Chapter 6
SPRAAK: Speech Processing, Recognition and Automatic Annotation Kit

Patrick Wambacq, Kris Demuynck, and Dirk Van Compernolle

6.1 Introduction

Over the past years several users (in Belgium, the Netherlands and abroad) have adopted the ESAT speech recognition software package (developed for over 15 years at ESAT, K.U.Leuven, [5, 10]) as they found that it satisfied their research needs better than other available packages. However, typical of organically grown software, the learning effort was considerable and documentation lacking. The software needed a major overhaul and this is accomplished with support from the STEVIN programme and a partnership consisting of Katholieke Universiteit Leuven, Radboud University Nijmegen, Twente University and TNO. At the same time the main weaknesses were addressed and the code base was modernised. This effort also addressed the need of the research community for a Dutch speech recognition system that can be used by non-specialists. We also found this to be an ideal moment to open up the code to the community at large. It is now distributed as open source for academic usage and at moderate cost for commercial exploitation. The toolkit, its documentation and references can be found at http://www.spraak.org.

In this article details of the SPRAAK toolkit are given in several sections: Sect. 6.2 discusses the possible uses of the toolkit, Sect. 6.3 explains the features of the different components of the software, some benchmark results are given in Sect. 6.4, system requirements of the software are mentioned in Sect. 6.5, licensing and distribution is covered in Sect. 6.6, relation to other STEVIN projects in Sects. 6.7 and 6.8 addresses future work. Finally a conclusion is given in Sect. 6.9.

This article tries to give as complete information as possible about SPRAAK and is therefore targeted at several audiences:

P. Wambacq (✉) · K. Demuynck · D. Van Compernolle
Katholieke Universiteit Leuven, ESAT, Kasteelpark Arenberg 10 box 2441,
B3001 Heverlee, Belgium
e-mail: patrick.wambacq@esat.kuleuven.be; kris.demuynck@esat.kuleuven.be;
dirk.vancompernolle@esat.kuleuven.be

P. Spyns and J. Odijk (eds.), *Essential Speech and Language Technology for Dutch*,
Theory and Applications of Natural Language Processing,
DOI 10.1007/978-3-642-30910-6_6,

- The speech and language technology community at large. These readers will find most useful information in Sects. 6.2 (introduction and Sect. 6.2.1), 6.3.1, 6.4 and 6.7;
- Potential non-expert users, who could additionally be interested in the other parts of Sect. 6.2, and in Sects. 6.5, 6.6 and 6.8;
- Speech recognition researchers who are also served with the technical details in Sect. 6.3.

6.1.1 Alternatives to SPRAAK

When the SPRAAK project was conceived, a survey of open source alternatives was made. There exist other alternatives that are either commercial or limited to internal use. Commercial systems were not considered because of their price, issues with research licenses or unavailability of the source code. At the time of the start of the project the most important competitors were the following:

- The freely available version of the well-known HTK toolkit [14] which has a less capable decoder which is limited to bigram grammars and is not fit for real-time applications. The code also has a complex global structure which makes it difficult to change.
- The CMU Sphinx4 system [3] which does not have state-of-the-art performance. The choice to implement the system in Java provides excellent flexibility, but is detrimental to the memory footprint of the system and limits the decoding speed.
- Julius [16] which is a fairly successful Japanese recogniser but the documentation (at that time only in Japanese) and language barrier in general is however an important issue.

We believe that the above mentioned obstacles have not yet been removed, except that now English documentation is being written for Julius. Since the start of the SPRAAK project, another open source initiative has seen the light in 2009: the RWTH ASR toolkit [20]. We don't have any experience with this software but from the documentation available on the website, features seem to be comparable to ours (with some extras but also some shortcomings).

6.2 Intended Use Scenarios of the SPRAAK Toolkit

SPRAAK is a speech recognition toolkit intended for two distinct groups of users:

- SPRAAK is designed to be a flexible modular toolkit for speech recognition research, yet at the same time
- SPRAAK provides a state-of-the art speech recogniser suitable for speech application deployment in niche markets.

In order to address the different needs, functionalities in SPRAAK may be addressed by a set of different interfaces. Application developers need control over

the run-time engine and over the resources that the engine uses. They are served by the High Level API (Application Programmers Interface). Developing new resources may imply the usage of and minor adaptations to a number of standalone scripts that make use of the lower level API.

Speech researchers will be served by the scripting framework for training and evaluating new systems. Certain types of speech research (e.g. feature extraction) may involve only limited programming, while others (e.g. changes to the decoder) are always a major programming effort. Depending on the type of research they may need to dig deeper into the internals for which they will make use of the low level API.

A speech recognition system is a piece of complex software and although an inexperienced user does not have know its inner workings, some basic knowledge is required to be able to use it. To this end we have organised hands-on seminars, and this material was then later reused to produce new and better tutorials that are now part of the documentation. Given enough interest, more seminars can be organised in the future.

6.2.1 SPRAAK as a Research Toolkit

SPRAAK is a flexible modular toolkit for research into speech recognition algorithms, allowing researchers to focus on one particular aspect of speech recognition technology without needing to worry about the details of the other components. To address the needs of the research community SPRAAK provides:

- Plug and play replacement of the core components
- Flexible configuration of the core components
- Extensive libraries covering the common operations
- Examples on how to add functionality
- Well defined and implemented core components and interaction schemes
- Programmers and developers documentation
- A low level API
- A direct interface with the Python scripting language
- An interactive environment (Python) for speech research
- Scripts for batch processing

6.2.2 SPRAAK for Application Development

SPRAAK is also a state-of-the art recogniser with a programming interface that can be used by non-specialists with a minimum of programming requirements. At the same time SPRAAK allows ASR specialists to integrate and harness special functionality needed for niche market projects that are not served by the existing

off-the-shelf commercial packages. To address the needs of this part of the user base, SPRAAK provides:

- Users documentation;
- A high level API;
- A client-server model;
- Standard scripts for developing new resources such as acoustic models.
- Demo models can be obtained from the Dutch-Flemish HLT Agency; these can serve as starting points for a recogniser and to train own models. This includes models for Northern and Southern Dutch for both broadband and telephony speech and a set of resources (acoustic and language models, lexica, . . .);
- On request the developers can also provide a set of reference implementations or frameworks for different applications.

6.2.3 SPRAAK Applications

Although it is possible to build conventional ASR applications for widely spoken languages with SPRAAK, the aim of the software is not to compete against well-known commercial packages with a large installed base. Rather we want to allow companies to build and implement ASR based solutions for niche markets. Example applications include but are not limited to the following:

- Speech transcription/alignment (“conventional ASR applications”) for low resourced or under-resourced languages, e.g. small languages in both developed countries (e.g. Frisian) and developing countries (e.g. Afrikaans);
- Speech segmentation, e.g. for speech corpus development;
- Subtitling;
- Medical applications e.g. speech evaluation for patients with speech disorders;
- Educational applications e.g. reading tutors, pronunciation training;
- Phonetic labeling in phonetics research;
- Speaker recognition.

For some of the listed tasks a set of scripts are provided. We expect the number of available scripts to grow rapidly over time as the user base of SPRAAK grows.

6.2.4 High Level API for the Run-Time Engine

The high level API is the convenient interface to communicate with the SPRAAK run-time engine. It allows for all necessary functionality: start and stop, setting of parameters, loading and switching of acoustic and language model and control over the audio device. This level of control suits the needs of people interested

in dialog development, interactive recogniser behaviour, speech recognition testing from large corpora, interactive testing, building demonstrators, etc.

The high level API is available in two forms: C-callable routines and a corresponding set of commands for a client-server architecture. There is a one-to-one mapping between these two forms.

In the client-server framework, client (application) and server (speech recognition engine) communicate with each other via a simple physical interface using the high level API commands. Feedback from the engine to the application is both client driven and server driven. All concepts known to the high level API may also be specified in an initialisation file that is loaded when the engine is started.

The alternative form of the high level API uses C-callable routines instead of the pipe/socket interface layer. Except for giving up the flexibility of the client/server concept, it has exactly the same functionality. This is the most compact and efficient implementation and would be the implementation of choice for (semi)-commercial applications on a standalone platform.

The high level API gives the user an intuitive and simple access to a speech recognition engine, while maintaining reasonably detailed control. The concepts defined in the high level API are concepts known to the speech recognition engine such as language models, acoustic models, lexica, pre-processing blocks and search engines. In that sense the SPRAAK high level API may not be as high level as in certain commercial packages which tend to work with concepts such as ‘user’, ‘language’, ‘application’. If a translation is necessary from intuitive user concepts to the speech recognition concepts then this is the task of the application, although SPRAAK provides the necessary hooks to work with these high level concepts.

Usage of the high level API should be suitable for users with a moderate understanding of speech recognition concepts and requires only a minimum of computational skills.

6.2.5 Low Level API

The low level API provides full access to all routines, including those that are not relevant for the run-time engine. In order to understand what is possible through this low level API it is necessary to have an understanding of the software architecture of the SPRAAK toolkit. In essence, the SPRAAK toolkit consists of a set of programs, modules (compiled code) and scripts. The modules are designed according to object oriented concepts and are written in C. Python is used as scripting language.

Standalone tools such as alignment, enrolment, operations on acoustic models, training of acoustic models are implemented in Python scripts that make use of the low level API. SPRAAK includes example scripts for most of these tasks, and the number of available scripts will grow over time.

Usage of the low level API gives the user full control over the internals. It is the vehicle for the ASR researcher who wants to write new training scripts, modify the recogniser, etc. New concepts may first be prototyped in Python. As

long as it doesn't affect the lowest level computational loops in the system the impact on efficiency will be limited. Nevertheless the inherent delays at startup time and the loss of efficiency in the scripting language make this setup less suited for applications.

The low level API is also the ideal tool for ASR teaching. It provides detailed enough insight into the speech recognition internals, it allows for visualisation of intermediate results and makes it possible to modify specific pieces of the code without the need to develop a full recognition system.

Usage of the low level API is intended for those who have a good understanding of speech recognition internals and who are at least skilful programmers at the script level.

6.2.6 SPRAAK Developers API

The SPRAAK developers API is mainly relevant to the developers of the SPRAAK toolkit. It handles low-level tasks such as parsing input, argument decoding, the object oriented layer, debug messages and asserts, ...

For a number of reasons, not all of the functionality of the developers API is made available on the Python level:

- Python provides its own alternatives (parsing, hash tables, the object oriented layer);
- The functionality is not relevant at the Python level (atomic operations);
- Exposing the functionality would be difficult and with little merit.

Routines that are part of the developers API only are tailor made to operate within the SPRAAK library and are (in theory) not fit for use outside this scope. Since these routines are only meant for internal use, their interface can also be changed without prior notice.

Extending and modifying the developers API is intended only for those who have a good understanding of the SPRAAK framework and who are skilful C-programmers.

6.3 Features of the SPRAAK Toolkit

6.3.1 Overview

Figure 6.1 shows the main components of the SPRAAK recogniser in a typical configuration and with the default interaction between these components.

In a large vocabulary speech recognition application typically all components will be activated. When using the SPRAAK engine for other tasks

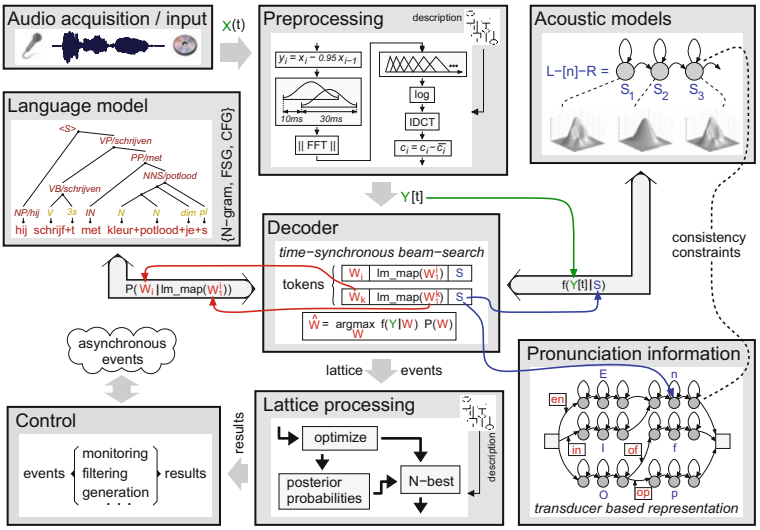

Fig. 6.1 Main components of the SPRAAK recogniser

(e.g. preprocessing, alignment, training of acoustic models, ...), only a subset of the same main components may be used. For highly experimental setups more complex configurations and interactions are possible.

The components shown in Fig. 6.1 are:

- The preprocessing;
- The acoustic model (AM);
- The lexicon containing pronunciation information, organised as a network/finite state transducer;
- The language model (LM);
- The decoder which connects all major components together;
- A lattice generation and processing block.

A detailed description of the underlying software implementation may be found in the developer's manual. Here we give a brief functional description of each of these modules.

6.3.2 The Preprocessing

Speech recognisers do not work directly on sampled data. SPRAAK as the great majority of speech recognition systems works with frames, i.e. features extracted from the data at regularly spaced time intervals.

SPRAAK has a wealth of signal processing routines and options built in. The algorithm to be executed is described by a scripting language in a signal processing

configuration file. It may be called ‘on-line’, i.e. processing is done as soon as input becomes available and the computed features are streamed towards further processing steps in an automatic way. This can be done both during training and recognition. This is the standard setup for most types of signal processing which require only minimal amounts of computing time on today’s systems. Processing may also be done ‘off-line’ in which the signal processing output is written to file. The latter may be the preferred mode of operation for complex signal processing algorithms.

A complete list of all signal processing modules can be found in the documentation. The implemented modules include standard speech processing blocks such as FFT, cepstra, mel-cepstra, LPC and PLP analysis, mean normalisation, histogram equalisation, vocal tract length normalisation, begin- and endpoint detection, time derivatives and silence/speech detection. The flexibility of the preprocessing is further extended with generic blocks such as linear transformations, linear filters, vector based function evaluation, frame stacking and order statistics. Support for neural network processing and gaussian mixture evaluation is also available.

Important properties and constraints of the preprocessing module are:

- Non-causal behaviour is supported by allowing the process routine to backhold data whenever deemed necessary;
- All preprocessing can be done both on-line and off-line;
- The fact that everything is available on-line is very handy, but requires some programming effort when writing new modules since everything has to be written to work on streaming data;
- Currently, only a single frame clock is supported. Changing the frame-rate (dynamically or statically) is not supported;
- Frames must be processed in order and are returned in order.

6.3.3 The Acoustic Model

The acoustic model calculates observation likelihoods for the Hidden Markov (HMM) states. These likelihoods can be provided by Gaussian mixture models, neural networks, discrete distributions, or can even be read from file. When using Gaussian mixture models, SPRAAK offers the unique possibility to share gaussians between states, on top of the traditional state tying. In our experience most tasks are served best with this unique modelling technique. In this setup each distribution in the acoustic model is computed as a weighted sum of gaussians drawn from a large pool. It is used as a generic approach that allows untied and fully tied mixtures as extremes of its implementation. Features are:

- Mixture gaussian densities with full sharing;
- Fast evaluation for tied Gaussians by data-driven pruning based on ‘Fast Removal of Gaussians’ (FRoG) [6];

- Model topology is decoupled from observation likelihoods, allowing for any number of states in any subword unit;
- Dedicated modules for initialising and updating the acoustic models (training and/or speaker adaptation);
- Access to all components of the acoustic model (the Gaussian set, FRoG, . . .);
- Implementations for discrete density and neural net models.

6.3.3.1 Tied Gaussian Mixtures

In continuous HMMs the model of the observation probabilities is organised in two layers, consisting of:

- A set of basis functions (typically multivariate gaussians) that are shared over all states;
- A set of weights (typically probabilities) linking the basis functions to the states.

The number of basis functions tends to be very large (several thousands), while only a few of them (typically around 100 in SPRAAK) will have non-zero weights for any individual state. Given this completely constraint-free tying scheme, other variants of tying can be accomodated: untied gaussians in which gaussians are private to a state, and fully tied gaussians in which for each state a weight is assigned to each gaussian. In this type of modelling the weight matrix linking basis functions to states tends to be very sparse (i.e. most weights are '0'). SPRAAK accommodates a flexible sparse data structure to cope with this efficiently.

The acoustic model returns normalised scores for a segment of data. By default the Viterbi algorithm is used, but other computational approaches and other acoustic models (than the standard HMM) are possible.

'Normalised' implies that the sum of scores over all possible states for any given time instant will be equal to 1.0. In practice log10-probabilities are returned. The motivation for using these normalised scores is:

- Normalised scores are much easier to interpret than unnormalised scores, whose values tend to fluctuate a lot under varying external conditions such as background noise, speaker identity, . . .
- These normalised scores are not posteriors, however, as the prior state probabilities are used and not the current state probabilities (which are expensive to compute);
- The normalisation is identical for all hypothesis, hence it does not affect the ranking of different hypotheses; nor does it affect the outcome of training;
- These normalised scores are a convenient input for the computation of confidence scores.

SPRAAK uses an efficient bottom up scheme to predict which gaussians in the pool need to be evaluated and which ones not ("Fast Removal Of Gaussians" – FRoG). This is done for the whole pool of gaussians at once and not on a state by

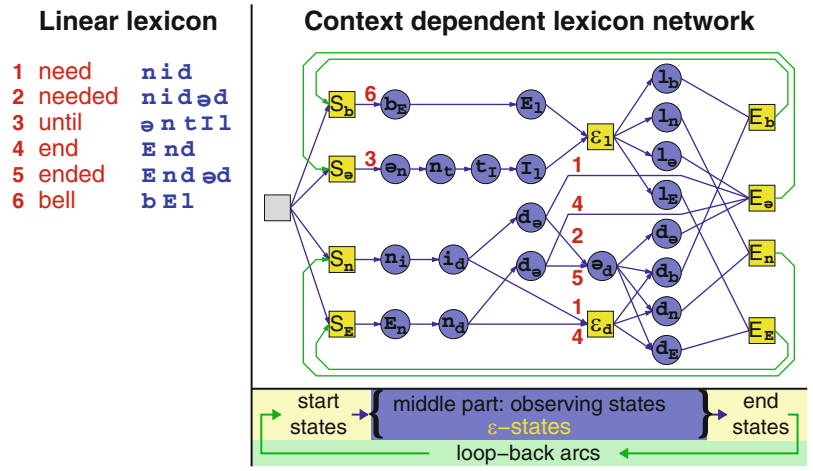

Fig. 6.2 Lexical network

state basis. The data structure describing for each axis which regions are relevant for which gaussians is computed once when the acoustic model is loaded.

6.3.4 The Lexicon and Pronunciation Network

The lexicon is stored as a pronunciation network in a (possibly cyclic) finite state transducer (FST) that goes from HMM states (input symbols of the FST) to some higher level (the output symbols of the FST). Typically, the output symbols are words. Having phones as output symbols is possible and results in a phone recogniser.

Apart from (word) pronunciations as such, this network can also encode assimilation rules and may use context dependent phones as learned by the acoustic model.

The same network may also be used to implement the constraints imposed by an FST-based language model. For example when training the acoustic models, the sentence being spoken (a linear sequence of words) is directly encoded in the pronunciation network, eliminating the need for a language model component.

Figure 6.2 gives an example of a pronunciation network using right context dependent phones and no assimilation rules. The final pronunciation network used by the decoder will also incorporate the tied-state information coming from the acoustic model. In order to not obscure the figure, this information was left out in the example.

The pronunciation network deviates from normal FST's in several ways:

- The input symbols are attached to the states, not the arcs;
- The network also contains some non-observing states: end-states, start-states and ϵ-states (non emitting states that allow more compact networks, cf. Fig. 6.2);

- The network contains two levels of output symbols: the phone identities and the items known by the language model.

6.3.5 The Language Model

The language model (LM) calculates conditional word probabilities, i.e. the probability of a new word given its predecessor words. For efficiency reasons the LM condenses all relevant information concerning the word predecessors in its own state variable(s).

Supported LM's and interfaces are:

- A word based N-gram which has low memory footprint and is fast. SPRAAK does not have its own tools for computing statistical language models but interfaces well with models generated with well established open source toolkits such as the SRI or CMU Language Model toolkits [2, 21, 22]. The language models produced by the mentioned toolkits come in the '.arpabo'-format (N-grams with backoff). In SPRAAK this format is converted to a compact binary format for efficiency reasons.
- A finite state grammar/transducer (FSG/FST). The FSG supports on the fly composition and stacking. The FSG has also provisions to behave exactly as an N-gram (correct fallback) and thus can replace the N-gram in situations where on-line changes to the LM are needed.

Furthermore an extension layer on top of these LM's allows for various extensions:

- Making class based LM's;
- Adding new words that behave similarly to existing words;
- Allowing multiple sentences to be uttered in one go;
- Adding filler words;
- Adding sub-models, e.g. a phone loop model to model the out-of-vocabulary words.

6.3.6 The Decoder

The decoder (search engine) finds the best path through the search space defined by the acoustic model, the language model and the pronunciation network given the acoustic data coming from the preprocessing block.

SPRAAK implements an efficient all-in-one decoder with as main features:

- Breadth-first frame synchronous;
- Allows cross-word context-dependent tied-state phones, multiple pronunciations per word, assimilation rules, and any language model that can be written in a left-to-right conditional form;

- Exact, i.e. no approximations whatsoever are used during the decoding, except for the applied pruning;
- Pruning: a threshold is used to discard hypotheses whose score falls a certain amount below the score of the most likely hypothesis; if most hypotheses have a similar score, a beam width parameter is used to keep only the best hypotheses;
- Provides as well the single best output as (word) lattices. Both outputs can be generated on-the-fly and with low latency;
- The backtracking can be instructed to keep track of the underlying phone or state sequences and to add them to the recognised word string and/or store them alongside the (word) lattice;
- The speed of the decoder is very high, e.g. depending on the complexity, a single pass in the model training is one or several orders of magnitude faster than realtime on a modern machine.

6.3.7 (Word) Lattice (Post)-processing

The (word) lattice (post)-processing consists, similar to the preprocessing, of a fully configurable processing flow-chart and a large set of processing blocks. The lattice processing can be fed either with stored lattices or can be coupled directly to the decoder using the built-in lattice generator.

The most important properties of the lattice processing component are:

- A low-latency data driven design suitable for use in real-time applications;
- Lattices that contain only acoustic model scores;
- Weak consistency checks when re-scoring for speed reasons (this may result in crashes if inconsistent knowledge sources are applied).

Among the available processing modules are:

- Lattice re-scoring (new LM) using a pseudo frame synchronous (breadth-first) decoder; this decoder is low-latency;
- The FLaVoR-decoder [8]. This decoder goes from phone lattices to word lattices using a fast and robust decoder. Its primary goal is to allow the use of advanced linguistic models (morpho-phonology, morpho-syntax, semantics, . . .);
- Calculating posterior probabilities given the (word) lattice and some finite state based language model. This is also the module that can be used to introduce language model constraints (and scores) in the lattice;
- Searching the best fit between two lattices, i.e. finding the best path through an input lattice given a reference lattice and a substitution matrix;
- A check module that verifies whether the input lattice adheres to requirements set forth by the different processing blocks;
- Input and output modules;
- Several optimisation modules;

- Lattices can also be converted to the HTK Standard Lattice Format (SLF), hence allowing the use of other external toolkits for operations such as N-best lists, confidence scoring via word posteriors, consensus networks, . . .

6.3.8 Extension to a Missing Data Theory Based Recogniser

In order to improve its robustness to noise and possibly reverberation, SPRAAK has been extended to include a Missing Data Theory (MDT) algorithm. This is the result of the STEVIN MIDAS project – cf. Chap. 16, p. 289. It does NOT improve robustness to pronunciation variation or accents.

The implementation is largely based on [27]. MDT requires an additional component, a mask estimator, which indicates which data is reliable enough for the recogniser to use and which data is unreliable (and will be reconstructed). Its output is a mask, a vector of variables that reveal to which extent each frequency channel is reliable (either binary or as a continuous variable between 0 and 1). Reliable data is merely copied from the input, while unreliable data is reconstructed or *imputed* based on the clean speech model. In the SPRAAK implementation, this imputation is guided by the search process, i.e. the missing data is reconstructed for each assumption the back-end makes about the speech identity. In this context, assumption means the visited state and selected Gaussian from its mixture. The missing data is reconstructed based on the maximum likelihood principle, i.e. it looks for the most likely values of the unreliable spectral components. This boils down to a constrained optimisation process that is solved with a gradient descent method, which is computationally efficient and requires only a few iterations to converge to a sufficiently accurate solution.

To use this MDT extension, the user needs to be aware of the following:

- It is assumed that the speech signal is disturbed by additive possibly non-stationary background noise. Significant reverberation cannot be handled, though mask estimation methods to handle reverberated speech are described in the literature [18, 19]. If the user wants robustness against reverberation, he will need to implement his own mask estimation technique.
- Tests on noisy data have shown that the MDT recogniser runs at about the same speed as when SPRAAK runs a standard MIDA front-end [7], but at a higher accuracy. Notice that noisy data requires longer decoding times due to the increased ambiguity.
- The MDT recogniser is implemented as an extension: the user needs to provide a mask estimator and additional data components, but the existing acoustic model, language model, lexicon, . . . remain unchanged.

6.4 SPRAAK Performance

During its history, SPRAAK and its predecessor have been used in many projects and for many tasks. It has also been benchmarked with internationally used and well-known reference tasks. Benchmark results achieved with SPRAAK include:

- TIDigits: 0.17 % WER using variable-sized word models (11–17 states).
- WSJ 5k closed vocabulary, speaker independent, WSJ-0 acoustic training data, bigram LM: 4.8 % WER on the nov92 test set (this is our AURORA-4 clean speech reference). When using more acoustic data (WSJ-1) and a trigram, a WER of 1.8 % can be achieved [24].
- WSJ 20k open vocabulary (1.9 % OOV rate), speaker independent, WSJ-1 acoustic training data, trigram: 7.3 % WER on the nov92 test set. This task runs in real time [11].
- Switchboard spontaneous telephone conversations, 310 h of acoustic training data, 3M words for training of the LM, open vocabulary: 29 % WER on the 2001 HUB5 benchmark [23]. This result was obtained with a single decoding pass and without any speaker adaptation, running at $4 \times$ RT on a pentium4 2.8 GHz. A similar real-time configuration gives a WER of 31 %.
- Dutch NBest Benchmark: 20.3 % WER on the Flemish Broadcast News test set [9].

Note that performances depend not only on the software but also very much on the models. A fair comparison of SPRAAK to other speech recognizers is therefore not easy if one wants to know the influence of the software only and not of the difference in the models. Processing times are also very volatile since hardware speed increases continuously and results on processing time reported some time ago cannot be compared to recent results. Therefore we opted not to try to include results from other systems.

On machines with multi-core processors, training of acoustic models can be executed in parallel, allowing a faster turn-around time when developing new models and applications. The goal of the SPRAAK developers is to extend the amount of code that can take profit from parallel hardware.

6.5 SPRAAK Requirements

SPRAAK is developed on the Linux platform and therefore all components are available on Unix(-like) operating systems, including Mac OS-X. On Windows XP or higher, a run-time engine and most standalone programs are available (though with some limitations); automatic parallelisation of some scripts and certain advanced functionalities (distributed computing and resource sharing, mainly relevant for resource development) are not available at this time.

Essential tools for installing and running SPRAAK include:

- A C99 compliant compiler for compilation (gcc recommended);
- Python (version 2.5 or higher) for installation and at run-time;
- Scons (version 0.98 or higher) for software installation.

For extended functionality and to build the documentation, additional applications, libraries and header files are useful or needed, as explained on the SPRAAK website http://www.spraak.org.

6.6 SPRAAK Licensing and Distribution

At the conception of the SPRAAK project it was found that the major overhaul of the old existing code was a great opportunity to at the same time open up the code to the community at large. It was decided to make the software an open source package, free for use for academic purposes. Therefore an academic license is available that is free of charge and available to research institutes or universities upon request. Before giving access to the software, a duly signed license agreement is required, which can be generated by registering on SPRAAK's website http://www.spraak.org/. A license key and instructions are sent out after reception of the agreement.

For companies, an evaluation license is available at low cost, that allows the evaluation of the software during a period of 1 year. A license for commercial use is also available. The aim of the software is not to compete against well-known commercial packages with a large installed base but rather to allow companies to build and implement ASR based solutions for niche markets. Revenues from commercial contracts are reserved for future upgrades of the SPRAAK software.

6.7 SPRAAK in the STEVIN Programme

As already mentioned, it was stated at the conception of the STEVIN programme that the availability of a speech recognition system for Dutch was one of the essential requirements for the language and speech technology community. Now that SPRAAK is available and that the STEVIN programme has ended we can verify how the software was put to use in other STEVIN projects. This is a list of projects that have some relation to SPRAAK:

- JASMIN-CGN: in this corpus project, SPRAAK and its predecessor HMM75 have generated phonetic annotations and word segmentations – cf. [4] and Chap. 3.
- MIDAS (Missing Data Solutions): this project tackles the noise robustness problem in ASR through missing data techniques (MDT). The results of the project have been integrated in the software – cf. Sect. 6.3.8, Chap. 16 and [13].
- NEON (Nederlandstalige Ondertiteling): this project evaluates the use of ASR in subtitling applications. SPRAAK was used as the ASR component – cf. [28].

- DIADEMO: this project has built a demonstrator that recognizes spoken dialects. SPRAAK is used as the ASR component. The algorithm is explained in [29].
- DISCO (Development and Integration of Speech technology into COurseware for language learning): this present project has developed and tested a prototype of an ASR-based CALL application for training oral proficiency for Dutch as a second language. SPRAAK was used as the ASR component – cf. [26] and Chap. 18.
- AAP (Alfabetisering Anderstaligen Plan): a demonstrator was built that uses speech technology to combat illiteracy for second language learners of Dutch – cf. [25].
- HATCI (Hulp bij Auditieve Training na Cochleaire Implantatie): this project uses automatic speech assessment to build an application to support rehabilitation therapy after cochlear implantation. SPRAAK was used for the speech assessment component – cf. [17].
- N-Best: this project has organised and executed an evaluation of large vocabulary speech recognition systems trained for Dutch (both Northern and Southern Dutch) in two evaluation conditions (Broadcast News and Conversational Telephony Speech). Two submissions to the evaluation have used SPRAAK to generate their results – cf. [9, 15] and Chap. 15.

6.8 Future Work

A package like SPRAAK is of course never complete. The aim is however to follow the main trends in current ASR research and to implement interesting new algorithms as they become available. The software also still lacks some methods that have been in existence since some time, such as more extensive speaker adaptation than the currently available Vocal Tract Length Normalisation (VTLN). The aim is to include to that end CMLLR (Constrained/Feature Space MLLR [1, 12]) in the short term. Also the following will be added:

- A high-level flow-chart scripting language to configure multi-stage recognisers, parallel recognisers with the available processing blocks;
- Further parallelisation of the code;
- Posterior probabilities on networks: this allows to build consensus networks where a consensus output is derived by selecting the word sequence with the best score, where scores can be formed in many ways such as by voting or using posterior probability estimates. Some external toolkits exist to produce these, starting from our lattices but they may fail for large lattices or with complex language models.

User contributions are also very welcome. The academic license terms specify that users who implement “modifications” (that change, improve or optimize an existing functionality) must make these available to the SPRAAK developers, and users who implement “new modules” (adding new functionality) must inform the developers

about them and make them available at reasonable conditions under the form of a non-exclusive license. This mechanism will hopefully contribute to the capabilities of the software and strengthen of the user base.

6.9 Conclusions

The STEVIN SPRAAK project has addressed one of the essential requirements for the language and speech technology community: the availability of a speech recognition system for Dutch. Researchers can use the highly modular toolkit for research into speech recognition algorithms (language independent and thus not only for Dutch). It allows them to focus on one particular aspect of speech recognition technology without needing to worry about the details of the other components. Besides that a state-of-the art recogniser for Dutch with a simple interface is now available, so that it can be used by non-specialists with a minimum of programming requirements as well. Next to speech recognition, the resulting software also enables applications in related fields. Examples are linguistic and phonetic research where the software can be used to segment large speech databases or to provide high quality automatic transcriptions. The existing ESAT recogniser was augmented with knowledge and code from the other partners in the project, as a starting point, and this code base was transformed to meet the specified requirements. The transformation was accomplished by improving the software interfaces to make the software package more user friendly and adapted for usage in a large user community, and by providing adequate user and developer documentation written in English, so as to make it easily accessible to the international language and speech technology community as well. The software is open source and freely available for research purposes. Details, documentation and references can be found at http://www.spraak.org/. Example recognizers, demo acoustic models for Dutch and their training scripts can be obtained from the Dutch-Flemish HLT Central.

Acknowledgements The SPRAAK toolkit has emerged as the transformation of KULeuven/E-SAT's previous speech recognition system (HMM75 was the latest version). This system is the result of 20 years of research and development in speech recognition at the KULeuven. We want to acknowledge the contributions of many researchers (too many to list them all) most of which have by now left the university after having obtained their PhD degree.

The SPRAAK project has also added some extra functionality to the previous system and has produced demo acoustic models and documentation. This was done in a collaborative effort with four partners: KULeuven/ESAT, RUNijmegen/CSLT, UTwente/HMI and TNO. The contributions of all these partners are also acknowledged.

References

1. Afify, M., Siohan, O.: Constrained maximum likelihood linear regression for speaker adaptation. In: Proceedings of the ICSLP-2000, Beijing, China, vol.3, pp. 861–864 (2000)
2. Clarkson, P., Rosenfeld, R.: Statistical language modeling using the CMU-Cambridge toolkit. In: Proceedings of the ESCA Eurospeech 1997, Rhodes, Greece, pp. 2707–2710 (1997)
3. CMUSphinx wiki available at. http://cmusphinx.sourceforge.net/wiki/start
4. Cucchiarini, C., Driesen, J., Van hamme, H., Sanders, E.: Recording speech of children, non-natives and elderly people for HLT applications: the JASMIN-CGN corpus. In: Proceedings of the 6th International Conference on Language Resources and Evaluation – LREC 2008, Marrakech, Morocco, 28–30 May 2008, p. 8
5. Demuynck, K.: Extracting, modelling and combining information in speech recognition. Ph.D. thesis, K.U.Leuven ESAT (2001)
6. Demuynck, K., Duchateau, J., Van Compernolle, D.: Reduced semi-continuous models for large vocabulary continuous speech recognition in Dutch. In: Proceedings of the ICSLP, Philadelphia, USA, Oct 1996, vol. IV, pp. 2289–2292
7. Demuynck, K., Duchateau, J., Van Compernolle, D.: Optimal feature sub-space selection based on discriminant analysis. In: Proceedings of the European Conference on Speech Communication and Technology, Budapest, Hungary, vol. 3, pp. 1311–1314 (1999)
8. Demuynck, K., Laureys, T., Van Compernolle, D., Van hamme, H.: FLaVoR: a flexible architecture for LVCSR. In: Proceedings of the European Conference on Speech Communication and Technology, Geneva, Switzerland, Sept 2003, pp. 1973–1976
9. Demuynck, K., Puurula, A., Van Compernolle, D., Wambacq, P.: The ESAT 2008 system for N-Best Dutch speech recognition Benchmark. In: Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop, Merano, Italy, Dec 2009, pp. 339–343
10. Demuynck, K., Roelens, J., Van Compernolle, D., Wambacq, P.: SPRAAK: an open source speech recognition and automatic annotation kit. In: Proceedings of the International Conference on Spoken Language Processing, Brisbane, Australia, Sept 2008, p. 495
11. Demuynck, K., Seppi, D., Van Compernolle, D., Nguyen, P., Zweig, G.: Integrating meta-information into exemplar-based speech recognition with segmental conditional random fields. In: Proceedings of the International Conference on Acoustics, Speech and Signal Processing, Prague, Czech Republic, May 2011, pp. 5048–5051
12. Gales, M.: Maximum likelihood linear transformations for HMM-based speech recognition. Comput. Speech Lang. **12**, 75–98 (1998)
13. Gemmeke, J.F., Van Segbroeck, M., Wang, Y., Cranen, B., Van hamme, H.: Automatic speech recognition using missing data techniques: handling of real-world data. In Kolossa, D., Haeb-Umbach, R. (eds.) Robust Speech Recognition of Uncertain or Missing Data. Springer, Berlin/Heidelberg (2011, in press)
14. HTK website. http://htk.eng.cam.ac.uk/
15. Kessens, J., van Leeuwen, D.A.: N-best: the Northern- and Southern-Dutch benchmark evaluation of speech recognition technology. In: Proceedings of the Interspeech, Antwerp, Belgium, pp. 1354–1357 (2007)
16. Lee, A., Kawahara, T.: Recent development of open-source speech recognition engine Julius. In: Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Sapporo, Japan, pp. 131–137 (2009)
17. Nogueira, W., Vanpoucke, F., Dykmans, P., De Raeve, L., Van hamme, H., Roelens, J.: Speech recognition technology in CI rehabilitation. Cochlear Implant. Int. **11**(Suppl. 1), 449–453 (2010)
18. Palomäki, K., Brown, G., Barker, J.: Missing data speech recognition in reverberant conditions. In: Proceedings of the ICASSP, Orlando, Florida, pp. 65–68 (2002)
19. Palomäki, K., Brown, G., Barker, J.: Techniques for handling convolutional distortion with 'missing data' automatic speech recognition. Speech Commun. **43**(1–2), 123–142 (2004)

20. Rybach, D., Gollan, C., Heigold, G., Hoffmeister, B., Lööf, J., Schlüter, R., Ney, H.: The RWTH Aachen university open source speech recognition system. In: Proceedings of the Interspeech 2009, Brighton, UK, Sept 2009, pp. 2111–2114
21. Stolcke, A.: SRILMan extensible language modeling toolkit. In: Hansen, J.H.L., Pellom, B. (eds.) Proceedings of the ICSLP, Denver, Sept 2002, vol. 2, pp. 901–904
22. Stolcke, A., Zheng, J., Wang, W., Abrash, V.: SRILM at sixteen: update and outlook. In: Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop, Waikoloa, Hawaii, Dec 2011
23. Stouten, F., Duchateau, J., Martens, J.-P., Wambacq, P.: Coping with disfluencies in spontaneous speech recognition: acoustic detection and linguistic context manipulation. Speech Commun. **48**(11), 1590–1606 (2006)
24. Stouten, V., Van hamme, H., Wambacq, P.: Model-based feature enhancement with uncertainty decoding for noise robust ASR. Speech Commun. **48**(11), 1502–1514 (2006)
25. Strik, H., Bakker, A.: Alfabetisering met een luisterende computer. DIXIT special issue on 'STEVIN en onderwijs' (in Dutch), p. 21. http://lands.let.ru.nl/~strik/publications/a148-TST-AAP-DIXIT.pdf (2009)
26. van Doremalen, J., Cucchiarini, C., Strik, H.: Optimizing automatic speech recognition for low-proficient non-native speakers. EURASIP J. Audio Speech Music Process. **2010**, 13 (2010). Article ID 973954. doi:10.1155/2010/973954
27. Van Segbroeck, M., Van hamme, H.: Advances in missing feature techniques for robust large vocabulary continuous speech recognition. IEEE Trans. Audio Speech Lang. Process. **19**(1), 123–137 (2011)
28. Wambacq, P., Demuynck, K.: Efficiency of speech alignment for semi-automated subtitling in Dutch. In: Habernal, I., Matoušek, V. (eds.) TSD 2011. LNAI, vol. 6836, pp. 123–130. Springer, Berlin/New York (2011)
29. Wu, T., Duchateau, J., Martens, J.-P., Van Compernolle, D.: Feature subset selection for improved native accent identification. Speech Commun. **52**, 83–98 (2010)