The Tutor-in-the-Loop Model for Formative Assessment

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen
University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften
genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Daniel Chu Herding

aus Münster (Westf.)

Berichter:

Prof. Dr. Ulrik Schroeder
Prof. Dr. Christian Spannagel


Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.
Acknowledgement

Parts of the research described in this thesis has been carried out within the SAiL-M project (Semi-automatic assessment of individual learning processes in mathematics). The project was funded by the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung, BMBF) in the support program “Hochschulforschung als Beitrag zur Professionalisierung der Hochschullehre – Zukunftswerkstatt Hochschullehre” (“Higher Education Research as a Contribution to the Professionalization of Higher Education – Future Workshop Higher Education”). Project management organisation was provided by DLR.

SAiL-M was a joint research project of these partners:

- Ludwigsburg University of Education
- Karlsruhe University of Education
- Weingarten University of Education
- RWTH Aachen University

1 SAiL-M project: sail-m.de/en/
2 DLR – Professionalization of Higher Education: http://www.dlr.de/pt_zwh
Abstract

Closed exercise types like multiple-choice tests are widespread in higher education, especially for purposes of summative assessment. However, they cannot be considered learner-centric because they do not allow individual solutions. In contrast, assessing open-ended tasks is considerably more complex. Intelligent tutoring systems (ITS) aim at giving individual feedback fully autonomously, but due to their high development cost, they are available only for few domains.

This thesis presents the tutor-in-the-loop model, which follows an alternative, semi-automatic approach to formative assessment. It combines automatic and tutorial feedback with the goal of letting the computer give feedback to standard solutions and typical mistakes. Students who need additional help or who have come up with extraordinary solutions can request feedback from their tutor. The students’ interactions with the electronic exercises are recorded to help the tutor comprehend their solution processes. This log data can also be used to perform learning analytics: all logs of a course can be analyzed to detect common mistakes or to detect students who may need help.

Various systems were developed to evaluate the tutor-in-the-loop model. A general framework was realized to cover aspects of automatic feedback which learning applications have in common. Several proofs of concept were implemented to show the applicability of the framework in different contexts. Finally, a logging component was realized based on an existing capture and replay toolkit. Evaluations showed intrinsic drawbacks of this toolkit in terms of deployment and analytics support. Therefore, it was replaced by a new system for logging and learning analytics.

During evaluations it was found out that, while learners indeed access automatically generated feedback when necessary, some are reluctant to request feedback from their tutors. Privacy concerns were a major impediment to tutorial feedback requests. Still, several students contacted their tutors, and their questions could be answered with the help of the solution process logs.
Contents

1. Introduction .................................................. 1

2. Feedback Theories ............................................ 5
   2.1. Purposes of Assessment ................................... 6
   2.2. Sources of Feedback ...................................... 9
   2.3. Timing of Feedback ...................................... 14
      2.3.1. Immediate Feedback .................................. 14
      2.3.2. Delayed Feedback .................................... 17
      2.3.3. Alternative Push Strategies ......................... 18
      2.3.4. Feedback on Demand ................................ 21
   2.4. Explicitness of Feedback ................................ 25
      2.4.1. Hints ................................................. 27
      2.4.2. Cheating Prevention ................................ 27
   2.5. The Feedback Loop ...................................... 30
   2.6. The Role of Learning Analytics in Assessment .......... 31

3. Related Work .................................................. 35
   3.1. Automatic Assessment .................................... 35
   3.2. Semi-automatic Assessment Systems .................... 40
   3.3. Interaction Analysis .................................... 42

4. Conceptual Model .............................................. 47
   4.1. The Tutor-in-the-Loop Model ............................. 49
   4.2. Research Questions ..................................... 53

5. Technological Foundations ................................... 55
   5.1. The Jacareto Framework ................................ 55
      5.1.1. Capture & Replay Engine ......................... 55
      5.1.2. User Interfaces for Jacareto ..................... 59
      5.1.3. Modules ............................................. 61
      5.1.4. Existing Use Cases for Jacareto ................. 62
   5.2. Cinderella .............................................. 66

6. A Generic Framework For Automatic Feedback ................ 69
   6.1. Data Model ............................................. 70
   6.2. Feedback Dialog ....................................... 72
   6.3. Problem Highlighting ................................... 74
   6.4. Tutor Feedback Requests ................................. 75
7. **Proofs of Concept**
   7.1. Co lnProof-M
       7.1.1. Domain
       7.1.2. Visualizations
       7.1.3. Assessment
       7.1.4. Exercises
   7.2. SetSails!
       7.2.1. Domain
       7.2.2. Visualizations
       7.2.3. Transformations
       7.2.4. Assessment
   7.3. Further Tools Using the Feedback-M Framework

8. **Interaction Logging for Tutorial Feedback and Learning Analytics**
   8.1. Solution Process Recording with Picorder
       8.1.1. Enhancements of Jac Areto
       8.1.2. Adaptation of Target Applications
       8.1.3. Inclusion of Semantic Events
       8.1.4. Evaluation
   8.2. Jac Areto as a Library
       8.2.1. Integration of Jac Areto into Learning Applications
       8.2.2. Extensions to the Capture and Replay Mechanism
       8.2.3. Transfer of Records
       8.2.4. Evaluation
   8.3. SMALA
       8.3.1. Development
       8.3.2. Evaluation

9. **Summary and Perspectives**

A. **Examples for Proofs in Set Theory**
B. **Bibliography**
C. **List of Tables**
D. **List of Figures**
1. Introduction

In institutional learning, assessment plays an essential role not only because it is the basis for evaluating students’ knowledge, but also because formative feedback gives them the chance to reflect on their performance and to learn from their mistakes. Besides exams, assessment is mainly performed in the form of homework for school students, and of regular – typically weekly – assignments at university. With the rise of technology-enhanced learning, computer-aided assessment (or computer-assisted assessment: CAA, Bull and McKenna 2004) has found its way into institutional learning. Drill-like exercises with closed tasks such as multiple-choice questions are predominant mainly because they are relatively easy to implement as software. Multiple-choice questions are especially widespread in the United States, which have a tradition of optical mark recognition and optical scanning (Snow et al., 1996) that predates modern computer-aided assessment. In Germany, multiple-choice questions are less prevalent. Still, an increased use of CAA at German universities is required because of expected rises in student enrollment numbers.\footnote{Among other factors, the number of student enrollments in Germany is rising because of the suspension of compulsory military service and because higher education entrance qualification has been changed from 13 to 12 years of school. This transition will lead to one year in which twice as many school graduates will enter university.}

More recent approaches to CAA have led to systems which are capable of assessing open-ended tasks like free-text assignments, geometric constructions, and programming exercises. In contrast to closed questions, such types of assessment are student-centered because they provide each student with feedback on his\footnote{For the sake of legibility, this thesis uses the generic masculine form without meaning to exclude female individuals.} individual solution, rather than on his selection from a set of predefined options. In order to automatically assess solutions to open-ended tasks, it is necessary to implement domain-specific tests which evaluate various aspects of the solution and which generate feedback accordingly. This process, which is referred to as model tracing, will be discussed in section 2.2. Unfortunately, developing model tracing algorithms is much costlier than realizing a simple multiple-choice system. It is especially laborious to develop automatic tests that can evaluate any solution which a learner may possibly come up with. Due to the high degree of freedom that
Chapter 1. Introduction

is intrinsic to open-ended tasks, it is hard to cover the entire solution space with automatic tests.

CAA tools are usually embedded in a context of institutional learning: students use them in addition to attending lessons, lectures, or tutorials. This combination of the traditional classroom and educational technologies is called blended learning (Bonk et al., 2005). Developers of learning applications often try to realize fully autonomous tools, neglecting the fact that teaching professionals (e.g., teachers, professors, and tutors) are available to provide assistance. At university level, many professors offer a consultation hour in their office during which students can request advice on the learning contents. While it is possible to shift this principle to online communication, this is rarely done outside of distance learning scenarios. Nevertheless, tutor consultation offers an opportunity for formative assessment: by combining it with the model-tracing approach, it is possible to let a computer evaluate standard solutions and detect typical mistakes, while the tutor supports students who need further help or who have come up with extraordinary solutions. Both types of feedback can be provided as an on-demand service to leave students in control of the pace.

In contrast to the on-demand approach, feedback on closed tasks such as multiple-choice tests is usually pushed to the user, either immediately after his answer, or with a delay. A summary of push strategies for feedback will be provided in section 2.3. When it comes to open-ended tasks, the principle of model tracing makes it possible not only to provide on-demand feedback, but also to let the application actively push feedback to the user. In this case, delayed feedback has the disadvantage of not being available during the solution process. Students are therefore at risk of being stuck and without any assistance. Immediate feedback does not have this problem: it interrupts the learner already during his solution process. This, however, has three major downsides. Firstly, the interruption might unnecessarily distract the learner and disrupt his flow of thoughts. Secondly, because each mistake is immediately reported, it is impossible for the student to grasp the real consequences of a mistake. Thirdly, students get no opportunity to spot their mistakes on their own.

There are attempts to improve the timing of automatic assessment so that feedback is pushed only in appropriate situations. In this endeavor, one has to keep in mind that there are different learner types among students (Felder and Silverman, 1988), which means that the system has to adapt to the characteristics of the learner. Intelligent tutoring systems (ITS) try to automatically guide the learner by running so-called knowledge-tracing algorithms during all learning processes.

---

3 In the context of intelligent tutoring systems, the term “tutor” refers not to a human, but to a software agent. This shows that ITS’s are supposed to work fully autonomously, replacing the human tutor.
Table 1.1.: Different feedback timings and sources. The feedback model presented in this thesis combines the three feedback types which are shaded in gray.

<table>
<thead>
<tr>
<th>On-demand feedback</th>
<th>Automatic feedback</th>
<th>Tutor feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model tracing</td>
<td></td>
<td>Tutor consultation</td>
</tr>
<tr>
<td>Pushed feedback</td>
<td>Intelligent tutoring systems</td>
<td>Learning analytics</td>
</tr>
</tbody>
</table>

(CORBETT and ANDERSON, 1995). The software is supposed to detect the strengths and weaknesses of the learner in order to navigate him through the learning application along an individualized path. The results of knowledge tracing, in combination with those of model tracing, could also be used to determine the optimal timing and formulation of feedback. Unfortunately, the development of such a system is very costly because it is supposed to work fully automatically. It is not only required to implement tests which can assess any solution that is theoretically possible, but developers also have to conceive and realize a system which classifies the learner style and adapts the feedback strategy accordingly. Because intelligent tutoring systems are highly domain-specific, it is difficult to reuse parts of existing ones.

More recently, a new, analytics-based approach has emerged in educational research. One goal of learning analytics is to gather data on the student and his learning behavior, and to “pick up on signals that indicate difficulties with learner performance” (SIEMENS, 2010). A learning analytics toolkit provides teachers with visualizations of these signals so that he can take appropriate measures, e.g. guide students who have been highlighted by the system. On a small scale, this principle could be used to analyze solution processes and to point the teacher toward students who might require assistance. This might be an addition to on-demand feedback so that students can receive assistance even when they are not aware of the difficulty. Unlike ITS’s, the learning analytics approach to assessment is semi-automatic because the analytics system only serves as an indicator; the decision whether or not to intervene is left to the teacher.
Overview of this thesis

After discussing feedback theories in the following chapter and comparing related assessment systems in chapter 3, this thesis presents the tutor-in-the-loop model in chapter 4. This feedback model is an alternative to the fully automatic ITS approach. It combines model tracing and tutor consultation with principles from learning analytics (see table 1.1) to provide learners with semi-automatic feedback. The description of a system which realizes the tutor-in-the-loop model is split over several chapters. Chapter 5 deals with the software that forms the technological foundation of the system. It is followed by a description of a generic Java framework to integrate automatic feedback mechanisms into learning applications. Chapter 6 presents several learning applications which serve as proofs of concept to show that the feedback framework can be used in various contexts. Finally, chapter 8 deals with the development of the logging module, which records solution logs that are required both for tutor consultation and for learning analytics. Because it has been developed iteratively in three phases, the evaluations of the logging module – and of the overall systems – are described at the end of each section inside that chapter.
2. Feedback Theories

One of the oldest research questions in computer-aided assessment is how to present feedback on exercises to maximize its effectiveness. This includes when and by whom feedback should be generated, how it should be formulated, and how its delivery should be triggered. Several theories which revolve around aspects of exercise feedback will be discussed in this chapter.

According to Rütter, one can classify exercises into the three categories “offene, halboffene und geschlossene Aufgaben”, i.e. open-ended, semi-open, and closed tasks (1973). In closed tasks, such as multiple-choice questions, all possible answers are visible to the learner, who simply selects one. In semi-open tasks, the correct solution is not visible, but is known to the system, as is the case with fill-in-the-blank texts. In contrast, open-ended tasks are those in which different paths can lead to a correct solution. An alternative classification is given by Bull and McKenna (2004), who discern objective and non-objective tests. Closed and semi-open tasks are always objective because they only have one correct solution. Open tasks, on the other hand, are often non-objective.

The taxonomy of cognitive domain by Bloom (1956), which was later revised by Anderson et al. (2000), is a classification of different learning objectives. It ranges from simple objectives such as remembering a fact or understanding a concept to complex skills like creating an artifact or evaluating someone else’s solution (see figure 2.1). Closed and semi-open tasks usually only assess whether a learner has remembered or understood the learning contents. It is possible to create a task in which the learner has to analyze a situation and choose an option based on his decision; however, this does not take the process into account, but only the final solution. Open-ended tasks are usually located on the higher levels of Bloom’s taxonomy. For instance, a task in which the learner has to solve a linear equation system assesses not only whether he understands the concepts of linear algebra, but also whether he can apply this knowledge. Because intermediate steps can be analyzed as well, the solution process can be taken into account to provide feedback.
2.1. Purposes of Assessment


1. “to inform learners about their own learning.”

2. “to inform teachers of the strengths and weaknesses of the learners and of themselves so that appropriate teaching strategies can be adopted.”

3. “to inform other stakeholders – society, funders, employers including the next educational stage.”

4. “to encourage learners to take a critical-reflective approach to everything that they do, that is, to self assess before submitting.”

5. “to provide a summative evaluation of achievement.”
These purposes can be grouped into two categories: summative and formative assessment. Summative assessment is “designed to judge the extent of students’ learning of the material in a course, for the purpose of grading, certification, evaluation of progress or even for researching the effectiveness of a curriculum” (BLOOM et al., 1971, p. 117, cited in WILIAM and BLACK, 1996, p. 537). In contrast, formative assessment is defined as “information communicated to the learner that is intended to modify his or her thinking or behavior to improve learning. […] The main aim of formative feedback is to increase student knowledge, skills, and understanding in some content area or general skill (e.g., problem solving)” (SHUTE, 2008, p. 156). Of the assessment purposes listed above, the third and last ones are summative, while the first and fourth ones are formative. The second purpose – informing teachers so that they can adopt appropriate teaching strategies – is mainly summative, but can have a formative characteristic if the measures are taken so promptly that the learners benefit from them. It is important to note that there need not be a strict separation between formative and summative assessment: data which has been collected during formative assessment activities can also be used to grade students. The ASSISTment concept, which blends assisting and assessment (FENG et al., 2009), is an example for this combination; it will be discussed later in this thesis.

Figure 2.2.: Student and staff rationales for constructive feedback, according to THOMAS (2009).
From the point of view of a student who is working on an assignment, the most relevant of the assessment purposes listed by Houston (2001) is that feedback helps him to reflect on his performance. This reflection is only possible when the learner receives feedback which informs him about the results of the assessment. Kulhavy broadly defines feedback as “any of the numerous procedures that are used to tell a learner if an instructional response is right or wrong” (1977, p. 211). It is, however, important to note that feedback is not restricted to this binary information. Positive feedback can contain a compliment, and negative feedback may carry information such as where the mistake is located, why it is a mistake, and how it could be fixed. As shown in a meta analysis by Hattie and Timperley (2007), one can say that feedback in general has a clearly positive influence on learning. This is also confirmed by Shute “formative feedback has been shown in numerous studies to improve students’ learning and enhance teachers’ teaching to the extent that the learners are receptive and the feedback is on target (valid), objective, focused, and clear” (2008, p. 182).

In summative assessment, feedback is optional, and is often limited to a score or a grade. In contrast, detailed feedback is an indispensable part of formative assessment. Whitelock notes that such feedback can be considered “advice for action”, as it “will assist [students] with future learning scenarios” (2010, p. 320). To fulfill its goal, feedback in formative assessment has to be constructive. Thomas points out that “student and teacher rationales for receiving and providing constructive feedback in fact mirror each other” (2009, p. 515). This duality is shown in figure 2.2. From a literature analysis, Nicol and Macfarlane-Dick (2006, p. 205) derive seven principles of good feedback practice. They state that good feedback:

1. “helps clarify what good performance is (goals, criteria, expected standards);”
2. “facilitates the development of self-assessment (reflection) in learning;”
3. “delivers high quality information to students about their learning;”
4. “encourages teacher and peer dialogue around learning;”
5. “encourages positive motivational beliefs and self-esteem;”
6. “provides opportunities to close the gap between current and desired performance;”
7. “provides information to teachers that can be used to help shape teaching;”
2.2. Sources of Feedback

There is a multitude of sources from which learners can receive feedback. \textsc{Race} (2001) calls these sources “assessors”. \textsc{Hattie} and \textsc{Timperley} state that “feedback is conceptualized as information provided by an agent (e.g., teacher, peer, book, parent, self, experience) regarding aspects of one’s performance or understanding” (2007, p. 81). This listing, however, needs improvement. Firstly, the book is listed as an agent. In fact, it is a passive medium which the student can use to look up information in the sense of self-assessment. Secondly, it is left unexplained how experience should be an agent which provides feedback. Thirdly, and most strikingly, feedback provided by computers (and other machinery) is completely left out of the enumeration.

In traditional learning scenarios at school, feedback given by a teacher is the most dominant form of feedback. Likewise, feedback at university is usually given by a teaching professional such as a professor, a research assistant, or a student who works as a tutor\footnote{For the sake of brevity, the word “teacher” or “tutor” in this thesis refers to any teaching professional who can give feedback to a student.}. To date, the professional experience of teachers makes their feedback more valuable than feedback from any other source. Tutors are able to give feedback which “is carefully designed to allow students to do as much of the work as possible while still preventing floundering” (\textsc{Merrill} et al., 1992, p. 283). “The goal of encouraging the student to tackle as much of the problem solving effort as possible suggests that human tutorial feedback may be superior in this regard to computer tutors.” (ibid., p. 298). In addition, human tutors are able to pick up signs which are an indication for the learner’s situation. “In face-to-face interaction the tutor is able to exploit not only a learner’s task actions but also their verbal and non-verbal acts of communication, both deliberate and spontaneous, in order to assess whether they seem likely to be on task, or are confused, concentrating, thinking or resting” (\textsc{Wood} and \textsc{Wood}, 1999, p. 154f.).

School teachers usually assign homework as a form of formative assessment, while exams have a primarily summative purpose. Similarly, many university lectures feature assignments and exams. The most widespread model of university assignments is that of weekly exercises, in which students receive a weekly assignment sheet which they work on; before a deadline, they submit their solutions to a tutor for correction and grading. The major drawback of this process is that students receive feedback only on their final submission, and do not get any assistance during their solution process (\textsc{Herding} et al., 2010). There are attempts to alleviate this problem by allowing students to re-submit solutions (\textsc{Malmi} and \textsc{Korhonen}, 2004), but this still leaves
out students who have trouble getting started with the exercise, and who cannot come up with a submittable solution.

While the assignment process was originally realized with assignment sheets and submissions on paper, learning content management systems (LCMS) have simplified online distribution, submission, and grading of exercises (STALLJOHANN et al., 2011). Standards such as IMS Content Packaging\(^2\) or SCORM\(^3\) have made LCMS’s more interoperable. Still, LCMS’s provide only infrastructural support for assignment management; the assessment itself is done by a human tutor. Some systems allow students to upload submissions before the deadline. Tutors can then give them preliminary feedback so that students can resubmit an improved version. Unfortunately, experiences with the exercise module of the \(L^2P\) LCMS have shown that this is rarely done in practice (STALLJOHANN et al., 2009).

Automatically generated feedback outdates the modern computer, having been realized already with the punchcard-based teaching machine of PRESSEY (1926). This device was restricted to multiple-choice tests. Despite its obvious limitations, at least it did give feedback to the learner – unlike the optical mark recognition or optical scanning approaches (SNOW et al., 1996) which are still widespread methods of summative assessment in the United States. As an alternative, there is a plethora of authoring tools that provide automatic feedback for closed questions. On top of those, systems such as HotPotatoes (WINKE and MAC-\_\_\_\_\_\_-GREGOR, 2001) support constructing semi-open exercises like fill-in-the-blank texts or matching tasks. IMS Question & Test Interoperability (IMS QTI)\(^4\) is a standard for exchanging such tests, but it is also limited to closed and semi-open questions.

Authoring tools which provide automatic feedback for open-ended tasks are naturally more difficult to implement than systems that only work on closed or semi-open questions. The reason for this is that the solution has to be evaluated dynamically, rather than compared to a fixed sample solution. In the ACT-R theory\(^5\), ANDERSON reasons that “cognitive skills are realized by production rules” (1993, p. 59). By implementing these production rules in software, it is possible to automatically analyze the student’s solution. This type of assessment is known as model tracing (ANDERSON and PELLETIER, 1991). In the process, “students’ problem-solving steps are compared with the reasoning of an underlying domain expert. This matching is used to provide

---

\(^2\) IMS Global Learning Consortium – Content Packaging Specification: [http://www.imsglobal.org/content/packaging/](http://www.imsglobal.org/content/packaging/) (accessed 2012-08-31).


\(^5\) The ACT-R theory and its predecessors are described in detail by MIELKE (2001).
2.2. Sources of Feedback

ongoing feedback to students while they progress through a problem” (MERRILL et al., 1992, p. 278). Such approaches are summarized under the heading “intelligent assessment” (BESCHERER et al., 2011), to distinguish them from the static evaluations performed to assess closed exercises. There is evidence that intelligent assessment is indeed effective in helping students (ALEVEN and KOEDINGER, 2000; WOOD and WOOD, 1999).

Feedback can not only be provided by teaching professionals. The most prevalent form of this is peer assessment, in which learners receive feedback from their fellow students. Because peer feedback is not as authoritative as tutor feedback (RACE, 2001, p. 6), it should not be used for summative, but only for formative assessment. Furthermore, the supervisor of the peer assessment activity has to make sure that the peers formulate “feedback that is honest and supportive in a manner and mode that does not ostracise the recipient, but gives encouragement to go on” (CHURCH, 2008). This can be achieved by instructing the peers on how to write useful and motivating feedback. Above that, students can be encouraged to give honest feedback by anonymizing the review process (WOLFE, 2004).

Because students are confronted to their peers’ solutions, they see the exercises from another point of view and may be confronted with approaches different from their own ones. This gives them the chance to get a deeper understanding of the contents. In addition to that, because the students have to evaluate and communicate in order to give feedback, they can gain important soft skills for group work. Furthermore, like automatic assessment, peer assessment relieves the tutors because they do not have to correct the students’ submissions. Of course, this comes at the expense of a higher workload for the students. Still, it is not mere laziness of tutors which leads to peer assessment activities being deployed in practice. RACE points out that students are often able to make assessment judgments “more objectively than would be made by someone (for example a tutor) who already knew how to do the task involved, and had not just learned how to do it” (2001, p. 5).

Peers must not always be fellow students who are attending the same course. Internet platforms such as online social networks, forums, chat rooms, or wikis can be combined by a student to form his personal learning environment (PLE, cf. CHATTI, 2010). Because these sites are not under the control of an educational institution, learning activities which take place there can be considered informal. Both online friends and strangers can be a source of feedback. Nevertheless, it remains an open research question whether (and if, how) informal learning activities can be acknowledged by formal educational institutions (HERDING et al., 2012).
Finally, each student can assess his own performance. Unfortunately, it is more difficult to spot one’s own errors than to correct someone else’s work (Gillespie and Lerner, 2003, p. 125). For this reason, self-assessment usually relies on feedback from another source, e.g. automatic or tutor assessment. The learner can combine the results of all available assessment processes to get an impression of his overall performance level, his strengths and weaknesses. In this case, self-assessment is a higher-level judgment activity. A lower-level alternative is to provide students with a catalog of criteria by which they are supposed to judge their own work (Race, 2001). Students can participate in creating this catalog, in which case self-assessment can be described as “the involvement of students in identifying standards and/or criteria to apply to their work and making judgements about the extent to which they have met these criteria and standards” (Boud, 1991, p. 5, cited in Thomas, 2009, p. 518).

The analysis of different feedback sources has shown that each one has advantages and disadvantages. Tutor assessment is most effective, but also costly. Automatic assessment can be provided more timely, and like peer assessment, it is more readily available; however, it does not reach the quality and adaptiveness of tutor feedback, and it is expensive to implement detection algorithms for all possible types of mistakes. Peer assessment is less reliable than feedback from teaching professionals, and it increases student workload. By combining feedback from multiple sources, one can try and overcome the disadvantages. For instance, a tutor who is short on time may choose to only underline feedback, expecting students to find out on their own why each marked position is wrong. This is a combination of tutor and self-assessment. Bull and McKenna give other examples of “mixed-mode feedback” (2004).

The most promising combination of feedback sources is a hybrid of automatic and tutor assessment. This combination is called semi-automatic feedback (Herdig et al., 2010). “Semi-automated assessment reduces the claim to automatically interpret all processes. Instead, it combines the ability of computers to detect standard solutions or mistakes and the professional skills of the tutors and lectures to understand exceptional solutions” (Bescherer and Spannagel, 2009, p. 433). The fact that the tutor acts as a fallback means that he has to answer less feedback requests than he would have to in a fully manual assessment scenario, and thus can concentrate on the students who have extraordinary problems. It also means that less work is required to develop learning applications, as they do not have to completely evaluate any possible solution which a student may come up with. Semi-automatic assessment is in line with the principles for good practice in undergraduate education by Chickering and Gamson (1987) in not only providing prompt feedback, but also encouraging student-faculty contact. Ahoniemi and Karavirta call semi-automatic feedback “a good
2.2. Sources of Feedback

Figure 2.3.: Students receive semi-automatic feedback from a computer and a tutor. Photo courtesy of DAAD/Andreas Hub.

combination of easing the teacher’s workload while still supporting the students well enough” (2009, p. 333). This description is a bit understated: it disregards further advantages of automatic assessment, e.g. being available at all times. These properties make it an ideal complement to individualized tutorial feedback.

A classification of feedback sources is given by HERDING et al. (2012). It splits assessment processes up into formal assessment, self assessment, network assessment, and open assessment (see figure 2.4). Formal assessment embraces assessment activities which are already established in institutional learning, i.e. feedback given by teachers and feedback generated by computers. Network assessment covers peer teaching and related scenarios. During self assessment, students reflect their own performance during assessment activities. Open assessment takes place during learning activities outside of the scope of educational institutions which students perform in their free time. Due to the broad spectrum of the field of assessment, and the vast number of open research questions surrounding it, this thesis is limited to formal assessment, i.e. tutor and automatic feedback in educational institutions.
2.3. Timing of Feedback

Timing is an important aspect to make feedback effective. However, which timing is optimal depends on the learning scenario. Numerous studies have been performed to analyze the influence of timing on the effectiveness of feedback. An overview of some of this research is given by Kulik and Kulik (1988).

2.3.1. Immediate Feedback

Defining immediate feedback as “informative corrective feedback given to a learner or examinee as quickly as the computer’s hardware and software will allow during instruction or testing” (Musch, 1999, p. 23). It is usually contrasted to delayed feedback (see section 2.3.2), but this distinction is not always clear. For instance, Kulik and Kulik (1988) list studies in which feedback given directly after a test is considered delayed (because it is being compared to immediate feedback after each test item), but also studies in which it is considered immediate (as it is being compared to feedback given several days after the test). A taxonomy for the timing of feedback that is supposed to clarify

6 Of course, computers are not the only possible source of immediate feedback (see section 2.2).
the distinction between immediate and delayed feedback can be found in [Dempsey and Wager, 1988].

Realized already in the teaching machine of [Pressey, 1926], assessment with immediate feedback belongs to the earliest topics of what is today known as technology-enhanced learning. As such, it has been researched for decades. Many feedback theories which support giving feedback immediately after a mistake has been made are based on behaviorism, more specifically on operant conditioning. This is a model of learning which has its roots in the experiments of [Thorndike, 1898], who observed the behavior of cats which were repeatedly locked into so-called puzzle boxes. In time, cats improved upon their initial trial-and-error approaches to disabling the locking mechanism, leaving out actions which had no effects on the lock. With the law of effect, [Thorndike] theorized that “[r]esponses that produce a satisfying effect in a particular situation become more likely to occur again in that situation, and responses that produce a discomforting effect become less likely to occur again in that situation” (cited in [Gray, 2010], p. 108f.).

[Skinner, 1938] generalized this principle into the operant conditioning model. It describes behavioral learning as internalizing patterns of stimulus – response – outcome. A learner who reacts to a stimulus in a certain way observes the (desirable or undesirable) outcome of his response, increasing or decreasing the probability that he will later show the same response to a similar stimulus. Desirable outcomes are positive reinforcement (e.g. giving the learner food) or negative reinforcement (e.g. switching off a loud noise), while undesirable outcomes are positive punishment (e.g. hurting the learner) or negative punishment (e.g. not giving the learner dessert) ([Zimbardo and Gerrig, 2003], p. 219ff.). Learning by operant conditioning relies on consistent outcomes: if reinforcement is omitted or reversed several times for a given response, then the learned stimulus-reaction-outcome pattern is discarded (extinction).

There is strong evidence that behavioral learning is most effective when the reinforcement immediately follows the response. Based on previous literature, [Neuringer] concludes: “The delay is found to weaken responding: rates of responding are lower than when reinforcement immediately follows a response, pauses are longer, new responses and discriminations take more time to learn, and a delayed reinforcement is less likely to be chosen than an immediate one” ([1969], p. 375). The same is true for negative outcomes: punishment (should it be unavoidable) is most effective when immediately following the undesired reaction ([Walters and Grusec, 1977], cited in [Zimbardo and Gerrig, 2003], p. 223).

While both Thorndike and Skinner originally researched animal behavior, they later applied their insights to human learning (Thorndike).
Skinner’s programmed instruction paradigm was realized with so-called teaching machines, which allow students to exercise on their own. A sequence of brief chunks of information is shown to the learner, each followed by a question. The learner must enter his answer and is allowed to proceed only after answering correctly. Skinner claims that “the machine, like any private tutor, reinforces the student for every correct response, using the immediate feedback not only to shape behavior most effectively but to maintain it in strength in a manner which the layman would describe as ‘holding the student’s interest’” (1968, p. 39, cited in Kulhavy, 1977, p. 213).

Skinner’s equalization of immediate feedback and reinforcement is controversial. Foppa criticizes that the behaviorist conception of feedback was based only on a vague analogy between conditioning and learning (1968, cited in Musch, 1999). In fact, operant conditioning deals with learning sequences of behavior, rather than with gaining knowledge. Kulhavy and Wagner note that “it seems difficult, if not impossible, to translate operant principles directly into an instructional system that works well with classroom students” (1993, p. 10). The way feedback is given in programmed instruction has also been criticized. Various studies found out that learning scenarios based on the law of effect “made little difference in terms of student learning” (ibid., p. 9). One reason for this is that students, because they know that they will receive the correct answer after the response, will put little thought into their answer (ibid., p. 9). Furthermore, even when assuming that immediate feedback works as a positive reinforcement, learning tools based on Skinner’s ideas offer little value above that. “Although correct behaviour was reinforced, incorrect responses, and even minor errors, such as misspellings or correct semantic substitutions, could not be dealt with because no diagnostic, explanatory or remediatary strategies existed in such systems. Further, there was no opportunity for reflection and intervention on the part of the student” (Ravenscroft, 2001, p. 5).

Many of those problems have been resolved in more recent exercise tools which support closed or semi-open questions. For instance, Bacon describes a system which is “designed to help students who answer incorrectly, by recognising the mistake made (where possible) and then providing constructive targeted feedback and another try” (2011, p. 1). For open-ended tasks, on the other hand, immediate feedback is less effective. Such tasks are more complex and usually consist of several solution steps. Thus, assessment can already be performed on an intermediate solution. However, interrupting the learner to give him immediate feedback might unexpectedly distract him from the solution process itself. Furthermore, while immediate feedback makes trial-and-error possible, it prevents higher-order forms of learning from mistakes. This is unfortunate because learning from mistakes is an important factor to make feedback effective (Hattie and Timperley, 2007).
2.3. Timing of Feedback

With immediate feedback, learners cannot learn to detect their own errors (Corbett and Anderson, 1991, p. 2), and because problems are reported immediately, learners never experience the consequences of mistakes on the following solution steps.

Error flagging is a weakened variant of immediate feedback which can be used for open-ended tasks. It works similarly to the way spelling mistakes are highlighted in a word processor. The ACT Programming Tutor (see section 3.1) is a programming tutor system which supports this principle:

“In the Error Flagging condition, the tutor responds immediately to feedback, but does not interrupt students. It ‘flags’ an error by displaying it in bold on the computer screen, but does not provide any explanatory text. Students are free to go back and fix the error, go back and ask for a comment on the error, or to continue coding” (Corbett and Anderson, 1991, p. 2).

To some extent, this alleviates the problem that immediate feedback prevents learners from experiencing how mistakes affect the following solution steps. However, like traditional immediate feedback, it does not foster independent thinking, as there is no chance for a learner to detect mistakes on his own.

2.3.2. Delayed Feedback

The delayed feedback condition is the default in traditional assignment correction: students hand in their solutions on paper, and the teacher returns the annotated submissions after he has finished correcting them. In this scenario, the delay exists not for didactical reasons, but because it is physically impossible for a teacher to give immediate feedback to each individual student, let alone to students who are working on their assignments at home. The same is, of course, true for exams: only afterwards can students receive feedback in the form of a grade, points, or annotations.

Besides these practical issues, there are also didactical reasons to prefer delayed to immediate feedback. Most of the theories which endorse delayed feedback are justified by the delay-retention effect (Brackbill et al., 1962), which says that “learners who receive immediate knowledge of the correct responses, or feedback, retain less than learners for whom feedback is presented after a period of delay” (Kulhavy and Anderson, 1972, p. 505). One generally accepted hypothesis to explain this phenomenon is the interface-perseveration theory: “learners forget their incorrect responses over the delay interval, and thus there
Chapter 2. Feedback Theories

is less interference with learning the correct answers from the feedback. The subjects who receive immediate feedback, on the other hand, suffer from proactive interference because of the incorrect responses to which they have committed themselves” (ibid., p. 506).

In an influential meta-analysis, KULIK and KULIK (1988) looked into different studies which compared immediate and delayed feedback. They analyzed 53 evaluations with learners from different class levels studying different topics. In the immediate feedback condition, students received feedback directly after each test item or after finishing the test. In the delayed feedback condition, feedback was given after intervals which ranged from few seconds after each test item to up to one week after the test. KULIK and KULIK conclude that on the one hand, delayed feedback is superior under certain experimental conditions: “When test-item stems are used as the stimulus material and the correct answer is the response to be learned, delayed feedback is reliably superior to immediate feedback” (ibid., p. 80). They attribute this to the interface-perseveration theory of KULHAVY and ANDERSON (1972). On the other hand, these circumstances are not necessarily given in practice, and immediate feedback was found to be more effective in applied studies using classroom quizzes (KULIK and KULIK, 1988, p. 93). The meta-analysis led to the conclusion that “delayed feedback appears to help learning only in special experimental situations and that, more typically, to delay feedback is to hinder learning” (ibid., p. 94).

Despite the practical advantages of immediate feedback reported by KULIK and KULIK, one should note that the studies they analyzed only dealt with closed questions (e.g. multiple-choice tests) or semi-open tasks (e.g. list learning). It is unclear how the findings translate to open-ended tasks. Similarly, KULHAVY and ANDERSON (1972) exclusively considered multiple-choice tests. Above that, immediate feedback has some inherent disadvantages (see section 2.3.1). Luckily, the distinction between immediate and delayed feedback is not discrete. There are several approaches for a compromise between helping learners during their solution process, and respecting student autonomy. Some of those strategies are discussed in the following two sections.

2.3.3. Alternative Push Strategies

Immediate feedback can be considered a push strategy: instead of letting learners choose when to access feedback, the assessor actively delivers it as soon as an answer is given. Certain realizations of delayed feedback can also be regarded as push strategies, for instance when the results of a test are automatically shown to the learner as soon as the evaluation process has finished. Besides these widespread push strategies, there are several others which fall into this category.
2.3. Timing of Feedback

Push strategies which lie between immediate and delayed feedback report problems immediately, but only under certain conditions. At first sight, one might think that such a strategy was in line with findings of behavioral research, namely with the partial reinforcement effect (BIT-TERMANN [1975]). In operant conditioning, partial (or intermittent) reinforcement means that the desired response is only reinforced during some rounds: either in fixed intervals or following a variable pattern. “The Partial Reinforcement Effect states that responses acquired under schedules of partial reinforcement are more resistant to extinction than those acquired with continuous reinforcement” (BOYD 2002, p. 111). TAYLOR and NOBLE describe a multiple-choice apparatus in which “knowledge of results (feedback) was given only on those trials pre-determined by a particular reinforcement schedule” (1962, p. 32). However, as already argued in section 2.3.1, it is a fallacy to equate reinforcement and feedback, as learning behaviors is not the same as acquiring knowledge. Furthermore, intermittent feedback is unpredictable for the learner; this inconsistency makes it hard to understand how to use the feedback system. Modern literature on feedback timing does not deal with partial reinforcement schedules anymore.

ALEVEN and KOEDINGER describe a learning application that “initiates help after two errors” (2000, p. 303). They do not motivate this intermittent feedback approach with the partial reinforcement effect. Instead, they argue that, even though students should be left in control as far as possible, some of them would not access feedback at all without a push strategy (ibid., p. 293). Their two-errors approach implies that all errors are equally grave. In contrast to this assumption, however, there may be learning applications which detect not only critical errors, but also suggest minor stylistic improvements. In such a case, two stylistic remarks would be pushed to the learner (interrupting his solution process), but a single, grave error would not. Hence, even though the two-errors rule is easy to implement, it is not applicable for learning tools with diversified error types.

BROOKHART describes methods for effective teacher feedback, including guidelines for the timing of feedback (2012, p. 228f.):

- “Give immediate feedback for knowledge of facts (right/wrong).”
- “Take a bit more time to allow for more comprehensive reviews of student thinking and processing.”
- “Never delay feedback beyond when it would make a difference to students.”
- “Give feedback as often as is practical, and definitely for all major assignments.”
Unfortunately, these guidelines are rather imprecise – for instance, it is unclear how delaying feedback could not in any way “make a difference to students”, and thus would be acceptable. While the guidelines might be a valid description for the behavior of an experienced teacher, they are too inaccurate for a computer to decide on the timing of automatic feedback. This limits the usefulness of these guidelines in e-learning.

BLIKSTEIN (2011) suggests developing metrics which help finding patterns in the learners’ interaction, and use this information for purposes of assessment. “The proposed metrics can be calculated during the programming assignment and not only at the end, so instructors and facilitators could monitor students in real time and offer help only when the system indicates that students are in a critical zone” (ibid., p. 115).

Actually, finding patterns in the assessee’s behavior has been the goal of intelligent tutoring systems (ITS) since that field of research has been established by ANDERSON et al. (1982). The idea behind ITS’s is to automatically trace the knowledge of the learner (CORBETT and ANDERSON, 1995) and to offer learning tools which can dynamically adapt themselves to the learner’s knowledge level and learning style. This is supposed to bring computer-based learning systems closer to the kind of support that a human teacher can provide.

While knowledge tracing mechanisms in intelligent tutoring systems have the theoretical potential to detect the optimal moment to push feedback, this approach is very ambitious. Among the challenges are:

- The knowledge tracing system needs to be implemented, including a persistence layer which stores the learner’s profile in a way that does not conflict with privacy requirements. There must be an application programming interface which makes it possible for learning applications to report relevant learning activities to the knowledge tracing system, and for that system to influence the feedback mechanism.

- Developers of learning applications must realize different types of feedback for different learner types. For each feedback message, the degree of detail must be adapted to the perceived knowledge level of the student. Therefore, for each possible error, several different feedback messages need to be formulated.

- The feedback requirements of a learner depend on his level of knowledge. This level can change during a solution process. Because an ITS adapts to the user’s perceived level of knowledge, it is hard for him to predict its behavior. From a human-computer interaction (HCI) perspective, such inconsistencies make a user interface difficult to use (RASKIN, 2000).
2.3. Timing of Feedback

- Modeling the student’s knowledge is a very difficult task. The same is true for modeling different learning styles. In 1986, YAZ-DANI stated: “There does not seem to be an agreement on an architecture for ITS” (p. 43). Despite the fact that ITS research has now been ongoing for over 30 years, there is still no reusable, domain-independent framework for learner knowledge tracing which is capable of deciding when and how to push feedback. Thus, model and knowledge tracing components have to be re-implemented for each learning application.

- The learner’s level of knowledge and learning style must be determined by the his interaction with the ITS itself. This means that during the first usage of the tool, the default behavior of the ITS will most likely not fit the learning style of the user.

The ITS research community is aware of these challenges. “Some researchers have become dissatisfied with the ITS approach to programming (as well as for other domains) claiming that student modeling is impossibly difficult, that students are arbitrarily unpredictable and that it is more productive to put research energy into the design of tools and environments with good HCI characteristics than into Intelligent Tutoring Systems since the immediate pay-off is much greater” (DU BOULAY, 1992, p. 37). Nevertheless, the ITS research community is still active (the well-established International Conference on Intelligent Tutoring Systems takes place regularly), and it is possible that in the far future, ITS mechanisms will help finding the optimal moment to push feedback.

2.3.4. Feedback on Demand

The feedback-on-demand principle is fundamentally different from all the strategies discussed in the previous sections. Unlike immediate feedback and similar push strategies, feedback on demand is a pull strategy. Analogous to the shift of learning architectures from knowledge-push to knowledge-pull (NAEVE, 2005), offering pullable feedback is more learner-centric, as it requires learner initiative. The goal is to “let students control and organize their own learning processes; the system should intervene as little as possible” (BURTON and BROWN, 1982, cited in ALEVEN and KOEDINGER, 2000, p. 292). The feedback-on-demand principle enables learners to request feedback on the current state of their solution at any time during the solution process. The assumption is that students would ask for feedback “in two kinds of situations: when they made an error which they could not

---

7 The ACT-R cognitive architecture is a widely accepted model based on Bayesian knowledge tracing (BAKER et al., 2008). However, there are only few applications which realize it in practice.
fix quickly, and on steps where they had little idea how to proceed” (ALEVEN and KOEDINGER 2000, p. 298).

**Constructivist approach**

Unlike immediate feedback, which is derived from behaviorist learning theories, feedback on demand follows a constructivist approach. According to constructivism, learning processes should be active and self-directed (HUANG, 2002), rather than guiding the learner through the solution by immediately pushing feedback. It is important to note that, even though the learner sets the pace, the tutor is still free how to guide him. The tutor will “leave decisions about when to seek help to the learner. The tutor then decides what help to provide and at what depth” (WOOD and WOOD 1999, p. 155). Feedback does not act as reinforcement, but as information – a principle derived from cognitivist theories (MUSCH 1999).

**Learning from mistakes**

Another important mechanism of constructivist learning is reflection (KOLB and FYR 1975). Even though learning from mistakes (in the sense of trial-and-error) is also possible in behaviorist learning environments, the immediate feedback prevents learners from experiencing and reflecting on the consequences of faulty solution steps (see section 2.3.1). Feedback on demand does not have this limitation: “This condition may be superior to the standard condition [i.e. immediate feedback] if students benefit from detecting their own errors” (COR-BETT and ANDERSON 1991, p. 2).

**Risks and opened-up opportunities**

Feedback on demand is not a panacea for the question of feedback timing, as it is not suitable for all learning contexts. “The risk here is that, by leaving help seeking to the learner, the additional cognitive demands, together with the potential threat to self-esteem, could act as impediments to learning” (WOOD and WOOD 1999, p. 155). On the other hand, WOOD and WOOD highlight the possible benefits of on-demand feedback, especially for experienced students. They hypothesize that “higher achievers […] are likely to perform better under tutoring both because they start with a more robust knowledge of the domain, and because they are better able to help the tutor to create a learning environment which is contingent on their needs” (ibid., p. 155). The question remains how to encourage weaker students to make adequate use of the feedback functionality.

**Do students know when they need help?**

ALEVEN and KOEDINGER (2000) raise similar objections, asking whether students have the required metacognitive skills to know when they need help. They conclude that many students refrain from requesting help even though they would probably benefit from it. However, this finding mostly concerns the glossary feature, a static reference which ALEVEN and KOEDINGER call “unintelligent help” (p. 293). Dynamically generated feedback was requested more often: “The students used the intelligent help facility on 29% of the answer steps and 22% of the explanation steps” (ibid., p. 297). Furthermore, it is important
2.3. Timing of Feedback

<table>
<thead>
<tr>
<th>Test 1</th>
<th>Immediate Feedback</th>
<th>Error Flagging</th>
<th>Demand Feedback</th>
<th>No Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Correct</td>
<td>67%</td>
<td>62%</td>
<td>67%</td>
<td>51%</td>
</tr>
<tr>
<td>Errors</td>
<td>5.9</td>
<td>6.6</td>
<td>4.3</td>
<td>10.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test 2</th>
<th>% Correct</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>45%</td>
<td>33.2</td>
</tr>
<tr>
<td></td>
<td>49%</td>
<td>22.1</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>19.5</td>
</tr>
<tr>
<td></td>
<td>34%</td>
<td>52.6</td>
</tr>
</tbody>
</table>

Table 2.1.: Results of the CMU Lisp Tutor post-tests (CORBETT and ANDERSON, 1991). For each feedback condition used during the practice phase, the table shows the percentage of correct solutions and the mean number of errors for that group.

To note that the participants of the study were school students. It is possible that university students, who have more experience with self-determined learning, know better when to request assistance.

The feedback-on-demand principle was first evaluated using the CMU Lisp Tutor, “an instructional program that assists students as they complete Lisp programming exercises” (CORBETT and ANDERSON, 1991, p. 1). “In the Feedback on Demand condition, the tutor takes no initiative in providing help to the student. Instead, at any time, the student can ask the tutor to check over the code. If an error is found, the tutor provides the same feedback message that the standard tutor would have presented automatically. If no error exists, the student is informed accordingly” (ibid., p. 2). They performed an experiment to compare the usage and effectiveness of immediate feedback, error flagging, feedback on demand, and no feedback at all. On average, students who got feedback on demand took longer to complete the exercises than those who received immediate feedback or error flagging. From a behavioral perspective, taking longer is a bad thing; but that is not necessarily the case from a constructivist point of view, as longer periods of self-directed learning are regarded as more effective than shorter periods of guided learning. However, CORBETT and ANDERSON remain skeptical whether the extra time expended leads to the development of additional skills (ibid.).

To operationalize the learning effect, CORBETT and ANDERSON (1991) conducted a post-test during which the students were on their own. They analyzed its results grouped by the feedback condition which the students had used while practicing. Looking at the percentage of students who successfully finished the post-test, it is clear that any form

\footnote{The students were paid to take part in the experiment (CORBETT and ANDERSON, 1991). In a field study, it would be unethical to deny some of the students feedback because they would have a disadvantage compared to their fellow students.}
Chapter 2. Feedback Theories

Immediate Error Demand No Feedback Flagging Feedback Feedback
Test 1 Evaluation
0.33 0.40 0.40 0.60
Debugging
0.30 0.50 0.27 0.57
Coding
1.00 0.98 0.58 1.83

Test 2 Coding
4.16 2.76 2.44 6.58

Table 2.2.: Results of the ACT Programming Tutor post-tests (Corbett and Anderson, 2001). Mean number of errors per answer in the paper and pencil tests.

Ten years later, Corbett and Anderson (2001) conducted a follow-up study with the ACT Programming Tutor, a progression of the CMU Lisp Tutor. Similar to the original evaluation, students were split in four groups, each with a different feedback condition. The results of the completion time comparison resembled those of the 1991 experiment. The paper-based post-tests featured evaluation, debugging, and coding tasks. Again, the generally positive impact of feedback was found, as “students who received feedback conditions were reliably more successful than the no-tutor students on the Test 1 coding problems […] and on the Test 2 coding questions” (Corbett and Anderson, 2001). On average, students made less mistakes after practicing with feedback on demand than those who had practiced with other feedback conditions, but this finding should be treated with caution because the “effect of tutor version was marginally significant in a two-way analysis of variance” (ibid., p. 249).

Musch (1999) notes that constructive learning systems are more challenging to develop because of the additional degrees of freedom. This general assumption is also true for applications with on-demand feedback. Because learners are allowed to continue after making a faulty solution step, the learning application must be able to behave in a reasonable way even when the learner is working on a partially wrong solution. Moreover, it is not enough to develop a generic submission mechanism which learners can use to request feedback. Staljohann et al. (2009) developed a system which allows students to submit intermediate solutions to the LMS for the purpose of requesting formative
feedback. During a survey, only 6% of the course participants stated that they had used this feature to receive feedback; most explained it with the time expenditure for having to upload the solution to the LMS multiple times (ibid., p. 292). This could be alleviated by integrating the feedback mechanism directly into the learning applications, even though this may be harder to implement than a generic LMS-based feedback mechanism. In general, feedback on demand should be as accessible as possible to encourage learners to make use of it whenever necessary.

2.4. Explicitness of Feedback

According to Kulhavy and Stock (1989), instructional feedback messages contain two separable components, namely verification and elaboration. Verification is merely the information whether the learner’s answer is correct. Anything above a “yes-no” or “right-wrong” flag is regarded as elaboration. Saul and Wuttke describe elaboration as “the informational component providing relevant cues to guide the student toward a correct answer” (2011, p. 4).

NarciSS (2006) conducted a meta-study on feedback classification schemes. She comes to the conclusion that there is a broad consensus when it comes to the classification of simple feedback types. These include components such as knowledge of response (an indicator whether the learner’s answer was correct or not) and knowledge of the correct result (the disclosure of the correct solution). These types of feedback are well-established for closed exercises such as multiple choice tests. For complex, open-ended tasks, she suggests elaborated feedback as a more appropriate response. Her definition of elaborated feedback differs from the definition of elaboration by Kulhavy and Stock (1989), who would already classify knowledge of the correct result as elaboration.

Musch (1999) states that, in most empirical studies, elaborated forms of feedback are shown to lead to better results than feedback which merely informs about the correctness of the given answer. He argues that feedback is effective not because of its reinforcement characteristic (which is already fulfilled by simple verification), but because its elaboration helps to correct mistakes (ibid.).

Elaborated feedback can subtly point to an aspect which needs improvement, or explicitly name the steps which the learner should take next. Shute distinguishes between directive and facilitative feedback: “Directive feedback is that which tells the student what needs to be fixed or revised. Such feedback tends to be more specific compared to facilitative feedback, which provides comments and suggestions
Chapter 2. Feedback Theories

to help guide students in their own revision and conceptualization [9] (2008, p. 157). On the question how much guidance a learning environment should provide, [MERRILL et al., 1992, p. 289] note that there needs to be a trade-off between preventing students from becoming lost and frustrated, and still leaving them in control.

Another classification of elaborated feedback is given by [CLARIANA, 2000], who distinguishes between explanatory (why the response is incorrect), directive (how to improve the solution), and monitoring (how the student is doing overall) feedback. From this and other research contributions, [NARCISS, 2006, p. 21ff] derives five categories of elaborated feedback:

**Knowledge on task constraints (KTC):** Advice on type of exercise, assignment rules, subexercises, and requirements.

**Knowledge about concepts (KC):** Clues, explanations, and examples for technical terms and their contexts.

**Knowledge about mistakes (KM):** Number of mistakes, position, type, and cause of the mistakes.

**Knowledge on how to proceed (KH):** Specific advice how to fix each mistake, general advice how to solve the exercise, advice on strategies, guiding questions, solution examples.

**Knowledge on meta-cognition (KMC):** Advice on meta-cognitive strategies, meta-cognitive guiding questions.

Scenarios which combine classroom teaching with technology-enhanced learning are called blended learning [BONK et al., 2005]. In most such scenarios, the knowledge about concepts and knowledge on meta-cognition is provided mainly during lessons or lectures. Knowledge on task constraints is provided by the assignment sheets. This leaves the task of offering knowledge about mistakes and knowledge on how to proceed to the learning technology. This thesis will focus on these two types of elaborated feedback. Assistance on how to proceed can further be separated into feedback and hints.

---

9 SHUTE, 2008 attributes this distinction to [BLACK and WILLIAM, 1998]; however, the terms directive and facilitative feedback do not appear in that contribution. Nevertheless, this terminology is established in teacher guidelines on feedback.

10 Blended learning scenarios which make use of intelligent tutoring systems (ITS, see section 2.3.3) are an exception, as ITS’s also try to track the learner on a meta-cognitive level.
2.4. Explicitness of Feedback

2.4.1. Hints

According to Clariana (2000), hints are a form of directive feedback, besides prompts and cues. Staljohann (2012, p. 18), on the other hand, states that hints can be distinguished from positive and negative feedback. Indeed, positive feedback complements the learner on expeditious solution steps, and negative feedback points him toward mistakes that have been made, whereas hints tell the learner how to fix these reported mistakes, or how to proceed with his solution. Hence, hints are not feedback which reflects on the learner’s previous steps, but rather feedforward in the sense of Hattie and Timperley (2007), telling the learner what to do next. In the classification of Narciss (2006) described in the previous section, hints fall into the category “knowledge on how to proceed”.

Many of the aspects of feedback which have been discussed earlier in this chapter can analogously be considered as aspects of hints. Learners can receive hints from different sources and at different times. The hint on demand pattern (Zimmermann et al., 2011) describe a pull strategy for hints which is supposed to help out students who are in danger of losing motivation because they are stuck. The pattern describes scenarios with on-demand hints coming from tutors, from peers, or from a computer. Of course, combinations of these hint sources are possible as well, which leads back to the idea of semi-automatic assessment.

2.4.2. Cheating Prevention

Providing learners with very detailed problem description texts and explicit hints leads to the risk that they overuse the feedback system. Formative assessment is supposed to assist the learner, but “by adopting the strategy of simply having the system guide one through the solution process, a subject does not learn much” (Shute et al., 1989, p. 264). Baker et al. summarize such strategies under the term “gaming the system” (2004b). They report that students who were “taking advantage of properties and regularities in the system” (ibid., p. 532) while using an intelligent tutoring system learned less than those who used the feedback mechanism as intended. Similar observations are reported by Alevén and Koedinger (2000).

Among others, the following approaches (or a combination of them) can reduce the likelihood that an automatic feedback system will be abused.

Note that the term “feedforward” is ambiguous: Hudspeth (1993) uses it to describe information which is provided to the learner before he starts working on the exercise.
Vague feedback: One way to foster independent thinking is to reduce the explicitness of feedback messages. This is especially important for hint texts, which is why Shute advises: “Avoid using progressive hints that always terminate with the correct answer” (2008, p. 179). In general, one can say that facilitative feedback is less prone to being used for cheating than directive feedback, as it is more vague. Unfortunately, it is often harder to implement an intelligent assessment system which yields vague hints than one that offers the student explicit instructions on how to continue. In an exercise for which a computer can efficiently calculate the next step toward a correct solution, the hint has to be blurred artificially to make it more vague.

Even when feedback and hint messages are less detailed, they may make it possible to solve the exercise by brute force. In an exercise type with a limited number of possible actions, it is possible to solve the exercise by randomly or systematically trying out all possible actions until no more mistakes are reported by the feedback system. This practice would, of course, also minimize the learning outcome. Furthermore, it is possible for students to get frustrated because the vague feedback is insufficient to help them achieve a correct solution.

Finally, even though facilitative feedback is more vague, completely dropping directive feedback does not resolve the issue. “Although hints can be facilitative, they can also be abused, so if they are employed to scaffold learners, provisions to prevent their abuse should be made” (Shute 2008, p. 179). This means that vague feedback should be combined with other countermeasures against cheating.

Overuse detection: It is possible to implement a function which determines whether the student is using the feedback system much more often than expected, and which restricts access to it in this case. For example, one could set a minimum time which must pass between demanding two hints (this makes cheating more time-consuming, but not impossible) or limit the number of hints available for each exercise. A similar approach is based on “associating some cost with using hints too frequently” (Shute et al. 1989, p. 265), for example by reducing the score or grade each time the learner makes use of the hint functionality. Such systems are described by Bacon (2011) and by Alevin and Koedinger (2000). Still, this can only be done in scenarios which involve

---

12 To prevent students from becoming stuck due to overly vague feedback, it is possible to realize multiple levels of hints (Moore et al. 2004). Students who do not understand vague feedback can then request more directive instructions. However, Alevin and Koedinger (2000) have found out that students often abuse such systems by clicking right through to the most directive hint.
2.4. Explicitness of Feedback

scoring or grading student performance – which is not always true for formative assessment. A more sophisticated, probabilistic approach to detecting overuse of an intelligent tutoring system is presented by [BAKER et al. (2004a)]. When taking hints leads to a penalty, students need to be informed about this regulation. This is important to satisfy [NICOL and MACFARLANE-DICK’s first principle of good feedback practice, namely to “clarify what good performance is” (2006).

Tutorial supervision: Another approach to prevent cheating is to provide the tutor with log files for each solution so that he can identify students who overuse the feedback system. This requires a logging module which records all uses of the feedback system and brings each requested feedback message in context with the current state of the solution, so that the teacher can judge whether taking feedback was appropriate in that situation. However, these checks are laborious for the tutor, and using the logs for purposes of surveillance may conflict with the learners’ privacy. Students should be informed beforehand that their solution processes are being tracked if their grades depend on their use of the hint functionality.

Hide on activity: For some types of exercises, computers can efficiently evaluate the current solution state in the background and generate feedback texts continuously. This makes it technically possible to refresh the displayed feedback message automatically whenever the learner modifies the solution. However, such an implementation facilitates the undesired brute-force approach: the learner can try out all possible solution steps until the reported problem vanishes. This problem can be attenuated by hiding the feedback message automatically when the learner modifies his solution in a way that affects the problem which is being displayed. It requires the user to perform a certain action to reopen the feedback display. However, a student who is willing to expend a lot of effort would still be able to use such a feedback system for trial and error.

Self-determined learning: Ideally, the learners are intrinsically motivated to solve the exercise, and therefore are not inclined to abuse the feedback system in the first place. Positive feedback on a student’s achievements can motivate him during an assessment activity (GARRISON and ANDERSON, 2003; HORTON, 2006), but the student already needs a level of self-determination even before starting the exercise to prevent him from abusing the feedback mechanism. Inducing motivation in learners is not an easy task, and is indeed a research field on its own [13].

[13] See, for example, the ARCS Model of Motivational Design (KELLER, 1999).
When comparing these approaches to cheating prevention, it becomes clear that there is no simple solution which completely rules out the possibility of overusing the feedback system. For instance, the vague feedback approach bears the risk of not providing the learners with enough details to enable them to solve the exercise. ALEVEN and KOEDINGER (2000) tried to compensate for this by designing a progressive hint model in which students were first given a vague hint, but could request more detailed hints when needed. In their evaluation, however, it turned out that students did not follow the desired help-seeking strategy. “When the students requested help, they requested to see all hint levels (i.e., they asked for more help until they had seen the bottom out hint) on 82% of answer steps and 89% of explanation steps” (ibid., p. 298). To be effective, countermeasures against overuse of the feedback system should combine several of the listed approaches.

2.5. The Feedback Loop

As already explained in section 2.1, formative assessment involves qualitative feedback to the learner which is “intended to modify his or her thinking or behavior for the purpose of improving learning” (SHUTE, 2008, p. 154). The actions of the learner and the corresponding feedback, along with the modified behavior of the learner, form a feedback loop (SADLER, 1989). In other literature, this process is referred to as the assessment cycle (WILIAM and BLACK, 1996). It consists of eliciting evidence of the learner’s performance, interpretation of this evidence, and a consequent action taken by the teacher and/or by the learner (ibid.). This action can lead to new evidence, which causes the feedback loop to restart.

According to the definition of SHUTE (2008), feedback which a student receives after finishing an assignment can be regarded as formative if it helps him to perform better in following assignments. In such a case, the feedback loop spans an interval of about one week. However, such a learning scenario does not exploit the full potential of formative feedback. “Unless students are able to use the feedback to produce improved work, through for example, re-doing the same assignment, neither they nor those giving the feedback will know that it has been effective” (BOUD, 2000, p. 158, cited in NICOL and MACFARLANE-DICK, 2006, p. 213). If learners receive accompanying feedback already during their solution process, the interval of the feedback loop is shortened, allowing more iterations and thus more opportunities for reflection and improvement.

In the context of assessment, the term feedback usually refers to information which flows from the assessor to the learner. However,
2.6. The Role of Learning Analytics in Assessment

Bescherer and Spannagel (2009, p. 432) note that teachers also have a demand for feedback. They may want to improve their teaching, e.g. by discussing common mistakes in class. Likewise, Angelo and Cross (1993) note that an important part of the feedback loop consists of feedback which teachers receive from students on their learning. While this is a given when the teacher is the assessor, it is harder for him to get an insight of student learning when assessment is performed automatically. Systems like Moodle are capable of showing an overview of student scores for summative purposes, but do not show common mistakes or misconceptions, making it hard for teachers to improve their teaching.

One way for teachers to get feedback on their teaching is through course evaluation. This is done by handing out questionnaires at the end of the semester, with which students rate various aspects of the teacher’s performance and of the course (Aleamon and Spencer, 1973). The goal is to make it possible for the teacher to spot and improve upon weaknesses, e.g. by comparing his results to those of colleagues. Unfortunately, this method has a very long feedback loop of one semester. Furthermore, it is very coarse-grained, as students merely give ratings averaged over the entire semester. To get a more substantive insight, it is necessary to observe and analyze learning processes in detail. Lately, analytics-based research has aimed at filling this gap.

2.6. The Role of Learning Analytics in Assessment

The concept of “analytics” originates from the domain of business intelligence. Analytics combines means of computer science, operations research, and statistics with the goal of inferring useful information from large data sets (data mining). Current research addresses the question how to use the principles of analytics in the domain of e-learning, or education in general.

Academic analytics has the goal of helping institutions of higher education in the procedure of decision-making, e.g. forecasting the future demand for courses or revealing enrollment trends (Goldstein and Katz, 2005). “Academic analytics can be thought of as an engine to make decisions or guide actions. That engine consists of five steps: capture, report, predict, act, and refine” (Campbell and Oblinger, 2007, p. 3). On a smaller scale, e.g. on course level, the tasks of predicting, acting, and refining can be left to a teacher without any automation. For capturing and getting reports of the students’ interactions, however, one needs technological assistance.

More recently, there has been a shift toward analyzing progression of smaller groups of learners. Research in this area forms the new field of learning analytics. According to Eliasi, its focus “appears to be on the selection, capture and processing of data which will be helpful for students and instructors at the course or individual level” (2011, p. 4). Siemens defines learning analytics as “the use of intelligent data, learner-produced data, and analysis models to discover information and social connections, and to predict and advise on learning” (2010). While it has a different goal, the methodologies of learning analytics resemble those of academic analytics, and the five-steps model of Campbell and Oblinger (2007) bears similarity to the process of learning analytics envisioned by Siemens (2010) which is shown in figure 2.5.

Figure 2.5.: The process of learning analytics envisioned by Siemens (2010).

One important disparity lies in the time dimension: while academic analytics supports long-term decision-making, “learning analytics is focused on building systems able to adjust content, levels of support and...
2.6. The Role of Learning Analytics in Assessment

other personalized services by capturing, reporting, processing and acting on data on an ongoing basis in a way that minimizes the time delay between the capture and use of data” (ELIAS, 2011, p. 4). This requirement makes it necessary to concentrate on data which can be collected continually during the semester (such as log data), rather than summative data (such as exam results). Because results and log data from formative assessment activities can improve instruction (ALAMUTKA and JARVINEN, 2004), they are highly relevant for the purpose of learning analytics. However, even though much research has recently taken place in learning analytics (see CHATTI et al., 2012, for an overview), only few of those research contributions also deal with formative assessment.

There are few assessment tools which support learning analytics. The ASSISTment system combines summative assessment and learning activities. It logs “how many scaffolding questions have been done, the student’s performance on scaffolding questions and how many times the student asked for a hint” (FENG et al., 2009, p. 249). The teacher plays a passive role while students work on semi-open tasks, and uses the tool afterwards to get an overview of individual students’ performance. BLIKSTEIN collects logs of students who are working on open-ended programming projects, and analyzes them trying to “infer patterns in how students go about programming [...] as well as detect critical points in the writing of software in which human assistance would be more needed” (2011, p. 111). This idea already points to a semi-automatic assessment process; however, analysis of the collected log files is still labor-intensive for the teacher. BLIKSTEIN regards his contribution as an “initial step” (2011, p. 115) toward monitoring and assisting students in real time while they are working on assignments.

In the ASSISTment system as well as in the scenario described by BLIKSTEIN (2011), the teacher can see each solution step and knows who did it at what time. FENG et al. emphasize this as an advantage of the system: “By clicking the student’s name shown as a link in our report, teachers can even see each action a student has made, his inputs and the tutor’s response and how much time he has spent on a given problem. The ‘Grade Book’ is so detailed that a student commented: ‘It’s spooky, he’s watching everything we do’” (2009, p. 250). But even though the quoted student explicitly expressed discomfort with this lack of privacy, the issue is discussed neither in the contribution at hand nor in a follow-up paper (FENG et al., 2011). In general, one has to say that questions of privacy do not get the attention they deserve in learning analytics. This disregard of privacy aspects is especially problematic when the analyzed data includes private communication. For instance, the LOCO-Analyst system (see section 3.3) makes it possible for the teacher to find out “to whom and about what [the students] talked” (FENG et al., 2009, p. 245).
in chat rooms or discussion forums and how their messages are related to the learning content being taught” (JOVANOVIC et al., 2008, p. 57).

CAMPBELL and OBLINGER note that “data collected and analyzed in an academic analytics project might be protected by federal, state, and institutional privacy regulations” (2007, p. 5). Those institutional regulations in particular can be hindering when exchanging learning applications between universities which have different policies. Research on the impact of learning analytics on privacy is sparse, and while BORCEA et al. (2006) mention the fact that few students are aware of privacy risks, it is safe to assume that many teachers and developers of learning technologies are not familiar with privacy regulations either. CAMPBELL asks: “What will be the reaction [of students] to ‘big brother’ collecting data?” (2012, sl. 6). This makes clear that privacy is not only a legal requirement, but also a factor in the acceptance of learning technologies by students.

Data reduction and data economy: pseudonymization

One guideline to privacy compliance is the German concept of “Datensparsamkeit und Datenvermeidung” (data reduction and data economy, BIZER, 2007). It mandates to record no personally identifiable information, or only that which is absolutely essential to the task at hand. This contradicts the standard data mining approach of collecting all available information to check for possible correlations later. One possible compromise lies in pseudonymization: by saving activities of each learner under a pseudonym (DYCKHOFF et al., 2011), the data may cease to be personally identifiable, in which case it is no longer subject to the strict privacy regulations.

Scope of this thesis regarding learning analytics

Only parts of the learning analytics process by SIEMENS (2010) (see figure 2.5) are addressed in this thesis, namely Learners off-put data, Intelligent data, and Analysis. The remaining steps depend on creating a learner profile. How to do this in a way that does not conflict with learner privacy is an open research question which is out of scope of this thesis. On the other hand, the step Learners off-put data can be performed in a much more detailed way than suggested in previous literature. CAMPBELL and OBLINGER (2007) and SIEMENS (2010) suggest measuring learning activity by tracking LMS logins and session lengths. Above that, it is possible to collect detailed information on the students’ assignment solution processes. As this data need not be personally identifiable, it is unproblematic from a privacy point of view. Integrating this detailed information into the feedback loop would make it possible to realize effective semi-automatic feedback.
3. Related Work

This chapter gives an overview of related software from the four categories shown in table 1.1: model tracing, intelligent tutoring systems, tutor consultation, and learning analytics. Due to the wide range of existing systems, only a selection from each category can be described here. Intelligent tutoring systems and learning tools which offer on-demand feedback via model tracing are summarized in the first section. After that, section 3.2 deals with tools which involve the tutor in the assessment process. Finally, the last section of this chapter discusses systems which enable learning analytics by recording interactions with a learning tool.

3.1. Automatic Assessment

Systems which provide automatic assessment are available for nearly all domains. In general, one can say that tools for closed and semi-open exercises (see chapter 2) are less complex than those which assess open-ended tasks. The reason for this is that more degrees of freedom lead to more diverse solutions. Closed and semi-open exercises, by their definition, have only one correct solution, which means that assessment can be done by performing a simple comparison. This is why, despite their shortcomings, closed and semi-open exercises are still widespread in computer-aided assessment.

One example for a tool which provides automatic feedback on closed and semi-open tasks is HotPotatoes ([WINKE and MACGREGOR] 2001). It makes it possible to define multiple-choice tests, matching exercises, and fill-in-the-blank texts, and to present them to the learner via a web interface. The Moodle system also supports these question types, but goes one step further: it offers an exercise type in which the student enters a short free text, which is then matched to a teacher-defined pattern that may include wildcards ([JORDAN and MITCHELL] 2009). Because these short-answer exercises do not rely on a static sample solution, they can be considered open-ended tasks. Still, this type of automatic assessment is very primitive. It cannot be regarded as model tracing in the sense of [ANDERSON and PELLETIER] (1991) see section 2.2 because the system has no understanding of the addressed concept. It
only takes the final result into account, not the learner’s reasoning or solution path.

The ASSISTment system, which also supports closed and semi-open tasks, makes it to some extent possible to define the semantics of an exercise. When a student gives a wrong answer, the system breaks the task down into subproblems by presenting him “scaffolding questions” (FENG et al., 2009). Like the original question, these are closed or semi-open questions. The assessment functionality of the ASSISTment system does not follow the model tracing approach. Instead, the creator of an exercise statically defines which wrong answer should lead to which scaffolding question. Students who answer the original question correctly are not confronted with scaffolding questions; in fact, they need not and cannot provide their solution path at all.

The FORMID project (GUÉRAUD et al., 2006) aims at helping teachers to synchronously monitor the progress of distance-learning groups. The FORMID system comprises three tools: Author, Learner, and Observer. Students who are using the FORMID-Learner component can request automatic validation of their intermediate solutions. Like those of the ASSISTment environment, the automatic assessment capabilities of FORMID are not based on the principle of model tracing, but on comparison of the current situation with states set by the teacher. For each exercise created with FORMID-Author, the teacher has to define which states are correct and which ones contain typical errors. Additionally, he must indicate which states should be reported as a correct, finished solution. Such a system with static tests is easier to create than a system which dynamically evaluates the solution based on model-tracing algorithms. However, each exercise variation requires the teacher to redefine correct and wrong states. Furthermore, it limits the students to those actions which have been foreseen by the teacher, and thus leaves no room for extraordinary solutions.

Besides those tools with static assessment capabilities, there are countless systems which are capable of automatically assessing open-ended tasks. Most of these deal with topics such as mathematics, computer science, or electrical engineering because most tasks in those disciplines are subjective, unlike those in the humanities. Praktomat (KRINKE et al., 2002) is an example for an assessment service for computer science, more specifically for Java programming. It is a web service which allows students to upload the source code of their solution so that it can be automatically evaluated. Assessment is done by performing several tests, including style check, compilation of the source code, and checking the output when running the byte code with different parameters. “Programs that cannot be compiled or fail the test are rejected” (ZELLER 2000, p. 90). Besides automatic assessment, Praktomat also supports peer feedback; this will be described in section 3.2.
Another assessment system for open-ended tasks is *Euclid Avenue* (LUKOFF, 2004), which deals with mathematics. It allows students to create propositional calculus proofs which are evaluated automatically. Like *Praktomat*, it performs the evaluation on solutions which students have submitted. This is delayed feedback as described in section 2.3.2 because it is unavailable during the solution process. LUKOFF reports that most solutions handed in by the students were either fully correct or worthy of no credit at all, i.e. completely wrong or incomplete (2004, p. 43). It is possible that students in such situations would perform better if they had the chance to already receive feedback during their solution processes.

The *EASy* system (GRUTTMANN et al., 2008) is different from *Euclid Avenue* because it provides feedback already while the student is working on the task. In *EASy*, it is the learner’s task to do mathematical proofs. The system evaluates the solution while the student is working on it. But although it features sophisticated model tracing mechanisms to verify the mathematical correctness of the student’s steps, its feedback capabilities are very limited: It immediately tells the student about mistakes, but is unable to give hints on what to do instead.
Such hints are available in the OUNL Exercise Assistant (Gerdes et al., 2008). It is an online learning tool on the topic of Boolean algebra, more specifically on transforming terms into disjunctive or conjunctive normal form (DNF or CNF). Like EASY, it uses model tracing for assessment, which means that it can evaluate any solution, not only those expected by the developers and teachers. This fully automatic assessment is possible because there are efficient algorithms to transform any term to DNF or CNF (Heeren et al., 2010). On each transformation entered by the learner, the Exercise Assistant immediately gives feedback (see figure 3.1). Hints are available on demand; the system is clearly intended for self-determined learners, as it incorporates none of the cheating prevention mechanisms discussed in section 2.4.2. The same is true for the SKA system, which also deals with DNF and CNF transformation (Schulz-Gerlach and Beierle, 2006). What all those tools have in common is that they need to have a very sophisticated model tracing engine because they are supposed to fully automatically assess any solution which a student might come up with.

The described model tracing systems generate feedback based on the current state of the learner’s solution. They do not take additional information into account, e.g. personal attributes of the student, or his performance during previous exercises. This is exactly what intelligent tutoring systems try to achieve, as explained in section 2.3.3. One of the most influential ITS’s is the CMU LISP Tutor (Corbett and Anderson, 1991), later known as LISPITS (Corbett and Anderson, 1992) and as the ACT Programming Tutor (Corbett and Anderson, 1995). It is a practice environment for the Lisp programming language. Like the systems discussed in the previous paragraph, the ACT Programming Tutor has a model tracing engine to assess the student’s solution. It is realized by a set of language-specific rules which together form an “ideal student model” (ibid., p. 256). This model is then used to perform knowledge tracing, an attempt to automatically estimate whether or not the learner has successfully learned a rule (as opposed to guessing correctly despite not having learned the rule). In accordance to the ACT-R theory of skill knowledge (see section 2.2), this evaluation is done probabilistically based on a Bayesian computational procedure (Corbett and Anderson, 1995). The calculated value is used as the mastery criterion: if the probability that a student has learned a rule is at least 0.95, he is allowed to proceed with a task which involves other rules (ibid., p. 261).

The PACT Geometry Tutor (Aleven et al., 1998) is another ITS which uses model tracing to assess solutions to open-ended tasks. It provides a graphical user interface with which learners can perform deductive proofs in Euclidean geometry (see figure 3.2). One important difference to the ACT Programming Tutor is that it does not give feedback

---

1 Newer versions also support Prolog and Pascal (Corbett and Anderson, 1995).
3.1. Automatic Assessment

immediately after a mistake has been made, but only on request of the learner. The evaluation by ALEVEN and KOEDINGER (2000) has shown that, while not all students made sufficient use of the feedback-on-demand functionality, the dynamically generated feedback is preferred to static references (see section 2.3.4). On the other hand, a similarity to the ACT Programming Tutor is that the PACT Geometry Tutor calculates conditional probabilities to trace the knowledge of each student. “The information in the student model is used to select appropriate problems and to advance the student to the next section of the curriculum at appropriate times” (ALEVEN and KOEDINGER 2000, p. 293). Thus, knowledge tracing information is not applied while the learner is working on a section, but only between them. So far, there are no ITS’s which use the information collected by knowledge tracing to determine the ideal time to push feedback.

Due to the vast number of available tools which support automatic assessment, only a small selection could be discussed in this section. An overview of further tools for automatic assessment of programming tasks is given by DOUCE et al. (2005). Intelligent tutoring systems are compared by YAZDANI (1986) and by NWANA (1990). In conclusion, one can say that automatic assessment systems for open-ended tasks are more effective for learning than those for closed and semi-open exercises. It does, however, require a much greater effort to develop them.

Further tools for automatic assessment

Figure 3.2.: A geometric proof in the PACT Geometry Tutor (ALEVEN et al., 1998).
3.2. Semi-automatic Assessment Systems

In comparison to the number of tools which support fully automatic assessment, there is far less software which follows the blended-learning approach of combining automatic and tutorial feedback. This is due to the tradition of intelligent tutorial systems which, since the *teaching machine* of *PRESSEY* (1926, see section 2.2), were supposed to replace the human tutor instead of complementing him. While the endeavor to reduce teacher workload by computer-aided assessment is reasonable, one should keep in mind that human tutors are still superior to computers for many types of open-ended tasks. For instance, automatically generating feedback on written text is still a novel approach (LIU and CALVO, 2012), and CAA research has a long way to go to match the performance of human tutors when it comes to unforeseen solutions. This section highlights some systems which follow a semi-automatic approach to assessment.

![Figure 3.3: The architecture for assessing Java exercises in DUESIE (translated from HOFFMANN et al., 2008).](image)

The German acronym DUESIE stands for “Das Uebungssystem der Informatik Einführung” (“The Exercise System of the Computer Science Introduction”) (HOFFMANN et al. 2008). The system deals with programming exercises and can handle Java source code as well as UML diagrams. Similarly to the Praktomat environment described in the previous section, it offers a web interface through which students can upload Java source code, which is then compiled and tested (see figure 3.3). DUESIE is not an e-learning platform, but a system which automates evaluation and assessment of exercises as far as possible (ibid., p. 178). Despite their attempt to automate the evaluation process, the developers of DUESIE are aware that the system might fail to assess a submission. In this case, a tutor or lecturer can subsequently override the automatic assessment by manually adjusting the evaluation (ibid., p. 182). In comparison, Praktomat does not offer tutorial feedback; instead, it
has a peer review mode in which learners can critique the programming style of their fellow students (Zeller 2000).

The ALOHA system (Ahoniemi and Reiniänen 2006) is another assessment environment for programming tasks. Like DueSIE and Praktomat, it offers dynamic tests. Its tutorial feedback capabilities, however, exceed those of DueSIE by allowing tutors not only to tweak the automatic evaluation, but to give elaborated feedback on aspects which cannot be assessed automatically. The focus of ALOHA is on consistent and objective grading. This is supported by a so-called rubric-based grading scheme. "The idea of rubrics is to divide the grading into small enough parts so that each part can be objectively graded following given instructions" (Ahoniemi and Karavirta 2009, p. 333). For example, there are rubrics to grade code layout, variable naming, and commenting. For each rubric, there are reusable feedback phrases which the tutor can select and, if necessary, edit (see figure 3.4). Ahoniemi and Karavirta (2009) call this process “semi-automatic phrasing”. This approach acknowledges the fact that most correct solutions follow an expected path, and most wrong submissions contain typical mistakes.

All the tools discussed so far are specialized on a particular domain. With the so-called Correctionflows, Ställjohann et al. (2011) suggest a domain-independent, modular alternative based on the workflow concept. In this model, the assessment process is split up into evaluation steps. They are connected with transitions, conditions, loops, and parallelizations to form a workflow. This means that the execution of the workflow can depend on the outcome of an evaluation step. The steps are parameterizable – for instance, the expected output of a unit test can
be defined as a parameter. This makes it possible to reuse a step in another workflow by using different parameters. One special evaluation step is a tutor-in-the-loop component which interrupts the correction-flow until a tutor has looked at it and given feedback \( \text{[ALTENBERG-GIANI et al., 2008]} \). So far, only a prototypical, SharePoint-based implementation of the Correctionflows model exists, the eAixessor.NET system \( \text{[BOLLING, 2010]} \).

3.3. Interaction Analysis

In order to give useful feedback, the teacher should not only look at the final solution, but at the entire solution process. This enables him to investigate the cause of a mistake, rather than merely stating that the mistake exists. To view the solution process, teachers need technological support. Students can use screen recording software to capture their solution processes, and send the video files – so-called screencasts – to their tutors. There is a wide range of both proprietary and open-source systems capable of recording a video of either the entire screen or a single application. These include Camtasia Studio and Adobe Captivate. A comparison of these and other systems is given by \( \text{[SPANNAGEL, 2007]} \).

Abdel Nabi and Rogers (2009) conducted a study in which students recorded annotated videos of their assignment solutions and uploaded them to a learning management system. Tutors could watch these videos to assess the students’ “procedural competency or conceptual understanding” (ibid., p. 25), rather than just the correctness of their outcome. Nevertheless, one has to note that, for exercises which take several minutes to solve, such videos become very large files. This means that it takes a lot of time to encode and to upload them. In the study of Abdel Nabi and Rogers, students had three weeks to work on the exercise and finally had to record only a single video. If one wanted to offer semi-automatic feedback during the entire solution process, the encoding and uploading requirements would become major drawbacks. Furthermore, video files do not carry any semantic information, which means that the only way for a tutor to assess them is for him to take his time and watch them. This thwarts large-scale quantitative analysis, and thus makes screen recorders unsuitable for the objectives of learning analytics.

The FORMID project, which has already been mentioned in section 3.1, provides a monitoring tool, FORMID-Observer. It is based on the Monitoring Distance Learning Activities (MDLA) model \( \text{[DESPRES, 2003]} \). Even though this is a model for synchronous monitoring, the software also includes a mode for asynchronous analysis. “One original feature of FORMID-Observer is that learning traces can be synchronously
observed or chronologically replayed to be later analysed” (GUÉRAUD et al., 2009, p. 22). This “replay”, however, cannot be compared to that of screencasts. The logging mechanism of FORMID-Observer does not allow the teacher to see the student’s view. Instead, it records “semantic events” which can give him an insight of “learner achievements and difficulties” (ibid., p. 21). Each recorded event represents a user action which has affected the solution process. The teacher takes a passive, monitoring role while using FORMID-Observer; there is no backchannel which could be used to directly contact the students.

As learning management systems (LMS) such as Moodle or Blackboard Vista are well-established in universities and other educational institutions, there is an ongoing endeavor to use LMS log data for the cause of learning analytics. Some LMS’s are shipped with built-in logging support, e.g. Moodle. However, LIBBRECHT et al. point out that “their logging capabilities in terms of collecting and analysing usage data is very limited” (2012, p. 114). They state that these systems only provide counters and tabular representations of learners’ page impressions, which is insufficient to get an insight of an individual solution process. More detailed logging mechanisms, such as the interactions report of SCORM 3, are limited to closed and semi-open exercises.

Besides those logging functions which are built into a particular LMS, there are systems that are compatible with a wider range of LMS’s. ZORRILLA and ÁLVAREZ state that their system MATEP (Monitoring and Analysis Tool for E-Learning Platforms) can collect logs from any LMS as long as they “follow the W3C specification” (2008, p. 612). Unfortunately, MATEP is restricted to collecting page impressions, i.e. information which page was loaded from which IP address at what time (HACKELÖER, 2011, p. 6).

The LOCO-Analyst system, which is directed at content authors and teachers, goes one step further. Besides page impressions, it can process assessment results such as a student’s score on a multiple-choice quiz, as well as information on discussions which have taken place in a chatroom. For each quiz, the teacher provides semantic annotations, e.g. to define that it deals with the contents of a certain lecture or reading. Based on these annotations, techniques of learning analytics are applied to try and determine reasons for good and bad performance in quizzes (JOVANOVIC et al., 2008). Feedback is automatically generated from the findings of this process, and teachers can retrieve it.

---

4 ZORRILLA and ÁLVAREZ (2008) do not state explicitly which W3C specification is meant, but it is likely to be the Extended Log File Format, which is currently a W3C Working Draft: http://www.w3.org/TR/WD-logfile.html (accessed 2012-08-31).
from a repository. Even though students are represented by generated pseudonyms⁵ a teacher may (inadvertently) identify a student by the logged chat messages. On the other hand, the pseudonymization prevents the teacher from contacting individual students in case he notices unfavorable learning behavior: the system, like FORMID-Observer, does not provide a backchannel to enable tutor-student communication. The views of LOCO-Analyst are integrated into the Reload Content Packaging Editor⁶ a tool which is directed at teachers, not students. For this reason, there is no possibility for students to get an insight into how their interaction and communication is being logged. Because The Reload Editor complies with the IMS Content Packaging specification (Ali et al., 2012), LOCO-Analyst is restricted to logging performance with closed questions such as multiple-choice quizzes, and cannot analyze student interaction with open-ended tasks.

Pardo and Delgado Kloos (2011) criticize learning analytics approaches based solely on the logs of LMS because “a significant part of the interaction during a learning experience is beginning to take part outside of the LMS”. In order to take all learning-centric interaction into account, they suggest a logging solution based on virtualization. Students are provided with images of preconfigured virtual machines, which they can set up on their personal computers. Certain user actions inside the virtual machine – such as opening web pages, or invoking internal commands in certain tools – trigger events that are stored in a folder which is shared with teammates and instructors. While this approach is capable of capturing learning activity outside of the LMS, it has three disadvantages. Firstly, students have to download and set up the virtual machine. Even though this might be easier than installing and configuring multiple applications required for the course, it is still inconvenient for students, especially compared to using learning tools which are deployed as web applications. Secondly, logging all interactions which take place inside the virtual machine bears the inherent risk of privacy violations. Even though the virtual machine was “portrayed to the students as the application to use when working on the course material”, it is possible for a student to use it for private activities. In such cases, it is difficult for instructors not to inadvertently breach the students’ privacy. Thirdly, asking the students to use the virtual machine for all course-related activities bars them from using their familiar learning environment (e.g. their web browser of choice).

All these disadvantages can be avoided by deploying learning tools as web applications, and providing an online logging infrastructure. T-Prox, the “Tracking Proxy”, is an attempt to realize such a logging system. Originally designed to support usability studies of web interfaces (Lilienthal 2008), it could also be used in education. T-Prox adds JavaScript code to each transferred HTML page. This code uses event

---

⁵ This can be seen in figure 2 and 3 of Ali et al. 2012.
⁶ Reload Editor: http://www.reload.ac.uk/editor.html (accessed 2012-08-31).
3.3. Interaction Analysis

Figure 3.5.: Processing of a request in T-Prox (LILIENTHAL 2008, p. 35):
1. Client requests web page
2. Proxy requests page
3. Server delivers page
4. Proxy inserts JavaScript
5. Proxy stores page
6. Proxy sends modified page to the client
7. Client sends notifications about user actions to the proxy
8. Proxy stores user actions

listeners to capture user actions such as mouse clicks. Each event is forwarded to the T-Prox server where it is stored for later analysis (see figure 3.5). One downside of T-Prox is that students must configure their web browsers to reroute all connections through it (and must remember to remove this proxy configuration after their learning activity). This is not only inconvenient, but leads to the danger of unintentionally logging private web browsing. Above that, T-Prox is limited to logging static web pages, and its records do not carry the semantics of the interaction. GURSCH (2010) briefly describes possible T-Prox extensions for AJAX-supported websites, but interaction with Flash applications, Java applets, and similar content is impossible to track due to the nature of T-Prox.

When comparing the discussed interaction analysis tools, there is a division between tools which capture user interactions in great detail, and those which record few, distinctive events that are supposed to carry the basic semantics of the learning process (see figure 3.6). Primitive LMS-based logging approaches are an exception here: they merely log that a file has been accessed, but neither record the interaction with this file, nor process the semantics of this interaction. Screen recorders, while recording each mouse movement, convey no semantics at all. The only way to interpret screencasts is to watch them in full length, which is inconvenient for tutors and makes it impossible to automatically generate data for learning analytics. On the contrary extreme, the LOCO-Analyst system provides semantic representations of learning contents, but only records final scores of closed tasks, not detailed solution processes of open-ended exercises.

The comparison has shown that there is a lack of tools which both gather the semantic information required for learning analytics, and
capture enough detail to allow an in-depth analysis of an individual solution process. The semantic event approach of the LOCO-Analyst and FORMID tools is promising, but should be extended to open-ended tasks. Furthermore, a backchannel should be provided so that the tutor can not only passively observe, but actively offer feedback and assistance.
4. Conceptual Model

As described in chapter 1, the fully automatic approach of intelligent tutoring systems requires an enormous effort to develop autonomous systems for model and knowledge tracing. This chapter will present an alternative by combining semi-automatic feedback on demand with techniques of learning analytics (see table 1.1). This model for feedback incorporates several principles which have been deliberated on in chapter 2. These principles are briefly recapitulated hereafter:

- Feedback on open-ended tasks
- Semi-automatic assessment
- The feedback loop
- Integrated feedback mechanism
- Evaluation of solution processes
- Feedback on demand

The review of literature the previous chapters has shown that assessment of closed and semi-open tasks is already a well-researched field. There are countless tools for both summative and formative assessment which support this type of tasks, and there are only few aspects which could still be improved in well-established formats like multiple-choice questions. When it comes to open-ended tasks, on the other hand, there are fewer studies on effects of feedback, and fewer tools which facilitate assessment. This is unfortunate, as open-ended tasks encourage students to show skills which are dispensable for closed tasks. These abilities include reasoning and communication (Cai et al., 1996), solving complex problems, and applying one’s knowledge in real-world contexts (Shepard, 2000).

One reason why automatic assessment of open-ended tasks is less common is the effort which is required to develop a system that can generate appropriate feedback in any situation. In some cases, it is even technically impossible to realize an automatic solution checker due to current limitations in artificial intelligence, e.g. in natural language processing (Liu and Calvo, 2012). On the other hand, tutors and
teachers often do not have the time to give all students feedback in the timely and detailed fashion that is required for formative assessment. A compromise with semi-automatic feedback (see section 2.2) is possible: The computer automatically generates feedback on standard solutions and typical mistakes, and refers the learner to a tutor for all other approaches.

Feedback loop

The analysis of related work in chapter 3 has shown that only few systems support semi-automatic assessment. The DUESIE system is one of those; however, its semi-automatic functionality is mainly intended to improve the automatic grading decisions, not to affect learning (Hoffmann et al., 2008). Formative assessment, unlike its summative counterpart, is based on a feedback loop, which should affect learning. As this loop is missing in DUESIE, that system realizes a summative form of semi-automatic assessment. Formative feedback should follow the principles of good feedback practice (Nicol and Macfarlane-Dick, 2006), which include encouraging dialog between teachers and students.

Integrated feedback mechanism

The exercise module of the $L^2P$ learning management system[1] does not support automatic feedback, but offers formative tutorial feedback. Nevertheless, students rarely requested and received preliminary feedback on intermediate solutions (Stalljohann et al., 2009). The main reason that prevented students from requesting feedback was the additional time needed for repeatedly submitting intermediate solutions (ibid., p. 292). This problem could be counteracted by integrating the feedback mechanism directly into the learning applications, thus removing the need to save the solution, switch to the learning management system, and upload it there. Of course, it is harder to integrate the feedback system into each learning application than it would be to rely on a generic upload mechanism. Modularization and abstraction of common functionalities (e.g. presentation of feedback and hints) can reduce the workload for these implementations.

Evaluation of solution processes

Most systems which assist tutors in giving feedback expect them to evaluate solutions only based on the current state of the submitted version. Any preceding steps which led to this solution are neglected in the assessment. Techniques of learning analytics can be applied to evaluate solution processes. So far, learning analytics has seen the teacher in a relatively passive role during the learning process: he acts as an observer, and only later reflects the findings, either alone or with the student (cf. Feng et al., 2009). In the sense of formative assessment, it would be useful for students to receive feedback from their teacher already during the solution process.

---

Finally, feedback should not hinder self-directed learning by unnecessarily revealing parts of the solution to the learner. Feedback on demand (see section 2.3.4) is a pull strategy that, unlike immediate feedback and other push strategies, respects the student’s autonomy by leaving the feedback timing decision to him. However, as described in section 2.4.2, mechanisms have to be put in place to prevent an overuse of the feedback functionality.

4.1. The Tutor-in-the-Loop Model

The tutor-in-the-loop model is a novel approach which combines semi-automatic assessment with solution process analysis to enable formative feedback on open-ended tasks. The name of the model is derived from the human-in-the-loop model, a general principle which describes how a person can override artificial intelligence judgments (cf. Zaidan and Callison-Burch 2009). The term “tutor-in-the-loop” was first used by Altenberd-Giani et al. (2008). In their contribution, the term denotes one specific component of an assessment correction flow (see section 3.2), not a complete model for formative semi-automatic feedback.

As can be seen in figure 4.1, the tutor-in-the-loop model is based on five interconnected modules.

Electronic exercises: By the nature of computer-based assessment, the learners’ solutions must be available in an electronic format. This means that learners either have to work on a computer, or digitize their solution papers to submit them. In the tutor-in-the-loop model, learners are supposed to receive feedback already during their solution process. This rules out the paper-based alternative.

The question how to represent exercises from different domains on a computer is challenging. On the one hand, it is easier to develop algorithms for automatic assessment when the learner obeys a strict format for his solution. On the other hand, overly harsh limitations might prevent the learner from deriving from the standard solution path, thus making it impossible to try out unorthodox approaches. Therefore, electronic exercises in the tutor-in-the-loop model must strike a balance by following a format which can (at least partially) be evaluated by a computer, while still allowing the learner to try and solve the exercise in a way which was unforeseen by teachers.
Intelligent assessment: The intelligent assessment component is responsible for automatically evaluating the learner’s (intermediate) solution and generating appropriate feedback. The term intelligent does not imply that the automatic assessment module must be capable of evaluating all aspects of any solution. The learning application should, however, know its limitations, so that it can advise the learner to seek feedback from his tutor when necessary. This is an important requirement for semi-automatic assessment.

In order to qualify as intelligent assessment, feedback messages have to be dynamically generated to take aspects of the learner’s solution into account. This separates intelligent feedback from static help functions such as glossaries (ALEVEN and KOEDINGER, 2000). An exercise whose assessment relies solely on static information cannot be considered an open-ended task. To validate various aspects of the solution, the computer needs to induce an abstract model of it in order to run tests on that model (model tracing, see section 2.2).

Feedback on demand: The purpose of this component is to wait for feedback requests from the learner, then to trigger intelligent assessment of the current solution state and to show the assessment.
results to the learner. For cases in which the feedback is insufficient to tell the learner how to proceed, a more explicit and directive message should be available; these hints, however, should only become visible on request of the learner (hint on demand). Because of the explicitness of hints, countermeasures have to be taken to prevent their overuse (see section 2.4.2). In addition to the hints, there needs to be a function which allows the learner to request feedback from his tutor. This is especially important when the intelligent assessment module has detected a solution which it cannot evaluate fully automatically.

In most learning applications, the contents are primarily depicted visually, especially textually. Following the modality principle [MAYER 2006], feedback messages should then be presented on the auditory channel in order not to interfere with the contents perceived on the visual channel. However, there are multiple downsides of audio feedback. Firstly, it restricts the context in which the application can be used, e.g. by forcing the user to wear headphones in classroom. Secondly, the sequential and volatile nature of audio messages makes it difficult for learners to recapitulate parts of the message which they did not understand immediately. Thirdly, and most importantly, recording speech is much more laborious than writing texts. This is especially significant because the intelligent assessment component generates feedback dynamically, making it hard or even impossible to prerecord all possible variants. Thus, textual feedback is a reasonable choice for the tutor-in-the-loop model. It can be augmented with graphical visualizations or visual highlighting.

**Interaction logging:** To enable the tutor to give adequate feedback, the system must provide him not only with the learner’s request and the current state of the solution, but also with information about the sequence of steps which led to that solution. This makes it possible to get an insight on the learner’s approach, and to detect possible misconceptions. Furthermore, the tutor can analyze which steps the learner has already tried, and what kinds of problems he ran into during these attempts. This allows the tutor to formulate feedback which is vague enough not to disclose the complete solution, but specific enough to enable the learner to go on.

To fulfill these requirements, the logging system must at least capture all interactions which led to a modification of the solution, and all feedback and hint requests. The protocol of each session has to be transferred to the tutor so that he can comprehend the solution process of the student who has requested help.
Analytics: In certain situations, it is useful for a tutor to not only look at the solution process of a single student, but to get an overview of the performance of the entire course. For instance, he might want to find out which mistakes were very common, or whether certain exercises were too easy or too difficult. To help the tutor answer these questions, a tutor-in-the-loop system should provide mechanisms to analyze and compare multiple sessions. Principles of learning analytics (see section 2.6) can be applied for this. Specifically, relevant information should be extracted from the gathered data, and displayed to the tutor in clear visualizations. One user-friendly form of presenting such information is the dashboard, which visualizes key indicators to support decision-making (Norris et al., 2008). Such a dashboard could enable the tutor to take the initiative and approach students who seem to need assistance. This can be a complement to the feedback-on-demand mechanism, in which the initiative comes from the students.

Visibility of the modules to the learner

Of the five described modules, the electronic exercises should be in the learner’s focus. Shifting one’s locus of attention (Raskin, 2000, p. 17ff) away from the exercise itself should only be necessary when requesting feedback – and even then, the exercise should remain visible so that the feedback messages can be comprehended in context. The intelligent assessment and interaction logging processes should run as transparent to the user as technically possible. Automatic assessment can be performed in the background when the user requests feedback, and only its results should become visible in the feedback on demand module, in the form of problem descriptions and hints. Similarly, the interaction logging process can start automatically when the learner starts working on an exercise, and can quietly run in the background.

Integration of the feedback module

From the learner’s point of view, the feedback module should not appear as separated from the learning application, but as a part of it. This tight integration of the feedback mechanism into the learning application is supposed to reduce extraneous cognitive load on the learner. Shortening the feedback loop also encourages him to make use of the feedback system whenever necessary. While the tutor request function is embedded in the feedback-on-demand component (and thereby integrated into the learning application), the response channel is not part of it. The reason for this is that it possibly takes a while until the tutor has noticed the feedback request and answered, and it is possible that the student is then no longer using the learning application. Therefore, an asynchronous medium of communication, such as e-mail, has to be used.
4.2. Research Questions

The main research question of this thesis is whether the tutor-in-the-loop model enables students to retrieve feedback on their individual learning processes. It is broken down into four subquestions.

Q1: *Is it technically possible to develop a framework for learning applications with semi-automatic feedback on open-ended tasks?* This includes realization of an intelligent assessment system, a modular feedback component, and a mechanism to log and analyze solution processes, as well as electronic exercises which serve as proofs of concept. While there are existing systems for many of those aspects, it remains to be shown that such components can be combined to provide semi-automatic feedback.

Q2: *Do students access automatically generated feedback on their own during their solution process?* Because the tutor-in-the-loop model follows the feedback-on-demand paradigm, its success depends on students requesting feedback. [ALEVEN and KOEDINGER (2000), see section 2.3.4] claim that this demand cannot be taken for granted at least for static help functions. It remains to be evaluated whether students actively request dynamic feedback.

Q3: *Do students contact their tutors when the automatically generated feedback is insufficient?* This does not mean their final submission, but requests for formative feedback during the solution process. A lack of awareness that help is required, as considered in research question Q2, would also mean that students would not contact their tutors. The evaluation of this thesis is supposed to show whether this or any other reason prevents students from requesting tutorial feedback.

Q4: *Can tutors extract information from the logs to provide useful feedback.* The tutor-in-the-loop model features two use cases for which the tutors requires logs of solution processes. Firstly, when a student requests tutorial feedback, as described in research question Q3, the tutor can inspect the corresponding log to understand the cause of the student’s problem. Secondly, principles of learning analytics can be used to detect and highlight relevant facts about the course-wide performance. This information is supposed to help the tutor to detect weaker students so that he can offer them assistance.

This thesis deliberately omits any hypotheses on the learning effects of the tutor-in-the-loop model. There are several reasons for this. Firstly, when operationalizing learning effect by looking at exam grades, there
are many confounding variables which cannot be eliminated. These include, but are not limited to learner style, mood during the exam, and the effectiveness of other offerings such as books, lectures, or peer contact. Secondly, it is difficult to compose a control group (i.e. a group of students who may not access tutor-in-the-loop feedback). When two distinct courses serve as treatment group and control group, the confounding effects of teaching style, course strengths etc. cannot be controlled. On the other hand, when all participants are from the same course and groups are randomized to eliminate the confounding variables, the teacher deliberately withholds information from the control group students. This would give the students in the treatment group an unfair advantage in the exams, and is therefore unacceptable.
5. Technological Foundations

The only way to evaluate the research questions listed in the previous chapter is to develop a system which realizes the tutor-in-the-loop model. None of the systems regarded in chapter 3 fulfills the requirements of the model. However, this does not mean that a new system has to be developed completely from scratch. There are existing tools and libraries that can be used to develop components of the tutor-in-the-loop model. The Jacareto framework serves as the foundation of an interaction logging component. The Cinderella dynamic geometry system offers interactive visualizations which can be embedded into electronic exercises, and its geometry model can be used for the intelligent assessment component of some learning tools. Besides Jacareto and Cinderella, which will be described in this chapter, several open-source libraries such as JDOM and JUNG serve as technological foundations of the developed applications.

5.1. The Jacareto Framework

The acronym Jacareto\footnote{Jacareto: http://jacareto.sourceforge.net/ (accessed 2012-08-31).} stands for “Java Capture & Replay Toolkit” \cite{Spannagel2007}. It can be used to capture the user interaction with another application, hereafter referred to as the target application. The records can then be analyzed quantitatively and qualitatively \cite{Spannagel2003}. This is possible because, unlike the screen recorders discussed in section 3.3, Jacareto creates an event-based representation of user interaction. Because it relies on the event model of the Java platform, it is only suitable for applications which are programmed in Java and have an AWT or Swing graphical user interface (GUI). Programs developed for other platforms are not supported.

5.1.1. Capture & Replay Engine

Before being able to capture or to replay interaction with a target application, Jacareto needs a so-called ApplicationStarter. The starter contains all the information on the target application that is needed...
to launch it. This includes the fully qualified name of the main class, the class path (which includes the relative or absolute paths of all required libraries), and the command-line arguments (cf. Spannagel, 2007, p. 174f.). It is possible to define different command-line arguments for capturing and for replaying. The starter information can either be entered manually, or automatically extracted from the manifest of a JAR file. Starters can be saved in XML format to distribute them, however adaptations may be needed when transferring a starter file to another computer because path information can differ. The target application runs in the same Java Virtual Machine (JVM) as Jacareto, even though its classes are loaded by another Java class loader to avoid undesired side effects.

Capturing events

In Java AWT and Swing applications, all user interactions such as mouse movements and keyboard presses lead to the creation of an AWTEvent. Likewise, each action of a program window (e.g. a new window opening) is represented by a special AWTEvent. Internally, all events which occur are passed through the so-called EventQueue, from which they are distributed to the responsible listeners within the application. Interestingly, this event queue is shared by all applications which run within the same JVM. This makes it possible for Jacareto to register an AWTEventListener at the event queue. The listener gets notified of any interaction with the target application and propagates this information to Catchers. This mechanism is described in detail by Gursch (2010).

ComponentsManager

When the target application opens a new window, the ComponentsManager of Jacareto recursively analyzes its component hierarchy. This analysis is reperformed regularly to detect changes to this hierarchy. The components manager creates a path representation for each found component (Spannagel, 2003, p. 27f.). For example, the path string JFrame_(1).JRootPane_(1).JPanel_(3).JButton_(2) stands for the second button within the third panel in the content pane of the target application main window. This path is unique for each component of the target application (components of the same type which have the same parent component can be distinguished because their paths are numbered consecutively), and therefore is the same during capture and replay. This makes it possible to recognize the affected component of each event at replay time.

Recordables

For each captured event, the path of the affected component is recorded, along with metadata such as the coordinates of a mouse click.

---

2 The Abstract Window Toolkit (AWT) is the foundation of all GUI classes of Java Standard Edition.

3 Registering an AWTEventListener is possible for any Java application; however, in an unsigned Java applet, the SecurityManager prevents direct access to the event queue. This makes it impossible to use Jacareto in an unsigned applet.
or the code of a pressed key. All this information is kept in a data structure called a Recordable. As different types of events carry different metadata, there is a whole range of subclasses of Recordable, one for each supported event. Furthermore, there are special recordables for startup information, such as the application starter, the screen resolution, or the state of the caps lock key. It is important to note that a single user action can cause multiple events. For instance, when the user clicks on a “Close application” button, Jacareto will at least capture a mouse press, a mouse release, a button activation, and the closing of the application window.

All the captured recordables form the “record of user and program interaction or just interaction record. An interaction record is a linear data structure whose elements are sorted by their chronological order” (Schroeder and Spannagel, 2003). Because of the high sampling rate of the capturing mechanism, this linear record becomes very long and confusing (Spannagel, 2007, p. 177). To improve clarity, a linear record can be converted to a hierarchical structure. The structuring process is similar to the way a compiler parses source code to generate an abstract syntax tree (Schroeder and Spannagel, 2003). The result is a tree of StructureElements, with Recordables as leaves. For example, in the default structure, consecutive MouseEventRecordables which collectively describe a smooth movement of the mouse cursor are grouped in a MouseMotion structure element.

The record is not image-based (i.e. based on screenshots or videos), but symbolic. “Only symbolic formats allow for performing automatic analyses. Therefore, in [Jacareto] every action is stored as [a] XML element with attributes describing the action” (Schroeder and Spannagel, 2006, p. 259). For each recordable and each structure element type, there is a Converter class which allows the conversion to an XML element, and vice versa. These converters are relatively simple because recordables are usually flat data structures: all attributes are either strings or primitive data types.

Jacareto does not merely simulate replays; it actually performs the interaction on the target application. Using the starter information contained in the record, it can restart the application. One by one, events are reconstructed from the recordables and automatically performed on the GUI of the target application. The user can watch the replay being performed. Afterwards, the target application is in the same state as it was at the end of the capturing process. The user can then interact with it normally.

---

4 Technically, Recordable is a subclass of StructureElement. This made it possible to realize StructureElement as a recursive data structure.
Replay modes

In order to make the replay feature flexible and compatible with as many target applications as possible, Jacareto offers several different replay modes. The replay modes are distinguished by the way time is represented:

**Real time:** Each recordable contains the duration of the event, measured in milliseconds. In real time mode, which is the default, these durations are preserved so that the record is replayed at the same speed as it was captured.

**Fixed time:** The fixed time mode replays the record with a fixed (customizable) interval between events. In order to save time, events of lesser importance, such as mouse movements, are skipped in favor of more relevant events like mouse clicks. This results in a step-by-step playback of the recorded interaction.

Mouse modes

Furthermore, replay modes differ in the way in which they emulate mouse input:

**Real mouse:** Despite its name, this mouse mode does not access the physical mouse device. Instead, it uses the class `java.awt.Robot` to generate native system input events. This causes the mouse cursor of the operating system to move automatically. On the one hand, the target application has no chance to differentiate those generated events from real ones, therefore it will behave as if it were controlled by the real mouse. On the other hand, Jacareto is able to filter input from the physical mouse while the replay is running to prevent the user from interfering with the replay process (Spannagel, 2007, p. 186f.).

**Pseudo mouse:** This mode controls the application not by injecting events at system level, but by dispatching the events directly to the affected component. Because this has no influence on the system mouse cursor, an additional, fake mouse cursor is drawn on top of the target application, using Java Swing glass panes. (Ibid., p. 187)

Fast-forward

Sometimes it is not desirable to watch the replay in its full length. Of course it is possible to stop it after the part of interest. Nevertheless, it may take several minutes to get to that part from the start of the record in real time replay. To avoid this waiting time, it is possible to “fast-forward” to any recordable within the record. Fast-forwarding works similarly to the fixed time replay mode with an interval of zero milliseconds, in that minor events such as mouse movements are skipped. After the chosen recordable has been reached, the replay automatically continues at normal playback speed. Due to the nature of event-based
5.1. The Jacareto Framework

replay, there cannot be a rewind feature, as there is no way for the replacer to undo the actions which have been performed on the target application.

5.1.2. User Interfaces for Jacareto

*Picorder* is a deliberately minimal user interface for *Jacareto*. It features a small control center window containing buttons to start and stop capturing and replaying (see figure 5.1). The target application is specified by a starter file which is passed as a command-line argument. Due to its unobtrusive interface, *Picorder* is most suitable for scenarios in which the user is supposed to fully concentrate on the target application.

![Figure 5.1: The control center window of Picorder in capture mode.](image)

In cases in which even the minimal control center would be a distraction, the learner can use the command-line argument `-i` (for “invisible”) to completely hide the control center window and to let *Picorder* silently capture or replay in the background. Because it is not possible for the user to manually stop and thereby save the record in invisible mode, a shutdown hook\(^5\) is registered so that the record is saved automatically when the user closes the target application.

One specialty of *Picorder* is the `-rc` command-line argument, which causes it to first replay a given record. When the replay has finished, the user is given control of the target application, while his interactions are automatically captured in the background. All recordables are appended to the original record file, which is saved automatically as soon as the target application is closed.

*CleverPHL*\(^6\) is a much more feature-rich, graphical *Jacareto* user interface. Tasks like starting a target application or loading a record can be performed using dialog windows. The same is true for changing


\(6\) The acronym *CleverPHL* consists of the initials of capture, log, evaluate, visualize, edit, replay, and the abbreviation of Pädagogische Hochschule Ludwigsburg (Ludwigsburg University of Education), where the *Jacareto* project was initiated (SPAN-NAGEL, 2007, p. 171).
preferences such as the replay mode. The CleverPHL main window displays records as hierarchical, expandable structures (see figure 5.2). Furthermore, the window comprises an editor component which makes it possible to view and modify the attributes of the currently selected recordable or structure element.

![Image of CleverPHL main window](image)

**Figure 5.2:** The CleverPHL main window showing a session of the ColProof-M learning application. The left side shows the record structure, the right side contains the editor for the currently selected recordable.

### Sessions

One CleverPHL instance can open several sessions simultaneously. Only one session – the one currently being displayed in the CleverPHL main window – is active at a time, with a tabbed interface to switch sessions. Each session holds one record and has its own preferences. When the user saves a session, CleverPHL creates a a directory and fills it with several files. Besides the record file described in section 5.1.1, the directory will contain an XML file holding the preferences of the session, as well as a file with the list of known starters.

### Data sets

It is possible to review the entire record inside the CleverPHL main window, down to details such as the timestamp and coordinates of each mouse movement. Nevertheless, this is insufficient to perform quantitative analyses for the purpose of learning analytics (see section 2.6), as there is a lack of support for statistical analysis and visualization. Spannagel et al. (2005) suggest extracting so-called data sets, a table-like structure whose rows are referred to as data cases, from the record for the purpose of performing quantitative analyses. CleverPHL comes with a number of converters which turn recordables into data cases,
which then form a data set that can be viewed in a dialog window. Furthermore, it is possible to export data sets to CSV or XLS format in order to statistically analyze the data with software such as SPSS, Fathom, or Microsoft Excel [SPANNAGEL (2003) p. 30ff).

### 5.1.3. Modules

The *Jacareto* features described so far are generic, as they can be used with any target application. SPANNAGEL (2003, p. 79) points out that, in most cases, additional, application-specific classes are required. This is especially important because it makes it possible to include semantic information to a record by adding it to the event queue whenever the state of the target application is changed. These semantic events make it easier to interpret the record without watching the entire replay. Semantic, application-specific events can be added without modifying the *Jacareto* codebase, thanks to a module system (GURSCH 2010, p. 46f.). Target application modules depend on the *Jacareto* module base, a small JAR file which provides access to the event queue so that semantic events can be posted there (see figure 5.3).

![Diagram of the capturing process of Picorder.](image)

**Figure 5.3.** The capturing process of Picorder.

Besides event types, a module can contain several other classes which implement application-specific variants of *Jacareto* concepts, for example:
**Recordable:** Specifies how to represent the event in the record.

**XML converter:** Converts the recordable to XML, and vice versa.

**Event editor:** Displays event details in CleverPHL. In the editor of a semantic event, the attributes are usually read-only because redefining the semantics would make it inconsistent with the recorded interaction.

**Data set:** Makes it possible to represent certain event information in tabular form to ease statistical analysis.

All these files need to be added to a module JAR file, which must be made available to Jacareto. On startup, Picorder and CleverPHL scan a certain directory for such module files. Each module contains a configuration file which lists all recordables, XML converters, etc. This information is used to dynamically add all the classes of the module to the Jacareto environment.

### 5.1.4. Existing Use Cases for Jacareto

**GUI Testing for software quality assurance**

Jacareto was originally designed for the field of software quality assurance. Its original objective was to develop a tool which automates testing of graphical user interfaces. With Jacareto, a use case of the target application can be captured; after modifications of the source code, one can replay the record to check whether the changes have unintentionally affected the use case. The process can be further automated by inserting test elements into the record structure (BEnkert and Bois, 2003); for example, a test element can check whether a certain button is enabled at a specific time. This transfers the principle of automated unit testing to GUI testing.

**Prototyping: semantic events and data sets**

Another use case of Jacareto lies in the field of usability engineering. User interface design relies on an iterative development model in which GUI prototypes are rapidly created and then tested with potential users. During these user tests, the interaction needs to be diligently observed in order to find usability problems and resolve them during the next development iteration. Protoreto (Herding, 2008; Herding et al., 2009) is a prototyping software which makes it possible to use Jacareto during user tests. There is a Protoreto module which is responsible for adding semantic information to the record, e.g. by inserting a MisclickEvent whenever the user fails to click on a component (Herding, 2008 p. 102ff). In addition, the module contains converters for data sets which can help analyze the recorded data. For instance, one data set shows a chronological list of all prototype screens which have been visited, with durations indicating how long the user...
5.1. The Jacareto Framework

has stayed on each screen. Data sets containing semantic events could play an important role in realizing the analysis component of the tutor-in-the-loop model.

*Jacareto* has already been used in several different educational scenarios. For instance, *CleverPHL* has been used to record and analyze interaction with a microworld environment. In the so-called *Number Game*, first graders are challenged to find the position of given numbers on a number line. Because the line is only partially labeled, students have to estimate the position ([KLAUDT] 2003). In order to analyze the children’s strategies of estimation, interaction with the number line is recorded with *Jacareto*. This makes it possible to classify strategies such as interpolation, counting, or doubling distances ([MÜLLER et al.] 2006; [KLAUDT and SPANNAGEL] 2004). The *Jacareto* module for the *Number Game* introduces several semantic events. The *NumberGameClick* event is triggered when the user has clicked on the number line, and keeps the affected number as metadata. Likewise, a *NumberGameMotionPause* event is casted when the cursor is moved without a consecutive click (i.e. a mouse motion followed by a 100 ms pause). Furthermore, the module defines structure elements which make it possible to analyze records on a coarse-grained level, rather than reviewing each piecemeal mouse movement (see figure 5.4). [KLAUDT and SPANNAGEL] (2004, p. 246) claim that one advantage of the module system is that it “decouples the qualitative and quantitative analysis mechanisms from the actual learning software and thereby makes them usable for the study of user behavior with other learning programs as well.” However, this reusability is limited to the *Jacareto* base system; application-specific parts still have to be redeveloped (ibid., p. 256). The domain-specific event types of the *Number Game* module cannot be reused for any other learning application that does not deal with number lines.

In another learning scenario, [SPANNAGEL and KORTENKAMP] (2009) recorded geometric construction processes in the *Cinderella* dynamic geometry system (DGS, see section 5.2). Semantic events provide *CleverPHL* with “information that is only known by the specialized tool DGS (e.g. the Euclidean coordinates of constructed points)” (ibid., p. 3f.). These semantic events can be used to automatically recognize different ways of constructing a perpendicular bisector for two given points, and to structure the record accordingly. Unlike the *Number Game*, there is a theoretical possibility to reuse parts of this *Jacareto* module with other DGS’s, as long as those are written in Java and have a similar style of interaction.

A third learning scenario is based on the cognitive apprenticeship model ([COLLINS et al.] 1989). This model is an approach toward a

---

7 My own translation.
Chapter 5. Technological Foundations

(a) Record before structuring (excerpt)

(b) Record after structuring (complete)

Figure 5.4.: CleverPHL showing an unstructured and a structured version of a Number Game record (Spannagel, 2007). Semantic events and structure elements starting with “Number Game” or “NG” belong to the module of the target application.

synthesis of schooling and apprenticeship. Its methods include modelling, scaffolding, and fading. “First, the teacher gives an example of how to solve a problem of a particular type (modelling) and externalizes appropriate solution strategies. Afterwards, students solve similar problems of reduced complexity. The teacher observes their solution
processes and gives feedback and hints (scaffolding). As the students become more and more experts, the teacher’s support is gradually reduced (fading)” (Schroeder and Spannagel, 2005, p. 100). Even though there are suggestions how to realize this scenario using Jacareto (ibid.), this setup has not yet been evaluated in practice.

Using Jacareto for the purpose of formative assessment has already been proposed in 2003:

“The student may also add his solution steps to the given record and send the whole record to an online judge who evaluates the transmittal. This way not only the product (solution to the programming task) but also the process of finding the solution can be evaluated. The same can be done in online assessments. Sometimes the solution process contains valuable steps which are not apparent in the result. Thus submitting the recorded solution process in addition to the result may help the evaluator to come up with a fair evaluation” (Schroeder and Spannagel, 2003, p. 2416f.).

However, like the cognitive apprenticeship suggestion, this proposal has never been realized with Jacareto before.
5.2. Cinderella

Due to the topics of the lectures in which the tutor-in-the-loop model could be evaluated, several of the learning applications which have been planned as proofs of concept deal with concepts of geometry. In order to supply users of such learning applications not only with static illustrations, but with interactive geometric constructions, a dynamic geometry system (DGS) was required. Because Jacareto is limited to the Java platform, this technology was mandatory for the proofs of concept, and thus for the DGS. From the range of available systems, Cinderella (Richter-Gebert and Kortenkamp, 2012) was chosen because it had earlier been proven to be compatible with Jacareto (Spannagel and Kortenkamp, 2009).

The major advantage of DGS’s like Cinderella is that they offer an environment which the user can interactively explore. Points and lines can be moved by simple drag and drop actions. However, there are learning scenarios which require limiting the degree of freedom to guarantee certain constraints, such as the parallelity of two lines or the perpendicularity of an angle. In Cinderella, elements which were constructed using such constraints keep these properties even when the user moves other parts of the drawing. This allows learners to safely explore a geometric construction by trial and error.

Applets

Due to the number of construction functions and other features that typical DGS’s offer, “the user may be overwhelmed by its complexity when searching for functions he wants to use” (Schimpf and Spannagel, 2010, p. 4). To avoid this cognitive overload, one can reduce the complexity of the application by removing or rearranging functions. This can be done by rewriting the DGS, or by using external tools such as CleverPHL (ibid.). The third alternative is based on the applet system of Cinderella. Exported applets can be used in any web browser without installing Cinderella. This makes it easy to embed an interactive construction into a web page (Richter-Gebert and Kortenkamp, 2012, p. 419ff.). Alternatively, applets can be included in rich-client Java applications (FEST, 2010a).

CindyScript

To make full use of the dynamics of the embedded Cinderella applet, the learning application needs to access its current state and influence it. This is made possible by the CindyScript interface. CindyScript is

---

8 The evaluations took place during courses held by SAiL-M project partners for students in mathematics teacher training.

9 There are several products competing in the market of dynamic geometry systems, each with different features and formats. Many of them, including Cinderella, are taking part in the Intergeo project (http://i2geo.net/ accessed 2012-08-31) with the goal of improving interoperability between DGS’s.

10 Cinderella offers a standalone, lightweight module for applets. Unlike the commercial Cinderella software package, this module can be distributed free of charge.
Figure 5.5.: A Cinderella applet inside a web browser. This Java applet contains none of the buttons that the full Cinderella application offers; the user interacts directly with the drawing using drag and drop.

a custom, functional language which is built-in in Cinderella (Richter-Gebert and Kortenkamp, 2010, p. 217ff.). The CindyScript editor window can be used to write code which operates on the drawing. Alternatively, arbitrary CindyScript commands can be executed using a Java interface, enabling the programmer of a learning application to add and remove elements or to read and change their properties.

Cinderella includes the CINErella module, which offers capture and replay functionality. Teachers can use CINErella to capture a construction process and distribute the record in the form of an interactive animation (Kortenkamp, 2005). Similar to Jacareto, the CINErella module "uses semantic event stream recording in order to provide record and playback features for the whole solution process" (Müller et al., 2006, p. 6). However, unlike the generic Jacareto framework, it is embedded with Cinderella. If an applet was included in a learning application, the CINErella module could only record the interaction with the applet, but
not with the rest of the learning application. In this case, CINErelia cannot capture the entire solution process, and is therefore not suitable for a general-purpose architecture for semi-automatic assessment.
6. A Generic Framework For Automatic Feedback

A primary goal of this thesis was to develop a generic system which can provide automatically generated feedback to the learner and make a log of the solution process available to the teacher. These functions are the topics of research questions Q2 and Q4. At first, it was considered to generate feedback based on the recorded log data. Nevertheless, it was decided to handle the logging and feedback aspects separately for two reasons. Firstly, interaction recording is a uni-directional process. The modular architecture of Jacareto does not offer a possibility for the capturing mechanism to influence the target application. Adding such a cyclic dependency would make the system very hard to maintain. Secondly, performing model tracing for automatic assessment requires domain knowledge which is present in the learning application, but not in the logging component. Jacareto can receive semantic events, but these are merely flat data structures (see section 5.1.1). This was no hindrance in the scenarios described in section 5.1.4 because the event metadata was very simple (numbers and coordinates). Other learning applications use much more complex data structures and algorithms. These are also required to evaluate the correctness of a solution; automatic assessment cannot be done on a flat representation of the solution.

This chapter covers the development of Feedback-M, a framework for automatic feedback. Following this, chapter 7 describes the design and implementation of learning applications which incorporate Feedback-M. Chapter 8 deals with the iterative development of a component for logging and tutorial feedback. Feedback-M and the learning applications only received comparatively minor improvements during these iterations. Therefore, the development of these parts is not described in chronological order. Because the separation between learning application, feedback component, and logging system is supposed to be transparent to the learner (for example, the learner is never confronted with the name Feedback-M), all three parts were evaluated together; the results of this evaluation can also be found in chapter 8.
6.1. Data Model

The Problem class is the central data structure of the Feedback-M framework. A Problem object encapsulates all the information on an automatically detected mistake in the learner’s (intermediate) solution. The problem description is stored in a human-readable string. Following the feedback-on-demand principle (see section 2.3.4), the problem description will only be shown when the user requests feedback. Likewise, a second string may contain a hint on how to solve this problem. The problem description is supposed to be a statement that something is wrong with the learner’s solution. In contrast, the hint text should contain instructions on how to fix the problem. This string is optional for the Problem class because a hint cannot always be generated automatically. Like the problem description, the hint will only become visible on the learner’s demand. A flag indicates whether the hint has already been requested.

Problem metadata

While the problem description and hint strings may be sufficient to present feedback to the learner, additional metadata may be required. For example, the problem description text might contain the line number at which the mistake was detected, or the cause of the mistake. It might be useful to additionally store this metadata in a machine-readable form. Developers of learning applications are encouraged to create subclasses of Problem to add these attributes. These metadata are useful for learning analytics. Logging all problems which have occurred during the learners’ solution processes should enable teachers to find out what types of mistakes occur most often, and in which situations these mistakes commonly occur. Furthermore, learners might be able to use this information to reflect their own performance level.

Tests

The Problem class is merely a passive data structure. Each Problem instance represents one mistake made by the learner. Classes which are responsible for detecting such mistakes are referred to as Tests (see figure 6.1). Detecting mistakes requires detailed knowledge of the domain. For this reason, Test is an interface that only declares one method, void searchForProblems(List<Problem> problems). Each test is responsible for one type of problem, and it adds all the detected problems to the list given as a parameter. This splitting of responsibilities is inspired by the rubric system (Ahoniemi and Karavirta, 2009) described in section 3.2. Tests yield Problem instances in the same way factories yield products in the abstract factory design pattern (Gamma et al., 1995). An abstract factory offers a method which returns instances of an abstract product. Its subclasses — concrete factories — overwrite this factory method to return concrete products. Similarly, the abstract Test class detects abstract problems, while concrete tests detect concrete problems.
Figure 6.1.: Simplified UML class diagram of central classes of the Feedback-M framework (left) and their subclasses in the ColProof-M learning application (right).
Chapter 6. A Generic Framework For Automatic Feedback

(see figure 6.1). One difference to the abstract factory pattern is that a factory always returns exactly one product, while tests may detect several problems of the same type at once. For example, an orthography test may find several spelling mistakes in one solution, or even no mistake at all.

Each learning application requires a subclass of the abstract class SolutionChecker. The solution checker keeps an instance of each type of test required for the domain. Its checkAll() method calls the searchForProblems() method of each test. The order in which the tests are carried out is important for two reasons. Firstly, the problems will be displayed to the learner in the order in which they were found. The learner should be informed about grave mistakes before getting details on minor things which could be improved. Secondly, one test may decide not to report a detected problem when it conflicts with another problem found by a previous test. For instance, when one test has concluded that one line written by the learner is completely wrong and should be deleted, following tests should not report spelling mistakes in this line. For this reason, the solution checker passes the test a list of all problems found by preceding tests.

6.2. Feedback Dialog

Accessing the feedback dialog

The Feedback-M framework offers a dialog window for feedback which can be integrated into the learning application. In accordance to the feedback-on-demand principle, the learner should be able to open the feedback dialog at any time during his solution process. Hence the learning application must provide a clearly visible GUI component (e.g. a toolbar button) which allows the user to open the feedback dialog at any time.

One problem at a time

It is possible for the solution checker to detect several different problems inside the solution. Due to his limited locus of attention, the learner cannot handle all these problems simultaneously. For this reason, the feedback only shows one problem at a time. As a side effect, this implementation saves screen space, making the modeless interface more useful as the feedback dialog does not conceal large parts of the learning application. The user can access further problems by using buttons labeled “Previous problem” and “Next problem” (see figure 6.2). The problems are displayed in the order in which they were reported by the solution checker, as described in section 6.1.

Feedback and hint display

The problem description text is displayed in the upper half of the feedback dialog. The lower half is blank until the learner uses the “Hint” button, in which case it also switches to a text area (see figure 6.2). The
6.2. Feedback Dialog

(a) Before requesting a hint

(b) Displaying the hint

Figure 6.2.: The Feedback-M dialog showing a problem description (a) and a hint (b). The feedback text includes a hyperlink which leads to a detailed description inside the learning application.

The Feedback-M dialog is not only used to deliver negative feedback. When no problems were found by the solution checker, the feedback dialog concludes that the learner has successfully solved the exercise, and displays a feedback message which compliments him for his solution. However, it is important not to show this message for an intermediate solution which is free of mistakes. Even though the learner

1 This is important for learning applications which limit the number of available hints per exercise (see section 7.1.3): it allows learners to browse problems without unintentionally increasing the hint counter.

2 JEditorPane only supports a small subset of HTML: [http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/Swing-Tutorial-JEditorPane.html](http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/Swing-Tutorial-JEditorPane.html) (accessed 2012-08-31). Nevertheless, it is sufficient to markup predefined feedback messages.
may not have made anything wrong yet, more solution steps are still required to complete the solution. For this reason, developers of learning applications using Feedback-M are supposed to implement a test to check this condition, and to possibly yield a Problem object which notifies the user that his solution is incomplete. This object may carry a hint that suggests which actions are useful to continue the task. The existence of the “unfinished solution” problem prevents the feedback dialog from congratulating the learner prematurely.

The feedback dialog is non-modal, i.e. the main window of the learning application is not blocked while the dialog is open. From a usability perspective, modeless interfaces are clearly preferable (RASKIN, 2000, p. 37ff). The feedback dialog allows the learner to check his intermediate solution and continue to work on it without switching between “editing mode” and “feedback mode”. The learning application is supposed to call the FeedbackDialog.refresh() method whenever the student modifies the solution. This re-assesses the solution steps and refreshes the feedback display. The modeless feedback system is supposed to encourage the student to make regular use of the feedback dialog throughout his solution process. This is different from systems like Praktomat (see section 3.1) which require the user to interrupt working on their solution in order to get it checked.

6.3. Problem Highlighting

According to the classification of feedback components by NARCISS (2006), which has already been discussed in section 2.4, a feedback component which provides knowledge about mistakes (KM) can contain information such as the number of mistakes, as well as position, type, and cause of each mistake. Even though the Feedback-M dialog does not display the number of detected mistakes, this figure can be found out by browsing through the feedback messages. Type and cause of a mistake can be presented textually, and it is up to the developer of a learning application to include this information in the problem descriptions. In order to show the position of the mistake, it is mandatory that the learner’s solution remains visible so that the location of the problem can be referenced or marked.

The feedback dialog displays problem descriptions in textual form. As most learning applications also present the learning contents visually, the question arises whether this will lead to distraction. According to CHANDLER and SWELLER (1992), the split attention effect occurs when multiple types of information with the same modality (in this case, visual) are displayed at the same time. The split attention leads to additional extraneous cognitive load and is therefore undesirable.
With the Modality Principle of Multimedia Design, Mayer (2006) suggests combining visual information with narration to avoid the modality conflict; however, this recommendation is unsuitable for the presentation of feedback. Firstly, creating voice recordings of feedback messages requires much more effort for developers of learning applications than writing feedback messages. Secondly, problem descriptions are generated dynamically depending on the current solution of the learner, so replaying static narrations is not an option.

Chandler and Sweller (1992) propose to reduce the effect of split attention by moving the pieces of visual information closer together. In this case, this would mean putting the feedback texts right next to the mistake in the learner’s solution. The disadvantage of this approach is that feedback pop-ups would regularly occlude parts of the solution. As a compromise, the Feedback-M framework supports problem highlighting. GUI components of the learning application can register to the FeedbackDialog as FeedbackListeners. Each time the dialog displays a problem, all listeners are notified (see figure 6.3). As described in section 6.1, subclasses of Problem can carry additional metadata. By setting attributes such as line numbers in instances of Problem subclasses, it is possible to highlight the problem within the GUI of the learning application.

6.4. Tutor Feedback Requests

The Feedback-M features described so far enable developers of learning applications to generate problem descriptions and hint messages and offer a user interface to provide the learner with these feedback texts. This is an implementation of the feedback on demand component of the tutor-in-the-loop model and constitutes the foundation of the intelligent assessment component. According to this model, the student is supposed to ask the teacher for feedback whenever the intelligent assessment component cannot evaluate the solution, or when the student is unable to comprehend the delivered feedback. Embedding the communication into the learning application itself is supposed to encourage students to contact their tutors whenever necessary. The workflows of the tutor-in-the-loop model imply that the automatically generated feedback should always be consulted first, so that the tutors will not spend time on questions which could have been answered by the computer. This is why the function for tutorial feedback requests is accessible only from within the feedback dialog.

The button to request tutorial feedback is labeled “Ask tutor a question”. It is placed right next to the buttons with which the learner can browse the detected problems (see figure 6.2). Once the learner has
Figure 6.3: Simplified UML class diagram showing the problem highlighting mechanism of the Feedback-M framework (left) and how it has been used in the ColProof-M learning application (right).
6.4. Tutor Feedback Requests

The Feedback-M dialog after the student has finished the exercise and typed a question to his tutor.

Figure 6.4.: The Feedback-M dialog after the student has finished the exercise and typed a question to his tutor.

6.4. Tutor Feedback Requests

completed the exercise (i.e. no problems have been found by the intelligent assessment component), the label of this button changes to “Send solution to tutor” (see figure 6.4). In both cases, the button opens a second dialog window: the message dialog. To send a message, the learner has to select his tutor from a drop-down list. The tutors’ names and e-mail addresses are read from a configuration file which is shipped with the learning application. A check box to attach the current state of the solution is marked by default, making it possible for the learner to reference it in the message text. Further text fields (e.g., for the student’s name) can be added using the configuration file. The protocols that are used to send the messages will be discussed in sections 8.2.3 and 8.3.1.

In order to realize a complete learning environment with semi-automatic assessment, the remaining components of the tutor-in-the-loop model are required: electronic exercises with intelligent assessment and a logging component with analytics support. The following chapter describes the development of various learning applications which provide the learner with electronic exercises and which make use of the Feedback-M framework. Subsequently, chapter 8 deals with the logging component which enables teachers to review individual solution processes in order to answer feedback requests. It also describes the...
analytics component that offers him an overview of course-wide performance.
7. Proofs of Concept

In the course of the SAiL-M project, several learning applications were developed which incorporate the Feedback-M framework. These applications are described in this section. The focus is on the tools ColProof-M and SetSails! because those are the ones that make use of the widest range of Feedback-M features.

7.1. ColProof-M

Proving mathematical theorems is an important part of the curriculum in introductory mathematics courses at university level. For example, lectures about elementary geometry are usually held in combination with tutorials in which students are asked to prove geometric statements. ColProof-M is a learning application which supports the learner in developing such proofs (Herding et al., 2010).

7.1.1. Domain

One approach to proving theorems in Euclidean geometry is the deductive proof. Proving is a creative task, and there are usually many possible solutions. There are some typical approaches for each proof assignment, and most learners who succeed in constructing a proof follow one of these. But even when two learners use the same approach, their solutions can vary, if only in terms of notation styles and formulations. These inconsistencies make it difficult to assess students’ solutions. To overcome this problem, one can follow formal notations of writing down proofs. The ColProof-M application uses one of those, namely the two-column proof format.

The two-column proof (Herbst 1999) is a format which is especially wide-spread in North America; however, similar formats can also be found in German literature on didactics of mathematics (Holland 1988). Herbst argues that “the two-column proof format brought stability to the geometry curriculum by providing a way to meld the proofs given by the text and the proofs asked from students” (2002, p. 304). Others dispute the usefulness of the two-column format.
USISKIN points out that the format is hardly usable for anything but proofs in elementary geometry, and he criticizes the required rigor, "forcing students to write down every step and give every reason" ([1980] p. 419). However, it is exactly this rigor that emphasizes an important principle of deductive reasoning, namely that each proposition must be justified using definitions, postulates, theorems, or propositions whose validity has been proven earlier. HOLLAND calls this requirement "Lückenlosigkeit" (freedom from gaps): each proof step must result from given definitions and theorems and from preceding lines of the proof ([1988] p. 38). Once students have internalized this principle, they can switch to less particularized, semi-formal mathematical notations or informal texts (ibid.). As a side effect, the standardized format makes evaluation easier for tutors as well as for computers.

Figure 7.1.: Example for a two-column proof ([SPARKNOTE] 2000).

As its name suggests, a two-column proof is written down in a table with two columns. The first rows of the proof usually hold given propositions, while the following rows contain the inferred proof steps. Usually, the table rows are numbered (cf. HOLLAND [1988]; HERBST [1999]). In each row, the left column contains a proposition, such as "The triangle ABC is isosceles". The right column holds one or more reasons...
why the proposition on the left is true. The reasons are references, either to propositions further above in the proof, or to general theorems whose validity may be presumed. Theorems are often referenced using an abbreviation, e.g. ASA for the Angle-Side-Angle postulate (Silver 2008). In contrast, propositions are often referenced by their row numbers (cf. Holland 1988). In some notation variants, references to propositions are left out, in which case each proof step is implicitly justified by the previous proof steps. Other variants only reference row numbers when the references are not obvious (see figure 7.1): “Anytime it is helpful to refer to certain parts of a proof, you can include the numbers of the appropriate statements in parentheses after the reason” (SparkNote 2000). The convention of referencing propositions by row numbers works fine for writing down finished proofs, but has a major disadvantage for learners who are in the process of constructing a proof: because rows can be reordered and new rows can be inserted, the row numbers are constantly changing. For this reason, using these row numbers as references would be very confusing for the learners. Instead, ColProof-M uses abbreviations not only for theorems, but also for propositions. These abbreviations can be used in the reasons column to reference previous proof steps.

Silver (2008) suggests letting students put together pieces of paper to build two-column proofs. Each piece of paper represents either a proposition or a reason which can be used for a proof step. In the ColProof-M application, the user can interact similarly, using drag and drop to place proof steps. Theorems which may be used can be looked up in a virtual book. At the beginning of an exercise, the proof table only contains given propositions. All the other propositions are available in an area which is underlaid with a thought bubble (see figure 7.2). In order to challenge the user, the creator of an exercise may also include distractors of two different types:

**Wrong propositions:** Proof steps which the learner is not supposed to use because they are incorrect statements.

**Unnecessary proof steps:** Theorems and propositions which are factually correct, but are not necessarily required in order to solve this particular proof.

Once a proof step has been moved to the solution table, the user can use the reason column to state why the proposition is valid. The table cell contains a drop-down list with all available theorems and all previously added propositions. Once a reason has been chosen, a further drop-down list automatically pops up so that additional reasons can be selected. Each drop-down list also contains a blank value which can be chosen to remove a reason from a proof step.
Chapter 7. Proofs of Concept

7.1.2. Visualizations

As ColProof-M is about proofs in geometry, it is obvious that each exercise should come with a sketch. These illustrations are important not only because they help the learner understand the relation of shapes, lines, and angles, but also because their labels reoccur as variable names in the proof steps. However, the learner is not supposed to deduce from information which was extracted from the sketch. [HOLLAND 1988, p. 40] gives a prominent example for such a mistake: a statement taken from a sketch of a triangle turns out not to be generalizable. This becomes apparent when regarding an illustration of a different triangle. This example highlights the danger of offering a static illustration to aid learners in their proof efforts.
Instead of static drawings, the ColProof-M interface was equipped with an applet window containing an interactive visualization (FEST 2010a). This feature was realized using the Cinderella dynamic geometry system (see section 5.2). Each ColProof-M exercise file comes with a labeled, dynamic sketch constructed in Cinderella. The learner can manipulate the sketch by dragging points, while Cinderella automatically adapts the remainder of the illustration so that important premises such as perpendicular or parallel lines persist. Furthermore, when the user clicks on a proof step in the ColProof-M main window, the corresponding parts of the illustration are automatically highlighted (see figure 7.3). This feature was realized by defining the related variable names for each proposition. When the proposition is clicked, a piece of CindyScript (see section 5.2) is executed, highlighting the elements of the sketch which are labeled with these variable names.

Besides the tabular representation, ColProof-M can also visualize the learner’s solution in the form of a proof graph. Such graphs are made up of vertices which represent propositions and directed edges which stand for inference relationships. Edges from proposition A and B to proposition C express that C is inferred from A and B. This means that given statements do not have incoming edges. HOLLAND (1988, p. 37) suggests labeling edges with theorems or definitions when these are used as reasons. A similar style is also used by KOEDINGER and ANDERSON (1990). ColProof-M uses a different notation in which theorems and definitions are also represented by vertices, labeled in italics. This is a simplification because edges do not need labels at all, and theorems...
Chapter 7. Proofs of Concept

Figure 7.4.: A proof graph for Thales’ theorem in the graph visualization of ColProof-M. Given statements and theorems do not have incoming edges.

are treated similar to given propositions. This makes it easier to represent proof graphs using standard graph frameworks. The Java Universal Network/Graph Framework (JUNG) was used to generate graph data structures, but also to render the proof graphs. A special JUNG graph layout, the ProofGraphLayout, was implemented in order to emphasize the typical structure of a proof graph (see figure 7.4).

7.1.3. Assessment

Truly automating the assessment of proofs would require an automatic theorem prover capable of working on the topics of all ColProof-M exercises. There are several approaches to combining methods of automatic theorem proving with dynamic geometry systems. An overview of this research is given by Kortenkamp and Richter-Gebert (2004).

1 “JUNG – the Java Universal Network/Graph Framework – is a software library that provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network.” http://jung.sourceforge.net/ (accessed 2012-08-31).
However, automatic theorem provers have two disadvantages when it comes to formative assessment:

1. Many automatic theorem provers cannot guarantee a maximal running time. Randomized theorem proving is an approach to limit the running time at the expense of guaranteed correctness (KORTENKAMP and RICHTER- Gebert 2004). Nevertheless, even randomized theorem proving is a computationally complex task. The tutor-in-the-loop model requires an automatic assessment component which can check intermediate solutions on demand. A feedback system which depends on a time-consuming theorem prover would have limited usefulness in this context.

2. Automatically generated proofs are often difficult for humans to understand. “One salient property of mathematical proofs as typically found in textbooks is that lines of reasoning are expressed in a rather condensed form by leaving out elementary and easily inferable, but logically necessary inference steps, while explaining involved ones in more detail. To date, automated proof presentation techniques are not able to deal with this issue in an adequate manner” (Horacek 1999, p. 142). It would be very challenging to extract information from an automatic theorem prover and use it to formulate easily understandable feedback and hint texts for the learner.

For these reasons, no theorem prover was integrated into ColProof-M. Instead, a sample solution has to be defined when creating an exercise. This sample solution can be used by the ProofSolutionChecker, a subclass of the SolutionChecker (see figure 6.1). It delegates to a large number of different tests to verify the structural and semantic correctness of the learner’s (intermediate) solution. Many types of problems which can occur in ColProof-M can be detected more efficiently by analyzing the proof graph than by evaluating the proof table. The main reason for this is that one can rely on established, highly optimized algorithms when working with graphs. Whenever the method ProofSolutionChecker.getTests() is called, a graph data structure is generated from the current state of the solution. This is done the same way the graph for the visualization component is computed. The generated proof graph is then shared by all tests in order to minimize computational requirements. In addition, all tests get access to the linear list of proof steps.

The learner’s goal is not to exactly reproduce the sample solution. As Holland (1988, p. 37) points out, several linear sequences of proof steps may lead to an equivalent proof graph. As long as the learner’s result can be mapped to the proof graph of the sample solution, the ProofSolutionChecker reports it as correct. Still, this approach confines the solution space to those theorems and propositions used
in the sample solution. In order to relax this restriction, a feature was added to ColProof-M which enables learners to add custom propositions or theorems to the proof. When the learner makes use of this feature, the application notifies him that the automatic solution checker will not be able to assess the custom propositions and theorems. The tutor-in-the-loop model can be used to fill this gap: a student who has added propositions, and whose solution therefore cannot be automatically evaluated, can at any time contact his tutor to ask for feedback.

In order to perform the automatic assessment of the student’s (intermediate) solution, the ProofSolutionChecker delegates to several Test instances. Many of these tests check whether the learner’s solution is well-formed, i.e. whether it is structurally sound. While structural tests work the same way for any exercise, other tests depend on the semantics of the propositions and theorems. The structural tests only work on the learner’s (intermediate) solution and the graph generated from it. Thus, the structural tests are a realization of the model-tracing approach (see section 2.2). The semantic tests, on the other hand, take the sample solution and its graph into account, due to the lack of a usable automatic theorem prover.

To develop tests for a learning application, it is necessary to categorize typical mistakes in the domain (BESCHERER et al., 2010). In the case of ColProof-M, this was done by building on the experience of domain experts from the Ludwigsburg University of Education. Hereafter, the resulting structural and semantic tests of ColProof-M are briefly described. They are listed in the order in which they are run by the ProofSolutionChecker.

**ExerciseNotStartedTest**: Checks whether the proof table contains only the given propositions. This means that it yields a Problem instance when the user opens the feedback dialog directly after opening an exercise. The feedback message is not formulated as criticism, but rather as an instruction on how to get started.

**WrongPropositionTest**: Examines whether the learner has used a proposition which is factually incorrect (e.g. the proposition “The triangle $\Delta ABC$ is equilateral” in an exercise about isosceles triangles). As stated in section 7.1.1 exercises may contain such wrong propositions as distractors. Each proposition of an exercise carries a flag by which this test can tell whether it is incorrect.

**UnnecessaryPropositionTest**: Like the WrongProposition-Test, this test analyzes whether the learner has included a distractor in his proof. This test, however, is responsible for those propositions which are factually correct, but unrewarding because they are not required to finish the proof.
CircleTest: Detects circular reasoning in the learner’s solution. The ColProof-M interface prevents the user from choosing a proof step as a reason for itself. Nevertheless, two or more proof steps can reference each other, thereby forming a circular chain of reasoning, a typical logical fallacy. Given the proof graph, one can efficiently detect circles by searching for a path from any node to itself. This was realized using Dijkstra’s shortest path algorithm which is provided by the JUNG framework. The nodes of detected circles are stored to give the learner elaborated feedback on which proof steps form the circle.

MissingIntermediateStepsTest: Let A and B be given propositions. Assume that according to the sample solution, C can be inferred from A and B, and D can be inferred from C. The learner could argue that D is true because of A and B. From a logical point of view, this is not a mistake because deduction is transitive. In this case, a MissingIntermediateStepsProblem with this description would be reported: “The reasons that you selected for proof step D aren’t wrong. But your proof would be easier to understand if you inserted an intermediate step.” The hint for this problem would be: “Take a closer look at this proposition: C.” The test compares the reasons from the learner’s solution to the one from the sample solution. Because this comparison is done recursively, this implementation works even when multiple intermediate steps are missing.

WrongReasonsTest: Compares the reasons of each proof step with those of the corresponding proof step in the sample solution. If a problem is detected, its hint will point to a proposition which should be added to or removed from the list of reasons. The comparison can bring up three different types of problems. Firstly, the learner may have selected some correct reasons, but missed out other necessary ones (missing reasons problem). Secondly, he may have chosen all the correct reason, but additionally selected incorrect ones which have to be removed (unnecessary reasons problem). Thirdly, a subset of the correct reasons may have been combined with some incorrect reasons, making it difficult to comprehend the learner’s misconception (wrong reasons problem). This test skips proof steps for which a MissingIntermediateStepsProblem has already been detected, as that problem description explicitly states that the reasons are not wrong (even though they deviate from the sample solution).

**ReasonNotInProofTest:** It is possible for the user to move a proposition out of the proof table again in order to remove it from the proof. If the removed proposition was used as a reason for another proof step, there is a gap in the deduction which this test can detect and report. Even though this is a mistake which is not expected to happen very often during solution processes, the test was realized because the mistake would break the fundamental structure of a proof.

**NoReasonsTest:** This very simple test checks whether the solution contains a proof step which is not given, but lacks a reason. When the user requests a hint for such a problem, the feedback system points to one proposition which can be used as a reason. This information is taken from the sample solution.

**MultiOccurTest:** Another very simple test which inspects whether a proposition occurs multiple times in the list of reasons for a single proof step. This is a minor problem which the user can easily fix at any time.

**WrongOrderTest:** Detects whether a proof step is inferred from a proposition which has been positioned further down the proof table. The rationale for this test is that it should be possible to read the solution from top to bottom. WrongOrderProblems are easy to resolve by reordering the proof steps using drag and drop. This test excludes parts of the proof which were already faulted by the CircleTest.

**UnfinishedTest:** The last line of the sample solution holds the proposition which is *quod erat demonstrandum* (Q. E. D.: what was to be demonstrated). The learner’s solution is incomplete as long as this proposition is missing. If no problems have been found in the intermediate solution besides the fact that it is unfinished, this test yields positive feedback, as the learner is on the right track. The hint for this problem points to a proof step which can be used to continue the solution.

**ProofStepsEditedTest:** This final test checks whether the user has introduced custom propositions. As discussed in section 7.1.3, this is a valid approach, but it prevents fully automatic assessment. If custom propositions were detected, the corresponding feedback message explains this circumstance and asks the learner to request feedback from his tutor.

Countermeasures against feedback overuse

It should not be possible for the learner to reveal the full sample solution by continuously requesting feedback and hints. Several possible countermeasures have been described in section 2.4.2. ColProof-M uses
7.1. *ColProof-M*

the *hide on activity* principle: The feedback dialog closes itself automatically whenever the learner modifies his solution in a way that affects the problem currently being displayed. This makes it more laborious to systematically change the solution until the problem vanishes. Furthermore, the problem descriptions are a form of vague feedback because they point to mistakes without giving away the correct solution. However, most problems carry a second level of feedback, namely a hint text (see section 6.1). These hints are meant to reduce frustration for students who are stuck. But because *ColProof-M* does not know about the semantics of the propositions, it is impossible to automatically obscure the hint texts to make them more vague. In order to prevent cheating despite this weakness, an *overuse detection* was implemented. Each time a hint is taken, a visible counter is decremented. Only three hints can be taken per exercise unless the learner resets his solution, in which case the hint counter is also reset. Additional compliments are given to learners who solve an exercise taking no or very few hints. This is supposed to foster self-determined learning, as it may encourage learners to take as little hints as possible in following exercises. As a fourth layer of protection against cheating, the logs which are collected for the tutor-in-the-loop system could also be used for *tutorial supervision* in order to find out whether students overuse the feedback dialog. However, this supervision might conflict with the learners’ privacy. This issue will be discussed in chapter 8.

Many feedback messages mention the row numbers of incorrect proof steps. To further help the user find the position of mistakes, the *FeedbackListener* interface (see section 6.3) was used to implement a highlighting feature. All *ColProof-M* problems except for the *UnfinishedProblem* support highlighting the affected parts of the solution. The *ProofStepTable* – a subclass of *JTable* which holds the learner’s proof – implements the *FeedbackListener* interface. Depending on the type and the metadata of the problem shown, it decides which rows to highlight, and whether to highlight only cells in the reasons column or all the cells. The *JTable.getCellRect()* method is used to calculate the coordinates where semi-transparent, red rectangles should be painted (see figure 7.2). Moreover, the *ProofStepThoughtBubble* – the component holding the unused propositions – is a *FeedbackListener* as well. When a *ReasonNotInProofProblem* is reported in the feedback dialog, the proposition in question is highlighted in the thought bubble. Like in the table, this is done by painting a red overlay over the GUI component. An *ExerciseNotStartedProblem* causes the thought

---

3 In teacher mode (see section sec:ColProof-M Exercises), one can modify the number of available hints for a certain exercise. This makes it possible to offer additional hints for very difficult tasks, or to reduce the number of hints to increase the challenge for easier tasks. Three hints is the default value which has been used during the evaluation of *ColProof-M*.  

8 Problem highlighting
bubble to highlight all its propositions in order to prompt the user to interact with them.

### 7.1.4. Exercises

In the current version, ColProof-M is shipped with five different exercise files. Each exercise includes a Cinderella file for the interactive visualization.

**Bisection of lines:** In this proof, the learner is supposed to show that two triangles formed by a bisecting intersection of two lines are congruent. This is a very simple assignment which can be solved in four steps. Its primary purpose is to give beginners an insight on the proof format and user interface of ColProof-M.

**Euclidean theorem:** This theorem can be proved using the Pythagorean theorem.

**Intersection of the perpendicular bisectors:** All three perpendicular bisectors of a triangle intersect in one point. Thanks to the interactive Cinderella-based visualization, one can see that this point may lie outside of the triangle.

**Thales’ theorem:** If the center of the circumscribed circle of a triangle lies on the midpoint of one side, then the triangle is right-angled. This exercise is shown in figure 7.2.

**Medial triangle:** Using the intercept theorem, one can prove that the line connecting the midpoints of two sides of a triangle is parallel to the third side, and half as long.

Teachers can define additional exercises by starting ColProof-M in teacher mode. This mode looks similar to the student view, but provides additional menus to define exercise properties and to add proof steps and theorems. The proof, as built up by the teacher, serves as the sample solution when the same exercise file is later opened by a student.

### 7.2. SetSails!

*SetSails!* is a learning application which deals with the algebra of sets and the Boolean algebra. The algebra of sets comprises sets, the operations of union, intersection, and complementation (and the set difference which is derived from those operations), as well as rules such as...
the commutative property of union and intersection and De Morgan’s laws. The algebra of sets is related to the Boolean algebra. “In general, a uniform way of determining the set-theoretic operation corresponding to a given truth function is to express the latter in terms of ¬, &, ∨, and then replace ¬, &, ∨ by ⊼, ∩, ∪ respectively. The statement letters need not be replaced since they can serve as set variables in the new expressions” ([MENDELSON] 1970 p. 37).

7.2.1. Domain

There are many types of tasks in propositional logic and set theory which students of mathematics and computer science regularly work on as exercises, e.g. resolution ([HUERTAS et al.] 2011) or transformation to conjunctive or disjunctive normal form (CNF and DNF, cf. [HEEREN et al.] 2010). Another task is to show that two logical expressions (or set theory terms, respectively) are equivalent. There are multiple approaches to proving such an equivalence:

**Exhaustive proof:** This is a brute-force approach in which all possible configurations of the variables are tested. In the Boolean algebra, one can say: “Two well-formed formulas P and Q are equivalent […] if and only if they have the same truth values under every interpretation” ([ALAGAR and PERIYASAMY] 2011 p. 138). This method can be visualized with the help of truth tables. “Truth tables provide an exhaustive proof method for propositional logic. To prove a claim Q from the premises P₁, P₂, . . . , Pₖ, one constructs a truth table […] and verifies whether or not in every row of the truth table Q is true” ([ALAGAR and PERIYASAMY] 2011 p. 139). The required runtime for the generation of the truth tables grows exponentially with the number of variables, but typical exercises only comprise a small number of variables, so this problem is negligible. The corresponding procedure for the algebra of sets is to draw Venn diagrams ⁴ for the left and for the right side of the equation, and to check whether both diagrams are congruent.

**Proof by cases:** In this method, one shows equality of two sets by proving that they are subsets of each other (analogously, one shows equivalence of two logical expressions by proving that they imply each other). For both statements about subsets, one assumes an element of one set and, using case differentiations, shows that it is also an element of the other set ([SOLOW] 2010).

---

A Venn diagram is a graphical representation of a set algebraic term. According to Wikipedia, “a Venn diagram for n component sets must contain all 2ⁿ hypothetically possible zones that correspond to some combination of inclusion or exclusion in each of the component sets.” [http://en.wikipedia.org/wiki/Venn_diagram?oldid=509795526](http://en.wikipedia.org/wiki/Venn_diagram?oldid=509795526) (accessed 2012-08-31).
p. 175ff.). The effort required for this grows with the complexity of the terms because it may be necessary to cascade case differentiation, as demonstrated in the example in appendix A. In fact, the exhaustive proof is a naive form of the proof of cases.

**Proof by equivalence transformations:** The prerequisite for this is that there is a pool of axiomatic transformation rules, such as De Morgan’s laws or commutative and associative properties of operations. The validity of these rules has been verified earlier by proofs by cases or exhaustive proofs. For this reason, it is safe to use them to transform terms. The equivalence of two terms can be proven by repeatedly transforming one side of the equation until it holds the same term as the other side. An example for this can be found in appendix A.

All of these proof methods possess a *raison d'être*. Despite the exponential runtime, exhaustive proofs can easily and quickly be performed automatically for equations with a small number of variables. In exercises, they are a good way to learn about the operators, but offer the learner little insight beyond that. The proof by cases and proof by equivalence transformations bear one fundamental difference: while the former works by investigating elements of the sets, the latter emphasizes the fact that the sets themselves are mathematical entities with which one can calculate. This is important from a didactical point of view because it helps the learner notice the similarity to the elementary algebra, and thereby better understand the concept of algebras in general. For this reason, *SetSails!* was developed as a learning application on the topic of proofs by transformations in the Boolean algebra and the algebra of sets (Herding and Schroeder, 2012). Similar to ColProof-M, *SetSails!* features a virtual book in which the learner can look up algebraic rules.

### 7.2.2. Visualizations

For both algebras supported by *SetSails!* there exist well-established visual representations for terms. These are truth tables for the Boolean algebra and Venn diagrams for the algebra of sets. Both visualizations can be dynamically generated by *SetSails!* to help the user understand particular terms. Truth tables are easy to render using the HTML support of Java Swing, which has already been mentioned in section 5.2. Rendering Venn diagrams, on the other hand, is more challenging. Bitmap graphics were drawn for diagrams with zero to four sets. They were specially crafted so that each intersection area is filled with a slightly different color value. Bitmasks are then used to recolor these intersections, according to whether they are part of the set denoted by the term. The visualization of the equation to be proven is
permanently visible in the SetSails! main window. For each transformation which the student performs, he can open a pop-up to compare the visualization of his selected term to that of the equation [ZIMMER-MANN and BESCHERER 2012]. Furthermore, there are visualizations of the equations in the rule book (see figure 7.5).

Currently, SetSails! is limited to terms with up to four variables. A Venn diagram consisting of four ellipses can be seen in figure 7.6. Truth tables would become very large for more variables (the length of truth tables grows exponentially with the number of variables), and Venn diagrams with more than four sets quickly lead to confusion.\footnote{See \url{http://en.wikipedia.org/wiki/Venn_diagram?oldid=509795526} (accessed 2012-08-31) for possible Venn diagram constructions with five or more sets.}

![Figure 7.5: SetSails! during a transformation using the set difference definition. Note that the list of term options contains distractors. Algebraic rules can be browsed in the virtual book on the lower left. Transformations can be performed top-down and bottom-up.](image)

### 7.2.3. Transformations

Unlike ColProof-M, which relies on a sample solution, SetSails! has a proper model of the terms and rules of the supported algebras. Internally, terms are represented using term trees. These are recursive data structures which consist of nullary, unary, and binary operations. The nullary operations are the universe \( U \) and the empty set \( \emptyset \) (\( t \) and \( f \) in Boolean algebra) as well as variables. The complement operator (like...
Chapter 7. Proofs of Concept

The logical negation) is unary, while union, intersection, and set difference (like logical or, and, implication, and equivalence) are binary. Each term class offers a `getChildren()` method which returns a list of all subterms (this list is empty for nullary operations).

In section [7.1.1](#), it was argued that rigor in notation is necessary in *ColProof-M* to emphasize that each proposition must be justified. A similar principle exists in equivalence transformations: for each transformation, the learner is supposed to state which rule was applied. In *SetSails!*, laws and definitions of the algebra of sets and of the Boolean algebra are implemented in subclasses of the abstract class `Rule`.

When students try to do proofs by equivalence transformations which resemble the exercises they have seen in textbooks, they often repeatedly transform the left side of the equation until they reach a term that is equal to the one on the right side. This is, however, not the only valid approach. One might as well start with the right side, transforming it until one reaches the left side. It is even possible to transform both sides simultaneously, with the goal of reaching a common term. In fact, this solution process is often the most intuitive one. The last proof shown in appendix [A](#) is most easily done by first resolving the set differences on both sides of the equation. *SetSails!* allows bidirectional transformations. It features two transformation tables: a top one for the left and a bottom one for the right side of the equation. The lower table works bottom-up (see figure [7.5](#)). The last row of upper table and the top row of the lower table each contain a drop-down list, with which the user can alternately select a rule that should be applied, and a term which results from this rule [6](#). Once both tables contain the same term, the user can merge both transformation sequences [7](#) ([Zimmermann and Herding, 2010](#)). Figure [7.6](#) shows the feedback dialog with the congratulation message which is displayed when there are no mistakes left once the transformation sequences have been merged.

Because it is difficult to enter terms of the algebra of sets or the Boolean algebra using a standard keyboard, the user can choose from a drop-down list containing possible result terms. These terms are generated by the method `getOptions(Term previousTerm, Rule selectedRule)` in the class `OptionsGenerator`. It delegates to the `transformRecursively(Term)` method of the selected rule, which returns a list of all terms which may result from applying that rule to

---

6 By default, all predefined rules of the algebra can be used. However, it is possible to restrict the available rules for certain exercises. For example, the Absorption Law is predefined for the algebra of sets. There is an exercise in which the learner has to prove that the Absorption Law is correct. In this exercise, the Absorption Law itself was made unavailable to avoid the tautological fallacy.

7 A merge button automatically appears between both tables when they end with the same term (see figure [7.8](#)). However, it will not show as long as there are serious mistakes in the proof. This is checked internally by running a `RuleMisappliedTest`, which will be described in section [7.2.4](#).
the given term or to any of its subterms. This is similar to applying rewrite rules of a term rewriting system (cf. Heeren et al., 2010).

Even though the options generator feature was originally built as a facilitation of term input, it also simplifies the solution process for the learner. Sometimes, there is only one way to apply a rule to a term, so there is only one option. This would mean that there is no more room for mistakes. To keep up the challenge, the options generator was augmented with distractor generation functionality. When there are only few valid options, distractors are added. These distractors are generated randomly, but were made to resemble typical mistakes which learners might make when solving a similar exercise on paper. Discussions with teachers at the Ludwigsburg University of Education (priv. comm.) revealed that, from their experience, the following two types of mistakes were the most common ones:

Mix-up of rules: Certain pairs of rules are often confused, for example the commutative and the associative law. Each Rule class implements a getCommonMixUps() method which returns a list of related rules. The options generator uses these related rules to generate distractors when needed. These kinds of distractor terms are equivalent to the original term; it is only the justification for this equivalence which is incorrect.
Misapplication of a rule: Such mistakes are made either because of carelessness, or because of a misconception. For example, by misapplying the distributive law on the term \( A \cap (B \cup C) \), one might reach the term \((A \cup B) \cap (A \cup C)\), which is in fact not equivalent: the union and intersection operators must be interchanged.

In order to generate such distractors, the previously mentioned method \( Rule.transformRecursively() \) was extended by a boolean parameter \( makeMistake \). If this flag is enabled, the rule will deliberately perform a miscalculation to return a non-equivalent result. This is similar to the “Buggy rules” approach described by GERDES et al. (2008, p. 3f.).

Randomization of distractors

When there are less than three valid options, the options generator inserts a random number of distractors; however, it will stop adding distractors once a total number of five options has been reached. The chances are fifty-fifty for each distractor to be generated by rules mix-up or by rule misapplication. Each distractor is inserted at a random position.

Custom term input

As has been noted earlier, the options drop-down list was implemented because it is difficult to enter terms with a keyboard. Nevertheless, there are situations in which this list does not contain the term which the learner wants to use. For instance, when applying the idempotence law of the intersection operator \( A = A \cap A \), the number of possible results grows exponentially with the length of the original term. For this reason, it is not practical to include all result terms in the drop-down list. In order to restrict possible solutions, a feature was developed which allows the learner to enter custom terms. This feature can be accessed via the last item of the drop-down list, labeled “Enter a custom term”. This opens a dialog window in which the user can enter a term. Special characters can be inserted either by typing a similar character (e.g. the letter \( u \) for the \( \cup \) symbol), or by using a button (see figure 7.7). Using the shunting-yard algorithm, the entered term is converted to Reverse Polish notation, from which it is easy to create a term tree using the data structures for SetSails! terms (see section 7.2.3). A similar dialog window is available to define new exercises.

Inclusion of custom rule

Similarly, it is possible for the user to enter a custom rule. For example, a student may want to justify a term transformation with an equation which was proven in an earlier lecture. When the user chooses to use a custom rule, the name of the rule has to be typed in. Afterwards,
the user has to enter a custom term which should result from the application of the custom rule. On choosing a custom rule, the learner is informed that the exercise can no longer be evaluated fully automatically, and is asked to make use of the tutor-in-the-loop feature.

7.2.4. Assessment

Unlike ColProof-M, the user interface of SetSails! leaves little room for the user to make structural mistakes. For this reason, SetSails! requires fewer test classes than ColProof-M and most of them address semantic problems. Most problems reported by the tests support highlighting the position of the mistake (see figure 7.8), similar to the ColProof-M feature described in section 7.1.3.

**RuleMisappliedTest**: Tests whether the original term and the result term are equivalent for each transformation. This is done internally by means of an exhaustive proof. If the two terms are not equivalent, then the rule cannot have been applied correctly, and a problem is reported. This test also works for custom terms and rules which the learner has typed in. To fix a RuleMisappliedProblem, the user has to remove the faulty transformation and all following ones.

**RulesMixedUpTest**: Checks whether the result term of a transformation is equivalent to the original term even though the selected rule is not the correct justification. Furthermore, this test checks
whether there is another rule which would be a correct justification. In order to allow the user to resolve the reported problem, the transformation table is a FeedbackListener. When a RulesMixedUpProblem is reported, it shows an additional drop-down list at the position of the wrong rule, which allows the user to replace that rule (see figure 7.8).

CircleTest: Unlike the identically named test of ColProof-M (see section 7.1.3), this test does not point to a logical fallacy. Instead, it merely detects whether the same term occurs multiple times in one transformation sequence. Such a repetition is not a mistake, but it makes the proof unnecessarily long. A CircleProblem can be fixed by removing all the transformations which form the circle. However, the user interface of the transformation table only allows the user to remove terms or rules from the last rows. Removing intermediate rows would, in most cases, break the transformation sequence, and is therefore not supported. In order to still allow the user to remove circles, the Feedback-M framework was extended by the Assistance concept. For each reported problem, the feedback dialog checks whether there is a subclass of Assistance which is responsible for this Problem type. If so, the feedback dialog is augmented by a button which can help the user resolve the problem. So far, the CircleProblem is the only problem class making use of the Assistance mechanism.

UnmergedTest: Reports when the learner has transformed both sides of the equation to the same term, but has not yet merged both tables. The problem description is formulated as positive feedback, congratulating the learner on the progress and telling him that the proof is nearly complete.

PreviouslyFinishedTest: This test is responsible for the rare case in which the user has reached the same term on both sides of the equation, but has then continued transformations instead of merging the two proof sequences.

UnfinishedTest: This is always the last test to run. It may report that additional transformations have to be performed in order to finish the proof. The UnfinishedProblem is omitted when a RuleMisappliedProblem has been detected previously. The rationale behind this is that serious mistakes should be fixed before performing further transformations, as continuing to transform a wrong term cannot lead to a correct solution.

Generating a hint which helps the learner complete his unfinished proof is a nontrivial task. SetSails! originally followed a brute-force approach in which all algebra rules are repeatedly applied to one side of the equation until the other side is reached. It quickly became obvious
7.2. SetSails!

The learner has reached the same term from both sides of the equation and may join both transformation sequences. However, there is also a minor mistake: commutative and associative laws have been mixed up. The additional drop-down list allows the learner to fix this mistake without resetting the subsequent transformations.

that this method was too slow to supply feedback on demand. Its runtime is exponential, and furthermore, rules such as the idempotence law of the union operator \((A = A \cup A)\) double the term length at each step, making the brute-force approach impractical when the solution is more than a few transformation steps away. Sometimes it is possible to guide the learner using production rules in the sense of model tracing (Anderson and Pelletier, 1991). For instance, when the left side of the equation contains a set difference, but the right side does not, it is usually a good idea to eliminate the set difference operator on the left side. For other equations, such production rules are not available.

Further possible hint strategies for proofs by equivalence transformation were discussed with Bastiaan Heeren and Johan Jeuring (priv. comm.). This resulted in the idea of transforming both sides of the equation to disjunctive normal form (DNF), then to align both DNF terms to show their equivalence. However, this strategy was not implemented in SetSails! because it often leads to long, bloated transformation sequences, which differs greatly from the elegant proofs that an experienced mathematician can find intuitively. Looder and Heeren have suggested refinements to the described strategy in order to achieve “expert-like equivalence proofs (i.e., proofs that appear non-mechanical)” (2011, p. 155). Their improved strategy tries to “search
for parts that do not have to be rewritten [to normal form] at each step” (ibid., p. 157). This results in shorter and more elegant proofs. Nevertheless, their prototypical implementation allows deviations from the calculated strategy, “also because they may prove to be clever short-cuts” (ibid., p. 155). This shows that further research is required in order to find a way to automatically generate hints which can compete with the intuition of a human teacher. Until then, the tutor-in-the-loop model remains as a viable compromise for these kinds of proofs.

Similar to the case of custom propositions in ColProof-M, the feedback dialog of SetSails! displays a special message when the learner has finished the exercise using a custom rule: “No mistakes were detected in your proof. However, you have inserted custom rules, making a fully automatic check impossible. Please send your solution to your tutor for revision.” It is then up to the tutor to decide whether it is reasonable to use the entered rule like that.

### 7.3. Further Tools Using the Feedback-M Framework

#### MoveIT-M

Apart from the reference implementations ColProof-M and SetSails!, the Feedback-M framework has been integrated in several other learning applications to show that it can be used in different domains and in varying technical contexts. One of those learning applications is MoveIT-M, which deals with plane isometries (FEST, 2010c). The final learning objective for users of this tool is to be able to simplify compositions of isometries (KRAUTER, 2005). To get to this point, learners can interactively explore properties of plane isometries in several so-called labs. They can arbitrarily switch between these labs by navigating with the so-called lab browser (FEST et al., 2011).

#### Cinderella applets

As an exploratory tool on a subdomain of geometry, it is natural that MoveIT-M features dynamic, interactive sketches. Like the visualizations in ColProof-M, they are based on Cinderella (see section 5.2). Because Cinderella applets were originally intended to be embedded in web pages, adaptations were necessary to integrate them into the Java Swing application MoveIT-M and to enable communication between the applet and the application. This communication was realized using the CindyScript interface (FEST, 2010b) and an event listener model (FEST, 2010a).

#### Textual and graphical feedback

Because of the intrinsically graphical nature of the learning environment, only some of the feedback that MoveIT-M provides is given in textual form. In one lab, the task is to estimate the position of the image which results from mirroring a figure. In this lab, the Feedback-M framework provides feedback on demand (see figure 7.9). In other labs,
7.3. Further Tools Using the Feedback-M Framework

The feedback is graphical rather than textual, e.g. by displaying auxiliary lines or by recoloring the figures to indicate whether their position is correct. Because the FeedbackListener interface (see section 6.3) had not been realized at the time Movelt-M was developed, textual and graphical feedback have not yet been coupled.

The Squiggle-M learning application deals with the mathematical concepts of mappings and functions. It uses the same lab browser as Movelt-M, with labs on topics such as injectivity and surjectivity, ladder diagrams, and function graphs (HIOB-VIERTLER and FEST, 2010). Cinderella applets are used to display interactive function graphs and mapping diagrams (FEST et al., 2011). In one lab, the learner has to read a proposition on injectivity and has to decide whether or not it is universally true. To show that he has not merely guessed the answer, but understood the principle, he has to either give mapping examples for which the proposition is true, or a counterexample which proves that it is false. After answering and giving examples, the learner can request textual feedback (see figure 7.10).

In both applications that are based on the lab browser, students can request tutorial feedback from the Feedback-M dialog. Furthermore, the main window of the lab browser contains a button which makes it possible to contact the tutors directly without opening the feedback dialog. This is different from the previously discussed tools, which expect learners to read the automatically generated feedback first (see section 6.4). While SetSails! and ColProof-M exercises are rather linear,
Chapter 7. Proofs of Concept

Figure 7.10.: The feedback dialog in Squiggle-M.

Movelt-M and Squiggle-M are exploratory learning environments. Due to their high degree of freedom, it is impossible to develop automatic tests that would cover all possible questions a student might have.

Finally, the ComIn-M learning application [Rebholz and Zimmermann, 2011] deals with the proof method of mathematical induction. It currently features ten different exercises. Each solution consists of four steps:

1. Defining an initial value for a variable, and showing that this serves as the base case.

2. Proposing the induction hypothesis (this step has been realized as a multiple-choice question).

3. Writing down the equation for the induction assertion.

4. Proving the induction assertion by algebraic transformation.

The algebraic transformations in the last step were inspired by Saraswati, a learning tool on systems of linear equations [Müller and Blescherer, 2005]. Like Saraswati, ComIn-M delegates all calculations...
to a remote server running a computer algebra system, namely Maxima\(^\text{10}\). Hence, it is the only learning application discussed in this chapter in which the electronic exercises and the intelligent assessment components (see section 4.1) are realized as separated pieces of software.

Unlike the other learning tools described in this chapter, ComIn-M is not a Java Swing application, but a web application realized with AJAX and SOAP web services. For this reason, it only uses the data structures and tests of the Feedback-M framework, not the feedback dialog. This shows that the model (class Problem) and the controller (classes SolutionChecker and Test) of the Feedback-M framework can be used separately without the view. The ComIn-M feedback display, which looks similar to the Feedback-M dialog, has been developed using HTML and JavaScript. The generated feedback and hint messages are displayed in a block element within the web page (see figure 7.11). When feedback is requested, several tests are performed, e.g. whether the base case is valid, the correct induction hypothesis has been added, and the induction hypothesis has been used in the algebraic transformations.

The automatic assessment of the transformations is performed by sending an AJAX request to the Maxima server, which checks the terms for equivalence. When no mistakes are reported, ComIn-M reports that the transformations are correct, but that it cannot evaluate whether they are sufficiently detailed (see figure 7.12). For this reason, it suggests to the learner that he should request feedback from his tutor when necessary. Thus, ComIn-M is another example of a learning tool with semi-automatic assessment.

Chapter 7. Proofs of Concept

Figure 7.11.: The beginning of a proof in ComIn-M, with a feedback message on the induction assertion.
Figure 7.12.: The end of a proof in ComIn-M, with a feedback message on the algebraic transformations.
8. Interaction Logging for Tutorial Feedback and Learning Analytics

After realizing a generic framework for feedback on demand and electronic exercises with intelligent assessment, three important parts of the tutor-in-the-loop model are still missing (see figure 4.1). This chapter describes the development and evaluation of these components in three iterations. The first iteration is focused on a logging mechanism which records the learner’s interaction with the electronic exercises and with the feedback dialog. The second iteration leads to a backchannel to make tutor feedback possible. Finally, the third iteration adds prototypical analytics support to help evaluate the logging data.

8.1. Solution Process Recording with Picorder

The main objective of the first development iteration was to test the acceptance of the learning tools with automatic feedback, and to test whether Jacareto is capable of capturing and replaying student interaction with these tools. During this phase, ColProof-M and SetSails! both included the Feedback-M framework. Tutorial feedback support was not part of this iteration. In order to minimize distraction of the learners, Picorder was chosen as the recorder. The graphical user interface of CleverPHL was deemed too complex for students who should concentrate on the interfaces of SetSails! and ColProof-M. However, CleverPHL was used for the evaluation of the records because of its analysis features (see section 5.1.2).

8.1.1. Enhancements of Jacareto

As described in section 5.1, Jacareto records can be stored as XML files, even though the overhead caused by XML tags has been criticized by Spannagel (2007, p. 248f.). Picorder and CleverPHL both use the same record XML format; however, CleverPHL stores additional data. Besides the record, loading and saving a CleverPHL session includes all its preferences and known starters (see section 5.1.2). Because CleverPHL expects the record and the session information to be stored in a
Chapter 8. Interaction Logging for Tutorial Feedback and Learning Analytics

particular directory, importing Picorder records is cumbersome. One has to create a new session, then import the record manually. In order to streamline data exchange between Picorder and CleverPHL, a new, unified file format was introduced. Even though the new Jacareto files have got the file extension .JAC, they are in fact ZIP archives. They contain the record XML file, and optionally may also include session information. In case these are missing, CleverPHL just assumes default preferences for the loaded session. For reasons of backwards compatibility, CleverPHL still offers a feature to load legacy sessions. A nice side effect of the new format is the size reduction which results from the ZIP compression. Smaller record file size also means that it is faster to transfer them from the student’s to the teacher’s computer. This effect was further enhanced by removing the structure from record files. When loading a session, it is easier to regenerate the structure from the linear record than to reconstruct it from XML. Saving is notably faster when omitting the structure.

One advantage of Jacareto is that it is platform-independent because it is based on Java. It is possible to capture user interaction on one computer and to replay and analyze it on another one, even when both devices run different operating systems. Nevertheless, there are differences between the various Java implementations, and Jacareto failed to take some of those into account. One of these issues concerns the classpath separator. On Windows machines, a semicolon is used to separate path entries, while a colon is used on Unix-like operating systems. This convention was adopted by Java for its classpath. Because the classpath is part of the starter information (see section 5.1.2), and the starter is saved as part of the record, the classpath had improper separators after transferring the record across operating systems. To overcome this problem, Jacareto was changed so that it ignores the path separator convention and uses semicolons on any operating system. Another limitation to platform independence was prompted by Jacareto starting target applications with the system default look and feel. Because there are different looks and feels on different platforms, this made the GUIs of target applications behave in an unexpected way, causing replay to fail. This issue was resolved by relying on the platform-independent MetalLookAndFeel instead.

Furthermore, support for additional types of mouse events had to be implemented. Firstly, recordables for mouse wheel events were implemented, as ColProof-M and SetSails! both require the user to scroll at various points. Secondly, ColProof-M requires drag and drop interaction to add propositions to the proof. In principle, Jacareto can handle drag and drop actions, as it simply treats them as a sequence of mouse-press, mouse-move, and mouse-release events. However, due

---

1 A more detailed description of this problem can be found in the Jacareto bug tracker: [http://sourceforge.net/tracker/?func=detail&aid=3000476&group_id=138313&atid=740403](http://sourceforge.net/tracker/?func=detail&aid=3000476&group_id=138313&atid=740403) (accessed 2012-08-31).
to a problem in Java, the mouse-release event on a DropTarget are not passed through the AWT event queue, making them unnoticeable for Jacareto. To still make it possible to record drag and drop events, the class DragAndDropWorkaround was added to Jacareto. For each new component detected by the ComponentsManager, it registers a DropTargetListener which forwards all drag and drop events to the AWT event queue.

Additional issues which had to be resolved in order to make Jacareto compatible with the developed learning applications are explained in (HERDING and SCHROEDER, 2012).

8.1.2. Adaptation of Target Applications

SCHROEDER and SPANNAEGEL (2006) state that “[in] principle, you can take any interactive software written in Java and use it together with CleverPHL”. However, they restrict their own statement in the same contribution. “Although many applications written in Java can be used together with CleverPHL, there are some problems which restrict the practicability of CleverPHL” (ibid.). As an example, they mention the problem which arises when the user interacts with a file selection dialog. Because the directory structure and the files of the capturing computer differ from the one on the replaying computer, the replay will fail at that point. As this problem cannot be resolved on the side of Jacareto, special versions of ColProof-M and SetSails! were created which spare file selection dialogs. The drawback of this solution is that it forces students to solve each exercise in a single session.

Another problem arises when the behavior of the target application is dependent on the result of a random number generator, as the effects of replaying a record are then unpredictable. ColProof-M shuffles the positions of unused proof steps (see figure 7.2), while SetSails! makes heavy use of a randomizer to generate distractors (see section 7.2.3). To enhance Jacareto compatibility, the influence of randomness was reduced. This could be done because the used class java.util.Random is in fact a pseudo-random number generator. By setting the seed of the randomizer to a number which only depends on the current state of the application, one can create a learning application which seemingly behaves randomly, but is consistent enough to replay a record.

---

A third problem that had to be resolved concerns applications with animations. For instance, the MoveIt-M learning tool shows a short animation when the user flips a geometric figure. In Jacareto, the capture and replay processes are timed by user input events. During animations, however, the state of the target application changes without any user intervention. The animations do not pass through the AWT event queue, hence Jacareto cannot record them. During replay, the animations can lead to timing problems, which can cause the replay to fail. This is especially problematic in the fast-forward mode of Clever-PHL (see section 5.1.1): it continues to emulate mouse clicks while the target application is still playing the animation. These inconsistencies leave the target application in an unexpected state. As a countermeasure, a special event type was introduced to notify Jacareto about animations. Before starting an animation, the target application can push a PauseEvent to the event queue. An attribute of this event holds the expected duration of the animation in milliseconds. When fast-forwarding, each PauseEvent interrupts the replay process to give the target application time to finish the animation.

All in all, one can conclude that certain types of Java applications are either incompatible with Jacareto or can only be adapted by modifying their source code. Jacareto expects the target application to run completely deterministically. The same sequence of user actions must always lead to the exactly same state of the target application. To make an application deterministic, it may be necessary to rewrite parts of it, and sometimes even to drop existing features.

8.1.3. Inclusion of Semantic Events

To ease the analysis of records, ColProof-M was extended so that semantic events are put into the record whenever an important user action is performed. This includes any interaction which may help the teacher comprehend the problem-solving approach:

- Opening an exercise. The exercise title is stored as an attribute of the event.
- Browsing the theorem book.
- Opening or closing the feedback, illustration, or graph dialog
- Browsing problem descriptions and hints in the feedback dialog. The full feedback texts are stored in the event.
8.1. Solution Process Recording with Picorder

- Moving a proof step (adding a step to the solution, removing it, or changing its position). This event includes the position inside the old component (e.g., a coordinate in the area of unused proof steps) as well as the position inside the new component (e.g., a line number of the proof table).

- Adding, removing, or changing a reason for a proof step.

- Editing or adding a custom proof step.

**SetSails!** features similar semantic events, with additional events for selecting terms and reasons. In order to avoid having to implement modules with dozens of classes for all the possible events (see section 5.1.3), a generic class `SemanticEvent` was created and added to the `Jacareto` module base so that it can be used by `ColProof-M`, `SetSails!`, and any other target application. Each `SemanticEvent` instance has a name and an arbitrary number of key-value pairs which hold the event attributes. Furthermore, a data set was added to export all semantic event data from a record.

### 8.1.4. Evaluation

Only parts of the tutor-in-the-loop model were realized in the first development iteration: Electronic exercises with intelligent assessment and a framework to provide automatic feedback on demand. The main goal of the first evaluation was to evaluate research question Q2, namely whether students would access the automatically generated feedback on their own.

#### Scenario

During the summer term of 2010, `ColProof-M` was used in the course “Einführung in die Geometrie” (“Introduction to Geometry”) at the Ludwigsburg University of Education (Herding et al., 2010). All course participants were students in mathematics teacher training. Among other topics, the course dealt with a set of geometric theorems (such as Thales’ theorem) which students were supposed to prove as an exercise. Traditionally, courses at the Ludwigsburg University of Education do not require students to hand in their assignments for assessment. In previous years, course participants who had worked on the proofs had done them on paper as a homework. The introduction of learning applications with intelligent feedback was just one part of the new course concept devised in the SAiL-M project (Bescherer et al., 2012). Another aspect was strengthening student autonomy: teachers...
suggest several tasks to the course participants, who are free to choose which ones of them they are going to work on (ibid.).

Deployment

At the time of the evaluation, there had never been a scenario in which learners were asked to use Jacareto on their home computers before. Despite the aforementioned improvements, Jacareto was still hard to install and configure for use with ColProof-M. Furthermore, it was unclear whether students would be able to locate the created record file and to forward it to their tutors for evaluation. For these reasons, students were not asked to use their private computers. Instead, the software was installed on the computers of a lab room at the Ludwigsburg University of Education. For four weeks, there were several tutorials at which students could use the computer room. Students worked in groups of two to four. A tutor was always present to provide technical assistance, but for content-related questions, students were instructed to consult the automatically generated feedback, rather than immediately asking the tutor. This was done because, in later development iterations, students were supposed to work on the exercises at home, without a tutor who could give direct feedback.

Assignments

There were four assignments on topics supported by ColProof-M (1. Euclidean theorem, 2. intersection of the perpendicular bisectors, 3. Thales’ theorem, and 4. medial triangle; see section 7.1.4), with a one-week working time for each proof. Because the tutors in Ludwigsburg considered the second exercise to be the most difficult one, the \texttt{--rc} mode of Picorder (see section 5.1.2) was used to give the students the first two steps of the solution in the form of an animated demo. Because of the autonomy concept of the course, students were not required to work on all four tasks.

Picorder setup

To relieve the students of having to start Picorder and the ColProof-M exercises via command-line parameters, a batch file was provided for each exercise. Each batch file starts Picorder in invisible mode (i.e. without the control center GUI), which then starts ColProof-M with the exercise and immediately begins to record in the background. The Jacareto capturing process is completely transparent to the students, except for the console window which pops up at startup to run the batch file.

Technical Challenges

Picorder is built to automatically save the record to disk as soon as the target application terminates. However, it turned out that this mechanism was unreliable in practice. During the second week, several records of solution processes were lost. Only Picorder logs were stored, and those gave no clue about why the JAC files were missing. At first, the \texttt{--rc} parameter was suspected as the cause of the problem, but tests
performed afterwards showed that this feature worked flawlessly. Instead, it turned out that the most probable reason for the data loss was that saving took a long time for the sessions, which were up to 45 minutes long according to tutors. Closing the console window in which Picorder was running (e.g. by forcing the computer to shut down) interrupts the saving process and discards the record. A profiling was performed to find bottlenecks in the saving mechanism. It turned out that, for each recordable added to the XML representation of the structure, the track model \cite{specht2004} was refreshed. For this reason, saving had a complexity of $O(n^2 \log(n))$, where $n$ is the number of recordables in the record. This problem was alleviated by removing the track model notification and no records were lost during the third and fourth week.

Nevertheless, it turned out that even for those records which were properly saved, the replay process was unreliable. There were several reasons for this, all of which needed tweaks to the Jacareto code base in order to make the records replayable. The first problem concerned mouse clicks performed to select a reason for a proof step. Reasons are selected using a combo box which is embedded inside the proof table (see section 7.1.1). A JTable which uses a custom TableCellEditor to include JComboBoxes inside table cells is an uncommon GUI layout which has never been tested before with Jacareto. During the capture process, mouse clicks were recorded both on the table and on the combo box. Consequently, the replay performed two clicks onto the same screen coordinate, causing the pop-up to open and to immediately close again. This led to an inconsistent state which caused replay to break down. This problem was resolved by extending the AWTEventRecorder with a duplicate detection which filters out consecutive mouse events which carry the same timestamp.

Secondly, the path representations which the components manager generated for each component (see section 5.1.1) turned out to be inconsistent. Like the previously described problem, this affected the combo boxes inside the proof table. The components manager counter mismatched between capture and replay, which caused the replay to stop because it could not detect the open pop-up menus. As a workaround, the counter was disabled for pop-up menus: because there can never be two pop-up menus on screen at the same time, the enumeration was

\footnote{Jacareto bug report – Saving large records takes very long: \url{http://sourceforge.net/tracker/?func=detail&aid=3002621&group_id=138313&atid=740403} (accessed 2012-08-31). After the first iteration, it was found out that saving can be further accelerated by entirely removing the structure from the file, and just to store the linear record. When loading a session, it is actually faster to regenerate the structure than to parse it from the XML file.}

\footnote{Jacareto bug report – Clicks recorded twice when combobox used as table editor: \url{http://sourceforge.net/tracker/?func=detail&aid=2895131&group_id=138313&atid=740403} (accessed 2012-08-31).}
unnecessary. Leaving it away made it possible to capture interaction with pop-up menus and properly replay it afterwards.

The third issue that caused the replay functionality to be unreliable did not affect normal replay, but only fast-forwarding. CleverPHL supports skipping delays between events to fast-forward to a selected position in the record (see section 5.1.1). These delays, however, are important for certain interactions. In particular, the TransferHandler, which is responsible for drag and drop in Java Swing, expects consecutive drag and the drop events to have different timestamps. To resolve this issue, an artificial one-millisecond delay is inserted before each mouse release event. This suffices to make drag and drop work in fast-forward mode. Despite all the described efforts, replay of ColProof-M sessions – especially in fast-forward mode – remains instable.

Log data analysis

Even though some records were not saved and not all sessions were fully replayable, the semantic events included in the records made it possible to analyze the solution processes (Herding et al., 2010). The following sessions were suitable for analysis:

- 12 of the 13 recorded sessions of the Euclidean theorem assignment: one participant did not perform any actions after starting ColProof-M.
- 4 of 13 sessions of the assignment on the intersection of the perpendicular bisectors: the remaining files were not saved, as reported above.
- 3 recordings of the Thales’ theorem assignment, due to a very low number of tutorial participants during that week.
- 5 of 6 sessions of the assignment on the medial triangle: one participant did not perform any actions.

The semantic events of each session were filtered and exported using the data set feature of Jacareto (see section 5.1.2), then analyzed in a spreadsheet application. Of the 28 evaluated sessions, 13 processes led to a correct solution. Regarding research question Q2 one can say that most groups did make use of the automatic assessment feature.

However, there were three sessions without any automatic feedback requests. As the students’ names were not recorded to preserve their privacy (all sessions were pseudonymized using the date and a counter), it remains unclear why they refrained from checking their solutions. One reason might be that they did not know about the feature, and did not recognize the button which opens the feedback dialog. To increase the visibility and affordance of this toolbar button, it got a textual label in later versions of ColProof-M, in addition to the icon. The labeled button can be seen in figure 7.2.

The exported semantic events were also analyzed to find out which types of problems were reported most often. For each session, semantic events were counted which represent reading a problem description or a hint. Problems were classified by the tests that found them (see section 7.1.3). The results can be seen in table 8.1. As expected, the ReasonNotInProofTest found no problem in any of the sessions; MultiOccurProblems did not occur either during the evaluation. For this reason, both problem types have been omitted from the table. To determine the number of distinct problems reported to each user, duplicates were filtered out. Such duplicate semantic events often occur when the user keeps browsing the problems feedback dialog back and forth. They can be detected by searching for problems with identical description texts and hint messages.

On average, students read 6.32 distinct problem descriptions per solution process, and took 0.89 hints. When comparing the hint numbers, there is no large difference between groups which managed to come up with a correct solution and those which did not. In contrast, there is a difference when it comes to the number of problem descriptions read. Successful groups made more use of the feedback mechanism (7.77 feedback messages read on average) than those who did not complete the task (5.07 messages). This is surprising at first, as one might think that good students (those who are able to finish the exercise) make less mistakes and thus have less opportunities to read about their mistakes. But on the other hand, one has to keep in mind that those groups which prematurely gave up used ColProof-M for a shorter time on average. Furthermore, it would be wrong to draw the conclusion that making more use of the feedback dialog automatically leads to better performance. On the one hand, there was one group of students which solved the assignment on the medial triangle reading only one problem description, and taking not a single hint. On the other hand, one group which received 15 distinct feedback messages was unable to come up with a solution for the Euclidean theorem.

Looking at individual feedback types, one can see that most problems occurred while selecting reasons for a proof step. This might be because German students are not used to the two-column format (see
Table 8.1.: For each problem type, this table shows the number of distinct feedback messages and hints (in parenthesis) read during each ColProof-M session.
8.1. Solution Process Recording with Picorder

section 7.1.1), in which each proof step must accurately be given reasons. The UnfinishedProblem, which tells the learner that what was to be demonstrated is still missing in his proof (and possibly commends him for being on the right track), was also reported to many students. This comes to no surprise: when students reason top-down, the Q. E. D. proposition is usually added last. For this reason, students will almost always see an UnfinishedProblem when they keep clicking on the “Next Problem” button in the feedback dialog.

The students primarily used hints not to get assistance with fixing mistakes, but to proceed with an unfinished solution. 20 of the 25 hints taken during the evaluation revealed a proof step to add or a reason to insert. The remaining five hints dealt with removing wrong propositions or reasons, or resolving circular reasoning. This shows that students are often able to locate and resolve mistakes with the information provided by the problem descriptions, while they need more directive assistance when they are stuck.

The session of group 07_19a is especially interesting because these students deviated from the standard solution by editing a proof step. The exercise on the medial triangle included a distractor, the proposition “BQ is half as long as AC”. The students realized on their own that this proposition is false for non-isosceles triangles. Instead of leaving the distractor out of their proof, they opted for editing the proposition in order to correct it, changing it to the correct statement “BQ is half as long as BC”. Afterwards, they added it to their proof. When the group later opened the feedback dialog to check their intermediate solution, the assessment component of ColProof-M was unable to evaluate the custom propositions (see section 7.1.3), and the students were told to contact their tutor. They complied by asking the supervisor of the lab session. Unfortunately, they were running out of time, and were unable to finish the exercise during the tutorial. They finished the exercise on paper at home, as they could not install Picorder and ColProof-M on their private computers.

Survey

A survey was conducted during the lecture at the end of the semester. In order to evaluate the subjective perception of the ColProof-M application and its feedback functionality, the questionnaire included one page with questions on the learning application. Of the 66 questioned survey participants, 53 filled in this page. Of those, 29 stated that they had used ColProof-M at least once. There was a steady decline in usage

\footnote{Note that the difference between the usage numbers stated in the survey and the number of Jacareto logs is a consequence of the fact that students worked in groups of two to four.}
Chapter 8. Interaction Logging for Tutorial Feedback and Learning Analytics

during the weeks in which the application could be used; this is a tendency which could already be observed in the evaluation of Jacareto log files. Part of this is a natural consequence of the autonomy concept of the course: students who had already solved the first exercise were not formally required to work on the remaining proofs (in fact, only seven participants stated that they had used ColProof-M for all four assignments).

While student autonomy can explain some cases of disuse of ColProof-M, there were also students who explicitly stated that certain aspects of the application had kept them from using it. One student complained about the necessity to work in the lab room, and five stated that they simply did not have enough time for the assignments. The most striking answers, however, were those of students who plainly rejected computers as means of assistance to solve assignments. Four students said so in the survey, one of them even using the words “Bin kein Computerfreak” (“I ain’t a computer geek”). Obviously, such a negative attitude toward computer-supported learning in general is impossible to counter with improvements to the learning application alone. Students who are technophobic will not try out the learning software on their own accord, so they will not find out about possible advantages it has to offer unless they are told so by fellow students or teachers. Still, a combination of technological improvements and encouragement by the teaching staff can lead to piecemeal advancements of the acceptance of technology-enhanced learning.

With the goal of improving ColProof-M and the Feedback-M module, the course participants were asked about positive and negative experiences with the learning application. 34 students wrote in specific commendations and criticism of the interaction with ColProof-M. Their comments were clustered to get an overview of the topics of highest importance. Students’ opinions were split on the usability of the application. Five of them criticized the interaction design for being inconvenient or time-consuming, while two said that the user interface was clearly arranged and easy to use, and one stated that the application could be used with little effort.

The survey participants were also of two minds about the fact that ColProof-M evaluates solutions based on a sample solution. On the one hand, eight students liked the fact that they could start with given propositions, which gave them a clue how to approach the task. On the other hand, six participants addressed the problem that it is hard to follow an individual approach because it is difficult to deviate from those given propositions. It became clear that students desired an application which can generate possible solution steps and suggest them to the students, but still allows for different solution paths.
Six participants commented on the part of ColProof-M which is most important in the scope of this thesis, namely the feedback component. Besides the concerns regarding the sample solution discussed earlier, all those comments were positive. One student appreciated the solution checking feature in general. Four others specifically liked the hint functionality, which helped them during their solution processes. One student commended the application because it enabled them to work self-reliantly. There was no complaint about the fact that the number of hints was limited to prevent cheating. On the contrary, there was a student who appreciated these limited tries as a challenging factor.

Further points of criticism expressed by single students include the lack of documentation and the fact that finished proofs could not be saved: Ergebnissicherung (saving the results) is an important didactical principle which the participating students had previously been taught. It was possible to print out the solution while in the lab room, but students could not save the files and transfer them to their private computers. Three participants noted that their solution processes sometimes led to trial-and-error approaches; however, one student cited exactly this possibility to explore the solution space as an advantage, and another one said that it was a good challenge that the exercises contained distractors (theorems which are irrelevant for the current proof). Finally, single students said they liked ColProof-M components such as the adaptive illustration, the book of theorems, or the two-column format.

Conclusion

During the first development iteration, the setup of the interaction capturing was based on the Picorder command-line interface for Jacareto. This installation did not support transfer of the record to the tutor (unless the student would manually look up the record in the file system and send it to his tutor), a fundamental part of the tutor-in-the-loop model. For this reason, Picorder only served as a provisional solution to test the interplay of ColProof-M and Jacareto in a context which came close to lab conditions. Nevertheless, it performed worse than expected even in this technically simplified scenario. During one week, the data of several records was lost because Picorder was not properly shut down at the end of the exercise. Even though Picorder is supposed to be fully transparent to the user, there is still a possibility for the learner to inadvertently interfere with it and disrupt the recording process, causing the record not to be saved to disk.

Comments on the feedback component

Further comments

Weaknesses of Picorder

---

7 At the time of the first evaluation, the manual for ColProof-M was missing. In the meantime, it is available online: [http://sail-m.de/sail-m/ColProof-M_guide](http://sail-m.de/sail-m/ColProof-M_guide) (accessed 2012-08-31).
Reflecting the analysis of interaction records, one can say that on the one hand, the Jacareto replay processes could rarely be used to get an insight into the solution processes of the students. Part of this was caused by technical difficulties, but more important was the fact that watching entire replays is very time-consuming. On the other hand, the semantic events, after being filtered out from the complete interaction record, made it possible to review and compare the students’ approaches. In particular, it was possible to analyze their use of the feedback and hint functionality. Common mistakes could be found by summarizing the recorded sessions; however, extensive manual work with external spreadsheet software was required for this analysis.

All in all, the feedback capabilities of ColProof-M have been largely appreciated by the students who took part in the evaluation. However, there is a relevant number of users who desired a learning application which provides a more intelligent kind of assessment which fully supports different solution paths. Even though ColProof-M can check correctness of the proof format regardless of the contents, the validation of semantical correctness, which is based on a sample solution for each exercise, turned out to be insufficient to satisfy the needs of these students. This is why the focus of the evaluation was shifted to another learning tool: during the following iterations, the SetSails! learning application was used. Other than ColProof-M, it has a true intelligent assessment component which can assess arbitrary solution paths.

In conclusion of the first iteration, the research question Q1 (see section 4.2) could be regarded as only partially affirmed. It was possible to realize electronic exercises which can automatically assess a student’s intermediate solution. A modular feedback component was implemented to provide feedback and hints on demand. Finally, a logging mechanism was set up to capture the solution process. Still, the tutor-in-the-loop model had not been fully realized at this development iteration: the analytics module was missing, just like the channel to request tutorial feedback over the Internet. As seen in table 8.1 nearly all students made use of the automatic feedback functionality. This indicates that research question Q2 can be affirmed; nevertheless, there were some students who did not request feedback despite being unable to finish the exercise. One group of students who came up with an extraordinary solution asked their tutor for feedback, which is in line with research question Q3. Nonetheless, further evaluations are necessary to find out whether students would also request tutorial feedback when they are not in class. Finally, the collected log files could be used to find out which types of problems were most common, and to see which students had extraordinary problems. This shows that the semantic events provide enough data for the use cases described in research question Q4. However, because the analytics module was still missing, much manual work was required for this evaluation.
8.2. Jacareto as a Library

The second development iteration of the logging component had the primary objective of introducing a channel which allows students to request feedback from their tutors, thus implementing one central part of the tutor-in-the-loop model. Besides this aspect, a number of issues had arisen during the first iteration which were supposed to be resolved. Students should be enabled to use the learning applications on their private computers, rather than be asked to use the computers in the lab room. To fulfill this requirement, the system needed to be completely platform-independent and easy to set up. Furthermore, a mechanism was needed to transfer logs over the Internet, so that tutors could give adequate advice to students for whom the automatically generated feedback had been insufficient.

8.2.1. Integration of Jacareto into Learning Applications

The Picorder system, which had been used to capture user interaction during the first evaluation, is problematic for two reasons. Firstly, its record saving routine had failed several times, leading to data loss during multiple sessions. Secondly, provided that the saving mechanism works, it stores the record on the user’s hard disk. It is difficult for the learner to find it inside his file system and to forward it to a teacher, even though transfer of sessions has been made easier by the JAC file format (see section 8.1.1).

As can be seen in figure 5.3, Picorder and the target application are loosely coupled. After the target application has been started, the communication flow is unidirectional. The only connection between Picorder and the application is provided by the module base, which makes it possible to post semantic events to the record. In this way, the application can append events to the record, but it cannot read from the record. This makes it impossible for Feedback-M (which is part of the target application) to attach the log to tutor feedback requests. Worse still, the user cannot even manually send the log to his teacher at the moment he has a question. Because the learning application (and thus Picorder) is still running, the record has not been finalized in that situation, and cannot be reopened. This means that the student would have to quit the learning application to send the log file. This contradicts the feedback-on-demand principle, which is a foundation of the tutor-in-the-loop model. For this reason, a completely new approach to saving and transferring record files was required.
8.2.2. Extensions to the Capture and Replay Mechanism

The new capturing mechanism is inspired by the classes `CaptureExample` and `ReplayExample` which can be found in the experimental `Jacareto` package `jacareto.trial`. Both are technology demonstrations which display a set of dummy GUI components. They demonstrate how to use `Jacareto` to capture interaction (or replay it, respectively) without using the `CleverPHL` or `Picorder` user interface. Instead of relying on these tools, the `CaptureExample` is linked directly to the `Jacareto` engine and controls the recording process by itself. This means that it is a target application which has full control over where to save the record file. Similarly, the `ReplayExample` can open any record file and replay its contents on its own GUI.

Based on the operating principle of these technology demonstrations, an adapter was developed which makes it possible to integrate capture and replay functionality into any application. This adapter comprises the classes `CaptureIntegration` and `ReplayIntegration`. The former has methods to start and stop the capturing process and to save and access the record. The latter offers methods to load and replay a record. Target applications which use the capture integration can post semantic events to the record using the established module base mechanism. Similar to the module base JAR file, the capture and replay integration adapter is provided as a JAR file which can be added to the classpath of the target application.

The `CaptureIntegration` was included in the `SetSails!` learning application. The recording process is completely invisible to the user – even more transparent than `Picorder`, of which the user can see a console window (see section 8.1.4). Capturing begins automatically at the moment an exercise is loaded. When the learner saves the exercise, the `CaptureIntegration` persists the current state of the record and adds the JAC record file to the `SetSails!` exercise file. The `.SETX` exercise files of `SetSails!` are in fact ZIP archives which contain an XML file. JAC files can simply be appended to these archives.

Even though it is technically possible to also integrate a replay mechanism into `SetSails!` (the `Jacareto` framework contains a working technology demonstration which uses the `ReplayExample` class), this possibility was waived. Replaying the entire file in real time is just one way for teachers to understand the students’ actions. As stated before, other important `Jacareto` features include fast-forwarding to a certain position, viewing semantic events, or exporting them for statistical analysis. These functions would not be available with a transparent replay integration. Instead, it was decided that tutors should use the full-fledged `CleverPHL` user interface to evaluate students’ sessions.
Unlike the Picorder setup, in which one needs a starter holding the initialization data of the target application (see section 5.1.1), SetSails! is now directly started by the user (see figure 8.1); therefore, a starter is not required for capturing. For replay, however, CleverPHL expects the record to contain a starter recordable. For this reason, the CaptureIntegration class generates a starter using runtime information (e.g. the classpath and the current working directory) which is read from Java Virtual Machine system properties. This artificial starter is then added to the record for CleverPHL. After extraction of the JAC file from the exercise file, the teacher can open the session for evaluation. In order to view the replay, it may be necessary to change the working directory, as SetSails! may be installed in different directories on the student’s and the teacher’s computer.

A command-line parameter called -replay was added to SetSails! As explained in section 5.1.1 a target application starter can define different command-line parameters for capturing and for replaying. This is used to set the -replay parameter only at replay time. It signals that the application should not start another capturing process in the background. This is not the only modification to the target application which was required to ensure replay stability. As described in section 5.1.1, there may be segments in the record which should not be replayed (Schroeder and Spannagel, 2006). For instance, if the user had chosen to print his proof, the teacher would not want the interaction with the print dialog to be replayed. The replay would either become inconsistent and crash because the teacher has installed different printer drivers, or cause an undesired printout. Similarly, interaction with the file open and save dialogs should be skipped during replay, and so should be the interaction with the tutor-in-the-loop feature. A teacher would not want to receive a duplicate of the feedback request just because he had watched the replay. To accommodate for this problem, two special event types were introduced. When the user opens one of the mentioned dialogs, a SkipInReplayEvent is posted to the record. A corresponding ContinueReplayEvent is added when the dialog is closed again. These events, when encountered during replay, cause CleverPHL to skip all events that occur in-between. The source code of the SetSails! target application had to be modified so that a pair of these events is pushed to the event queue whenever a print, open, or save dialog is shown.

Because the Jacareto record is saved inside the exercise file, it is possible for students to save and later reopen an exercise while still offering tutors a complete replay. When opening an exercise file containing a record, newly captured events will simply be appended to that record, thus extending the previous session. This feature was implemented to accommodate students who had, during the first iteration, criticized the lack of Ergebnissicherung (see section 8.1.4). Though extending a session makes it possible to analyze all the events which
led to a solution, there is a problem when trying to replay the combined records. As described in section 5.1.1, Jacareto keeps track of the GUI components of the target application by assigning each one a unique component path. Paths of similar components are numbered consecutively. After an exercise is reopened, however, this numbering restarts with zero. To make up for this inconsistency, a special event called \text{ResetComponentsManagerEvent} has been introduced. \text{Set-Sails!} posts it to the record when reopening an exercise. When Jacareto comes across it during replay, it resets the \text{ComponentsManager} to get the component path numbering in line with the record.

![Diagram of the capturing process with Jacareto integrating into the target application. Compare this to the capturing process of Picorder depicted in figure 5.3.]

**Figure 8.1.:** The capturing processes with the Jacareto capturing mechanism integrated into the target application. Compare this to the capturing process of Picorder depicted in figure 5.3.

### 8.2.3. Transfer of Records

With the capturing mechanism in place, the question was then how to transfer the recorded sessions to the tutors so that they could address feedback requests. The idea of including the feedback channel into the learning management system (LMS) was quickly discarded. There is a multitude of LMS's on the market\textsuperscript{8}, and even though there are standards such as SCORM and LOM which specify how to make learning contents available, there is no common interface to communication channels such as forums.

\textsuperscript{8} For instance, the Ludwigsburg University of Education uses the Moodle LMS, while the Heidelberg University of Education uses Stud.IP. Both institutions took part in the SAiL-M project and were scheduled for the evaluation of this development iteration. The software should be usable regardless of the deployed LMS.
Instead, the standard e-mail format was used to realize the feedback channel. [HUETT](2004) lists several advantages of e-mail as a feedback medium, among them motivational aspects and the ubiquity of this communication protocol. To encourage usage of the mechanism, the feedback dialog calls for the learner to contact his tutor in certain situations (e.g., when the student has defined a custom rule, as described in section 7.2.3). Beyond that, in accordance with the feedback-on-demand principle, the tutor can be asked at any time about any problem. The learner can select his tutor from a drop-down menu, then send him a question (see figure 6.4). The list of tutor e-mail addresses and other settings of the feedback dialog are defined in a properties file which is shipped with SetSails! E-mails are sent via an SMTP server provided by the SAiL-M project. By default, the current state of the solution (which includes the Jacareto record) is attached to this mail.

Using e-mail as a feedback channel is, unfortunately, not free of risk. The private e-mail address is a piece of private information, and many people are reluctant to declare it (for instance because they fear an increase of spam). Nevertheless, [BORCEA-PFITZMANN and STANGE](2007) have found out that, on average, users are more willing to disclose their personal e-mail address for purposes of e-learning than they would be for other applications in general. There may be students who fear that revealing their identity during a request for help might cause their teacher to give them worse grades. However, an e-mail address

\[9\] A different SMTP server can be set by modifying a configuration file of Feedback-M.
does not necessarily identify a person. Most of the people who are very
concerned about disclosing their personal e-mail address in fact have
multiple e-mail accounts (ibid.).

As can be seen by comparing of figures 4.1 and 8.2 the result of this
second iteration already comes close to being an implementation of the
tutor-in-the-loop model as described in chapter 4. The students work
on electronic exercises in the GUI of the SetSails! learning application.
Their intermediate solutions are automatically assessed with the help of
a model of algebraic terms and rules. Other than the sample-solution-
based proof check of ColProof-M, this is a form of intelligent assessment
because it can dynamically evaluate arbitrary solution paths. From the
SetSails! GUI, the learners can access feedback on demand by using the
Feedback-M functionality. In case the automatically generated feedback
is insufficient, a feedback request can be sent to a tutor, who will reply
by e-mail. To assist the tutor in understanding the student’s approach,
an interaction log recorded by the built-in Jacareto library is attached to
the tutorial feedback request.

One important discrepancy between the tutor-in-the-loop model and
this realization, however, is the way interaction logs are made accessi-
ble to the tutor. According to the tutor-in-the-loop model, logs of all
learners should be made accessible to the tutor to allow him not only
to better assist individual students, but also to statistically evaluate the
performance of the entire course in the sense of learning analytics. In
contrast, the Jacareto-based realization only sends the tutor the logs of
those students who have requested tutorial feedback, and the trans-
ferred session records end after the feedback request. A complete col-
collection of all students’ interaction logs is impossible because SetSails!
is a desktop application which can be used without an active Internet con-
nexion. Furthermore, even students who are online during their solu-
tion processes might become suspicious of unexpected network traffic
and deny the establishment of a connection using a desktop firewall.
For this reason, the logging component can only serve to review indi-
vidual solution processes, but not to collect exhaustive statistical data.

8.2.4. Evaluation

The SetSails! application with integrated Jacareto capturing mechanism
was evaluated during two courses for math teachers: one course dur-
ing the winter term of 2010/2011 at the Ludwigsburg University of
Education, and another course which was held one semester later at the
Heidelberg University of Education. Like during the first develop-
ment iteration, students were free to choose whether or not to use the
offered applications, and submitting assignments was not mandatory.
For the Ludwigsburg version, the feedback dialog contained a list of all

university teachers and tutors who were involved in the course. The Heidelberg lecture was held without the help of student assistants, so feedback requests had to be directed to the professor.

Evaluation in Ludwigsburg

Unfortunately, not a single student in Ludwigsburg made use of the tutor feedback functionality, even though some could not finish the exercises on their own and might have benefited from assistance. Because of the way the transfer of Jacareto records was realized (logs being attached to tutor requests), no interaction recordings were available. For this reason, there was not enough data of SetSails! usage to repeat the analysis of error types that had been done earlier with ColProof-M (see table 8.1).

At the end of the summer term of 2010/2011, a survey was conducted at the Ludwigsburg University of Education. During a lecture, a questionnaire was handed out which was similar to the one used half a year earlier to evaluate ColProof-M (see section 8.1.4). 88 students took part in the survey. Of those, only 36 had chosen to use SetSails! for at least one exercise. The remaining 52 participants noted various reasons not to use the software. These reasons, which were written into a free text field, were clustered for evaluation; some students gave two or more reasons. 19 students stated that they prefer paper and pen over learning software, with four people claiming that working with a computer would not help to prepare for the exam, which was to be solved on paper. Eight said that they do not like working on exercises with a computer. This generalized rejection of computer-supported learning was similar to the situation in the geometry course half a year earlier, in which four students showed signs of technophobia (see section 8.1.4).

Ten students had tried to use SetSails!, but had given up due to technical problems during the installation. Five students said that they had no Internet connection to download the software, or no access to a computer on which to install it. Five other participants stated that they did not have time to work on the exercises.

While two students mentioned that they had problems handling the SetSails! user interface, three commended its clarity, and one said that the input mechanism was easy to use. Several survey participants highlighted helpful parts of the user interface which are similar to those of ColProof-M, namely the rule book (mentioned 14 times), the dynamically generated Venn diagram visualizations (two times), and the tabular representation of the solution (two times).

Similar to the ColProof-M evaluation during the first iteration, four students commended SetSails! for giving them a list of options from which
they could choose the next solution step. One participant pointed out that the distractors initially irritated him. This was an expected reaction; in fact, the distractor functionality was implemented to make the exercises more challenging in the first place (see section 7.2.3). Besides this, there were only few negative comments on the possible solution paths which the learning application had to offer. Unlike the proof steps in ColProof-M, the term options of SetSails! are dynamically generated based on the previous steps which the user has taken. This explains why, compared to the first iteration, there were less students who felt restricted by the given options. Two survey participants claimed that the automatic assessment had rejected transformations which they had considered correct. Subsequent tests did not uncover any bugs, therefore it is likely that the students had either overlooked a miscalculation, or had tried to perform multiple transformations in one line, as had also been suggested by a student in Heidelberg (see next section).

Regarding the feedback component, there were two survey participants who wrote positive comments about the possibility to automatically check one’s solution. Two further students commended the hint functionality. On the other hand, eight participants wished for additional or more helpful assistance. This gap was supposed to be filled out by the tutor-in-the-loop principle which, unfortunately, was not used by a single student in Ludwigsburg. Interestingly, one survey participant said anyway that she liked the possibility to ask her tutor questions. Another student, who complained that the hints were not very helpful for her, said: “ich wollte nicht jedes mal die Tutoren nerven” (“I did not want to annoy the tutors each time”).

**Evaluation in Heidelberg**

During the summer term of 2011, one student at the Heidelberg University of Education sent three messages to her professor over a period of eight days. The first one generally praised the SetSails! learning tool. Afterwards, she asked whether it was acceptable to perform two transformations in one line. This is not possible with the notation of SetSails! (cf. appendix A), unless one defines a custom rule. The third comment included the question whether there was a shorter solution than the one she found for a particular exercise. Her solution was a correct proof, even though she had performed some unnecessary transformations. The professor could answer both questions by reviewing the attached solutions. The advantage for the student was that, because she knew her solution would be attached, she could write very brief messages and did not have to describe her approach in textual form. No other student took the opportunity to request assistance from the professor.
Due to the way the log collection mechanism had been implemented, no records were available for analysis except for the ones of the student who sent requests to her tutor. Even though a low number of collected records was expected from the experiences in Ludwigsburg a few months earlier, there was not enough time to reengineer the logging component so that records of all students (regardless of whether they contacted their tutor or not) would be collected in a central repository. Instead, a more comprehensive survey was created with a focus on how feedback – both automatically generated and from the tutor – was used. Unlike the questionnaire used in Ludwigsburg, it mostly consisted of Likert scales and yes/no questions; only few free-text fields were used. The questionnaire was split up into three parts: two similar sequences of questions on the ComIn-M and SetSails! learning applications, and a set of general questions on exercising with and without a computer.

Of 32 students who took part in the survey, only eight stated that they had used SetSails!, and 12 said they had used ComIn-M. Therefore, parts of the questionnaire which dealt with details of these learning applications (e.g. “Did the feedback on problems or mistakes fit to your approach?”) could not be reliably evaluated due to the small sample size. Only one of the ComIn-M users and none of the SetSails! users said that the e-mail function had been used to ask the professor for feedback (obviously the student mentioned above, who had requested SetSails! feedback from the professor, did not participate in the survey). Students who did not ask their professor were asked why they refrained from doing so. Some of those students stated that they had not needed any help or that the automatic feedback was sufficient. Other participants said that they did not want the teacher to see their individual solution, or that they preferred peer feedback.

Conclusion

During the first two development iterations, enormous efforts were made to simplify the installation process of Jacareto and the learning applications. Despite the fact that Jacareto was integrated as a library and required no configuration at all, several students said that their computer skills were insufficient to unpack and start SetSails! and ColProof-M. It is therefore desirable to make learning applications accessible via a web browser without any installation requirements.

Even though several Jacareto bugs have been fixed to improve replay stability, replaying a record remained unreliable. Due to the nature of Jacareto, replay failures are very hard to avoid when capturing and replaying on two different environments (e.g. varying operating systems, screen resolutions, etc.). However, experience from the evaluations has
shown that these replays are often not necessary. Usually, teachers do not have enough time to watch the entire replay anyway. Instead, it is possible to skim through the list of events, with a focus on the semantic events. In fact, the non-semantic events are superfluous in this situation, and need to be filtered out to analyze the solution process. This is in line with the assumption of GUÉRAUD et al. “A trace of elementary actions on the interface (e.g. mouse clicks at such and such locations...) would be useless because such events are too far removed from the task semantic and do not really inform on the learner’s difficulties” (2006, p. 477).

Attempts to perform learning analytics on the recorded solution processes were impacted by the fact that only few logs were available. Due to privacy concerns, logs were only transferred via e-mail when the student requested tutorial feedback. One way to resolve this conflict between privacy rights and interests of learning analytics is to anonymize or pseudonymize the logs (see section 2.6). However, this is impossible to realize with traditional e-mail transfer: because students’ e-mail addresses usually contain their names, they cannot be used for pseudonymous communication.

Some students refrained from asking their tutors for feedback not because of privacy concerns, but because they prefer discussing their solutions with their fellow students. Peer feedback is a sensible alternative to automatic and tutor assessment (see section 2.2). While it is not part of the tutor-in-the-loop model, a peer assessment feature might be a useful addition.

8.3. SMALA

In the third development iteration, it was decided that radical measures had to be taken to overcome the difficulties encountered during the previous evaluations. The Jacareto-based logging mechanism showed several intrinsic weaknesses which hindered tutorial feedback and learning analytics. For this reason, a new software for logging-based learning analytics was developed in the course of the SAiL-M project (LIB-BRECHT et al., 2012). This system is called the SAiL-M Architecture for Learning Analytics (SMALA)\(^{10}\) and is currently in alpha stage. Unlike Jacareto, which is a desktop application, it is a web application written in Java EE. Because SMALA is web-based, it is possible to make the learning applications available via a web browser. This solution overcomes the installation requirement which discouraged students from using the applications with Jacareto.

\(^{10}\) [SMALA](http://sail-m.de/sail-m/SMALA_en) (accessed 2012-08-31).
While the focus of Jacareto is on replaying interactions, SMALA concentrates on learning analytics. For this, SMALA uses the established concept of semantic events, but drops other event types such as mouse movements and keyboard presses. These have turned out not to be helpful for analyzing solution processes. Consequently, there is no more replay feature; the analysis is based purely on captured semantic events. The basic class Event, like its Jacareto counterpart, has a type, a description string, and attributes in the form of key-value pairs. By dropping the replay feature, SMALA forgoes the level of detail which has proven unnecessary with Jacareto-based analytics. Instead, it aims at a higher level of semantic representation than comparable tools (see figure 8.3).

### 8.3.1. Development

For each deployment of a learning tool in a course, a so-called activity configuration has to be defined on the SMALA server. This is done by creating an XML file which contains all the information required to start and initialize the learning tool[11]. This metadata includes the URL of the learning tool and parameters such as the language. In this respect, it parallels the starters of the Jacareto framework (see section 5.1.1). An activity configuration can inherit attributes from another one, which

---

serves as a prototype. This makes it possible to adapt an existing tool for a new course without redefining all attributes: the prototype is an abstract configuration defining the default setting for a tool. Inheriting configurations override these settings and extends them with course-specific information.

To make it easy for students to access the learning tools from their accustomed environment, SMALA allows LMS integration. Currently, there are two supported LMS’s: Stud.IP and Moodle. The ALIIA plug-in\(^\text{[12]}\) is used to embed SMALA activities into Stud.IP course rooms. The advantage of this plug-in is its ability to authenticate the LMS users toward SMALA. For Moodle, there is no similar feature to forward the user to an external site. Instead, SMALA realizes a screen scraping approach to identify the learner. There is a minimal Java applet which silently loads the Moodle user preference page in the background and extracts the name and e-mail address from it. This applet is signed so that it can initiate this HTTP request. There are three reasons why SMALA needs these data. Firstly, the system can recognize the learner when he later starts another exercise. This is important for learning analytics. Secondly, the learner can be identified to provide tutorial feedback. Thirdly, it is possible to create exercises which are only accessible to course participants.

Pseudonymization

In the surveys of the second iteration, some students raised concerns that tutors might lower their grades after seeing their solution processes. This might prevent them from requesting feedback, or even from using the learning tools at all. For this reason, SMALA stores logs not under the students’ names, but under pseudonyms. In the current version, the pseudonym is generated by concatenating first name, last name, and e-mail address, and calculating the MD5 sum of this string. This is only a prototypical implementation, as it does not completely fulfill the privacy requirements. The MD5 algorithm is known to be weak\(^\text{[WANG and YU, 2005]}\). Hence, it might be replaced with a more secure hash such as SHA-1 in future versions of SMALA. Furthermore, teachers, who have access to the list of course participants, can easily calculate the hashes of all students and use it to identify them by their pseudonym. This problem could be resolved by initiating each hash calculation with a salt which is unknown to the teachers.

Applets

To make it possible to run the learning tools in a web browser, applet versions of SetSails!, ColProof-M, Movelt-M and Squiggle-M were created. The graphical user interface of a Java application can be converted to an applet by using the JApplet class instead of JFrame. Above that, modifications have to be made so that dialog windows (e.g. the feedback dialog) are displayed properly. Finally, the feature to save and later reopen exercises was removed again. Like in the first

iteration, students are expected to finish the exercises in a continuous session. Otherwise, it would have been too difficult for tutors to comprehend solution processes which are split up into several sessions.\footnote{In the second iteration, a feature was implemented to automatically join consecutive sessions (see section \ref{sec:iteration2}). However, this mechanism turned out to be too complicated and unreliable, which is why no attempt was made to reimplement it in SMALA.}

The Feedback-M framework, which has formerly been a stand-alone project (see chapter \ref{chap:feedback}), was integrated into the SMALA client API. This makes it easier for developers of learning tools to realize automatic assessment functionality. The feedback dialog configuration – e.g., the tutors’ contact data – is no longer read from the user’s hard disk. Instead, the settings are passed as applet parameters after being defined in the activity configuration. The feedback dialog reports semantic events when the learner requests feedback or asks for a hint.

Because one cannot expect tutors to regularly visit the SMALA website to check for incoming questions, they receive a notification e-mail for each feedback request. Unlike the first two development iterations, questions to a tutor are also modeled as events in SMALA. Hence, each tutor feedback request is part of a session. This makes it possible for tutors who view a log to see exactly at which point the student asked him for feedback. However, he could also use this information to identify...
the student. The student’s e-mail address is part of each feedback request because e-mail is used to realize the backchannel in SMALA (see figure 8.4). This information makes it possible for the tutor to circumvent the pseudonymization. Nevertheless, this only concerns sessions which contain a tutor feedback request. One may assume that students who initiate communication with their tutor are willing to reveal their identity.

Being a web application, it is obvious that SMALA follows a client-server approach. Most learners never directly interact with the server. Instead, they simply launch the client (the learning tool) from their LMS. The client then sends events to the SMALA server (see figure 8.5). SMALA uses a semantic event concept which is very similar to that of Jacareto. This analogy can be seen in the Event base class which, like its Jacareto counterpart, has a timestamp, a description, and a list of properties.

One major difference between Jacareto and SMALA events is that those of SMALA can contain binary data. This can be used to include screenshots or other representations of the current solution state in an event. Binary contents are not supported in Jacareto because it is built to save records in XML format, which is not suitable for representing binary contents. SMALA uses the property list (plist) format to transfer events between client and server. This format is similar to the widespread JSON format, but additionally supports binary data. For each type of event, there is a client-side and a server-side event class. After being transferred from the client to the SMALA server in plist format, the event is converted to the server-side representation. Each server-side event class is tagged with annotations of the Java Persistence API so that it can be stored to a relational database.
Because SMALA is written in Java EE, its client API is only available to learning tools which are also written in Java. Nevertheless, it is possible for other web applications to mimic the behavior of the SMALA client. This was done to add logging functionality to WebDale, a sandbox environment to learn PHP, HTML, and MySQL (Königshofen, 2011). As WebDale itself is written in PHP, its code cannot be linked to the SMALA API. Instead, WebDale emulates the SMALA client: it generates the property list code of persistent events and sends it directly to the SMALA server. Because WebDale, being a website, cannot take a screenshot of itself, it offers an alternative way of attaching the current solution state to feedback requests. When the learner sends his tutor a message, WebDale saves its current state to its internal database and assigns a URL to it. The tutor can later visit this URL to see the state of the learning tool at the time of the feedback request.

Teachers can view the students’ sessions after logging in to the SMALA website. A session is represented by a linear, chronological list of its semantic events. By default, each event is shown with its timestamp and its description; this is done by the DefaultEventRenderer. For event classes which carry additional metadata, it is possible to create a subclass of EventRenderer which generates more detailed JSP output. This concept is similar to the Editors in CleverPHL, which are created to make the attributes of semantic events visible (see section 5.1.3). Each SMALA activity is associated with a so-called RendererProvider, which is responsible for mapping event types to event renderers. For example, the event renderer for the events triggered by Feedback-M show the problem description and hint text which were presented to the learner. SetSails! events which describe how the user has selected a transformation rule include the name of that rule as visible metadata.

In addition to the chronological representation of the solution process, tutors require a static snapshot to see the solution state at the time of a feedback request. In CleverPHL, the solution state can be seen by waiting for the replay to finish and then looking at the target application (provided that the replay does not fail). This is not possible in SMALA due to the lack of a replay function. Instead, learning tools can implement the ScreenShotProvider interface of the SMALA API to realize a screenshot mechanism. A snapshot of a Java applet can be generated by calling Swing routines to draw the GUI into a BufferedImage. The screenshot is then attached to the feedback request (see figure 8.6).

The log views are meant to help tutors understand the solution process of a single student. The main use case for this is answering a student’s feedback requests. However, the tutor-in-the-loop model features other use cases which involve not a single log, but the logs of
Figure 8.6.: Sample of a tutor feedback request in SMALA (LIBBRECHT et al. 2012). The event contains a screenshot showing the state of the solution at the time of the request.

the entire course. Firstly, the teacher may want to detect common mistakes and find out whether certain exercises were too easy or too difficult (see section 4.1). Secondly, a student may be unaware that he needs assistance, or may be too shy to ask for help. Such cases have appeared during the second iteration (see section 8.2.4). For such analyses, SMALA offers mechanisms to generate overviews of the logs of an activity. Developers of learning applications can override template methods of the SummaryViews class to realize two types of overviews. Firstly, a dashboard summary can give an overall summary of an activity. One could, for example, generate a tabular representation of read feedback messages, similar to table 8.1. Such a dashboard summary can be seen in figure 8.7. Secondly, it is possible to create a user activity summary. This is an overview of the performance of single students. For instance, one could create a table for each student that shows for
### Bearbeitete ComIn-M-Aufgaben:

<table>
<thead>
<tr>
<th>Aufgabe</th>
<th>Gesamtanzahl</th>
<th>richtig</th>
<th>falsch</th>
<th>Lösung Prüfen</th>
<th>Tipps</th>
<th>Tutoranfragen</th>
</tr>
</thead>
<tbody>
<tr>
<td>induction1</td>
<td>33</td>
<td>21</td>
<td>12</td>
<td>177</td>
<td>31</td>
<td>3</td>
</tr>
<tr>
<td>induction10</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>21</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>induction2</td>
<td>21</td>
<td>16</td>
<td>5</td>
<td>99</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>induction3</td>
<td>7</td>
<td>12</td>
<td>5</td>
<td>39</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>induction4</td>
<td>6</td>
<td>11</td>
<td>5</td>
<td>46</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>induction5</td>
<td>10</td>
<td>14</td>
<td>4</td>
<td>75</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>induction6</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>25</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>induction7</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>23</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>induction8</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>58</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>induction9</td>
<td>4</td>
<td>9</td>
<td>5</td>
<td>25</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 8.7.: A SMALA dashboard summary view, showing numbers of correct and wrong solutions, feedback and hint requests, and tutor questions for each the ComIn-M exercise. The table shows data from the 2011/2012 course at the Heidelberg University of Education. Note that, due to a programming error, some sums displayed are incorrectly.

Each exercise whether he has successfully solved it and how many feedback messages and hints he has accessed. There are tabular dashboard summaries and user activity summaries for the ComIn-M application (Libbrecht et al., 2012).

Besides those tabular overviews, it is also possible to implement graphical dashboard summary views. For instance, there is a graph feature which displays the number of daily hits of ComIn-M exercises during the last week. This function was realized as a JSP page which feeds session information to a JavaScript snippet. The jqPlot library\(^\text{14}\) is then used to draw a timeline (see figure 8.8).

\[^{14}\text{jqPlot}: \text{http://www.jqplot.com/} (accessed 2012-09-02)\]
Figure 8.8: A dashboard summary view in SMALA, showing the number of sessions during the previous week. Note that the depicted graph only shows dummy data.

Besides the automatic and tutorial feedback which is part of the tutor-of-the-loop model, SMALA also allows for other types of assessment. There is a prototypical implementation of a peer feedback mechanism. Instead of requesting feedback from a tutor, learners can choose to send their question to one of several channels. This concept is similar to an online forum. For each activity, there can be multiple channels for topics such as the learning tool, the contents, or specific exercises. So far, the channel feature is only a proof-of-concept. Furthermore, learners can log in to the SMALA server to view their own logs. This feature is not only useful for self assessment purposes, but also for privacy reasons: it provides transparency of how tutors see the solution process.

8.3.2. Evaluation

A first version of SMALA was evaluated at the Ludwigsburg University of Education during the winter term of 2011/12. The ComIn-M, Squiggle-M, and SetSails! learning tools were embedded into the Moodle course room of the “Mathematik betreiben 1: Zahlen und Operationen” (“Applying mathematics 1: numbers and operations”) lecture. Like in previous semesters, students were free to work on their exercises either with or without the tools.

Shortly after the students had finished working with the learning tools, but before any analysis had been done, an upgrade to the SMALA software was installed. This upgrade changed the relational schema of the database. Such a modification requires all existing events to be converted to the new schema. During this conversion, all events which are stored in the relational database are serialized to property list format, then saved to the database again using the new schema. Unfortunately,
there was a programming error in the conversion script. Instead of creating one file for each session, all the sessions were saved under the same filename because a counter was not properly incremented. This led to sessions being overwritten, and thus to a great loss of data.\footnote{SMALA bug tracker – Stream storage produces only a file: \url{http://youtrack.i2geo.net/youtrack/issue/SMALA-49} (accessed 2012-08-31).} Using backups of the virtual machine which was running the SMALA server, it was possible to restore some sessions.\footnote{SMALA bug tracker – Recovery of lost log entries from DB snapshots: \url{http://youtrack.i2geo.net/youtrack/issue/SMALA-50} (accessed 2012-08-31).} However, inconsistencies in logs indicated that some events were still missing after the recovery. For this reason, the logs could not be reliably evaluated.

At the end of the semester, a survey was conducted during the lecture. It was similar to the questionnaire described in section 8.2.4 containing questions on the used tools with a focus on feedback usage. Of the 85 participants, 27 (31.8\%) stated that they had used \textit{SetSails!} at least once. The question whether the automatic feedback was considered helpful showed that the feature, while still leaving room for improvement, was generally regarded as positive (see figure 8.9). Of the 27 students who had used \textit{SetSails!}, six (22\%) said that they had asked their tutor or professor for assistance. The remaining ones were asked what prevented them from asking for help. Nine stated that they needed no help or that the automatic feedback had been sufficient. Two students said that they preferred asking questions during tutorials, and one reported that he found it hard to articulate his problem. Another student claimed that she did not want to reveal her personal information. This shows that, contrary to the assumption of section 8.3.1, there are students who insist on pseudonymity even when it comes to communication with tutors.

Finally, the survey asked the question whether the participants prefer solving their mathematics exercises on paper or with a computer. An overwhelming majority of 96\% claimed to favor paper, while only a single participant said that he rather uses a computer. This striking rejection of e-learning, which has already been reported in section 8.2.4, might be the main reason why only a minority of course participants used the \textit{SetSails!} learning tool.

At the same time as the evaluation in Ludwigsburg took place, students of the Heidelberg University of Education had the chance to use the \textit{SetSails!} and \textit{ComIn-M} learning tools. Both were embedded to the \textit{Stud.IP} course room of the “Foundations of Mathematics I (Primary School)” course. Only a minority of the students made use of the learning applications: of the 46 participants of the end-of-semester survey, only 14\% stated they had used \textit{SetSails!}, and 24\% said that they had used \textit{ComIn-M}. Like in Ludwigsburg, solving exercises on paper was preferred by the vast majority of students (96\%). As only six students
Chapter 8. Interaction Logging for Tutorial Feedback and Learning Analytics

Figure 8.9: Answers to the question "How helpful was the automatic feedback while solving the exercises?" in the 2011/2012 evaluation at the Ludwigsburg University of Education. Students could answer using a six-level Likert scale.

used the SetSails! application, the sample size is too small for a reliable analysis of feedback usage.

Failed third evaluation round

A third evaluation of the SMALA environment was planned for the summer term of 2012 at the Ludwigsburg University of Education. Unfortunately, due to a SMALA bug, the logs of SetSails! were not stored on the server. For this reason, no analysis could be performed for this course.

Conclusion

All in all, the SMALA-based realization covers all aspects of the tutor-in-the-loop model. Nevertheless, the evaluation showed that it still suffers from stability problems. Due to technical problems, the logs of several solution processes were not properly persisted, and could therefore not be analyzed. In particular, this affects the evaluation of the research questions Q3 and Q4 (section 4.2), which depend on these logs. Still, these problems were only caused by smaller programming errors which either are easy to fix or have already been resolved. In contrast, the difficulties during the first two development iterations were due to intrinsic weaknesses of the Jacareto environment. Hence, using

17 SMALA bug tracker – persistence exception preventing sessions: [http://youtrack.i2geo.net/youtrack/issue/SMALA-64](accessed 2012-08-31).
SMALA instead of Jacareto for the logging component of the tutor-in-the-loop implementation was a reasonable decision, and continued development of the SMALA framework will certainly lead to a reliable assessment system.
9. Summary and Perspectives

Closed and semi-open exercises like multiple-choice tests restrict the learner to the solutions anticipated by the developer or teacher. This is contrary to learner-centric paradigms which foster abilities such as reasoning, solving complex problems, and applying one’s knowledge in real-world contexts. Open-ended tasks are more difficult for a computer to assess. The traditional approach of intelligent tutoring systems involves sophisticatedly tracing a model of the solution and of the student’s knowledge. The tutor-in-the-loop model was designed as an alternative which reduces the development cost while still offering students adequate feedback. Following the semi-automatic assessment paradigm, the computer evaluates standard solutions and detects typical mistakes. Any students who come up with extraordinary solutions or need further help can receive feedback from a human tutor. Because he does not have to assess the standard solutions, the tutor has time to look at feedback requests in detail: he can access a log of the learner’s interaction with the electronic exercise to comprehend the solution process. Furthermore, the logs can be used for learning analytics, e.g. to detect mistakes which are common in the entire course or to find students who need additional support despite not asking for tutor feedback.

The Feedback-M system for feedback on demand was developed as a generic framework for learning applications with open-ended tasks. Besides feedback texts, it supports optional hint messages which provide knowledge on how to proceed. The data structures of the framework have been successfully used to realize various desktop and web applications with semi-automatic feedback. For Java Swing applications, the framework features a dialog window to display the generated feedback messages. A listener mechanism is available to graphically highlight faulty parts of the student’s solution. From the feedback dialog, students can access a function to directly contact their tutors in case the automatic feedback is insufficient.

To provide helpful feedback, the tutor must regard not only the submitted intermediate solution, but the entire process that led to it. Initially, the Jacareto capture and replay toolkit was chosen to realize the logging component of the tutor-in-the-loop model. Unlike screen recorders, Jacareto records the user’s interaction on an event level. This makes it possible not only to replay the interaction, but also to export the event
data for statistical analysis. However, during evaluations it was found out that the replay function is not as important as expected because tutors often cannot find the time to watch an entire solution process. Furthermore, evaluations have shown that Jacareto suffers from three intrinsic disadvantages. Firstly, the replay function is unreliable, and despite all efforts to improve stability, some solution processes cannot be completely replayed. Secondly, Jacareto itself does not provide analytics support. Export, analysis, and visualization of data has to be done manually using external tools. Thirdly, because Jacareto is not web-based, it is difficult to deploy it on the students’ computers and to collect the log files afterwards. For these reasons, it was decided to replace it with a new logging mechanism.

The newly developed SMALA framework was used to collect and analyze logs during the third and last development iteration. Because SMALA is web-based, it can be used without any client-side installation. Unlike Jacareto, it does not record interaction on a mouse-movement level, and thus spares the replay mechanism which was deemed unnecessary in the previous evaluations. Instead, it offers summary views and visualizations which can be used for learning analytics, e.g. to detect common mistakes or to find and assist weaker students. Despite still being in alpha stage, SMALA already supports multiple learning tools and has been used in several courses.

Several learning applications with intelligent assessment have been developed to show that the tutor-in-the-loop model is applicable to various domains and can be realized using different technologies. Among those applications are ColProof-M, which deals with deductive proofs, and the SetSails! tool for algebraic transformations. Both applications incorporate the Feedback-M framework. While ColProof-M relies on a sample solution to assess each exercise, SetSails! uses a term replacement system for model tracing. This makes it possible to assess any transformation sequence based on the supported algebraic rules. When a student derives from these and defines a custom rule, assessment is delegated to a tutor. After initially supporting capture and replay with Jacareto, the learning tools were switched to SMALA-based logging during the third development iteration.

Regarding the research questions listed in section 4.2, question Q1 could not be affirmed with the Jacareto framework. Its analysis features were insufficient for learning analytics, and the installation requirements deterred several students from using Jacareto-compatible electronic exercises. After replacing it with the newly developed, web-based SMALA environment, research question Q1 could be affirmed. The combination of SMALA and the learning applications covers all components of the tutor-in-the-loop model. Nonetheless, there is still potential for extending its learning analytics functionality, and the SMALA framework needs further improvements in terms of stability.
One focus of the evaluation was on the students’ usage of automatic feedback: research question Q2 deals with whether learners are aware of their difficulties and request automatic feedback when needed. Unfortunately, several technical problems led to partial data losses during two of the three iterations. This affected the logs of several solution processes during the first evaluation with the Picorder-based setup, as well as those of two courses during the third iteration with SMALA. Furthermore, only few logs could be evaluated in the second iteration because the log transfer was coupled with the tutorial feedback request mechanism. The log file analysis during the first iteration showed that students indeed do make use of the automatic feedback mechanism as intended, even though there were a few group of students who refrained from doing so. This observation is consistent with survey results during all three iterations.

Research question Q3 dealt with tutorial feedback requests. It was not evaluated in depth during the first iteration because the tutorial feedback channel had not been implemented at that time. During the second and third iteration, several students used the tutorial feedback mechanism to submit finished solutions, either to prove their success or to ask whether there is a more elegant solution. Furthermore, some learners asked for help while they were working on an exercise because they were stuck. Yet, the log data shows that many students could not finish exercises and gave up without asking their tutors for help. For this reason, the assumption that students would contact their tutors when automatically generated feedback is insufficient cannot be universally confirmed. Measures should be taken to encourage more students to make use of the tutorial feedback mechanism when needed. Among the reasons for rejection which were named by students in the surveys were privacy concerns and preference of peer feedback over tutorial assessment.

As described in research question Q4, there are two use cases for which a tutor can use the recorded log data. He can inspect the log of a single solution process to answer a feedback request of a single student, or he can analyze the logs on a course-wide level. Even though fewer students requested tutorial feedback than expected, all those questions could be answered using the information provided by the logging component. During the time Jacareto was used, the replay feature was not needed to comprehend the students’ approaches and to give tutorial feedback. This is why no such function was implemented in SMALA. Extracting course-wide information was difficult with Jacareto, as all records had to be manually exported and analyzed with external tools. Since Jacareto has been replaced, this analysis can be done directly on the SMALA website. However, as of yet, there are only few dashboard summary views in the form of tables and graphs. Furthermore, even if a tutor noticed a student group in need of assistance, the pseudonymization would prevent him from approaching them unless
they request feedback. For these reasons, the course-wide aspect of re-
search question \(Q4\) has not yet been evaluated.

**Outlook**

Because some of the research questions could not be fully affirmed
during the evaluation, it is not yet possible to verify the primary as-
sumption that the tutor-in-the-loop model enables students to retrieve
feedback on their individual learning processes. It may be possible to
confirm it during further evaluations in the future. This, however, re-
quires several improvements to the software which was presented in
this thesis. In particular, the logging and analytics components need
to be developed further. The following outlook highlights some possible
features for various components which would further improve the
realization of the tutor-in-the-loop model.

According to \cite{Zimmermann2011}, the feedback dialog of the
*Feedback-M* framework could be improved by offering a progressive
hint model. Independent thinking could be fostered if a hint request
would only lead to a vague hint. Only students who need additional
help would then ask for a further, more explicit hint. Nevertheless, im-
plementing such a multiple-stage hint model would be risky for three
reasons. Firstly, \cite{Aleven2000} point out that some students tend to immediately click through to the most detailed hint.
Hence, cheating prevention mechanisms have to be put in place to de-
ter students from requesting a second hint before the first one has been
pondered over. Secondly, implementing a progressive hint model is
a usability challenge. The current feedback dialog is minimalistic to
reduce extraneous cognitive load, and this should be considered be-
fore adding more controls to it. Thirdly, more effort would be required
from developers of learning applications because multiple hint texts
would have to be written for each problem type. These three issues
should be taken into account when adding a progressive hint model to
*Feedback-M*.

**Logging API’s for further programming languages**

All learning applications which have been connected to *SMALA* so far
are written in Java: they are either realized as applets or as Java EE web
applications. The only exception to this is *WebDale*, which is written
in PHP. As it could not be linked to the *SMALA* API, it emulates the
*SMALA* client by sending raw event data directly to the server (see sec-
ton 8.3.1). Besides the API for Java, there should be *SMALA* API’s for
various programming languages to make it easier to connect learning
tools created with different web technologies.

**Additional visualizations**

*SMALA* features several different functions that show tabular and
graphical overviews of student activity. So far, there are only proto-
typical overviews for the *ComIn-M* application, and none for the other
learning tools. For this reason, tutors were unable to perform learn-
ing analytics while students worked on their exercises. Additional
overviews could be implemented to realize a full dashboard for each
learning application. Such a visualization would make it easier to find students in need of help and to detect common misconceptions among course participants.

One use case of the tutor-in-the-loop model is not supported by the current SMALA-based implementation. When tutors analyze the logs and notice a student who is stuck, but who did not request tutorial feedback, there is no way for the tutor to contact him. This is caused by the pseudonymization feature, which is meant to protect the students’ privacy. It would be possible to develop a mechanism for pseudonymous communication. The students’ contact data (e.g., e-mail addresses) can be known to the system without being visible to the tutors. This would allow tutors to use SMALA to approach students in need of help while still maintaining their privacy.

Once a system has been set in place to keep the learners’ personal data securely stored on the SMALA server while only displaying pseudonyms to tutors, it could be extended to improve the learning analytics capabilities. Currently, the pseudonymization prevents correlations between usage of exercises and factors such as gender, academic major, or exam grades. The SMALA system could offer a feature that allows the tutor to upload these pieces of information, which would then be matched to the exercise results. This would allow him to perform correlation analyses without seeing any personal data, making privacy-conformal learning analytics possible. Nevertheless, developers of such a system have to exercise caution. It must prevent tutors from deliberately or inadvertently circumventing the students’ pseudonymization. For instance, in a course with very few male students, it should not be possible to correlate gender and exercise success (Dyckhoff et al., 2011).

As pointed out in chapter 2, computers and tutors are not the only possible sources of feedback. During the evaluation, multiple survey participants stated that they prefer peer feedback to tutorial assessment. The channel feature of SMALA should make it possible to request feedback from fellow students, but so far it is only a prototypical implementation. For instance, there is no notification mechanism which would let students know that someone has asked for help, and pseudonymous communication between students is missing as well. With further improvements, it may be possible in the future to extend SMALA to allow feedback from a network of fellow students, friends, and external domain experts. With these extensions, the tutor-in-the-loop model could progress to become the network-in-the-loop model.
A. Examples for Proofs in Set Theory

Proof by Cases

The following is an example for a proof by cases in set theory. To some extent, it follows the format used by SOLOW (2010, p. 175ff.).

Proposition: If $A$, $B$, and $C$ are sets, then 
$$(A \cup B) \cap (A \cup C) \cap (B \cup C) = (A \cap B) \cup (A \cap C) \cup (B \cap C).$$

Using the definition for equality of two sets, this is true if and only if
$$(A \cup B) \cap (A \cup C) \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C) \cup (B \cap C)$$
and
$$(A \cap B) \cup (A \cap C) \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C) \cap (B \cup C).$$

To show the first statement, one has to show that for every element $x \in (A \cup B) \cap (A \cup C) \cap (B \cup C)$, it holds that $x \in (A \cap B) \cup (A \cap C) \cup (B \cap C)$.

Let $x \in (A \cup B) \cap (A \cup C) \cap (B \cup C)$.

Thus $x \in A \cup B$ and $x \in A \cup C$ and $x \in B \cup C$.

Show that $x \in A \cap B$ or $x \in A \cap C$ or $x \in B \cap C$.

Case 1: Assume that $x \in A$.

Case 1a: Furthermore, assume that $x \in B$.

Thus $x \in A \cap B$. ✓

Case 1b: Furthermore, assume that $x \notin B$.

Because of $x \in B \cup C$, $x \in C$.

Thus $x \in A \cap C$. ✓
Case 2: Assume that \( x \notin A \).
Because of \( x \in A \cup B, x \in B \).
Because of \( x \in A \cup C, x \in C \).
Thus \( x \in B \cap C. \checkmark \)

To complete this proof, the second statement,
\[
(A \cap B) \cup (A \cap C) \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C) \cap (B \cup C),
\]
needs to be considered as well. This can be done with a similar case differentiation, and is left out here for the sake of brevity.

**Proof by Equivalence Transformations**

The following is an alternative proof for the correctness of the previous equation. It shows equivalence of both terms by applying rules of the algebra of sets.

\[
\begin{align*}
(A \cap B) \cup (A \cap C) \cup (B \cap C) & \quad | \text{Distributive law} \\
= (A \cup ((A \cap C) \cup (B \cap C))) \cap (B \cup ((A \cap C) \cup (B \cap C))) & \quad | \text{Absorption law} \\
= (A \cup (B \cap C)) \cap (B \cup ((A \cap C) \cup (B \cap C))) & \quad | \text{Absorption law} \\
= (A \cup (B \cap C) \cap (B \cup ((A \cap C) \cup (B \cap C)))) & \quad | \text{Distributive law} \\
= (A \cup B) \cap (A \cup C) \cap (B \cup ((A \cap C) \cup (B \cap C))) & \quad | \text{Distributive law} \\
= (A \cup B) \cap (A \cup C) \cap (B \cup (A \cap C) \cap (B \cap C)) & \quad | \text{Commutative law} \\
= (A \cup B) \cap (A \cup C) \cap (A \cup B \cap (B \cup C)) & \quad | \text{Idempotency law} \\
= (A \cup B) \cap (A \cup C) \cap (B \cup C) & \quad | \text{Distributive law} \\
\end{align*}
\]

A proof using the stricter notation in which the union and intersection operations are binary would be longer, as it would require the commutative and associative law to be applied several times.

A proof for a shorter equation which describes a property of the set difference operator:

**Proposition:** If \( A, B, \) and \( C \) are sets, then \( (A \setminus B) \cap C = (A \cap C) \setminus B. \)

\[
\begin{align*}
(A \setminus B) \cap C & \quad | \text{Set difference} \\
= (A \cap B^c) \cap C & \quad | \text{Associative law} \\
= A \cap (B^c \cap C) & \quad | \text{Commutative law} \\
= A \cap (C \cap B^c) & \quad | \text{Associative law} \\
= (A \cap C) \cap B^c & \quad | \text{Set difference} \\
= (A \cap C) \setminus B & \quad | \text{Set difference}
\end{align*}
\]
B. Bibliography


KELLER, John M. (1999). Using the ARCS Motivational Process in Computer-Based Instruction and Distance Education. New Directions for Teaching and Learning, 1999(78):37–47.


THORNDIKE, EDWARD L. (1903). Educational Psychology. Lemcke and Buechner.

USISKIN, ZALMAN (1980). What should not be in the algebra and geometry curricula of average college-bound students. Mathematics Teacher, 73.


C. List of Tables

1.1. Different feedback timings and sources. .................................................. 3
2.1. Results of the CMU Lisp Tutor post-tests ............................................. 23
2.2. Results of the ACT Programming Tutor post-tests .............................. 24
8.1. Feedback usage in ColProof-M ............................................................. 116
D. List of Figures

2.1. Bloom’s revised taxonomy .................................................. 6
2.2. Rationales for constructive feedback ..................................... 8
2.3. Students receive semi-automatic feedback ............................... 13
2.4. Classification of assessment activities .................................... 14
2.5. The process of learning analytics ......................................... 32

3.1. The OUNL Exercise Assistant ............................................ 37
3.2. The PACT Geometry Tutor ............................................... 39
3.3. Architecture of DUESIE .................................................. 40
3.4. The grading view of ALOHA ............................................. 41
3.5. Processing of a request in T-Prox ....................................... 45
3.6. Comparison of existing interaction analysis tools ...................... 46

4.1. The Tutor-in-the-Loop model ............................................ 50

5.1. The Picorder control center ............................................... 59
5.2. The CleverPHL main window ........................................... 60
5.3. The capturing process of Picorder ...................................... 61
5.4. Unstructured and structured records in CleverPHL .................. 64
5.5. A Cinderella applet ..................................................... 67

6.1. Class diagram of Feedback-M ........................................... 71
6.2. The Feedback-M dialog .................................................. 73
6.3. Class diagram of the problem highlighting mechanism ............ 76
6.4. The tutor feedback dialog ................................................ 77

7.1. Example for a two-column proof ........................................ 80
7.2. ColProof-M with feedback dialog ...................................... 82
7.3. Dynamic illustration in ColProof-M ..................................... 83
7.4. ColProof-M proof graph ................................................ 84
7.5. Bottom-up transformations in SetSails! ................................. 93
7.6. Finished exercise in SetSails! .......................................... 95
7.7. SetSails! custom term entry ............................................ 97
7.8. Mix-up of rules in SetSails! ............................................ 99
7.9. MoveIt-M with feedback dialog ........................................ 101
7.10. Squiggle-M with feedback dialog ..................................... 102
7.11. Beginning of a ComIn-M proof ......................................... 104
7.12. End of a ComIn-M proof ............................................... 105

8.1. The capturing processes with Jacareto integration .................... 124
8.2. Realization of the tutor-in-the-loop model with Jacareto ........... 125
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3</td>
<td>Comparison of existing interaction analysis tools with <em>jacareto</em> and <em>SMALA</em></td>
<td>131</td>
</tr>
<tr>
<td>8.4</td>
<td>Realization of the tutor-in-the-loop model with <em>SMALA</em></td>
<td>133</td>
</tr>
<tr>
<td>8.5</td>
<td>Architecture of <em>SMALA</em></td>
<td>134</td>
</tr>
<tr>
<td>8.6</td>
<td>Tutor feedback request in <em>SMALA</em></td>
<td>136</td>
</tr>
<tr>
<td>8.7</td>
<td><em>SMALA</em> dashboard summary for <em>ComIn-M</em></td>
<td>137</td>
</tr>
<tr>
<td>8.8</td>
<td>Dashboard summary view in <em>SMALA</em></td>
<td>138</td>
</tr>
<tr>
<td>8.9</td>
<td>Student acceptance of automatic feedback in <em>SetSails!</em></td>
<td>140</td>
</tr>
</tbody>
</table>