Parallel Distributed Grammar Engineering for Practical Applications

Stephan Oepen[♠], Emily M. Bender[♠], Uli Callmeier[♡], Dan Flickinger[♠♡], Melanie Siegel[♣]

[♠]CSLI Stanford
Stanford (CA)

[♡]YY Technologies
Mountain View (CA)

[♣]DFKI GmbH
Saarbrücken (Germany)

{oe, bender, dan}@csli.stanford.edu

{uc, dan}@yy.com

siegel@dfki.de

Abstract

Based on a detailed case study of parallel grammar development distributed across two sites, we review some of the requirements for regression testing in grammar engineering, summarize our approach to systematic *competence and performance profiling*, and discuss our experience with grammar development for a commercial application. If possible, the workshop presentation will be organized around a software demonstration.

1 Background

The production of large-scale constraint-based grammars and suitable processing environments is a labour- and time-intensive process that, maybe, has become somewhat of a growth industry over the past few years, as companies explore products that incorporate grammar-based language processing. Many broad-coverage grammars have been developed over several years, sometimes decades, typically coordinated by a single grammarian who would often draw on additional contributors (e.g. the three HPSG implementations developed as part of the VerbMobil effort, see Flickinger, Copestake, & Sag, 2000, Müller & Kasper, 2000, and Siegel, 2000; or the LFG implementations developed within the ParGram consortium, Butt, King, Niño, & Segond, 1999).

More recently, we also find genuinely shared and distributed development of broad-coverage grammars, and we will use one such initiative as an example—viz. an open-source HPSG implementation for Japanese jointly developed between DFKI Saarbrücken (Germany) and YY Technologies (Mountain View, CA)—to demonstrate the technological and methodological challenges present in distributed grammar and system engineering.

2 Parallel Distributed Grammar Development—A Case Study

The Japanese grammar builds on earlier work performed jointly between DFKI and the Computational Linguistics Department at Saarland University (Germany) within VerbMobil; much like for the German VerbMobil grammar, two people were contributing to the grammar in parallel, one building out syntactic analyses, the other charged with integrating semantic composition into the syntax. This relatively strict separation of responsibilities mostly enabled grammarians to serialize incremental development of the resource: the syntactician would supply a grammar with extended coverage to the semanticist and, at the onset of the following iteration, start subsequent work on syntax from the revised grammar.

In the DFKI – YY cooperation the situation was quite different. Over a period of eight months, both partners had a grammarian working on syntax and semantics simultaneously on a day-to-day basis; both grammarians were submitting changes to a joint, version-controlled source repository and usually would start the work day by retrieving the most recent revisions. At the same time, product building and the development of so-called 'domain libraries' (structured collections of knowledge about a specific domain that is instantiated from semantic representations delivered from grammatical analysis) at YY already incorporated the grammar and depended on it for actual, customer-specific contracts. Due to a continuous demand for improvements in coverage and analysis accuracy, the grammar used in the main product line would be updated from the current development version about once or twice a week. Parallel to work on the Japanese grammar (and simultaneous work on grammars for English and Spanish), both the grammar development environment (the open-source LKB system; Copestake, 2002) and the HPSG run-time component powering the YY linguistic analysis engine (the open-source PET parser; Callmeier, 2002) continued to evolve, as did the YY-proprietary mapping of meaning representations extracted from the HPSG grammars into domain knowledge—all central parts of a complex system of interacting components and constraints.

As has been argued before (see, for example, Oepen & Flickinger, 1998), the nature of a large-scale constraint-based grammar and the subtle interactions of lexical and constructional constraints make it virtually impossible to predict how a change in one part of the grammar affects overall system behaviour. A relatively minor repair in one lexical class, numeral adjectives as in '*three* books were ordered' for instance, will have the potential of breaking the interaction of that class with the construction deriving named (numeric) entities from a numeral (e.g. as in '*three* is my favourite number') or the partitive construction (e.g. as in '*three* have arrived already'). A ripple effect of a single change can thus corrupt the semantics produced for any of these cases and in the consequence cause failure or incorrect behaviour in the back-end system. In addition to these quality assurance requirements on grammatical coverage and correctness, the YY application (like most applications for grammar-based linguistic analysis) utilizes a set of hand-constructed parse ranking heuristics that enables the parser to operate in best-first search mode and to return only one reading, i.e. the analysis that is ranked best by the heuristic component. The parse ranking machinery builds on preferences that are associated with individual or classes of lexical items and constructions. The set of preferences is maintained in parallel to the grammar, in a sense providing a layer of performance-oriented annotations over the basic building blocks of the core competence grammar. Without discussing the details of the parse ranking approach, it creates an additional element of uncertainty in assessing grammar changes: since the preference for a specific analysis results implicitly from a series of local preferences (of lexical items and constructions contributing to the complete derivation), introducing additional elements (i.e. new local or global ambiguity) into the search space and subjecting them to the partial ordering can quickly skew the overall result.

Summing up, the grammar and application engineering example presented here illustrates a number of highly typical requirements on the engineering environment. First, all grammarians and system engineers participating in the development process need to keep frequent, detailed, and accurate records of a large number of relevant parameters, including but not limited to grammatical coverage, correctness of syntactic analyses and corresponding semantic forms, parse selection accuracy, and overall system performance. Second, as modifications to the system as a whole are made daily—and sometimes several times each day—all developers must be able to assess the impact of recent changes and track their effects on all relevant parameters; gathering the data *and* analyzing it must be simple, fast, and automated as much as possible. Third, not all modifications (to the grammar or underlying software) will result in 'monotonic' or backwards-compatible effects. A change in the treatment of optional nominal complements, for example, may affect virtually all derivation trees and render a comparison of results at this level uninformative. At the same time, a primarily syntactic change of this nature will not cause an effect in associated meaning representations, so that a semantic equivalence test over analyses should be expected to yield an exact match to earlier results. Hence, the machinery for representation and comparison of relevant parameters needs to facilitate user-level specification of informative tests and evolution criteria. Finally, the metrics used in tracking grammar development cannot be isolated from measurements of system resource consumption and overall performance (specific properties of a grammar may trigger idiosyncrasies or software bugs in a particular version of the processing system); therefore, and to enable exchange of reference points and comparability of experiments, grammarians and system developers alike should use the same, homogenuous set of relevant parameters.

3 Integrated Competence and Performance Profiling

The integrated competence and performance profiling methodology and associated engineering platform, dubbed [incr tsdb()] (Oepen & Callmeier, 2000)[1] and reviewed in the remainder of this sec-

[1] See 'http://www.coli.uni-sb.de/itsdb/' for the (draft) [incr tsdb()] user manual, pronunciation rules, and instructions on obtaining and installing the package.

tion, was designed to meet all of the requirements identified in the DFKI – YY case study. Generally speaking, the [incr tsdb()] environment is an integrated package for diagnostics, evaluation, and benchmarking in practical grammar and system engineering. The toolkit implements an approach to grammar development and system optimization that builds on precise empirical data and systematic experimentation, as it has been advocated by, among others, Erbach & Uszkoreit (1990), Erbach (1991), and Carroll (1994). [incr tsdb()] has been integrated with, as of June 2002, nine different constraint-based grammar development and parsing systems (including both environments in use at YY, i.e. the LKB and PET), thus providing a pre-standard reference point for a relatively large (and growing) community of NLP developers. The [incr tsdb()] environment builds on the following components and modules:

- test and reference data stored with annotations in a structured database; annotations can range from minimal information (unique test item identifier, item origin, length et al.) to fine-grained linguistic classifications (e.g. regarding grammaticality and linguistic phenomena presented in an item), as they are represented in the TSNLP test suites, for example (Oepen, Netter, & Klein, 1997);
- tools to browse the available data, identify suitable subsets and feed them through the analysis component of processing systems like the LKB and PET, LiLFeS (Makino, Yoshida, Torisawa, & Tsujii, 1998), TRALE (Penn, 2000), PAGE (Uszkoreit et al., 1994), and others;
- the ability to gather a multitude of precise and fine-grained (grammar) competence and (system) performance measures—like the number of readings obtained per test item, various time and memory usage statistics, ambiguity and non-determinism metrics, and salient properties of the result structures—and store them in a uniform, platform-independent data format as a *competence and performance profile*; and
- graphical facilities to inspect the resulting profiles, analyze system competence (i.e. grammatical coverage and overgeneration) and performance (e.g. cpu time and memory usage, parser search space, constraint solver workload, and others) at variable granularities, aggregate, correlate, and visualize the data, and compare among profiles obtained from previous grammar or system versions or other processing environments.

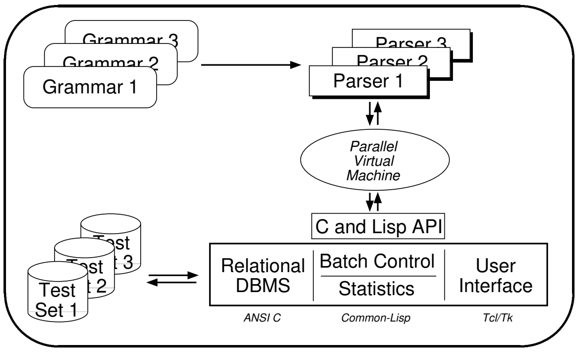

Figure 1: Rough sketch of [incr tsdb()] architecture: the core engine comprises the database management, batch control and statistics component, and the user interface.

As it is depicted in Figure 1, the [incr tsdb()] architecture can be broken down into three major parts: (i) the underlying database management system (DBMS), (ii) the batch control and statistics kernel (providing a C and Lisp application program interface to client systems that can be distributed across the network), and (iii) the graphical user interface (GUI). Although, historically, the DBMS was developed independently and the kernel can be operated without the GUI, the full functionality of the integrated competence and performance laboratory—as demonstrated below—only emerges from the combination of all three components. Likewise, the flexibility of a clearly defined API to client systems and its ability to parallelize batch processing and distribute test runs across the network have greatly contributed to the success of the package. The following paragraphs review some of the fundamental aspects in more detail, sketch essential functionality, and comment on how they have been exploited in the DFKI – YY cooperation.

Abstraction over Processors The [incr tsdb()] environment, by virtue of its generalized profile format, abstracts over specific processing environments. While grammar engineers in the DFKI – YY collaboration regularly use both the LKB (primarily for interactive development) and PET (mostly for batch testing and the assessment of results obtained in the YY production environment), usage of the [incr tsdb()] profile analysis routines in most aspects hides the specifics of the token processor used in obtaining a profile. Both platforms interprete the same typed feature structure formalism, load the same set of grammar source files, and (unless malfunctioning) produce equivalent results. Using [incr tsdb()], grammarians can obtain summary views of grammatical coverage and overgeneration, inspect relevant subsets of the available data, break down analysis views according to various aggregation schemes, and zoom in on specific aggregates or individual test items as appropriate. Moreover, processing results obtained from the (far more efficient) PET parser (that has no visualization or debugging support built in), once recorded as an [incr tsdb()] profile, can be used in conjunction with the LKB (contingent on the use of identical grammars), thereby facilitating graphical inspection of parse trees and semantic formulae.

Parallelization of Test Runs The [incr tsdb()] architecture (see Figure 1) separates the batch control and statistics kernel from what is referred to as client processors (i.e. parsing systems like the LKB or PET) through an application program interface (API) and the Parallel Virtual Machine (PVM; Geist, Bequelin, Dongarra, Manchek, & Sunderam, 1994) message-passing protocol layer. The use of PVM—in connection with task scheduling, error recovery, and roll-over facilities in the [incr tsdb()] kernel—enables developers to transparently parallelize and distribute execution of batch processing. At YY, grammarians had a cluster of networked Linux compute servers configured as a single PVM instance, so that execution of a test run—using the efficient PET run-time engine—could be completed as a matter of a few seconds. The combination of near-instantaneous profile creation and [incr tsdb()] facilities for quick, semi-automated assessment of relevant changes (see below) enabled developers to pursue a strongly empiricist style of grammar engineering, assessing changes and their effects on actual system behavior in small increments (often many times per hour).

Structured Comparison One of the facilities that has proven particularly useful in the distributed grammar engineering setup outlined in Section 2 above is the flexible comparison of competence and performance profiles. The [incr tsdb()] package eases comparison of results on a per-item basis, using an approach similar to Un∗x `diff(1)`, but generalized for structured data sets. By selection of a set of parameters for intersection (and optionally a comparison predicate), the user interface allows browsing the subset of test items (and associated results) that fail to match in the selected properties. One dimension that grammarians found especially useful in intersecting profiles is on the number of readings assigned per item—detecting where coverage was lost or added—and on derivation trees (bracketed structures labeled with rule names and identifiers of lexical items) associated with each parser analysis—assessing where analyses have changed. Additionally, using a user-supplied equivalence predicate, the same technique was regularly used at YY to track the evolution of meaning representations (as they form the interface from linguistic analysis into the back-end knowledge processing engine), both for all readings and the analysis ranked best by the parse selection heuristics.

Zooming and Interactive Debugging In analysing a new competence and performance profile, grammarians typically start from summary views (overall grammatical coverage, say), then single out relevant (or suspicious) subsets of profile data, and often end up zooming in to the level of individual test items. For most [incr tsdb()] analysis views the 'success' criteria can be varied according to user decisions: in assessing grammatical coverage, for example, the scoring function can refer to virtually arbitrary profile elements—ranging from the most basic coverage measure (assigning at least one reading) to more refined or application-specific metrics, the production of a well-formed meaning representation, say. Although the general approach allows output annotations on the test data (full or partial constituent structure descriptions, for example), developers so far have found the incremental, semi-automated comparison against earlier results a more adequate means of regression testing. It would appear that, especially in an application-driven and tightly scheduled engineering situation like the DFKI – YY partnership, the pace of evolution and general lack of locality in changes (see the examples discussed in Section 2) precludes the construction of a static, 'gold-standard' target for comparison. Instead, the structured comparison facilities of [incr tsdb()] enable developers to incrementally approximate target results and, even

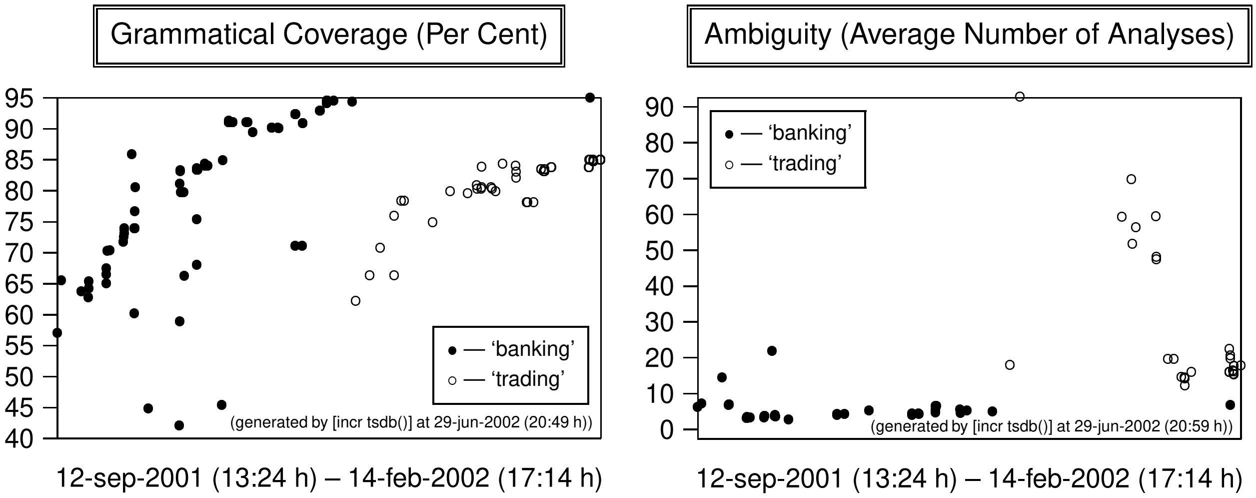

Figure 2: Evolution of grammatical coverage and average ambiguity (number of readings per test item) over a five-month period; '*banking*' and '*trading*' are two data sets (of some 700 and 400 sentences, respectively) of domain data.

in a highly dynamic environment where grammar and processing environment evolve in parallel, track changes and identify regression with great confidence.

4 Looking Back—Quantifying Evolution

Over time, the [incr tsdb()] profile storage accumulates precise data on the grammar development process. Figure 2 summarizes two aspects of grammatical evolution compiled over a five-month period (and representing some 130 profiles that grammarians put aside for future reference): grammatical coverage over two representative samples of customer data—one for an on-line banking application, the other from an electronic stock trading domain—is contrasted with the development of global ambiguity (i.e. the average number of analyses assinged to each test item). As should be expected, grammatical coverage on both data sets increases significantly as grammar development focuses on these domains ('*banking*' for the first three months, '*trading*' from there on). While the collection of available profiles, apparently, includes a number of data points corresponding to 'failed' experiments (fairly dramatic losses in coverage), the larger picture shows mostly monotonic improvement in coverage. As a control experiment, the coverage graph includes another data point for the '*banking*' data towards the end of the reporting period. Two months of focussed development on the '*trading*' domain have not negatively affected grammatical coverage on the data set used earlier. Corresponding to the (desirable) increase in coverage, the graph on the right of Figure 2 depicts the evolution of grammatical ambiguity. As hand-built linguistic grammars put great emphasis on the precision of grammatical analysis and the exclusion of ungrammatical input, the overall average of readings assigned to each sentence varies around relatively small numbers. For the moderately complex email data[2] the grammar often assigns less than ten analyses, rarely more than a few dozens. However, not surprisingly the addition of grammatical coverage comes with a sharp increase in ambiguity (which may indicate overgeneration): the graphs in Figure 2 clearly show that, once coverage on the '*trading*' data was above eighty per cent, grammarians shifted their engineering focus on 'tightening' the grammar, i.e. the elimination of spurious ambiguity and overgeneration (see Siegel & Bender, 2002, for details on the grammar).

Another view on grammar evolution is presented in Figure 3, depicting the 'size' of the Japanese grammar over the same five-month development cycle. Although measuring the size of

[2]Quantifying input complexity for Japanese is a nontrivial task, as the count of the number of input words would depend on the approach to string segmentation used in a specific system (the fairly aggressive tokenizer of ChaSen, Asahara & Matsumoto, 2000, in our case); to avoid potential for confusion, we report input complexity in the (overtly system-specific) number of lexical items stipulated by the grammar instead: around 50 and 80, on average, for the '*banking*' and '*trading*' data sets, respectively (as of February 2002).

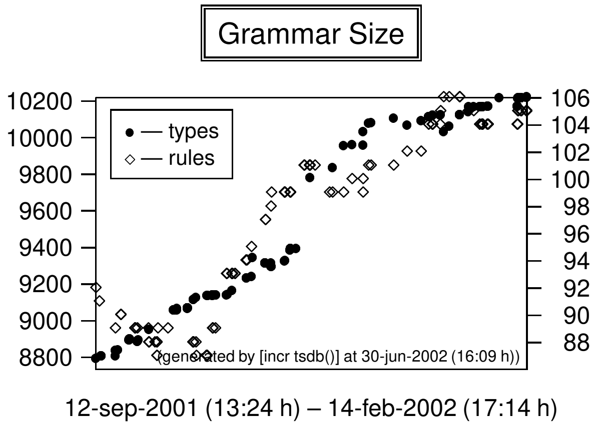

Figure 3: Evolution of grammar size (in the numbers of types, plotted against the left axis, and grammar rules, plotted against the right axis) over a five-month period.

computational grammars is a difficult challenge, for the HPSG framework two metrics suggest themselves: the number of types (i.e. the size of the grammatical ontology) and the number of grammar rules (i.e. the inventory of construction types). As would be expected, both numbers increase more or less monotonically over the reporting period, where the shift of focus from the '*banking*' into the '*trading*' domain is marked with a sharp increase in (primarily lexical) types. Contrasted to the significant gains in grammatical coverage (a relative improvement of more than seventy per cent on the '*banking*' data), the increase in grammar size is moderate, though: around fifteen and twenty per cent in the number of types and rules, respectively.

5 Conclusions

At YY and cooperating partners (primarily DFKI Saarbrücken and CSLI Stanford), grammarians (for all languages) as well as developers of both the grammar development tools and of the production system all used the competence and performance profiling environment as part of their daily engineering toolbox. The combination of [incr tsdb()] facilities to parallelize test run processing and a break-through in client system efficiency (using the PET parser; Callmeier, 2002) has created an experimental development environment where grammarians can obtain near-instantaneous feedback on the effects of changes they explore.

For the Japanese grammar specifically, the grammar developers at both ends would typically spend the first ten to twenty minutes of the day obtaining fresh profiles for a number of shared test sets and diagnostic corpora, thereby assessing the most recent set of changes through empirical analysis of their effects. In conjunction with a certain rigor in documentation and communication, it was the ability of both partners to regularly, quickly, and semi-automatically monitor the evolution of the joint resource with great confidence that has enabled truly parallel development of a single, shared HPSG grammar across continents. Within a relatively short time, the partners succeeded in adapting an existing grammar to a new genre (email rather than spoken language) and domain (customer service requests rather than appointment scheduling), greatly extending grammatical coverage (from initially around forty to above ninety per cent on representative customer corpora), and incorporating the grammar-based analysis engine into a commercial product. And even though in February 2002, for business reasons, YY decided to reorganize grammar development for Japanese, the distributed, parallel grammar development effort positively demonstrates that methodological and technological advances in constraint-based grammar engineering have enabled commercial development and deployment of broad-coverage HPSG implementations, a paradigm that until recently was often believed to still lack the maturity for real-world applications.

Acknowledgements

The DFKI – YY partnership involved a large group of people at both sites. We would like to thank Kirk Oatman, co-founder of YY and first CEO, and Hans Uszkoreit, Scientific Director at DFKI, for their initiative and whole-hearted support to the project; it takes vision for both corporate and academic types to jointly develop an open-source resource. Atsuko Shimada (from Saarland University), as part of a two-month internship at YY, has greatly contributed to the preparation of representative data samples, development of robust pre-processing rules, and extensions to lexical coverage. Our colleague and friend Asahara-san (of the Nara Advanced Institute of Technology, Japan), co-developer of the open-source ChaSen tokenizer and morphological analyzer for Japanese, was instrumental in the integration of ChaSen into the YY product and also helped a lot in adapting and (sometimes) fixing tokenization and morphology.

References

Asahara, M., & Matsumoto, Y. (2000). Extended models and tools for high-performance part-of-speech tagger. In *Proceedings of the 18th International Conference on Computational Linguistics* (pp. 21 – 27). Saarbrücken, Germany.

Butt, M., King, T. H., Niño, M.-E., & Segond, F. (1999). *A grammar writer's cookbook.* Stanford, CA: CSLI Publications.

Callmeier, U. (2002). Preprocessing and encoding techniques in PET. In S. Oepen, D. Flickinger, J. Tsujii, & H. Uszkoreit (Eds.), *Collaborative language engineering. A case study in efficient grammar-based processing.* Stanford, CA: CSLI Publications. (forthcoming)

Carroll, J. (1994). Relating complexity to practical performance in parsing with wide-coverage unification grammars. In *Proceedings of the 32nd Meeting of the Association for Computational Linguistics* (pp. 287 – 294). Las Cruces, NM.

Copestake, A. (2002). *Implementing typed feature structure grammars.* Stanford, CA: CSLI Publications.

Erbach, G. (1991). An environment for experimenting with parsing strategies. In J. Mylopoulos & R. Reiter (Eds.), *Proceedings of IJCAI 1991* (pp. 931 – 937). San Mateo, CA: Morgan Kaufmann Publishers.

Erbach, G., & Uszkoreit, H. (1990). *Grammar engineering. Problems and prospects* (CLAUS Report # 1). Saarbrücken, Germany: Computational Linguistics, Saarland University.

Flickinger, D., Copestake, A., & Sag, I. A. (2000). HPSG analysis of English. In W. Wahlster (Ed.), *Verbmobil. Foundations of speech-to-speech translation* (Artificial Intelligence ed., pp. 321 – 330). Berlin, Germany: Springer.

Geist, A., Bequelin, A., Dongarra, J., Manchek, W. J. R., & Sunderam, V. (Eds.). (1994). *PVM — parallel virtual machine. A users' guide and tutorial for networked parallel computing.* Cambridge, MA: The MIT Press.

Makino, T., Yoshida, M., Torisawa, K., & Tsujii, J. (1998). LiLFeS — towards a practical HPSG parser. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics* (pp. 807 – 11). Montreal, Canada.

Müller, S., & Kasper, W. (2000). HPSG analysis of German. In W. Wahlster (Ed.), *Verbmobil. Foundations of speech-to-speech translation* (Artificial Intelligence ed., pp. 238 – 253). Berlin, Germany: Springer.

Oepen, S., & Callmeier, U. (2000). Measure for measure: Parser cross-fertilization. Towards increased component comparability and exchange. In *Proceedings of the 6th International Workshop on Parsing Technologies* (pp. 183 – 194). Trento, Italy.

Oepen, S., & Flickinger, D. P. (1998). Towards systematic grammar profiling. Test suite technology ten years after. *Journal of Computer Speech and Language*, *12 (4) (Special Issue on Evaluation)*, 411 – 436.

Oepen, S., Netter, K., & Klein, J. (1997). TSNLP — Test Suites for Natural Language Processing. In J. Nerbonne (Ed.), *Linguistic Databases* (pp. 13 – 36). Stanford, CA: CSLI Publications.

Penn, G. (2000). Applying constraint handling rules to HPSG. In *Proceedings of the first international conference on computational logic* (pp. 51 – 68). London, UK.

Siegel, M. (2000). HPSG analysis of Japanese. In W. Wahlster (Ed.), *Verbmobil. Foundations of speech-to-speech translation* (Artificial Intelligence ed., pp. 265 – 280). Berlin, Germany: Springer.

Siegel, M., & Bender, E. M. (2002). Efficient deep processing of japanese. In *Proceedings of the 19th International Conference on Computational Linguistics.* Taipei, Taiwan.

Uszkoreit, H., Backofen, R., Busemann, S., Diagne, A. K., Hinkelman, E. A., Kasper, W., Kiefer, B., Krieger, H.-U., Netter, K., Neumann, G., Oepen, S., & Spackman, S. P. (1994). DISCO — an HPSG-based NLP system and its application for appointment scheduling. In *Proceedings of the 15th International Conference on Computational Linguistics.* Kyoto, Japan.