Universität Tübingen
Seminar für Sprachwissenschaft
Wilhelmstraße 19
72074 Tübingen

Eberhard Karls
Universität
Tübingen

M.A. Thesis in Computational Linguistics

A Flexible Annotation-Based Architecture for Intelligent Language Tutoring Systems

Ramon Ziai
rziai@sfs.uni-tuebingen.de

April 14, 2009

First Examiner and Supervisor:	Prof. Dr. W. Detmar Meurers
Second Examiner:	Dale Gerdemann, PhD

Abstract

This thesis presents a general architecture for Intelligent Language Tutoring Systems. The architecture makes use of annotation-based processing to encode different linguistic information at deep and shallow levels in a flexible manner and associate it with the learner input string. At the same time, it leaves the original input untouched, which is different from traditional pipeline approaches used in other Intelligent Computer-Assisted Language Learning systems. We show that this is necessary in order for feedback to be both error-specific and useful to the learner.

Furthermore, the architecture incorporates a general mechanism for the use of activity models in guiding processing and feedback. The approach uses error types as a means of formulating pedagogical goals of individual activities. These error types are then mapped to analysis requirements that can be used to adapt processing of the learner input to the activity. More importantly, error types are used to influence the feedback strategy towards the errors that a particular activity centers on. We also demonstrate a similar use of error counts in the learner model to guide feedback.

We have evaluated the system on the basis of example scenarios. They show that we are indeed able to give more useful feedback than the baseline system TAGARELA (Amaral, 2007) in a number of cases. This is the result of both annotation-based processing and integration of activity and learner model into the feedback process.

Zusammenfassung

In der vorliegenden Arbeit wird eine generelle Architektur für intelligente Sprachtutorensysteme vorgestellt. Die Architektur bedient sich einer annotationsbasierten Verarbeitungsmethode, die tiefe und flache linguistische Information flexibel integrieren und mit der Lernereingabe verknüpfen kann. Dabei bleibt die ursprüngliche Eingabe unangetastet, was im Gegensatz zu traditionellen Pipeline-Ansätzen steht, wie sie in anderen intelligenten Sprachlernsystemen verwendet werden. Es wird gezeigt, dass dies notwendig ist, um fehlerspezifische und nützliche Rückmeldung für den Lerner zu generieren.

Weiterhin beinhaltet die Architektur einen generellen Mechanismus zur Integration der Aufgabenstellung in die Verarbeitung und den Prozess der Rückmeldung. Der Ansatz verwendet Fehlertypen, um pädagogische Ziele von Aufgaben zu formulieren. Diese Fehlertypen werden dann in Anforderungen für die Analyse der Lernereingabe abgebildet, um die Verarbeitung aufgabenspezifisch anpassen zu können. Außerdem werden die Fehlertypen zur Adaption der Rückmeldung an die pädagogischen Ziele der Aufgabe verwendet, was einen wichtigen Unterschied zu anderen Systemen darstellt. Zudem wird eine Möglichkeit demonstriert, lernerspezifische Rückmeldung zu geben.

Das System wurde auf Basis von Beispielszenarien evaluiert. Diese zeigen, dass das System tatsächlich in den gezeigten Fällen in der Lage ist, nützlichere Rückmeldungen als das Ausgangssystem TAGARELA (Amaral, 2007) zu liefern. Letzteres stellt ein Resultat sowohl der annotationsbasierten Verarbeitung als auch der Integration von Information über Lerner und Aufgabe in den Rückmeldungsvorgang dar.

Contents

List of Tables

List of Figures

1 Introduction

Learning a new language is nowadays essential to many vocational fields, especially academics. The use of foreign languages as a means of cross-cultural communication is important for a large number of people around the globe. For instance, this becomes apparent in the phenomenon of English as a lingua franca between many countries.

However, teaching resources are not always sufficient. For example, trained instructors may not be available or if they are, they may not always be able to devote enough time to helping individual students with their difficulties. Specifically, instructors are often not available to oversee students doing homework and cannot give them immediate feedback on their produced language. Moreover, students are usually taught in groups, where it is difficult to let every student learn at his own speed. Furthermore, students may prefer working with a computer in cases where social factors make human-human interaction difficult.

Intelligent Language Tutoring Systems (ILTSs) have emerged in the field of Intelligent Computer-Assisted Language Learning (ICALL) over the past two decades as a computer-based supplement to foreign language teaching. They pursue the idea of computerized language exercises with on-the-spot feedback on learner input. ILTSs employ Natural Language Processing (NLP) technology in order to identify linguistic properties in learner language, a necessary prerequisite for error-specific feedback. A number of ILTSs have been proposed, from highly ambitious conversation machines (cf. e.g. DeSmedt (1995)) to more moderate workbook-like approaches (cf. e.g. Heift and Nicholson (2001); Nagata (2002)).

However, current systems use inflexible processing approaches: fixed pipelines of NLP modules, where the output of one module is fed into the next module, process learner input. Consequently, they are only able to deal with a limited set of activities where the pipeline is adequate for the task, as outlined in Amaral and Meurers (2007).

In addition, most systems do not take the specifics of activities and learners into account when giving feedback. Yet this is necessary because different activities pursue different pedagogical goals. A reading comprehension activity is designed to teach learners to reproduce content whereas a vocabulary activity focuses on particular grammatical constructions. Furthermore, not all learners make the same errors, so feedback strategies of ILTSs should take characteristics of learners into account.

The TAGARELA system (Amaral, 2007), an intelligent workbook for Portuguese, was an important step in the direction of a pedagogically sound ILTS. It features a well-designed user interface and uses a combination of deep and shallow NLP processing techniques to analyze learner input. Yet it is still designed for a predefined set of activity types and furthermore does not vary its feedback strategy according to specific activities or learners.

A number of ideas have since been developed to change these shortcomings. Amaral and Meurers (2007) propose the use of activity models to guide processing and feedback. Moreover, they conceptualize an extended view of modeling learner's competence in Amaral and Meurers (2008). Finally, they argue for the use of annotation-based processing for ICALL applications in Amaral and Meurers (2009). However, no concrete proposal for these ideas exists.

This thesis takes these ideas seriously and presents an attempt at putting some of them into practice. To that end, we build upon the foundations of the TAGARELA system and generalize it in two ways. First, we introduce annotation-based processing as a vital component of ICALL systems that is able to encode deep linguistic properties and associate them with surface forms. Second, we integrate the activity model and learner model to a larger degree than what is currently realized. Specifically, we show how activity models can be used to guide the analysis process, and how both activity model and learner model can be used to flexibly alter the feedback strategy.

The paper is organized as follows. Chapter 2 introduces important preliminaries that are used in the following elaborations. Chapter 3 explains in more detail the research context and the issues we try to tackle. Then, chapter 4 explains the central components in our architecture and their interaction. Chapter 5 delves into the implementation of the system and also presents the various analysis modules. In chapter 6, we illustrate the benefits of our architecture on the basis of concrete examples before we give our conclusions in chapter 7.

2 Preliminaries

This chapter will introduce the necessary notions for the rest of the paper. Where appropriate, references are given to more thorough treatments of the respective topics.

2.1 CALL

Computer-Assisted Language Learning (CALL) is a wide field that spans over a number of other areas. Levy (1997) defines CALL as "the search for and study of applications of the computer in language teaching and learning" (Levy, 1997, p. 1). Naturally, this will include areas that are not related to NLP. Computers are generally used for two main goals in standard CALL: Computer-mediated Communication (CMC) and representation of language.

CMC is the sub-field of CALL that uses computer networks as a means of communication between the participants of the learning process. The latter may be learners or tutors. Concrete examples of CMC range from the application of conventional communication tools for language learning (e.g. e-mail) to the creation of specialized new communication platforms, such as shared whiteboard applications[1].

Shared whiteboard applications pursue the idea of a virtual classroom, placing each participant of a course into a chat room and giving them means of presenting and discussing learning material. For our purposes, these applications are not relevant as language is not represented or stored, let alone analyzed.

Learning Management Systems (LMSs) also provide a platform for courses but offer more language-oriented features in the sense that language is represented and stored for various purposes. Representation is understood here as the task of organizing and structuring language electronically to make it accessible to learners. A prominent LMS is Moodle[2]. Besides organizing learning material such as course

[1]See `http://www.open-steam.org` for an example.

[2]`http://moodle.org`

slides, Moodle lets instructors create tests for students. Answer types for test questions typically do not allow free sentential input, instead relying on multiple choice or fill-in-the-blanks to restrain input variation, a point that comes up in comparison to ICALL systems (see section 2.1.1).

These kinds of tests can be generalized to so-called frame-based CALL systems. In such systems, student input is compared to a set, or frame, of predefined correct and incorrect answers. Depending on what the student input matches, the system then gives positive or negative feedback. However, because no analysis of student input is done, the system can only distinguish between “wrong” and “correct”. Frame-based systems can be classified into three different groups: linear, branching and generative (Bowerman, 1990). These groups differ in how they proceed through the set of available questions.

Linear systems proceed with the next question regardless of whether the student answered the previous one correctly or not, thus the systems are non-adaptive. In branching systems, questions are organized in different layers. If a student answers a question correctly, the system stays on the same layer. If not, the system branches to a lower question layer. If the question on the lower layer is then answered correctly, the systems returns to the higher layer. Otherwise, the next lower layer is entered and so on. Finally, generative systems use a generation algorithm to create questions and reduce the predictability of the system.

Another substantial body of research in CALL is concerned with the use of corpora for language learning (Nerbonne, 2003). Compared to standard school textbooks, corpora offer the advantage of more authentic text. Furthermore, corpora are getting bigger and bigger while still being searchable efficiently through concordancers, for instance. Concordancers are frequently the tool of choice when students are sent to explore authentic language on their own. A recent example of this is described in Widmann et al. (2008). However, corpora seem only of use to advanced learners whereas beginners are overwhelmed with the wealth of information presented to them (Nerbonne, 2003, p. 681).

What is important to keep in mind with all these approaches is that knowledge of language is always made fully explicit and extensional. In frame-based systems, target representations are fully specified and compared on a character-by-character basis. In some cases, not even superfluous spaces are acceptable and treated as an error. Concordancing tools can search for tokens and lemmas at the most, but no further NLP is typically employed. This shallow approach differs fundamentally from the one pursued in ICALL, which we introduce in the next section.

2.1.1 ICALL

ICALL is a relatively young field and generally understood to be the sub-field of CALL to which NLP is applied in order to encode linguistic knowledge into language learning programs. ICALL is interdisciplinary, involving linguistics, computer science and cognition. The contribution of the latter field centers around learner modeling, which is discussed in section 2.1.2.

The general benefit of ICALL in comparison to traditional CALL systems is that as ICALL systems "understand" language to a certain degree, they can support an actual analysis of learner language. This in turn makes it possible to give more detailed feedback than the binary "right" and "wrong" of traditional CALL systems. Recent research (e.g. Nagata (1995, 1996)) argues that such feedback is necessary in order to help the learner achieve the learning goal of a particular activity. By activity, we mean here a concrete setting that involves a learning context and a task. An example would be a reading comprehension question where the learning context is the text to be read and the task is to answer a set of questions using information from the text. Section 3.1 gives several examples of activities typically modeled in ICALL systems.

NLP also makes a wider spectrum of activities possible. Consulting the previous example of reading comprehension again, it is clear that the answer to a reading comprehension question can be expressed in a number of ways. This is due to the fact that the goal of the activity is expressing certain *content* instead of sticking to a particular construction or *form*. If we take a historical text about the British Commonwealth, for instance, a question could be "Which are the member countries of the British Commonwealth?". Although the question is rather basic and thus the answer should be fairly predictable, the student still has a choice of formulating the answer: "The members are. . . " vs. "The Commonwealth consists of. . . " vs. "X and Y and Z belong to the Commonwealth".

In traditional CALL, each and every one of these possibilities would have to be enumerated and stated explicitly to allow for exact matches. Furthermore, even partially correct answers would be classified as wrong by the system because they would not produce an exact match. ICALL, on the other hand, is equipped to meet the challenge: Bailey and Meurers (2008) present an approach that matches units of student and target answers on different levels. If matching on the surface level fails, they resort to representations such as lemmas, synonyms (using a resource such as WordNet, cf. Fellbaum (1998)) and employ pronominal anaphora resolution in order to resolve deictic references to the question.

Systems that support these kinds of activities and detailed feedback on learner input are called ILTSs. Students are presented with activities and are expected to provide input that is analyzed on-the-fly. Naturally, these activities have to be produced first. This is where another important area of ICALL comes in: Authoring systems and exercise generators. While activities may and sometimes have to be fully hand-crafted, doing so is not always necessary.

The purpose of an authoring system in ICALL is to generate learning content for an ILTS. For example, Toole and Heift (2002b) present the *Tutor Assistant*, an authoring tool that helps teachers reduce the time required to create exercises. The program offers three exercise types: Build-a-Sentence, Drag-and-Drop, and Fill-in-the-Blanks. The teacher is then required to provide the actual student task and the possible answers. After that, the new exercise can be tested in order to see whether the ILTS that contains the NLP can actually handle this particular example.

But as Toole and Heift (2002a) report, teachers spent a good amount of time trying to come up with meaningful and varied examples for exercises. To deal with this problem, they developed the *Task Generator*, a system that automatically generates exercises from arbitrary input text. It does so by identifying the linguistic phenomena specified by the teacher in the input text and extracting sentences which contain occurrences of the phenomena. Furthermore, the system makes sure that the extracted examples only contain words known to the ILTS.

An attempt to integrate both authoring tool and ILTS into one coherent environment is sketched by MIRTO (Antoniadis et al., 2004). MIRTO is based on the idea that NLP *functions* can be packaged into so-called *scripts*, units that can fulfill the NLP part of an *activity*. The activity is then designed by a language teacher by combining a script with a didactic context (e.g. a text or picture) and a set of instructions. Activities can then be combined into *scenarios*, in which the learner proceeds from activity to activity according to his or her performance.

This section introduced several typical ICALL systems. We explain how our proposed architecture differs from these existing approaches in chapter 3.

2.1.2 Learner Modeling

Learner Modeling is a subfield of User Modeling which in turn belongs to Artificial Intelligence (AI). Where User Modeling is generally concerned with tracking user behavior in electronic systems, Learner Modeling concentrates on learner progress in an ILTS. O'Shea and Self (1983) define a learner model as "any information which a teaching program has which is specific to the particular student being taught" (O'Shea and Self, 1983, p. 143).

As such, a learner model collects information about the learner. How this information is used depends on the actual system. In ICALL systems, learner models can be used to adapt the behavior of the ILTS to the particular learner needs. In an ILTS, the learner model is the only part that is specific to a particular learner. The information that needs to be collected can be gathered in two different ways:

- Explicitly, through questions: e.g. "What is your native language?"
- Implicitly, through observation and analysis results: How well did the learner do at what task?

Implicit methods have generally focused on recording linguistic competence (Amaral and Meurers, 2008). That is, ICALL systems mostly take into account what grammatical structures are problematic for the learner in terms of the quantity of errors made when asked to produce the respective phenomena.

For example, the *E-Tutor* (Heift, 2001, 2003, 2004) counts learner's mistakes in terms of specific grammatical phenomena such as subject-verb agreement and subcategorization. After completing a set of questions, the learner is then notified what his or her specific difficulties are. Other characteristics of the learner are not taken into account.

Amaral and Meurers (2008) propose a learner model that is informed by Second Language Acquisition (SLA) theory and extends the conceptualization of students' skills beyond linguistic competence. Specifically, they argue that learner models should include the strategic competence of the student with respect to the task. A reading comprehension question, for instance, requires scanning a text for specific pieces of information that the student needs to answer the question. In contrast to producing a certain grammatical phenomenon, this is a strategic skill that does not express linguistic competence or incompetence. Moreover, Amaral and Meurers argue for the inclusion of L1[3] transfer information into the learner model in order to help the ICALL system pinpoint the source of an error. An example is a typical lexical transfer error from English to German: English "sensible" is very similar to "sensibel" in German, but while the former means "reasonable", the latter means "sensitive".

In the context of this thesis, learner models are important because our ILTS can

[3]The term 'L1' generally refers to the learner's native language.

dynamically adapt the feedback strategy to the student's needs. How this is done is explained in chapter 4.

2.2 NLP Architectures

NLP applications usually involve multiple analysis components feeding their results to the next component. Thus, one can speak of an NLP application as a pipeline. Traditional NLP pipelines face two major problems, outlined below.

First, in a pipeline, the output a component gives does not necessarily contain all the information that was present in its input. Instead, the input is replaced by the new analysis of the component. In particular, access to the original input text is usually not possible for later analysis components.

Second, such pipelines are often idiosyncratically designed and incompatible with each other, ad-hoc solutions that work only for a very specific task. This leads to a high amount of re-implementation as software components are not easily reusable.

To address these issues, NLP was recast in recent years as an annotation process: Instead of one component replacing its input with its output, the original input is gradually enriched with annotations that represent analysis results. The impact of this idea can also be seen in the success of the eXtensible Markup Language (XML)[4] as the language of choice for corpus annotation schemes such as the one devised by the Text Encoding Initiative[5]. Annotating a corpus then means to enrich the original text with annotations that contain additional information about parts of the text.

One annotation-based framework that has become more and more popular in the NLP community in recent years is UIMA. As UIMA is an important structural ingredient of our system, we introduce it in the next section.

UIMA

The Unstructured Information Management Architecture (UIMA) (Ferrucci and Lally, 2004) is an attempt by the IBM Corporation to provide a standardized architecture for analysis of text and other unstructured data. An important aspect of UIMA is modularity: By providing definite interfaces for analysis components to hook into, UIMA ensures that no component does more than it should do or knows

[4]http://www.w3.org/XML

[5]http://www.tei-c.org

more than necessary to fulfill its task. We will introduce the important concepts in the UIMA framework in the following.

Analysis results are stored and passed on in the Common Analysis System (CAS). The CAS (Götz and Suhre, 2004) is a highly flexible data repository which contains the original input and an index for annotations. An important property of UIMA's annotation system is that it is **referential**: annotations are stored separately from the original text in contrast to **additive** annotation, where annotations are inserted into the text (cf. e.g. Wunsch (2003, ch. 3)). Annotations that are to be placed into the index must first be declared in the user-defined type system. UIMA builds on the idea of typed feature structures known from grammar formalisms such as Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994). Specific annotation types derive from the general type `Annotation` and declare what features they have and what the types of the features are. Through this general approach, type systems can be arbitrarily complex.

Type systems are declared in XML files independent of a specific programming language. For Java, the most widely used language in conjunction with UIMA, a utility called *JCasGen* then translates the type declarations into concrete Java classes which can be used in implementations.

The modules in a pipeline, also called Analysis Engines (AEs), store their results by creating instances of annotation types and associating them with the input, i.e. putting them into the CAS index. Consequently, the CAS always retains the original input while making it possible to add any number of annotation layers. AEs can be accumulated to form aggregate AEs, thus forming an NLP processing chain. AEs that are run later in the chain can retrieve results from earlier AEs and build on them.

All UIMA components have to declare their metadata in XML descriptor files. Among the information that has to be provided there is what type system is to be used and what the name of the implementation class is. An important possibility for AEs is that input and output specifications can be set: Every AE can declare what annotations it needs in order to do its task.

Additionally, UIMA features the so-called Flow Controller whose task it is to direct the CAS within an aggregate AE. In its simplest form, the flow controller has a linear sequence of AEs and runs one after the other on the CAS it receives from the framework. However, UIMA is modular enough to allow users to create their own flow controller implementations that use other criteria to decide which AE to apply when. We illustrate the possibilities this offers to our ICALL system in section 4.2.

For now, consider a simple example for illustration: Suppose we take as input a single document of a corpus and want to do Named Entity Recognition (NER) on it. In order to do that, we need tokenization and part of speech (POS) tagging. Thus, the tokenizer can create annotations of type `Token` which the tagger can then enrich with POS information. Finally, the NER module can use these enriched `Tokens` for its task. All three components would be combined into an aggregate AE with a linear flow, so that the order in which they run is fixed.

Note that through the CAS abstraction, none of the individual AEs had to directly rely on a particular other AE. Instead, they rely on particular annotations that have to be present. This makes it possible to swap a particular implementation of e.g. a tokenizer without disrupting the whole system. In contrast to relying on a particular tokenizer, other AE simply declare that they need annotations of type `Token` to work on.

3 Research Issues and Context

Having introduced the essential notions in chapter 2, we can now turn to the central ideas of this thesis. As we stated in the introduction, this thesis has two general aims.

First, we advance the notion that ICALL benefits directly from annotation-based processing in contrast to traditional NLP pipelines. To illustrate why we think this is necessary, we introduce the advantages in section 3.2.3 and give several references to more traditional systems, outlining their deficiencies in section 3.4.

Second, we claim that ICALL applications and ILTS in particular need to integrate different information sources in order to give meaningful and helpful feedback to learners. Specifically, ILTS need to integrate explicit activity models and learner models into their diagnosis process. We present our vision of a *demand-driven* ILTS that adapts to the needs of learners and instructors in section 3.3.

However, before we launch into these key questions, we present the ideas that led to this thesis. As our system is based on the TAGARELA system, we start with a brief discussion of TAGARELA itself before presenting the open issues in section 3.2.

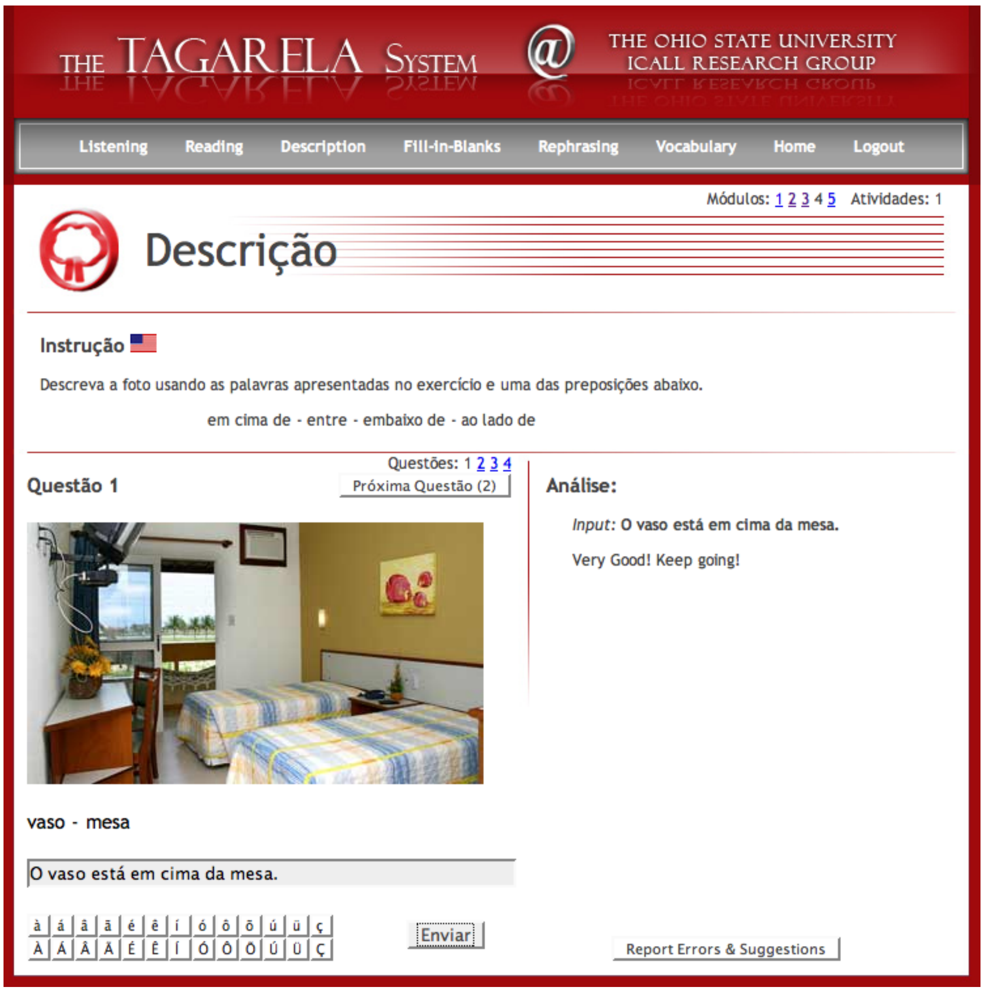
3.1 TAGARELA

TAGARELA is an intelligent workbook for Portuguese, developed by Luiz Amaral at The Ohio State University in the context of his PhD dissertation (Amaral, 2007) for a specific Portuguese language course. It is web-based and features six different activity types:

- *Listening Comprehension*: Learners listen to a short recording and answer questions about it
- *Reading Comprehension*: Learners read a text and answer questions about it
- *Picture Description*: Learners describe a picture using given words
- *Fill-in-the-blanks*: Learners fill in the correct forms of missing words

- *Rephrasing*: Learners express a sentence in different words
- *Vocabulary*: Learners form sentences using specific vocabulary

Each of these activity types has one or more modules that differ with respect to difficulty and covered material. TAGARELA takes the learner input to a task, analyzes it and reports feedback to the learner. It employs AJAX[1] technology to keep response times short by avoiding to reload the whole page. An example screenshot of the description activity is shown in figure 3.1.



Instrução

Descreva a foto usando as palavras apresentadas no exercício e uma das preposições abaixo.

em cima de - entre - embaixo de - ao lado de

Questão 1

Análise:

Input: O vaso está em cima da mesa.

Very Good! Keep going!

vaso - mesa

O vaso está em cima da mesa.

à á â ã é ê í ó ô õ ú ü ç
À Á Â Ã É Ê Í Ó Ô Õ Ú Ü Ç

Figure 3.1: The TAGARELA description activity

TAGARELA's analysis is driven by a central module, the analysis manager. It

[1]http://en.wikipedia.org/wiki/AJAX

calls the other modules in order to obtain information about the student input. Two kinds of analysis are used: **form**-based and **meaning**-based analysis. Form-based analysis is concerned with linguistic well-formedness of the input whereas meaning-based (or content-based) analysis compares the learner input to the most likely target answer and determines whether the relevant concepts have been included in the learner input. The various analysis modules are discussed in section 5.3 where we contrast our implementation with the original TAGARELA implementation.

After the analysis modules have been run, the result is passed to the feedback manager. The feedback manager has two main tasks: error diagnosis and feedback message generation. Error diagnosis is based on the information gathered by the analysis modules. The feedback manager looks at the analysis results and decides what to report feedback on. The generated feedback message is then passed back to the learner. The TAGARELA architecture pursued by Luiz Amaral is shown in figure 3.2.

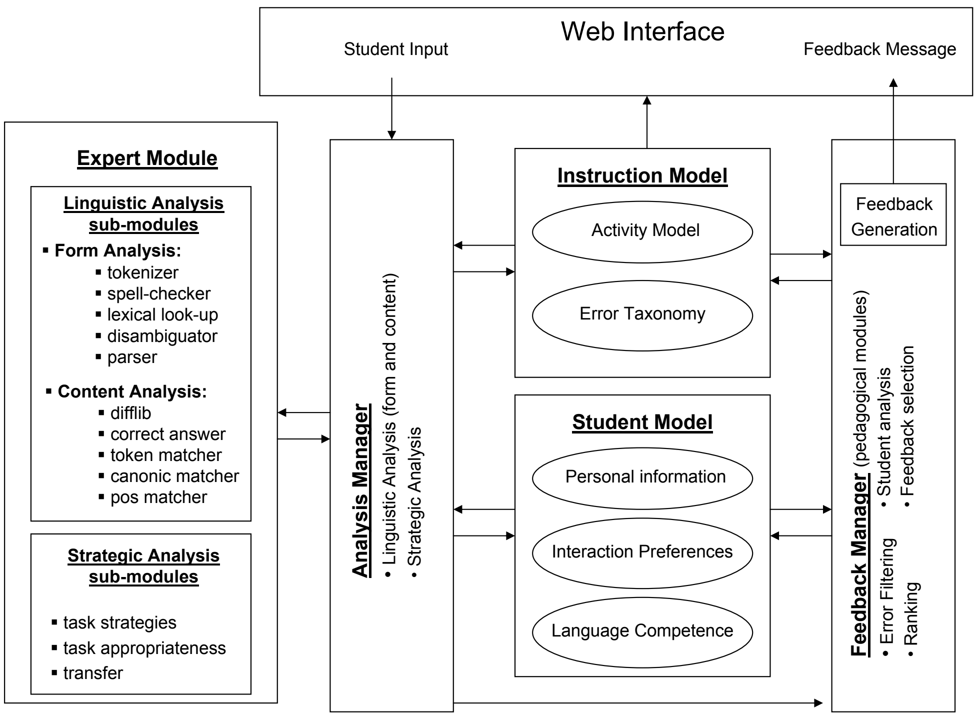


Figure 3.2: The architecture envisioned for TAGARELA (Amaral, 2007)

Feedback

TAGARELA and its activities were designed with pedagogics in mind. Apart from the design of the user interface, this primarily concerns the feedback process. TAGARELA's feedback strategy follows two key principles:

1. Only one feedback message is shown at a time. Recent research (Heift, 2003) has shown that simply listing all errors is not helpful for learners. TAGARELA uses scaffolding (cf. e.g. Hyland and Hyland (2006)): It tries to gradually lead the learner towards the correct answer through feedback messages, intending to help learners develop self-editing skills.
2. Feedback on meaning-based errors is preferred over feedback on form-based errors. If a learner makes both types of errors, conceptual errors are considered more important and hence feedback concentrates on them.

The combination of these two constraints means that the system has to choose the most significant error that is to be used for feedback.

3.2 Open Issues

Although TAGARELA can be considered an impressive achievement both from an SLA perspective and in terms of robustness concerning the implementation, there is room for improvement in several research directions. Amaral and Meurers have developed specific ideas in recent papers, which however have not been implemented. We will present these ideas here and show how they lead to the present work.

3.2.1 Explicit Activity Models

Amaral and Meurers (2007) argue for explicit **activity models** and their use to dynamically adapt NLP analysis and feedback to the needs of a particular activity. The general idea is that the more the system knows about the goals and properties of an activity, the better it can adapt to it. This adaptation is proposed for two specific places in the system.

First, the analysis should only try to detect information not already present in the activity model. The implication is that e.g. target answers in activity models can already be annotated with linguistic properties. This is particularly desirable if automatic analysis is known to be problematic for a given case. Instead of re-establishing the linguistic analysis, the analysis manager would know that certain properties are already present and thus further NLP can concentrate on producing analysis results not already present in the activity model.

Second, the activity model should determine the feedback strategy. This relates to the idea that in designing a particular activity, instructors usually have certain learning goals in mind which differ substantially between activity types. For example, an instructor might prefer the system to give feedback on missing concepts for a reading comprehension task, because the learning objective is to identify certain content in the text and use it for the answer. A vocabulary exercise, on the other hand, might be targeted at helping the learner learn specific constructions, thus it focuses on form rather than content. If the preferred type of feedback is given in the activity model, the system can react to it and prioritize its feedback accordingly.

However, Amaral and Meurers (2007) give no concrete proposal as to how these ideas should be implemented. TAGARELA primarily draws on the activity model for presentation of exercises, not for processing or feedback.

3.2.2 Extended Learner Models

In (Amaral and Meurers, 2008), the authors present their view of how **learner models** could be extended beyond capturing learner's grammatical competence. Recall that state-of-the-art ICALL systems (e.g. *E-Tutor*, see Heift and Nicholson (2001)) only record how good learners are at producing certain language phenomena, such as subject-verb agreement.

Amaral and Meurers (2008) propose instead to widen the perspective of learner models to include other criteria. Their proposed learner model is divided into three core areas:

1. Personal Information: A repository for general information such as age, gender, and native language.
2. Language Competence: A record of the language skills, sub-divided into the following areas:
    - Linguistic Properties: Form-driven and content-driven properties, reflecting exactly what can be detected by the analysis modules in TAGARELA (see section 5.3).
    - Task Appropriateness: Keeps track of how well the learner demonstrates linguistic knowledge with respect to specific activities.
    - Task Strategies: Keeps track of skills such as scanning a text which are required to successfully complete a task. Required strategies are given in

the activity model.

3. L1 Transfer: Stores information on possible L1 transfer errors based on the learner's native language.

Amaral and Meurers also emphasize the use of such a learner model in an actual annotation-based ILTS to support inferences about learner competences and make learner-specific feedback possible. For example, if a learner repeatedly failed to include relevant concepts in his answer to a reading comprehension question, the system can deduce that he did not correctly use the task strategy of scanning a text. The system can then react with an appropriate feedback message, such as "Try to scan the text more carefully to include all the key concepts in your answer".

3.2.3 Annotation-Based Processing

Besides motivating explicit activity models and their use, Amaral and Meurers (2007) also motivate an **annotation-based processing** architecture for ICALL systems. We introduced annotation-based NLP in section 2.2 as a recent trend in NLP that views automatic processing as parallel to manual annotation.

Following this idea, Amaral and Meurers (2009) illustrate concrete linguistic phenomena whose treatment can benefit from annotation-based processing. Two phenomena are discussed, based on the analysis of TAGARELA's system logs. In both cases, the result of inflexible pipeline processing results in unhelpful feedback for the learner and can be remedied by a true annotation-based architecture.

The first phenomenon concerns contractions. In Portuguese, prepositions can contract with several other part of speech, such as determiners, for instance. We shall use the determiner case to illustrate the problem. Example 1, taken from Amaral and Meurers (2009), shows a possible answer to a reading comprehension question in TAGARELA.

(1)	a.	O	Amazonas	fica	**na**	região norte.
		the	Amazon	lies	in $\text{the}_{\text{FEM.SG}}$	region North
	b.	*O	Amazonas	fica	**no**	região norte.
		the	Amazon	lies	in $\text{the}_{\text{MASC.SG}}$	region North
		'The Amazon is in the northern region.'				

The sentence in 1a is the correct answer. The preposition *em* is contracted with the determiner *a* to yield *na*. In 1b, the learner used the incorrect *no* (*em* + *o*) and thus the system should respond with an agreement error between *no* and *região*.

However, the system has already transformed *no* to *em o* internally and reports an agreement error in the sequence *o região norte*. This confuses the student who did not type in any *o* and fails to direct him towards the correct answer.

In an annotation-based architecture, the system could have kept both the surface form *no* and a deep representation *em o* and, by associating the two, could have given feedback based on the surface form instead the deep form.

The second phenomenon are accented characters. An accented character in Portuguese sometimes is the only orthographic difference between totally unrelated lexical items. Again, we illustrate the problem by an example taken from Amaral and Meurers (2009), shown in 2.

(2) a. Marcos **é** brasiliano.
Marcos is Brazilian.

b. * Marcos **e** brasiliano.
Marcos and Brazilian.

The sentence in 2a again presents the correct answer, using the third person singular of *ser* ('to be'). However, the learner entered the sentence in 2b which uses the unaccented *e* ('and').

The system tells the learner that an important verb is missing and that there are unnecessary words in the sentence. Again, this is confusing to the student, who, depending on his native language, might not perceive a striking difference between accented and unaccented characters.

A more helpful message in the direction of missing or extra accents could be supported by an annotation-based architecture. Concretely, Amaral and Meurers (2009) suggest annotating multiple possible analyses of strings known to have accented and unaccented versions so that the system can draw on this information when disambiguating a sentence. Furthermore, they suggest drawing on the learner model proposed in Amaral and Meurers (2008) in order to spot typical errors based on the learner's native language.

However, Amaral and Meurers (2009) do not show what data structures are needed for such annotations and how mappings from surface tokens to other analyses should be encoded.

3.3 Demand-Driven ICALL

In section 3.2, we have seen several possible improvements for TAGARELA and ILTSs in general. They affect all parts of the system except the user interface: NLP processing, error diagnosis based on the processing, and finally feedback generation.

On the whole, these improvements are driven by pedagogic demands that seek to make the ILTS more effective: Improved learner models help to make feedback more individualized, a clear advantage over streamlined one-for-all systems. Putting information into the activity model allows the activity designer to adapt the behavior of the ILTS to the specific needs of the activity. And last but not least, annotation-based processing makes for an extensible, powerful processing mechanism that can be conveniently adapted to the specifics of various natural language phenomena in order to flexibly annotate linguistic properties needed for error diagnosis.

Current systems such as the ones discussed in section 3.4 do not provide a complete big-picture perspective of an ILTS that can accommodate innovations such as the ones described above. Rather, they can be characterized as ad-hoc systems that use a fixed sequence of NLP modules for a tightly constrained set of activities or restrict the input of learners to be able to process it.

Although TAGARELA was an important step towards a unified architecture, it did not fully realize the ideas presented in section 3.2. In particular, TAGARELA implements only a default feedback strategy that is not geared towards the needs of either activity or learner model. Moreover, it does not use annotation-based processing as is possible using the UIMA framework. Instead, the analysis manager calls each NLP module specifically and the modules themselves have to deal with transformation of their input into the data structures they can work with.

This thesis presents an architecture that integrates the different pedagogic demands outlined in section 3.2 into one coherent system while still retaining and improving the functionality of the original TAGARELA. For that purpose, TAGARELA was recast in the new architecture and transformed from a prototype into an example instance of our architecture that illustrates its power and flexibility.

A high-level view of the central concepts in the architecture is presented in chapter 4. Chapter 5 then deals with the actual implementation.

3.4 Other Systems: Rigid Sequences of NLP Modules

To illustrate the shortcomings of recent ICALL systems, we will present some of them here and show in what way they seem restricted to us.

3.4.1 Herr Kommissar

Herr Kommissar (DeSmedt, 1995) is a conversation machine for German that puts the learner into the role of a police inspector. The system answers questions about specific information related to the case posed by the learner. If the learner makes errors, the system gives feedback about them.

The NLP analysis used includes spell checking, syntactic parsing and even semantic mapping based on an internal micro-world. Through this modeling, *Herr Kommissar* is able to infer an appropriate answer for the learner's questions. A natural language generation component then produces the final answer string that is presented to the learner.

The clear problem with this highly ambitious approach is domain coverage. *Herr Kommissar* aims to model actual world knowledge so that it can answer learner's questions. This is only possible by restricting the possible learner input to the type of questions that a police inspector poses to witnesses. Consequently, no domain variation can be expected from the system.

3.4.2 ALICE-CHAN

ALICE-CHAN (Levin and Evans, 1995) is an ILTS for introductory Japanese. Students are presented with exercise instructions that provide a context for the question they need to answer. The example presented in Levin and Evans (1995, p. 79) shows a list of historical events about which the learner then has to answer a question.

The NLP modules are organized in a strict pipeline fashion where each module feeds its output to the next. As soon as a module fails, processing is aborted. For example, if the parser fails, no more analysis is done and the student is presented with the message "Your sentence could not be understood". The system does not try to recover by using the output of the morphological analyzer in a more shallow way.

This error handling approach is problematic because there is the danger of not giving any helpful feedback to the student in case an intermediate NLP module fails. The system goes through its analysis modules like through a decision tree: if specific conditions are met (e.g. the sentence cannot be parsed), it terminates.

3.4.3 German Tutor

The *German Tutor* (Heift and Nicholson, 2001) is a web-based ILTS that fulfills the grammar part of a web-based German course. It contains six activity types: Dictation, Build-a-Phrase, Which-Word-is-Different, Word Order Practice, Fill-in-the-Blanks and Build-a-Sentence.

Although there seems to a be a rich choice of activities, they are geared towards expecting input that one specific NLP pipeline can handle. This creates the impression that pedagogics are tailored to meet NLP possibilities instead of NLP serving pedagogics.

Also, similar to *ALICE-CHAN*, the input goes through one module after the other until an error is found or the end of the pipeline is reached. This ties error diagnosis directly to analysis which reduces the flexibility of the system. As we will see in the next chapter, in an annotation-based architecture, the diagnosis module can choose which information to use after all analysis has been done.

4 Concepts and Design

After having motivated the need for a unified architecture in chapter 3, we now turn to the central concepts in the architecture.

From a high-level perspective, the task of an ILTS can be viewed as follows: Given a learner **input** from a certain known **learner** to a known **task**, assess the input and give feedback about it. Thus, the three sources of information for the system that can be directly derived from this description are the input itself, the activity model and the learner model. Diagnosis and feedback should take these three information sources into account and base their response on them.

Naturally, the learner input needs to be analyzed first, and this is where NLP comes in. In order to be able to say something about the input produced by the learner, certain NLP analysis modules need to be called to determine linguistic properties of the input. As argued in Amaral and Meurers (2007), the needs of activities differ with respect to linguistic processing: A fill-in-the-blanks activity can be realized with spell-checking and lexicon lookup, but a rephrasing activity needs full syntactic parsing. This variation suggests that there should be an analysis controller that handles the interdependencies between activity needs and analysis modules.

In this chapter, we introduce all these elements and describe their task in the system. Section 4.1 starts with the three possible sources of information before section 4.2 continues with the analysis controller. Finally, section 4.3 explains how the information obtained can be combined for pedagogically effective feedback.

4.1 Information Sources

The learner input, activity model and learner model constitute the body of information an ILTS can use in order to do its job. We will present what these sources of information contain in our system in the following.

4.1.1 Learner Input

The learner input starts out as the bare unannotated string the learner types into the system's user interface. In order to make error diagnosis possible, the system needs to run a series of NLP modules on it which identify and annotate linguistic properties. Annotation is done using a well-defined hierarchical system of annotation types, similar to a corpus annotation scheme. As pointed out in section 2.2, NLP annotation is thus parallel to human annotation.

If a target answer is available to the system, it can be processed in the same way as the learner input. TAGARELA does this, for example, to be able to compare linguistic properties such as canonic forms of tokens in both answers. Other systems such as BANZAI (Nagata, 2002) also process the target answer similarly to the learner input.

Using an annotation scheme makes it possible for the system store information at different levels. In traditional NLP pipelines, different layers of information can only be added in a *sequential* fashion: a stream of characters is split into tokens by a tokenizer, then tagged by a POS tagger. It is not possible for the tokenizer to already add POS information.

However, situations such as the contraction case in Portuguese (see section 3.2.3) show that natural language phenomena exist where this is desirable: a contraction can be analyzed as having a deep representation that consists of two tokens. If the tokenizer knows these special cases, it can already assign POS to the underlying tokens and thus make it possible for other analysis modules (such as a parser) to consult this deep representation for their task. We present concrete instances of this representation advantage in section 5.3.

Our annotation scheme is organized around three main units to which all other information is attached. **Tokens** are the smallest unit that we identify. Attached to a token is the result of spell checking and lexical entries (containing POS and morphological information) according to the system's lexicon. As there is frequently more than one lexical entry for a given surface form, the list of lexical entries is subsequently reduced to one by a disambiguation module. **Phrases** represent partial or complete parse trees, built of tokens and other phrases. For this purpose, they have a list of daughter nodes that represent their subtrees. Also, as phrases can belong to other phrases, a reference to their parent node can be stored with them. Phrases can be annotated with the result of daughter agreement, such as subject-verb agreement in the case of complete parse trees. How this is done is described in detail in section 5.3.7. The last type of annotation does not refer to any particular portion of the input string but rather to the input as a whole. **Analysis results** are a repository for information such as basic string matching or the result of the content assessment module (see section 5.3.8) where tokens are matched on different levels in the learner input and the target answer. It also holds global agreement checking results that can be directly used by the diagnosis part of the system.

4.1.2 Activity Model

The activity model stores everything the system needs to know about a certain activity. This includes information the system needs only for presentation, such as the task instructions and possibly other learning material such as hints or pictures, depending on the type of activity. The advantage of specifying this explicitly in one place makes the system easily extensible: adding a new activity means defining an activity model and providing the required information in it.

Using explicit activity models in ICALL systems is a relatively recent concept. To our knowledge, no system prior to TAGARELA attempted to isolate activity-specific information in one central place. Rather, most ICALL systems are made to handle a fixed set of activities whose specifics are encoded into the systems themselves. However, this leaves implicit how and to what extent activities differ among each other.

Apart from presentational information, other types of information in the activity model exist that can be used for processing and feedback. **Target answers** provide the system with a reference that the learner input can be compared to. The content matching modules (see section 5.3.8), for example, draw heavily on this as matching is of course only possible with a direct counterpart to the learner input. As the original TAGARELA, our system supports multiple target answers in order to allow more variation on the learner's part. In addition to target answers, a list of **required words** can be specified which put additional constraints on the lexical material the learner uses. This makes sense for activities where the use of specific words is a learning objective, as is the case with TAGARELA's vocabulary activities. Table 4.1 gives an excerpt from an activity model to illustrate the information provided there.

The most crucial piece of information for our architecture is the definition of **feedback preferences** in the activity model. As stated in section 3.3, one of the aims of this thesis is to develop further the idea of giving activities control over the behavior of the ILTS. Amaral and Meurers (2007) argue for control over two

L1 instructions	Re-write the sentences using the words between parentheses.
L2 instructions	Reescreva a frase abaixo usando a expressão entre parênteses.
Exercise/Question string	Meu nome é Paulo da Silva. (chamo)
Target answer	Eu me chamo Paulo da Silva.
Required words	Paulo, Silva
Preferred error type	`FormError` (NB: all form errors)

Table 4.1: An excerpt from a Rephrasing activity model

specific aspects: the types of annotations that NLP should provide and the type of feedback the system should give.

The need for NLP guided by the activity model raises the question of how exactly such guidance should be encoded in the model. We will describe three strategies and explain their advantages and disadvantages. First, activity models could specify their activity type, such as *Fill-in-the-Blanks* and the system could then map this to a particular processing approach, such as "Spell checker + lexicon lookup". However, this does not really give the activity model control because the mapping from activity types to processing strategies would still have to be encoded in the system. It would be a global, finite encoding of well-known activity types, thus limiting the possible processing strategies. A second approach would be to specify processing components in the activity model, in which case the above-mentioned Fill-in-the-Blanks case would specify "Spell checker + lexicon lookup" directly. This is more flexible, as any possible processing chain could be specified, but burdens the activity designer with the necessity of knowing about individual NLP modules and how they can be used for a given activity. In our opinion, this knowledge should be encoded in the system.

We settled for a third option that we call **feedback-driven processing**. What activity designers are really interested in is how the system responds to input for their activity, i.e. what feedback is given. After all, this is the only thing the learner sees after typing in the input. Hence it seems natural to us that the processing strategy should be inferred by the feedback needs. If the type of preferred feedback is given in the activity model, the system can deduce[1] what annotations are needed for such feedback. As a first step, we encode the preferred feedback type in terms of the **error types** we want to diagnose. As we describe in more detail in section

[1]How this mapping is done is explained in section 4.3.1.

Activity type	**Targeted error types**
Reading Comprehension	Meaning-based errors
Listening Comprehension	Meaning-based errors
Description	Missing concepts, agreement errors, word form errors
Fill-in-the-Blanks	Word form errors, missing concepts
Rephrasing	Form-based errors
Vocabulary	Missing concepts, extra concepts, word form errors

Table 4.2: Activity types and their targeted error types

4.3.1, our system uses an explicit error taxonomy that covers all errors which can be detected by the system. Through the specification of its "preferred error types", an activity can tell the system what to do. For example, a Fill-in-the-Blanks activity will typically want to focus on form-based feedback on the word level in order to help learners master the use of specific lexical material. In order to achieve that, the activity designer can specify the preferred error type `WORDFORM` to indicate that wrong word forms should be targeted. By deducing the required annotation types for `WORDFORM`, the system can call the appropriate modules to provide these annotations.

For illustration purposes, we list the activity types and their targeted error types[2] in table 4.2. The variation in targeted error types makes clear that activities do differ in terms of didactic purpose. By encoding targeted error types into the activity model, our architecture allows for the formulation of such didactic aims. Concrete examples where this is relevant are shown in section 6.1.2.

Although activity models in our system already encode crucial information and are used to a wider extent than in the original TAGARELA, there is room for improvement. Amaral and Meurers (2008) suggest to also make the nature of expected input explicit: if the system knows whether the input is to be individual words, phrases or sentences, it can adapt better. At the moment, the distinction is made implicitly by the system.

Another suggestion by Amaral and Meurers (2008) is the inclusion of information that can be used for inferences about learner performance. As we reported in section 3.2.2, one of the ideas is to associate activities with **task strategies** that are necessary to complete the activities. Using these associations, the system can infer

[2]The error types were provided by Luiz Amaral who is a trained language instructor and SLA researcher.

whether a learner repeatedly failed to use a certain strategy, such as scanning a text or listening to a recording.

But there are also possibilities that directly affect the assessment of the learner's answer. **Answer typing** (cf. e.g. Krishnan et al. (2005), Pinchak and Lin (2006)) is a sub-discipline of Question Answering which tries to determine the semantic type of a given question, such as location, time or person. The connection from ICALL to Answer Typing has already been established by contributions such as (Gerbault, 1999). It can improve NLP processing in an ILTS because it provides information on what type of answer is expected, which in turn enables more accurate feedback. Consider the question "Who was the third president of the United States?" as an example. If the system knows that a *person* is looked for, it can respond to wrong answers such as "The president of the United States is very powerful." with a hint such as "Try including a name into your answer.". The default TAGARELA feedback message, on the other hand, would simply point out that too many relevant concepts are missing in the learner answer, which is also correct but less informative.

4.1.3 Learner Model

As introduced in section 2.1.2, the learner model is a repository for all kinds of information concerning a specific learner. In our system, similar to the activity model, the learner model contains information for presentation and, more importantly, indicators of the learner's performance.

The presentation part records the last input the learner gave to a certain question. Along with the learner ID, this enables the system to show the learner what he entered previously if he chooses to redo an exercise, making it possible for him to correct his mistakes. This feature was already present in the original TAGARELA and we adopted it in our system.

As far as performance indicators are concerned, the main difference to the learner model implemented in TAGARELA is the use of **error counts** for the error types the system can detect. This is similar to the learner model of *E-Tutor* (cf. e.g. Heift (2003)) which also models student performance in terms of detectable errors. In our system, the error taxonomy is explicit (see section 4.3.1) and contains all error types that can be detected by our system at the moment. In a way, the learner model is a concrete instance of the error taxonomy for a specific learner: detection possibilities (error types) are mapped to detected instances (error counts).

By comparing counts to the total number of detected errors, the system can then get a measure of which errors are typical for a given learner. This knowledge can in turn be used to prioritize feedback on the typical errors. We describe the process of prioritizing errors in more detail in section 4.3.2 and give concrete examples in section 6.1.3.

Although our learner model is superior to the one actually implemented in the original TAGARELA, it must still be considered preliminary. As we described in section 3.2.2, Amaral and Meurers (2008) propose extending the learner model to include indicators about language produced in the context of concrete activities which can then support much finer-grained inferences about their skills. A first step in this direction from the point where we are now would be to count errors per activity instead of counting them in total. This would provide information on which errors occur for which learner in connection with a certain activity. We plan to integrate this in the near future. In general, it can be said that the learner model should make references to the activity model wherever possible to give the system an interpretation context for the learner language produced.

In the long term, we also plan to use the learner model **during** NLP processing for disambiguation. A good example is the Portuguese accent case described in section 3.2.3: If the learner model can capture the fact that a certain student often forgets to use accented characters instead of unaccented ones, NLP analysis can consult the learner model and add the accented version as an additional annotation. This in turn enables feedback centered on the missing accent instead of a wrong lexical choice (the unaccented version represents a different lexeme).

4.2 Analysis Controller

As stated in the introduction to this chapter, the analysis controller in our architecture is responsible for determining which NLP modules need to be run for a given activity such that the feedback requirements of the activity can be met. The idea is to read the activity model and, according to the pedagogical purpose and feedback policy encoded there, decide on a processing strategy. As described in section 4.1.2, encoding of feedback preferences (and hence the pedagogical goal of an activity) is currently realized through the specification of targeted error types. The analysis controller reads these and asks the diagnosis manager which diagnosis sub-modules can detect the given error types. Having determined the relevant sub-modules, the diagnosis manager collects from them the information on what annotations are needed to diagnose the given errors. These are passed back to the analysis controller which can then start the actual NLP processing. The procedure is shown in figure 4.1.

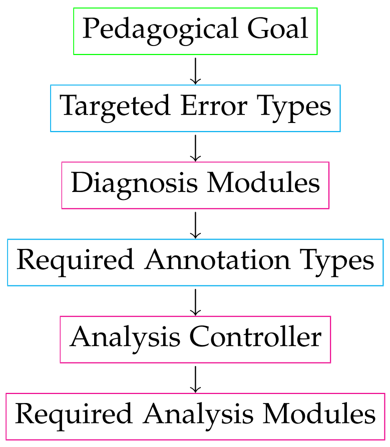


Figure 4.1: Path from error types to analysis modules

The last step where the analysis controller figures out which actual modules to run to get the required annotations is made possible through input and output specifications of analysis modules. As we describe more concretely in section 5.3.1, modules make their input and output annotations known. By using this information, the analysis controller can map the required annotation types to actual analysis modules.

This approach achieves two important goals. First, NLP is driven by **pedagogical feedback requirements**. By specifying an error type in the activity model, the activity designer can directly influence NLP without having to know about it. For example, specifying agreement errors as targeted error type will require the annotation feature 'agreementResults' of annotation type 'AnalysisResults'. To provide 'agreementResults', the analysis controller has to run the following modules: tokenizer, lexicon lookup, local disambiguator, parser, agreement checker. This also implies resolving module dependencies, we explain how this can be done in section 5.3.1.

The second goal is the **extensibility** of the system. By making information such as error types and annotation types explicit, adding new capabilities to the system is straightforward. New analysis modules provide new annotations which in turn make new detectable error types possible. However, the general way the analysis controller works does not have to be changed.

Although the infrastructure for fully dynamic NLP is in place, the current analysis controller still uses a predefined linear sequence of annotation modules. The reason for this is that running less than all available modules currently does not yield any benefit: we have exactly one module for each task (e.g. tokenizing, parsing) and having more annotations on the input than needed for the required feedback does not have any immediate disadvantages. For these reasons, we decided to leave the controller as is in order to concentrate on other aspects of the system.

However, using a dynamic controller will become crucial once we have more than one analysis module for a given task: we then need to decide which one is best suited for the input data we have. We will sketch informally here how such a decision could be modeled: if we view the set of available analysis modules as a set of states with transitions from module to module, a particular processing approach would be represented by a possible path through these states. Hence, we would have an **automaton of analysis modules**. As there will be multiple outgoing transitions from some states, we need to decide which transition to take. This decision can be based on the result of previous analysis results. For example, if string matching between the learner answer and the target answer succeeds, there is no need for any further analysis and we can proceed to a final state. Moreover, the decision can be based on particular learners. If we know from experience that a certain parser does not work well with input from a certain learner, we can dynamically choose another parser instead.

Such decision factors could be combined according to definable criteria and attached to the transitions in the analysis automaton as **weights**. In order to represent different combinations of learners and activities, different sets of weights could be stored to effect different processing behavior. Moreover, a scoring mechanism could be used to adjust the weights based on experience. The result would be a system that adjusts to learners and activities and gets better over time. However, it is still unclear how scoring of a given path through the analysis automaton could be realized, i.e. how the relative success of an analysis module on a given input should be evaluated. This is a topic for future research.

superficially well-formed	*and*	appropriate	**erroneous or correct**
superficially well-formed	*but*	inappropriate	**erroneous**
superficially deviant	*but as far as can be judged*	appropriate	**erroneous**
superficially deviant	*but as far as can be judged*	inappropriate	**erroneous**

Table 4.3: Error distinction made in Corder (1974, p. 124)

4.3 Diagnosis and Feedback

Diagnosis and feedback are often confused in the literature and used interchangeably. We explicitly separate these concepts here and make the following distinction:

- **Diagnosis** examines the annotated input and determines the errors made by the learner
- **Feedback** is the system's response to the learner, informed by the results of diagnosis

Section 4.3.1 will give details about the diagnosis process before section 4.3.2 discusses feedback.

4.3.1 Diagnosis

Diagnosis deals with determining the learner's errors. It can draw on Error Analysis, a subfield of Applied Linguistics, that investigates the process of manually finding errors. Generally, Error Analysis is concerned with recognizing and describing errors on multiple linguistic levels, such as syntax and semantics. Corder (1974) categorizes language produced by learners according to *superficial well-formedness* and *appropriateness*. The combination of these two classification categories yields the possibilities listed in table 4.3.

According to Corder (1974), language produced by learners can be erroneous either because of inappropriate content or non-wellformedness or both. Corder's distinction is roughly equivalent to TAGARELA's division of errors into form-based and meaning-based errors Amaral (2007, p. 91/92). Both in TAGARELA and in our system, these correspond to the two groups of analysis modules that do form-based and content-based analysis, respectively[3]. In contrast to TAGARELA, where the

[3]See section 5.3 for a description of the individual modules.

Meaning-based	Form-based
Wrong lexical choice	Punctuation error
Missing concepts	Capitalization error
Extra concepts	Spelling error
Missing and extra concepts	Wrong word order
	Agreement error
	Wrong word form
	Missing preposition
	Extra determiner

Table 4.4: Error taxonomy

error taxonomy is implicit and hard-coded into the diagnosis module, our error taxonomy (given in table 4.4) is explicitly declared, separately from the actual diagnosis module.

The original TAGARELA diagnosis module works in a data-driven way. It examines the output of the various analysis modules in a fixed order and stops as soon as it finds an error. The order in which analysis results are examined reflects the feedback priority of TAGARELA: feedback on content is prioritized over feedback on form.

Diagnosis can be seen as an interpretation task: Analysis results support statements about learner errors. The interpretation context for these statements is given by the learner model and the activity model. As an example, let us imagine a case where the content assessment module (see section 5.3.8) matched all the tokens but simple string matching of learner and target answers failed. In such a case, the system can deduce a word-order error, because all and only the required tokens are present but string matching still does not succeed. Hence, the system concludes that the words present must be in the wrong order.

TAGARELA's default diagnosis strategy works well, but does not adapt to specific activities or learners, which is one of the core points of this thesis. In fact, TAGARELA's error diagnosis works similar to the pipeline approach in *German Tutor* (Heift and Nicholson, 2001) in the sense that only one error is diagnosed and the computation stopped after that error is found. In a way, TAGARELA shifted the decision tree-like approach of *German Tutor* and other systems from analysis to diagnosis. However, this approach does not permit the detection and accumulation of **multiple** learner errors. Nevertheless, multiple errors do occur in learner input, as discussions such as Heift (2003) show. In addition, determining only one error does not support building a representative learner model: As Heift (2003) shows, multiple learner errors should be captured in order to record as much information as possible about the learner. Otherwise, crucial information that is in fact present in the learner input would be discarded.

To address this problem, we split up the diagnosis module into several submodules. Each is responsible for detecting one or more errors declared in the error taxonomy. Instead of calling them in a predefined order and stopping once an error is found, as TAGARELA does, we call all relevant diagnosis modules and obtain **all** detectable error instances. These are then used to update the error counts in the learner model before any feedback is given.

However, treating multiple errors is not as straightforward as one might think. During the development of our system, we observed that the same linguistic phenomenon in the learner input is often diagnosed multiple times on different levels of linguistic representation. Consider spelling errors as an example. Spelling errors in our system are technically non-words, i.e. words that do not exist in the language. Nevertheless, the diagnosis module responsible for detecting missing concepts in our system can also interpret such cases as missing concepts because the correct surface forms have not been produced. Thus, spelling errors and missing concepts may not be independent because they may point to the same underlying problem. The question of **error interdependencies** is an open issue in ICALL research and there is virtually no discussion of it in relevant literature. In order to describe multiple errors accurately and correctly, future research should take this issue into account.

By making explicit exactly which errors they can detect, diagnosis modules also play a role in guiding the analysis process: They encode the **mapping between detectable error types and needed annotations**. By asking the diagnosis modules, the system can determine which annotations are needed and hence which analysis modules must be called, as we already described in section 4.2.

4.3.2 Feedback

Feedback builds on the results of the diagnosis process. The task of feedback is to pick the error from the diagnosis result that is most important. In recent years (cf. e.g. Heift (2003)) it has been established that, in case of multiple errors, reporting all of them is counterproductive to the learning process. Instead, an ILTS should concentrate on one error for actual feedback even if all errors are recorded in the learner model.

We have already mentioned that TAGARELA uses a scaffolding technique for feedback. That means that the student is led by feedback messages towards the correct answer. The further away the learner is from the correct answer, the less specific the feedback gets (Amaral, 2007, p. 111). The question for our system is which error to choose for feedback in case the learner made more than one error.

As we are trying to make the system more adaptive to learners' and instructors' needs, it follows that errors should be chosen for feedback according to these preferences. The central idea to achieve this is to **rank the errors** according to the error preferences in the activity model (see section 4.1.2) and the error counts in the learner model (see section 4.1.3).

After diagnosis has determined a list of errors, the list is given to the ranking procedure, which is essentially a sorting algorithm. In addition to the list of errors, the sorting algorithm takes two other error lists into account: the targeted error types of the activity and the error types from the learner model, sorted by error count. It then compares the observed errors in the list from the diagnosis module as follows.

For any two errors to be compared:

1. If one of the error types is preferred by the **activity**, prefer that error
2. Otherwise, if one of the error types has a higher error count in the **learner model**, prefer that error
3. Otherwise, fall back to the **default** TAGARELA strategy: Prefer meaning-based errors over form-based errors

The feedback module then returns the most important error according to the comparison strategy presented above.

Heift (2003) uses a similar idea to ours that she calls 'error priority queue': An ordered list of error types determines which errors are reported first. However, it only takes the instructor's preferences into account, not the problems of the learner. Moreover, all errors are reported, only the order of reporting them is changed.

Although our strategy seems reasonable to us, all has not been said and done about error ranking here. Other comparing strategies will be investigated. Another topic for future research is how feedback is phrased: currently TAGARELA gives feedback messages such as "There is an agreement error between the noun and the verb in the sequence *o menina come* ('The girl ate')". Linguistic terminology such as "agreement" cannot be assumed to be common ground, so feedback messages should take the meta-linguistic competence of learners into account.

5 Implementation

The original TAGARELA was written in the Python programming language[1], an interpreted scripting language well-suited for rapid prototyping. It was deployed as a web-based system using mod_python, which enables the use of Python in a standard web server. To be able to use UIMA as a processing architecture, we had to implement our system in the Java programming language[2]. This also involved re-implementing all TAGARELA analysis modules as UIMA annotators. The development effort we put into this was considerable, yet it was necessary in order to build our ILTS on the solid groundwork of a well-tested, enterprise-scale processing framework.

In a Java web application, requests from users are handled by servlets. A servlet accepts the parameters sent by the browser and returns a response. In our case, the analysis servlet receives the learner input and the parameters needed to identify the learner and the activity. After loading the corresponding activity model and learner model, the diagnosis manager is asked about required annotation types. The diagnosis manager responds with an explicit set of annotation types which the servlet uses to load and run an aggregate AE on the learner input. The CAS used for this analysis contains the learner answer and the target answer chosen for comparison. After all needed annotation modules have been run, the annotated CAS is passed to the diagnosis manager along with the activity model and learner model. The diagnosis manager returns a feedback result which is then passed back to the user interface, completing the request. The components in our architecture along with their connections to each other are visualized in figure 5.1.

Section 5.1 presents the type system used for analysis. Section 5.2 then explains how multiple answers can be stored in the same CAS before section 5.3 details the various analysis modules. Section 5.4 very briefly documents the implementation of learner and activity model before section 5.5 finally gives implementation details about diagnosis and feedback.

[1]`http://python.org`

[2]`http://java.sun.com`

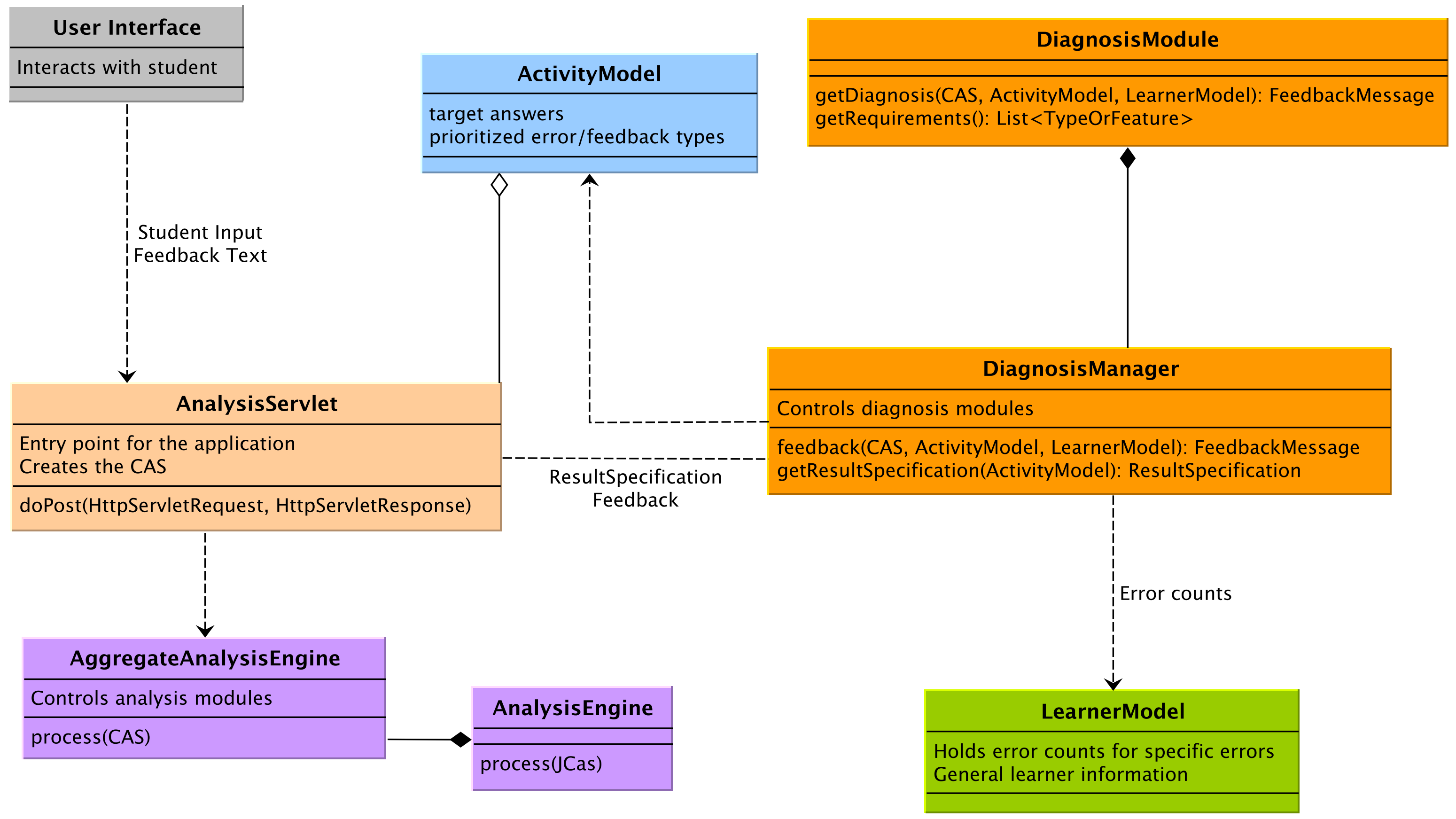

User Interface
Interacts with student
Student Input
Feedback Text
ActivityModel
target answers
prioritized error/feedback types
DiagnosisModule
getDiagnosis(CAS, ActivityModel, LearnerModel): FeedbackMessage
getRequirements(): List<TypeOrFeature>
AnalysisServlet
Entry point for the application
Creates the CAS
doPost(HttpServletRequest, HttpServletResponse)
ResultSpecification
Feedback
DiagnosisManager
Controls diagnosis modules
feedback(CAS, ActivityModel, LearnerModel): FeedbackMessage
getResultSpecification(ActivityModel): ResultSpecification
Error counts
AggregateAnalysisEngine
Controls analysis modules
process(CAS)
AnalysisEngine
process(JCas)
LearnerModel
Holds error counts for specific errors
General learner information

Figure 5.1: A simplified diagram of our architecture. Components marked with the same color work closely together. **AnalysisEngine** and **DiagnosisModule** are example instances of individual analysis engines and diagnosis modules, respectively.

5.1 The Type System

One of the key benefits of a UIMA application is the abstraction created through the type system. As outlined in section 2.2, components interact directly in a classical pipeline architecture, one feeding its output into the next. This creates unnecessary dependencies on particular modules and the data structures they use.

UIMA decouples the modules from each other by forcing a common type system on all modules. If a piece of information should be annotated by a given module, it has to be declared first in the type system and thus made explicit. This prevents an Analysis Engine (AE) from relying on internal data structures used in other AEs. Instead, AEs must convert their internal data representations into instances of types declared in the type system before they can be annotated in the CAS.

As described in Götz and Suhre (2004, p. 480), UIMA types are equivalent to typed feature structures used in formalisms such as HPSG (Pollard and Sag, 1994). A type is either atomic, such as a string or an integer, or a complex data structure. Complex types define a set of feature-value mappings where the value of a given feature is again of a primitive or complex type. Types can derive from other types in which case the subtype inherits all features of the supertype.

To illustrate this concept and at the same time introduce our type system, we will visualize types as attribute-value matrices, as is common in feature-based grammar formalisms. As described in section 4.1.1, our system uses three main types: `Token`, `Phrase` and `AnalysisResults`. `Token` and `Phrase` both derive from `Constituent`, which defines a single string feature 'symbol'. In the case of a `Token`, 'symbol' is the part of speech whereas for `Phrase` it is the category name. Defining `Token` and `Phrase` as subtypes of `Constituent` lets us treat both uniformly if no distinction is needed (as is the case with the children of a `Phrase`, see below) but still allows for access to the specific features of each type through explicit typecasts.

Figure 5.2 shows the `Token` for the pronoun *Ele* ('he'). As `Token` is a subtype of the general annotation type `uima.tcas.Annotation`, it also has the features 'begin' and 'end' which encode the beginning and end of the token. As can be seen, *Ele* in this case ranges from position 0 to position 3. Its 'symbol' feature is set to 'pronoun', reflecting the part of speech.

The other features have complex values. `SpellingAnalysis` is a type used by the spell checker (see section 5.3.3) to indicate whether a token is misspelled and if so, to store a list of suggestions with the token. The 'lexiconDefinitions' feature holds a list of `LexiconInfo` instances, each corresponding to a lexicon entry for this token,

$$
{}_{\textit{Token}}\left[\begin{array}{ll}
\textsc{begin} & 0 \\
\textsc{end} & 3 \\
\textsc{tokenString} & \text{'Ele'} \\
\textsc{symbol} & \text{'pronoun'} \\
\textsc{spellingAnalysis} & {}_{\textit{SpellingAnalysis}}\left[\begin{array}{ll} \textsc{misspelled} & \text{false} \\ \textsc{suggestions} & \left\langle \textsc{null} \right\rangle \end{array}\right] \\
\textsc{lexiconDefinitions} & \left\langle {}_{\textit{LexiconInfo}}\left[\begin{array}{ll} \textsc{person} & \text{'3s'} \\ \textsc{number} & \text{'si'} \\ \textsc{gender} & \text{'m'} \\ \textsc{pos} & \text{'pronoun'} \\ \textsc{canonic} & \text{'ele'} \\ \textsc{frequency} & 35 \\ \textsc{source} & \text{'lexicon'} \end{array}\right] \right\rangle \\
\textsc{deepForm} & \left\langle \textsc{null} \right\rangle
\end{array}\right]
$$

Figure 5.2: The token *Ele* ('he') represented as a typed feature structure

thus allowing for lexical ambiguity. The 'deepForm' feature is used for contractions and encliticizations where the surface form is syntactically complex. Each element of the 'deepForm' list is itself a `Token`. The use of 'deepForm' is described in section 5.3.2 where we discuss the tokenizer.

Figure 5.3 shows the `Phrase` for the NP *menina bonita* ('beautiful girl'). For reasons of space and clarity, features with null values as well as nested types not relevant for illustration have been omitted. `Phrase` again has the features 'begin' and 'end' defining its range over the input. In addition to inheriting the 'symbol' feature from `Constituent`, it defines a children list of type `Constituent`. The elements in this list can be either `Tokens` or `Phrases`. In this case, one is the token *menina* and the other is an AP with the single child token *bonita*.

`AnalysisResults` (see figure 5.4) holds information that pertains to the input as a whole. As such, it is not really an annotation although, for convenience reasons[3], it derives from the `uima.tcas.Annotation` type. The defined features relate to results of individual AEs: 'stringMatch' holds information about the result

[3]Types deriving from `uima.tcas.Annotation` are easier to retrieve because they are stored in the default annotation index.

$${}_{\mathit{Phrase}}\begin{bmatrix}
\textsc{begin} & 0 \\
\textsc{end} & 13 \\
\textsc{symbol} & \text{'NP'} \\
\textsc{children} & \left\langle {}_{\mathit{Token}}\begin{bmatrix}
\textsc{begin} & 0 \\
\textsc{end} & 6 \\
\textsc{symbol} & \text{'noun'} \\
\textsc{spellingAnalysis} & {}_{\mathit{SpellingAnalysis}}\begin{bmatrix} \ldots \end{bmatrix} \\
\textsc{lexiconDefinitions} & \left\langle {}_{\mathit{LexiconInfo}}\begin{bmatrix} \ldots \end{bmatrix} \right\rangle \\
\textsc{tokenString} & \text{'menina'}
\end{bmatrix},
{}_{\mathit{Phrase}}\begin{bmatrix}
\textsc{begin} & 7 \\
\textsc{end} & 13 \\
\textsc{symbol} & \text{'AP'} \\
\textsc{children} & \left\langle {}_{\mathit{Token}}\begin{bmatrix}
\textsc{symbol} & \text{'adj'} \\
\textsc{tokenString} & \text{'bonita'} \\
\ldots &
\end{bmatrix} \right\rangle \\
\textsc{parent} & {}_{\mathit{Phrase}}\begin{bmatrix} \ldots \end{bmatrix}
\end{bmatrix} \right\rangle
\end{bmatrix}$$

Figure 5.3: The phrase *menina bonita* ('beautiful girl') represented as a typed feature structure

$${}_{\mathit{AnalysisResults}}\left[\begin{array}{ll} \textsc{stringMatch} & \text{'EXACT\_MATCH'} \\ \textsc{agreementResults} & \left\langle {}_{\mathit{Agreement}}\left[\ldots\right], {}_{\mathit{Agreement}}\left[\ldots\right] \right\rangle \\ \textsc{tokenMatch} & {}_{\mathit{TokenMatchResult}}\left[\ldots\right] \\ \textsc{canonicMatch} & {}_{\mathit{CanonicMatchResult}}\left[\ldots\right] \\ \textsc{posMatch} & {}_{\mathit{POSMatchResult}}\left[\ldots\right] \end{array}\right]$$

Figure 5.4: The analysis results type

of direct string matching of learner and target answer, 'agreementResults' stores all agreement results found in the input and the other features store the results of shallow semantic analysis. We describe the types of these features more closely in section 5.3.

Finally, it is worth noting that UIMA type systems are declared independently of programming language. An XML type system descriptor is used to describe types and features which is then translated to a Java class hierarchy. In that hierarchy, types are realized as Java classes and their feature values can be obtained from instances of the respective classes. This allows for direct use of the types in Java code without having to write the code for the types oneself.

5.2 Multiple Views for Multiple Answers

As we have mentioned several times throughout the paper, target answers from the activity model are used in many activities as a reference for evaluation of the learner answer. As there are often multiple target answers per question, the system needs to either pick one or analyze all of them. The more answers are to be processed, the higher the computational cost of analysis complexity gets, so we chose to pick one target answer, just as TAGARELA did. Also similar to TAGARELA, we employ a distance measure as a heuristic to find the target answer that is closest to the learner answer. While TAGARELA uses the Python-specific difflib module, which is based on a pattern matching approach described in Ratcliff and Metzener (1988), we use the well-known Levenshtein distance (Levenshtein, 1966). The target answer with the smallest distance to the learner answer is chosen for further comparison.

In order to compare learner answers to target answers with respect to linguistic

Learner Answer				Target Answer			
Ele	se	cham**o**	Carlos.	Ele	se	cham**a**	Carlos.
`Token`	`Token`	`Token`	`Token`	`Token`	`Token`	`Token`	`Token`
NP	VP			NP	VP		
Type System							

Table 5.1: Simplified multi-view CAS

properties, the same linguistic representations must be identified in both. For example, tokens need to be identified in both the learner and the target answer. For the comparison itself, both learner and target answers need to be available, along with their linguistic annotations. Thus, we need to be able to flexibly refer to both answers, analyzing them separately first and contrasting them later. These requirements make it clear that a standard UIMA CAS, which can hold only one body of text, is not enough for our purposes. Creating a separate CAS for each answer would let us annotate both answers separately, but references from one to the other would not be possible. Moreover, the learner and target answer are clearly related: both are answers to the same question and can hence be expected to express the same content. Consequently, they should be in the same data structure if possible.

UIMA provides a solution that involves multiple "views" on the same CAS. A standard CAS holds one document and stores the annotations licensed by the type system in indexes. A multi-view CAS stores multiple documents and uses the same type system for all of them, providing separate indexes for each document. The UIMA user guide shows the usage[4] of multiple views for a text in different languages, for example. The "documents" in our case are the learner and target answers, which we can thus annotate according to the same criteria. For comparisons, we can consult both views and store comparison results using pointers to specific annotations in both views. Section 5.3.8 shows how this is relevant for content assessment.

Table 5.1 shows a simplified multi-view CAS for the sentence *Ele se chama Carlos* ('He is called Carlos.'). Token annotations are made independently in both views using the same type system. In this example, the learner made an agreement error, using the first person *chamo* instead of third person *chama*.

[4]`http://incubator.apache.org/uima/downloads/releaseDocs/2.2.2-incubating/docs/html/tutorials_and_users_guides/tutorials_and_users_guides.html#ugr.tug.mvs`

5.3 Analysis Modules

This section describes the various analysis modules in detail. Section 5.3.1 explains how AEs declare their input and output specifications before the other sections illustrate the implementations and document the improvements over the original TAGARELA implementation.

5.3.1 Input and Output Specifications

Each analysis module in UIMA conforms to the same specification, i.e. it implements the *AnalysisEngine* interface. This makes it possible for the framework to treat all AE the same. However, AEs naturally differ in their tasks and capabilities, which makes it necessary that they specify metadata in descriptor files. The information encoded in these descriptor files includes the type system the AE expects and, most importantly, what the AE expects as input and what output can be expected of it.

As an example, consider our parser AE. The parser needs `Tokens` to operate on. Specifically, it needs `Tokens` that have the features 'symbol' and 'tokenString' set. The tokenizer provides `Tokens` but the 'symbol' feature is set by the disambiguator, which in turn requires the lexicon lookup.

Input and output specifications support the formulation of linear precedence constraints. Such constraints can be used by the analysis controller to dynamically compute a sequence of analysis modules where each module is provided with the needed input annotations. If a particular analysis module provides only annotations that are not needed for a given task, it can be skipped by the controller.

5.3.2 Tokenizer

The tokenizer is arguably the most crucial pre-processing component in the chain. It decomposes the input string into its basic building blocks, usually equivalent to lexemes. All of the following modules make use of token annotations. Just like the original TAGARELA tokenizer, ours can handle contractions and encliticizations. Contractions, already presented in section 3.2.3, occur between prepositions and determiners, among other cases. Example 3 again shows one such case, the contraction *no*.

(3) *em* + *o* = *no*
of + the = of the

$$
{}_{\textit{Token}}\left[\begin{array}{ll}
\textsc{begin} & 0\\
\textsc{end} & 2\\
\textsc{tokenString} & \text{'no'}\\
\textsc{symbol} & \text{'prep'}\\
\textsc{lexiconDefinitions} & \left\langle {}_{\textit{LexiconInfo}}\left[\begin{array}{ll} \textsc{pos} & \text{'prep'}\\ \textsc{canonic} & \text{'no'}\\ \textsc{frequency} & 31\\ \textsc{source} & \text{'lexicon'} \end{array}\right]\right\rangle\\
\textsc{deepForm} & \left\langle \begin{array}{l}
{}_{\textit{Token}}\left[\begin{array}{ll}
\textsc{begin} & 0\\
\textsc{end} & 2\\
\textsc{tokenString} & \text{'em'}\\
\textsc{symbol} & \text{'prep'}\\
\textsc{lexiconDefinitions} & \left\langle {}_{\textit{LexiconInfo}}\left[\begin{array}{ll} \textsc{pos} & \text{'prep'}\\ \textsc{source} & \text{'tokn'} \end{array}\right]\right\rangle
\end{array}\right], \\
{}_{\textit{Token}}\left[\begin{array}{ll}
\textsc{begin} & 0\\
\textsc{end} & 2\\
\textsc{tokenString} & \text{'o'}\\
\textsc{symbol} & \text{'det'}\\
\textsc{lexiconDefinitions} & \left\langle {}_{\textit{LexiconInfo}}\left[\begin{array}{ll} \textsc{pos} & \text{'det'}\\ \textsc{source} & \text{'tokn'} \end{array}\right]\right\rangle
\end{array}\right]
\end{array}\right\rangle
\end{array}\right]
$$

Figure 5.5: The contraction *no* ('of the') represented as a typed feature structure

As explained by Amaral (2007, p. 98), the tokenizer can already annotate part of speech in several cases because otherwise ambiguous surface forms are unambiguous if they are part of a contraction. Amaral (2007) implements this feature but does so by replacing the contracted surface form with the two underlying tokens. However, as discussed in section 3.2.3, this prevents the use of the surface form in feedback later which is confusing for students. In our annotation-based architecture, we are more flexible: instead of replacing the original input string, we annotate the surface `Token` with an underlying form in the feature 'deepForm' (see figure 5.5). 'deepForm' is a list of `Tokens`, each of which can again be associated with lexical entries and other features of the type `Token`. In the case of *no*, the underlying *em* and *o* can be annotated with their respective part of speech, as shown in figure 5.5.

Encliticizations are handled in a similar way. They occur between verbs and pronouns, as shown in example 4. In addition, the enclitic pronoun sometimes undergoes morphological changes, as Amaral (2007, p. 99) describes. In order to capture all this information and associate it with the surface token, we again use 'deepForm' to store the list of underlying `Tokens`. For the case in example 4, this means storing the list *comprar* + *a* as underlying form of *comprá-la*.

(4)	*comprar*	+ *a*	=	*comprá-la*
	buy	+ it	=	buy it

As the atomic forms of underlying tokens in both the contraction case and the encliticization case differ from their representation in the combined form, we store their "stand-alone" form in the feature 'tokenString'. Because underlying tokens use the same type `Token` as surface tokens, this feature is set to the surface token string for ordinary tokens. This introduces a small measure of data redundancy as the surface form can always be extracted using the 'start' and 'end' positions. But it offers the advantage of treating underlying tokens in exactly the same way as surface tokens in later analysis steps: the parser can, for example, use the expanded underlying form for building syntactic trees, extracting their "stand-alone" representations from the 'tokenString' feature.

As far as 'start' and 'end' positions for underlying tokens are concerned, we decided on the following strategy. For contractions, 'start' and 'end' are set to the same values as the surface token. This is done on the grounds that deciding where a token inside a contraction starts and where it ends is a complex morphological question that we cannot answer here. However, it is reasonable to assume that both underlying tokens are in fact represented in the surface form, thus both can be said to range over the same span of input. For encliticizations, we set the 'start' and

$${}_{\textit{Token}}\left[\begin{array}{ll} \textsc{begin} & 0 \\ \textsc{end} & 7 \\ \textsc{tokenString} & \text{'Carloss'} \\ \textsc{spellingAnalysis} & {}_{\textit{SpellingAnalysis}}\left[\begin{array}{ll} \textsc{misspelled} & \text{true} \\ \textsc{suggestions} & \left\langle \textsc{'Carlos', 'Carlos s'} \right\rangle \end{array}\right] \end{array}\right]$$

Figure 5.6: Spelling information for the non-word 'Carloss'

'end' features of the verb part to the string before the hyphen and the ones for the pronoun part to the string after the hyphen. We believe this is a reasonable strategy because encliticizations are already orthographically marked by a hyphen and thus provide a clear indication of where their component tokens start and end.

5.3.3 Spell Checker

The spell checker is based on the state-of-the-art solution Hunspell[5], used in notable projects such as the office suite OpenOffice.org or the popular Firefox browser by Mozilla. Hunspell improves over the older Ispell[6] program used in TAGARELA in that it captures morphological peculiarities of natural languages better and offers a superior suggestion mechanism. Ispell merely uses the Damerau-Levenshtein distance (Damerau, 1964) to identify orthographically close words and report them as suggestions. Hunspell, on the other hand, models derivational processes such as compounding and moreover supports the use of Unicode, thus facilitating the creation of dictionaries in various languages.

The task of the spell checker in our system, as in the original TAGARELA, is to identify non-words, i.e. words that are not valid lexemes of Portuguese. In addition to identification, suggestions have to be stored in case a word is misspelled. The type `SpellingAnalysis` is used to hold spelling information, see figure 5.6 for an example. Hunspell is integrated using a Java Application Progamming Interface (API) implemented by Flemming Frandsen[7] and called for each input token identified by the tokenizer. We use the standard Hunspell dictionary for

[5] `http://hunspell.sourceforge.net`

[6] `http://www.lasr.cs.ucla.edu/geoff/ispell.html`

[7] `http://dren.dk/hunspell.html`

$$ {}_{\textit{LexiconInfo}}\left[\begin{array}{ll} \textsc{number} & \text{'si'} \\ \textsc{gender} & \text{'m'} \\ \textsc{pos} & \text{'propNoun'} \\ \textsc{canonic} & \text{'carlos'} \\ \textsc{frequency} & 2 \\ \textsc{source} & \text{'lexicon'} \end{array}\right] $$

Figure 5.7: The lexicon entry for the proper noun *Carlos*

Brazilian Portuguese created by the BrOffice project[8]. In case Hunspell finds the word misspelled, it is additionally checked against the lexicon (see next section) because cases exist where the BrOffice dictionary does not contain a word that is nevertheless present in our lexicon. This is also done in TAGARELA, see Amaral (2007, p. 100/101).

5.3.4 Lexicon Lookup

In the absence of a POS tagger, TAGARELA uses a full-form lexicon to look up part of speech and other lexical information for surface tokens. The lexicon used is based on the lexicon of CURUPIRA (Martins et al., 2003), a general-purpose Portuguese parser by the NILC group[9]. The lexicon used by TAGARELA is a subset of the NILC lexicon, with its most frequent nouns, verbs, adjectives and adverbs (see Amaral (2007, p. 101) for details), along with the verb forms used in the Portuguese course TAGARELA was made for. It contains approximately 43000 entries.

We used the same lexicon in our implementation. The lexicon lookup checks whether a token already has its part of speech annotated by the tokenizer, and adapts its annotation strategy accordingly. If part of speech has been annotated, the corresponding lexical entry is chosen from the set of available lexical entries for a token. If not, all available lexical entries are annotated. This strategy is also inspired by the original TAGARELA module. For representation, the lexicon module uses the `LexiconInfo` type, an example of which is given in figure 5.7.

Aside from the part of speech and morphological features, `LexiconInfo` stores the relative frequency of the respective lexical analysis as observed in the NILC

[8] `http://www.broffice.org/verortografico`

[9] Núcleo Interinstitucional de Lingüística Computacional, see `http://www.nilc.icmc.usp.br/nilc/index.html`

Eu	*o*	*entreguei*	*a*	*João.*
`=pronoun`	`pronoun`	`=verb`		
I	it	delivered	to	John

Table 5.2: A disambiguation rule

corpus. Moreover, the canonic form, i.e. the lemma of the word form, is available. The 'source' feature indicates whether a given entry was annotated by the tokenizer or the lexicon lookup. Making lexicon information explicit in the type system enables switching to other knowledge resources: because we do not rely on internal data structures of a particular lexicon, it can be exchanged for another one that encodes the same information in a different way.

5.3.5 Disambiguator

As explained in the previous section, the lexicon lookup does not deal with lexical ambiguity, instead annotating all possible lexical entries for a given surface form. The disambiguator takes care of ambiguous forms by applying a set of local disambiguation rules, similar to Constraint Grammar (Karlsson, 1990).

Local disambiguation rules essentially work by specifying three parts: a left context, the part of speech to choose, and a right context. The rule applies if both contexts are satisfied. In case the rule applies, the lexical entry corresponding to the middle part of the rule is chosen and the other `LexiconInfo` instances are removed from the token. Contexts are generally specified as parts of speech. In TAGARELA, the rules are encoded in the form of conditional statements in the programming language itself. This makes them hard to extend and sometimes also difficult to understand.

We improved that by creating a general rule application mechanism. A rule is specified using a simple syntax that involves the POS to be chosen and the contexts, as shown in table 5.2. The word *o* can be either a determiner, a pronoun or a noun (Amaral, 2007, p. 104). But if it occurs between another pronoun and a verb, as is the case here, it must be a pronoun. Our rule mechanism tests the contexts, and if they are satisfied, applies the rules by reducing the `LexiconInfo` instances of the respective token to one. Rules can also be read from files, making the approach fully generic by putting grammar information outside the code.

If the rules do not apply for a given ambiguous token, the analysis with the highest 'frequency' value (based on the NILC corpus) in `LexiconInfo` is chosen.

Thus, statistical information is used if rule-based disambiguation fails. In order to make local disambiguation more flexible, we plan to base a future version of the disambiguator on the Constraint Grammar compiler VISLCG3[10], developed by Tino Didriksen and Eckhard Bick. VISLCG3 has already seen use in ICALL settings (see e.g. Bick (2005a,b)) and offers the full expressive power of Constraint Grammar.

5.3.6 Parser

Our parser[11] is an implementation of the Cocke-Younger-Kasami (CYK) algorithm, as in the original TAGARELA. Its task is to identify all well-formed subtrees that are licensed by the grammar for further analysis and diagnosis. The grammar is a context-free grammar in Chomsky Normal Form and thus allows only binary branching. It does not contain mal-rules[12] and thus licenses only well-formed substrings of Portuguese. The parser does not check features, thus agreement errors are not detected by it. Instead they are handled by the agreement checker discussed in the next section. Parsing with a context-free grammar has the advantage of being fast and providing a backbone for further analysis on trees. A similar approach is used in Lexical Functional Grammar (LFG) (Kaplan and Bresnan, 1983) where the "c-structure" represents the context-free backbone and the "f-structure" encodes grammatical functions pertaining to the "c-structure".

Parse trees are represented by instances of the type `Phrase`. As already described in section 5.1, `Phrases` have pointers to child constituents and parent `Phrases`. In contrast to the original TAGARELA implementation, our parser transforms the full chart into annotations. TAGARELA's parser, on the other hand, returns only the tree that covers the biggest input span.

In addition, our parser makes use of UIMA's annotation capabilities to annotate multiple possible analyses. Annotations ranging over the same input span are not a problem for UIMA, because all of them can refer to the input via the 'start' and 'end' positions. This enables us to put as many `Phrase` instances into the annotation index as there are well-formed subtrees in the CYK chart.

Representation of multiple analyses is necessary because our parser is not probabilistic and thus does not choose one particular analysis. This may seem like a drawback, but on the other hand, storing multiple analyses makes it possible for later modules to choose an analysis, possibly based on external disambiguation

[10]`http://beta.visl.sdu.dk/cg3.html`

[11]We are indebted to Bettina Demmer, Kristiina Muhonen and Niels Ott for the implementation basis.

[12]Mal-rules model ungrammatical constructions, see Weischedel and Sondheimer (1983).

information[13], and traverse parse trees top-down or bottom-up. The combination of bottom-up parsing strategy and annotation of all well-formed subtrees makes sure that no information is discarded, which provides a good basis for diagnosis.

5.3.7 Agreement Checker

As stated in the previous section, agreement checking is handled by a separate module. As agreement is a common learner error, this is an important task. The original TAGARELA uses ad-hoc functions defined in the programming language itself to check agreement, similar to the original approach in the disambiguator. Rules express constraints such as "If an NP has two daughters and both are tokens, they must agree in number and gender". Similar rules exist for subject-verb agreement and more complex noun phrases, such as coordinated ones.

These constraints can be seen as global constraints specific to the phrase label that they apply on, essentially doing pattern-matching on trees and subtrees. This contrasts with the local constraints implemented in formalisms such as LFG (Kaplan and Bresnan, 1983) or HPSG (Pollard and Sag, 1994). In HPSG, for example, the head-feature principle enforces agreement between the head daughter of a phrase and the phrase itself. Features are thus propagated from head daughter to phrase which makes specific global constraints unnecessary.

Although we followed the general idea of the TAGARELA agreement checker, we improved it in two ways. First, agreement constraints are represented as Java classes that derive from a common supertype `Rule`. This makes adding a new rule straightforward as it corresponds to creating a new class. Second, `Rule`s all use the same mechanism to compute agreement instead of re-implementing the same task over and over again in every rule.

Results of agreement checking are stored using the `Agreement` type, see figure 5.8 for an example. Besides the resulting feature values for gender, number and person, `Agreement` holds references to the two constituents involved if there was an agreement error. This enables easy retrieval of the constituents for feedback. Agreement results are stored globally in the `AnalysisResults` type.

A future implementation of the agreement checker could build on the Tregex query engine (Levy and Andrew, 2006). That would improve the speed of the implementation and reduce the amount of code needed to check tree constraints. However, we will consider replacing the parser and the agreement checker by a

[13]For example, an already annotated unambiguous target answer.

$${}_{\textit{Agreement}}\left[\begin{array}{ll}
\textsc{label} & \text{'subj-verb'} \\
\textsc{person} & \text{'error'} \\
\textsc{number} & \text{null} \\
\textsc{gender} & \text{null} \\
\textsc{offendingConstituents} & \left\langle {}_{\textit{Phrase}}\left[\begin{array}{ll} \textsc{begin} & 0 \\ \textsc{end} & 3 \\ \textsc{symbol} & \text{'NP'} \\ \ldots & \end{array}\right], {}_{\textit{Token}}\left[\begin{array}{ll} \textsc{begin} & 7 \\ \textsc{end} & 12 \\ \textsc{symbol} & \text{'verb'} \\ \ldots & \end{array}\right] \right\rangle
\end{array}\right]$$

Figure 5.8: Subject-verb agreement error in *Ele se chamo Carlos* ('He is called Carlos.')

parser that can handle complex categories. Other systems, such as *E-Tutor* (Heift and Nicholson, 2001), also use parsers for feature-based grammar formalisms.

5.3.8 Shallow Semantic Matching

The shallow semantic matching modules represent the content-based part of the linguistic analysis. As TAGARELA, our system follows the approach presented by Bailey and Meurers (2008), where the idea is to match the learner answer to the target answer on multiple levels of linguistic representation. The approach is inspired by techniques in Machine Translation Evaluation, particularly the METEOR metric (Banerjee and Lavie, 2005).

The levels of representation we use are tokens, canonic forms and POS. If a unit is not matched on the token level, we abstract to the canonic form, and then to the POS if necessary. Whenever some unit has been matched, it is not matched again on the more abstract levels. The matching algorithm itself is not specific to a particular representation: it works in the same way on tokens, canonic forms and POS. Thus, we decided to isolate the algorithm in a separate class and use it from three different analysis modules corresponding to the three representation levels. This has the advantage of extensibility: adding another level (e.g. synonyms) is easy because the same algorithm can be reused again.

The matching algorithm itself is relatively simple. It traverses both answers and computes mappings from the units in one answer to the units in the other answer. In a sentence such as the one in table 5.3, *chamo* ('call'-1.SG) in the learner answer is matched to *chama* ('call'-3.SG) on the canonic level whereas all other units

Target Answer	*Ele*	*se*	*chama*	*Carlos*
Learner Answer	*Ele*	*se*	*chamo*	*Carlos*
	`Token`	`Token`	`Canonic`	`Token`

Table 5.3: A content matching example

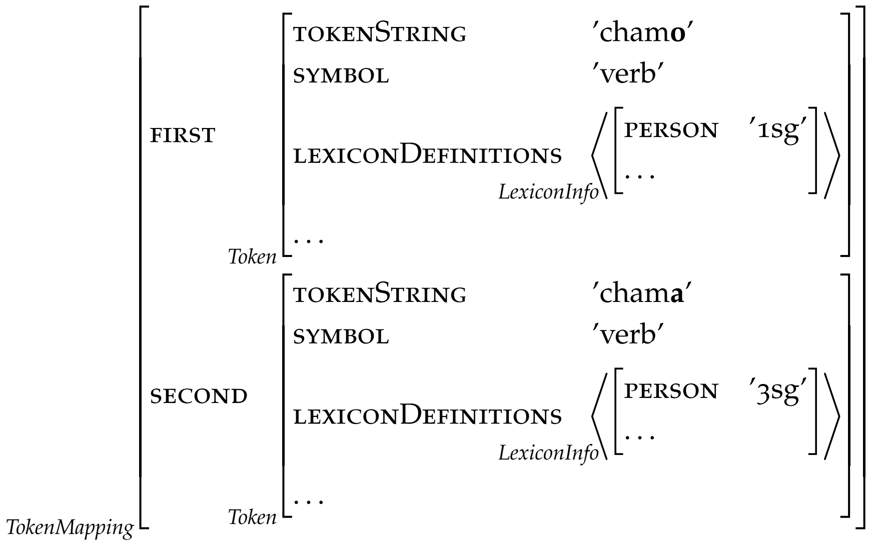


Figure 5.9: A token mapping

are matched on the token level. In case there are multiple possible mappings[14], mappings to units that have similar positions in the target answer are preferred.

The results of matches are annotated globally in the `AnalysisResults` type. Both successful and unsuccessful matches are recorded for each representation level. For successful matches, pointers to the respective units in both answers are stored, which makes retrieval for feedback easy: to report a form error such as the one in table 5.3, a diagnosis module can retrieve the mapping on the canonic level and generate a message such as "Try to change the form of 'chamo'.". Figure 5.9 shows a mapping between two tokens.

In the future, we plan to include matching based on synonyms, based on lexical-semantic resource such as a wordnet. This would enable us to close the representational gap between canonic forms and entire word classes.

[14]This is the case if a given word occurs multiple times in the same answer.

5.4 Learner Model and Activity Model

The learner model and the activity model are primarily repositories of data, not software components. The main choice we had to make was how to organize the data.

For the learner model, we adopted TAGARELA's choice of a simple key-value database[15]. It stores the learner ID, the last input string for each activity and the error counts. For the error counts, the error type is the key and the count is the value.

The activity model is realized as an XML file for each activity. XML's hierarchical structure enables us to organize activity information in a well-structured way and keep presentational information apart from processing-related information.

5.5 Diagnosis and Feedback

As stated in section 4.3.1, diagnosis in the original TAGARELA is a single module that tests the findings of the analysis modules and stops at the first error found. We diverted from this approach in that the task of diagnosis is split up into modules, each responsible for detecting a group of error types and returning a feedback result. A diagnosis module declares two important pieces of information: the types of errors it can detect and the annotation types it needs to detect these errors.

A central diagnosis manager controls the individual modules in a generic way. When prompted for information about required annotation types by the analysis controller, the diagnosis manager obtains the annotation types from the diagnosis modules and collects them in a UIMA `ResultSpecification`. A `ResultSpecification` is a collection of types and features that can be passed to analysis components in order to tell them what types and features should be annotated. This fits in nicely with what we need: our aim is to control analysis through feedback requirements and `ResultSpecifications` provide a mechanism for guiding analysis.

Error types are represented as global constants belonging to a certain class. As explained in section 4.3.2, we compare error instances found during diagnosis according to criteria from the activity model and learner model in order to decide which error is to be used for feedback. By representing error types as objects, we

[15] `http://www.oracle.com/technology/products/berkeley-db/db/index.html`

are able to implement a comparison strategy using standard Java facilities[16]. We are also able to easily switch to another comparison strategy if that should prove necessary. Once we can draw on more extensive information in the activity model and learner model, for example, the comparison strategy will have to be adapted.

[16]`http://java.sun.com/javase/6/docs/api/java/util/Comparator.html`

6 Evaluation

In order to really assess the benefits of our system compared to others such as the original TAGARELA, we would have to run tests with a sufficient number of learners in both systems. There would have to be a control group using the original system and a test group using our system and we would have to find out whether our system really does have a positive effect on second language acquisition.

Besides the methodological questions, such a quantitative analysis requires resources in terms of time and people that are clearly out of the scope of a Master's thesis. We therefore chose not to tackle this problem in the time we had. We are not alone in this matter: most sophisticated ICALL systems are never evaluated properly. An example is the MIRTO project (Antoniadis et al., 2004), in which considerable effort was put into a coherent design for different kinds of users (NLP specialists, instructors, students). Unfortunately, MIRTO was never tested in a real learning environment. This applies to many other ICALL systems, with the exception of Trude Heift's systems: Heift worked on the foundation of *German Tutor* and *E-Tutor* for years before finishing her PhD in 1998 (see Heift (1998)). As a result, she was able to publish SLA studies based on the systems in recent years, such as Heift (2001, 2003, 2004). Clearly, ICALL systems involve a significant development overhead that has to be dealt with before any meaningful quantitative evaluation can be carried out.

For these reasons, we showcase the benefits of the different improvements in our system through concrete use cases in section 6.1 instead of aiming at a quantitative evaluation of the full system. A quantitative evaluation of the lexical and grammatical resources in TAGARELA is presented in section 6.2, along with a discussion of the problems. Finally, we note other observations and improvements in section 6.3.

Although we cannot comprehensibly evaluate the system in this thesis, it will be tested in the near future. Concrete plans involve collaboration with Luiz Amaral at the University of Massachusetts Amherst, who will use our system for Portuguese language courses in September 2009.

6.1 Qualitative Evaluation: Example Cases

This section illustrates the improvements of our system over the original TAGARELA by means of concrete example cases. Three areas are presented: annotation-based processing (section 6.1.1), activity model (section 6.1.2) and learner model (section 6.1.3).

6.1.1 Annotation-Based Processing

As detailed before in section 3.2.3, annotation-based processing enables the use of different representations for analysis and feedback. We will illustrate this advantage here with two error cases: a capitalization error and an agreement error where a contraction is involved.

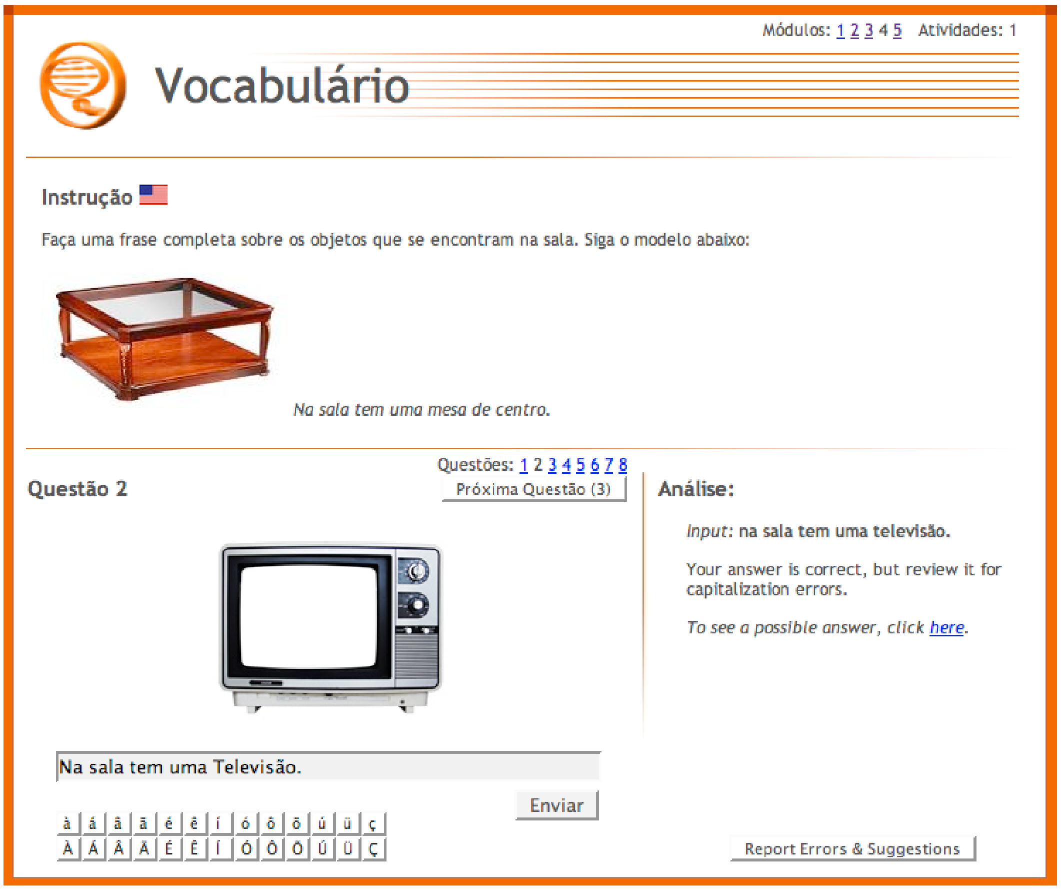
Case 1: Capitalization

The first case we want to illustrate concerns capitalization and is not discussed in Amaral and Meurers (2009) as it is rather simple. Nevertheless, it presents another problem that can be easily solved by using annotation-based processing, as we show below.

Figure 6.1 shows a vocabulary activity in the original TAGARELA system. The task is to describe objects one has in the living room, using the example sentence as guidance. The learner entered the sentence *Na sala tem uma Televisão* ('In the living room we have a TV set'). The input is correct, except that *Televisão* should not be capitalized. The system correctly answers with a message about capitalization errors. However, it reports the learner's input in lower-case letters **only**.

Naturally, this is confusing for the learner because the input shown by the system differs from what he entered. Moreover, the learner is likely unsure what representation the system refers to in the feedback message: the sentence he really typed in or the input reported by the system. Did the system make a mistake or was it his fault?

The problem is that the system reported exactly the representation of the input that it used for analysis. In order to check whether capitalization errors are present, the string matcher converted both learner and target answer into their lower-case versions. A comparison of the two lower-case versions yields a string match, and the system reports the capitalization error, referring to the lower-cased version because the original one is no longer available.


Módulos: 1 2 3 4 5   Atividades: 1

Vocabulário

Instrução

Faça uma frase completa sobre os objetos que se encontram na sala. Siga o modelo abaixo:

Na sala tem uma mesa de centro.

Questões: 1 2 3 4 5 6 7 8

Próxima Questão (3)

Questão 2

Na sala tem uma Televisão.

Análise:

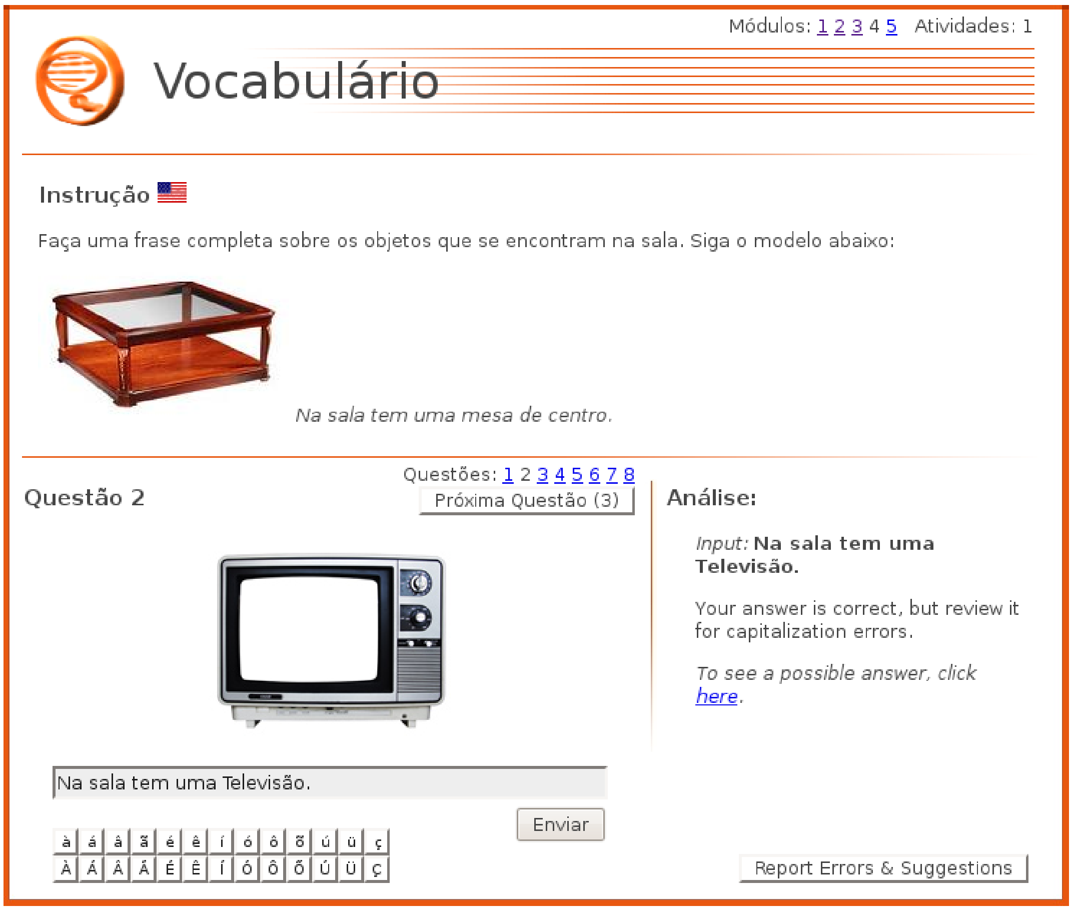
Input: na sala tem uma televisão.

Your answer is correct, but review it for capitalization errors.

To see a possible answer, click here.

Enviar
à á â ã é ê í ó ô õ ú ü ç
À Á Â Ã É Ê Í Ó Ô Õ Ú Ü Ç
Report Errors & Suggestions

Figure 6.1: Problematic feedback on a capitalization error (TAGARELA)


Módulos: 1 2 3 4 5 Atividades: 1

Vocabulário

Instrução

Faça uma frase completa sobre os objetos que se encontram na sala. Siga o modelo abaixo:

Na sala tem uma mesa de centro.

Questões: 1 2 3 4 5 6 7 8

Questão 2

Próxima Questão (3)

Análise:

Input: **Na sala tem uma Televisão.**

Your answer is correct, but review it for capitalization errors.

To see a possible answer, click here.

Na sala tem uma Televisão.

Enviar
à á â ã é ê í ó ô õ ú ü ç
À Á Â Ã É Ê Í Ó Ô Õ Ú Ü Ç
Report Errors & Suggestions

Figure 6.2: Correct feedback on a capitalization error (our system)

Annotation-based processing can remedy this problem without any further work necessary. By keeping the input string to be analyzed separate from the result of string matching, our system can report exactly what the learner typed in while using the lower-cased representation as the basis for the feedback message (see figure 6.2). As described in section 5.1, string matching results are stored in the global `AnalysisResults` structure, keeping the original input untouched. As a consequence, the learner is helped in spotting the mistakenly capitalized *Televisão* and can enter the correct *televisão* instead.

Case 2: Contraction

We already discussed the contraction case in previous sections. Recall that contractions are syntactically complex, consisting of two underlying tokens.

Figure 6.3 presents one of TAGARELA's reading comprehension activities. The text is about the different regions of Brazil and the student has to answer different questions after reading the text. The question in the screenshot can be translated as 'Where is the Amazonas river?'. The learner entered *O Amazonas fica no região norte* ('The Amazonas is in the Northern region'). The system reports an agreement error in the sequence *o região norte*. This is confusing because the learner did not type any *o*. Rather, *o* belongs to the system's internal representation *em o* of the contraction *no* (see section 3.2.3).

Our system handles this case in a better way. Agreement errors are annotated with reference to the syntactic constituents that do not agree (see section 5.3.7). Moreover, complex surface tokens such as contractions are associated with their underlying form (see section 5.3.2). When generating the feedback message, we refer to the surface form using position information from the underlying forms. The underlying form *o* has the same starting position as *no*. Thus, the feedback message in our system correctly reports the sequence *no região norte* (see figure 6.4).

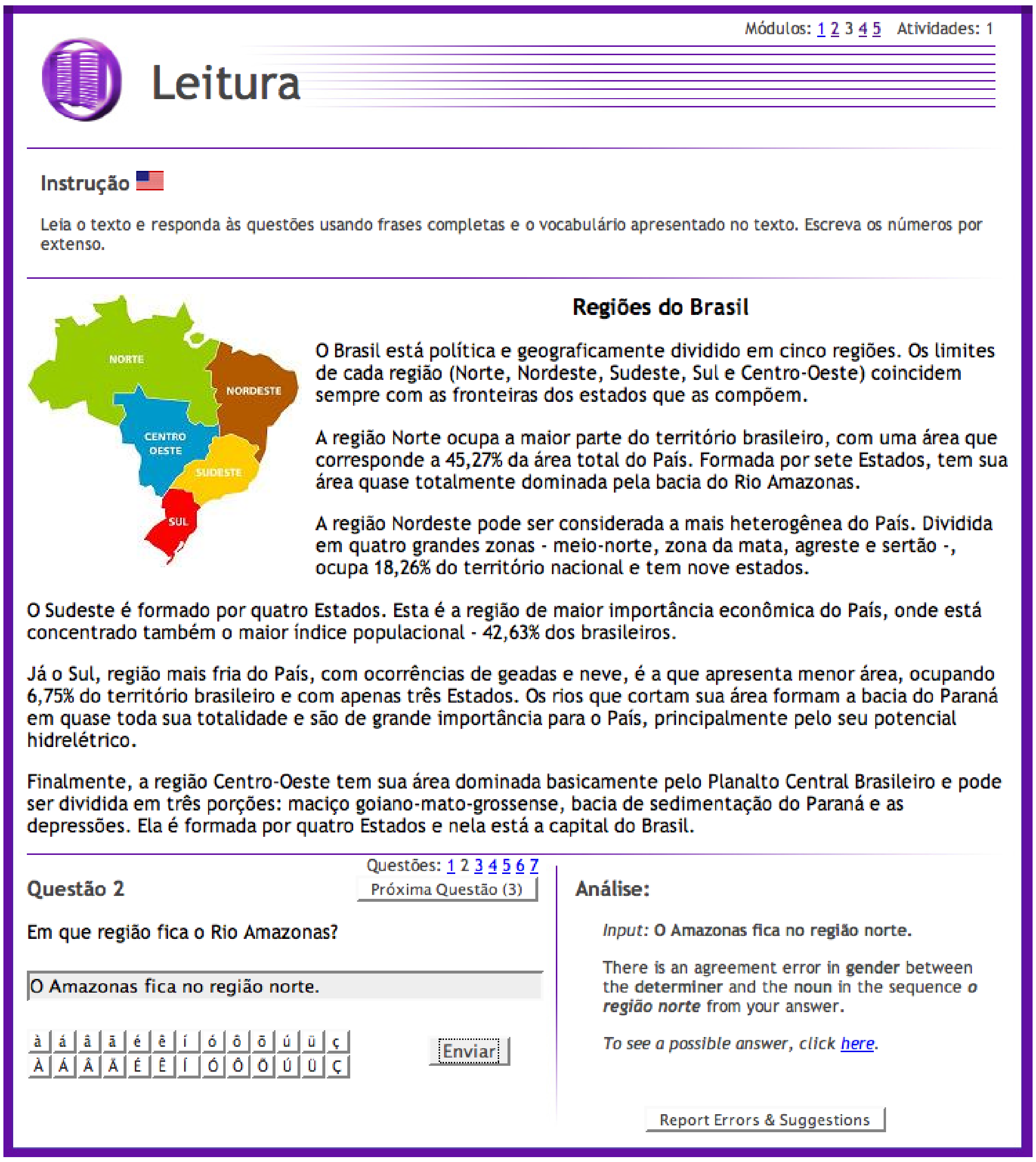
6.1.2 Activity Model

In section 4.1.2, we noted a main difference of our activity model compared to the one used in the TAGARELA system: the use of error types that an activity focuses on. We shall show here how this pedagogical decision can influence the feedback process by demonstrating two specific cases.

Case 1: Reading Comprehension

The first case is drawn from TAGARELA's reading comprehension activities. For this type of activity, the general TAGARELA feedback strategy of prioritizing meaning over form makes sense: reading comprehension questions aim to test the learner's understanding of a given text. They are not designed to test certain grammatical constructions or morphological specifics.

In order to explicitly encode this decision, we specified `MeaningError` in the activity model for reading comprehension activities, which denotes the class of all meaning errors. The system internally translates this to the actual meaning-related errors, such as missing or extra concepts.



Módulos: 1 2 3 4 5 Atividades: 1

Leitura

Instrução

Leia o texto e responda às questões usando frases completas e o vocabulário apresentado no texto. Escreva os números por extenso.

Regiões do Brasil

O Brasil está política e geograficamente dividido em cinco regiões. Os limites de cada região (Norte, Nordeste, Sudeste, Sul e Centro-Oeste) coincidem sempre com as fronteiras dos estados que as compõem.

A região Norte ocupa a maior parte do território brasileiro, com uma área que corresponde a 45,27% da área total do País. Formada por sete Estados, tem sua área quase totalmente dominada pela bacia do Rio Amazonas.

A região Nordeste pode ser considerada a mais heterogênea do País. Dividida em quatro grandes zonas - meio-norte, zona da mata, agreste e sertão -, ocupa 18,26% do território nacional e tem nove estados.

O Sudeste é formado por quatro Estados. Esta é a região de maior importância econômica do País, onde está concentrado também o maior índice populacional - 42,63% dos brasileiros.

Já o Sul, região mais fria do País, com ocorrências de geadas e neve, é a que apresenta menor área, ocupando 6,75% do território brasileiro e com apenas três Estados. Os rios que cortam sua área formam a bacia do Paraná em quase toda sua totalidade e são de grande importância para o País, principalmente pelo seu potencial hidrelétrico.

Finalmente, a região Centro-Oeste tem sua área dominada basicamente pelo Planalto Central Brasileiro e pode ser dividida em três porções: maciço goiano-mato-grossense, bacia de sedimentação do Paraná e as depressões. Ela é formada por quatro Estados e nela está a capital do Brasil.

Questões: 1 2 3 4 5 6 7

Questão 2

Próxima Questão (3)

Em que região fica o Rio Amazonas?

O Amazonas fica no região norte.

à á â ã é ê í ó ô õ ú ü ç
À Á Â Ã É Ê Í Ó Ô Õ Ú Ü Ç

Enviar

Análise:

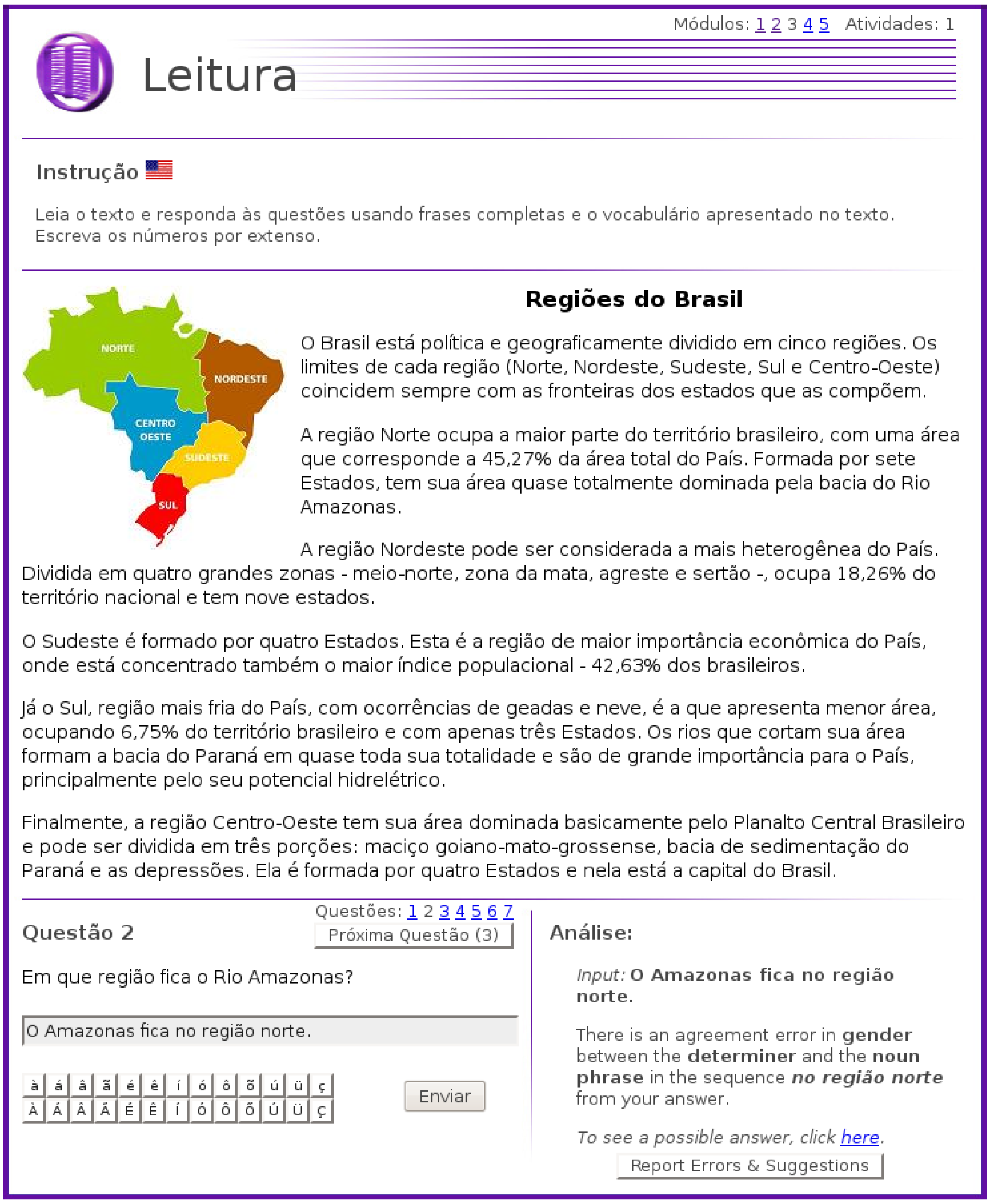
Input: O Amazonas fica no região norte.

There is an agreement error in **gender** between the **determiner** and the **noun** in the sequence ***o região norte*** from your answer.

To see a possible answer, click here.

Report Errors & Suggestions

Figure 6.3: Problematic feedback on an agreement error (TAGARELA)



Instrução

Leia o texto e responda às questões usando frases completas e o vocabulário apresentado no texto. Escreva os números por extenso.

Regiões do Brasil

O Brasil está política e geograficamente dividido em cinco regiões. Os limites de cada região (Norte, Nordeste, Sudeste, Sul e Centro-Oeste) coincidem sempre com as fronteiras dos estados que as compõem.

A região Norte ocupa a maior parte do território brasileiro, com uma área que corresponde a 45,27% da área total do País. Formada por sete Estados, tem sua área quase totalmente dominada pela bacia do Rio Amazonas.

A região Nordeste pode ser considerada a mais heterogênea do País. Dividida em quatro grandes zonas - meio-norte, zona da mata, agreste e sertão -, ocupa 18,26% do território nacional e tem nove estados.

O Sudeste é formado por quatro Estados. Esta é a região de maior importância econômica do País, onde está concentrado também o maior índice populacional - 42,63% dos brasileiros.

Já o Sul, região mais fria do País, com ocorrências de geadas e neve, é a que apresenta menor área, ocupando 6,75% do território brasileiro e com apenas três Estados. Os rios que cortam sua área formam a bacia do Paraná em quase toda sua totalidade e são de grande importância para o País, principalmente pelo seu potencial hidrelétrico.

Finalmente, a região Centro-Oeste tem sua área dominada basicamente pelo Planalto Central Brasileiro e pode ser dividida em três porções: maciço goiano-mato-grossense, bacia de sedimentação do Paraná e as depressões. Ela é formada por quatro Estados e nela está a capital do Brasil.

Questão 2

Em que região fica o Rio Amazonas?

O Amazonas fica no região norte.

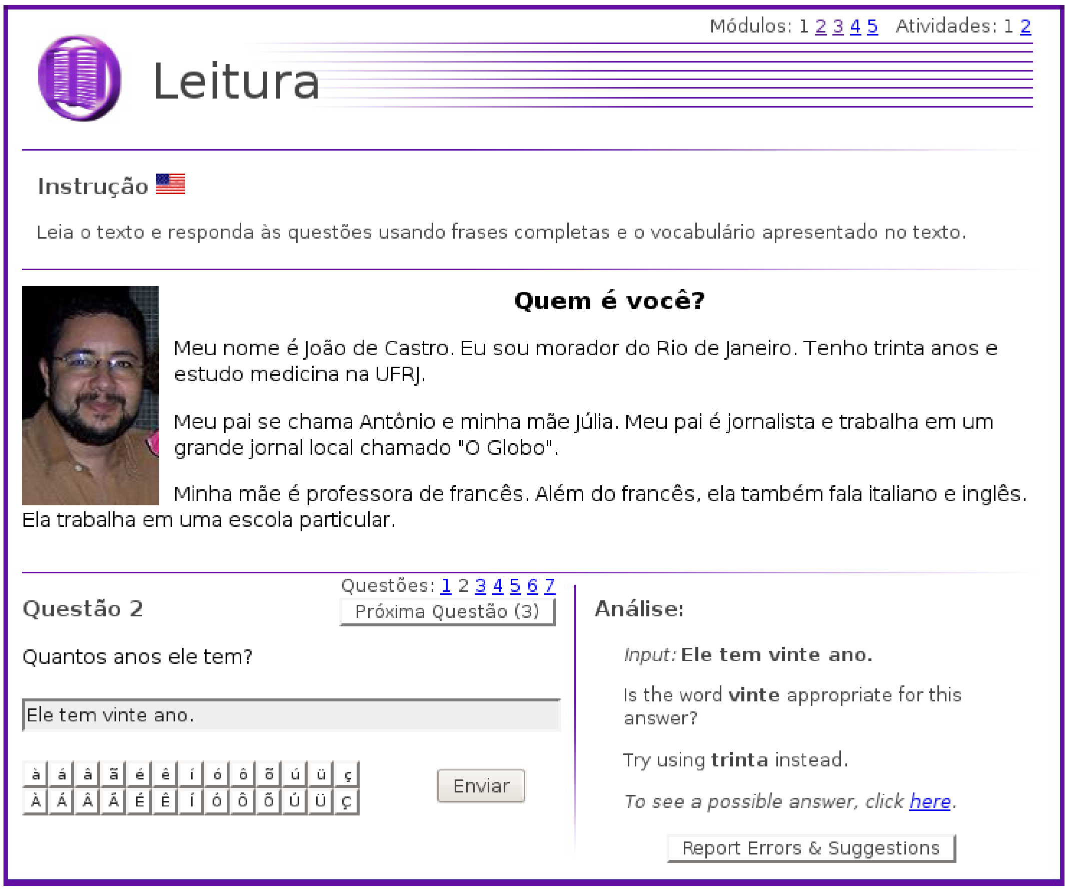
Análise:

Input: **O Amazonas fica no região norte.**

There is an agreement error in **gender** between the **determiner** and the **noun phrase** in the sequence ***no região norte*** from your answer.

To see a possible answer, click here.

Figure 6.4: Correct feedback on an agreement error (our system)


Módulos: 1 2 3 4 5 Atividades: 1 2

Leitura

Instrução

Leia o texto e responda às questões usando frases completas e o vocabulário apresentado no texto.

Quem é você?

Meu nome é João de Castro. Eu sou morador do Rio de Janeiro. Tenho trinta anos e estudo medicina na UFRJ.

Meu pai se chama Antônio e minha mãe Júlia. Meu pai é jornalista e trabalha em um grande jornal local chamado "O Globo".

Minha mãe é professora de francês. Além do francês, ela também fala italiano e inglês. Ela trabalha em uma escola particular.

Questões: 1 2 3 4 5 6 7

Questão 2

Próxima Questão (3)

Quantos anos ele tem?

Ele tem vinte ano.

à á â ã é ê í ó ô õ ú ü ç
À Á Â Ã É Ê Í Ó Ô Õ Ú Ü Ç

Enviar

Análise:

Input: **Ele tem vinte ano.**

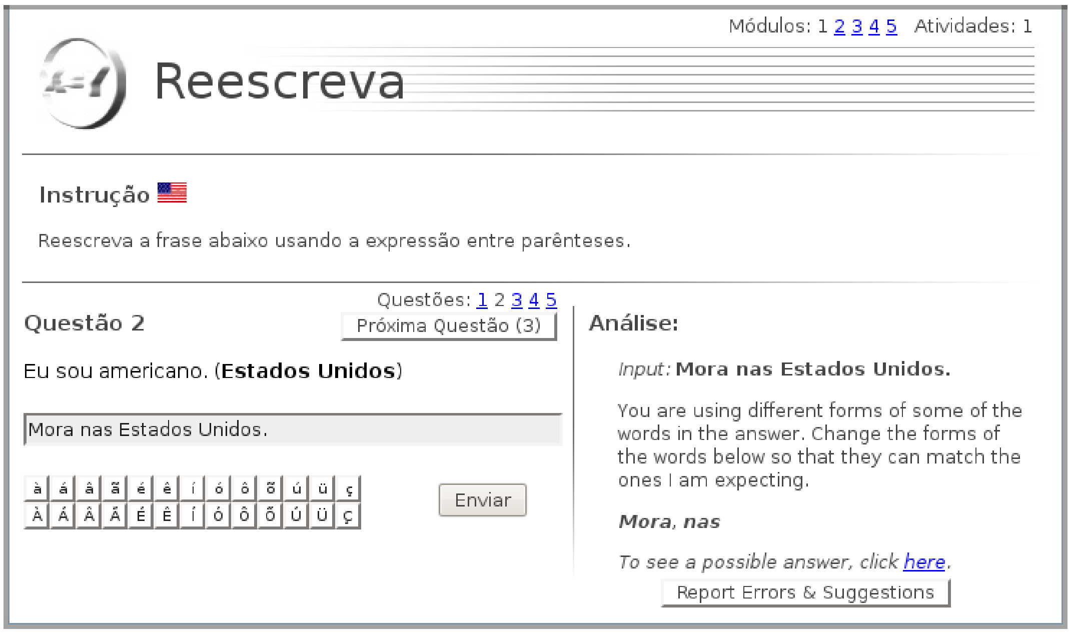
Is the word **vinte** appropriate for this answer?

Try using **trinta** instead.

To see a possible answer, click here.

Report Errors & Suggestions

Figure 6.5: Meaning-based feedback for Reading Comprehension


Módulos: 1 2 3 4 5 Atividades: 1

Reescreva

Instrução

Reescreva a frase abaixo usando a expressão entre parênteses.

Questões: 1 2 3 4 5

Questão 2

Próxima Questão (3)

Eu sou americano. (**Estados Unidos**)

Mora nas Estados Unidos.

Enviar

Análise:

Input: **Mora nas Estados Unidos.**

You are using different forms of some of the words in the answer. Change the forms of the words below so that they can match the ones I am expecting.

Mora, nas

To see a possible answer, click here.

Report Errors & Suggestions

Figure 6.6: Form-based feedback for Rephrasing

Figure 6.5 shows a reading comprehension activity. In the text, a man named João presents himself. The question shown in the screenshot can be translated as "How old is he?", where "he" refers to João. The learner entered *Ele tem vinte ano* ('He is twenty years old'), but made several errors. First, *ano* is in singular form where it should be plural in order to agree with *vinte* in number. Second, according to the text, João is thirty years old and not twenty, as the learner wrote. Both errors are diagnosed by the system and saved to the learner model.

The agreement error is a form-based error whereas the wrong numeral is a meaning-based error. Because we targeted meaning-based errors in the activity model, the system gives feedback on the inappropriate numeral.

Case 2: Rephrasing

The second case we want to demonstrate comes from a Rephrasing activity. In contrast to Reading Comprehension, Rephrasing activities are aimed at helping the learner with acquiring certain grammatical constructions. The task of Rephrasing is to express the given content in a different way. Thus, Rephrasing can be said to stress form instead of content, a fact that we encoded into the activity model by specifying `FormError` there.

Figure 6.6 shows such a Rephrasing activity where the learner is asked to rephrase the sentence *Eu sou americano* ('I am American'). The learner is to use the words *Estados Unidos* ('United States') for the rephrased answer. The answer entered by the learner is *Mora nas Estados Unidos* ('Live in the United States'), which again contains multiple errors. First, the learner failed to enter the pronoun *Eu* ('I'). Second, both *Mora* ('live') and *nas* ('in the') are in incorrect forms: *Mora* (3.SG) should be *Moro* (1.SG) and *nas* (FEM.PL) should be *nos* (MASC.PL). Again, we have a form-based error and a meaning-based error.

This time the system picks the form-based error because we stressed this aspect in the activity model. This differs from TAGARELA's reaction, which would be to prefer meaning over form. However, as we stated earlier, the Rephrasing activity demands a different strategy, a requirement that our architecture is able to fulfill.

6.1.3 Learner Model

As mentioned in section 4.1.3, our learner model stores error counts for each error type the system can detect. Similar to the error types in activity models, they can then be used to alter the feedback behavior. In this case, the feedback becomes learner-specific instead of activity-specific. We will demonstrate the systems reactions to two different learners in this section. In both cases, the activity is the same, a vocabulary activity that trains the learner in the use of time statements. Moreover, we assume here that both learners give the same input.

The task in both cases is to express the time displayed by a clock shown in the activity setup. We will demonstrate how the system reacts to the same input for the same activity with different feedback according to the learner's characteristics.

Student 1

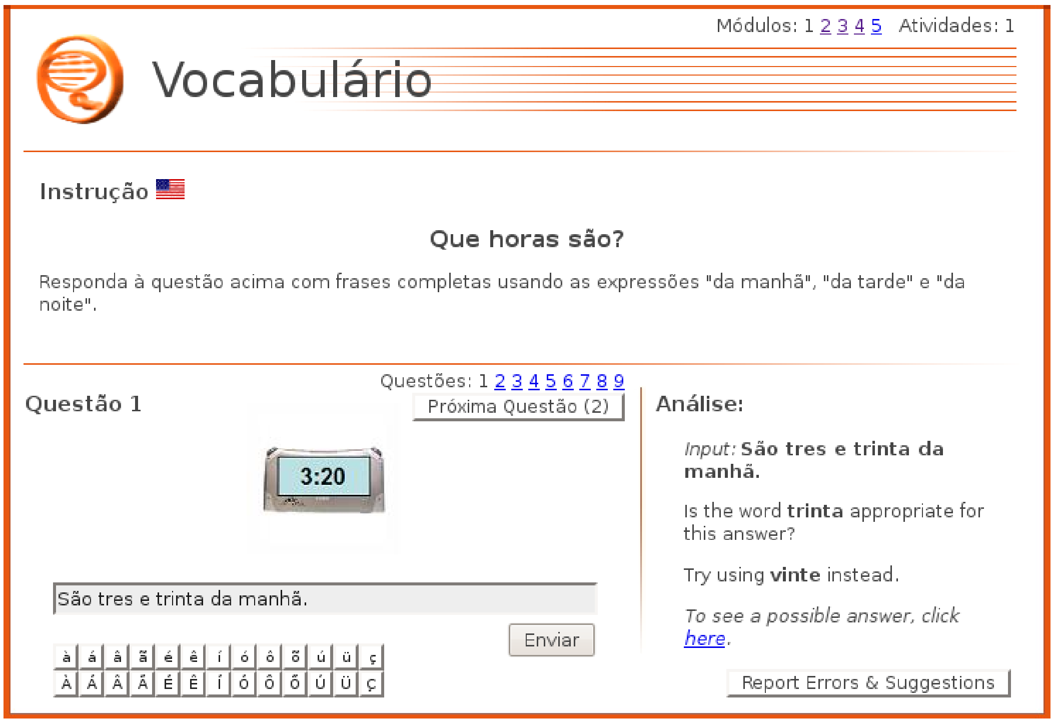
Student 1 mostly makes meaning-based errors, as shown in table 6.1. His most prominent error type are wrong lexical choices with a value of 7, indicating that he often does not choose the appropriate words. On the other hand, form-based errors seem not to be much of an issue for him, except for some spelling errors.

Figure 6.7 shows student 1's answer to the vocabulary activity we mentioned above. Student 1 entered *São tres e trinta da manhã* ('It is three thirty in the morning'). Two errors are detected: a spelling error, where *tres* ('three') should be *trés*, and a lexical choice error, where *trinta* ('thirty') should be *vinte* ('twenty').

The system responds with feedback on the lexical choice error because that is

Error type	Error count
Punctuation error	2
Capitalization error	1
Spelling error	5
Wrong word order	2
Agreement error	1
Wrong word form	0
Missing preposition	0
Extra determiner	0
Wrong lexical choice	7
Missing concepts	6
Extra concepts	3
Missing and extra concepts	6
Total	30

Table 6.1: Learner model of student 1


Módulos: 1 2 3 4 5 Atividades: 1

Vocabulário

Instrução

Que horas são?

Responda à questão acima com frases completas usando as expressões "da manhã", "da tarde" e "da noite".

Questões: 1 2 3 4 5 6 7 8 9

Questão 1

Próxima Questão (2)

São tres e trinta da manhã.

Enviar

Análise:

Input: **São tres e trinta da manhã.**

Is the word **trinta** appropriate for this answer?

Try using **vinte** instead.

To see a possible answer, click here.

Report Errors & Suggestions

Figure 6.7: Feedback on wrong word choice for student 1

Error type	Error count
Punctuation error	3
Capitalization error	5
Spelling error	9
Wrong word order	3
Agreement error	2
Wrong word form	2
Missing preposition	0
Extra determiner	0
Wrong lexical choice	1
Missing concepts	3
Extra concepts	1
Missing and extra concepts	1
Total	30

Table 6.2: Learner model of student 2

what student 1 gets wrong most of the time, according to the learner model. Hence, it makes sense to alert him to this prominent problem.

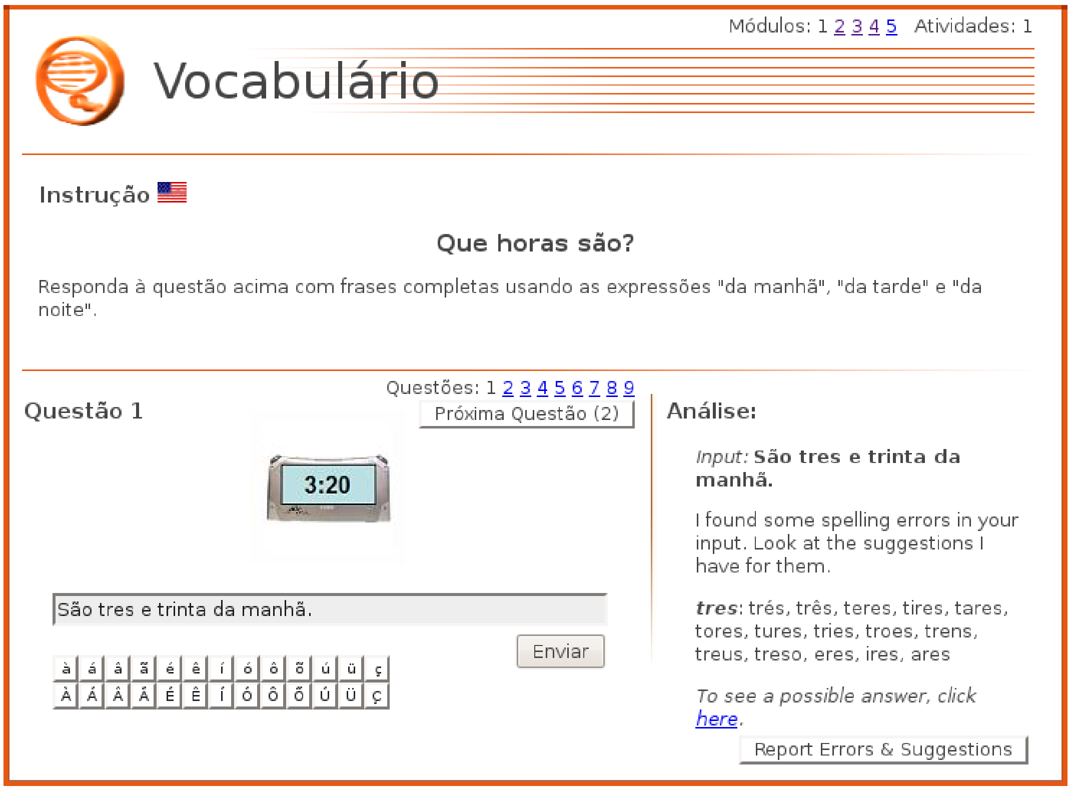
Student 2

Student 2 gets the same task and gives the same input. In contrast to student 1, whose main problem area are meaning-based errors, student 2 has more trouble with form. Table 6.2 shows his learner model. Spelling errors seem to be a major issue, with a count of 9. However, student 2 seems to provide appropriate content in most cases.

Student 2 also entered *São tres e trinta da manhã* ('It is three thirty in the morning') which is again diagnosed with two errors. But where the system picked the lexical choice error for student 1, it chooses to give feedback on the spelling error for student 2 as shown in figure 6.8.

6.2 Quantitative Evaluation of Individual Components

Although we did not attempt an evaluation of the whole system, we discuss two individual components of TAGARELA, namely the lexicon and the parser. Both are evaluated against the target answers found in TAGARELA's activity models. The rationale for doing this is that the target answers present the domain of language


Módulos: 1 2 3 4 5 Atividades: 1
Vocabulário
Instrução

Que horas são?

Responda à questão acima com frases completas usando as expressões "da manhã", "da tarde" e "da noite".

Questões: 1 2 3 4 5 6 7 8 9
Questão 1
Próxima Questão (2)
3:20

São tres e trinta da manhã.

Enviar
à á â ã é ê í ó ô õ ú ü ç
À Á Â Ã É Ê Í Ó Ô Õ Ú Ü Ç
Análise:

Input: **São tres e trinta da manhã.**

I found some spelling errors in your input. Look at the suggestions I have for them.

tres: trés, três, teres, tires, tares, tores, tures, tries, troes, trens, treus, treso, eres, ires, ares

To see a possible answer, click here.

Figure 6.8: Feedback on spelling error for student 2

	Types	Tokens
Known	480	2383
Unknown	30	52
Total	510	2435
Coverage	94.12%	97.86%

Table 6.3: Lexicon coverage of TAGARELA

that the components should cover. Hence, it is useful to know how well the lexicon and parser perform on this material and where the problems are.

It is important to note that we do not perform standard evaluation in terms of precision and recall here. The target answers are just as un-annotated as the learner input is when it is analyzed. Thus, no gold standard to compare against is available.

6.2.1 Lexicon Coverage

As presented in section 5.3.4, TAGARELA's lexicon is a full-form lexicon that is used for lookup during analysis. As such, it is expected to contain all forms that are used in the activities' target answers. Lexicon information is used for most subsequent processing steps, including parsing and shallow content matching. Consequently, good lexicon coverage of the target material is vital.

The lexicon was evaluated as follows. In all target answers, tokens were identified, annotated with lexicon information and disambiguated as in the regular system (see section 5.3). We then counted the different tokens that occurred and their total number, resulting in the familiar type/token distinction. Both types and tokens were categorized into known and unknown classes, depending on whether at least one lexicon entry was present for them or not. The results of this procedure are shown in table 6.3.

Coverage in terms of types is 94.12%, a result that seems satisfactory. Token coverage is even higher, coming in at 97.86%. The unknown words are mostly less common nouns from a specific domain, such as *espaguete* ('spaghetti') or *caipirinha*, both denoting something to eat or drink. On the whole, TAGARELA can rely on sufficient lexical coverage with respect to the current activities.

	Number of answers
Successful parses	208
Failed due to missing POS	49
Failed due to lacking grammar coverage	145
Failed parses	194
Total	402
Coverage	51.74%

Table 6.4: Grammar coverage of TAGARELA

6.2.2 Grammar Coverage

Similarly to the lexicon, we also evaluated the grammar used by the bottom-up chart parser in TAGARELA with respect to coverage. Obtaining a full analysis of the input is crucial when identifying agreement errors, one of the most important error types in ICALL systems.

In order to test the parsing process, we also had to run the tokenizer, lexicon lookup and disambiguator in addition to the parser. Again, all target answers were processed, with the exception of those in the fill-in-the-blanks activities: as they are not expected to be phrases but individual words, we excluded them from the grammar test. The total number of target answers to be parsed was 402. Table 6.4 shows how the parser did on these answers. Only complete parses that range over the whole input were counted as successful.

As can be seen, TAGARELA's grammar coverage leaves room for improvement. 51.74% total coverage is not a desirable result, but it is important to find out why exactly parsing failed. The 49 failed parses resulting from missing POS information can be blamed on the lexicon, as discussed in the previous section. However, 145 answers could not be parsed due to a lack in grammar coverage. Among these, we could identify the following main problematic areas:

- **Named entities**: Multi-word names such as *Estados Unidos* ('United States') cause the parser to fail because they have been disambiguated wrongly. *Unidos* can also be a verb (participle of *unir* ('unite')) and is disambiguated as such in some cases. Sentences with *Argentina*, *Brasil* and *Nova York* fail similarly. A fairly straightforward solution would be to treat capitalized tokens specially in the disambiguator: if a token is capitalized, its lexical entry for proper noun should be preferred if such an entry exists.

- **Coordination structures**: Coordinated NPs as in *Ela fala francês, italiano e inglês* ('She does French, Italian and English') seem not to be handled properly by the grammar rules.
- **Multi-word statements of time**: Expressions such as *na quinta a tarde* ('at 5 o'clock in the afternoon') or *3 de outubro* ('3rd of October') pose problems to the grammar.

We propose that future work on the TAGARELA grammar should concentrate on these issues.

6.3 Other Improvements and Observations

Apart from the improvements outlined in section 6.1, we also changed the original TAGARELA in other ways. An example is the diagnosis of missing concepts: in TAGARELA, missing concepts are diagnosed in terms of surface tokens. This works well as long as the learner truly does include extra lexical material. However, if a word form error is made, the surface token strategy fails because the wrong form is treated as a blend of missing and extra concepts.

We dealt with this problem by abstracting from the surface form to the canonical form of the tokens. As we showed in section 6.2.1, TAGARELA's lexicon coverage is sufficient. Encouraged by that result, we changed the diagnosis of missing concepts to use the canonic match of the learner answer to the target answer. This ensures that word form errors are not accidentally interpreted as missing concepts.

The case described above points out a crucial problem in the treatment of multiple errors in learner answers: the interdependencies of errors that we already mentioned in section 4.3.1. If we analyze the learner input on different levels, from shallow methods such as token matching to deep methods such as full parsing, we run the risk of repeatedly interpreting the same error. Consider another example: an agreement error can be diagnosed by doing full parsing with agreement checking, but can also be diagnosed as a word form error by using the result of canonic matching between learner and target answer. Clearly, we should not diagnose both errors at the same time because they refer to the same phenomenon. This kind of consideration is crucial if one intends to build a correct learner model with accurate error counts.

At the moment, we do not have a general answer to the error dependency issue. Concrete cases will need to be examined and then generalized to a common approach. A first strategy could be to not diagnose multiple errors that range over the same span of tokens in the input, on the rationale that they likely point to the same problem. However, further research is needed to verify (or falsify) this hypothesis. We intend to follow up on this issue.

Another observation that we made is the interesting proportion of analysis modules to the feedback they enable in the end. Few diagnosis modules make use of deep processing results such as agreement. On the other hand, shallow methods such as content matching are used extensively for a variety of errors, from missing concepts to wrong lexical choices. The rather poor grammar coverage we showed in section 6.2.2 also supports the assumption that shallow methods are much more relied on than deep processing strategies.

7 Conclusion

We have presented a general architecture for ILTSs. The architecture makes use of annotation-based processing to encode different linguistic information at deep and shallow levels in a flexible manner and associate it with the surface input string. At the same time, it leaves the original input untouched. We have shown that this is necessary in order for feedback to be both error-specific and useful to the learner.

Furthermore, we have presented a general mechanism for the use of activity models in guiding processing and feedback. The approach uses error types as a means of formulating pedagogical goals of individual activities. These error types are then mapped to analysis requirements for processing. More importantly, they are used to influence the feedback strategy towards the errors that a particular activity centers on. We have also demonstrated a similar use of error counts in the learner model to guide feedback.

We have evaluated the system on the basis of example scenarios. They show that we are indeed able to give better feedback than the original TAGARELA in a number of cases. This is the result of both annotation-based processing and integration of activity and learner model into the feedback process. Moreover, we have evaluated TAGARELA's lexicon and grammar which revealed that more work needs to be done on the grammar as it only covers 51.74% of the target answers with a full parse.

To our knowledge, our system is the only ILTS to use an enterprise-scale processing framework such as UIMA. We make use of UIMA's strengths to define common annotation data structures that do not depend on specific processing modules. We have recast all original processing modules of TAGARELA in UIMA, improving them in many cases, especially from an engineering perspective. Using UIMA makes our system easily extensible because components that are already available for UIMA can be added to the system with little additional work.

Nevertheless, more work remains to be done. Specifically, we want to concentrate on the following points:

- The system needs to be run with real learners so that its usefulness can be assessed properly. As we have already mentioned, this is planned for September 2009 at the University of Massachusetts Amherst.
- The question of error interdependencies needs to be addressed. As we have noted before, multiple errors in learner input are not always independent. A general approach needs to be developed that detects redundant diagnoses and decides on one of them.
- In addition to the content analysis present in the system, we want to integrate other methods such as the ones described in Bailey and Meurers (2008). A first step into that direction would be to match words on the synonym level, possibly using a resource such as Wordnet.Br[1]. Other possibilities include matching phrases or other multi-word expressions instead of single words.
- The analysis controller needs to be made more flexible. Currently, NLP modules can interleave information through annotation-based processing but they are still run in a predefined order. A controller that can dynamically decide what module is needed when would be highly useful.
- The system can be further generalized by classifying analysis and diagnosis modules by language. UIMA supports this classification readily. The language information could be specified in the activity model.
- Currently, activity models are hand-constructed. An Authoring Tool would help instructors in adding new activities to the system. We can then also determine whether our use of error types as a means of encoding pedagogical goals is suitable for instructors or whether another encoding mechanism is needed.

[1]`http://www.nilc.icmc.usp.br/~arianidf/WordNet-BR.html`

Bibliography

Luiz Amaral. *Designing Intelligent Language Tutoring Systems: integrating Natural Language Processing technology into foreign language teaching*. PhD thesis, The Ohio State University, 2007.

Luiz Amaral and Detmar Meurers. Putting activity models in the driver's seat: Towards a demand-driven NLP architecture for ICALL. EUROCALL. September 7, 2007. University of Ulster, Coleraine Campus, 2007. URL `http://www.ling.ohio-state.edu/icall/handouts/eurocall07-amaral-meurers.pdf`.

Luiz Amaral and Detmar Meurers. From recording linguistic competence to supporting inferences about language acquisition in context: Extending the conceptualization of student models for intelligent computer-assisted language learning. *Computer-Assisted Language Learning*, 21(4):323–338, 2008. URL `http://purl.org/dm/papers/amaral-meurers-call08.html`.

Luiz Amaral and Detmar Meurers. Little things with big effects: On the identification and interpretation of tokens for error diagnosis in ICALL. *CALICO Journal*, 27(1), May 2009. Forthcoming.

G. Antoniadis, S. Echinard, O. Kraif, T. Lebarbé, M. Loiseau, and C. Ponton. NLP-based scripting for CALL activities. In Lothar Lemnitzer, Detmar Meurers, and Erhard Hinrichs, editors, *COLING 2004 eLearning for Computational Linguistics and Computational Linguistics for eLearning*, pages 18–25, Geneva, Switzerland, August 28 2004. COLING. URL `http://acl.ldc.upenn.edu/coling2004/W6/pdf/3.pdf`.

Stacey Bailey and Detmar Meurers. Diagnosing meaning errors in short answers to reading comprehension questions. In Joel Tetreault, Jill Burstein, and Rachele De Felice, editors, *Proceedings of the 3rd Workshop on Innovative Use of NLP for Building Educational Applications, held at ACL 2008*, pages 107–115, Columbus, Ohio, 2008. Association for Computational Linguistics. URL `http://aclweb.org/anthology-new/W/W08/W08-0913.pdf`.

Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization at the 43th Annual Meeting of the Association of Computational Linguistics (ACL-2005)*, June 2005. URL `http://www.cs.cmu.edu/~alavie/papers/BanerjeeLavie2005-final.pdf`.

Eckhard Bick. Live use of corpus data and corpus annotation tools in CALL: Some new developments in VISL. In Henrik Holboe, editor, *Nordic Language Technology, Arbog for Nordisk Sprogteknologisk Forskningsprogram 2000-2004 (Yearbook 2004)*, pages 171–186. Museum Tusculanum, Copenhagen, 2005a. URL `http://beta.visl.sdu.dk/pdf/corpus_and_CALL_form.pdf`.

Eckhard Bick. Grammar for fun: IT-based grammar learning with VISL. In Peter Juel, editor, *CALL for the Nordic Languages*, Copenhagen Studies in Language, pages 49–64. Samfundslitteratur, Copenhagen, 2005b. URL `http://beta.visl.sdu.dk/pdf/CALL2004.pdf`.

Chris Bowerman. ICALL: An underview of the state of the art in computer-aided language teaching. *CALL*, 3, 1990. URL `http://www.cet.sunderland.ac.uk/cbowww/PAPERS/icai2.txt`.

S. Pit Corder. Error analysis. In *The Edinburgh Course in Applied Linguistics*, volume 3, chapter 5, pages 122–154. Oxford Universtiy Press, 1974.

Fred Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, March 1964.

William DeSmedt. Herr kommissar: An icall conversation simulator for intermediate german. In Holland et al. (1995), pages 153–174.

Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

D. Ferrucci and A. Lally. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3–4):327–348, 2004.

J. Gerbault. Towards an analysis of answers to open-ended questions in computer-assisted language learning. In S. Lajoie and M. Vivet, editors, *Proceedings of AIED*, pages 686–689. IOS Press, 1999.

Thilo Götz and Oliver Suhre. Design and implementation of the UIMA Common Analysis System. *IBM Systems Journal*, 43(3):476–489, 2004.

Trude Heift. Error-specific and individualized feedback in a web-based language tutoring system: Do they read it? *ReCALL*, 13(2):129–142, 2001.

Trude Heift. Multiple learner errors and meaningful feedback: A challenge for ICALL systems. *CALICO Journal*, 20(3):533–548, 2003. URL `https://www.calico.org/a-293-Multiple%20Learner%20Errors%20and%20Meaningful%20Feedback%20A%20Challenge%20for%20ICALL%20Systems.html`.

Trude Heift. Corrective feedback and learner uptake in CALL. *ReCALL*, 16(2): 416–431, 2004.

Trude Heift. *Designed Intelligence: A Language Teacher Model*. PhD thesis, Simon Fraser University, 1998.

Trude Heift and Devlan Nicholson. Web delivery of adaptive and interactive language tutoring. *International Journal of Artificial Intelligence in Education*, 12(4): 310–325, 2001. URL `http://aied.inf.ed.ac.uk/members01/archive/vol_12/heift/paper.pdf`.

V. Holland, J. Kaplan, and M. Sams, editors. *Intelligent Language Tutors. Theory Shaping Technology*. Lawrence Erlbaum Associates, Inc., New Jersey, 1995.

Ken Hyland and Fiona Hyland. Feedback on second language students' writing. *Language Teaching*, 39(2):1–46, 2006.

Ronald Kaplan and Joan Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–381. MIT Press, Cambridge, MA, 1983.

Fred Karlsson. Constraint grammar as a framework for parsing running text. In *Proceedings of COLING*, pages 168–173, 1990.

Vijay Krishnan, Sujatha Das, and Soumen Chakrabarti. Enhanced answer type inference from questions using sequential models. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 315–322, 2005.

V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

Lori S. Levin and David A. Evans. ALICE-chan: A case study in ICALL theory and practice. In Holland et al. (1995), pages 77–98.

Michael Levy. *Computer-Assisted Language Learning: Contex and Conceptualization*. Oxford University Press, New York, 1997.

Roger Levy and Galen Andrew. Tregex and Tsurgeon: tools for querying and manipulating tree data structures. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, 2006.

Ronaldo Martins, Graça Nunes, and Ricardo Hasegawa. Curupira: A functional parser for Brazilian Portuguese. In *Computational Processing of the Portuguese Language*, Lecture Notes in Computer Science, pages 195–200. Springer Berlin / Heidelberg, 2003.

Noriko Nagata. BANZAI: An application of natural language processingto web based language learning. *CALICO Journal*, 19(3):583–599, 2002. URL `https://www.calico.org/a-425-BANZAI%20An%20Application%20of%20Natural%20Language%20Processing%20to%20Webbased%20Language%20Learning.html`.

Noriko Nagata. An effective application of natural language processing in second language instruction. *CALICO Journal*, 13(1):47–67, 1995. URL `http://www.usfca.edu/japanese/CALICO95.pdf`.

Noriko Nagata. Computer vs workbook instruction in second langauge acquistion. *CALICO journal*, 14(1):53–75, 1996. URL `http://www.usfca.edu/japanese/CALICO96.pdf`.

John Nerbonne. Natural language processing in computer-assisted language learning. In Ruslan Mitkov, editor, *The Oxford Handbook of Computational Linguistics*. Oxford University Press, 2003. URL `http://www.let.rug.nl/~nerbonne/papers/nlp-hndbk-call.pdf`.

Tim O'Shea and John Self. *Learning and teaching with computers*. Prentice Hall, 1983.

Christopher Pinchak and Dekang Lin. A probabilistic answer type model. In *Proceedings of the 11th Conference of the European Chapter of the Association of Computational Lingustics (EACL)*, pages 393–400, 2006.

Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, 1994.

John W. Ratcliff and David Metzener. Pattern matching: The Gestalt Approach. *Dr. Dobb's Journal*, 6:46, 1988.

Janine Toole and Trude Heift. Task-generator: A portable system for generating learning tasks for intelligent language tutoring systems. In *Proceedings of ED-MEDIA 02, World Conference on Educational Multimedia, Hypermedia and Telecommunications, Charlottesville, VA*, pages 1972–1978. AACE, 2002a.

Janine Toole and Trude Heift. The tutor assistant: An authoring tool for an intelligent language tutoring system. *Computer Assisted Language Learning*, 15(4):373–386, 2002b. ISSN 0958-8221. URL http://www.informaworld.com/10.1076/call.15.4.373.8266.

Ralph M. Weischedel and Norman K. Sondheimer. Meta-rules as a basis for processing ill-formed input. *Computational Linguistics*, 9(3-4), 1983.

Johannes Widmann, Kurt Kohn, and Ramon Ziai. The SACODEYL search tool: exploiting corpora for language learning purposes. In *Proceedings of the 8th Teaching and Language Corpora Conference*, 2008.

Holger Wunsch. Annotation Grammars and their compilation into Annotation Transducers. Master's thesis, University of Tübingen, 2003.

A List of Abbreviations

AE	Analysis Engine
API	Application Progamming Interface
AI	Artificial Intelligence
CALL	Computer-Assisted Language Learning
CAS	Common Analysis System
CMC	Computer-mediated Communication
DTD	Document-type Definition
EFL	English as a Foreign Language
FLA	Foreign Language Acquisition
FLT	Foreign Language Teaching
HPSG	Head-driven Phrase Structure Grammar
ICALL	Intelligent Computer-Assisted Language Learning
ILTS	Intelligent Language Tutoring System
LFG	Lexical Functional Grammar
LMS	Learning Management System
NER	Named Entity Recognition
NLP	Natural Language Processing
POS	part of speech
RDD	Responsibility-driven Design
SLA	Second Language Acquisition
SOFA	Subject of Analysis
UIMA	Unstructured Information Management Architecture
XML	eXtensible Markup Language