useful out of incorrect "near-miss" translations (*e.g.,* cut-and-paste), and are not unduly penalized for an occasional end-run around the translator and writing directly in the target language (*i.e.,* WIL or its pseudo-English paraphrases).
- The size of each corpus to be translated is large enough to amortize customizing the translator for a particular writing style, vocabulary, and topic of discourse.

Statistical Chart-Parsing

Natural language parsers have traditionally required elaborate hand-crafted grammars which subtly depend on the chosen parsing algorithm. This has made them complex to develop and maintain, even for computational linguists. We needed a parser that could be maintained by our prospective end-users: engineers, not linguists. Instead of a large, pre-specified covering grammar, the KITSS grammar has grown incrementally from training on example sentences from our domain corpus of scenarios. The statistical chart-parsing technology used in KITSS was inspired by the recent success of statistical methods in several areas of natural language processing, including part-of-speech tagging, bilingual corpora alignment and OCR postprocessing. Statistical knowledge can be used very effectively to limit search and to order alternatives.

To "train" the parser on a sample set of sentences, a human feeds the parser a **parse tree** of the sentence (if the parser cannot already parse it correctly). Parse trees, for KITSS, are basically annotated versions of familiar elementary-school "sentence diagrams". The part-of-speech categories in KITSS were taken from the Brown Corpus [Francis and Kucera 1982]. The non-terminal categories are the standard ones in a simple linguistic theory such as `S` (sentence), `NP` (noun phrase), *etc.* The parser assists in the bracketing process by providing analyses for major sentence constituents that it can find. In addition, there is a facility for guessing parts-of-speech for new vocabulary.

Each node in the parse tree is implicitly identified with some context-free rule. Each non-terminal category also has a predefined semantic type (*slot, filler, slot-filler pair*) that is used to construct a case-frame representation. Relations such as prepositions (`IN`) are of type *slot.* Entities such as noun phrases (`NP`) or verb phrases (`VP`) are of type *filler.* Modifiers such as adjective phrases or prepositional phrases are of type *slot-filler pair.* To construct the case frame, the parser associates one or more semantic templates with each syntactic rule. The templates play a similar role (without unification) to the functional equations in LFG.

For example, the template `(@?1 :OBJECT ?2)` is associated with the rule `VP -> V NP`. The variable `?1` is bound to the semantic interpretation of the first right-hand side constituent (`V`). The template specifies that this interpretation is to spliced in and followed by the slot `:OBJECT` and its filler from the interpretation of the `NP`. A semantic template only needs to be explicitly defined for the rule if it cannot be "guessed" by the parser from the semantic types of the non-terminals and general linguistic knowledge about the heads of syntactic phrases (*e.g.,* from X-bar theory [Jackendoff 1977]). For example, for the syntactic rule `VP -> VP PP`, the parser will supply a default template of the form `(@1 @2)`. In most cases, users need not explicitly specify these semantic templates.

The case frame representation includes predicate-argument information and syntactic features such as tense. It is often much "flatter' than a parse tree. Figures 2 and 3 give the parse tree and case frames for the

```
(S (VP (VP (VP (VB "Place")
               (NP (NNS "calls")))
           (PP (IN "to")
               (NP (NNS "stations")
                   (NPR (NPR "B3")
                        (CC "and")
                        (NPR "D1")))))
       (CC "and")
       (VP (VB "make")
           (SBE (NP (PPO "them"))
                (JJ "busy")))))
```

Figure 2: Parse Tree

```
(AND :CONJ1
     (PLACE :OBJECT (CALL :NUMBER PLUR)
            :TO (STATION :NUMBER PLUR
                         (AND :CONJ1
                              (:NAME "B3")
                              :CONJ2
                              (:NAME "D1"))))
     :CONJ2
     (MAKE  :PRED (BE :ACTOR (THEM)
                      :ADJ-MOD BUSY)))
```

Figure 3: Case Frame

example sentence (1.1) "Place calls to stations B3 and D1 and make them busy".

Rule-Based Phrase Conversion

The second phase of the translator converts the parser's case frames into WIL logic statements describing precise facts about events in a hypothetical telephone test lab. In contrast to the large number of things that might be said in a document about testing (instructions to testers, runing commentary, explanatory headings), there is but a small number of facts specifically relevant to executing a particular test. The tasks of the phrase converter are:

- identifying references to relevant test lab actions or observations in a sentence (normalize verb concepts),
- determining which pieces of lab apparatus are to participate in the action or observed event (resolve nominal references).
- identifying simple plan-like discourse structures in the text (*e.g.*, sequence and purpose).
- paraphrasing its fully normalized and resolved understanding of each sentence back into an English-like form.
- accepting corrections from the user in the form of edited versions of near-miss paraphrases.

With various degrees of success, the phrase converter also applies a set of rewrite rules to handle a number of natural language phenomena including syntactic transformations (*e.g.*, active-passive), semantic transformations (*e.g.*, "button at station X" and "button associated with station X" are equivalent), and collective/distributive readings of conjunctions. These rewrite rules include difficult situations such as the first sentence in Figure 1 where the conjunction *and* should be interpreted as "and thus" rather than "and then". After determining a canonical representation, then the nominal descriptions are further resolved into entities in the domain model. Finally, the resulting form is converted into WIL.

For example, in our sample sentence 1.1, the phrase converter identifies the referent of "them" in the second clause, namely "**B3 and D1**", associates the verbs with specific WIL action predicates (**place-call** and **busy-out-station**), and links the referents of the subjects and objects of each clause with the required parameters of each action. Next, it guesses that "and", in this context, does not mean sequence ("and then") but rather purpose ("and thus"), based on the underspecificity of the first clause (calls from whom? how many?). Finally it reads "make B3 and D1 busy" distributively as "make B3 busy and make D1 busy". The final (paraphrased) translation of sentence 1.1 into WIL, is simply:

```
Busy-out station B3.
Busy-out station D1.
```

The underspecified information in the first clause of the sentence blocks its translation into WIL, and so it is dropped from the final translation as it occupies only a subsidiary discourse role. The user could, of course, re-introduce the missing information directly by typing an explicit WIL paraphrase, but in this case no harm was done.

Analyzing WIL Test Scripts

Converting each English scenario sentence correctly into a WIL formula does not guarantee that the whole sequence of formulas constitutes a valid test script. To repair the anomalies noted earlier – missing steps and null actions, underspecified and steps, implicit boundary conditions – our analysis module performs two different audit techniques on the scenario:

First, a background theory of telephony, about 250 hand-coded temporal logic axioms at present, is used to audit each scenario step in isolation; the logical conjunction of this background knowledge and the scenario step itself (*i.e.*, an action plus follow-on observations) is fed into a theorem prover, which forward-chains additional facts about the step. The background theory primarily describes user-telephone signaling conventions (ringing, tones, pickups, hangups, button-presses, lamp flashes, timeouts) and the procedural building blocks of all modern telephone call control (call origination, acceptance, rejection, bridging, transfer, redirection, drop, and hold, *etc.*). Deliberately **excluded** from this theory is any formal specification of the particular features being described in our

scenarios.

Second, at each step, the scenario so far is compared with a library of hierarchically structured plans for stereotypical telephone usages, currently numbering about 70. This comparison is complex because telephone usages typically involve multiple agents, and multiple plan executions (*e.g.*, several simultaneously active telephone calls) may be interleaved within a given scenario. Furthermore, plan recognition must be updated after each scenario step in order to synchronize with the translation process, and recognition and plan instantiation (*i.e.*, fleshing out underspecified scenario actions) are freely intermixed. These requirements have forced us to engineer a complex, incremental plan recognition/instantiation algorithm based on propagating disjunctive sets of plan hypotheses regarding each scenario step.

The final desired result of this analysis is a reconstruction of the original scenario as a threaded forest of plan executions, in which every node, whether leaf or interior point, has been examined and elaborated using the background domain theory. As this is being computed, a variety of corrections and clarifications are carried out:

- calculating the detailed "state" of a telephone switch test lab before and after each scenario action using the background theory of telephony,
- finding "hidden" linguistic referents in the scenario by plan recognition, which were missed by the phrase converter. For instance, consider the fragment

  ```
  B1 goes offhook.
  B1 dials the extension of B2.
  B2 answers the call.
  ```

  There is no immediate referent here for "the call"; its identity can only be inferred by matching this fragment against plan knowledge about placing calls.
- deducing desirable but unstated intermediate observations that should be made during the course of a test (such as always making sure a phone is actually ringing before attempting to answer a call),
- determining whether each action is possible and legal in the current interpolated state, and if not, whether one single missing prior step (such as forgetting to hang up a phone – the most common error – or not waiting for a timeout to expire) could account for the discrepancy,
- elaborating details of abstract or underspecified goals (such as how to make a station "busy"). Plan selection is guided by entity types (how many calls a particular class of station can handle simultaneously), the current state (how many calls it is currently handling; which other stations are now free), and pragmatic concerns (whether it is faster to make a station busy with outgoing or incoming calls).
- Using the plan library to diagnose missing finalizations in a scenario and for selecting appropriate error recovery actions. For instance, if some action sequence implicitly activates a feature, that feature should be explicitly deactivated in case of aborting the test.
- deducing any special capabilities or privileges required by any of the participants in the scenario (*e.g.*, the ability to forward telephone calls), which need special administrative setup actions.

Finally, just as for the language translation module, the scenario analyzer had to be designed robustly so that its own failure to understand, especially on some fairly minor point, does not block KITSS from producing output. It accommodates manual on-the-fly "patching" of scenarios by cut-and-paste of WIL paraphrases, followed by incremental re-auditing. It permits the user to resolve contradictions by explicitly denying facts it has deduced and asserting others. In the face of massive apparent nonsense, it degrades gracefully, amid much complaining, to a credulous mode where most of its audits, except for plan instantiation, are disabled. This credulous mode persists until KITSS unambiguously recognizes the start of a new plan and resynchronizes itself with the scenario.

Empirical Results and Lessons Learned

We used KITSS experimentally to translate a graduated corpus of 100 scenarios covering seven advanced calling features of a private telephone switch, written by five different authors. A very experienced user of KITSS successfully converted about 70% of these scenarios into executable form, taking only a few minutes apiece, as opposed to hours without machine aid. These 70% required either zero or minimal post-hoc editing of the final output to produce a "perfect" test script, as judged by experienced test engineers. The remaining 30% were evenly divided between those requiring enough manual touch-up to nullify KITSS' productivity advantage over manual conversion, and those for which KITSS provided no useful help at all (commonly due to fatal bugs in the tricky incremental re-auditing code of the analyzer).

For about one-third of the 70% of scenarios successfully processed, KITSS degraded into "credulous" mode part of the time, requiring its human user to interpolate corrections which it otherwise would have provided. In our experiments, human performance on these troublesome cases varied over a three-fold range, even among members of our development team. This shows that although the basic technology and architecture of KITSS may be sound, more work is needed to refine it into an industrial-strength tool for general engineering use.

Emergent Natural Language Technology

The chart parser is one of the clearest research successes on the KITSS project, and the most readily

transferable to other application domains. Evaluated in isolation through split-corpus experiments on a 429-sentence subset from our scenario corpus, the parser, even though still quite obviously undertrained, performed as follows:

- 77% overall chance of parsing a novel sentence correctly.
- 85% chance of success when the sentence contains no new words.
- 96% success in determining the correct parse in cases where **any** possible parses were found.
- Parsing speed averaging a few seconds per sentence and **linear** in the length of the sentence.

Thus, the parser is sufficiently accurate and predictable to be a productivity aid, and it is fast enough to be considered "real-time" for interactive use. Furthermore, most of its non-vocabulary related failures occurred with long, awkward constructions or embedded parenthetical expressions. Thus, writers can produce more parsable prose just by improving their writing style.

Our positive experience with the statistical parser convinced us that natural language parsing technology is poised for much wider exploitation as a computer interface technique. The skills required to prepare a training set for an adaptive parser are much more mundane than those formerly required to write a computational grammar from scratch; most high school students are trained in sentence diagramming, which is a form of parsing.

Although the phrase converter worked, it did not scale up nearly as gracefully as the parser. Nominal reference resolution for physical objects like calls, stations, buttons, and lamps was implemented using classification combined with a simple recency heuristic and worked reasonably well. The variability of the English language, however, worked against finding any compact set of normalization heuristics for phrases in general. Each new idiom encountered, and each significant variation in verb tense or word order, needs its own normalization rule. The number of these continues to grow linearly with the number of scenarios.

It should be noted that the 100 English scenarios which we attempted to process were originally written for humans and not machines. If a system such as KITSS were deployed earlier during the writing of the scenarios, there would be less inherent variability, and hence less of a problem with extensibility and maintainability in the phrase converter. In the end, we learned to live within the converter's limited repertoire of normalizations by performing manual touch-up on partially normalized sentences. This was still faster and easier than unassisted manual English-WIL translation.

Generic Domain Models

The scenario analyzer was the largest component of KITSS, its greatest time bottleneck, and the largest source of lingering program bugs. Notwithstanding the obvious engineering complexity of the tightly coupled, incremental, interruptible, restartable audit routines, the greatest lessons learned from the analyzer derive from its 250-axiom, 70-plan "generic" domain model for modern telephony. While simplified models of old-fashioned telephony are common textbook exercises in formal specification, efforts to formalize modern telephony have been few and mostly unfruitful. To our knowledge, KITSS included the first formal, fully machine-interpretable model of the underpinnings of modern telephony, and it has directly catalyzed further work in this field [Zave and Jackson 1991].

The largest intellectual challenge we faced in structuring this knowledge was separating general, re-usable constraints about telephony from specific knowledge of particular features. For instance, instead of directly describing the operation of the Call Forwarding capability, we considered the general rules and constraints for redirecting calls from one location to another, of which Call Forwarding was but one example. In a pure logic notation where each axiom is textually independent, it is especially easy to write constraints that are either overly or underly general, and these have mysterious effects on the advice offered by the analyzer. Our prior work on automated specification induction using the WATSON system ([Kelly and Nonnenmann 1991]) yielded useful heuristics for tracking down and fixing such errors, but we could only apply these manually, since the size of KITSS' domain model was beyond anything WATSON's automated techniques could handle.

Engineering the plan library posed two major challenges. First of all, plans had to do double-duty for both recognition and instantiation. For instantiation, they had to be very complete and detailed, but this complicated recognizing a plan from a scenario in the presence of both interference from other interleaved plans (*e.g.*, other telephone calls), and "observational noise" from omitted steps. We encountered a tradeoff between cluttering our plan representation with many explicit recognition cues or using a more complex set of recognition heuristics, which is the course we ultimately chose. Second, in a testing application like KITSS, deliberate plan failures are frequent, such as attempting to complete a telephone call to a busy station. We need to reason about the many ways a plan might fail, without cluttering the plan library with variants for each possible failure mode. Our compromise was to associate failures only with goals, not with individual plans, while relying on the densely hierarchical structure of the plan library (short plans, deeply nested) to help localize the point of plan failure. Again, we accepted more complex but less complete planning algorithms, in the interest of a simpler, clearer domain model.

KITSS, like many other systems that reason about actions, had to handle the "frame problem" and other non-monotonic effects, but the size, scope, and incre-

mental growth of our domain model limited our solution approaches. Since our domain model had to be inspectable by human telephony domain experts, we wanted to keep our logic notation as "clean" as possible, uncluttered by abnormality predicates or other forms solely intended to guide non-monotonic inference. After experimentation, we solved our frame problem extra-logically by defining an arbitrary *persistence* partial order on state-predicates, enforced by controlling the order in which clauses were fed to the theorem prover. While this approach is not state-of-the-art, it worked for our domain. We would have used a more modern, general technique, had we found one appropriate for our logic that did not entail massive obfuscation of the domain knowledge base and unacceptable slowdown of the analyzer. We suggest that more empirical scaling research is needed on the practical computational demands for the currently favored methods of reasoning about actions and state.

Acknowledgements

Many individuals contributed to KITSS. Mark Jones wrote the parser, with help from his summer student, Jason Eisner. Robert Hall built the phrase converter. Van Kelly is responsible for the analyzer and its domain model of generic telephony. John Eddy, an experienced test engineer, was our in-house domain expert and designed our knowledge base ontology and taxonomic schema. Uwe Nonnenmann built the user interface and integration tested all our code. We also wish to thank Bruce Ballard, Lori Alperin Resnick, Tom Kirk, and Jim Piccarello for their technical assistance. This project would not have been possible without far-sighted management support and encouragement from G. D. Bergland, Ron Brachman, and Jim Shanley.

References

Ballard B. and Stumberger D. 1986. Semantic Acquisition in TELI: A Transportable, User-Customizable Natural Language Processor. In *ACL-24 Proceedings*, pp. 20-29. Association For Computer Linguistics

Francis, W. and Kucera, H. 1982. *Frequency Analysis of English Usage.* Houghton Mifflin, Boston, 1982.

Jackendoff, R. 1977. *$\bar{X}$ Syntax: A Study of Phrase Structure.* Cambridge, MA.: MIT Press.

Jones, M.A. and Eisner, J.E. 1992. A Probabilistic Parser Applied to Software Testing Documents. In *Proc. of the 10th National Conference on Artificial Intelligence*, 322-328. San Jose, CA: AAAI Press.

Kaplan, R. and Bresnan, J. Lexical-Functional Grammar: A Formal System for Grammatical Representation. In *The Mental Representation of Grammatical Relations*, 173-281. New York: John Wiley & Sons Inc.

Kelly, V. and Nonnenmann, U. Reducing the Complexity of Specification Acquisition in *Automating Software Design*, pp. 41-64, Menlo Park, AAAI Press.

Kelly, V. and Nonnenmann, U. Inferring Formal Software Specifications from Episodic Descriptions, in *Proc. of the Sixth National Conference on Artificial Intelligence*, pp. 127-132 Menlo Park, AAAI.

Nonnenmann, U., and Eddy J.K. 1992. KITSS - A Functional Software Testing System Using a Hybrid Domain Model. In *Proc. of 8th IEEE Conference on Artificial Intelligence Applications.* Monterey, CA: IEEE.

Zave, P. and Jackson, M. Techniques for Partial Specification and Specification of Switching Systems. In *VDM '91: Formal Software Development Methods (proceedings of the Fourth International Symposium of VDM Europe).* pp. 511-525 Springer Verlag ISBN 3-540-54834-3