Direkt Profil – Implementation of a text critiquing system for non-native students of French

Diploma Thesis

Fabian Kostadinov

Department of Informatics
University of Zurich
Prof. Dr. Michael Hess
Supervisor: Dr. Manfred Klenner

In cooperation with the

Department of Romance Languages
Lund University
Dr. Jonas Granfeldt

Zürich, 4th of April 2005

Fabian Kostadinov
Ziegelfeldstrasse 73
CH-4600 Olten
fkostadinov@gmx.ch

English Abstract

Linguistic research has shown that people, who learn a second language, go through certain developmental stages with their language skills. On every stage, certain attributes or "indicators" can be measured which are characteristic for exactly this stage. By generating a profile based on indicators, a linguist can determine the developmental level of a language learner. *Direkt Profil* is a piece of software developed with the purpose of generating such profiles out of a text. It is developed for French second language acquisition. To detect the stage indicators, the program uses a natural language parser in combination with a bunch of condition/action rules applied to the text. This paper describes the software's theoretical background and explains the text analysis process.

Deutsche Kurzfassung

Die Linguistikwissenschaften haben gezeigt, dass Leute, welche eine zweite Sprache lernen, bezüglich ihrer Sprachfähigkeiten eine typische Folge von Entwicklungsstufen durchlaufen. Jede Stufe hat ihre eigenen Attribute oder „Indikatoren", welche gemessen werden können und für ebendiese Stufe charakteristisch sind. Ein Linguist kann die Entwicklungsstufe eines Sprachenlerners bestimmen, indem er ein Profil, basierend auf den Indikatoren, anfertigt. *Direkt Profil* ist eine Software mit der Aufgabe, solche Profile aus einem Text zu erzeugen. Es wurde für Französisch als Zweitsprachenerwerb entwickelt. Um die Indikatoren der Stufen zu ermitteln benutzt das Programm einen Parser für natürliche Sprache in Kombination mit einem Bündel Regeln der Form Bedingung/Aktion. Diese Arbeit stellt den theoretischen Hintergrund der Software vor und erklärt den Ablauf der Textanalyse.

Chapter overview

The paper is structured as follows:
In the first chapter, a general introduction to computer-assisted language learning is given. The chapter contains a short history of computer-assisted language learning, describes the relation to natural language processing, and gives an overview of several projects of interest in the corresponding field.
In the second chapter the theoretical linguistic background about the second language acquisition process is given. It is shown, how this acquisition process can be measured.
The third chapter explains the fundamental ideas and goals of using a software tool to support this task. The software is called *Direkt Profil*.
In the fourth chapter are presented: The second language learner corpus, the program's text annotation scheme, the control group of manually annotated texts and the French dictionary used by the program.
In the fifth and the sixth chapters, the technical program details are discussed. The fifth chapter introduces the more general ideas of how our software measures the second language acquisition process, whereas the sixth chapter discusses some implementation design details and software patterns.
In the seventh chapter, the program's analysis qualities are discussed, an example of an analyzed text is given, the pros, cons and limitations of the chosen approach are discussed and a possible solution to a bunch of problems connected to the chosen approach is presented.
Finally, the most remarkable points are summarized in the eighth chapter.
An installation guide for *Direkt Profil* can be found in the appendix, besides some DTDs, used in the program.

My thank goes out to all the ‘profilers’ – Emil, Jonas, Pierre and Suzanne –, who enthusiastically provide me with their support, and made it possible for me to spend a wonderful time in the beautiful city of Lund, Sweden.
The project has shown that ideas can grow – once they have fallen on a fertile ground.

“Emil, are you on Skype today?”

Fabian Kostadinov, April 2005

Contents

1 Computer-assisted language learning (CALL)

From the beginnings of the first computers in the 1950ties, computers have changed in their appearance, their human-machine interface, their computing power and their tasks. Nowadays, computers play a central role in our daily life. Times are gone since only a few technicians and academic experts had access to a few highly complicated machines, which required huge amounts of space and energy to deliver a result. Besides processing data, with the sudden appearance of the internet and diverse mobile devices, communication with the rest of the world has become a second main task to be fulfilled by a computer. Despite the wide spread of computers, it is astonishing to see that for many other tasks to be fulfilled by human beings computers can only provide poor support. As soon as it comes to issues highly creative and hard to plan in advance, there is a significant lack of knowledge of how to implement the necessary software programs. In such situations, humans must often play this role.

Learning and teaching a new language is such a task. Usually, a person who wants to learn a new language joins a group of students led by a teacher. Text and exercise books are available for many languages. Supporting software is available for the more wide spread languages, however, as far as the author can see, many of the available programs simply more or less reproduce the text and exercise book electronically. In such a situation, the software's advantages, compared to a text and exercise book, are only partially taken advantage of.

1.1 What is CALL?

CALL is the abbreviation of the term **computer-assisted language learning**. Computer-assisted language learning is a subset of the more general **computer-assisted learning (CAL)**. In a broad context, CALL can be seen as every kind of support given by computers, which serves a person to improve her[1] language skills and to adapt to a language. [Lindstedt 1998] suggests a differentiation between programs aiming at supporting the learning process – the **CAL programs** – and other programs that are not created with the intention to support learning process – simply called the **application programs**[2]. However, according to the author, CAL programs nowadays must be differentiated further: The **conventional CAL**

[1] In this paper, we will consequently refer to a language learner or computer user as a female.
[2] [Lindstedt 1998] defines an application program as follows: „A working program, such as a word-processing, statistical or drawing program."

programs and **"semi-application CAL programs"** (or new kinds of CAL programs). Whereas conventional CAL programs are more or less easily distinguishable from application programs due to the criterion of the intention behind they were developed with, the "semi-application CAL programs" must be located somewhere in between application programs and conventional CAL programs, because they show characteristics of both program types. They subsume those programs that lack the classifications of conventional CAL programs but still have the clear goal to assist learning and thus cannot be put into the category of application programs. "New Kinds of CAL programs may seem more like application programs, but are designed to serve learning purposes as well. It is typical that these programs often require a great deal of external material and that they are not self-evaluative. A response may be given, but it is not as denotative as in conventional programs. One typical feature of the new programs is that they are not structured (because of an application program imitation)." [Lindstedt 1998]

[Lindstedt 1998] enumerates four different (partially overlapping) categories of conventional computer-assisted learning systems: Tutorials, drills, simulations and (instructional) games.
Tutorials are close to traditional text books. They present a certain topic to a student in the form of a dialogue. Mostly in the end of a chapter, some exercises (multiple choice, gap fillers or others) can be solved to assess the student's progress.
Drill programs are often strongly repetitive. They resemble the process of learning new words with small memory cards (with the word in one's mother tongue on the one side and the foreign translation on the other side of the card). They repeat the same exercise again and again until at least a certain score is reached. Some exercises might also be picked up in a later loop.
Simulations imitate a certain process to instruct a student in it. A good example would be a program in physics to simulate how a rock will fly through the air if parameters like the gravity, the conditions of the air's state, the thrower's force etc. are manipulated.
Instructional games try to teach the student a phenomenon in a (hopefully) very motivational way. There are for example detective games where the player takes the part of a police officer to solve a complicated case, where she during the advancing game is taught the vocabulary and grammar of a foreign language.

It should be self evident that such a categorization only holds for conventional CAL programs. There are numerous programs already subsumed in the category of "semi-

application CAL programs". Furthermore, different software programs can be attached to one or the other group depending on the point of view one takes. To give an example: Microsoft Word's orthographic and grammar checker are not seldom used by writers in "abuse" of their original intention as a medium for learning a foreign language rather than as an error diagnosis and correction tool. Therefore, one could even count a writing assistance program as Word's orthographic and grammar checker to be a CA(L)L-system.

From a broad, scientific point of view, computer-assisted language learning is related to computer science, computational linguistics, linguistics of natural languages, educational sciences (and thus to psychology) and artificial intelligence. [Chappelle 2001] describes how six different fields related to computer application in second language acquisition (CASLA) have had a significant impact: educational technology, computer-supported collaborative learning, artificial intelligence, computational linguistics, corpus linguistics, and computer-assisted assessment. Human computer interaction (HCI) logically would be another field connected to CALL, however some voices argue that there was in fact little historic impact of HCI on CALL[3].

1.2 A short history of computer-assisted language learning

The beginnings of computer-assisted language learning reach back somewhere into the 1960s and 1970s. "It can be divided into three distinct periods which Warschauer refers to as *behavioristic* CALL, *communicative* CALL, and *integrative* CALL." [Sealander & Tholey 2002], they refer to [Warschauer 1996].
According to [Sealander & Tholey 2002] behavioristic CALL "was based on behaviorist theories of learning" from the 1950s, but set up not earlier than in the 1960s and 1970s. The core concept could be outlined as 'learning through repetition'. This naturally led to exercises with a focus on drill. A learner was given the same material repeatedly, because this strategy was thought to be beneficial to learning.

"The basis for the second period of CALL, Communicative CALL, was the communicative approach to teaching. This phase emphasized authentic communication that the drill and practice programs were unable to accomplish (Underwood 1984)." ([Sealander & Tholey

[3] See [Chappelle 2001] p.27 for some hints about this discussion.

refer to [Underwood 1984]). The focus was re-laid on practice and communication. To be able to communicate, it is not so much of importance whether grammar rules are learnt by heart (explicit knowledge), but their spontaneous application has more weight (implicit knowledge).
During this period, there were several types of programs developed. "[…] programs to provide skill practice, but not in a drill format (for example, paced reading, text reconstruction, and language games) [,… programs] to stimulate discussion, writing, and critical thinking [… and programs that] did not necessarily deal with language teaching material but allowed students to use or understand the use of language (for example, word processors, spelling and grammar checks, and desk-top publishing programs)."[Sealander & Tholey 2002].

The 1980s were marked through the employment of artificial intelligence techniques in the field of CALL. According to [Sealander & Tholey 2002], a third phase of CALL systems, integrative CALL, is now about to emerge driven by the diffusion of two key technologies: Multimedia and the internet. The former "allows the integration of skills like listening and reading, but it rarely integrates meaningful and authentic communication." [Sealander & Tholey, 2002] The latter, in its nature, supports the idea of task- or project-based language learning.

1.3 CALL systems and Natural Language Processing (NLP)

Computer-assisted language learning is related to **natural language processing (NLP)**. Natural language can be both – spoken and written language. If the user wants to interact with the software and get feedback, then a CALL system must at least be able to process her inputs to a certain extent, but also to return a corresponding answer as an output. This is an easy task as long as the given exercises are constrained to e.g. multiple choice questions or gap filling exercises. The number of possible inputs to and outputs from the system is small in such exercises. This is not the case when dealing with natural language anymore! "Whenever the range of possible answers is large or even infinite, specialized tools are needed. In the case of exercises requiring users to produce sentences in the language they are learning, natural language processing (NLP) tools are necessary to analyze the answer and produce intelligent feedback." [Vandeventer 2003].

NLP is a wide field including voice recognition and speech synthesis, automated translation, error diagnosis and detection techniques, text analysis, information extraction and retrieval and many more.

NLP systems always have to deal with a set of typical problems related to natural language. To name some of them:

- Natural language usually contains a lot of noise, like meaningless utterances, doubled words, etc. This is especially true for spoken language.
- Sometimes, input sentences are **ill-formed**. Common to written language are syntactical and orthographical mistakes.
- Input can also be semantically contradictory or wrong.
- Language might even be correct but still be subject to stylistic weaknesses.
- Language can be ambiguous as demonstrated with the example: *I saw the man with a telescope.*[4]
- The processed language might contain words unknown to the system. If the NLP system should be usable for a longer time period, the dynamics of a continuously transforming language must be taken care of.

Ill-formedness of natural language is an important problem, especially in CALL systems. During the language acquisition process, a language learner usually commits lots of different mistakes which, with increasing experience and knowledge, disappear gradually. It is a central issue to every CALL system, which makes use of NLP techniques, to accept and process even ill-formed input.

[Chomsky 1985] differentiates between the **competence** and the **performance** of a language user. Whereas he refers to competence as the tacit and "ideal" knowledge of a language and its rules, a language user has to express her knowledge also by actually applying the language. Performance is the (observable) result of employing the language. "Apart for spelling errors, which are often due to a lack of competence in this specific area, mistakes made by adult native speakers are due in a very large part to performance issues. Language learners, however, make mistakes for both performance and competence reasons. Thus, input provided by language learners contains on average more errors than input produced by native speakers." [Vandeventer 2003].

[4] Who is it, who is carrying the telescope – me or the man?

In the error correction process, two terms must be differentiated: diagnosis and correction.

- **Diagnosis** is the process of identifying a mistake.
- **Correction** is the replacement of an erroneous language construct with another one.

"To propose a diagnosis, one must state whether there are mistakes in a given sentence, give their locations, and indicate the error types to which they belong. Establishing a diagnosis does not involve giving correct alternatives for the errors detected. Providing correct alternatives is part of the correction process. Naturally, the correction process must start with a diagnosis phase, otherwise it would be unable to know what needs correction. Therefore correction includes diagnosis and completes it with possible solutions for the errors." [Vandeventer 2003]. Classic and wide spread tools for error detection are spell checkers. They can give the user a choice of several suggested altnernative words to replace the misspelled one.

Since input to a NLP system might contain errors, a requirement to NLP enabled software is **robustness**. A program that can handle ill-formed input can be called robust. Performance, flexibility, usability, reliability, security and accuracy/precision are further requirements. In a CALL environment, special care must be given to the feedback: The program should not suggest error prone "solutions" due to **overdetection** or misclassification of phenomena. Overdetection is the detection of an error by a software system even if there is none in reality! Reasons why a system overdetects can be manifold. Overdetection is not only frustrating for the user but, which is worse, such behavior could even mislead a language learner to adapt wrong language constructs! A good way to go in cases of uncertainty could be to give the user the possibility to choose between several suggested and perhaps ranked solutions the one she thinks is correct. Another problem with overdetection is that it steers the user's attention in a pedagogically undesirable way towards the overdetected phenomena.

Although the treatment of errors is an important issue to every NLP system (and also of course to CALL systems in general), it should not be neglected that error diagnosis and correction plays only a part of the whole system, mostly when it comes to giving feedback to the user during the interaction process. However, CALL software is more than just language processing. CALL has the declared goal to assist the learner during the language acquisition process. It must take care of the user's language abilities as of her progress during time. Furthermore, a user does not only want to know what is wrong or right according to a given language's grammar, but she or a teaching person also might want to make visible the

student's development. A CALL system must thus be prepared to give "positive feedback" to tell her, what she is able to express already and what she is able to communicate. From a pedagogical point of view, it is probably not even very desirable to insist on correcting every single mistake. A language learner should be encouraged to develop her language skills further by actively employing her language. Denoting every single language mistake might be, especially on a beginners' level, a rather frustrating experience. A positive approach would be to show a language learner what she is already able to do instead of solely pointing out the lacks of language skills. A CALL system must contain some kind of knowledge about a language and the developing acquisition process. NLP techniques are good means when dealing with natural language, but they are only the means a CALL system makes use of. They are not the CALL system itself. To the same extent, error diagnosis and correction is only part of a CALL system, but it is not the CALL system itself. The detection of what is correct and "good" has at least the same importance in a CALL system.

1.4 Current and older projects in the field of CALL and NLP

There are numerous CALL programs available now. The same counts for systems making use of NLP techniques. We will now present a selection of them. This list's elements are all of interest in one or another aspect to *Direkt Profil* – the CALL system we will describe below in this paper.

Some first implementations of a NLP systems stem from the 1980s. ***PLNLP*** stands for Programming Language for Natural Language Processing. "The intention of *PLNLP* is to correct orthographic mistakes, grammar and style. It addresses essentially but not exclusively to users writing in their mother tongue, be it English or French. The grammar checker of *PLNLP* performs a complete syntactic sentence analysis.
PLNLP works by using a set of binary unification rules – for English more than one hundred different rules – to build complete syntactic parse trees for all sentences. It takes a two step approach. First, a bottom-up parsing analysis is realized. The prepositional groups and subordinate clauses are reattached to their closest possible neighbors in the trees. In a second step, certain groups are reattached by using statistical knowledge." (Translated from French to English; from [Granfeldt et al. 2005]) During the correction phase, *PLNLP* uses a set of condition/action rules that are then applied to the tree. The conditions describe a certain

grammar or style error. When an error is found, the action part is applied proposing an alternative formulation.

The *Epistle* and the *Critique* systems, both developed under the patronage of IBM, were first implementations of *PLNLP* for English. Later on, implementations for French, German and other languages were added. The implementations were not meant to be CALL systems, but should support the text writing and correction process.

As we will see, *PLNLP* is similar to *Direkt Profil* in its idea of using binary condition/action rules that are applied to a text. Although the intention in its design is rather to support the authoring process and it is not designed to be a CALL system, certain important insights could be taken from *PLNLP* for our project.[5]

An interesting, currently running project is the ***FreeText*** project[6]. *FreeText* is a CALL software system for learners who study French as their second language. "The goal of the *FreeText* project is to establish a CALL system for French ([Granger et al. 2001] and [Vandeventer 2003]). The system aims at processing authentic documents (texts written freely by language learners) and it follows a communicative approach to language learning. The target of *FreeText* is to offer the user several levels of functionality. A first level produces communicative exercises, which the system corrects automatically by using techniques of NLP. Furthermore, the NLP tools are always accessible to the learner to test and visualize the analysis of a learner's texts. The error diagnosis system uses error typologies extracted from a learners' corpus for correction of written texts in the exercise environment." (Translated from French to English; from [Granfeldt et al. 2005].) On the project webpage, one can read: "[…] The system will also include reference aids such as a linguistically motivated reference grammar, teacher-oriented authoring capabilities, and an evaluation tool." (See the footnote for a link to the project homepage.)

The ***Granska*** project[7] (in the meanwhile renamed to *CrossCheck*) has built a grammar checker for second language learners of Swedish. The system is specially intended to support the writing process in Swedish. *Granska* contains three important modules: "An orthographic checker, a part-of-speech tagger and a partial syntactic analyzer. The partial analyzer uses

[5] See also [Jensen et al. 1993] for a detailed discussion of the *PLNLP* approach.

[6] *FreeText* is being developed at the *University of Manchester Institute of Science and Technology*, the *Université de Genève*, the *Université Catholique de Louvain* and the *Softissimo SA* company. The project homepage can be found under http://www.latl.unige.ch/freetext/index.html.

[7] *Granska*'s project homepage is http://www.nada.kth.se/theory/projects/granska/index-en.html. *Granska* is developed at the Numerical Analysis and Computer Science Royal Institute of Technology, Sweden.

syntagmatic, manually constructed rules that it applies to the [already] tagged text. It detects adverb, noun, adjective, prepositional, verb and infinitive groups (Knutsson et al. 2003)." (Translated from French to English; from [Granfeldt et al. 2005].)

Dialang is a project running with the support of *The European Commission Directorate General for Education and Culture*. Many different partners contribute to it[8]. *Dialang* is a software tool to test one's language abilities for 14 different European languages, including French. For every language it provides a series of different exercises in the categories of reading, writing, listening, grammar and vocabulary, which can be solved individually without an instructing person. Some exercises include some sort of interactive features, for example short movies or radio plays in combination with multiple choice tests, short games (where a task description must be understood first to reach a correct result), gap fillers, short writing tasks etc. *Dialang* uses the scale of the Common European Framework (CEF) to evaluate performance. The CEF is a standardized 6 stage scale (A1, A2, B1, B2, C1 and C2) for all the fourteen languages to indicate the language level. It will be presented further below. A central issue is self-evaluation of the language learner.

The source of inspiration for *Direkt Profil* is an already existing language profiling tool called ***Rapid Profile***. *Direkt Profil* does not only deduce its name from *Rapid Profile*, but it also shares some of the theoretical psycholinguistic background. However, *Direkt Profil* treats written (French) language of L2 learners only, whereas *Rapid Profile* is developed for spoken (English and German) language of L2 learners.

"Rapid Profile is a computer-assisted procedure for screening speech samples collected from language learners to assess their level of language development as compared to standard patterns in the acquisition of the target language."[9] In *Rapid Profile*, first a speech sample is collected from the person who should be analyzed (the informant). For this purpose, a simple communicative task is given to the informant, for example describing a series of pictures to another person. A third, specially trained person – the analyst – has a predefined table of indicators (as a software interface), where she can check a box every time when the informant happens to use a construct indicating her skill level in the language acquisition process. When

[8] *Dialang*'s project homepage is http://www.dialang.org/. A fully functional version of the software is freely downloadable from the homepage.

[9] Excerpt from a description paper about *Rapid Profile* available on http://www-fakkw.upb.de/institute/Anglistik_Amerikanistik/Personal/Kessler/Rapid_Profile/English_Flyer.pdf. *Rapid Profile* was developed at the English Department, Linguistics Program of the University of Paderborn under the guidance of Manfred Pienemann.

enough information has been gathered to return a significant analysis of the informant's language skills, the software automatically computes the informant's language level by comparing the given checked boxes to a standard profile. There is also a functionality giving feedback to the analyst about language phenomena which is underrepresented in the analysis, so that she can choose another oral task for the informant to receive a more reliable result.

2 The progress of learning a language[10]

In this chapter it is shown in its basic shape, how French as a foreign language is learnt. The whole chapter relies heavily on research being done by Inge Bartning and Suzanne Schlyter about the language acquisition process of French. See also [Bartning & Schlyter, 2004] and [Schlyter 2003] for a further discussion of the topic. For a broader theoretical background about profiling the second language acquisition process, see also the work of Harald Clahsen (e.g. [Clahsen 1985]) and Manfred Pienemann (e.g. [Hyltenstam & Pienemann 1985]).

2.1 The language learning process

Linguistic research has clearly shown that the acquisition of specific grammatical construction in a new language follows a certain order. For instance, a learner of French must, irrespectively of whether it is a child learning the subtleties of its mother tongue or a grown-up student acquiring a language during a holiday trip to France, first have mastered a verb's present tense to a certain degree before she can start heading for past tense forms, such as the French *passé composé* or *imparfait*. Text books and language teaching courses are only to some extent built up according to these insights. Though the given example is still intuitive to every reader, research has detected further details in the order of how a language is acquired, many of these details being discovered only through the systematic use of empirical tools. As an example, we could cite the increasing complexity of phrase structures which may be built by a learner, or the gradually disappearing number of wrong set verb tenses (e.g. usage of *passé composé* instead of *imparfait*).

What is well understood by linguists is how the language acquisition process progresses for first language learners. In the past, knowledge about the second language acquisition process has not been developed by researchers to the same extent. It may be surprising to hear that this process underlies a specific order too, independent of how a language is actually learnt: in a heavily natural, unstructured way, let us say during a several weeks trip to a foreign country, or a led and structured language course with clear goals of what to learn when (compare

[10] In this paper we will show many examples of written French language. To make it as easy as possible for the reader to understand those examples, we will highlight them from the text by using an italic font. Additionally, language mistakes will be marked with a leading asterisk sign (*) to indicate an erroneous construct. We might sometimes use the abbreviation L2 for second language, meaning a person learning a foreign language not as its mother tongue.

[Bartning & Schlyter 2004]). If a student is taught a language's advanced grammar explicitly already at an early language level and the student might be able to solve exercises and tests, she might still fail to reproduce the correct rules during **spontaneous language production**. The student might be able to produce correct results in multiple choice tests or give correct answers to questions in grammatical tests, but as soon as the task is to reproduce language spontaneously (as in a conversation or in a written essay), the learner will again follow the order of the language acquisition process introduced above. (For a further discussion of this topic, see for instance [Pienemann 1998].)

For different languages, some researchers have introduced several stages or levels of language skills, which split up the acquisition process as a whole into several shorter phases. Currently, there are stage models available for (amongst other) the languages: French, Italian, German, Swedish, Spanish and English. However, for some languages only the earlier stages have been subject to more detailed research.

In a complete stage model, the stages reach from absolute beginner's level with practically no knowledge of the language at all, through some intermediary stages of already higher skilled learners, up to a level where a person is absolutely familiar with all the details of the acquired language and shows a very high competence in daily life. A second language learner on this level could be thought of practically having reached bilingual skills.

With the *Common European Framework of Reference for Languages* (*CEF*) a standard for the language acquisition process was proposed by the Council of Europe. The model should be seen as an attempt to introduce a general base for learning a language and assessing the acquired language skills for most European languages. The *CEF* suggests totally 6 stages (the descriptions are taken from [CEF]):

A (Basic User)	**A1**	Can understand and use familiar everyday expressions and very basic phrases aimed at the satisfaction of needs of a concrete type. Can introduce him/herself and others and can ask and answer questions about personal details such as where he/she lives, people he/she knows and things he/she has. Can interact in a simple way provided the other person talks slowly and clearly and is prepared to help.
	A2	Can understand sentences and frequently used expressions related to areas of most immediate relevance (e.g. very basic personal and family information, shopping, local geography, employment). Can communicate in simple and routine tasks requiring a simple and direct exchange of information on familiar and routine matters. Can describe in simple terms aspects of his/her background, immediate environment and matters in areas of immediate Basic need.

B (Independent User)	**B1**	Can understand the main points of clear standard input on familiar matters regularly encountered in work, school, leisure, etc. Can deal with most situations likely to arise whilst travelling in an area where the language is spoken. Can produce simple connected text on topics which are familiar or of personal interest. Can describe experiences and events, dreams, hopes and ambitions and briefly give reasons and explanations for opinions and plans.
	B2	Can understand the main ideas of complex text on both concrete and abstract topics, including technical discussions in his/her field of specialisation. Can interact with a degree of fluency and spontaneity that makes regular interaction with native speakers quite possible without strain for either party. Can produce clear, detailed text on a wide range of subjects and explain a viewpoint on a topical issue giving the advantages and Independent disadvantages of various options.
C (Proficient User)	**C1**	Can understand a wide range of demanding, longer texts, and recognise implicit meaning. Can express him/herself fluently and spontaneously without much obvious searching for expressions. Can use language flexibly and effectively for social, academic and professional purposes. Can produce clear, well-structured, detailed text on complex subjects, showing controlled use of organisational patterns, connectors and cohesive devices.
	C2	Can understand with ease virtually everything heard or read. Can summarise information from different spoken and written sources, reconstructing arguments and accounts in a coherent presentation. Can express him/herself spontaneously, very fluently and precisely, differentiating finer shades of meaning even in more complex situations

Table 1 The Common European Framework of Reference for Languages

Dialang, introduced in chapter 1.4, relies exactly on these stages.

Of course, this is only a general framework, and it must first be transferred to a language to give a more concrete idea of where a language learner can be located in the acquiring process. The descriptions are of a very general nature. Though, with these rather simple developmental descriptions, it is possible to investigate one's personal language skills.

2.2 Developmental stages for learners of French

For French, a much more detailed stage model has been developed both by Bartning and Schlyter. Independently on each other and independently of the CEF they proposed 6 stages to be an adequate description of the language learning process. The proposed stages reflect spontaneous usage of the language rather than formal usage as a response to an exercise in a text book. These stages must on the other hand not be understood as a pedagogical prescription of how to best teach a language. As stated above, they are rather to be seen as either a description of the natural language adaptation process or as a source of experience that can be consulted to find out what can and what cannot be demanded from a learner on a certain level. For instance, one could imagine a student being taught and also able to

recognize and produce a conditional verb form in an exam. However, the same student probably will fail to reproduce it during an oral conversation with the teacher, if she is still on a stage on where she struggles to differentiate between present tense and infinitive. Even with intensified teaching of the conditional, the student will probably not be able to master the verb's mode correctly in a spontaneous situation.

We will discuss the stages and how to define them in greater detail below. What can be revealed here already is a general overview of the most influential criteria to build up stages as described by [Bartning & Schlyter 2004]:

- **Nominal Utterance Structure:** Language learners in the early stages tend to use short sentences with many nouns and pronouns. Often they do not use any verb at all. As they progress, the usage of nouns becomes more balanced with verbs.
  Example: Thus a beginner might say **Je faim!* when she is hungry, whereas a more advanced language learner will not make the same mistake anymore and say correctly *J'ai faim!*
- **Negation:** Early stage language learners often use wrong negation forms. For example, the negation *non* immediately followed by a lexical word is used. Also it is common in early stages to use negations *ne* without *pas* or *pas* without *ne*, like **j'aime pas*[11]. When language learners reach stage 3 or 4, most negations are normally correct.
  Example: An early stage learner might say **chambre <u>non</u> grand lit* to express the fact that there is no big bed in a room. Later on, the same language learner will use a more complicated (but correct) form, for instance *Il n'y a pas de…* to express the same fact.
- **Finite and infinite verb forms:** Early stage language learners often do not conjugate verbs and use infinite verb forms (infinitives, participles etc.) instead. In stage 1, only around 50% - 75% of all verbs are finite. In stage 4, around 90% - 98% of verbs are finite.
  Example: In early stages constructions like **il parler* can still be found regularly. This will disappear gradually in later stages.

[11] The reader must be aware of the fact that the negation *pas* without a leading *ne* is common in spoken French. Many French speaking persons with highly developed French language skills use formulas, in which they drop the leading *ne*, like in *Je pense pas*. However, this has nothing to do with the language acquisition process. During the early stages of the language acquisition process, the learner has not yet mastered the point, where she can use the leading *ne* or drop it 'on demand'. This fact is mirrored in the table.

- **Agreement between the sentence's subject and verb:** In early stages, agreement between the subject of a sentence and the verb (in person and number) can be met nearly as often as not. In stage 4 and above, usually agreement is correct.
- **Tense, mode, aspect (TMA):** As a language learner progresses, she will be able to use more "advanced" conjugated verb forms to express different tenses, like *passé composé, imparfait, subjonctif* etc. Aspect refers to the ability of a language learner to actively use the verb form which is obligatory from the given context in the sentence, for example differentiating between the usage of *imparfait* and *passé composé*, depending on whether something is a completed act or has a still continuing effect.
- **Elision/Cliticisation:** In many languages, there are rules to "merge" certain syllables if they follow one another. Sometimes, this is referred to as *incorporation* of a word in another word. In early stages, learners still often hurt the exact elision rules.
  Examples: **le enfant → l'enfant, *je écoute → j'écoute, *je le ai pris → je l'ai pris*
- **Gender and agreement:** Agreement between the article and the noun and also between the adjective and the noun. In early stages, the choice of articles is correct in around 60% – 65 %, in the later stages almost all articles are used correctly.
  Example: **la soleil → le soleil*.
- **Incorporation of article and preposition:** This point goes in a similar direction as elision/cliticisation. In French, an article is sometimes incorporated into a preposition. In early stages, learners often do not merge the two.
  Examples: **de le → du, *à les → aux*
- **Subordination:** Language learners of early stages will use only a few or none subordinated sentences. Such constructs are more often made use of in more advanced stages.
  Example: *L'homme <u>qui</u> habite à Paris…*

We will now take a deeper look into the stages and their development. The following table is the current result of a systematization of stages and a number of corresponding developmental sequences for around two dozens of phenomena examined by Schlyter. Once again it is to be stated that the table is based on language produced spontaneously by learners, which means mostly **oral** language. It still needs some more research to be done first to clarify to which extent the listed phenomena can be transferred for written language. For instance, stage indicator 1, the percentage of verbs used in sentences, is clearly related to spoken language. It is an assumption that this stage indicator will differ for written language. However, it is

Schlyter's expectation that most developmental sequences in the table also can be successfully reused for written language. First attempts for a deeper analysis seem to strengthen these believes.

Table of stages and phenomena:

	Stages	**1**	**2**	**3**	**4**	**5**	**6**
	Stage Name	***Initial***	***Post-initial***	***Intermédiaire***	***Avancé bas***	***Avancé moyen***	***Avancé élevé***
	Unguided learners	1 - 5 months	4 - 9 months	8 - 13 months	12 - 24 months	3 years	> 3 years
	Approximative scholar instruction time; study time of guided learners	Ca. 25h – 100h	School 1 – 2 years; ca. 80h - 200h	School 3 – 5 years; ca. 150h – 500h	University 1st/2nd semester; ca. 300h - 800h	University 3rd/4th semester	Professional French speaker
	Finiteness						
1	Percentage of sentences with verbs in conversations	20-40%	30-40%	50%	60%	70%	75%
2	Type of verbs showing opposition between finite and infinite forms in relevant contexts	No opposition for type of verbs	10-20% of types of verbs show opposition	ca. 50% of types of verbs show opposition	The most of verbs show opposition	+ (All verbs show opposition.)	+
3	Number of finite forms / Total number of verb forms (finite and infinite)	50-75%	70-80%	80-90%	90-98%	+	+
4	For copular and auxiliary verbs: The same verb is used correctly conjugated for 1st and 3rd person singular (and later on for 2nd pers. sg)	No opposition shown (except for multi word expressions like *c'est* or *j'ai)*	Opposition is about being learnt: *je suis* vs. *il est*; *j'ai* vs. *il a*	Opposition is mastered in most cases (isolated errors like **j'a* or **je va)*	+	+	+
5	1st person plural has a correct *–ons* ending	- (no production of 1st person plural)	70-80% of the produced 1st person pl. forms have a correct *-ons* ending (the rest is e.g. **nous a*)	80-95% correct	Only errors in complex constructions	+	+
6	3rd person plural has correct –ont ending (for verbs like *sont, ont, font, vont*)	- (no production of 3rd person plural)	Occurrences of 3rd pers. pl but wrong form: **ils est* or **Paul et son chien va à la maison.*	50% of 3rd pers. pl verbs have correct *-ons* ending (the rest is e.g. *ils est)	60-80%	+	+

7	As in 6 for verbs, where the difference in 3rd pers. pl compared to 3rd pers. sg can be heard for *-ent* endings (*viennent, finissent, prennent, veulent etc.*)	- (no production of 3rd person plural)	Occurrences of 3rd pers. pl but wrong form: **ils prend* etc.	Some correct occurrences occasionally	Ca. 50% correct	Still a few problems	+
	Temps/Mode/Aspect						
8	*Usage of auxiliary modal+ infinitif* and *futur proche*	- (no production)	first usages of both	Both usages are mastered. First occurrences of *futur simple* for guided learners.	*Futur simple* is used besides futur proche sometimes.	*Futur simple* is used moreoften.	Usages of *futur simple* are correct. (But also *futur proche* is used.)
9	*Conditionnel*	usage of v*oudrais* only	as in stage 1	as in stage 1 and 2	*Conditionnel* appears	*Conditionnel* is used moreoften	+ (*Conditionnel* is mastered)
10	How many times in a past context the verb is marked? (Without differentiation between *passé composé* and *imparfait*)	0-10%	10-40%	40-60%	60-90%	90-100%	+ (~100%)
11	Differentiation in usage of *passé composé* and *imparfait*	*Passé composé* in isolated cases.	The appearing *passé composé*s function in a perfect context	Present tense is not used anymore in a *passé composé* context.	+ Differentation between *passé composé* and *imparfait*	+	+
12	*Imparfait*	- (No usage of *imparfait*)	Usage of *imparfait* only in rare cases, often incorrect	Usage of *imparfait* but still also often usage of *présent* instead	Differentiated usage of *passé composé* vs. *imparfait* appears for avoir *avait/était*	As stage 4 but also for modal auxiliary verbs	+ (with all verbs)
	Plus-que-parfait	- (No usage of *plus-que-parfait*)	- (No usage of *plus-que-parfait*)	- (No usage of *plus-que-parfait*)	*Plus-que-parfait* is appearing at this stage with uncertain usage	*Plus-que-parfait* is more or less ok	+

14	*Subjonctif*	- (No usage of *subjonctif*)	- (No usage of *subjonctif*)	Usage of *subjonctif* for guided learners in isolated cases	*Subjonctif* appears	Usage of *Subjonctif* improves	*Subjonctif* is more or less ok
	Other stage indicators						
15	*Négation*	Negation takes form like: negation + noun/verb (*non chien*) or *C'est ne pas bien.* Occurrences of *Jensaispas*.	*ne* without *pas* or *pas* without *ne*: *Je mange pas.* (correct for spoken French!)/*Je ne mange.*	Guided: Tendency to use *ne*+verb instead verb+*pas*. Unguided: Tendency to use verb+*pas*; beginning usage of *rien/jamais*	Usage of correct negations: (*ne*+)verb+*pas*	Additionnally: usage of *personne ne.../rien ne...*	+
16	*Pronoms d'objet*	- (no usage of object pronouns)	Subject+verb+object (SVO): *Je vois lui*	Subject+conj.verb+object+main verb (S(v)oV): *Je veux le voir./*J'ai le vu.*	Subject+object+conj.verb+main verb (SovV) appears: *Je l'ai vu.*	*Je l'ai vu.* is mastered	+ Additionally *y* and *en* are used.
17	Gender/number agreement of article and noun	55-75%	60-80%	65-85%	70-90%	75-95%	90-100%
18	Agreement of determiner+adjective +noun	ca. 15%	ca. 25%	ca. 50%	ca. 70%	ca. 80%	ca. 90%
19	du/des/au/aux	Usage of de+noun or à+noun (which sometimes is correct)	Nearly all different combinations: **du le, *de le, de, du etc. (also for à+article)*	As in stage 2	Correct usage of de as a preposition, but still mistakes for de as partitive article	Wrong usage of **de le* has disappeared	+
20	Correct usage of *des*	20-60% (of all cases are correct)	40-80%	60-90%	90-98%	+	+
21	elision of subject+verb (*j'ai*) or article+noun (*l'enfant*)	Non guided learners have both elisions and not elisions	Ca. 60% of elisions correct	Ca. 90% of elisions correct	+	+	+

22	Complexity of subordinated clauses	No subordinated clauses	Usage of *qui* in isolated cases	Start to distinguish between qui and que	Improving, higher variation of subordinated clauses already	Usage of autonomous relative clauses and *gérondif*	*gérondi*f is mastered; usage of *auquel*

Table 2 The stages and developmental sequences

An **opposition** of a type of verb is the appearance of the same verb in at least 2 different forms, both of which are "target like". "Target like" means that their appearance corresponds to native French and should therefore be seen as correct.
The term **marked** means that in a given context an obligatory phenomenon is shown. For instance must in the sentence starting with *Hier, je…* (English: *Yesterday, I…*) a past tense verb form be marked.

Along the horizontal axis, for each language phenomenon the progress as a function of time and increased experience is shown. Along the vertical axis, the reader will find for each stage a description of an indicator typical for this stage. There are numerous developmental sequences or phenomena for each stage. A minus sign (-) inside the table indicates that the corresponding phenomenon will not occur in the production of a learner who is actually on this stage. The plus sign (+) on the other hand indicates that a learner on this stage has mastered the corresponding phenomenon. It must also be noted that the stated percentage values are only an approximations. A variation of +/- 10% can be expected for many of the indicated stage indicators.
Furthermore, the table is still subject to ongoing research.

It would go too far to explain the whole table in detail here. We will only take a look at a few examples and then give a summary of the stages. The interested reader is referred to [Bartning & Schlyter 2004] for a detailed discussion of the topic.

This is an example of how to read the table:
Stage indicator 16 inspects the development of the position of the object pronoun in the sentence. Where will a language learner put the object pronoun in production according to her language skills? As can be seen in the table, in stage 1 nothing is indicated at all for this stage. The minus sign means that at this stage the language learner has no knowledge about the phenomena at all in online production. It can be expected that she is not able to produce the

phenomenon at this stage at all (or only randomly). Instead, a learner will omit object pronouns and prefer using alternatives.

On stage 2, a progress has been made by the learner. Simple *sujet – verbe – objet* (*S-V-O*) patterns are produced as **je vois lui* (correct: *je le vois*).

Only on stage 3 the learner starts to put the object pronoun in an "intermediate" position in between the (conjugated) auxiliary verb and the main verb. Typical examples on this stage would be *Je veux le voir.* and *Je peux le faire.* on the one hand, but also **J'ai le vu.*

On stage 4, the correct order of *sujet – objet – verbe auxiliaire – verbe* (*S-O-(v)-V*) has established (*je l'ai vu*), but still wrong word orders might be found.

On stage 5, the learner has deepened her skills and the object pronoun is usually put in the position before the finite verb or after other auxiliaries than *avoir* or *être*.

On stage 6, additionally *y* and *en* are established in the spontaneous language production.

This is a general short description of phenomena encountered on every stage.

- **Stage 1 – *le stade initial*:**
  - Strong usage of nominal structures but also of other grammatical means.
  - No differentiation between "short finite forms" and infinite forms.
  - No differentiation between persons of a verb.
  - Very often elision rules are not applied.
  - Negation often takes the form: *negation X* as in **non grand-lit*.
  - Frequent multi word expressions like *je (ne) sais pas* or *je m'appelle*.
  - *Passé composé* is produced occasionally; however it is rare in the obligatory past contexts.
  - Occurrences of connectors as *et*, *mais* or *puis*.

- **Stage 2 – *le stade post-initial*:**
  - Some grammatical phenomena occur, but still with a high variability.
  - Simple subordination sentences, introduced with *quand, parce que*.
  - *qui* and *que* appear during language production.
  - *négation préverbal* (*ne* without *pas*) and *négation postverbale* (*pas* without ne) are both used.
  - Learners starts using modal verb forms (followed by an infinitive) and the *futur périphrastique*.
  - More productive usage of *passé composé*.

- Guided learners sometimes use the *imparfait* forms *était* and *avait*.
- Infinite verb forms in finite contexts are still frequent but the number of ‚short finite forms' increases.
- For the verbs *être* and *avoir* the difference between 1st person singular and 3rd person singular is mastered (*j'ai* and *il a* respectively *je suis* and *il est* are differentiated correctly). For 1st person plural and 3rd person plural, this is not yet the case. Correct 1st person plurals (*nous V-ons*) alternate with incorrect constructs (*nous V*). For instance: *Nous parlons* can be found, but also *nous parle*.
- Object pronouns are in general placed after the verb.

- **Stage 3 – *le stade intermédiaire*:**
  - A first more systematized and regular but still simple interlanguage emerges.
  - However, the interlanguage still contains overextensions and regularizations not according to the norms of the target language.
  - Negations now have the correct form *ne Vfin pas* (= *ne* + *finite verb* + *pas*).
  - Mostly, in a past tense context, a past tense verb form is also used. The same counts for a future tense context (mostly the *futur périphrastique*, in isolated cases also the *futur simple* for guided language learners).
  - Object pronouns are placed before the lexical verb for both composed and simple verb forms – often incorrectly after the auxiliaries *est/a*.
  - Already rich usage of subordinated clauses.
  - Fewer occasions of infinite verb forms in a finite context.
  - 1st person plural verb forms (*nous V-ons*) are marked in most of the times.
  - The differentiation between the verbs' 3rd person singular and plural forms is established in cases like *sont*, *vont* etc.

- **Stage 4 – *le stade avancé bas*:**
  - Appearance of specific and more complex and varied structures of French: the cliticised pronoun placed before the auxiliary verb (*j'ai* instead of **je ai*), the *conditionnel*, the *plus-que-parfait* and the *subjonctif*.
  - More complex verb forms are appearing now like the *subjonctif* or forms that are relying on discursive relations. However, these forms are not yet systematically marked with the obligatory verb forms.
  - Negations also with *ne…rien/jamais/personne*. Most of them are placed correctly.

- Nearly no infinite verb forms like **ils parler* in a finite context anymore.
- Domination of 3rd person plural verb forms like *ils/elles ont, sont, font* over the wrong **ils/elles a, est*. Lexical verbs have plural marking for verb types like *ils prennent* side by side with incorrect forms as **ils prend*.
- Clitisation of the article and preposition (*au, aux, du, des* etc.) and also of the pronoun.
- Still problems with the gender agreement between the article and the noun (**la soleil* instead of *le soleil*).

- **Stage 5 – *le stade avancé moyen*:**
  - Stage 5 is characterized by the further development of inflected morphology:
  - 3rd plural verb forms like *ils sont, ont, font* etc. are applied correctly now.
  - A few problems still with irregular 3rd person plural verb forms for the *présent*.
  - *Plus-que-parfait, conditionnel* and *futur simple* are applied correctly in most cases
  - Subjonctif has become more productive.
  - Still agreement between the article and the noun or the determiner (*de* or *à*) + article and the noun is not fully mastered.
  - Negations of the form *personne/rien ne + verbe* are used where the negation has a subject's function.
  - *Relatives* with *dont* and *gérondif* can be found.

- **Stage 6 – *le stade avancé superieur*:**
  - Stable inflected morphology, also in complex sentences and subordinate clauses
  - Connectors like *enfin* or *donc* are used naturally
  - Compact clause combining structures are used like the *gérondif*.

(These descriptions of the stages and the phenomena are to a far extent taken from [Bartning & Schlyter 2004].)

The table as a whole describes the phenomena a learner is able to produce spontaneously on a certain level. It does not so much lay the focus on the question of what is done wrong or right, but rather on the question: What can the learner already do with her current language skills?

It must be made a point that all the phenomena the table encodes represent rather implicit knowledge than explicit knowledge, so that a learner can reproduce those phenomena as a

spontaneous task. As a guided learner, she might have learnt the correct rules of how to set the object pronoun in every different case. She might further be able to apply all those rules correctly given a text book's linguistic exercise or give the right answer to most of the questions in a specifically prepared test. But this is explicit knowledge rather than implicit, and the learner first must progress to a stage of automated usage. In this respect, the situation is actually similar to learning how to drive a car: During the first few lessons, a driver will have to focus her whole attention on the steering mechanism and even then not be a good driver, although she knows all the details about the gas, the breaks, the steering wheel and so on. With increasing experience, her driving abilities will noticeably improve. The above described conception is built on further theories in the field of psychology and linguistics, as for instance the Processability Theory formulated by [Pienemann 1998].

One point remains to be mentioned: The table heavily relies on a corpus of Swedish-speaking learners of French (the corpus will be discussed later in chapter 4.1). The examined students already knew English as a second language besides their mother tongue and French was the second foreign language to learn for them. It is still a matter of research to find out to what extent the language acquisition process is the same for people with different languages as their mother tongue.

3 Introduction to Direkt Profil

Profiling the process of second language acquisition is not a completely new idea (see for instance [Clahsen 1985].) In this chapter, we will introduce the reader to *Direkt Profil* and our motivation to build it. *Direkt Profil* (or *DP* as it is abbreviated) is a software tool to detect the stages discussed above, which is currently being developed at the Lund University. The *Direkt Profil* project is a cooperation of the Department of Romance Languages and the Institute of Informatics.

A current version of *Direkt Profil* can be tested under http://www.rom.lu.se:8080/profil/logon.jsp.

3.1 A short history of the DP-project

Analyzing a text manually to detect circa two dozens of indicators to define the linguistic stage of a writer is a task, which requires a great deal of energy and time. It would therefore be very useful to have a software tool, which does this work in an automated way. A user would only have to give a text to the program, and the program then would find indicators, highlight them, calculate the most probable stage and finally present the result to the user. Of course, the software should be easy enough to use and give reliable results; otherwise the effort to use it could exceed the potential gains.

Furthermore, during our work we observed that although there are several, both commercial and non commercial, products with a background in linguistics, pedagogical sciences or computational linguistics, there is an astonishing gap in the availability of software which not only offers the language learner some static exercises connected to an amount of didactically prepared material, but incorporates in its design and logic the knowledge of how a language is adopted by a human.

Direkt Profil was born out of this idea: Having such a piece of software which makes visible the different stages, a language learner is passing through.

So, in autumn 2003 a first prototype was programmed during a few weeks school project by Kostadinov and Thulin[12]. It was still a prototype stuck in its early stages, but it defined the fundamental ideas of how such a piece of software could be implemented. Later on, the still unfinished version of the prototype was given further to Lisa Persson, a master student at the

[12] See also: [Kostadinov & Thulin 2003].; „A text critiquing system for Swedish-speaking students of French“

University of Lund, to make a first functioning version and to iron out certain serious drawbacks the prototype had had. Emil Persson joined the project as a freelancer and has ever since worked on *Direkt Profil*. The project stands under the guidance of Jonas Granfeldt (PostDoc researcher at the Department of Romance Languages, Lund University) and Pierre Nugues (lecturer for computational linguistics at the Institute of Informatics, Lund Institute of Technology) and is based heavily on the linguistic research being done by both Suzanne Schlyter (professor for French language at the Department of Romance Languages, Lund University) and Jonas Granfeldt.

3.2 Goals of the Direkt Profil-project

We formulated the main goal of the *Direkt Profil* project as follows:
"Le but central du projet est de developper une analyse automatique partielle des textes écrits en français par des apprenants de langue maternelle suédoise. [...] L'idée de base est d'implémenter les phénomènes linguistiques des itinéraires d'acquisition dans le logiciel ce qui permettra une analyse et, à long terme, une évaluation automatique des textes. Il est important de souligner que le système n'est pas un correcteur grammatical ou orthographique."[13] [Granfeldt et al. 2005]
As its name indicates, *Direkt Profil* is a **profiling tool** of written texts rather than one of the already well known checkers: such as spell checkers, grammar checkers, style checkers etc. Although the implementation of *Direkt Profil* may make use of traditional (spell or grammar) checking algorithms if necessary, it must not be confused with such a system. *Direkt Profil* does not mainly concentrate on finding erroneous language constructs, but it focuses to the same or even further extent on finding correct constructs also. On the other hand, *Direkt Profil* might never even try to find all possible mistakes in a text, if it is not necessary to tell us something about the developmental stage of a language learner. The program shall give feedback to the user not only to tell her what she has done wrong, as most classic CALL systems do, but essentially what she has done correct! The user will thus be provided with a neutral and balanced critique of what she is already able to do and where the limits of her current language skills lie, instead of only mirroring to her what she has done wrong.

[13] Translation: „The central goal of the project is to develop an automated, partial analysis of texts written in French by learners with Swedish as a mother tongue. […] The basic idea is to implement the linguistic phenomena of the acquisition steps in the [program] logic that carries out the analysis and, in the long run, an automated evaluation of the texts. It is important to make the point that the system is neither a grammar checker nor an orthographical [checker]."

Under the viewpoint of diagnosis and correction of language mistakes, the program must be located on the diagnostic side, since it does not support the user with hints or alternatives that could be useful for correcting a mistake and neither does it apply automated correction (for instance as *Microsoft Word*'s spell checker can do[14]). As should be clear from above, all the same *Direkt Profil* is a diagnostic tool for language errors only to the degree where the detected erroneous constructs fall in the catalogue of the phenomena we search for.

Seen in this way, we thoroughly implement linguistic knowledge about the learning process in our program. However, this knowledge is represented implicitly and not directly visible to the user. Whereas in most traditional CALL systems somebody has to prepare and select suitable exercises, which are given to the learner, *Direkt Profil* simply takes as an input free text production on whatever the learner wants to write on.
It combines the disciplines of linguistics, computational linguistics and pedagogy. There are several fields where our program could be used, for example to support further linguistic research about the mentioned language acquisition process. There are still many questions left unsolved for the researchers. For instance, it is a matter of interest to inspect the development of many morphosyntactical criteria for written language, which are already studied well for oral language. It is an easy task for a software program to analyze a whole corpus of texts at once, whereas it is very tedious to do the same work manually.
Furthermore, *Direkt Profil* could support students learning French. It could be used by teachers to supervise their students' progress systematically. However there are of course natural limitations in the use of software. To name one: *Direkt Profil* is not a style checker. It can never replace a teacher's experience in not only deciding what is wrong or right, but what makes a style of writing a sentence preferable to an alternative.

In a first attempt, *Direkt Profil* does not goal to detect all existing stage indicators. With future versions, the program's functionality will grow and also will the number of detected stage indicators.

[14] For a further discussion of Microsoft Word's spell check functionality see [Heidorn].

3.3 Using a software tool to detect stages

An important source of inspiration for *Direkt Profil* was *Rapid Profile* (presented first in chapter 1.4), which was developed at the University of Paderborn under the guidance of Manfred Pienemann.

Now, let us take a first glance at how *Direkt Profil* works.
The program is written as a client/server-solution. On the client side, a user can access the program through a normal web browser. She can type a French text and then send the text to the server for a detailed analysis. The server will colorize all the indicators in the original text, calculate some statistics, and send a result window back to the user. The user receives a detailed report about her stage in the language acquisition process.

The basic idea on the server side is to find as many indicators corresponding to the developmental sequences in a text. In the best case, all of the indicators underlying a text can be found. The information on what to find and how to find it exactly is encoded in a set of rules that were written before by a linguist. The linguist does not have to have any programming skills, but she needs knowledge of how *Direkt Profil* works internally at a high-level. (At this moment, there is no interface for writing the rules, thus a person additionally has to know the basics of XML and DTD, since all the rules are stored in a file in a XML-format.) The rules are written once and can be applied multiple times. They can be changed independently of the program. The independency of the rules from the program logic is a core concept of our approach. All the same, the program must be restarted after the rules were changed.

We can compare *Direkt Profil* to the other projects presented in chapter 1.4 to see, what makes *Direkt Profil* different from them and where it relies on existing ideas.

- ***Direkt Profil* and *PLNLP*:** *Direkt Profil* mainly differs from *PLNLP* (and the Epistle and Critique system) in its intention. *Direkt Profil* wants to analyze and profile the language acquisition process of a second language learner. *PLNLP* was developed to support the text authoring process. It aims to detect orthographic and grammar mistakes and gives support in the matter of style. *PLNLP* is thus not a CALL system by definition, but rather an application system (following the definitions of chapter 1.1).

- ***Direkt Profil* and *FreeText*:** As far as the author can see, *FreeText* can be categorized rather as a conventional CA(L)L-system (as introduced in chapter 1.1), whereas *Direkt Profil* rather belongs to the category of "semi-application CA(L)L-systems". *FreeText* is related to *Direkt Profil* in such a way that it aims to process authentic learner's input for the French language. This is a considerably larger challenge than dealing with non-learners' French, because the number of expected errors is higher on the second language learner's side. A further important distinction can be made concerning the chosen technical approach. *FreeText* uses a full parsing strategy to process a user's input, whereas *Direkt Profil* has implemented a partial parser in its core system. The difference between the two strategies will be pointed out more detailed in a later chapter.
- ***Direkt Profil* and *Granska:*** A difference between Direkt Profil and Granska is of course that *Granska* targets at the second language acquisition of Swedish, whereas *Direkt Profil* deals with French. For *Direkt Profil*, the most eye-catching aspect of *Granska* is the fact that the program aims to process second language learner's input and that not a full parser (as in *PLNLP* or *FreeText)* is employed, but a partial parser instead was chosen. The developers of the *Granska* system justify their approach with a higher robustness compared to full parsing, which, according to them, makes partial parsing a more suitable alternative if dealing with input texts with many errors.
- ***Direkt Profil* and *Dialang*:** *Dialang* has goals similar to *Direkt Profil,* but of course for many more languages than *Direkt Profil*: Determining the stage of a language learner. However, as far as the author can see, the system is not able to process any kind of freely written input text. Indeed the given exercises are very limited in the freedom of choice of possible answers. The given stage descriptions are taken from the CEF, which is rather a general framework and makes it hard to estimate the stage more precisely. *Direkt Profil* uses a much more detailed stage description.
- ***Direkt Profil* and *Rapid Profile:*** The two programs are related very closely from their linguistic intention. Rapid Profile however concentrates on spontaneously spoken (English or German) language and Direkt Profil on (French) written language. They both build up a language profile out of a given input sample. Both are based on detailed stage descriptions and thus encode linguistic knowledge in their program logic. Technically, they do of course differ completely in their implementation, because *Direkt Profil* makes use of NLP techniques to process written text.

As far as we know, no profiling software for written language similar to *Direkt Profil*'s goals exists nowadays.

4 Corpus and annotation

This chapter first presents the corpus of L2 French texts, which is being used as an input to *Direkt Profil*. It will be shown what is annotated during the text analysis and what annotation scheme the program uses. Besides the corpus, a few texts are manually annotated and build the standard *Direkt Profil* aims at. This excerpt of texts is discussed also. Finally, the program's French dictionary will be presented shortly.

In early days of computational linguistics, it was often the case that the available programs indeed were written in a very academic manner. Many of them failed to show correct behavior when fed with input not invented by the programmer herself but stemming from examples encountered in everyday life. Nowadays, researchers in the field of computational linguistics broadly agree that software applications must be able to handle authentic input. This is especially true for computer-assisted language learning systems, since such systems will have to deal with input from users who are still bound to make many mistakes, a person with very good language skills (e.g. a journalist or a translator writing in a foreign language professionally) would not produce anymore.

4.1 The CEFLE-corpus (Corpus Écrit de Français Langue Étrangère de Lund)[15]

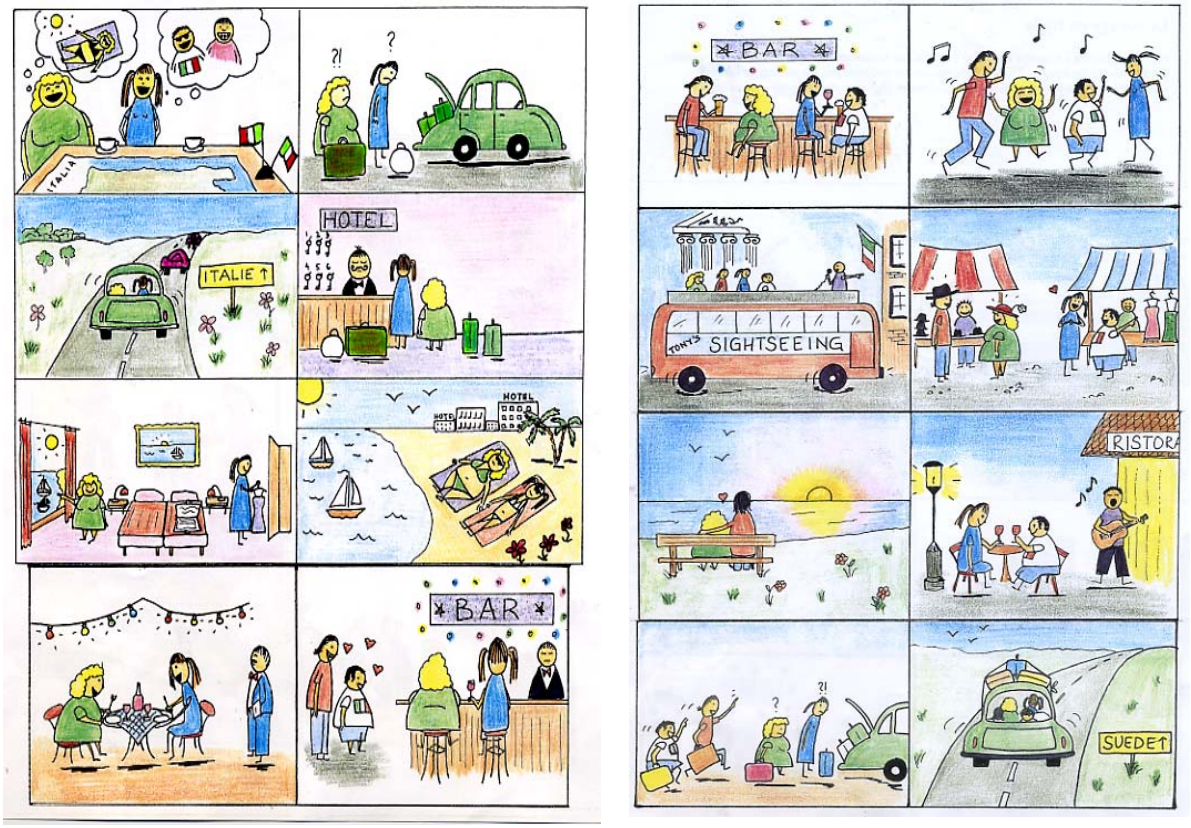
As the working material for the *Direkt Profil*-project we use an L2-corpus of approximately 100'000 words, consisting of short texts all written by 85 Swedish high school students who learn French as a foreign language (first foreign language for them is English). Additionally, the corpus contains texts of a control group of 22 French students of the same age. Many of the texts are the result of retelling a short story given in pictures to the student. The students had to write the text spontaneously on a computer, which means without too much preparation or reflection time. It was assured by the teacher that they did not have access to any word or grammar checking program and neither to the internet. The texts are therefore authentic examples of what can be expected as an input to our program. After being written, the texts were collected, annotated with some additional information about the text's subject and the

[15] The corpus was collected, prepared and brought into the format of XML by Malin Ågren (Department of Romance Languages, Lund University). We thank her for allowing us to use the corpus freely for the project.

writer and further information and finally brought in the format of XML. The corresponding DTD can be found in appendix A.

The corpus includes texts from stage 1 to stage 4 only as presented in the table in chapter 2.2. It does not contain the stages 5 and 6; only in the control group these stages might be represented.

The following is an example of such a story. It carries the title “Voyage en Italie”.



Picture 1 The picture story: "Voyage en Italie"

And here is what a high school student has written:

<SAMPLE SUBJECT_ID="Angelika">
<TEXT>
Elles sont deux femmes. Elles sont a italie au une vacanse. Mais L'Auto est très petite. Elles va a Italie. Au l'hothel elles demande une chambre. Un homme a le clé. Le chambre est grande avec deux lies. Il fait chaud. C'est noir. Cette deux femmes est a une restaurang. Dans une bar cet deux hommes. Ils amour les femmes. Ils parlons dans la bar. Ils ont tres bien. Le homme et la femme participat a un sightseeing dans la Rome. Ils achetons une robe. La robe est verte. La femme et l'homme reste au un banqe. Ils c'est amour. La femme et l'homme est au une ristorante. Les hommes va avec les femmes. L'auto est petite.
</TEXT>

```
    <INFO     TASK_NAME="VOYAGE_ITALIE"     GROUP_SUBJECT="MAIN"
    SUBJECT_LEVEL="1" SOURCE_SCHOOL="POLHEM"/>
</SAMPLE>
```

The text is formatted with a few XML tags to make it processable with an XML parser. There is an informational tag for every text containing some attributes:

- The `SUBJECT_ID` identifies the original writer of the text.
- The `TASK_NAME` identifies the general topic of the text. Common topics in our corpus are "*Voyage en Italie*", "*Ma famille*", "*L'homme sur l'île*" (a story about a man living on a small island) and "*Souvenirs d'un voyage*".
- The `GROUP_SUBJECT:` The group subject is set to either the value `main` or `control` to indicate whether it is a student from the main group (stage 1 to 4) or the control group (mother tongue).
- The `SUBJECT_LEVEL` is the stage in the school curriculum at which the student was studying when he or she wrote the text. This is not to be confused with the stages in table of stages and linguistic phenomena above (as defined by Bartning & Schlyter 2004]). A value, set to `CONTROL`, indicates that the text was written by a member of the control group of French students.
- The `SOURCE_SCHOOL` identifies the name of the school the students were studying at.

As can be seen easily, the text contains a number of mistakes of different types. There can be found missing accents and orthographical mistakes, wrong conjugations, missing or unnecessary words, wrong agreement between the article and the noun etc. Simple and short sentences are used, verbs are mostly inflected in present tense and active constructions are preferred to passive ones. Despite of all these facts, the written text clearly reflects the story in the pictures above. It is also evident that the author's French skills are located on a beginner's or at most on an intermediate stage. Of course we want to have a more precise estimation of the situation. For this purpose a deeper inspection can bring to light many indicators to estimate the stage of the author in the language acquisition process.

In the corpus, the texts themselves are not annotated to any further extent. We did not include any tags for the separation of noun, verb and other phrase structures, we did not mark the part-of-speech of words (this will be done later implicitly during the analysis process of our program) and neither did we tag language mistakes.

Besides the corpus, we maintain a test set of 25 manually, completely annotated texts. See chapter 4.5 for more information on this.

4.2 French language annotation

In all language annotation schemes available for European languages, tag specifications exist for a syntactical annotation of at least the noun and the verb phrases. Usually, the annotation schemes also provide further tags for prepositional, adjective and adverb phrases, and many schemes do differentiate to an even more detailed granularity of annotation. For French, there is an attempt to produce a standardized annotation scheme driven forth by the *Groupe Langues, Information et Représentations (LIR)*. They are working on bringing together several existing annotation standards for French[16]. This annotation scheme on the one hand should denote syntactic groups, but it should also show the relations between the groups, between words and between words and groups. For this purpose, mainly two sets of annotation tags are introduced: One set for the groups and another set to show dependencies and relations between the groups.

However, for fulfilling the goals of *Direkt Profil* such an annotation is not wholly adequate. To cite [Borin 2002]: "Interlanguage goes through a number of stages, terminating in a final (hopefully close) approximation of the target language. This has some implications for linguistic annotations of learner language production, whether in learner corpora (longer texts) or in analyzers of free learner language production in ICALL [Intelligent Computer-Assisted Language Learning] language exercises. Thus, part-of-speech (POS) tagging or parsing of learners' interlanguage may have to deal with categories absent from the canonical target language grammar as reflected in an LT [Language Technology] standard, etc., but which can be related […] to categories in the learner's native language [or others]." [Borin 2002]. He suggests that besides multiple possible annotations for the same linguistic object, there should also be annotations to show relations between those linguistic objects. An analysis could then link its own linguistic categorization or interpretation of a linguistic object to the object itself. "The linguistic categories provided by annotation standards would need to be different from the ones used by native speaker experts (which is arguably most often the kind of annotation aimed for now) if they are to be used for formulating feedback to language learners." [Borin 2002]

[16] The project homepage can be found under http://www.limsi.fr/Recherche/CORVAL/easy/.

The interest in *Direkt Profil* of what to annotate is not focused primarily on the annotation of syntactic groups but on the annotation of stage indicators. A stage indicator might fall together with the boundaries of a verb, noun or maybe adjective group, but it does not necessarily have to. A further problem in working with this standardized annotation is that *Direkt Profil* needs to connect the annotated groups to the developmental stage indicators. This is not foreseen by the *LIR*'s suggested annotation definition. As far as known to us, at the moment no such standard for CALL systems as suggested by [Borin 2002] exists. For these reasons, *Direkt Profil* uses its own annotation scheme.

4.3 Direkt Profil's annotation scheme

For the *Direkt Profil*-project we realized that the annotation for French language as described in the last chapter does not fit our purposes fully. On the one hand, we do not differentiate between phrase structures to the same level as the LIR organization provides the means to. On the other hand, we were in need to store information in our annotation scheme that is related to the indicators found during the language analysis process. If we want to show the user not only when the program has found a noun phrase or verb phrase structure, but also that for instance the verb phrase contains a "*passé composé* with a missing *accent aigu* on the past participle" then an annotation must be enabled to hold such information.

A further point we had to consider is the fact that the manipulation of character strings is an expensive task in a programming language in terms of computation time and processing power. We tried to avoid extensive manipulation of the original text by adding and taking a lot of annotation information never used later in the program and preferred to keep the relevant information mostly inside programming language structures (such as objects in terms of Object Oriented Programming). Of course, the annotation work still has to be done at least in the end of the analysis process when we want to produce a result and no logical annotation step is ever skipped. We simply keep this information inside the program logic until the end of the analysis process.

Let us now take a look at an analyzed and annotated sentence (which carries an erroneous agreement between the verb and the pronoun): **D'abord, j'a appelé mon frère*.

```
<span id="stringp1_t1">
```

```
…
D'abord,
<span class="p01_t01_c5210">
     <span title="pro:nom:sg:p1:invgen">j</span>
     <span title="unknown">'</span>
     <span title="ver:ipre:sg:p3">a</span>
     <span title="ver:ppas:mas:sg">appelé</span>
</span>
mon frère.
…
</span>
```

In this sentence a verb in the tense of *passé composé* can be found as a stage indicator. Furthermore, the main verb *a* does not agree in person (but only in number) with the pronoun *je*. In the sentence, the whole structure is marked with `<span> </span>` tags, containing a `class` attribute with the value `p1_t1_c5210,` which is equal to the name of one of our annotation classes. In our annotation ontology (presented in the next chapter), this value can be looked up and returns to be "A verb in *passé composé*. The conjugated auxiliary verb does not agree in person and number with the sentence's subject pronoun." Inside the stage indicator construct, all words are annotated with a `<span> </span>` tag, containing an attribute `title`. This attribute reflects the inflection information and the part-of-speech of the corresponding word.

Not all words are treated as holding relevant results of the analysis process (*D'abord*, *mon* and *frère*). These words are not annotated.

The whole text is surrounded by another pair of `<span>` tags with an attribute `id`. Roughly spoken, this tag differentiates whether what it contains is a (fixed) multi word expression to be detected or a stage indicator. We will not further discuss the details of this annotation tag here.

Furthermore in the output file, there is a tag for every annotation category connected to some styling information for the browser. (This is not shown in the example above.)

The annotation scheme should be seen rather as a compromise between traditional and "pure" linguistic annotation guidelines and technical goals, as to process the output with a common

(HTML-parsing) web browser. The advantage is that the used `<span>` tags belong to the XHTML standard 1.0 for web browsers[17]. Every modern browser can be expected to parse these spans correctly. The `<span>` tags are the only ones not connected to some default styling information in most browsers. That is why in a first attempt we decided to use them.
However, we are aware that such an annotation does not have much to do anymore with tagging as it is done usually in computational linguistics. We are thinking of alternative annotation schemes, which would more closely reflect the suggested standards for language annotation and on the other hand not be very expensive to process with a web browser.

4.4 An annotation ontology

There are roughly three different types of stage indicator groups in *Direkt Profil*: stage indicators belonging to a verb group, to a noun group or to an adjective group. Stage indicators belonging to verb groups had a higher priority than the other two stage indicator groups; this is why they were implemented before the others. For several technical reasons discussed further below, we decided first to build a prototype processing only verb group stage indicators. Treatment of noun and adjective group stage indicators will be added as functionality in a later iteration cycle.
With respect to the available functionality, the following informational aspects should be extracted from texts:

- **Tenses:** Check the tense of the verbs. The following tenses should be detected: *(Indicatif) présent, futur simple, imparfait, passé composé, plus-que-parfait*.
- **Mode:** Besides the tenses, also *conditionnel* should be detected.
- **Finite/infinite verbs:** Check for sentences where the main verb is infinite. *Infinitif* and *participe passé* should be detected.
- **"Composite" verb forms:** There are mainly two types of "composite" verb forms, either regular *passé composé*s and *plus-que-parfait*s or a (conjugated) auxiliary modal[18] verb together with an infinitive. Constructs like *laisser faire*, *savoir faire*, *devoir faire*… are common in French. They should be detected and counted. At the

[17] The W3C-consortium is the standardizing organization for web contents. See http://www.w3c.org for more details on XHTML 1.0.
[18] The expression "auxiliary modal verb" originally stems from German. In *Direkt Profil*, the term is used for the verbs *vouloir, pouvoir, savoir, devoir, faire, laisser, falloir* that often appear together with an infinitive (e.g. *Je dois rentrer à la maison.*) Although in French other "auxiliary modal verbs" exist as well, for instance *aller*, the current version of *Direkt Profil* does not treat them as such.

moment, the following list of auxiliary verbs should be treated explicitly: *vouloir, pouvoir, savoir, devoir, faire, laisser, falloir* + infinitive.

- **Agreement:** If a sentence contains a conjugated verb, the program should check whether it agrees in person and number with the sentence's subject.

Out of these categories, many combinations can be deduced. For example, a sentence could contain a "composite" verb form, where the conjugated verb does not agree with the sentence's subject, like **Ils doit apprendre pour l'examen*. Sometimes, certain combinations overlap each other. A *plus-que-parfait* can be seen as both a *plus-que-parfait* but also as a conjugated *imparfait* of *être* or *avoir* followed by a *participe passé*. In these situations, it is generally a good guideline to count the most complex structure, although we might not have kept the guideline always.

We took into account certain kinds of language errors. Very common to French are forgotten or misplaced accents (like writing **ecouter* instead of *écouter*). Sometimes, words are derived wrongly (**prendu* instead of *pris*). Counting such mistakes can be combined with the categories above too.

This "annotation ontology" shows what effectively will be detected and accounted. Lines marked with a bold font are summing up several atomic or already summed up phenomena.

Level	name	Counter ID
0	***Is there a verb at all?***	
0.1	The sentence contains no verb.	p1_t1_c0000
0.2	The sentence contains a verb with a misspelled or forgotten accent.	p1_t1_c1000
1	***Is the verb finite or infinite?***	***p1_t1_c4000***
1.1	A common[19], not conjugated verb.	p1_t1_c4100
1.2	**A common conjugated verb.**	**p1_t1_c4200**
1.2.1	A common conjugated verb that does not agree in person and number with the sentence's subject pronoun.	p1_t1_c4210
1.2.2	A common conjugated verb that agrees in person and number with the sentence's subject pronoun.	p1_t1_c4220
2	***Simple verb tenses and modes***	***p1_t1_c5000***
2.1	**Total number of verbs in *présent*.**	**p1_t1_c5100**
2.1.1	**A conjugated form of être or avoir in *présent* (total).**	**p1_t1_c5110**
2.1.1.1	A conjugated form of être or avoir in *présent* that does not agree with the sentence's subject pronoun.	p1_t1_c5111
2.1.1.2	A conjugated form of être or avoir in *présent* that does agree with the sentence's subject pronoun.	p1_t1_c5112
2.1.2	**An auxiliary modal verb in *présent* (total).**	**p1_t1_c5120**
2.1.2.1	An auxiliary modal verb in *présent* that does not agree in person and number with the sentence's subject pronoun.	p1_t1_c5121

[19] A "common verb" in this context means that the verb stems neither from the auxiliaries avoir or être nor from one of the "auxiliary modal verbs": *vouloir, pouvoir, savoir, devoir, faire, laisser, falloir.*

2.1.2.2	An auxiliary modal verb" in *présent* that agrees in person and number with the sentence's subject.	p1_t1_c5122
2.1.3	**A common (lexical) verb in *présent*. (total)**	**p1_t1_c5130**
2.1.3.1	A common verb in *présent* that does not agree in person and number with the sentence's subject pronoun.	p1_t1_c5131
2.1.3.2	A common verb in *présent* that agrees in person and number with the sentence's subject pronoun.	p1_t1_c5132
2.2	**Total number of verbs in *passé composé* (total).**	**p1_t1_c5200**
2.2.1	A verb in *passé composé*. The conjugated auxiliary verb does not agree in person and number with the sentence's subject pronoun.	p1_t1_c5210
2.2.2	A verb in *passé composé*. The conjugated auxiliary verb agrees in person and number with the sentence's subject pronoun.	p1_t1_c5220
2.2.3	**A verb in *passé composé*. The *participe passé* has a misspelled or forgotten accent (total).**	**p1_t1_c5230**
2.2.3.1	A verb in *passé composé*. The conjugated auxiliary verb does not agree in person and number with the sentence's subject pronoun. The *participe passé* has a misspelled or forgotten accent.	p1_t1_c5231
2.2.3.2	A verb in *passé composé*. The conjugated auxiliary verb agrees in person and number with the sentence's subject pronoun. The *participe passé* has a misspelled or forgotten accent.	p1_t1_c5232
2.2.4	**A verb in *passé composé*. The *participe passé* was derived wrongly (total).**	**p1_t1_c5240**
2.2.4.1	A verb in *passé composé*. The conjugated auxiliary verb does not agree in person and number with the sentence's subject pronoun. The *participe passé* was derived wrongly.	p1_t1_c5241
2.2.4.2	A verb in *passé composé*. The conjugated auxiliary verb agrees in person and number with the sentence's subject pronoun. The *participe passé* was derived wrongly.	p1_t1_c5242
2.3	**An auxiliary modal verb followed by an *infinitif* (total).**	**p1_t1_c5300**
2.3.1	An auxiliary modal verb followed by an *infninitif*. The conjugated auxiliary modal verb does not agree in person and number with the sentence's subject pronoun.	p1_t1_c5310
2.3.2	An auxiliary modal verb followed by an *infninitif*. The conjugated auxiliary modal verb agrees in person and number with the sentence's subject pronoun.	p1_t1_c5320
2.4	**Total number of verbs in *imparfait* (total).**	**p1_t1_c5400**
2.4.1	**A conjugated form of être or avoir in *imparfait* (total).**	**p1_t1_c5410**
2.4.1.1	A conjugated form of être or avoir in *imparfait* that does not agree with the sentence's subject pronoun.	p1_t1_c5411
2.4.1.2	A conjugated form of être or avoir in *imparfait* that agrees with the sentence's subject pronoun.	p1_t1_c5412
2.4.2	**An auxiliary modal verb in *imparfait* followed by an *infinitif* (total).**	**p1_t1_c5420**
2.4.2.1	An auxiliary modal verb in *imparfait* followed by an *infinitif*. The conjugated auxiliary modal verb does not agree in person and number with the sentence's subject pronoun.	p1_t1_c5421
2.4.2.2	An auxiliary modal verb in *imparfait* followed by an *infinitif*. The conjugated auxiliary modal verb agrees in person and number with the sentence's subject pronoun.	p1_t1_c5422
2.4.3	**A common (lexical) verb in *imparfait*. (total)**	p1_t1_c5430
2.4.3.1	A common verb in *imparfait* that does not agree in person and number with the sentence's subject pronoun.	p1_t1_c5431
2.4.3.2	A common verb in *imparfait* that agrees in person and number with the sentence's subject pronoun.	p1_t1_c5432
2.5	**Total number of verbs in *futur simple* (total).**	**p1_t1_c5500**
2.5.1	A conjugated form of être or avoir in *futur simple*.	p1_t1_c5510
2.5.2	An auxiliary modal verb in *futur simple.*	p1_t1_c5520
2.5.3	**A common (lexical) verb in *futur simple*. (total)**	p1_t1_c5530

2.5.3.1	A common verb in *futur simple* that does not agree in person and number with the sentence's subject pronoun.	p1_t1_c5531
2.5.3.2	A common verb in *futur simple* that agrees in person and number with the sentence's subject pronoun.	p1_t1_c5532
3	***Total number of verbs in advanced tenses/modes (total).***	***p1_t1_c6000***
3.1	**Total number of verbs in *plus-que-parfait* (total).**	**p1_t1_c6100**
3.1.1	A verb in *plus-que-parfait*. The conjugated auxiliary verb does not agree in person and number with the sentence's subject pronoun.	p1_t1_c6110
3.1.2	A verb in *plus-que-parfait*. The conjugated auxiliary verb agrees in person and number with the sentence's subject pronoun.	p1_t1_c6120
3.1.3	**A verb in *plus-que-parfait*. The *participe passé* has a misspelled or forgotten accent (total).**	**p1_t1_c6130**
3.1.3.1	A verb in *plus-que-parfait*. The conjugated auxiliary verb does not agree in person and number with the sentence's subject pronoun. The *participe passé* has a misspelled or forgotten accent.	p1_t1_c6131
3.1.3.2	A verb in *plus-que-parfait*. The conjugated auxiliary verb agrees in person and number with the sentence's subject pronoun. The *participe passé* has a misspelled or forgotten accent.	p1_t1_c6132
3.1.4	**A verb in *plus-que-parfait*. The *participe passé* was derived wrongly (total).**	**p1_t1_c6140**
3.1.4.1	A verb in *plus-que-parfait*. The conjugated auxiliary verb does not agree in person and number with the sentence's subject pronoun. The *participe passé* was derived wrongly.	p1_t1_c6141
3.1.4.2	A verb in *plus-que-parfait*. The conjugated auxiliary verb agrees in person and number with the sentence's subject pronoun. The *participe passé* was derived wrongly.	p1_t1_c6142
3.2	**Total number of verbs in conditionnel (total).**	**p1_t1_c6200**
3.2.1	A conjugated form of être or avoir in *conditionnel*.	p1_t1_c6210
3.2.2	An auxiliary modal verb in *conditionnel.*	p1_t1_c6220
3.2.3	**A common (lexical) verb in *conditionnel* (total)**	**p1_t1_c6230**
3.2.3.1	A common verb in *conditionnel* that does not agree in person and number with the sentence's subject pronoun.	p1_t1_c6231
3.2.3.1	A common verb in *conditionnel* that agrees in person and number with the sentence's subject pronoun.	p1_t1_c6232
3.3	A conjugated form of être or avoir in another category (e.g. *passé simple, subjonctif* etc.).	p1_t1_c6300

Table 3 The annotation ontology

As can be seen, the current version of *Direkt Profil* tries to identify around three dozens of different (atomic) phenomena. Those phenomena actually correspond to a high extent with many of the verb group stage indicators. In the right column, the internal ID of each phenomenon to be detected is shown. These IDs will be used also to annotate a text.

4.5 The "Gold Standard"

To check the analysis quality (mainly expressed in precision and recall of the analysis) *Direkt Profil* is able to produce, we were in need of a "reference standard" or control group of texts. We call that the "Gold Standard", because in an ideal case the program produces exactly the

same results as given by the standard. Besides our corpus, we maintain a group of 20 texts to be used as a reference standard plus 5 further texts. The 20 texts (5 texts for stages 1, 2, 3 and 4, but no texts for stages 5 and 6) are written by Swedish second language learners. The remaining 5 texts are, as in the corpus, written by French students. They serve as a means for cross-checking.
The texts were first annotated by the program and after this procedure corrected manually to ensure the correctness of the annotation and to reach a complete and error free reference standard.

We strictly separated the texts from the normal corpus as an input to the program for development and testing of the program and the reference standard of texts on the other hand for generating statistical measures about the program's detection qualities. In the best case, *Direkt Profil* will find all the annotated stage indicators in the Gold Standard texts but not more, in the worst case the program does not find any of the manually annotated indicators but produces a lot of overdetection. The reference standard is reused and updated regularly for every updated program version. The procedure ensures to detect a possibly occurring regression between two following program versions.

4.6 The French Dictionary

During the process of analyzing a text, our software has to find out for each encountered word its actual inflection, part-of-speech or lemma information. The program needs a source providing it with the corresponding information. *Direkt Profil* relies on a French full text dictionary with an estimated number of 300'000 words. The dictionary used by our project is freely available under the distribution of *l'Association des Bibliophiles Universels (ABU)*. It can be downloaded from http://abu.cnam.fr.
The dictionary contains all common French words. Names, multi word expressions (e.g. "*La Grande République*"), abbreviations and the like are not included.
The dictionary stores for each word:

- The word itself,
- The word's part-of-speech (noun, adjective, verb, determiner etc.),
- All different possible inflections of the word,
- The word's lemma.

For our own purposes we also added

- The word's stem, but at the moment only for verbs.

For regular verbs, we could generate the stems automatically by cutting off the word's lemma's ending. The stems of a list of irregular verbs were added manually.

During our work with the ABU-dictionary, we detected a couple of inconsistencies in the dictionary. Some words seemed to have wrong inflection information, some important words were forgotten in the dictionary. Sometimes the dictionary used different shortcuts for the same concept. In a process of cleaning up the incongruences, we gave the dictionary a new look and put the entries and their information inside XML-tags. The dictionary can now be processed by using one of the many wide spread XML-parsers, such as SAXParser in the Java programming language. An entry now looks like the following:

```
<wordform entry="écouterais" lemma="écouter" pos="ver">
     <stem>écout</stem>
     <feature>cpre:sg:p1</feature>
     <feature>cpre:sg:p2</feature>
</wordform>
```

It tells us that the word *écouterais* is a verb, which has *écouter* as its lemma. There are 2 features named: The word can be either a 1st person singular *conditionnel présent* or a 2nd person singular *conditionnel présent*. And last, it tells us that this entry's stem is equal to the character string *écout*.

For certain words, which could occur in a sentence as a subject pronoun (like *je, tu, il, elle, on nous* etc.) we added the abbreviation 'nom' (for 'nominative pronoun'), whereas for the other pronouns (like *moi, toi, lui* etc.), which never appear as a sentence's subject pronoun, 'nnom' was used instead.

The exact DTD of the dictionary can be found in appendix A.

5 How Direkt Profil works

We have now broadly discussed the goals, the linguistic theory and the ideas underlying the *Direkt Profil* project. We will shortly summarize what we know already about *Direkt Profil* before we turn to the more technical aspects of the software implementation in this chapter.

The project's goal is to build a software tool that incorporates the linguistic knowledge about the progress of the language acquisition process in written learner's French. There is a set of phenomena, the indicators, which may be inspected in a text to gain information about the developmental stage of a language learner. The program crawls through a given input text trying to detect as many linguistic indicators as possible to decide upon the stage of the writer. Let us look on how this is done in *Direkt Profil*.

5.1 Full parsing or partial parsing? A discussion.

In chapter 4.4 we selected a subset of all indicators we want to be able to detect in a text. In that chapter we still neglected the question of how a complex indicator structure as a *plus-que-parfait* (possibly even negated) could ever be detected. This question is our topic now.

Generally spoken, *Direkt Profil* uses a strategy of **partial parsing** (or sometimes also referred to as a **shallow parsing** or **chunking**) of natural language. Partial parsing, in contrary to **full parsing** (or also called **deep parsing**), does not try to build up a complete syntax tree of a sentence.

Full parsing treats sentences as a whole. It has access to a set of usually recursive grammar rules. The grammar rules might be written in their own specialized language. Depending on the strategy the full parser follows – usually there can be called the two: top-down parsing and bottom-up parsing – it tries to produce a syntax tree for each sentence. There exist different algorithms for each of the two approaches, for instance the relatively easy to implement Shift-Reduce Algorithm as an example of a bottom-up parser. With help of the syntax tree the sentence can be split into phrases, which are then annotated by the parser with special tags. According to [Nugues 2004], most annotation schemes use tags for at least noun phrases, verb phrases, adjective phrases, adverbial phrases and prepositional phrases.

Sometimes, several syntax trees can be built from the same sentence. Nowadays, in such situations full parsers use statistical probability rules to decide, which alternative has the highest probability to appear in a text. The corresponding probability measures are preproduced from a manually annotated corpus, the so called treebank. The mathematical product of all probabilities of all applied grammar rules in a sequence returns the total probability of a single parsing solution.[20]

Full parsers, however advanced they might be, have some serious drawbacks. In general, a full parser expects an input sentence to be already free of language errors. If the input sentence contains serious grammatical mistakes, the parser will not be able to match it against any sequence of grammatical rules, it has stored. To easen the burden, most parsers follow a strategy of relaxing constraints given by the grammar if a first attempt to parse the sentence has already failed. This means, after having relaxed the grammatical constraints, the parser will accept slightly wrong input also and guess what the user probably aimed to express. If after a first relaxation the parser still cannot come up with a solution, it will continue relaxing constraints until at least one possible solution can be generated.
Error detection in full parsers follows this strategy. Partial trees are built as far as possible. If there is an error in the text, the probability is high that it is located at a position where two subtrees do not fit together. The program tries out whether with a slight variation (the relaxation of constraints) of the original input sentence again a meaningful result could be produced. Often, there are different possible solutions for the same case and the alternative with the highest probability is computed and selected.

Such a strategy of relaxed constraints is applied in the *FreeText*-project [Vandeventer & L'Hair 2003]. This approach has proven to give good results for input with none or a relatively limited number of errors. Difficulties arise when we deal with authentic learner's French, especially in early stages. Often authentic learners' texts contain many mistakes of all types, so that a full parser will have serious trouble to still process the input. According to [Vandeventer 2003], to give a more concrete example, the recall of certain types of language mistakes detected by the *FreeText*'s full parser falls down to 30%.
Another drawback of full parsers are the relatively high costs to build complex grammar structures, if they should work for a broad spectrum of input sentences.

[20] A more detailed discussion of full parsers can be found for instance in [Nugues 2004].

Partial parsing is a strong alternative to full parsing. In many cases, it is questionable whether a syntactic parse tree is really necessary to meet the chosen goal. Instead of building such a complete syntactic tree, partial parsing only parses parts of the sentence. The rationale behind a partial parser is that it is often sufficient for the application to parse only certain kinds of groups in a sentence, for instance only noun and verb groups, instead of parsing them all. Thus, it implements a strategy focusing on local information only, which limits the analysis' complexity. As a consequence of locality, the parser will not fail to process subsequent groups by cascading eventually occurring errors from precedent groups. The advantage of a partial parser clearly lies in its **robustness** which, as we have seen above, is one of the requirements for CALL systems.

Furthermore a partial parser naturally supports the idea of being developed incrementally. Whereas a full parser more or less needs a complete grammar structure already to work correctly for most sentences, partial parsers can be extended easily step by step, concentrating on parsing only a few and simple structures in the beginning – for example processing only verb groups – and adding more and more functionality to it (noun groups, adjective groups etc.).

An example of a partial parser used for processing authentic second language learner texts can be found in the *Granska* project for Swedish language [Knutsson et al. 2003].

5.2 The analysis engine – a partial parser

In *Direkt Profil*, our object was to be able to detect different linguistic phenomena, which serve as indicators for the progress of the language acquisition process. In the chapter 2.2 we introduced around two dozens of such stage indicators sequences. Many of the stage indicators either fall in the category of a noun group or a verb group (if we understand the term "group" in a broader sense than the usually relatively narrow view of it). For this reason, we decided to first implement a partial parser that should be able to detect and process verb group-like structures and in a next step to extend the same parser to detect and process noun groups also. We just recently finished the parser for verb groups, and we are now about to start with the implementation of noun groups.

Our parser – in *Direkt Profil* we often refer to it as the **analysis engine**, for it is the responsible unit to perform the analysis of a text – is written in pure Java. It relies to an

important extent on the regular expressions functionality introduced by Sun Microsystems in Java v1.4 (therefore the parser's source code cannot be compiled with an earlier version of Java's Software Development Kit).

The parser cascades through five different levels of processing:

Input: Raw sentence

↓

1. Tokenization
↓
2. Detection and annotation of certain multi word expressions
↓
3. Determination of the part-of-speech and inflection information
↓
4. Detection and annotation of groups/indicators
↓
5. Collection of the results + generating statistics
↓

Output: Annotated sentence + counters

0) Input

The input to the analysis engine is expected to be a single character string in the format of ISO-8859-1, since French as a language contains many special characters as the *cedil* (*ç*), different accents (é, è, ê), the trema (*ë*) and others.

The input text does not contain any special annotation at this point in time. The character String is sent from the browser to the server where it is processed.

1) Tokenization

During the tokenization step the input text is split up into tokens. We differentiate between 3 different kinds of tokens: delimiters, words and non-words. From a processing point of view a token is nothing more than a (non-empty) character string.

- A **word** corresponds to the common denotation of the term. A word is mostly a token that consists of all letters of the French alphabet. In the current implementation, it may not contain special characters. A word like *aujourd'hui* will thus be split into the three tokens [*aujourd*], the apostrophe ['] and [*hui*].

- A **non-word** is a token that is not a real word in a language but neither is it a group delimiter. Non-words are encountered often in written texts. A number, written in digits, (*123456*), a hyphen (-) or the dollar sign *$* are examples of non-words.
- A **delimiter** is a token that separates groups and/or sentences, for instance the French words *et* and *mais* or punctuation signs as question marks, exclamation marks, full stops, colons, commas or semicolons.

Direkt Profil uses regular expressions to split the text into tokens. Corresponding to the 3 types of tokens, there exist 3 types of tokenization rules. The rules are saved independently from the program logic itself in a special file – the rule file – in XML format. They are loaded dynamically at the software's startup time. This is an example of a delimiter tokenization rule:

```
<delimiter_tokenize_rule>
    <regex>[.!?;:]|\b(mais|et)\b</regex>
</delimiter_tokenize_rule>
```

If an input character string matches the rule's regular expression, the token will be extracted and be set as a delimiter token.

During the analysis process, first the word tokenization rule splits the sentence into word tokens. Second, the delimiter and the non-word tokenization rules are both applied to the tokens. All tokens are stored in an array of token objects. Each token thus can be identified by its position in the token array, numbered starting from 0 for the first token until the end of the array. Each token object stores information about its own state: the position in the text, whether it is a delimiter, a non-word and further information. An example of the tokenization of the sentence *Je pense, donc je suis!* is: [*Je*] [*pense*] [,] [*donc*] [*je*] [*suis*] [*!*].

There have been abundant discussions in the past about the matter of what to treat as a word and how word boundaries are to be set. We will not discuss the topic any further here.

2) Detection and annotation of certain multi word expressions

After the tokenization, certain multi word expressions are to be discovered. Some multi word expressions we want to detect are for instance *il y a, c'est, je m'appelle* and others. A multi word expression is an N-Gram of words ("a few words which are logically grouped together") that form a fixed expression. Some multi word expressions are so common to our daily language that we even do not recognize them as such: *The White House, European Union, La Grande République* etc. Multi word expressions play an important part in the language acquisition process since they are learnt as a whole and not as single, combined words.

Therefore, an expression like *je m'appelle* must be excluded and treated separately from the further analysis process since it should not be counted as one of the normal stage indicators.

3) Determination of the part-of-speech and inflection information

Third, the part-of-speech of each token is detected. This step cannot be clearly separated from step 4) since both steps are done simultaneously. During the analysis process, every word is looked up in the dictionary to find its part-of-speech and all possible inflections. The information is not really inserted in the original text string with tags, but stored inside different program objects. Often, there are several different suiting possibilities for the part-of-speech/inflection and the word's syntactical context in the sentence has to be taken into account to decide, which one should be chosen. Many words that appear as past participles exist also in the usage of an adjective or sometimes also as a noun. To give an example: The French word *aimé* can appear as a noun, an adjective or a past participle, all of them with the inflection information "masculine + singular". A first loop through the text by the analysis engine follows a "positive" strategy: As soon as it encounters a word that is marked with at least one matching part-of-speech in the dictionary, the engine treats this information as given. It does not check further at this point in time the syntactical context. Later on in the analysis process, a second check might be done to detect the agreement between a pronoun and a verb form. Then, at least some ambiguities can be eliminated.

However, the problem with undissolvable ambiguities does not occur as frequently as one might expect, what minimizes the grade of severity of the problem. The reason is that we mostly search for different verb forms only and relatively seldom for words with other part-of-speech information. At the moment, the program actually does never search for adjectives or nouns, only pronouns should be detected besides verbs. Detection of stage indicators inside noun groups will be a feature of a future program version.

4) Detection and annotation of groups/indicators

This is the most complex of the steps. Together with the part-of-speech and inflection information the analysis engine tries to identify the stage indicators hidden in the text. In chapter 4.4 we introduced an annotation ontology to show what should be detected as a stage indicator by our program. This ontology must be brought in a form in which it can be implemented into the program. *Direkt Profil* contains a set of rules, all written in XML, which

all together encode the ontology. The set of rules is organized as a (binary) rule tree[21]. It embodies linguistic information implicitly in the form of the stage indicators, but also holds processing information the program needs. The rule tree is indeed a mixture between procedural processing instructions and linguistic knowledge.

The rules are independent of the program and loaded at startup time, so they can be changed without further programming (a good understanding of XML/DTD and how *Direkt Profil* works internally is necessary all the same).

Direkt Profil's analysis engine goes through the text by applying one rule after the other to it. Since every rule is actually a node in the binary rule tree, every rule (except the terminal rules) has exactly two child rules. Always after having applied a rule, the software decides which of the child rules should be applied next. According to this behavior, a path inside the tree is followed from the root rule through many different rules in between till a terminal rule is met. When arrived at a terminal rule, a stage indicator or at least parts of it are found along the navigation path through the rules or the inspected construct in the sentence is skipped. If a stage indicator is found, some annotation information is added to the sentence. As seen in the anotation ontology, the annotation contains an identification code of the stage indicator and also the part-of-speech and inflection information for every word inside the stage indicator group. Furthermore, a special counter is incremented to keep track of the occurrence of such a stage indicator. In the end of the analysis, the learner's stage will be computed[22] based on the counters.

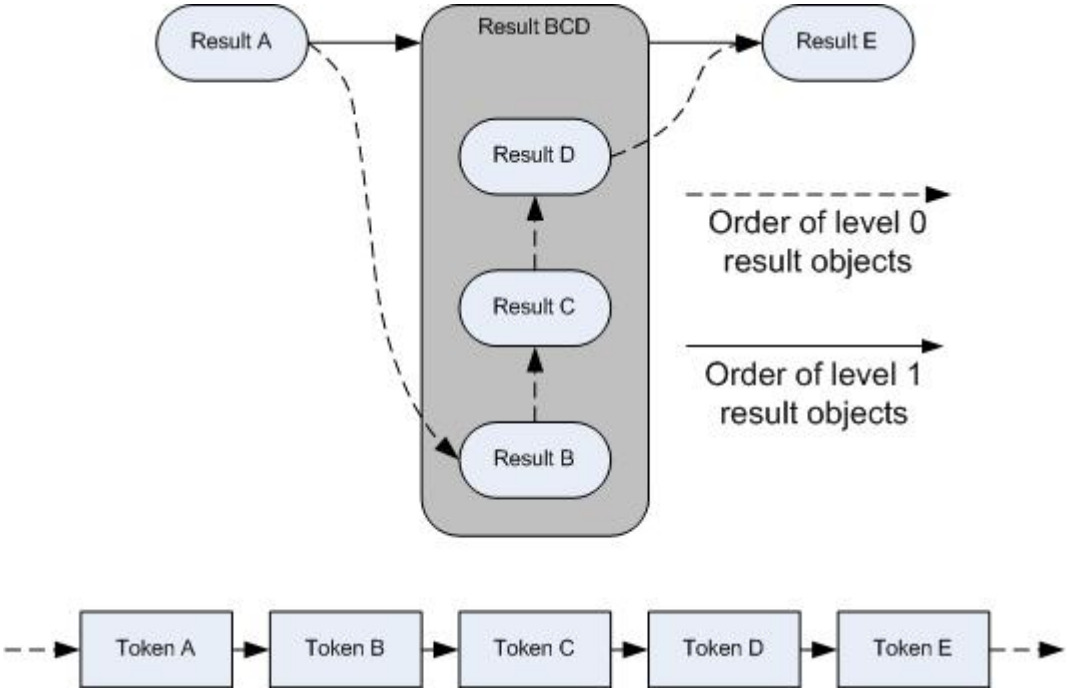
5) Collection of the results

As a fifth step, it is necessary to merge all the different parts again. Some tokens are marked as fragments of multi word expressions, some tokens as stage indicators, but most tokens are not annotated at all. They all need to be brought together again. The mechanism to accomplish this task is quite complicated and we will only take a superficial look here. Roughly spoken, during the analysis process a "chain of result objects" is built up internally. This **result chain** represents both the original tokens and their annotation tags. A result object in the result chain can hold another result object recursively. In the picture, result object BCD holds result objects B, C and D. In this way, it is possible to introduce different "levels of results". For example, a token can belong to a result object that holds its part-of-speech and annotation

[21] To be more precise: The rule tree does not suffice the classic definition of a binary tree. It is indeed rather a directed, circle-free graph structure where every node may have not more than 2 child nodes. But since we allowed leaves in the tree to belong to more than one parent node, the definition of a classic tree is hurt. Without this violation in the leave nodes it would though be effectively a binary tree.

[22] However, in *Direkt Profil* v1.5.1 no such functionality is yet implemented.

information. At the same time, the result object itself can belong to another result object of a higher level which represents, let us say, a stage indicator.



Picture 2 The result chain

One could imagine the following tokenized sentence: [*aaa*][*c*][*'*][*est*][*eee*] that contains a the multi word expression *c'est*. The analysis engine invents a level 0 result object (the result objects A, B, C, D and E) for every parsed token (the token objects A, B, C, D and E). Furthermore, the analysis engine detects that token B, C and D together build a multi word expression. It groups them together in a new level 1 result object – the result object BCD. The references between the result objects are reset. Later on, when the program wants to detect stage indicators in step 4, it can reuse the information provided by the level 1 objects and skip the analysis for every object already grouped in another higher level result object. Finally, all the result objects are merged in a single annotated character string, which can be sent back to the client.

The whole result chain concept aims on the task of collecting together what has been processed separately. The current version of *Direkt Profil* is not yet enabled to inspect dependencies between stage indicator groups, for instance between noun group stage indicators and verb group sage indicators. This is planned for a future version of *Direkt Profil*. According to our implementation concepts, the rule chain shall serve as the base for dealing with such dependencies.

Something completely else that must be done during this step also: Generating statistics and computing the language learner's stage! For this purpose, the counters can be used. Independent from the specified rules, there are a few more standard counters incremented automatically by the program. They count the number of tokens, the number of unknown words (words not contained in the dictionary) etc. However, at the current version of *Direkt Profil* v1.5.1 no statistical functionality is implemented yet. This is also an important task to be done for a future version. Therefore, we will skip this point for the moment.

6) Output

In the end, the annotated text is embedded in a regular (X)HTML[23]-enabled web page, which can be sent to every browser. The user should be enabled to toggle on and off highlighting of stage indicators in the text. For this purpose, JavaScript commands and styling information are added to the webpage also. The user receives back an interactive browser readable webpage where she interactively can inspect the analyzed text.

After having seen how the analysis engine works generally, some points need to be emphasized here:

- As it should be clear from the description above, the analysis engine – since it is a partial parser – focuses on processing indicator groups merely than whole phrases/sentences. In our eyes, it is not necessary to build a complete syntactic tree, and we believe that the robustness of partial parsers is an advantage if dealing with early language learners.
- Therefore, it is also clear that certain types of error diagnosis, especially between different sentences cannot be reproduced by our partial parser. This is however not necessary for the limited purposes of our project. We want to detect language acquiring stages and not primarily do error detection/correction.
- Our partial parser always tries to extract the longest matching indicator construct, running from the beginning to the end in a sentence. In special cases, this behavior might lead to confusion if the beginning of one indicator group could be interpreted as the end of another indicator group. This problem is very hypothetical and rarely or never encountered during the tests we did with the program.
- Many parsers, be it full or partial parsers, work with a strategy of relaxing constraints of the parsing rules, if the parser is not able to produce a correct result. In *Direkt*

[23] See http://www.w3c.org/TR/2004/WD-xhtml2-20040722/ for more information about XHTML.

Profil, two strategies of relaxing constraints can be named: The strategy of searching for forgotten or wrong set accents (the **accent search**) and the strategy of wrong derived forms for past participles of verbs (the **stem search**). Both of them will be discussed in further detail below.
An often encountered strategy of relaxing constraints is the phonological reinterpretation, as used for example in the *FreeText* project. No such functionality is implemented or foreseen in the *Direkt Profil* project.

Direkt Profil is still under construction. It is to be expected that the analysis engine will go through several minor or major changes in its design. The precise behavior and design of the analysis engine will the topic in the next chapters.

5.3 The text analyzing process

The more general frame of how *Direkt Profil*'s analysis engine works was discussed in the last chapter. It is now time to turn to the analysis engine's core algorithm.

The analysis engine can be imagined as a machine that is being moved mostly forth and sometimes back over a long array of small boxes. In every box there is exactly one token. The machine can inspect the contents of the box and perform certain manipulations to it, for instance put a label onto the box. In terms according to computational linguistics, this would be the annotation of a token. To put a label onto the box the machine has to decide what is inside the box. For this purpose it can inspect the token, send some information to a database it has access to and the database will return information about the token – in *Direkt Profil* usually the part-of-speech and inflection information of a token. The database is nothing else than a dictionary with around 300'000 French words.
The machine is steered by a simple program: the rules or the rule tree as we call it. The program tells the machine what it should look for, and it also gives instructions about what steps should be taken if it successfully finds something matching inside the boxes or not. The machine therefore receives from the steering program the criterion what to search for and it further receives the inspection report from the database. It compares both and decides whether the token inside the box matches the criterion or not.

The steering program is conceived as an endless loop: Every time the end of the program is reached, it simply starts again and moves further to the next box until it has finally reached the last of all boxes. This means that *Direkt Profil* applies the root rule in the rule tree again and again to the tokenized input text until at least one token matches the root rule's search criterion.

As soon as a token is found in a box that matches the steering program's criterion, the steering program commands the machine to mark this box temporarily with a pointer. The pointer will later on be removed from the box but at the moment it is needed to recognize the position. The machine now establishes a "frame" over the next few boxes to inspect them deeper. Simultaneously, the steering program loads the next few instructions with a new search criterion (loading the next rule in the rule tree). The just now loaded instructions will be applied to one box after the other in the frame until again at least one correspondence is found between the in the database looked up information about the box's token and the steering program's new search criterion. If this is the case, then again a new frame is established, new instructions are loaded and the machine moves on inspecting boxes.

This pattern can be repeated several times until the steering program does not have any more new instructions to be loaded for the current situation. In *Direkt Profil* this situation is met after processing a terminal rule in the rule tree. In this case, the machine first labels all boxes inside the earliest established frame, increments some special counters, takes away all the remaining frames, sets the pointer to the current position and finally the machine starts all over again with the first instructions in the steering program and a new box.

Sometimes, it is necessary to re-inspect the contents of an earlier box to check whether the current box contains a token that agrees with the other token in some criteria. For this purpose, the machine accesses a small memory of the last few inspected boxes. It saved all the relevant information about the inspection process and its result in the memory. It can even return to an earlier box and change the label of that box again if necessary.

In *Direkt Profil* the machine is of course our analysis engine. The analysis engine runs through an array of token objects targeting to find the next stage indicator. The steering program as written above corresponds to the rule tree that contains both procedural instructions for the analysis engine and implicit linguistic knowledge. For instance, if we want to detect a French *passé composé,* how can this be done? A *passé composé* can be seen as the combination of a verb in indicative present tense of either the auxiliary verb *être* or *avoir* and a past participle of any other verb (and possibly following a pronoun or a noun in a sentence).

Furthermore, the *passé composé* can be negated (*je n'ai pas mangé*) and there can be words inserted in between the second part of the negation …*pas*… and the past participle (*je n'ai pas encore mangé*). The basic idea of how to detect a *passé composé* as a stage indicator is:

1. Look for a pronoun,
2. Try to find an auxiliary verb of *être* or *avoir*,
3. Check whether the auxiliary verb agrees in person and number with the pronoun,
4. Try to find a past participle of a verb.

This algorithm of how to detect a *passé composé* can be understood as a sequence of four separated rules or instructions piled together. They all have some common characteristics:

- They all search for something (we can say that rule 3 searches for an agreement). Thus, they all can produce basically two results. Either the search criterion is fulfilled which means that something was found – we will refer to this as a **match** – or the search criterion is not fulfilled and nothing was found during the application of the rule – to be called a **no match**.
- They all rely on the result from a precedent rule (except the first rule), since a rule will never be applied if the precedent rule was not successful in finding what it was searching for. For instance: If an *indicatif présent* tense of the verb *être/avoir* cannot be found, there is for sure no *passé composé* in a sentence. The program should not waste time then with trying to find a past participle.
- If a rule is applied, be it successfully or not, then the program should keep track of the information somehow. Every time when a *passé composé* is detected in the text the program should annotate it and remember that such a stage indicator was found.

The result of the attempt to combine both the annotation ontology and these rather vague ideas of how a basic algorithm to detect an indicator in a text would look like, was the first prototype of *Direkt Profil* produced in autumn 2003 by Kostadinov/Thulin[24]. We invented and introduced the concepts of a **rule**, the **rule tree** and a **counter**. The idea is to give the parser a set of rules how to detect a stage indicator. This is the skeleton of a rule:

[24] See [Kostadinov & Thulin 2003] for more information about this prototype. The paper is also available on http://www.rom.lu.se/durs/usif.htm under "Publikationer".

Rule		
Search:	contains a search criterion	
	frame size:	specifies how far in the sentence the analysis engine should try to search
Action:	**match:**	next rule is r1, (increment counter c1), (annotate text with c1)
	no match:	next rule is r2, (increment counter c2), (annotate text with c2)

Each rule consists of two parts: The **search** (or conditional) and the **action** part. The search part encodes a specific criterion that should be applied to each token the parser encounters. This is what we want to look for. Applying the search part to a token in a sentence will always produce either a **match** or a **no match** depending on whether the currently inspected token matches the search part's criterion or not. The action part on the other hand specifies three things for both cases – if the search criterion matched and also if it did not match:

1. Which rule should be applied next.
2. Whether one or several special counters should be incremented at this point. If not specified, no counters will be incremented.
3. Whether the rule should try to annotate the text at this point. If not specified, this rule will not annotate the text at all.

A counter is not much more than a simple (non-negative) integer value that can be incremented during the application of the rule to indicate that the rule succeeded or failed to produce a match. Each counter can be given a "name" (an identification code) and some styling information. The name can be used to annotate an indicator group in a text with an equally named tag. The styling information is for presentational matters only; it includes coloring, special font styles and other information. Several counters can be grouped together to have the same styling information.

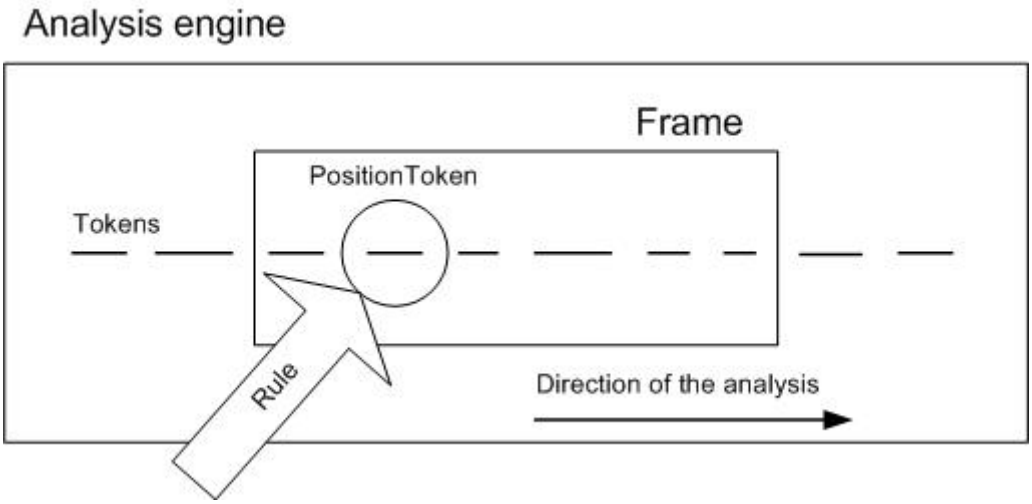
Certain restrictions are given to the a detection process: First of all, it is not very clear how far a parser should continue looking for a past participle after having detected a conjugated form of *être/avoir*. How many words may be inserted between the auxiliary verb and the past participle so that we still can be sure that we have found a *passé composé* and nothing else? If no restriction of the search range exists, even in a sentence like *Il a un magasin d'antiquités ruiné.* where it is clear from the context that this is not a *passé composé* one occurrence would be detected (*Il…a…ruiné.*) if no restrictions are given! What is needed is a restriction of the search range. To tell the rule how far to continue its search, there is the **frame size** to be

specified. A **frame** is nothing else than an excerpt a few text tokens in sequence. Alternatively, one could think of it as a window, which the analysis engine is peering through to see only a part of the text to analyze. If in the example sentence a frame size equal to 3 words would have been specified, the analysis engine would not anymore detect a *passé composé* (*Il…a…ruiné*) because it would first find the *indicatif présent* tense of avoir *a*, but then it would stop applying the rule's search criterion to look for a past participle after 3 unsuccessful trials. It would therefore never look further for past participles than the word *de* which is the third token after *a*. We will see later on that the frame size approach, although it solves some serious problems of overdetection, is not without problems neither.

A rule contains some further attributes that will be discussed below in further detail.

As can be seen from the rule skeleton above, each rule has two descendant rules, one in case of a match and another one in case of a no match. This structure naturally leads to the idea of a binary tree where every node possesses two child nodes – our rule tree. The rule tree is an association of linked rules. During the analysis process, the root rule of the tree is applied to the text again and again. When the rule's search criterion matches the current token, then the frame of the next rule is established, the next rule is loaded and the analysis engine jumps to the next token to apply the new rule. The program follows a path inside the tree from the root rule until it arrives at a terminal rule. A terminal rule is a rule with no descendant rule respectively with the root rule as its descendant. At the terminal rule either a stage indicator was found or parts of a stage indicator were found. This is the place where the counters are incremented and the annotation of the text is triggered.

Here is a picture of the procedure[25]:



Picture 3 The analysis engine at work

There is one problematic point not discussed yet. An important issue in the stage indicator table is the agreement between two words, for example between the pronoun and the verb. To solve this problem, a later rule would need knowledge about its precedent rules. Aggravatingly, the program which was in its last step looking for, let us say, a *futur simple* of the verb *faire* and now would like to know whether the found verb matches in person and number with the pronoun, cannot know how many rules might have been applied in between the discovery of the pronoun and the conjugated verb. In fact, ten or even more different rules might have been applied before the program reached the current rule. To solve this problem, the program has access to a **rule stack** (the term **rule chain** is sometimes used in *Direkt Profil* because it resembles to the taken path inside the rule tree). Every time a rule is applied, it is also pushed on the rule stack (or "added to the rule chain"). The rules in the rule stack not only store the applied rule, but furthermore a flag indicating whether they once produced a match or not, the position of the token and the token itself, if the rule's search had matched, the token's part-of-speech and inflection information and the start and end position of the frame.

It is easy now to check an agreement between the current token and any token that has been processed before by another rule. Agreement rules always have the following structure: Does there exist a token with a specified part-of-speech and inflection information which was processed successfully by an earlier rule? If there is, does it agree with the current token on the given search criteria by the currently active agreement rule?

[25] The PositionToken is basically nothing more than the currently processed token.

An example: Imagine the (wrong) sentence written by a student: **Alors, ils n'a pas faim.* Let us assume that we have the following three rules piled together: i) Search for a pronoun. ii) If found a pronoun, then search for a verb. iii) If found a verb, then check whether pronoun and verb agree in person and gender. If you find the three, then an (invented) stage indicator is detected where a verb agrees with a pronoun.

The analysis engine will first tokenize the sentence: [*Alors*][,][*ils*][*n*][‘][*a*][*pas*][*faim*][.]. When it comes to the point to apply the first rule to the sentence, it searches for a pronoun and finds the token [*ils*]. This is a 3rd person (masculine) plural pronoun, as it finds out by looking up the word *ils* in the dictionary, and it is the third token in the sentence. This information will be stored inside a rule object that is pushed on the rule stack.

Now the analysis engine starts applying the second rule on token [*n*]. In the dictionary, it finds *n* belonging to the first part of a negation *ne…pas*. This is not a verb, thus the search is continued until the token [*a*] is met. In the dictionary *a* is described as a verb in *indicatif présent* tense, 3rd person singular from the lemma *avoir*. *a* is the 6th token in the sentence. Also this rule is pushed on the rule stack.

The last rule is the agreement rule. Since it is an agreement rule, the analysis engine is not moved to the next token but stays on token [*a*]. The agreement rule accesses the rule stack and searches backwards through the rules until it finds a rule that has stored as a result of its application a token which agrees with *a* in person and number. However, no such rule can be found in the rule stack, since the token [*ils*] does only agree in person, but not in number, with the verb *a*. The conclusion is that only parts of a stage indicator were found. If she wants, the user could have introduced a counter for this special case too.

The advantage of working with rules in this way clearly lies in its generality: It is to a far extent independent of which stage indicator we would like to detect – be it a combination of an auxiliary verb in any tense and a past participle, or be it the combination of a noun followed by an adjective. In fact, the basic idea behind the rule mechanism can be reused for many different cases. Another point that makes this rule mechanism very attractive is that it is actually applicable to all European languages, and not only to French. Similar rules with the same structure could be developed for other languages too.

On the other hand, *Direkt Profil*'s rule based approach has of course limits as well. The pros and cons of such an approach will be discussed in a later chapter.

Now we know everything to put together the whole analysis engine's core algorithm. Before the analysis engine starts its work, the situation looks as follows:

- Initially, the parsing position is set to a dummy token located one place before the first real token.
- The next rule is the root rule in the rule tree.
- The rule stack is empty.
- All counters' values are set to 0.

The reader must be aware that to start searching for multi word expressions or stage indicators, the analysis engine must already have finished the tokenization of the text (step 1).

Direkt Profil's analysis algorithm:

1. Move 1 token to the right. Load the next rule in the rule tree. Push the loaded rule on the rule stack. If the loaded rule is an agreement rule go to the special procedure for agreement rules, otherwise continue with step 2.

2. Establish a frame over the tokens of the size as specified in the loaded rule. The frame starts from the point where you are right now.

3. *For every rule except agreement rules:*
   Search from the left to the right through the tokens inside the frame. If the search requires looking up in the dictionary the part-of-speech and inflection information of an inspected token, do it now. Stop when at least one of these conditions is fulfilled:
   i) A token that matches the rule's search criteria is encountered (**match**):
      Set a flag to indicate a match for this rule. Store the matching token and its position in the token array. Store the beginning position of the frame also.
   ii) The end of the frame is reached without finding a matching token (**no match**):
      Set a flag to indicate a no match for this rule. No matching token is stored. Store the beginning position of the frame and the end position of the frame.
   iii) A token is encountered which is a delimiter token (**no match**):
      Same tasks to be done as in ii).
   iv) The last of all tokens in the text is reached without a match (**no match**):
      Same tasks to be done as in ii).

4. Check the rule's action part to decide which rule shall be loaded next depending on whether the application of the rule resulted in a match or a no match. If the current rule is a terminal in the rule tree go to step 5. Otherwise go to step 1.

5. Apply the action part of every rule: Pop one rule after the other from the rule stack and apply its action part. Applying an action means incrementing statistical counters and adding annotation information to the original text. If an earlier popped rule's action has already added annotation information, delete it and put the most recently popped rule's annotation information instead. This behavior ensures that it will always be the earliest met rule during the path migration along the rule tree which will succeed in annotating the text.
   Continue until there are no more rules on the rule stack.
   Go to step 1 or stop the analysis process if you have reached the last of all tokens.

Agreement rule procedure:

Stay on the current token. Access the rule stack. Search the rule stack backwards until you find an older rule with a stored token that agrees upon the agreement criteria with the current token. If there is such a rule in the rule stack, return a match, otherwise return a no match. Go to step 4.

There are several reasons why the action part is executed not until a terminal rule is applied instead of executing it immediately after the same rule's search has succeeded or failed. Those reasons will be discussed below. At the moment it suffices to recognize that the action part can:

- increment a counter
- and annotate the text with a tag with an identification code of the stage indicator (if one is found or parts of it otherwise).

After a terminal rule in the rule tree is applied to the text, under normal circumstances the root rule will be chosen as the next rule and it will be applied again to the following token in the text. The whole algorithm continues to proceed until the last of all tokens is processed.

5.4 Unknown words

During the analysis process, it is not an uncommon scenario to meet (word) tokens in a sentence, which are unknown to the analysis engine. Usually, names of persons (*Amélie*, *Sartre*), places (*Nice*, *Europe*), brands (*Peugeot*), abbreviations (*ONU*[26]), named objects, words from languages other than French, new words (*mél*[27]) and many others are all unknown words since they all cannot be found in a regular dictionary. A straightforward definition of an unknown word is thus:

An **unknown word** is a word token that is neither a non-word nor a delimiter token and is not included in *Direkt Profil*'s dictionary.

However, according to this definition there exist two further important categories of unknown words: orthographically erroneous words and unknown words due to incompleteness of the dictionary. Especially misspelled words are expected to be frequent in beginners' stages.

Unknown words are problematic because we cannot determine their part-of-speech or inflection information. Without any relaxation of the rules' search condition constraints, such words must simply be skipped by the analysis engine. If the analysis engine encounters an unknown word, it returns a no match for the current token. All the same, in many cases a teacher would be able to guess with a high degree of certainty what the student intended to express and take the trial into account when looking for stage indicators.

In *Direkt Profil* at the moment two strategies are implemented to deal with unknown words:

- The search for words with misplaced or forgotten accents (for instance somebody has written **ecouter* instead of *écouter*),
- and for past participles only the search for words with wrong derived forms (for instance **prendu* instead of *pris* for the verb *prendre*).

The accent search and the stem search, as we call them, will be subject to further discussion below.

Another possible strategy could be to implement a special dictionary for misspelled words. Unknown words could then be checked against this dictionary and it could be tried to find the highest ranked alternative. Additionally, the semantic context of the sentence could be taken into account too. At the moment, no such plans exist for the *Direkt Profil* project.

Every attempt to guess what the author intended to say has its limitations. Simply said, the more often the program guesses and therefore overrides the user's input the smaller is the

[26] Organisation des Nations Unies
[27] *Mél* is the French version of the originally English term "email" and stands for *messagerie électronique.*

chance to still make any meaningful statement about the correctness of the result of the analysis process. To make things worse, the program can never be totally sure that the user in fact did want to express something else than what she wrote. From a user's perspective it is a nasty and annoying experience if a program overrides a per se meaningful input containing unknown words and giving wrong results because of overinterpretation. And finally, there is the danger of cascaded guessing: If an unknown word is guessed, the program still cannot be absolutely sure that for instance an agreement between the guessed word token and another token is not the product of a wrong guess. It could lead to problematic chains of several guessed unknown words. For these reasons, in *Direkt Profil* we use every sort of "guessing" only with care.

5.5 Understanding the rules and the rule tree

All rules are stored in a single file, the **rules file**, in the format of XML. A corresponding DTD is provided with *Direkt Profil* and can be found in appendix A. The rules can be manipulated independently from the program logic. They are loaded from the file at startup time by a standard XML parser (Java's SAXParser). Internally during runtime, the rule tree is stored as a Hashtable structure, guaranteeing a linear access time of the order **O(n)** for **n** accesses. A linguist or any other person thus may change the rules without doing any programming.

The tokenization rules are the only rules with a different structure from the rest. They simply specify a regular expression of what should be used as the tokenization criterion. We have already seen an example of a tokenization rule above. The following is an example of a very simple rule tree with only four rules. Some counters are included to complete the example. A counter is not a member of the rule tree structure, but they are also specified in the rule file. The counters will be discussed in the following chapter.

```
<!-- Counter #1 -->
<counter id="p01_t01_c000" name="imparfait_avoir">
    <description>Imparfait of avoir.</description>
    <format>
          <!—black on dark blue -->
          <color fg="0, 0, 0" bg="105, 175, 244" />
          <style font_style="normal" font_weight="normal"
          decoration="none" />
    </format>
</counter>


<!-- Counter #2 -->
<counter id="p01_t01_c010" name="agreement_verb_and_pron">
    <description>Agreement between verb and pronoun.</description>
    <format>
          <!-- black on light red -->
          <color fg="0, 0, 0" bg="252, 70, 66" />
          <style font_style="normal" font_weight="normal"
          decoration="none"/>
    </format>
</counter>


<!-- Counter #3 -->
<counter id="p01_t01_c020" name="pqp_avoir_with_agreement">
    <description>Plus-que-parfait of avoir with agreement between
    verb and pronoun.</description>
    <format>
          <!-- black on purple -->
          <color fg="0, 0, 0" bg="249, 85, 245"/>
          <style font_style="normal" font_weight="normal"
          decoration="none"/>
    </format>
</counter>
```

```
<!-- The root rule; Rule #1 -->
<rule id="p01_t01_r000">
    <description>Look for a nominative pronoun in the
    text.</description>
    <example>
          <ex_match>Il est gentil.</ex_match>
          <ex_nomatch>Le chat est gentil.</ex_nomatch>
</example>
    <search framesize="max">
          <inflection category="pronoun">
               <nominative value="yes"/>
          </inflection>
    </search>
    <action>
          <match nextrule="p01_t01_r010"/>
          <nomatch nextrule="p01_t01_r000"/>
    </action>
</rule>


<!-- Rule #2 -->
<rule id="p01_t01_r010">
    <description>Look for a present tense indicative conjugation of
    avoir.</description>
    <example>
          <ex_match>Elle avait une voiture.</ex_match>
          <ex_nomatch>Elle était une voiture.</ex_nomatch>
    </example>
    <search framesize="5">
          <lemma>avoir</lemma>
          <inflection category="verb">
               <tense value="imperfect"/>
          </inflection>
    </search>
    <action>
          <match nextrule="p01_t01_r020" dotagging="p01_t01_c000">
               <incrcounter value="p01_t01_c000"/>
          </match>
          <nomatch nextrule="p01_t01_r000"/>
    </action>
</rule>


<!-- Rule #3 -->
<rule id="p01_t01_r020">
    <description>Check for agreement between the verb and the
    pronoun.</description>
    <example>
          <ex_match>Il avait mangé le pomme.</ex_match>
          <ex_nomatch>Ils avait mangé le pomme.</ex_nomatch>
    </example>
    <search framesize="0">
          <agree>
               <criteria>
                    <criterion value="number"/>
                    <criterion value="person"/>
               </criteria>
```

```
                <category value="pronoun"/>
                <category value="verb"/>
          </agree>
    </search>
    <action>
          <match nextrule="p01_t01_r030" dotagging="p01_t01_c010">
                <incrcounter value="p01_t01_c010"/>
          </match>
          <nomatch nextrule="p01_t01_r000"/>
    </action>
</rule>


<!-- Rule #4 -->
<rule id="p01_t01_r030">
    <description>Look for a past participle in the
    text.</description>
    <example>
          <ex_match>Il avait mangé le pomme.</ex_match>
          <ex_nomatch>Il avait le pomme.</ex_nomatch>
    </example>
    <search framesize="3">
          <inflection category="verb">
                <tense value="past"/>
                <mode value="participle"/>
          </inflection>
    </search>
    <action>
          <match nextrule="p01_t01_r000" dotagging="p01_t01_c020">
                <incrcounter value="p01_t01_c020"/>
          </match>
          <nomatch nextrule="p01_t01_r000"/>
    </action>
</rule>
```

Picture 4 An example rule tree

It should not be too hard to understand the rules. The root rule looks for a nominative pronoun. The rule #2 looks for an *imparfait*, indicative form of the verb *avoir*. Rule #3 checks for agreement between the verb and the pronoun. Finally rule #4 tries to detect a past participle. If all the rules can successfully be applied to a text, then we have found a *plus-que-parfait* (with *avoir*) in the text! The reader should be aware that as the rules are written, no check for an agreement between the past participle and the auxiliary verb (or the pronoun) is committed. The rules would accept an input like **Il est allée à la maison*. However, a rule to check the agreement between the past participle and the auxiliary verb could be added easily.

Furthermore, this rule tree is unbalanced because for every rule in case of a no match the following rule is the root rule again. This could be different, for instance a new rule could be inserted that if no *imparfait* verb with *avoir* as a lemma is found, first the program should try

to detect an imparfait verb for the lemma *être*. Doing so, even *plus-que-parfait*s of verbs built with *être* as an auxiliary verb could be detected.
The implication is that in this example, all the rules are terminal rules for the case of a no match, and the fourth rule is a terminal rule in every case (thus it is a leave node in the tree). A terminal rule simply points to the root rule again. It lies in the responsibility of the user not to write recursive rules – with the exception of terminal rules pointing to the root rule! This means, a user must check that the rules she writes never contain circularities amongst the rules, otherwise the program's behavior is not defined and it might crash or show an unexpected behavior. Maybe in a future version of *Direkt Profil* an explicit check for circularities will be introduced. At the moment such functionality has not shown to have high priority.
What is checked though at startup time by the XML parser is whether the references from a rule to its descendant are valid rule identifiers or not. The program will tear down the startup phase if invalid references to other rules are encountered.

In rule #1 and in rule #3, special frame sizes are specified for the search. In rule #1 – the root rule – the frame size has a value set to "max". Setting the frame size of a search to the maximum value will have the effect that this rule's search will work with a frame of the same size as the whole token array. The root rule will be applied over and over again to the token array. The no match case of the first rule's search has its next rule set to the first rule itself. The resulting behavior is simply that the rule is applied to a token, and if it does not match, it skips the token and is applied to the next one. As soon as the first rule matches, all the descendant rules are applied.
Rule #3 has a frame size set to 0 (negative frame sizes are not allowed). A zeroed frame size tells the program not to move to the next token but stay on the same token where the precedent rule was applied. This makes sense if we want to apply an agreement rule, because the agreement rule should not compare the following token with an earlier one, but the same token as its precedent rule with an earlier one. If the frame size of rule #3 was set to 1 instead of 0 in the tokenized sentence [*Je*][*ne*][*suis*][*pas*][…], after the successful application of rule #2 on the token [*suis*] the agreement rule would continue with comparing [*pas*] instead of [*suis*] with the token [*Je*] and this is not what was intended. A frame size of 0 does not have the meaning of "no frame at all" but rather "stay on the token".

Every rule can be uniquely identified by the ID-attribute of the `<rule>` tag. We introduced a naming convention for rules: A rule ID contains a "process ID" reserved for future use, the ID of the tree it belongs to (this will be explained below) and an ID amongst the rules in a tree. Additionally to the search and the action part it also contains an explanatory part with a short description and some examples for a match and a no match. The description part has no influence on the program's behavior and is solely included for facilitation for the user to write and understand rules.

The full structure of a non-specific[28] rule is the following[29]:

```
rule (id)
   description
      examples
         ex_match
         ex_nomatch
   track_result
   search (framesize)
      inflection (category)
         person
         number
         gender
         tense
         mode
         nominative
      lemma
      regex
   action
      match (nextrule, dotagging)
         incrcounter
      nomatch (nextrule, dotagging)
         incrcounter
```

[28] Agreement rules, rules that look for erroneous derivation forms of past participles and rules that look for words with wrongly set accents may differ from this structure.
[29] The full DTD of the rule structure can be found in appendix A.

An important point to see is that a search can look for three different things or any combination of them:

- Regular expressions,
- Lemmas,
- Specified inflection and part-of-speech information.

Every combination of the three search criteria is possible. For example a user could at the same time want to search for something with the part-of-speech of a verb that has future tense but only for the verb *être* (where the token's lemma is set equal to *être*). Rule #2 in our example is a similar case. Searching for regular expressions is used for the detection of multi word expressions mainly.

Every inflection tag must be given an attribute with the part-of-speech of a word to look for. The possible values are to a high degree dependent on the granularity of the dictionary's entries. The part-of-speech of a word can be *noun, verb, adjective, pronoun, int_pronoun* (for interrogative pronoun)*, determiner, adverb, preposition, conjunction, numeral, interjection, abbreviation* and *residual*. *Residual* is the part-of-speech for all tokens, which do not fit in any of the other categories.

The following features might be specified for inflected words with exactly one of the corresponding values:

- Person: 1^{st}, 2^{nd}, 3^{rd},
- Number: singular or plural,
- Gender: masculine or feminine,
- Tense: future, present, imperfect or past,
- Mode: indicative, conditional, infinitive, participle, subjonctif,
- Nominative: yes or no.

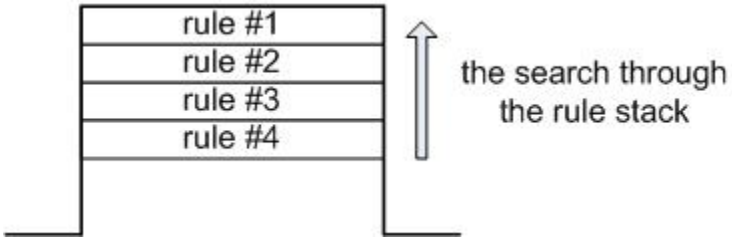
For rule writing, *Direkt Profil* only roughly follows traditional linguistic definitions of these terms. The reason is of course that a rule can only be applied to a single token at the same time. It is for example impossible to write a single rule to detect a *passé composé*. If such a composite verb form needs to be detected, then a user must write several rules. This prohibits a meaningful specification of any other tenses except those named above, since they would all be a combination of several tokens.

Mode then has a little bit the function of a pool for everything that cannot be put easily into another category.

We had to introduce the category ‘nominative’ to differentiate between pronouns. In the dictionary of ABU CNAM, there was not a precise differentiation between nominative pronouns (*je, tu, il/elle, nous, vous, ils/elles*) and other pronouns (*moi, lui, son, leur* etc.).
It can also be seen that nonsensical combinations are technically possible: Somebody could write a rule to look for a conjugated verb with future tense, 1st person singular and set at the same time the mode to infinitive. Or a person and tense could be specified for a token with the word’s part-of-speech set to determiner. *Direkt Profil* will treat every input seriously. It does not check whether a given combination makes sense or not. So it will simply endeavour to find a token that fits all the specified criteria – and find nothing since there does not exist such a word in any European language. So the result of applying a nonsensical rule will always be a no match.

A rule’s `<action>` tag always consists of a `<match>` tag for the match and a `<nomatch>` tag for the no match case. Both tags have exactly the same structure. They wear an attribute `nextrule` which is a reference to a descendant rule’s ID attribute. If the rule should be a terminal rule for either case then this reference should simply point to the root rule of the tree again. This is the only situation when a “circularity” in the tree structure must be allowed. In the example, it can be seen how the analysis engine follows the references from one rule to the next, during the application of the rules to the tokens.
If a rule wants to annotate the text, then an attribute `dotagging` can be specified. Its value must be set to the ID of a counter. The rule will then try to put an annotation around its own frame (if the frame size is zeroed, then the current word is annotated only). However, a later rule may again try to annotate the text. This case can be seen if comparing rule #2, rule #3 and rule #4 in case of a match. All these rules compete to annotate the text with their own counter name. If this happens, the following guidelines are to be respected:
First, the reader must remember that an annotation of an analyzed text always has a left and a right tag to indicate the start and the end of a stage indicator.

1. The analysis engine searches backwards in the rule stack to find a rule which wants to annotate the text with its own annotation tags. Usually this is the currently loaded rule, which is in most cases a terminal rule in the rule tree. It is the then this rule’s right to define the annotation’s



name. If in the whole rule stack no rule can be found that would like to annotate the text, simply no annotation is inserted.

2. Furthermore, it is this rule's power to set the right annotation tag in the text. If its search returned a match then the right annotation tag will be inserted just after the matching token's position. If it returned a no match then the right annotation tag will be inserted just after this rule's end of frame.
3. The left annotation tag though is inserted in the text at a position specified by the earliest applied rule in the rule stack. The left annotation tag will always be set before the earliest applied rule's beginning of the frame.

This behavior makes sense because the information about where a stage indicator group begins and ends is divided over the first and the last applied rule in the rule stack. The problem is that the first applied rule knows where to put the beginning of an annotation tag in a text but does not know where the last token of a stage indicator will be located. The last rule on the other hand does know where a stage indicator ends but has no more knowledge about where it once started. A combination is necessary.

Rule references between rules from different trees are not allowed at the current version of *Direkt Profil*. Still, in some situations, we must switch between the rule tree for multi word expressions and the rule tree for stage indicators. `<track_result>` is a special tag occurring only in rare situations. We will not discuss it in detail here. To simplify it, it serves as an entry point for the analysis engine if the engine has to reload a new rule set to apply it on the text.

5.6 Understanding the counters

In the last chapter's example, some counters were shown. Despite the many different tags inside a counter specification, technically a counter is still nothing more than a simple non-negative integer value connected to some text styling information, which can be incremented during the analysis process. From a logical perspective, a counter represents a stage indicator respectively a category in the annotation ontology and thus a linguistic phenomenon of the language acquisition process – or parts of it.

There is no limit in the number of counters a rule can increment for both a match and a no match, and of course a rule also may increment no counter at all as rule #1 in our example. At the same time, the same counter may be incremented by many different rules. For instance a counter representing *futur simple* should probably be incremented for every conjugated verb with *futur simple*, independent of whether the verb has the lemma *avoir* or *être* or any other lemma.

As can be seen in the example above, three different counters were specified:

- Counter #1: This counter is introduced to show that a verb was found with *imparfait* as a tense and *avoir* as its lemma. Thus, this counter will be incremented if one of the words *avais, avait, avions*[30]*, aviez, avaient* is found. Counter #1 is incremented by rule #2 in case of a match.
- Counter #2: This counter is introduced to indicate an agreement between the verb and the pronoun. It is incremented in case of a match by rule #3.
- Counter #3: This counter is introduced to indicate a full *plus-que-parfait* with the verb avoir. It is incremented in case of a match by rule #4.

The counters themselves do not contain any concrete meaning. They are nothing more than passive objects. The rules in combination with the annotation ontology are the responsible to give them their inherent semantic meaning. A counter can be named and incremented freely. It is the task of the rules to know which counter to increment.

As can be seen, during the detection process of a certain stage indicator often at least parts of other stage indicators are found and several counters are to be increased. To detect a *plus-que-parfait* it is a necessity to first detect an *imparfait* of either *être* or *avoir* because of course every *plus-que-parfait* in French is built either with a conjugated form of *être* or *avoir* in *imparfait* as an auxiliary.

Depending on what should be detected, several additional counters could be added, for instance a counter for situations where the auxiliary verb does not agree with the pronoun.

[30] The case of *avions* is interesting since it is ambiguous. *Avions* can indeed be the 1st person singular, imperfect tense of the verb *avoir* (English: *had*), but it might also be the plural form of the French word *avion* (English: *airplane* respectively in plural *airplanes*). Without a semantic analysis of the sentence, the case cannot be decided with complete certainty! *Direkt Profil* follows a "positive strategy with no fallback": It will try to extract the longest possible match as a stage indicator without considering descendant or later rules at the moment of decision. *Avions* would thus be treated as a verb in a first parsing attempt. The reason is simply the high complexity of a software implementation where the parser will take into account information about not yet processed tokens. The quest would be to include an in advance-parsing module in the analysis engine, which would clearly complicate the design – with questionable costs to get a presumed only slightly higher precision.

The <counter> tag's ID attribute follows the same naming conventions as a rule's ID. The <description> tag is for commenting the counter only and holds no further relevant procedural or instructional information. Since a rule can annotate a group of tokens which together build a stage indicator, the counter must be given information about how to highlight a stage indicator in the text shown to the user. Inside the <format> tag, font attributes and coloring can be specified for this purpose.

There is also a possibility to group counters together up to four cascading levels. If a user would like to have counters for each past tense (like *imparfait*, *passé composé* or *plus-que-parfait*) and furthermore a special "past tense counter" to sum them all up, she can specify the "past tense counter" as a **grouping counter**. Every counter that is attached to a grouping counter will be highlighted in the same way as its grouping counter, independent of how it specifies its own annotation style. It is in the rule author's responsibility to check that the grouping counter will also be incremented if a member of its group is incremented. Thus, grouping counters has presentational character only.

5.7 The current rule tree to detect stage indicators

It is time to turn to the currently implemented rule trees. As we have seen above, currently there are multiple rule trees applied to the text: One for the detection of multi word expressions and another one for the detection of selected stage indicators in the annotation ontology. The detection of multi word expressions will be the topic of the next chapter, now we will turn only to the stage indicator tree.

At the moment, the stage indicator tree contains around 45 different rules. As can be seen easily from the picture below (see next pages), in fact the rule structure roughly resembles a binary tree where each rule can have two descendant rules. In the picture, a rule is indicated by a small square with a unique ID number inside the rule tree. Circles are indeed not rules but indicate that a rule is a terminal node for a corresponding match/no match case. To simplify the drawing, back references from terminal nodes to the root rule have been left out. This is a directed graph. It contains no circularities! The analysis engine starts at the top and during the application of one rule after the other to the tokens it chooses a path through the tree until it reaches a terminal node. If the rule tree would contain circularities, then in theory

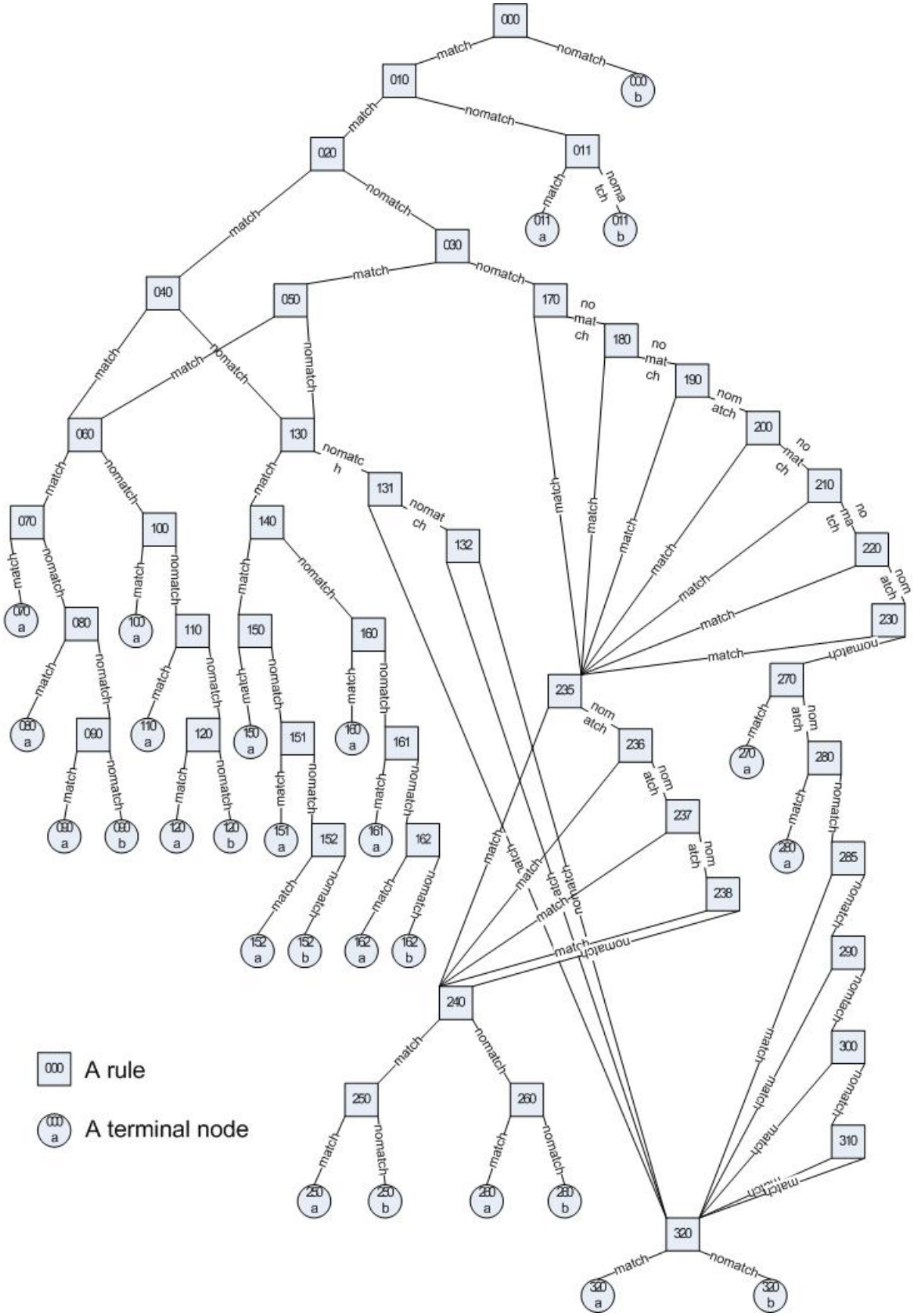
it could happen that the analysis engine gets trapped in a never ending loop applying the same bunch of rules again and again to the same token.

ID	Full rule ID	Rule's task
000	p01_t01_r000_nominative_pronoun:	Look for a nominative pronoun in the text.
000b		No pronoun has been found.
010	p01_t01_r010_verb	Look for a verb within 5 words from the pronoun.
011	p01_t01_r011_verb_with_accent:	Look for a verb within 5 words from the pronoun. Check for accents.
011a		A verb with erroneous accents was found.
011b		No verb with an erroneous accent was found.
020	p01_t01_r020_avoir:	Check if the found verb is a form of 'avoir'.
030	p01_t01_r030_etre:	Check if the found verb is a form of 'être'.
040	p01_t01_r040_pres_avoir:	Check if the form of 'avoir' is in present tense.
050	p01_t01_r050_pres_etre:	Check if the form of 'être' is in present tense.
060	p01_t01_r060_pres_accord_etre_avoir:	Check agreement between 'être/avoir' and the pronoun.
070	p01_t01_r070_pres_accord_participe:	Check if there is a past participle within 3 words from the 'être/avoir' with agreement.
070a		A *passé composé* is found where the conjugated verb agrees with the pronoun.
080	p01_t01_r080_pres_accord_participe_accent	Check if there is a past participle with an erroneous accent within 3 words from the 'être/avoir' with agreement.
080a		A guessed *passé composé* is found where the past participle had an erroneous accent and the conjugated verb agrees with the pronoun.
090	p01_t01_r090_pres_accord_participe_stem	Check if there is a past participle stem within 3 words from the 'être/avoir' with agreement.
090a		A guessed *passé composé* with a wrong derivation form of the past participle is found where the conjugated verb agrees with the pronoun.
090b		A *présent* form of *avoir/être* is found that agrees with the pronoun.
100	p01_t01_r100_pres_not_accord_participe	Check if there is a past participle within 3 words from the *être/avoir* without agreement.
100a		A guessed *passé composé* is found where the conjugated verb does not agree with the pronoun.
110	p01_t01_r110_pres_not_accord_participe_accent	Check if there is a past participle with a misplaced or forgotten accent within 3 words from the 'être/avoir' with agreement.
110a		A guessed *passé composé* with an erroneous accent on the past participle is found where the conjugated verb does not agree with the pronoun.
120	p01_t01_r120_pres_not_accord_participe_stem	Check if there is a past participle with a wrong derivation form within 3 words from the 'être/avoir' without agreement.
120a		A guessed *passé composé* with a wrong derivation form of the past participle is found where the conjugated verb does not agree with the pronoun.
120b		A *présent* form of *avoir/être* is found that does not agree with the pronoun.
130	p01_t01_r130_impf_etre_avoir	Check if the form of 'être/avoir' is in 'imparfait'.
131	p01_t01_r131_future_etre_avoir	Looking for a future tense of être/avoir.
132	p01_t01_r132_conditional_etre_avoir	Looking for a conditional1 of être/avoir.
140	p01_t01_r140_impf_accord_etre_avoir	Check agreement between 'être/avoir' and the pronoun.

150	p01_t01_r150_impf_accord_participe	Check if there is a past participle within 3 words from 'être/avoir' with agreement.
150a		A *plus-que-parfait* is found where the conjugated verb agrees with the pronoun.
151	p1_t1_r151_impf_accord_participe_accent	Check if there is a past participle with an erroneous accent within 3 words from 'être/avoir' with agreement.
151a		A guessed *plus-que-parfait* with an erroneous accent on the past participle is found where the conjugated verb agrees with the pronoun.
152	p1_t1_r151_impf_accord_participe_stem	Check if there is a past participle with a wrong derivation form within 3 words from the 'être/avoir' with agreement.
152a		A guessed *plus-que-parfait* with a wrong derivation form of the past participle is found where the conjugated verb agrees with the pronoun.
152b		An *imparfait* form of *avoir/être* is found that agrees with the pronoun.
160	p01_t01_r160_impf_not_accord_participe	Check if there is a past participle within 3 words from 'être/avoir' without agreement.
160a		A *plus-que-parfait* is found where the conjugated verb does not agree with the pronoun.
161	p1_t1_r161_impf_not_accord_participe_accent	Check if there is a past participle with an erroneous accent within 3 words from 'être/avoir' without agreement.
161a		A guessed *plus-que-parfait* with an erroneous accent on the past participle is found where the conjugated verb does not agree with the pronoun.
162	p1_t1_r162_impf_not_accord_participe_stem	Check if there is a past participle with a wrong derivation form within 3 words from the 'être/avoir' without agreement.
162a		A guessed *plus-que-parfait* with a wrong derivation form of the past participle is found where the conjugated verb does not agree with the pronoun.
162b		An *imparfait* form of *avoir/être* is found that does not agree with the pronoun.
170	p01_t01_r170_auxmod_1_vouloir	Check if the verb is an inflected form of the auxiliary verb *vouloir*.
180	p01_t01_r180_auxmod_2_pouvoir	Check if the verb is an inflected form of the auxiliary verb *pouvoir*.
190	p01_t01_r190_auxmod_3_savoir	Check if the verb is an inflected form of the auxiliary verb *savoir*.
200	p01_t01_r200_auxmod_4_devoir	Check if the verb is an inflected form of the auxiliary verb *devoir*.
210	p01_t01_r210_auxmod_5_faire	Check if the verb is an inflected form of the auxiliary verb *faire*.
220	p01_t01_r220_auxmod_6_laisser	Check if the verb is an inflected form of the auxiliary verb *laisser*.
230	p01_t01_r230_auxmod_7_falloir	Check if the verb is an inflected form of the auxiliary verb *falloir*.
235	p01_t01_r235_auxmod_lex_present	Check if the auxiliary modal verb has *présent* tense.
236	p01_t01_r236_auxmod_lex_future	Check if the verb has *futur simple* tense.
237	p01_t01_r237_auxmod_lex_imperfect	Check if the verb has *imparfait* tense.
238	p01_t01_r238_auxmod_lex_conditional	Check if the verb is in *conditionnel* mode.
240	p01_t01_r240_accord_auxmod	Check if there is an agreement between the auxiliary verb and the pronoun.
250	p01_t01_r250_accord_infinitif	Check if there is a verb in infinitive within 3 words from the auxiliary verb with agreement.

250a		A *verbe auxiliaire + infinitif* structure has been found where the conjugated verb agrees with the pronoun.
250b		A conjugated verb (*vouloir, pouvoir, savoir, devoir, faire, laisser, falloir*) has been found that agrees with the pronoun.
260	p01_t01_r260_not_accord_infinitif	Check if there is a verb in infinitive within 3 words from the auxiliary verb without agreement.
260a		A *verbe auxiliaire + infinitif* structure has been found where the conjugated verb does not agree with the pronoun.
260b		A conjugated verb (*vouloir, pouvoir, savoir, devoir, faire, laisser, falloir*) has been found that does not agree with the pronoun.
270	p01_t01_r270_verb_lex_1	Check if the verb is an infinitive.
270a		A sentence with an *infinitif* but no other conjugated verb has been found.
280	p01_t01_r280_verb_lex_2	Check if the verb is a participle.
		A sentence with a *participle* but no other conjugated verb has been found.
285	p01_t01_r285_verb_lex_present	Check if the verb has *présent* tense.
290	p01_t01_r290_verb_lex_future	Check if the verb has *futur simple* tense.
300	p01_t01_r300_verb_lex_imperfect	Check if the verb has *imparfait* tense.
310	p01_t01_r310_verb_lex_conditional	Check if the verb is in *conditionnel* mode.
320	p01_t01_r320_accord_lex	Check if there is agreement between the verb and the pronoun.
320a		A common conjugated verb has been found which agrees with the pronoun.
320b		A common conjugated verb has been found which does not agree with the pronoun.

Table 4 The stage indicator rules



Picture 5 The stage indicator rule tree

What can hardly be seen from the drawing alone are the different parts of the tree belonging together. The first two rules are the most important ones. The root rule (the rule with the ID `p1_t1_r000`) is looking for a nominative pronoun (*je, tu, il/elle, nous, vous, ils/elles*) token over and over again. As soon as it finds one, the rule looks in a frame of (currently set to) 5 further tokens for any kind of verb (rule ID 010). At this point in time, no difference is made whether the verb agrees with the pronoun, whether it is finite or infinite etc. If no verb is found, rule ID 011 looks in the same frame for unknown words and if it finds one tries to guess whether maybe a verb could be built from the unknown word if a set accent (*accent aigu, accent grave, accent circonflexe, tréma*) is taken away, moved to another character position or added to the unknown word. If still no verb can be produced from the unknown word, then the program assumes to deal with a "sentence without a verb" (counter `p1_t1_c0000` in the annotation ontology) and increments the corresponding counter in terminal node ID 011b.

However, if a verb is found in the step before, a bunch of different checks is applied to the verb.

One part of the rules tries to identify the auxiliary verbs *avoir* or *être* (rule ID 020, 030) in *présent* tense (rules ID 040, 050) or *imparfait* tense (rule ID 130) and in combination with a *participe passé* (rules ID 070, 080, 090, 100, 110, 120, 150, 151, 152, 160, 161, 162) in a frame of 3 tokens after the verb to detect either a *passé composé* or a *plus-que-parfait*. Rules ID 080, 110, 151 and 161 explicitly check again for missing or erroneous accents on a *participe passé*, since it is a common mistake for French learners to forget to put an accent on the past participle's ending. Rules ID 090, 120, 152 and 162on the other hand try to find past participles with wrong derived forms (**il a prendu*). Sometimes, a language learner does not know the exception of how to derive a past participle correctly and thus she may end up with **prendu*, because she thinks that the past participle from *prendre* (which would be *pris*) must be derived like the past participle from *rendre* (which is *rendu*). In such situations, rules 090, 120, 152 and 162 are especially useful. Rules ID 060 and 140 additionally check whether the conjugated auxiliary verb agrees in person and number with the pronoun found in rule ID 000 or not.

If a finite form of *avoir/être* is found, but it is not a *présent* or *imparfait* but for instance a *futur simple* (rule ID 131) or a *conditionnel* (rule ID 132), then they should not be treated as belonging to a composite verb form but as standalones.

If the found verb is not any form of *avoir/être* it still can belong to a composite structure. Rules ID 170 (*vouloir*), 180 (*pouvoir*), 190 (*savoir*), 200 (*devoir*), 210 (*faire*), 220 (*laisser*) and 230 (*falloir*) are combined with rules ID 250 and 260. The last two rules are looking for an infinitive in a frame of 3 words after one of the former conjugated verbs has been detected. The idea is to find combinations of an auxiliary + infinitive like *vouloir faire*, *pouvoir faire*, *savoir faire* etc. which would be counted in the terminal nodes ID 250a/b and 260a/b. In between there are rules for checking for different tenses and modes (rule ID 235 for *présent*, 236 for *futur simple*, 237 for *imparfait*, 238 for *conditionnel*). Rule ID 240 checks the agreement between the conjugated verb and the pronoun.

If the verb is not one of those listed in the last paragraph's beginning, it is assumed to be a (common) lexical verb. In this case, the program would like to denote the verb's inflection. Since the verb could also be an infinite form, as for example in **Je manger ça.*, rule ID 270 looks for an infinitive whereas rule ID 280 looks for standalone past participles (**Je mangé ça.*). Rule ID 285, 290, 300, 310 look for (*indicatif*) *présent*, *futur simple*, *imparfait* and *conditionnel*. Finally, rule ID 320 checks for an agreement between the conjugated verb and the pronoun.

Comparing to *Direkt Profil* version 1.4, in *Direkt Profil* v1.5.1 we introduced a couple of new rules. The biggest changes are the explicit possibility to write rules that can check for missing or erroneous accents in a word, a rule checking for wrong derivation forms for past participles, and a couple of new rules for a more detailed differentiation between the conjugated verb's tenses and its mode.
Our experience has shown that an especially weak point of the v1.4's rules was the terminal node 3b: The part-of-speech of the `p1_t1_c0000` (sentence without a verb). In many cases, the verb was not detected due to misspellings and not because the sentence really did not contain a verb. Rule ID 011 easens the burden to a certain extent. If a language learner simply has forgotten to set an expected accent on the verb, then the v1.5.1's analysis engine still can guess that the author probably wanted to put a verb. It is questionable whether in the terminal node 3a the analysis should be continued. Node 3a means means that an unknown word was found in the sentence and trying to set accents on different positions resulted in a well known verb. So, the program guessed that probably a misspelled verb was found. To continue here would mean to rely with the further analysis on a guess. This strategy might turn out to be dangerous with a tendency to overdetection. For this reason at the moment no descendant rule

is specified to terminal node 3a. We are however thinking of changing this “dead end category”.

Comparing the rule tree to the stage indicator table from chapter 2.2, the following stage indicators are actually implemented:

- **Nominal Utterance Structure:** The program detects *phrases sans verbes* (sentences containing no verb) in terminal node ID 011b.
- **Negation:** No functionality implemented to explicitly inspect the negation.
- **Finite and infinite verb forms:** The program can differentiate between sentences where the verb is infinite as in terminal nodes ID 270a (*infinitif*) and 280a (*participe*) and where it is finite.
- **Agreement between sentence’s subject and verb:** Checks for agreement between the conjugated verb and the leading pronoun according person (1^{st}, 2^{nd}, 3^{rd}) and number (singular or plural) are possible. They are used in several places as for example in rules ID 060, 140 or 320.
- **Tense, mode, aspect (TMA):** Extensive checks for different tenses (*présent, passé composé, plus-que-parfait, imparfait, futur simple*) and for mode (*conditionnel*) are being performed. No checks for *subjonctif* or *passé simple* are foreseen now though the functionality exists.

  Comparing different aspects in a single sentence is not possible. However, comparing the number of occurrences between (for instance) verbs in different tenses could maybe be used too. For instance, one could compute the relation of the number of verbs in *imparfait* divided by the total number of past tense verbs.
- **Elision/Cliticisation:** No functionality implemented to inspect elision/cliticisation.
- **Gender and agreement:** Since with the current version of *Direkt Profil* only verb groups are being processed, no agreement check can be performed between nouns, adjectives and participles. Such functionality will be implemented in a future version.
- **Incorporation of article and preposition:** No functionality implemented.
- **Subordination:** No functionality implemented.

The focus of the rule tree was laid clearly on the analysis of verb groups’ stage indicators.

One point must be mentioned. As can be deduced from the rule tree, the tree does not only specify what rules should be applied, but also in which **order** the rules are applied to the text. The rule tree naturally defines a precedence order of the rules. Sometimes, the order is given

logically. For example is it necessary to first detect that a certain token is a verb, and only then can be checked whether the verb is (e.g.) a form of *avoir* or *être*. In other cases, the responsibility is laid fully in the hands of the person writing the rules. It is not always clear, which order is the optimal, and it might turn out that different precedence orders have different advantages and disadvantages. The order has a strong influence on what will be detected in the analysis! Sometimes, the precedence of one rule over the other leads to wrong results. When working with *Direkt Profil*, we often encountered situations like this: In the sentence **Il reussit à organiser son voyage.* by mistake an accent was not set in the main verb *réussit*. In the rule tree, the rule 010 ("Look for a verb.") has precedence over rule 011 ("Look for a verb with accent error.") Since the frame size of rule 010 is set to 5 tokens after a pronoun was found, rule 010 will search for a verb in the frame [*reussit*] [*à*] [*organiser*] [*son*] [*voyage*]. In the first attempt to find a regular verb, *organiser* (as an infinite verb form) is found but not the misspelled word *reussit*. The program then continues with the application of rule 020 ("Check if the found verb has *avoir* as its lemma.") and the rule 011, looking for misspelled verbs, is never applied to the text. The precedence order prevented a correct analysis in this case.

As can be seen from the rules above, the current version of *Direkt Profil* only processes verb phrases but no noun, adjective or other phrases! The current version of *Direkt Profil* is thus not yet able to detect or process stage indicators concerning the development of phenomena connected to noun or adjective groups. Our project group aims at finishing a running version of the program with the implementation of noun and adjective group stage indicators in late summer 2005. Connected to this task is also the problem that the current version of *Direkt Profil* does not yet handle sentences, which do not contain a pronoun! In a sentence like *L'homme est allé à la maison.* – although it contains a *passé composé* as a stage indicator – **no** stage indicator will be detected by the current implementation of the analysis engine because the sentence does not contain any (nominative) pronoun. At the moment, *Direkt Profil* is still a prototype and under construction. This serious "gap" of functionality should also be filled with the future version. The reasons why no implementation is finished with the hereby described approach are discussed below.

5.8 Processing multi word expressions (MWE)

Multi word expressions must be excluded with care from the stage indicator analysis. Linguistic research has shown that such expressions are learnt in the language acquisition process as a whole and by heart, which means that they should not be counted as a regular stage indicator even if a multi word expression would contain such an indicator. For this purpose, the program actually does not only store a single rule tree for the stage indicators but also a second rule tree which can be processed independently from the former. In fact, *Direkt Profil* applies two rule trees to the text one after the other, we sometimes refer to this behavior as the pre-process for detecting multi word expressions and the main process for detecting the stage indicators. First, the rule tree for multi word expressions is applied and the results are collected. At this step information about how to annotate multi word expressions in the text is stored in a special data structure, which we called the result chain (presented in chapter 5.2). Only then the second rule tree for stage indicators is processed and results are put into the result chain too.

The same generalized algorithm as introduced in the annotation ontology is also used to detect multi word expressions. Indeed, detecting a multi word expression can be seen as the detection of a special stage indicator. The following is a list of multi word expressions, the program wants to detect: *Je m'appelle, je voudrais, je ne sais pas, s'il vous plaît, c'est, qu'est-ce que c'est, j'ai mal, il y a, il faut, il n'y a pas*.

Multi word expressions are detected mostly by searching for a chain of tokens that match a chain of regular expressions. The multi word expression *c'est*, which is tokenized as [*c*][‘][*est*], can be detected by looking for the regular expressions

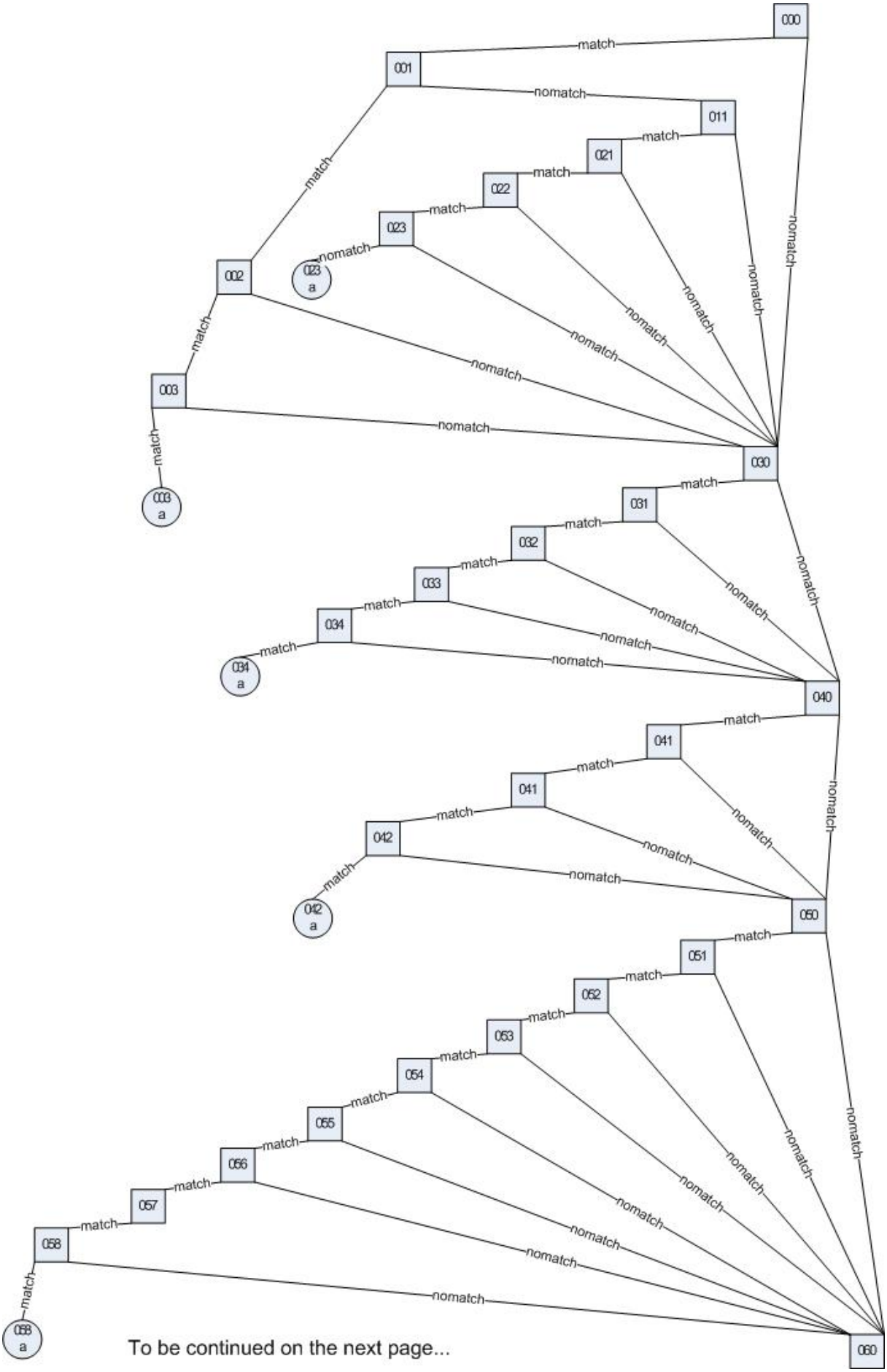
1. `(C|c)` or `[Cc]`
2. `(‘|´)` or `[‘´]`
3. `est`

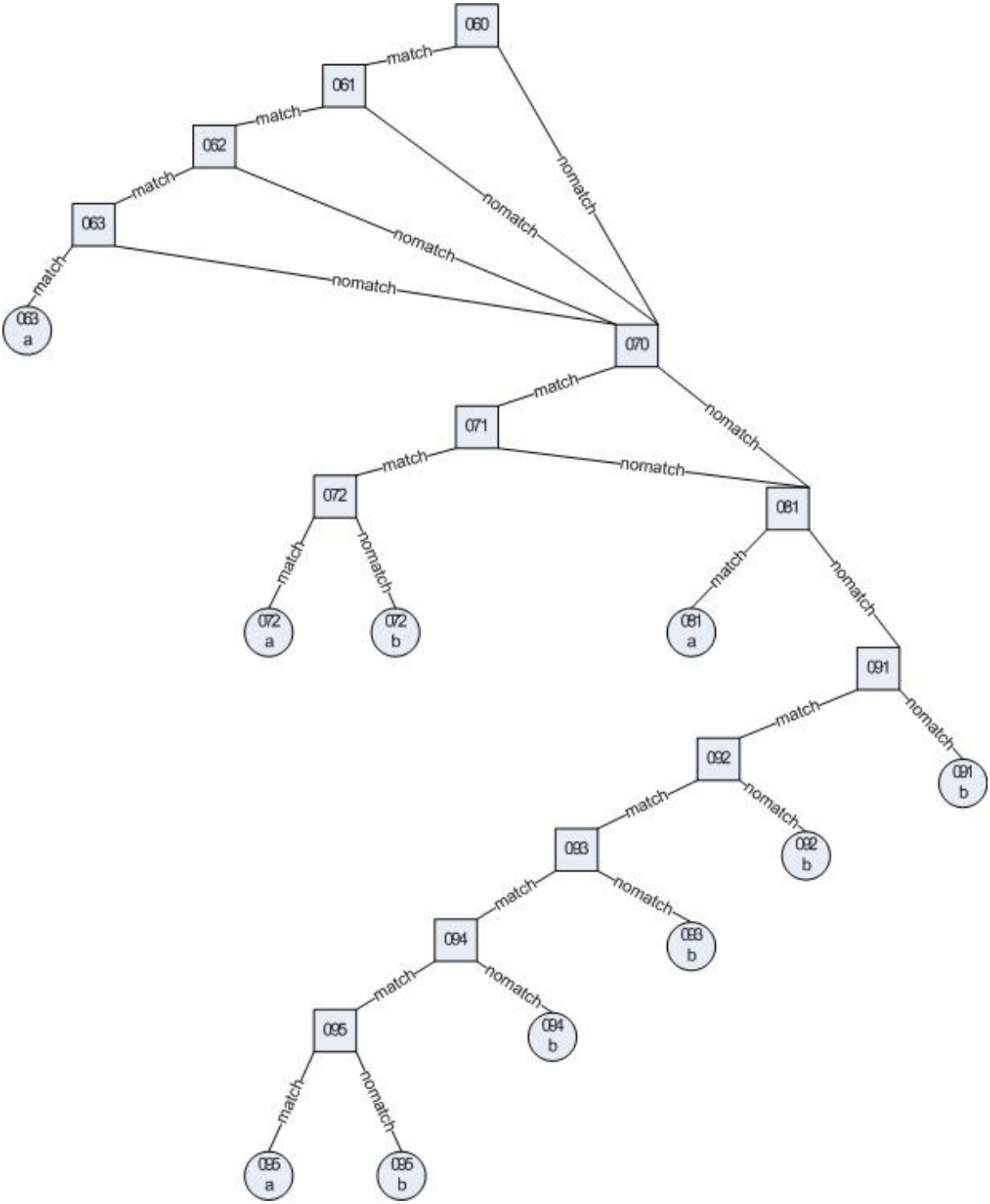
in the given order. The idea of applying a specialized multi word expression rule tree to the text remains the same as with the stage indicator tree: On the first token, try whether it matches the regular expression `[Jj]e`. If it matches, then move to the next token and try to match it with the regular expression `m`. If the token does not match, stay on the token and try to match the regular expression `voudrais`. If it still does not match, then again try to match the regular expression `ne`, etc. The following multi word expression rule tree is implemented in the current version of *Direkt Profil*:

ID	Full rule ID	Rule's regular expression to look for
000	p00_t00_r000_je	[Jj]e
001	p00_t00_r001_m	[Mm]
002	p00_t00_r002_apostrophe	['´]
003	p00_t00_r003_appelle	appelle
003a		*je m'appelle*
011	p00_t00_r011_voudrais	voudrais
012	p00_t00_r021_ne	ne
022	p00_t00_r022_sais	sais
023	p00_t00_r023_pas	pas
023a		*je ne sais pas*
030	p00_t00_r030_s	[Ss]
031	p00_t00_r031_apostrophe	['´]
032	p00_t00_r032_il	il
033	p00_t00_r033_vous	vous
034	p00_t00_r034_plait	plaît
024a		*s'il vous plaît*
040	p00_t00_r040_ce	[Cc]e
041	p00_t00_r041_apostrophe	['´]
042	p00_t00_r042_est	est
042a		*c'est*
050	p00_t00_r050_qu	[Qq]u
051	p00_t00_r051_apostrophe	['´]
052	p00_t00_r052_est	est
053	p00_t00_r053_hyphen	-
054	p00_t00_r054_ce	ce
055	p00_t00_r055_que	que
056	p00_t00_r056_c	c
057	p00_t00_r057_apostrophe	['´]
058	p00_t00_r058_est	est
058a		*qu'est-ce que c'est*
060	p00_t00_r060_j	[Jj]
061	p00_t00_r061_apostrophe	['´]
062	p00_t00_r062_ai	ai
063	p00_t00_r063_mal	mal
063a		*j'ai mal*
070	p00_t00_r070_il	[Ii]l
071	p00_t00_r071_y	y
072	p00_t00_r072_a	a
072a		*il y a*

072b		Not a multi word expression!
081	p00_t00_r081_faut	`faut`
081a		*il faut*
091	p00_t00_r091_n	`n`
091b		Not a multi word expression!
092	p00_t00_r092_apostrophe	`['´]`
092b		Not a multi word expression!
093	p00_t00_r093_y	`y`
093b		Not a multi word expression!
094	p00_t00_r094_a	`a`
094b		Not a multi word expression!
095	p00_t00_r095_pas	`pas`
095a		*il n'y a pas*
095b		Not a multi word expression!

Table 5 The multi word expression rules


000
001
011
021
022
023
002
023 a
003
003 a
030
031
032
033
034
034 a
040
041
041
042
042 a
050
051
052
053
054
055
056
057
058
058 a
060
match
nomatch
To be continued on the next page...



Picture 6 The multi word expression rule tree

Cases where only fragments of a multi word expression are found, like in *il y est* or *qu'est-ce qu'il a dit*, are not treated specially.

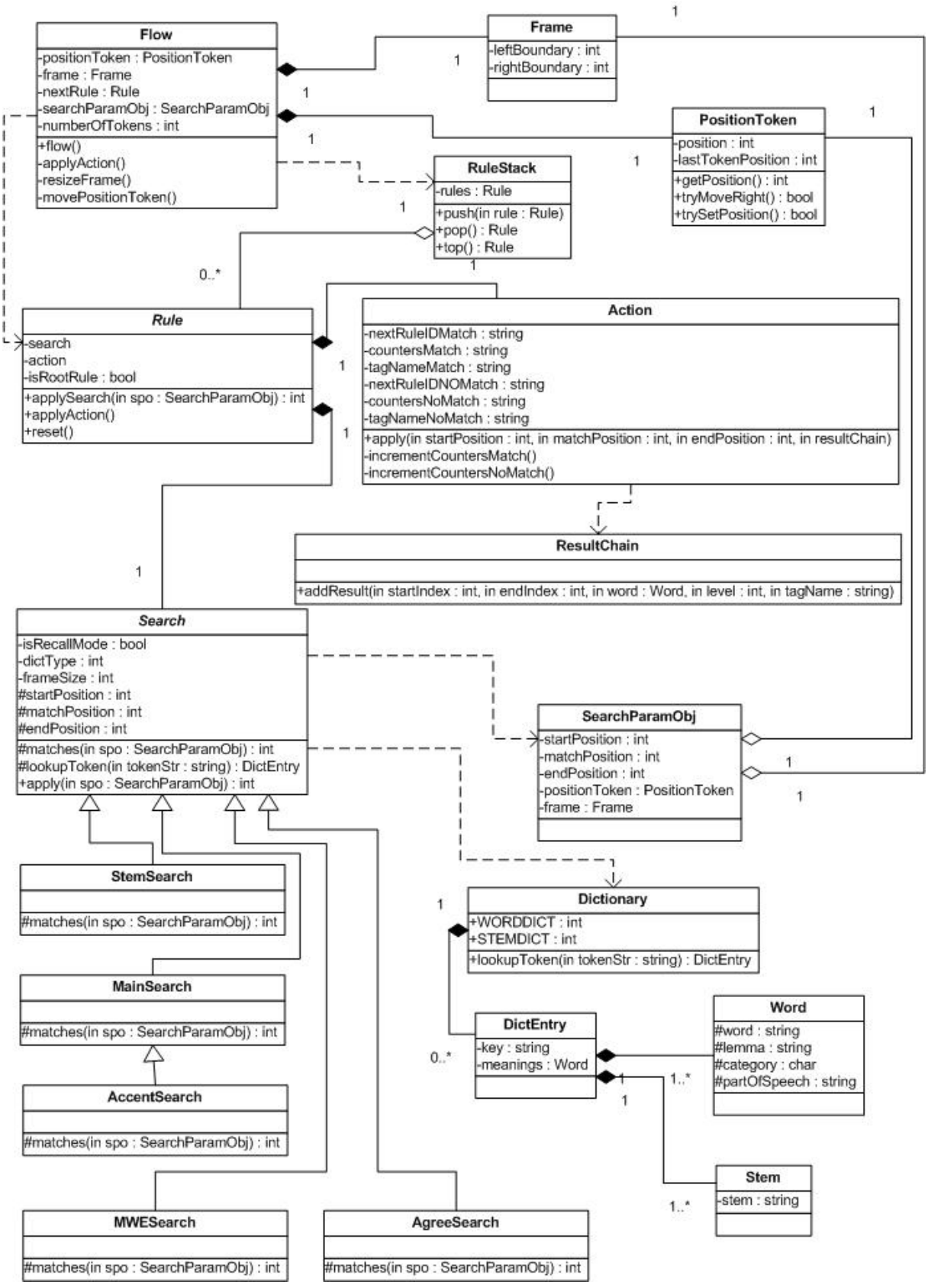
6 Architecture of Direkt Profil

In this chapter we will take a deeper look into the software architecture of *Direkt Profil*. The chapter is rather technical. It can be skipped by the reader if the implementation details are of no interest. However, for a profound understanding of the program and for writing new rules that are then added to the rule tree, the following explanations should be read.

6.1 The architecture

Direkt Profil is implemented as a client/server solution. Multiple clients can access the analysis engine's services at the same time. Through a traditional internet browser a text can be submitted (using the HTTP-protocol). On the server side, an Apache Tomcat server will accept ingoing connections. For every new user, a new analysis engine is instantiated. All analysis engines use the same single instance of the dictionary. The user's analysis engine processes the text by tokenizing, analyzing, annotating and putting it together again. A dynamically generated website is created that is finally sent back to the client.

Direkt Profil's whole program architecture shall not be discussed here. We used standard technologies and design patterns for well known software architecture problems as far as possible. Though, the implementation details of the analysis engine itself might be a matter of interest. The following drawing is a class diagram of those classes which together provide the *Direkt Profil*'s core functionality.



Picture 7 The analysis engine's design

Of course this drawing leaves out many technical details and connections between classes. This is especially true for the result collection process which is not too interesting from a language processing point of view.

The classes are:

- **Flow:** Flow is the most important class in the analysis engine. It incorporates *Direkt Profil*'s core analysis algorithm.
- **Frame:** Frame is a passive object keeping track of the currently loaded Search object's frame size. It restricts the search on the left and right side.
- **PositionToken:** PositionToken is a passive object that is being moved over the array of tokens. Its only task is to keep track of the current analysis position.
- **SearchParamObj:** SearchParamObj has a rather technical meaning. It is a passive container for the many different parameters used by a Search object when applied to a token.
- **Rule:** For every rule in the rule tree a corresponding Rule object exists. A Rule object mainly contains a Search and an Action. It delegates calls to apply its Search's matching procedure or execute its action to the corresponding objects.
- **RuleStack:** Every rule Flow applies to a token is pushed on the rule stack. The rule stack is emptied as soon as all rules' actions are triggered.
- **Search:** The Search class is an abstract class. A Search object's criterion can be matched against the current token by applying its `match()` method. Several different subclasses are implemented. For the different search criteria a rule can have, several subclasses of Search exist. Every subclass is specialized on a different search criterion.
  - **MainSearch:** The MainSearch can be given a regular expression, a lemma or a certain word part-of-speech plus inflection information (or any combination of these) as a search criterion.
  - **AccentSearch:** The AccentSearch can be given the same search criteria as the MainSearch but it behaves differently. Additionally to MainSearch's functionality, it can inspect misspelled tokens that are subject to missing or misplaced accents.
  - **StemSearch:** The StemSearch can look for wrongly derived past participles.
  - **MWESearch:** The MWESearch looks for multi word expressions. Its search criterion is a regular expression.

    - **AgreeSearch:** The AgreeSearch can check for agreement between given criteria of two given words, for instance between the person and number of a verb and a pronoun.
- **Action:** The Action object incorporates the action being done by a rule after this rule's search is applied. It covers two cases: What should be performed when a match was produced and when a no match is produced. An Action object has two main tasks: incrementing counters and annotating a text. It adds a result object to the result chain.
- **ResultChain:** The ResultChain is a collection of result objects. The analysis of a token generates information which can be stored in such a result object.
- **Dictionary:** The dictionary is a passive object. There are two different instances of it in *Direkt Profil*: the "word dictionary" and the "stem dictionary"[31]. It can be used to look up the part-of-speech and inflection information of the current token object.
- **DictEntry:** The dictionary contains entries as DictEntry objects. A DictEntry object always consists of a (search) key and a list of Word entries that have the same key in common. For instance the word *aimée* can be an adjective, a noun or a past participle. The key entry *aimée* will then hold entries for every different meaning.
- **Word:** Contains the part-of-speech, the inflection information and its lemma for a token that is a word. There are subclasses for every word part-of-speech: for verbs, nouns, adjectives, adverbs, determiners etc.
- **Stem:** Contains the stem of a word. In *Direkt Profil* v1.5, there are stem entries available only for verbs.

The most important class is called **Flow** (in resemblance to a work flow). Flow is the class incorporating the core algorithm as introduced in chapter 5.3. After instantiation, the analysis process can be started by calling the public method `flow`. Flow has access to many things: the array of tokens (not included in the diagram), all the rules in the rule tree (not included in the diagram) and the current frame (an instance of class **Frame**) are the most important ones. It has also access to two special objects unique to an analysis engine called the **PositionToken** and the **SearchParamObj** ("search parameter object"). The PositionToken is a passive object that is being moved over the array of tokens by the Flow. Its task is only to keep track of the current position. The SearchParamObj is also a passive object which has to be a container for the many different parameters that the Flow must give to a rule to execute

[31] We use the term „word dictionary" to more clearly differentiate between the non-stem dictionary (= word dictionary) and the stem dictionary. However, mostly the project team simply refer to the word dictionary as „the dictionary" since it is used by far more often during the analysis process than the stem dictionary.

its search's matching procedure. (The composition or dependency connection between Flow and SearchParamObj is left out in the drawing.)

The Rule object consists of two further important objects: The Search object and the Action object. The class Search has many subclasses, one for each different search that should be performable.
A rule object is a black box to the Flow. Flow does not have to deal with anything that can be kept hidden inside Rule, Search and Action. It is not interested in the details of the matching procedure or how exactly the action must be performed. There are two interface methods inside the class Rule that it can call to trigger the search's matching procedure – the `applySearch` method – and taking the action respectively – the `applyAction` method.
If it wants to match a rule's search against the current token, it simply calls the rule object's `applySearch` method, giving it a bunch of parameters, all stored together in the SearchParamObj. This method then delegates the matching procedure task to the search's `match` method. The same counts for the `applyAction` method, it also delegates every call further to the actions `apply` method.
Indeed, every subclass of Search has its own matching procedure and thus must implement its own version of the search's `match` method. The search's `match` method will return an integer value, indicating whether the token successfully or not matched against the search's criteria. The value is then returned to Flow again from the method `applySearch`.
During the matching procedure, all necessary information for later steps is stored inside the Search object. When after the matching procedure the rule's `applyAction()` method is called by Flow, the Action object will gather itself the necessary information to decide what to do exactly. Flow does not have to decide for the Action object how to behave.

The basic idea is that Flow loops over all the tokens in the token array.

1. It loads the next rule,
2. adds the newly loaded rule to the rule stack,
3. calls the rule's `applySearch` method by giving it the necessary parameters bundled in the SearchParamObj,
4. memorizes whether the `applySearch` returned a match or a no match and according to the result tries to find out which rule will be the next to apply.
5. If the next rule is the root rule again (= the current rule is a terminal node for the matching procedure's result), the rule's `applyAction` method is called now. In this

case, the last rule is popped from the rule stack and its Action's `apply` method is called. Then the next rule is popped from the rule stack and this Action's `apply` method is called also. This pattern is repeated as long as there are rules on the rule stacks.

6. As a next step, Flow must decide where to move the position token and how to set the frame boundaries. The next rule has to take into account two parameters: The frame size (possibly set to 0) and furthermore it can have the parameter `recall mode` set to `true`. The exact meaning of these parameters will be explained in the next chapter. Flow now moves the PositionToken to the correct position – mostly just to move one token to the right in the token array – and sets the frame's left and right boundaries

This algorithm reflects the analysis engine's behavior as encountered it already in chapter 5.3.

When Flow starts to work, many tasks have already been completed before. The text has been tokenized already with the usage of the tokenization rules, and an array of Token objects (not included in the drawing) has been produced. As we have seen earlier, the same analysis engine but different rule trees can be used for the detection of multi word expressions and stage indicators. This means, that when Flow is executed for the first time for the detection of multi word expressions, no result exists at this moment and the result chain is still empty. If Flow however is executed the second time to detect stage indicators, the result chain already contains result objects.

When Flow has finished its work, the result chain is processed to reproduce the annotated text. The responsible classes are left out in the diagram.

6.2 The Search

With the diversity of the search criteria that *Direkt Profil* should be able to deal with when analyzing a text, there are multiple subclasses of the abstract class Search. Every subclass is specialized on searching on different criteria.

MainSearch is the most commonly used by a rule. It can try to match a token against regular expressions ("Does the current token match the regular expression `[Aa]utomobile`?"), it can match against lemmas ("Does the current token has the specified lemma *avoir*?") and it

can match inflection information ("Is the current token a verb with the tense of *futur simple* and 3rd person singular?"). Combinations of the search criteria are also possible.

The **AccentSearch** can be looked at as some kind of constraint relaxation technique. It is a common mistake for language learners to forget or misplace accents in French. It can match tokens against all the search criteria as the MainSearch. It will behave exactly the same way as the MainSearch – except when it comes to a token which is an unknown word. If this is the case, the AccentSearch will produce a list of permutations of the given input string by setting every thinkable possibility of accent combinations (*accent grave, accent aigu, accent circonflexe, tréma*) on the vowels in the word. For instance, the misspelled word **hotel* will result in a list of combinations (*hotel,) hotél, hotèl, hotêl, hotël, hôtel, hôtél, hôtèl, hôtêl* and *hôtël*. Of course most of these combinations do not exist in French. After the production of the list of permuted words, the AccentSearch tries to find at least one of these words in the dictionary, in our example *hôtel*. If such a word is found, then again, like in MainSearch, the matching criteria are applied to it. Only if it now matches, the AccentSearch as a whole will return a match success.
A short performance test of the accent search module showed that the time consumption to produce such a list permutations is negligibly small: Only a few dozens of milliseconds are needed to produce a complete list of different spellings for a word like *automobiliste* with 7 different vowels. In French, only a few words can be expected to have so many different vowels.
In the beginning, I (as the author of the accent search module) was unsure whether ambiguities in the produced list could occur, leading to wrong analysis results. This fear turned out to be unnecessary. Until now we never met a situation, where the list contained two suitable alternatives and the wrong one was chosen. Usually, out of a long list of different spellings, only a single alternative is a meaningful word in French.

The **StemSearch** is also some kind of constraint relaxation technique. Its goal is to detect wrong derived forms of past participles like **prendu* (instead of *pris*). With the stem search, *passé composé*s and *plus-que-parfait*s can be detected as a stage indicator where the author of a text seemingly wanted to express such a tense. It works by cutting of the suffix, leaving the stem and consults the stem dictionary to find out whether it is contained in the dictionary. If such a stem is contained and since there are only stems for verbs in the stem dictionary, the program can guess that the writer produced a derivation mistake.

An example: In the nonexistent word **prendu*, what is the product of a wrong derivation, the suffix *–u* is first cut off from the word, what leaves the stem *prend*. The dictionary is consulted whether an entry *prend* exists. And really, there is such an entry (for instance as the stem *prend* of the existing word *prendrait* as for others) and the entry tells that the stem *prend* is derived from a verb, and not from an adjective, noun, adverb etc. Because **prendu* has the stem *prend* of a verb and the suffix *–u* of a past participle, it is highly probable that the writer in fact tried to build a past participle of the verb *prendre* (which, as we can assume, she thought must be derived like the past participle of *rendre* – namely *rendu*).
The problem of ambiguities when taking this approach turned out to be bigger than in the accent search case. Many words seem to have a regular *participe passé* ending (*-é, -ée, -és,* etc.) in combination with a stem from a regular verb. For instance the preposition + article *du* (*de + le*) already fulfils the criteria for a stem search, since it has a regular participe passé ending *–u* and a regular verb stem *d-*[32]. In the following case, the stem search guesses to have met a *passé composé*, where the *participe passé* was derived wrongly (counter `p1_t1_c5240` in the annotation ontology): *Il a un vin du Valais.* where *Il a…du*. is thought to be a corresponding case.

The **MWESearch** is like a slim version of MainSearch. The only criteria it can look for are regular expressions. Otherwise it behaves like MainSearch.

The **AgreeSearch** tries to match two different tokens against each other according to given matching criteria. The “left token” is the token encountered earlier in a text when reading from the left to the right, whereas the “right token” is the token at the current position of the analysis engine. It then compares the two tokens according to the given criteria. Mostly this is used for situations where the program should find out if a pronoun and a conjugated verb agree in person and number. As seen in the stage indicator table in chapter 2.2, there are several stage indicators relying on the functionality to investigate the agreement between the pronoun and the conjugated verb. In chapter 5.3 the algorithm of the agreement search procedure was described already.
An agreement search must by default have a frame size set to 0. This is because at least one token – the right token – must be fixed. Otherwise it would not be clear which two words to compare exactly.

[32] *d-* can for example be the stem of the verb form *dirent* – the 3rd person plural, *indicatif passé simple* of the verb *dire*.

Every subclass inherits the abstract Search's `match` method which it must overwrite. When the Rule class' `applySearch` method is called, it calls the search object's local `match` method and giving it the search parameter object. The search inspects all the tokens in the current frame beginning from the first token and continuing until it finds at least one token that matches its criteria.

The search always operates in a clearly defined frame of tokens. It loops through the frame of tokens trying to find a matching token. If it finds a matching token or if the end of the frame is reached without success, the search stops. What has not been said so far is that the analysis engine must be provided with precise instructions of how to set the frame's left and right boundaries and on which token exactly to continue with the analysis for the next rule. Two options influence the decision.

i. *"Halt rule":* If the next rule's frame size is set to 0, the analysis engine should be applied on the current token again. This is called a "halt rule". If it is set to any positive value greater than 0, the next rule must be applied on a frame of tokens.
ii. *"Recall mode"* – this option is only important if the next rule is not a "halt rule" as specified in ii.: Should the next rule be applied again in the same frame as the just now applied rule or in a subsequent frame.[33]

Let us make an example to understand things better. We take [*1*][*2*][*3*][*4*][*5*][*6*][*7*][*8*][*9*] as a hypothetical example of a tokenized sentence where a bracketed number would indicate a tokenized word. We assume that we just arrived from applying rule *R* on token [5]. Rule *R* had started on token [*2*] with a frame size of 4 tokens. Rule *R* has two descendant rules. In case of a match, rule S_{match} should be chosen, in case of a no match rule $S_{no\ match}$ instead. Let us now assume that *R* has just produced a match on token [*5*]. S_{match} is thus loaded by Flow. If S_{match} has a frame size set to 0, then it will be applied on token [*5*] again, since a frame size of 0 means: "Do not move the position token at all." If S_{match} however has a frame size set to – let us assume – 3, the analysis engine additionally must respect the "recall mode" option's value. A "recall mode" set to *true* would result applying S_{match} to the tokens [*2*][*3*][*4*] since it starts again where *R*'s frame already had started. For S_{match}'s frame size is set to 3 (and not to 4 as in rule *R*) the frame will be set to [*2*][*3*][*4*] only instead of [*2*][*3*][*4*][*5*] as in *R*.

If S_{match} does have set "recall mode" to *false*, the analysis engine continues its work where *R* stopped, resulting in the tokens [*6*][*7*][*8*] to be processed.

[33] Setting a recall mode (to *true* or *false*, whatever) and at the same time also a frame size of 0 results in ignoring the recall mode option.

So, all in all three different cases might occur:

Next rule's frame size is set to 0.	**Next rule's frame size is set to >0.**	
	Next rule is set to recall mode = *true*.	**Next rule is set to recall mode = *false*.**
Do not move the position token. The frame's left and right boundaries are set to the current token's position.	Move the position token to the current frame's start position. The next frame will start at the current frame's start position.	Move the position token 1 token to the right. The next frame will start 1 token to the right of the current token's position.

For every search that is applied, the frame's left boundary is stored inside the search object as the start position. This position will constitute the left annotation tag in the text. The right annotation tag's position is set by either the position of a matching token or, if no token inside the frame matches, by the frame's right boundary as the Search's maximal end position.

The `match` methods of every subclass of Search work in a comparable way except for the agreement search. In the beginning, the start position is stored inside the search object. The `match` method inspects one token after the other inside the specified frame until it finds a token which matches the search's criteria. If not a single token inside the frame matches the search criteria, the `match` method stores the frame's right boundary as the end position and returns a no match.

For ever inspected token, first the token is looked up in the dictionary by using it as a key value to search for. Every subclass of Search knows automatically which dictionary to use, the word or the stem dictionary. It receives back a DictEntry object containing a list of either word objects or stem objects for the given key. The list is searched through to find whether any of the given word/stem objects match the specified criteria. If at least one does so, a match is returned, and this word/stem object is stored inside the search object as the matching one. Furthermore, the position of the matching token in the text is stored also as the matching position. If none of the word/stem objects in the list matches, the next token in the frame is inspected as long as not the end of the frame has been reached.

The stored word/stem objects are later on used for two purposes: First, when an agreement search is processed, it needs access to those word/stem objects to compare them as the left token in the text with the right token it is sitting on. Second, when the action is triggered, a

result object is created with annotations added to the stored start and matching position in case of a match or the start and end positions in case of a no match.

6.3 The Action

The Action has two main tasks: Incrementing counters and adding annotation tags to the text. It contains double entries for the match and the no match cases. Not every triggered action will increment counters or annotate a text. The behavior depends on how the action is specified in the rule file.

An action object has access to a list of all counters. The action object itself contains an array of counter IDs it should increment in the case of a match and also a similar list for the case of a no match. When triggered, first a decision is made about the search's matching success and all of the match respectively no match counters are incremented. A new result object is added to the result chain. That object again contains necessary information about where and which annotation tags should be added. To do so, every action object contains annotation labels. The annotation labels might actually be the ID of a counter of a certain stage indicator or the name of a counter of parts of a stage indicator. The algorithm of how to add annotations to the text was described in chapter 5.5.

6.4 The Dictionary

In *Direkt Profil* v1.5 the used dictionary structure nothing else than a (very big) hashtable for the word dictionary and a (smaller, but still big) hashtable for the stem dictionary. In the dictionary, a DictEntry is stored for every key. A DictEntry consists of a list of word or stem entries. A word entry holds information about inflection, part-of-speech and lemma. For the stem dictionary, the solely fact that a key is included in the dictionary suffices it the context where it is used. Maybe, in future versions of *Direkt Profil* a more advanced stem dictionary will be implemented.

The stem entries contained in the stem dictionary might in a few cases differ somewhat from what linguistic sciences traditionally define as the stem of a verb. Our intentional focus was laid on the detection of wrongly derived *participe passé*s.

The design – using hashtables as a "database" – has several serious disadvantages. One disadvantage is the relative inflexibility of the approach. Whereas modern database systems allow many different kinds of queries – including imprecise queries – a hashtable is a rather primitive approach to take. A second disadvantage is the huge amount of memory *Direkt Profil* needs at the moment to work – more than 130 MB of RAM are necessary because millions of objects are instantiated during the start-up process of *Direkt Profil* when the dictionary files are loaded, although most of the objects are never accessed during the text analysis process.

However, using a hashtable has also advantages: They are 100% pure language constructs in the chosen programming language (Java) and easy to implement and use. No database system must be installed before. And access to them is very fast.

The reason, why we used a Java hashtable and not a real, professional database system like MySQL is explained quickly: Nobody in the project team has had time until now to make *Direkt Profil* work with a database system. Of course, this is however planned for a future version of the program.

6.5 Direkt Profil's technology

Direkt Profil relies on standard technologies. The program itself is written purely in the programming language **Java**. However, since it relies on Java's regular expression processing packages, only versions of Java 1.4 and above can be used to compile the program and the program is not compatible to earlier versions of the Java Development Kit (JDK).

The rules are initially stored in an XML file (called `rules.xml` or similar) and are loaded by the program at startup time. To process XML, the program uses the **SAXParser** classes. These classes (and also the similar working DOMParser classes) are not distributed together with the JDK packages, but can be

To compile the whole program we used **Ant**, a freely available tool distributed by the Apache Group for compiling Java programs similar to the `make` program for C/C++. *Direkt Profil* is distributed with a build file called `build.xml`, which contains instructions for Ant. Personally I used Ant version 1.6.2, what worked fine on my side.

The program itself is not a standalone version but runs on a **Tomcat 4.1** server, which is also freely available under the distribution of the Apache Group. Tomcat exists both as a

standalone server and as a plugin to the widely spread Apache webserver[34]. We already run the server version under Windows (XP and 2000), Linux and Macintosh's OS X. We never tried it on other platforms, but the program should work on every platform for which there are distributions available of Tomcat.

Be aware that the current version of Direkt Profil uses approximately 130 MB of memory! To run it on the server you must assure that your computer has enough free memory space. Since the Java Virtual Machine, where the instance of the Tomcat server usually runs on, is by default set to use not more than 80 MB of memory, Tomcat's CATALINA_OPTS parameter must be set like:

```
set CATALINA_OPTS="-Xmx180m"
```

to make sure that enough memory will be provided for loading the whole dictionary. This can be specified in a file called startup.bat (in Windows) respectively startup.sh (in Linux/Unix). It is surely not a bad idea to also set the JAVA_OPTS parameter:

```
SET JAVA_OPTS="-Xmx180m"
```

This is done in a file called catalina.bat (Windows) respectively catalina.sh (Linux/Unix).

The used dictionary is basically the one distributed by ABU CNAM as described in chapter 4.6. However, we "changed its look" by bracketing the entries with XML tags to enable processing with a standard XML-parser.

[34] We recently encountered some rather mysterious problems with the server version of *Direkt Profil*. Sometimes the server did not respond to our connection attempts, although it was up and running. Only after a few minutes of waiting and several trials was it possible to receive the login screen. We assume that the problems are related to the plugin version of Tomcat, since the same problems never occurred when using a standalone version of Tomcat.

7 Analyzing texts

In this chapter we will now turn to the result of an analysis and give a short discussion about the limitations, the pros and cons of the binary rule based approach. As we will see, the current version produces good results, but is still limited strongly to verb group stage indicators led by a subject pronoun. We will outline the basic shape for a planned, future version of *Direkt Profil*. The future version will still be based on the same ideas of how to process stage indicators with a set of binary rules, but an additional layer will be implemented, what makes it possible to apply the same mechanism to stage indicators of other groups too.

7.1 Recall, Precision and F-Measure

To measure the quality of a computer-assisted language learning system, several points must be considered. First of all it is very important to recognize that a CALL system not only consists of its technical implementation details. A holistic approach to measure the quality of a CALL system would have to take into account many "soft factors" as well like user friendliness of the graphical user interface, general graphical design, quality of feedback and help functionality and many others. These soft factors are not to be neglected, because in the end they might decide upon the fact whether a user actually really works with a program or not.

However, because this paper focuses on the technical implementation details, we will concentrate on a technical measure only.[35]

The main measures of the quality of an analysis are its **precision** and its **recall**. They are defined as follows:

$$\textbf{Precision} = \frac{\text{Number of detected structures which are correct}}{\text{Number of totally detected structures}}$$

$$\textbf{Recall} = \frac{\text{Number of detected structures which are correct}}{\text{Number of available structures in the text}}$$

[35] For a further discussion of how to measure a computer-assisted learning system in general, see for instance [Lindstedt 1998].

$$\textbf{F-Measure} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

The F-Measure is a combination of precision and recall and often used to give an impression about the combined quality of precision and recall. The "structures" to be detected are of course the stage indicators. Here are the numbers for *Direkt Profil* v1.5.1:

Direkt Profil v1.5.1	*Stage 1*	*Stage 2*	*Stage 3*	*Stage 4*	*Control Group*	*Total*
Number of (theoretically) available structures[36]	23	97	101	119	85	*425*
Number of totally detected structures[37]	27	98	100	112	92	*429*
Number of detected structures which are correct[38]	15	81	89	96	73	*354*
Number of structures which were not detected[39]	5	16	12	20	11	*64*
Number of overdetected structures[40]	10	17	11	17	19	*74*
Recall	**0.65**	**0.84**	**0.88**	**0.81**	**0.86**	**0.83**
Precision	**0.56**	**0.83**	**0.89**	**0.86**	**0.79**	**0.83**
F-Measure	**0.60**	**0.83**	**0.89**	**0.83**	**0.82**	**0.83**

Table 6 Direkt Profil v1.5.1 - recall, precision and F-measure

In the table, the number of theoretically available structures corresponds to the "Gold Standard" which was annotated manually. The number of totally detected structures denotes how many structures (correct and incorrect ones) the program detected. Some of those detected structures are however the result of overdetection, which means that the program annotates a stage indicator in a place where there none. Finally, some stage indicators in the text are not found by the program – the "number of structures which were not detected".

The following equation does **not** hold exactly in every situation:

[36] This measure size is given by the manually annotated „Gold Standard".
[37] The number of structures, which were detected by the program and claimed to be a stage indicator.
[38] Out of the detected structures, only a certain percentage are indeed stage indicators.
[39] Some stage indicators as annotated in the „Gold Standard" were not detected by the program.
[40] This measure size shows the number of structures that were claimed by the program to be a stage indicators but were actually none according to the „Gold Standard".

Number of totally detected structures – Number of overdetected structures

=

Number of structures which are correct

The reason is easy to understand. Sometimes an overdetected structure prevented the detection of another (theoretically available) structure, resulting in two analysis errors: One overdetected structure and one not detected structure.

As can be seen, the results of the analysis are indeed good for stage 2, 3 and 4 (and also for the control group) with a recall between 81% and 88% and a precision between 83% and 89%. However, the stage 1 is – as expectable – the most difficult one. There the recall shrinks to around two third of all stage indicators and a precision of 56%. The reason surely must be seen in the nature of stage 1 texts. They contain lots of language mistakes of every type.
What cannot be taken out of this table is a more detailed view of the detected structures. The weakest stage indicator to be detected of all is clearly the "sentence without a verb" (counter ID `p1_t1_c0000`), that means sentences without (recognized) verbs. Here, indeed the program produces still too much overdetection. Many examples can be found here with slightly misspelled words like **Elles boient du vin.* or **Ils voulent aller à la plage.* where the writer clearly wanted to express a conjugated lexical verb but failed to do so. Another typical mistake is the detection of *ce* as a subject pronoun in constructs like *ce matin...*, where the first rule is successfully applied to the token [*ce*], but no verb follows in the frame of the next five tokens, resulting in a classification of a sentence without a verb. We hope to improve this misbehavior with the introduction of the "clips system" in a later version as described below.

We can compare these results also to those from v1.4 to see the difference:

Direkt Profil v1.4	*Stage 1*	*Stage 2*	*Stage 3*	*Stage 4*	*Control Group*	*Total*
Number of (theoretically) available structures	17	91	99	115	93	415
Number of totally detected structures	27	102	101	104	101	435
Number of detected structures which are correct	9	81	90	96	81	357

Number of structures which were not detected	7	5	8	16	19	55
Number of overdetected structures	17	19	9	24	19	88
Recall	**0.53**	**0.89**	**0.91**	**0.83**	**0.87**	**0.86**
Precision	**0.33**	**0.79**	**0.89**	**0.92**	**0.80**	**0.82**
F-Measure	**0.41**	**0.84**	**0.90**	**0.88**	**0.84**	**0.84**

Table 7 Direkt Profil v1.4 - recall, precision and F-measure

The results from both tables must be compared with care. The categorization of what should be annotated how has changed several times while upgrading from v1.4 to v1.5.1. The tables are thus not fully comparable. But besides minor "incompatibilities", what can be seen on the first sight is a significant improvement of the detection quality of stage 1 (the earliest stage). We were able to improve recall and precision at the same time, resulting in an increase of the F-Measure for approximately +19% (in absolute terms) as well. We have not inspected in detail the reasons for this increase of the analysis quality, but it is to assume that especially the possibility to find verbs with erroneous accents increased precision and recall of the notoriously bad stage indicator `p1_t1_c0000` (sentence without a verb), because before the introduction of this functionality, misspelled verbs due to accent errors were treated as unknown words and ignored by the analysis. Despite of this improvement, the analysis quality for stage 1 needs to be improved further.

The other changes in precision and recall are too small to have a significant weight, except maybe a slight decrease in the F-Measure of stage 4 (due to decreased recall and precision). It is very hard to tell, whether this decrease is a random product without significance or not.

Interestingly, the number of not detected structures was increased totally by 9 going from 55 to 64. It is to assume that this is related to the slight increase of the totally available structures. At the same time, the number of overdetected structures decreased by 14 from 88 to 74.

The decrease of totally available structures in the control group is mainly due to the fact that we took out a rule from v1.4, which was detecting common, conjugated (lexical) verbs and served as a stage indicator for "all the rest". Therein fell all conjugated verbs that did not fit anywhere else. The control group had many occurrences of verbs in *passé simple*. In v1.5.1,

this “garbage bin rule” was split up to a higher degree of precision. The passé simple should not be explicitly detected, what finally resulted in a decrease of totally available structures for the control group.

To summarize it all: We believe that with *Direkt Profil* v1.5.1 we more or less reached the limits of where significant gains can be made without doing bigger changes to the program’s core functionality. As a last functionality, we will probably implement an extended version of the stem search not only for past participles but also verbs in general, so that constructs like **ils voulent* (instead of *ils veulent*) or **ils boient* (instead of *ils boivent*) can be detected. This should (hopefully) further increase the analysis quality especially in stage 1, where such wrongly derived verb forms are still spread.

The introduction of the clips system will probably not so much result in an increased analysis quality for all stages, but rather result in a strong growth of totally detectable stage indicators that are in the tables shown in chapter 2.2, but cannot be discovered with the current program version. So the number of totally available structures in the text should significantly increase, according to our expectations.

7.2 Analysis quality for accent search and stem search

To test the analysis quality of the accent and the stem search we faced the problem that the cases, when they were applied to the text, occurred too rarely in the control group, so that we could not measure them systematically. A further problem is that the corpus is not annotated manually. Input for a systematic test might therefore not be extracted from the corpus. So, in lack of other possibilities we had to invent some examples to gather enough input to produce a more or less stable measure size. For this purpose, Suzanne Schlyter and Jonas Granfeldt produced a test set – a list of 20 incorrect sentences and 20 correct sentences – for both the stem and the accent search, based partially on some real examples in the corpus they knew were there, and also on other invented examples based on their experience as language teachers. They can be expected to be authentic language sentences from students. We then measured the analysis quality with these lists. Of course, we are aware that such a procedure has only a reduced power of prove.

1) Stem search

The incorrect input sentences contained the following list:

Test set with incorrect *participe passé*s	Test set with correct *participe passé*s
1. *Il a metté le livre sur la table.	21. Il a mis le livre sur la table.
2. *Nous a prendu les clés.	22. Nous avons pris les clés.
3. *Ils a prené un taxi ensemble.	23. Ils ont pris un taxi ensemble.
4. *Tu as pouvé voir le train ?	24. Tu as pu voir le train ?
5. *Nous a boivé du vin ensemble.	25. Nous avons bu du vin ensemble.
6. *Il avait buvé beaucoup.	26. Il avait beaucoup bu.
7. *J'ai ayé un chien pendant dix ans.	27. J'ai eu un chien pendant dix ans.
8. *J'ai aller avec avion.	28. Je suis allé en avion.
9. *Tout de suite il a rendé le argent.	29. Toute de suite, il m'a rendu l'argent.
10. *Elle ne a batté fortement.	30. Elle ne l'a pas battu fort.
11. *Il a écrivé une lettre.	31. Il m'a écrit une lettre.
12. *Nous n'a pas finissé avec le travail.	32. Nous n'avons pas encore fini le travail.
13. *Vous avez lisé ce livre ?	33. Avez-vous lu ce livre ?
14. *Tu as pas rié au film ?	34. Tu n'as pas ri au film ?
15. *Il a savé que c'est vrai.	35. Il a su tout le temps que c'était vrai.
16. *Nous avons le saché tout le temps.	36. Nous l'avons su tout le temps.
17. *Je ai souvené mon école maintenant.	37. Je me suis souvenu de mon école maintenant.
18. *Il as tené un sac dans la main.	38. Il a tenu un sac dans la main.
19. *Elle a appellé son chien.	39. Elle a appelé son chien.
20. *Nous a parler longtemps.	40. Nous avons parlé longtemps.

Table 8 The stem search's analysis quality

It can be seen that all the wrongly derived forms usually follow a systematic pattern, where a student took an existing word stem (*pren-* as in *prenait*, *pouv-* as in *pouvons*, *boiv-* as in *boivent*) and added *participe passé*'s ending. The stem search (implemented for *passé composé*s and *plus-que-parfait*s) should thus be able to detect the cases with a certain regularity of how these wrong *participe passé*s are derived.

Marked in a green color[41] are those constructs, where either a *passé composé* or a *plus-que-parfait* was detected correctly. (As stated above, the stem search is not applied or implemented for any situations except for the detection of a *participe passé* – at the moment these are only *passé composé*s and *plus-que-parfait*s.)

The red marked sentences are such, where the analysis did not identify something. In two cases, a wrong form of *participe passé* with an equal spelling to the infinitive of the verb (**J'ai aller…* and **Nous a parler…*). In such cases, the program treats the wrong *participe passé* simply as an infinitive and cannot recognize a *passé composé* or *plus-que-parfait*. The third case however is interesting, because **Elle a appellé* is clearly meant to be a *passé*

[41] Be aware that the chosen colors in this chapter have no relation to the colors used for highlighting the analysis results in the program itself.

composé, but the stem search cannot find something because *appell-* is not contained as a regular stem of a verb in the stem dictionary (only *appel-* would be contained).

For the correct sentences on the right side, the situation differs. Usually, for these cases the stem search never gets triggered, because a regular *passé composé/plus-que-parfait* has been found. The correct examples therefore do not show much about the analysis quality of the stem search. They are rather added here for completeness.

(The yellow marked *vous lu* in sentence 33 is another wrong analysis of the program, and it should of course be *avez-vous lu* recognized as *passé composé* with inverted word order. At the moment, *Direkt Profil* is not able to detect such an inversion, thus the *passé composé* is not detected but the program claims *vous lu* to be a sentence with "lexical, not conjugated verb", counter ID `p1_t1_c4100` in the annotation ontology.
The other yellow marked structure *ce livre* in sentence 33 is a misclassification that happens quite often, which cannot be detected without further analysis. The program recognizes the token [*ce*] as a subject pronoun and then tries to find a verb. Since there is a verb suitable entry *livre* – 3rd person singular, *indicatif présent* – from the verb *livrer* in the dictionary, the program assumes to have met a similar structure like *il livre*. If taken together with the (semantic) context in the sentence, *ce livre* cannot be a subject pronoun + verb combination. However, from a computational point of view, this classification could also be noted as correct. We expect to treat such situations accordingly with the introduction of the clips system as described below.
Detected but not highlighted in the table above are stage indicators or multi word expressions not related to the stem search in any way, for instance in sentence 35 the stage indicator *c'était* or in sentence 15 the multi word expression *c'est*.)

To summarize, the stem search seems to deliver reliable results once triggered and in cases where the wrongly derived stem does not contain too much of irregularity as in the examples without a regular *participe passé*'s ending.
Since these are atomic test examples, they have only partially a power of prove. The analysis engine is a complex piece of software, and what is neglected in such isolated test examples is the possibility of interference as a result of the engine's algorithmic behavior. An open question is whether the stem search is actually triggered "at the right time" – which means

how it works in the context of the whole analysis of a text. This is a question that cannot be answered here.

2) Accent search

For the accent search, the test set was the following:

Test set with misspelled words due to accent errors	Test se with correctly placed accents
1. *Elles reve d'un voyage en Italie. 2. *Arrivée en Italie, elle déscend à l'hôtel. 3. *Si tu reussis tes examens cette année alors tu pars en *Italie pour les vacances d'été. 4. *Il m'ecrit souvent d'Italie. 5. *Elles achétent des souvenirs pour tout le monde. 6. *En fait il régardé la télé tous les jours. 7. *L'été dernier, il a eté en Italie avec ses amis. 8. *Il repond pour dire qu'il ne veut plus. 9. *Elles pensè de garcon est du soleil 10. *Je e´cris à mon frère en Italie. 11. *Le jour après, elles decident d'aller à la plage. 12. *Elle a parle d'un copain qu'elle connait. 13. *Il creé une très grande entreprise. 14. *Après avoir mangé, nous etions tous très contents de l vie. 15. *Je ne me séntais pas à l'aise dans ce groupe. 16. *Il a prèféré rester sur son île. 17. *Je vous prèfère al la appartement, c´est grande et belle vieux. 18. *J'habité Malmö pendant 12 ans. 19. *Souvent, nous ecoutions des disques ensemble. 20. *Mon frère est vantard, il exagêre tout!	21. Elles rêvent d'un voyage en Italie. 22. Arrivée en Italie, elle descend à l'hôtel. 23. Si tu réussis tes examens cette année alors tu pars en Italie pour les vacances d'été. 24. Il m'écrit souvent d'Italie. 25. Elles achètent des souvenirs pour tout le monde. 26. En fait il regarde la télé tous les jours. 27. L'été dernier, il a été en Italie avec ses amis. 28. Il répond pour dire qu'il ne veut plus. 29. Elles pense au garcons est au soleil. 30. J'écris à mon frère en Italie. 31. Le jour après, elles décident d'aller à la plage. 32. Elle a parlé d'un copain qu'elle connaît. 33. Il crée une très grande entreprise. 34. Après avoir mangé, nous étions tous très contents de la vie. 35. Je ne me sentais pas à l'aise dans ce groupe. 36. Il a préféré rester sur son île. 37. Je vous préfère. 38. J'habite Malmö depuis 12 ans. 39. Souvent, nous écoutions des disques ensemble. 40. Mon frère est vantard, il exagère tout!

Table 9 The accent search's analysis quality

Here the accent search should be applied in two (respectively three) cases:

- To find a verb in a second attempt, if otherwise the analysis has failed to find on in a first attempt (because a misspelled verb is an unknown word),
- To find *participe passé*s with missing or wrong accents for *passé composé*s and *plus-que-parfait*s, mostly due to a missing accent on the ending.

As for the stem search, the green highlighted structures are detected correctly by the accent search. Usually, an accent was simply not set, the wrong accent was chosen (e.g. *accent grave* instead of *accent aigu*) or the accent was placed on the wrong letter.

The red structures are those not detected correctly by the analysis engine. A deeper inspection of the input sentences shows that in nearly all cases, the accent search never got triggered.

The misdetection is caused merely by the precedence of rules in combination with the static and sometimes too big frame sizes. For instance in the sentence **Elles reve d'un voyage.* first the token [*Elles*] is found. Second, a frame with the size of 5 tokens is set up: [*reve*][*d*][‘][*un*][*voyage*]. Third, the engine searches for a verb inside the frame. The first token in the frame is [*reve*] which is not in the dictionary and thus an unknown word. It continues its analysis on the other tokens and finds *voyage* as a regular 3rd person singular, *indicatif présent* of *voyager*. It combines *Elles* with *voyage* and applies an agreement check to these tokens. The accent search is never applied on [*reve*]. The same situation can be found in

- Sentence 8 (*Il…dire*): The case is similar to the described one. The token [*repond*] is not recognized as a verb in a first attempt, but [*dire*] is. The program categorizes as a "common, not conjugated verb", counter ID `p1_t1_c4100`.
- Sentence 9 (*Elle…est du*): This is an interesting case. In this case, several problems cause the wrong classification. First, the combination *Elle…est* is detected. The token [*est*] was actually meant to be [*et*] but misspelled by the writer – as is the sentence's main verb: *pensè*. According to the precedence of rules in combination with the frame size, *Elle…est* is detected instead of *Elle pensè*, where an accent search should have been applied. The situation looks for the analysis engine like the beginning of a *passé composé*. It tries to identify a stem in a frame of 3 tokens after the verb *est*, fails, uses an accent search to identify a misspelled *participe passé*, fails again and finally applies a stem search to the same frame. Because there is a regular stem *d-* in the dictionary for different verbs, and because *–u* is a regular ending of a participe passé, the engine guesses that a wrongly derived stem is encountered, and that what the author wanted to express was a *passé composé*.
- Sentence 11 (*elles…aller*): Same case as in sentence 8.
- Sentence 12 (*Elle a*): No passé composé is found with the accent search, because the misspelled *participe passé parle* in the frame [*parle*][*d*][‘] is already an existing verb form, therefore the accent search will skip this word.
- Sentence 13 (*Je…entreprise*): Similar case as in sentence 8 and 11. The only difference is that *entreprise* exists as a conjugated 3rd person singular, *indicatif présent* verb in the dictionary.
- Sentence 15 (*Je ne*): In a first attempt, no verb is found. Second, an application of the accent search on the same frame returns *né* (English *born*). The preliminary negation *ne* itself is not contained in the dictionary, thus *ne* is treated as an unknown word and the accent search finds a *participe passé* of *naître* instead.

- Sentence 18 (*Je…pendant*): Similar case as in sentence 8, 11 and 13. *pendant* is a *participe présent* of *pendre* in the dictionary.
- Sentence 19 (*nous…disques*): Similar case as in sentence 8, 11, 13 and 18. *disques* is a 2nd person singular, *indicatif présent* from the verb *disquer*.

Sentence 10 is a technical problem related to encoding, which might occur sometimes. On Swedish keyboards – the texts in the corpus were written in Sweden, except the ones in the control group –, the accents sometimes seem to use a different character encoding. Transferred from one system to the other, this might result in undesirable situations like *e´cris*.

In the group of correct sentences, the underlined structures are examples of correctly detected stage indicators.
The yellow highlighted structures were not correctly detected for reasons not related to the accent search. We saw the problem connected to *ce groupe* already earlier with *ce livre*; there is a verb *grouper* in the dictionary. In the cases of *tu réussis* and *Il m'écrit* once more the problem comes from the order, how the rules are applied to the text, because they are first detected as *participe passé*s, before they are checked whether they could also be common conjugated verbs.

As a conclusion for both the stem and the accent search, we can state that they must be seen in the context of the whole analysis. It does not make much sense of testing a single stem search rule or accent search rule to some sentences or even tokens. As can be seen, the order or how the rules are applied plays an important rule of what will be detected or not. If the program reaches a point where they are really applied, then the results are normally reliable.

7.3 Examples of analyzed texts

To give an impression of how an analyzed text looks like, two analyzed texts are shown now. The colors are reproduced exactly as the text would be highlighted by *Direkt Profil*. Brackets with numbers are inserted artificially to give an explanation further below. The indicated subject levels are not a result of the analysis process, but this is how they are classified in the CEFLE corpus (see chapter 4.1 for an explanation).

Author: Hans – Subject Level 1

C'est[1] un trés chaud jour au été. **Il y a**[2] un garçon sur une petite île. Il s'appeler[3] Michel. **Il y a**[4] une vert maison avec un bleu porte, le garçon habiter a la maison. Le garçon aimer le fleures avec la île. Il arrive[5] á la île dans un petite bateau. Un jour quand Michel ~~il vouloir builder un bridge~~[6] au une different île, arrive deux garçons. Les garçons builder un ville dans la different île. Le ville est trés grosse. Michel est malheurese, parce que le voisin-île est trop de grosse.

Correctly detected:

[1], [2], [4]: p0_t0_c0000: A multi word expression is found.

[3] p1_t1_c4100: Found a sentence, where the verb has an infinite form.

[5] p1_t1_5132: A common conjugated verb in *indicatif présent* agreeing in person and number with the subject pronoun.

Cases to be discussed:

[6] What is found here is actually a p1_t1_5121, the auxiliary modal verb *vouloir*, which does not agree in person and number with the subject pronoun *il*, thus the internal classification is correct. However, the highlighting goes too far (till the word *bridge*) and gives a wrong impression. This is an algorithmic problem we have not yet solved completely, though the suggested analysis is correct.

Conclusion: Out of 6 structures, 5 are detected and classified correctly, as while as one is only classified (but not highlighted) correctly.

Of course, if a text has only such a few structures to be detected (out of which 50% are multi word expressions), a computation of stages has only a limited explanatory power. The text contains no advanced verb group stage indicators at all. Three of the (correctly detected) stage indicators are multi word expressions, the other two are a *présent* and an *infinitif*. This is surely not an advanced language learner. Without certainty, we could guess that the learner belongs probably to stage 1 or 2, but not to a higher stage.

Here is another analysis of a subject level 4 text.

Author: Inga – Subject Level 4

J'étais[1] en 7ème et j'avais[2] 13 ans. J'aimais[3] beaucoup la France, et voulais certainement apprendre le français. Le prof étais sympa et on passait[4] des leçons très agréable. Il n'y

avait[5] pas beaucoup des élèves et on apprenait[6] vite. Les premieres mots c'étaient[6] le jour de la semaine, oui et non et des trucs assez simple. J'aimais[7] bien aller en cours en collège. J'aimais[8] bien le français, et j'étais interéssée[9] et les cours n'étaient pas seulement des leçons d'école. Quand j'avais[10] 16ans j'ai commençais[11] le lycée. / j'ai commencé[12] le lycée ./ J'aimais[13] toujours la France comme pays, mais je suis devenue[14] assez lasse à l'école, en cours de français aussi. On faisait presque rien pendant[15] les cours ou bien on faisait des choses très[16] ennuyants et les élèves trouvaient ça nul.Mais comme je ne voulais plus travailler[17] à la maison (il y avait[18] trop des devoirs!!!) je décidais de passer[19] un an d'ailleurs. Je voulais quitter[20] la Suède qui m'ennuyait[21]. ~~Je me souvient~~[22] excatement comment c'était pris[23], la décision. Pendant un repas avec la famille j'ai crié[24] avec haute voix, en mangant que j'allais[25] aller en France pour faire des études. Mes parents étaient contentent. J'ai pris[26] l'avion le premier Septembre 2000. J'étais[27] hyper nerveuse parce que j'allais[28] habiter chez une famille française toute l'année! Mais, bon c'était[29] très bien, ils étaient[30] très gentilles avec moi. Je suis allée[31] au lycée français et ils m'ont mis[32] en premier L (littéraire) un. **C'est**[33] à dire 6 heures de français par semaine!!! C'était[34] très dur, mais j'ai beaucoup appris[35]. Non seulement la lange mais plein des choses dans la littérature française. bon, on doit términer.[36] J'espère[37] que ça ~~vous a aidé~~[38].

Correctly detected:

[1], [2], [3], [4], [5], [6], [7], [8], [10]. [13], [18], [21], [25], [27], [28], [29], [30], [34] p1_t1_c5400: *Imparfait*, where the conjugated verb agrees in person and number with the subject pronoun.

[12], [14], [24], [26], [31], [32], [35] p1_t1_c5200: *Passé composé*, where the conjugated verb agrees in person and number with the subject pronoun.

[17], [20] p1_t1_c5300: Auxiliary modal with agreement of person and number between subject pronoun and verb, followed by a verb in *infinitif*

[22] p1_t1_c5131: A common conjugated, *indicatif présent* verb is found that does not agree in person and/or number with the subject pronoun (therefore it is crossed through).

[23] p1_t1_c6120: A *plus-que-parfait* is found where the conjugated verb agrees in person and number with the subject pronoun (*c'*).

[33] p0_t0_c0000: A multi word expression is found.

[36] p1_t1_c5122: An auxiliary modal verb in *indicatif présent*, agreeing in person and number with the subject pronoun, is found followed by an infinitive.

[37] p1_t1_c5132: A common conjugated verb in *indicatif présent* is found with agreement between the verb and the subject pronoun.

Overdetection or not detected:

[15], [16] p1_t1_c5422: An auxiliary modal verb (correct: *faisait* is internally classified as an auxiliary modal verb) followed by an infinitive is found (incorrect).

[11] It is uncertain what the author wanted to express here, a *passé composé* or an *imparfait*. However, the case needs a further inspection, because at least *j'ai* should have been detected as a p1_t1_5112 (conjugated form of *avoir* agreeing with the subject pronoun).

[19] p1_t1_c4100: Found is *je…passer* – a sentence with an infinite verb. *décidais* is not recognized. The result is the wrong classification. Be aware that this example causes recall and precision to decrease: First, one stage indicator has not been detected (*je décidais*) for the reason of misspelling, second *je…passer* is detected instead and by mistake counted as a p1_t1_c4100.

[38] p1_t1_c5210: A *passé composé* is found where the conjugated verb does not agree with the subject pronoun. The error is caused by the fact that at the moment *ça* is not treated as a subject pronoun by *Direkt Profil* and *vous* is detected as the subject pronoun instead. Again, this situation causes a decreasing precision and recall at the same time.

Cases to be discussed:

[9] p1_t1_c6130: What is detected here is an occurrence of a *passé composé*, where the auxiliary verb *étais* agrees in person and number with the subject pronoun *je*. Furthermore, the *participe passé interéssée* is subject to accent error(s) (the accent search guesses that *intéressée* is probably meant instead). Now, it can be discussed whether this categorization is correct or not, because one could argue that *être intéressé* is not a *passé composé* but rather a verb + adjective.

Conclusion: Out of 38 to be detected structures, Direkt Profil detected 32 correctly and failed in 5 situations. Structure [9] is not clear how to be counted.

If both analyzed texts are compared, it is easy to see that the second text uses more advanced forms of verb group stage indicators. The *imparfait* is spread all over the text, even the *plus-que-parfait* can be found. Without knowing the exact stage at this point in time and by using the table in chapter 2.2 we can guess that the second text cannot be a text of stage 1, because

the imparfait will not appear on stage 1. Taking the *imparfait* alone already results in a first guess of stage 3 or stage 4. Together with the other stage indicators, a more precise text analysis is possible.

There is also a general difference in the highlighting colors between the texts. In texts of later stages, the colors are in general chosen to be darker than in earlier stages' texts. This already gives a first impression of the language skills the author must have had. The next step to be done would be to compute the most probable stage for both of the texts.

7.4 Discussion of the frame size problem

A central unsolved issue from a linguist's point of view to be faced with *Direkt Profil* v1.5 is the **frame size problem**. As seen above, for every rule's search a frame size must be specified. For the root rule, the frame size can be set to "infinite" with a value of `max`. A zeroed value will tell the analysis engine not to continue but to stay for application of the next rule on the current token.

The question arises to which value the frame size should be set for a specific rule. When a pronoun token is found already and the program should try to detect a verb token, how many tokens might be inserted between the pronoun and the verb to know that they still belong together? Three tokens? Or maybe five tokens? Or as many tokens as possible, but not longer than a sentence lasts? Imagine the tokenized sentence [*Il*][*a*][*un*][*pantalon*][*volé*][.]. From the context it is clear that it is not the *passé composé Il a…volé* what is expressed in this sentence but the word *volé* must be seen as an adjective that belongs to *pantalon.* In *Direkt Profil* v.1.5's rule tree, the rule that looks for a *participe passé* specifies a frame size of 3. In the given example, the analysis engine will overdetect a *passé composé Il a…volé* as a stage indicator. From an algorithmic point of view, this is a correct indication, but from a linguistic perspective the result is undesirable. On the other hand, if the frame size here would be reset to 2, other cases might be missed because now the *participe passé* lies out of the reach of the rule's frame size. As far as we know, no systematic research has been undertaken to solve such questions. At the moment, the frame sizes are estimated values[42].

The frame size problem is a direct consequence of the chosen approach. The fact that the frame sizes must be specified statically in the rules file already at start-up time and are not

[42] This is of course only true for frame size values set to 0 < frame size < max.

adapted dynamically to the processed input text prohibits a more accurate control over the analysis process. It is further an open question whether the possibility of specifying a single frame size value for every encountered situation is a sufficiently precise tool at all.

The frame size problem can be solved with the introduction of the “clips system” as described below.

7.5 Theoretical and practical limits of our approach

The frame size problem is not the only limitation of the chosen binary rule based approach. We have not yet discussed two other very important points.

1. What happens if the sentence’s subject is not a pronoun but a noun, maybe in combination with an adjective, instead? (Example: *Le chien aime les fleurs.*)
2. What happens when the order of the words is changed, as it is for French often the case in questions? (Example: *Est-elle allée à la maison?*)

In the first example, the rules will not detect a pronoun when processing the sentence and not report anything. The fact that there is a conjugated verb *aime* agreeing in person and number with the sentence’s subject *Le chien* will simply be neglected. A solution to this problem with only the given approach is not possible. Although one could easily write a rule following the no match case of the root rule (looking for a pronoun) which checks a second time for a noun also, the chance would be too high that a met noun is actually not the sentence’s subject but any other noun in the sentence. It is not decidable with the implemented functionality whether a met noun is a subject or not. Another mechanism than solely the rule based approach is needed here.

In the second example, the rules will detect the token [*elle*] as a pronoun but skip the leading token [*Est*]. The analysis engine will not discover the “sprayed” *passé composé* of *elle est allée*. At least, in this situation it would theoretically be possible to write corresponding rules to detect both token orders [*elle*][*est*][*allée*] and [*est*][-][*elle*][*allée*]. A drawback would still be that we would have to write more or less the same rules twice only with a different order of what is checked first, the pronoun or the verb. This would lead rapidly to lot of redundancy and complexity in the rule tree.

As with the frame size problem, these two issues can be solved (to a certain extent) with the introduction of a “clips system” that will be described in its basics below.

Some other problems are of a more general nature, for instance some fundamental problems related to natural language and NLP, algorithmic behavior problems and the like.

Orthographic mistakes are a big problem for every written text processing software system. If in Direkt Profil a word is misspelled, as a consequence it cannot be found in the dictionary. *Direkt Profil* in some cases follows a strategy of applying rules with relaxed constraints if the predecessor rules should have not matched. The accent search for finding word tokens with missing or wrongly set accents is such strategy, which re-inspects a certain frame of tokens on the base of relaxed constraints.
However, no further functionality is implemented at this moment in the program. Manipulation techniques for traditional character strings with a ranking of every conducted manipulation step could be added as a further subclass of Search to find unknown words due to other orthographic mistakes.

Another problem arises through the capacity of the dictionary. Although it contains approximately 300’000 French words and whereas in daily language a vocabulary of a few thousand words is usually enough to express oneself in conversations, still certain words might not be included in it and therefore drop out of the analysis. Names (like names of persons, places, brands, companies etc.) are often used as a sentence’s subject, but they are not contained in the dictionary. This will be a problem to face later on. For *Direkt Profil* v1.6, it will still be set aside.
Furthermore, it cannot be excluded that the dictionary contains errors or missing information introduced through human failure. Indeed, during the work with the dictionary, the project team already detected and corrected a couple of mistakes.

As seen above, often problems arise through the order of how the rules are applied to a text. Dependent on which rule comes first, the result of the analysis might differ. Different improvements for this problem might be thought of. Once more, we expect that the introduction of the clips system in *Direkt Profil* v1.6 will solve the problem to a certain extent.

Syntactical mistakes are not an aching problem for the analysis engine. Because the engine is parsing only parts of a sentences and never the sentence as a whole (like a full parser would do trying to extract a full syntax tree of the sentence), the analysis engine should still be able to produce at least parts of results for incomplete or erroneous input too. Though, if the syntax errors are too widely spread over the whole input sentence, then the analysis engine might miss a construct that should otherwise be counted as a stage indicator.
At the moment of writing it is not sure whether and to what extent the nature of a binary rule tree could lead to problem when dealing with syntactical errors. As seen above, the engine parses from the left to the right in a sentence applying first the root rule to every token. If the natural order of tokens is changed due to syntactic errors, the analysis engine the consequence can be that the analysis engine misses an important token and therefore a stage indicator, because at the current moment it is looking for another one, coming only later in the sentence. This point lies close to the rearrangement of words in questions and "Q-sentences" (*Que? Quand? Qui? Quoi?* Etc.) as pointed out in the beginning of this chapter. Stage indicator 17 from the stage indicator table in chapter 2.2 inspects where the object pronoun is set in a sentence. The sequential, binary rule approach is insufficient to cover this situation.

Further points currently uncovered (but in a way related to verb group stage indicators) are

- the cliticisation (*je + ai → j'ai, le + enfant → l'enfant*),
- the relative placement of object pronouns
- and the negation

– these are all relevant stage indicators that cannot be detected. The sequential application of rules to the text as presented in this paper does not seem to be sufficient to cover such problems, at least not the latter two.

It can be discussed on a general base if a sequential, binary rule tree is an appropriate mean to process natural language. Sometimes, rules are inserted in the rule file redundantly, just to cover a single scenario differing only slightly from another one. A sequential, binary rule tree does not allow a person to reuse a rule if the analyzed phenomena should be counted separately by each rule.

But despite of all these drawbacks, we are quite satisfied with the results of the current version of *Direkt Profil*. As the precision and recall statistics show, in most cases the program produces good results.

7.6 The clips system

We will show here shortly the baselines of an improvement planned for *Direkt Profil* v1.6, which we call the introduction of a **clips system**. **Clips** is a term mainly used by Jacques Vergne, sometimes the same idea is also referred to as **chinks and chunks**. As described above, the current binary rule system alone comes to its limits as soon as we want to process not only verb groups anymore but also noun group stage indicators, and also when the order of the noun and verb groups is not predefined anymore. Also setting the frame sizes can be problematic. Another problem arises the with precedence order of the rules in the rule tree. We have thus recently started working on the next version of *Direkt Profil* where we shall introduce an important amendment. The basic idea is to split up the sequential analysis of a sentence into a multi step approach, parsing several segments and putting the pieces together.

First, groups are detected. These can be noun groups, verb groups, adjective groups or any other group we would like to detect. In a pre-process, The program looks for special "stop words", we call them the **clips tokens** or simply **clips**. A clip is a token that delimits a certain group from another. The program has access to a list of clips words, specified by the user in a special rule besides the tokenization rules. Clips split up the sentence into many small parts, the **segments**. A segment might also contain only a single token. The segments can again be put together to a **group**. The groups should correspond to noun, verb and adjective stage indicator groups.
Strong candidates for clips are for instance many coordinating conjunctions (*et, ou, mais…*) and articles (*le, la, les, une, un…*), but also many non-word tokens like punctuation signs[43]. The idea of the clips is not exactly the same as the determiners introduced in chapter 5.2. The introduction of the determiners and non-words was necessary to reduce the frame size artificially when the analysis engine came for example to the end of a sentence. Then the program should not continue to look for something over the borders of the sentence. This was a rather rough means to ensure that the engine respects the borders of the sentence. Groups were identified only very imprecise through non-word tokens. The clips on the other hand operate on a much more detailed level. They really produce a set of groups and do not just steer the algorithmic behavior of the analysis engine.

Second, every group is analyzed independently with the traditional approach of the binary rules described here in this paper. The frame size is simply set to the size of a group. In this

[43] See [Vergne 2004] for a short discussion about the topic.

way, a linguist does not have to specify the frame size for every rule statically in advance. It is decided upon dynamically during the analysis process. Annotation tags can be added to each group.

Third, the relations between the groups are analyzed. Does a noun group agree in person and number with the main verb? Does an adjective group agree in number and gender with a noun group? Such questions can be solved during this step. Annotation tags can be added or existing annotation tags can be changed over the range of several groups.

With *Direkt Profil* v1.5.1, the token *ce* in principle forces us to choose between two alternative treatments: Either we consequently treat the token [*ce*] as a subject pronoun and thus return with a wrong analysis result situations like *ce matin…*, or alternatively we never treat [*ce*] as a subject pronoun, what produces wrong results for other cases. With the introduction of a clips system, *ce* should be classified depending on the segment type it belongs to. If once classified, the program automatically applies an analysis for a noun group stage indicator or a verb group stage indicator (or any other) instead of blindly guessing.
Many words show this or a similar problem.

At the moment, the details of this approach are not yet worked out. But what can be seen already is that the frame size problem can be solved elegantly: A frame is always set to the size of a segment. Inside a segment, the rules are simply applied to all of the tokens not yet analyzed. The question, how far to continue with the analysis, does not arise.
Also the severe limitation to processing only (verb group) stage indicators beginning with a pronoun will cease to exist. As a consequence, we will be in need of a mechanism to propagate relations between segments to check agreements. The agreement search will probably be replaced by something else.
It is hard to say, to which extent the order of the single groups will still be a problem (but even if still problematic also with the clips approach, at least all different groups – and not just verb group stage indicators – shall be processed).
We hope to improve the analysis quality of the stem search and accent search also. The reason is that we expect problems to weigh less, which are related to the precedence order of the rules. We will have to deal with many small segments instead of a few big frames. Therefore, the accent and stem search will be applied more concisely to single tokens, which, as we hope, will result in a higher recall.

7.7 Generating statistics and computing the stages

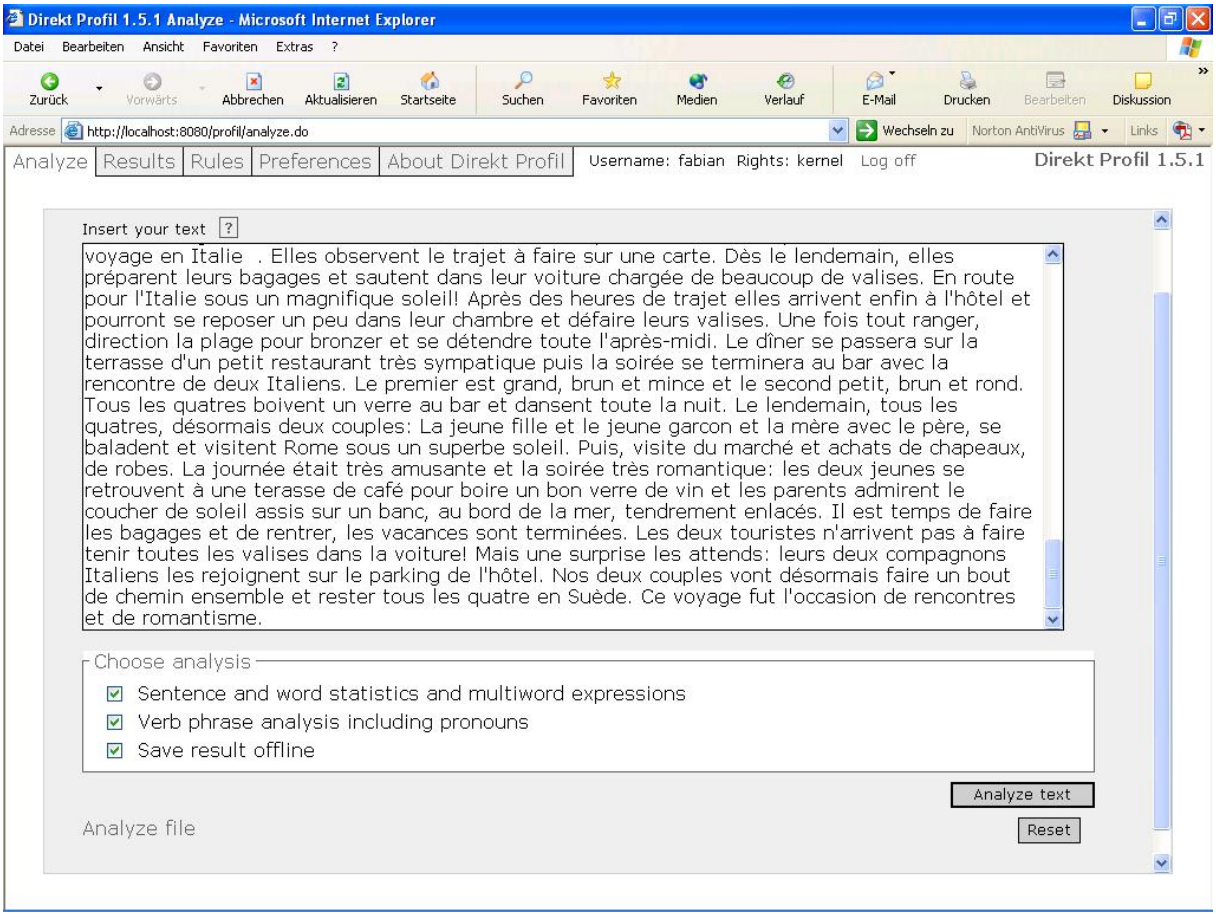
Once a text is analyzed and annotated and all necessary counters are incremented, a last thing remains to be done: computing the stages. During the analysis, the program gathered much information. Besides the counters, which all represent different aspects of stage indicators, further statistical measures were collected in the background: The number of words in the input text, the number of sentences etc. The program has enough information to compute in a final step the stage of a language learner. This functionality is not yet implemented, because there are too many things left unclear at the moment. Besides the fact that the important noun group stage indicators are completely left out from the analysis in *Direkt Profil* v1.5.1, questions related to computing statistical measurements are not yet clarified. These questions are not so much of a technical nature, but concern the theory of the language acquisition process. Some further research is necessary first.

On the other hand, the result window with the analyzed text already gives a vague hint for a user about her stage. In v1.5.1 we started coloring the annotated text following a simple scheme: The more advanced a certain detected structure is, the darker will be the color with which it is highlighted. This might give a first and easy to glimpse impression of a learner's language skills.

7.8 Working with Direkt Profil as a user

Direkt Profil can be accessed through any common web browser like Internet Explorer or Mozilla Firefox. Because *Direkt Profil* offers an interactive user interface, JavaScript must be turned on in the browser's preferences/options. The usage of a web browser interface makes further installations on the user's side unnecessary. A running version of *Direkt Profil* can be found at http://www.rom.lu.se:8080/profil/.

If a user wants to work with *Direkt Profil*, she first has to log in with her username and a password. (In the given address, a new username can be registered.) The user enters the input screen:



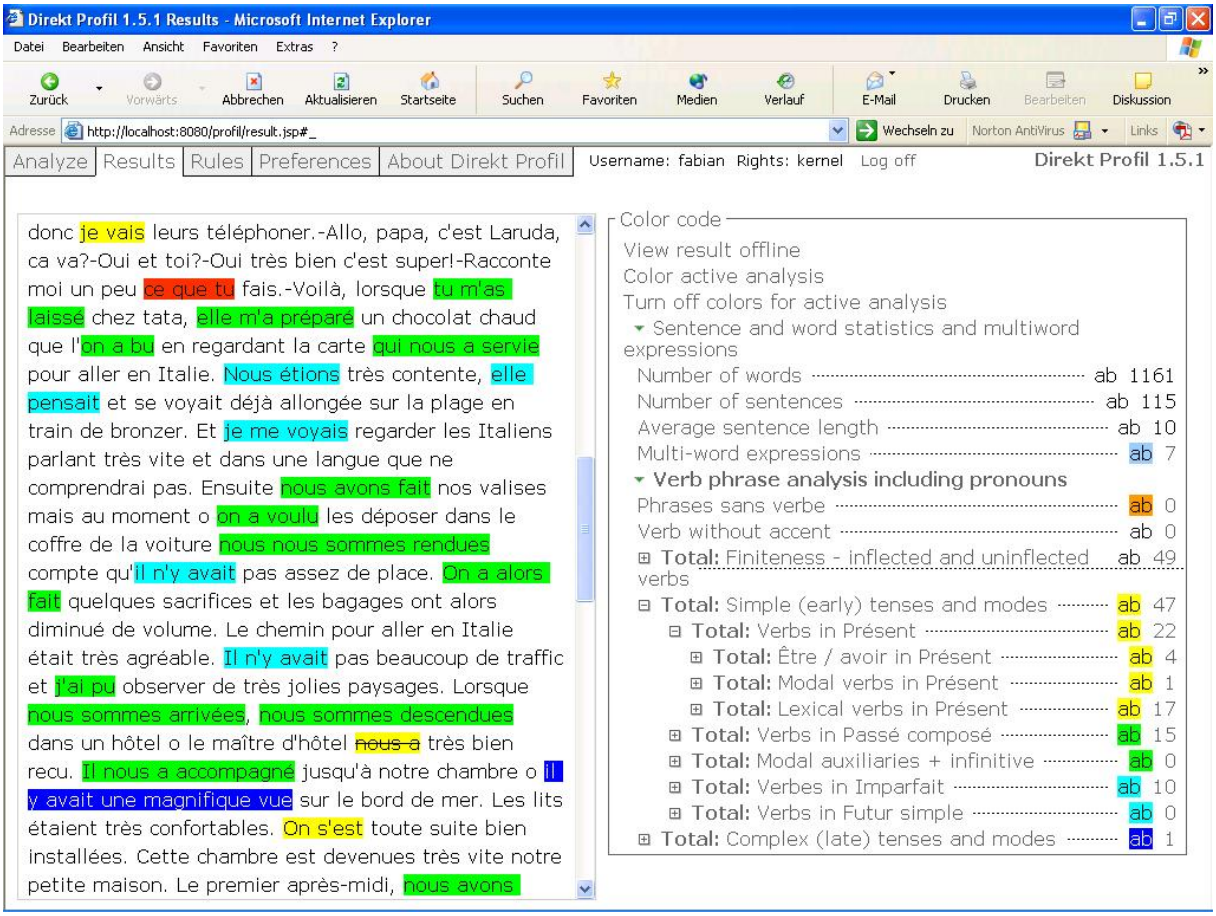
Picture 8 The input window

She can type or copy + paste her text into the input field. There are three checkboxes which can be marked.

- *Sentence and word statistics and multiword expressions:* If this checkbox is marked, multi word expressions are searched in the text. If a user is not interested in the detection of multi word expressions, she simply does not mark the checkbox. Every eventually occurring multi word expression will not treated specially, thus for instance the multi word expression *je m'appelle* will be treated as a normal stage indicator with a conjugated verb agreeing with the pronoun.

  Furthermore the user can toggle on and of with this checkbox some statistics related to the average sentence length, the number of words in the text etc.
- *Verb phrase analysis including pronouns:* If this checkbox is marked, stage indicators are detected.
- *Save result offline:* If this checkbox is marked, an additional accessible webpage is generated on the server's side that contains no JavaScript and is thus not interactive. The webpage can be saved locally to store the analysis result.

The user then presses the **Analyze text** button. The text is transmitted to the server and analyzed. The user receives a webpage similar to the following one:

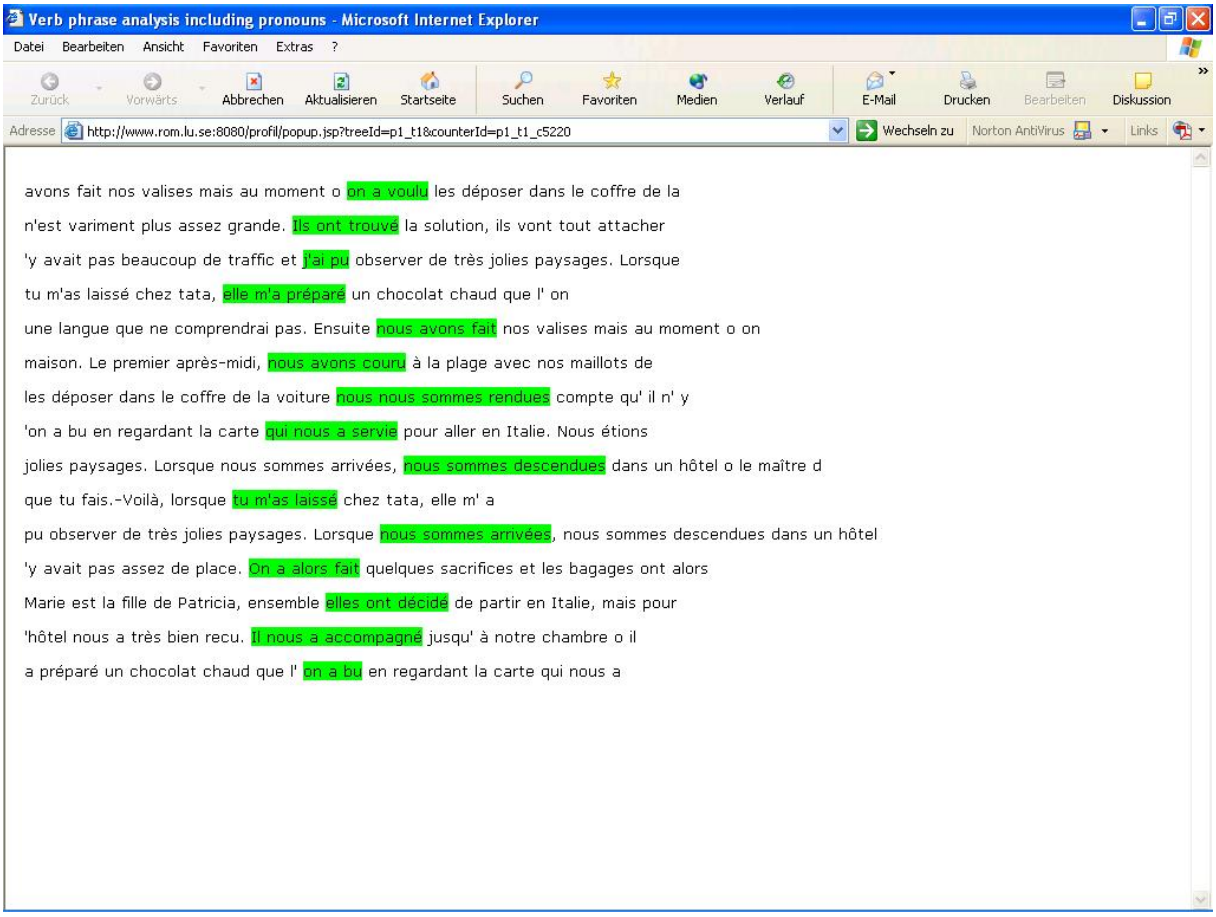


Picture 9 The output window

The result webpage consists of two parts: On the left hand the user sees again her text, where interesting phenomena are highlighted with different colors. The right side is the interactive part. In the top of the frame, some special features can be found for turning on and off all colorized phenomena and for viewing a (not interactive) result window without any JavaScript involved. This is followed by some general statistics about the number of words, the number of sentences and the average sentence length. Third, the counter for multi word expressions can be found, and finally the real stage indicators.

As can be seen, some of the counters are summed up by different more atomic counters. For instance the counter "*Être / avoir in Présent*" is summed up by two counters: One that counts *être/avoir* verbs in *présent* with agreement in person and number between the sentence's subject pronoun and the conjugated verb, and a second counter for the same situation but where the conjugated verb does not agree in person and number with the sentence's subject pronoun.

In the text shown on the left side, the highlighting for different phenomena can be turned on and off by pressing the corresponding counters. If the denotation number, which shows how many occurrences of a stage indicator have been found, is pressed, an additional window opens with all the detected occurrences of the stage indicator:



Picture 10 Occurrences of a certain stage indicator

The user thus receives a detailed report of all the detected stage indicators. The specific colors do not have any further intentional meaning, except one: Stage indicators for more advanced phenomena in the language acquisition process are highlighted in darker colors. For instance, a *conditionnel*, a verb form that is usually not mastered in the early stages of the language acquisition process, is shown in dark blue, whereas a common conjugated verb is marked yellow.

As said above, at the moment no concrete functionality is implemented yet for effectively computing the language skill level out of the detected stage indicators.

7.9 Improvements done to the program during my master thesis

I developed the idea to use a binary rule tree structure for the detection of stage indicators in autumn 2003 during a two weeks project in a course about computational linguistics hold by Pierre Nugues. I was confronted with the problem how to implement a piece of software that could detect at least some of the stage indicators for the language acquisition process as they were presented to me by Suzanne Schlyter and Jonas Granfeldt (the project's "principals"). Of course it would have been possible to simply program those stage indicators in a programming language like Java in one way or the other. However, this approach would have had serious drawbacks: Once programmed, the rules could only be changed by writing program code and recompiling it. Not only would this be a rather tedious and laborious task, but the main issue would be that nobody except the programmer herself would actually be able to do any changes to the stage indicators inherent to the program code.

Furthermore, they could not be adapted easily, if for instance another language should be added to the same program (not that I really took into account at that time that something like this could happen in close future, but still I wanted to have a more "beautiful" solution to the problem).

I tried hard to identify a most general structure behind as many stage indicators as possible and found that they all have two attributes in common: They all search in a text for something or try to match a couple of words against a certain condition, and they all should annotate the text if the condition was met. Based on these requirements, I developed the concept of a rule tree, where every node corresponds to a rule with a conditional part and an action part. The rules should all be written in XML solely, so that they can be changed independently from the program. I wanted a rule to be able to identify three different things: Either should it detect regular expressions, or should it detect words matching to certain morphosyntactical information or words having a certain lemma.

Together with Jonas Thulin we built a first prototype that was able to encode some of the stage indicators in a text. Our first prototype, although we had troubles making it run and it let open many questions, already incorporated the core idea of the program as it still is today.

The program was developed further by Lisa Persson and Emil Persson after my departure from Lund to Switzerland. When I joined the project to write my master thesis about it, the program had grown significantly in size, "feel and look" and functionality.

We were confronted with the problem of erroneous input. Misplaced or forgotten accents were very common language mistakes, especially if dealt with language learners' texts. In

these cases, *Direkt Profil* v1.4 still was not able to detect anything (except if searching for regular expressions), because misspelled words were simply treated as unknown words that as a result could not be found in the dictionary.
As I wanted to develop a functionality to identify such words and process them appropriately, I found it impossible to simply add the modules I had written. The program's core analysis engine which steered the parser's algorithmic behavior had grown to a point where it could not be changed easily anymore. Also, such a module needed the possibility for the analysis engine to jump back several tokens and reprocess them again with different parameters. This functionality had not been foreseen earlier and thus did not exist. So, my programming partner Emil Persson and I decided to first reengineer the most central classes of the program and decouple them as far as possible, following a better OOP-approach. The result was *Direkt Profil* v1.5's new class diagram presented in chapter 6.1. This needed quite some time, but clearly improved the software's internal structure's quality. Afterwards, it was easy to add the accent search.

Another feature I implemented in v1.5 is the stem search for past participles. Here again, we regularly were faced the problem that a language learner tried for instance to express a *passé composé* but failed to derive the necessary *participe passé* correctly (like writing **prendu* instead of *pris*). In this case, her attempts would be neglected by the software, although in the text clearly something should be counted. Mostly, the erroneous *participe passé*s took the form of an existing stem of the chosen verb in combination with a *participe passé*'s ending (*-é, -ée, -és, -ées, -i, -ie,* etc.). For an implementation of the stem search, I had to make changes to the dictionary. I added a second instance of the dictionary ordered by the word stems. For this purpose, first the stems had to be extracted automatically by the program. In a second step, around 2000 forms of irregular verbs were added manually. This instance was then used in a further step together with the corresponding stem search modules.

For v1.5, I also added lots of new rules to the program. Whereas in v1.4 around two dozen different rules existed in the rules file for stage indicators, in v1.5 we nearly doubled the number of stage indicator rules applicable to a text, resulting in a bigger variety of stage indicators to be detected. For instance I inserted rules to detect the *conditionnel*, the *futur simple*, the *imparfait* and the *présent* for all verbs (and not only for some chosen ones). And this is not yet the end: As can be seen in the rule tree in chapter 5.7, when a verb with an erroneous accent is detected in rule 011, the analysis engine stops a further analysis at this

point. Since we were not sure about the analysis quality, the accent search would deliver, and we did not want to carry off errors from an earlier analysis, we decided to implement the functionality and stopping the analysis if such a case occurs. As the accent search seems to deliver good and reliable results, we plan to extend its usage now and insert new descendant rules to rule 011.

8 Points to be remembered and conclusions

Direkt Profil is a "semi-application CALL software". Its goal is not to explicitly teach language learners grammar or vocabulary, but it should support the learner on her way through the language acquisition process. *Direkt Profil* must thus not be seen as an alternative to existing (traditional) CALL programs or text and exercise books, but as a complement to them. Also for linguistic research, the program can be useful. *Direkt Profil* deals with written French language.
Direkt Profil is a program still under development. Many important program features, like processing noun group stage indicators or computing the most probable stage, are still missing. But as the project is planned to go on until the end of 2007, it will go through several changes in look and behavior.

The software is based on linguistic theories of second language acquisition of French (see [Bartning & Schlyter 2004]) and on how to profile second language acquisition (see [Hyltenstam & Pienemann 1985] and [Clahsen 1986]).

What has been shown in this paper is the core idea of how a set of binary condition/action-rules can be used to generate text profiles. A rule is applied to the text by establishing a frame of tokens. Inside the frame, every token is matched against the rule's search criteria until at least one suitable token is found, or no token inside the frame matched. In this case, the next rule is applied to the text in the same way. If a terminal node in the rule tree is encountered, after its application to the text, the actions of all rules are triggered in a backward manner, where the most recently applied rule triggers its action first.
By applying one rule after the other to a tokenized text, the program follows a path inside the rule tree. Doing so, it can extract and annotate stage indicators and multi word expressions. To annotate the tokenized words in the text with their lemma, part-of-speech and further information, a French dictionary is used. The precedence order of the rules plays an important role, which can possibly lead to wrong results. *Direkt Profil* uses a partial parser to accomplish its tasks.

As one can see, the principles behind *Direkt Profil* are indeed highly independent of whether dealing with French or other European languages.

9 Bibliography

[Bartning & Schlyter 2004] Bartning, Inge & Schlyter, Suzanne (2004); "Itinéraires acquisitionnels et stades de développement en français L2"; Journal of French Language Studies, 14:3; p.281-299; Cambridge University Press; The paper is also available on http://journals.cambridge.org/bin/bladerunner?30REQEVENT=&REQAUTH=0&500000REQSUB=&REQSTR1=S0959269504001802

[Borin 2002] Borin, Lars (2002); "Where will the Standards for Intelligent Computer-Assisted Language Learning Come from?"; Workshop Proceedings, International Standards of Terminology and Language Resources Management; Las Palmas, Spain; p61 – 68

[Chapelle 2001] Chapelle, Carol A. (2001); "Computer Applications in Second Language Acquisition – Foundations for teaching, testing and research"; Cambridge Applied Linguistics; Cambridge; p.27 – 43

[Chomsky 1985] Chomsky, Noam (1985); "Aspects of the Theory of Syntax"; The MIT Press; Cambridge, MA; as cited in [Cornu 1997]

[CEF] *CEF – Common European Framework of Reference for Languages*; Council of Europe, Strasbourg; available from Cambridge University Press; (ISBN : HB 0521803136 - PB 0521005310)

[Clahsen 1985] Clahsen, Harald (1985); "Profiling second language development: A procedure for assessing L2 proficiency"; in [Hyltenstam & Pienemann, 1985] (eds); p.283 – 322

[Clahsen 1986] Clahsen, Harald (1986); „Die Profilanalyse. Ein linguistisches Verfahren für die Sprachdiagnose im Vorschulalter."; Wissenschaftsverlag Spiess; Berlin: Edition Marhold;

[Clément et al.] Clément, Lionel; Sagot, Benoît; Lang, Bernard; "Morphology based automatic acquisition of large-coverage lexical"; INRIA – Institut National de Recherche en Informatique et en Automatique Domaine de Rocquencourt, Le Chesnay, France.

[Clergerie 2004] de la Clergerie, Éric (2004); "Appel à la contribution sur morpho-syntaxe"; ATOLL – INRIA Rocquencourt; http://atoll.inria.fr; 07/04/2004; Réunion RNIL, AFNOR.

[Cornu 1997] Cornu, Etienne (1997); "Correction automatique des erreurs morphologiques et syntaxiques produites à l'écrit en langue seconde"; Imprimerie de l'Evole SA, Neuchâtel

[Duquette & Laurier 2000] Duquette, Lise & Laurier, Michel (2000); "Apprendre une langue dans un environnement multimédia"; Les Éditions Logiques; Québec, Canada

[Gamon et al. 1997] Gamon, Michael (1997); Lozano, Carmen; Pinkham, Jessie; Reutter, Thomas; "Practical Experience with Grammar Sharing in Multilingual NLP"; Microsoft Research, Advanced Technology Division, Microsoft Corporation, Redmond, USA;

03/06/1997; (as presented at the “From Research Commercial Applications” workshop of the ACL/EACL 97 conference in Madrid, Spain).

[Gamon et al. 1997] Gamon, Michael & Reutter, Thomas (1997); “The Analysis of German Separable Prefix Verbs in the Microsoft Natural Language Processing System”; Microsoft Research, Microsoft Corporation, Redmond, USA; 25/09/1997.

[Gendner & Vilnat 2004] Gendner, Véronique & Vilnat, Anne (2004); “Les annotations syntaxiques de référence PEAS v1.6“; EVALDA-TECHNOLANGUE; 30/05/2004; Link: http://www.limsi.fr/Recherche/CORVAL/easy/

[Granfeldt 2003] Granfeldt, Jonas (2003); “L’acquisition des catégories fonctionnelles, Étude comparative du développement du DP français chez des enfants et des apprenants adultes”; Department of Romance Languages, Lund University, Lund, Sweden

[Granfeldt & Schlyter 2004] Granfeldt, J. et Schlyter, S.(2004); ”Cliticisation in the acquisition of French as L1 and L2.” From: P. Prévost et J. Paradis (dir.), *Acquisition of French: Focus on Functional Categories.* Amsterdam: Benjamins.

[Granfeldt et. al 2005] Granfeldt, Jonas; Nugues, Pierre; Persson, Emil; Kostadinov; Fabian; Schlyter Suzanne (2005); „Direkt Profil“; in the conference paper of TALN 2005 (http://taln.limsi.fr/site/)

[Granger et al. 2001] Granger, Sylviane; Vandeventer, Anne; Hamel, Marie-Josée (2001); „Analyse de corpus d’apprenants pour l’ELAO basé sur le TAL“; TAL 42(2); p.609 – 621; as referred to in [Granfeldt et al. 2005]

[Haas & Tanc 1987] Haas, Joachim & Tanc, Danielle (1987); “Französische Grammatik”; Verlag Moritz Diesterweg GmbH & Co.; Frankfurt am Main

[Heidorn] Heidorn, George E.; „Intelligent Writing Assistance“; Microsoft Research, Redmond, Washington (Reprinted from Robert Dale, Macquarie University, Sidney, Australia; Hermann Moisl, University of Newcastle Newcastle-upon-Tyne, England; Harold Somers, UMIST, Manchester, England; „Handbook of natural language prossecing”); © 2000 by Marcel Dekker, Inc.

[Hyltenstam & Pienemann 1985] Hyltenstam, Kenneth & Pienemann, Manfred (1985); “Modelling and Assessing Second Language Acquisition”; Multilingual Matters Ltd; Clevedon, England;

[Håkansson] Håkansson, Gisela; “Rapid Profile – en snabbdiagnos av grammatisk nivå i inlärarspråk”; Institutionen för lingvistik, Lunds Universitet

[Jensen et. al 1993] Jensen, Karen; Heidorn, George E.; Richardson, Stephen D. (1993); „Natural Language Processing: The PLNLP Approach”; Kluwer Academic Publishers; Boston/Dordrecht/London

[Jurafsky & Martin 2000] Jurafsky, Daniel & Martin, James H. (2000); “Speech and Language Processing – An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition”; Prentice-Hall Inc.; Pearson Higher Education, Upper Saddle River, New Jersey

[Kostadinov & Thulin 2003] Kostadinov, Fabian & Thulin, Jonas (2003); “A text critiquing system for Swedish-speaking students of French”; Lund Institute of Technology, Sweden; available on http://www.cs.lth.se/Education/Courses/EDA171/Reports/2003/jonas_fabian.pdf

[Knutsson 2003] Knutsson, Ola; Bigert, Johnny; Kann, Viggo (2003); “A Robust Shallow Parser for Swedish”; NODALIDA; Reykjavik, Island; as referred to in [Granfeldt et al. 2005]

[Laenzlinge & Wehrli 1991] Laenzlinge, C; Wehrli, E. (1991); “Fips: Un analyseur interactif pour le français”; T.A Informations 2; p.35 – 49

[Lightbrown & Spada 1999] Lightbrown, P; Spada, N. (1999); “How languages are learnt”; Oxford University Press;

[Lindstedt 1998] Lindstedt, Juha P. (1998); “Computer-assisted language learning analysis”; University of Helsinki, Department of Education; Helsinki; p.13 – 15, 161, 218

[Loyd 2000] Loyd, Les (2000); “Teaching with Technology: Rethinking Tradition”; Information Today, Inc.; Medford, New Jersey

[Nugues 2004] Nugues, Pierre (2004); “An Introduction to Language Processing with Perl and Prolog”; Unfinished working manuscript; Tryckeriet i E-Huset; Lund

[Persson 2004] Persson, Lisa (2004); “Direkt Profil, Ett verktyg för morfologisk analys at skriven inlärarfranska“; Department of Computer Science, Faculty of Science, Lund University, Sweden; (English Title: “A tool for morphological analysis of written learner French”).

[Pienemann 1998] Pienemann, Manfred (1998); “Language Processing and Second Language Development – Processability Theory”; John Benjamins Publishing Company Amsterdam/Philadelphia

[Pinkham 2004] Pinkham, Jessie (2004); “Grammar Sharing Between English and French”; Microsoft Research, Advanced Technology Division, Microsoft Corporation, Redmond, USA; 09/2004.

[Sealander & Tholey 2002] Selander, Staffan & Tholey, Marita (2002); Lorentzen, Svein; “New educational media and textbooks, The 2nd IARTEM Volume”; Stockholm Institute of Education Press, Sweden; Stockholm; p. 57 - 59

[Ressources Normalisés 2003] Ressources Normalisées en Ingénerie de la Langue (2003); “Contribution au projet ISO/AWI 24611 «Language resource management – Morpho syntactic annotation framework»; CN RNIL N6; Association Française de Normalisation, Saint-Denis La Plaine Cedex, France; http://www.afnor.fr; 25/11/2003

[Schlyter 2003] Schlyter, Suzanne (2003); “Stades de développement en français L2 – Exemple d’apprenants suédophones, guides en non-guidés, du “Corpus Lund””; Department of Romance Languages, Lund University, Sweden; 18/12/2003

[Taylor 1989] Taylor, M.B, and Perez, L.M. (1989); “Something to Do on Monday; La Jolla, CA; Athelstan; as cited in [Sealander & Tholey, 2002]

[Taylor 1980] Taylor, R (1980); “The computer in the school: Tutor, Tool, Tutee”; New York: Teachers College Press; as cited in [Sealander & Tholey, 2002]

[Underwood 1984] Underwood, J. (1984); “Linguistics, Computers, and the Language Teacher L A Communicative Approach”; Rowley, MA: Newbury House; as cited in [Sealander & Tholey, 2002]

[Vandeventer 2004] Vandeventer Faltin, Anne (2003); “Syntactic Error Detection in the context of Computer Assisted Language Learning”; Faculté des Lettres de l’Université de Genève; Switzerland;

[Vandeventer & L’Hair 2003] Vandeventer Faltin, Anne & L’Hair, Sébastien (12/2003); “Diagnostic d’erreurs dans le projet FreeText”; Université de Genève; Switzerland;

[Vergne 2004] Vergne, Jacques (2004); “Découverte locale des mots vides dans des corpus bruts de langues inconnues, sans aucune ressource”; 7es Journées internationals d’Analyse statistique des Données Textuelles (JADT); Université de Caen

[Warschauer 1996] Warschauer, M. (1996); “Computer-Assisted Language Learning: An Introduction”; as cited in [Sealander & Tholey, 2002]

[Werner 1999] Werner, Laura (1999); “Efficient text searching in Java: Finding the right string in any language”; IBM; 01/04/1999;
Link: http://oss.software.ibm.com/icu/docs/papers/efficient_text_searching_in_java.html

9.1 WWW, Homepages, Links:

Apache Tomcat Server: http://jakarta.apache.org/tomcat/index.html

Apache Ant: http://ant.apache.org

Association des Bibliophiles Universels: http://abu.cnam.fr

Java (Software) Development Kit and Java Runtime Environment: http://java.sun.com

Direkt Profil, project homepage: http://www.rom.lu.se/durs/usif.htm

Dialang, project homepag: http://www.dialang.org

FreeText, project homepage: http://www.latl.unige.ch/freetext/

Granska/CrossCheck, project homepage:
http://www.nada.kth.se/theory/projects/xcheck/index-en.html

Rapid Profile, project homepage: http://www-fakkw.upb.de/institute/Anglistik_Amerikanistik/Personal/Kessler/Rapid_Profile/

Unicode organization: http://www.unicode.org

10 Appendix A: DTDs

The DTD of the rule file

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT rules           (process+)>

<!ELEMENT process         (delimiter_tokenize_rule?,
                          word_tokenize_rule?,
                          non_word_tokenize_rule?,
                          tree+)>
<!ATTLIST process
          id              ID            #REQUIRED>

<!ELEMENT delimiter_tokenize_rule       (regex, regex)>
<!ELEMENT word_tokenize_rule            (regex)>
<!ELEMENT non_word_tokenize_rule        (regex)>

<!ELEMENT tree            (description,
                          track_result?,
                          counters,
                          rule+)>
<!ATTLIST tree
          id              ID            #REQUIRED
          initrule        CDATA         #REQUIRED>

<!ELEMENT counters        (counter*)>
<!ELEMENT counter         (description,
                          group_by?,
                          format?,
                          grouper?)>
<!ATTLIST counter
          id              ID            #REQUIRED
          name            CDATA         #REQUIRED
          order           CDATA         #IMPLIED
          sub_order       CDATA         #IMPLIED
          view_res        (yes|no)      #REQUIRED
          exp             (yes|no)      #REQUIRED
          level           CDATA         #REQUIRED>

<!ELEMENT group_by        (#PCDATA)>

<!ELEMENT format          (color?, style?)>

<!ELEMENT color           EMPTY>
<!ATTLIST color
          fg              CDATA         #IMPLIED
          bg              CDATA         #REQUIRED>

<!ELEMENT style           EMPTY>
```

```
<!ATTLIST style
          font_style      CDATA         #IMPLIED
          font_weight     CDATA         #IMPLIED
          decoration      CDATA         #IMPLIED>
<!ELEMENT grouper         (description?)>

<!ELEMENT rule            (description,
                           example?,
                           search,
                           action)>
<!ATTLIST rule
          id              ID            #REQUIRED>
<!ELEMENT description     (sv,en,fr)>

<!ELEMENT sv              (#PCDATA)>

<!ELEMENT en              (#PCDATA)>

<!ELEMENT fr              (#PCDATA)>

<!ELEMENT example         (ex_match?, ex_nomatch?)>

<!ELEMENT ex_match        (#PCDATA)>

<!ELEMENT ex_nomatch      (#PCDATA)>

<!ELEMENT search
          (track_result?,((regular_expr,lemma?,inflection?)|(l
emma,inflection?)|(inflection)|(agree)|(stem)))>
<!ATTLIST search
          framesize       CDATA         #IMPLIED
          mode            (recall|normal)   #IMPLIED
          type            (mwe)         #IMPLIED>

<!ELEMENT track_result    (tree_result+)>

<!ELEMENT tree_result     (#PCDATA)>
<!ATTLIST tree_result
          mode            (use|discard) #REQUIRED>

<!ELEMENT regular_expr    (regex+)>

<!ELEMENT regex           (#PCDATA)>

<!ELEMENT lemma           (#PCDATA)>

<!ELEMENT stem            EMPTY>
<!ATTLIST stem
          category        (noun | verb | adjective | pronoun |
int_pronoun | determiner | adverb | preposition | conjunction
| numeral | interjection | abbreviation | residual) #REQUIRED>
```

```
<!ELEMENT inflection      (gender?, number?, person?, tense?,
mode?, nominative?)>
<!ATTLIST inflection
          category        (noun | verb | adjective | pronoun |
int_pronoun | determiner | adverb | preposition | conjunction
| numeral | interjection | abbreviation | residual) #REQUIRED
          accent_search   (on | off )   #IMPLIED>


<!ELEMENT person          EMPTY>
<!ELEMENT number          EMPTY>
<!ELEMENT gender          EMPTY>
<!ELEMENT tense           EMPTY>
<!ELEMENT mode            EMPTY>
<!ELEMENT nominative      EMPTY>

<!ATTLIST person
          value           (1 | 2 | 3)   #REQUIRED>
<!ATTLIST number
          value           (sg | pl)     #REQUIRED>
<!ATTLIST gender
          value           (feminine | masculine) #REQUIRED>
<!ATTLIST tense
          value           (future | present | imperfect |
past) #REQUIRED>

<!ATTLIST mode
          value           (indicative | conditional |
infinitive | participle | subjunctive)  #REQUIRED>
<!ATTLIST nominative
          value           (yes | no)    #REQUIRED>
<!ELEMENT agree           (criteriums, category+)>
<!ELEMENT criteriums      (criterium+)>
<!ELEMENT criterium       EMPTY>

<!ATTLIST criterium
          value           (gender | number | person)
#REQUIRED>

<!ELEMENT category        EMPTY>

<!ATTLIST category
          value           (noun | verb | adjective | pronoun |
int_pronoun | determiner | adverb | preposition | conjunction
| numeral | interjection | abbreviation | residual) #REQUIRED>


<!ELEMENT action          (match, nomatch)>
<!ELEMENT match           (incrcounter*, decrcounter*)>
<!ATTLIST match
          nextrule        IDREF         #REQUIRED
```

```
          dotagging       CDATA         #IMPLIED
          register_tree_result CDATA    #IMPLIED>


<!ELEMENT nomatch         (incrcounter*, decrcounter*)>
<!ATTLIST nomatch
          nextrule        IDREF         #REQUIRED
          dotagging       CDATA         #IMPLIED
          register_tree_result CDATA    #IMPLIED>

<!ELEMENT incrcounter EMPTY>
<!ATTLIST incrcounter
          value           IDREF         #IMPLIED>
```

The DTD of the dictionary

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE dict [
<!ELEMENT dict            (wordform+)>
<!ELEMENT wordform        stem*,feature*)>
<!ATTLIST wordform
          entry           CDATA         #REQUIRED>
<!ATTLIST wordform
          lemma           CDATA         #REQUIRED>
<!ATTLIST wordform
          pos             (ver | nom | det | pre | pro | adj |
adv | con | int | abr | qpro | ono)     #REQUIRED>
<!ELEMENT stem            (#PCDATA)>
<!ELEMENT feature         (#PCDATA)>
]>
```

The DTD of the CEFLE-corpus

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE CORPUS [<!ELEMENT CORPUS (SAMPLE+)>
<!ELEMENT SAMPLE          (TEXT,INFO)>
<!ATTLIST SAMPLE
          SUBJECT_ID      CDATA         #REQUIRED>
<!ELEMENT TEXT            (#PCDATA)>
<!ELEMENT INFO            EMPTY>
<!ATTLIST INFO
          TASK_NAME       (VOYAGE_ITALIE | SOUVENIR_VOYAGE |
MA_FAMILLE | HOMME_ILE)   #REQUIRED>
<!ATTLIST INFO
          GROUP_SUBJECT   (PILOT|MAIN|CONTROL)   #REQUIRED>
<!ATTLIST INFO
          SUBJECT_LEVEL   (1|2|3|4|5|CONTROL)    #REQUIRED>
<!ATTLIST INFO
          SOURCE_SCHOOL   (PETRI | BORGAR | KATTE | POLHEM |
SPYKEN | NEVERS | PARIS)  #REQUIRED>
]>
```

11 Appendix B: Installation of Direkt Profil

Prerequisites:

- Apache **Tomcat 4.1** server or higher. The Apache Tomcat server is freely available at http://jakarta.apache.org/tomcat/index.html.
- As described in the Apache Tomcat installation files, the Tomcat server needs the **Java Runtime Environmen**t (JRE) as well as the **Java (Software) Development Kit** (called JDK or Java SDK). Both, JRE and JDK should be at least version 1.4 or higher! Be aware that the Tomcat server needs a preinstalled JDK also to work, the JRE alone is not sufficient. Both can be freely downloaded from http://java.sun.com.
- ***Direkt Profil* source files** including the dictionary files
- At least **140 MB main memory** (RAM)
- (If compilation is necessary, we recommend Apache Ant 1.6 or higher. A build file is provided with *Direkt Profil*. Ant is also freely available and can be downloaded from http://ant.apache.org/. If compiling, make sure that you have at least Java Development Kit version 1.4 or higher.)

The installation procedure:

1. First, if not already installed on your machine, install the Java Runtime Environment (JRE) and the Java Development Kit (JDK). For help, see the description of the installation guide provided with both of them. Make sure that you have set the `PATH` variable correctly so that java.exe/javac.exe (for Windows) respectively java/javac (for Linux/Unix) are both callable from everywhere. In the end, the `PATH` variable (on Windows) might look somehow like:
   `PATH=.;c:\program files\Java\J2RE_1.4.2\bin;c:\program files\Java\J2SDK_1.4.2\bin;…many other paths…`
2. Second, install Apache Tomcat. Tomcat needs 2 environment variables to be set to function properly. The `JAVA_HOME` variable must point to where the JDK is installed. (In our example, this would be thus something like `JAVA_HOME=c:\program files\Java\J2SDK_1.4.2`.) Furthermore, the `CATALINA_HOME` variable must be set to where the Tomcat server resides. (This might result in something like `CATALINA_HOME=c:\program files\Apache\Tomcat 4.1`.)
3. Make sure that after the installation procedure, no instance of Tomcat is running. If it is, shut the instance down. We first need to configure the server for our purposes. Be

aware that on some systems after the installation Tomcat might be started automatically when the operation system was restarted.

4. If you have not compiled the source code of *Direkt Profil* yet, do it now. You might want to use Ant for this purpose. A build file called `build.xml` can be found in the folder called `/work`. To compile the program, make sure that the settings in *Direkt Profil*'s property files are set correctly. For this purpose, open a file in the folder `/work/props/build.properties` with a common text editor. Set the `classpath` variable to the directory, where you installed Tomcat before. **Be aware that you have to use slashes '/' instead of backslashes '\' for separating directories – also if operating on Windows! This counts for every entry denoting a path in this file!** The entry might in the end look somehow like `classpath=c:/program files/apache/Tomcat 4.1`.
   You can set the distribution variable to where you want your compiled *Direkt Profil* files to be placed. In the case of uncertainty, do not change this entry. It might then look like: `distribution=../distribution`

   Once compiled, you will have a directory called `/distribution/profil` with a similar structure to the `/work` directory. Explanation of the directory structure:
   The subfolder `/distribution/profil/dict/a_z` contains the dictionary files.
   The subfolder `/distribution/profil/rules` contains the rules file. The corresponding DTD can be found in the folder `/distribution/profil/dtd`.
   The subfolder `/distribution/profil/props` contains some properties files.
   The subfolder `/distribution/profil/logs` contain log files. Usually, logging is turned off. It can be turned on only in the source code by setting a class's DEBUG variable to true and recompiling the program.
   The subfolder `/distribution/WEB-INF` contains the source code itself.
5. We need to change the configuration files of Tomcat. In the directory, where Tomcat is installed, there is a subdirectory called `/conf`. There is a file called `server.xml` which must be changed. Open the file with a text editor. Search for an entry starting with a tag `<Context…` . The easiest way is to copy+paste an existing example entry, change it accordingly and uncomment the other entries. Set the `<Context…>` tag's attribute `path` to a name you want to reach your instance of *Direkt Profil* later, for

instance `path="/direktprofil"`. This will be part of the URL to be typed in the browser, for instance http://www.foo.com:8080/direktprofil or http://localhost:8080/direktprofil. **Do not forget to put a slash '/' in front of the chosen name.** The docBase attribute must point to the directory, where the file `logon.jsp` is located, for example `doctBase="c:/Program Files/DirektProfil_v1.5/distribution/profil"`.

Be aware that everything is treated case sensitive! Again, use slashes '/' instead of backslashes '\' for separation of directories in paths also for Windows.

6. Finally, we must make sure that the Java Virtual Machine of the Tomcat instance has enough of memory to operate. The common standard for a Java Virtual Machine is usually set to 80 MB what is not enough for our purposes (a typical mistake in this case is that the program loads the dictionary files only from A.xml till I.xml but does not process further). We need at least 140 MB for *Direkt Profil* v1.5.1. The most Java Virtual Machines can be given a parameter "–Xmx140m" at startup time specifying to use more than the foreseen amount of memory, in this case 140 MB instead. This parameter can be specified in startup.bat (in Windows) or startup.sh (in Linux/Unix) by setting `CATALINA_OPTS="-Xmx140m"`.

   Be aware on <u>how</u> you start your instance of Tomcat! Make sure that the option is given to the Java Virtual Machine. We experienced it that some users installed Tomcat locally on their Windows machine and always started it through the Start → Programs → Apache Tomcat → Start Tomcat menus. However, this procedure started a Java Virtual Machine without ever giving the necessary parameter to it – so it is better to open a shell and start Tomcat through calling `startup.bat/startup.bin` directly from it.
7. Tomcat is listening by default on port 8080. If Tomcat is running locally on your machine, open a Webbrowser and type the URL http://localhost:8080/direktprofil. You should see the login screen now.