Logging of High-Level Steps in a Mechanized Math Assistant

Bakkalaureate Thesis
at IICM,
Graz University of Technology

Franz Kober
kober@student.TUGraz.at

April 24, 2012
# Contents

1 Introduction 2
   1.1 A joint project on ISAC’s dialogs ........................................ 2
   1.2 State-of-the-art in dialog architecture ................................ 4
       1.2.1 History and current state ......................................... 4
       1.2.2 Confirmation of ISAC’s original design .......................... 6
   1.3 ISAC’s high-level steps in doing mathematics ......................... 7
   1.4 The most urgent tools for dialog authoring in ISAC? ................. 8
   1.5 Review of ISAC’s ‘old’ User Logger .................................. 9

2 A “User Logger” for ISAC 11
   2.1 Review of ISAC’s existing dialog architecture ....................... 11
       2.1.1 Classdiagram .................................................... 11
       2.1.2 Interaction diagrams .......................................... 12
   2.2 Design considerations: DialogGuide and UserLogger ............... 22
   2.3 Integration of the User Logger ..................................... 24

3 Implementation and administration of the project 25
   3.1 Installation of the experimental system ................................ 25
   3.2 Dialog architecture: analysis ....................................... 27
   3.3 Detailed design of the UserLogger .................................. 28
   3.4 Integration of the UserLogger ....................................... 29
   3.5 Develop and test the usecases ..................................... 30

4 Guidelines to UserLogger for dialog authors 32

5 Summary and conclusion 34
# List of Figures

1.1 Interaction in the Seeheim Architecture ........................................ 5
1.2 Interaction in the MVC Architecture ............................................. 6
1.3 Basic ZSAC architecture for calculations ....................................... 7

2.1 Overview in form of a classdiagram ................................................... 11
2.2 SCOPE of interaction diagrams ....................................................... 12
2.3 Interaction diagram for first click on a formula ................................ 13
2.4 Interaction diagram for “append a formula” ................................... 14
2.5 Interaction diagram for “replace a formula” .................................... 15
2.6 Interaction diagram for “Show tactic context” .................................. 16
2.7 Interaction diagram for “Next Button” ............................................. 18
2.8 Interaction diagram for “Login Screen” .......................................... 19
2.9 Interaction diagram for “Login” ..................................................... 19
2.10 Interaction diagram for “Open BrowserDialog (Example, Method, Theory or ProblemBrowser)” ................................................................. 20
2.11 Interaction diagram for “Dependency of Browsers” ............................ 21
2.12 Interaction diagram for “Open an example” .................................... 22

3.1 Overview dialog architecture .......................................................... 28
3.2 UserLogger Design ........................................................................... 28
3.3 Schema of Test-Driven-Development .............................................. 30

4.1 Database record next step ............................................................... 33
4.2 Database record final result .............................................................. 33
4.3 Database record picking a tactic ....................................................... 33
Abstract

This Baccalaureate Thesis is joint work with [Kie12] the comprehensiveness of the task to reorganise the dialog of \textit{\$ISAC\$} which is under development at TUG. The goal of the reorganisation is to allow non-programmers to adapt learners’ interaction with \textit{\$ISAC\$}; these non-programmers probably will be called “dialog authors” and shall be able to concentrate on cognitive science and educational theories.

The challenges expected for dialog authors result from the power of an emerging new generation of educational mathematics assistants \textit{\$ISAC\$} where is a prototype of. These assistants are based on Theorem Proving (TP) technology. TP-based systems are expected to cover the whole range of stepwise solving mathematical problems, to check free user input generously and reliably and to be able to automatically generate a next step. These features raise questions like: Which parts of the next step shall be presented to the learner, the rule to apply or the formula? Which part of formula or rule should be omitted according to error patterns? In which situations is the learner allowed to request a next step (not in exams)? Etc. Implementation work which wants to seriously cope with such questions cannot cope with Java programming at the same time. Such implementation work requires an appropriate development environment for future dialog authors.

The two theses together accomplish decisive steps towards such a development environment for dialog authors. The theses review the history and the state-of-the-art in the architecture of dialog-based applications, research a considerable collection of available tools and select two tools for implementation in \textit{\$ISAC\$}. One of the tools is a rule-engine which allows to describe the sequence of interactions by simple rules. The rule-engine is the core of an expert system which is commercially applied in the development of business solutions, but the rule-engine is free. The other tool is a standard relational database which shall serve as a first approach to a user history enabling simple operations on the history.

This thesis focuses the second tool, the database. After a careful analysis of the existing dialog architecture and of the implementation in \textit{\$ISAC\$} a UserLogger has been implemented. Both steps, analysis and design as well as implementation are described in detail. The variant found for implementation is very code saving variant and also appropriate for dialog authors: each record in the database is a high-level step of interaction. A step comprises the user (in the multi-user system), the input of the user and the response of the system checking the user input. Such a step is called “high-level” for good reasons: a step promotes the construction of a solution within a logical context (and a user not aware of the context will fail to do such a step). Or such a step concerns a lockup in the underlying mathematics knowledge, which is also sensitive to the context.

The thesis provides preliminary elements of guidelines for future dialog authors and an extensive bench of use cases implemented for various experiments. The thesis concludes with a preview on tasks for dialog authors which are expected to succeed based on the technology provided by this thesis.
Chapter 1

Introduction

1.1 A joint project on ISAC’s dialogs

ISAC is a prototype of an emerging generation of (educational) mathematics assistants based on Theorem Proving (TP) technology. The prototype has been under construction already since 2002¹. The process of prototyping also clarifies theoretic foundations and contributes [Neu] to the emergence of a new academic field “TP-based educational mathematics systems” addressed by two workshops presently, one focusing technology ² and one focusing education ³.

By now the characteristics of the new TP-based generation has been clarified as follows:

1. TP-based systems cover the whole range of mathematical problem solving: The range starts from formal specification of the problem (more or less prepared and hidden from the learner), continues through stepwise construction of a solution within a logical context and finishes with an automated proof that the solution found fulfills the post-condition of the specification.

2. TP-based systems have all underlying mathematics knowledge in a human readable format: Following the LCF-paradigm [GMW79] this knowledge is mechanically proved from “first principles”; thus this knowledge is structured by deductive aspects in analogy to comprehensive textbooks in mathematics, but with links which can be followed interactively. ISAC experiments with separate structures for specifications (application-oriented aspect) and for methods (algorithmic aspect).

3. TP-based systems allow to check user input generously and reliably: Given a logical context from Pt.1 above, user input establishes a proof situation: Can the input step be derived from the context? TP technology is the most general technology to answer such a question reliably. The automation of such checks by TP technology enables to model stepwise problem solving in software. (Without TP technology the check for each step required separate code, which cannot be accomplished for variants desirable in problem solving.)

4. TP-based systems can combine deduction and computation such that automated generation of next steps is possible [Neu12] in general at any step of problem solving. So we can expect such a system to suggest a next step if the learner gets stuck at some step in problem solving. “Automated generation” of these steps requires one program describing the solution of the respective problem class, a program written in an

¹http://www.ist.tugraz.at/projects/isac/ and http://www.ist.tugraz.at/isac/
²THedu http://www.uc.pt/en/congressos/thedu
³eduTPS http://sites.dmi.rs/events/2012/CADGME2012 working group “Theorem-Prover based Systems (TPS) for Education”
emerging kind of TP-based programming language [HKN10]. Lucas-Interpretation [Neu12] of such a program can cope with a very wide range of variants of user input within steps towards a solution of the respective problem class.

Given these features, TP-based systems have the potential to tackle a kind of questions which are out of reach for systems like CAS (Computer Algebra Systems), DGS (Dynamic Geometry Systems) and Spreadsheets, the most general tools available for the construction of educational systems. Here are some examples drawn from early work [KN08] on dialogs in ISAC:

• Which parts of the next step shall be presented to the learner, the rule to apply or the formula resulting from rule application? The rule at which level (elementary, a whole simplifier, etc)?

• Which part of formula or rule should be omitted? Does the current step concern difficulties encountered (error patterns) in previous sessions of the individual learner, or in the respective course?

• In which situations is the learner allowed to request a next step? During assessment sessions (written exams, etc) learners won’t be allow in general, but how establish a continuum between learning and evaluating?

• How can the wealth of research and experience in didactics of mathematics concerning instructional design and design of learning scenarios made be fruitful for user modeling and dialog design?

• How can the wealth of research and experience in didactics of mathematics concerning misconceptions made be fruitful for dialog design? Error patterns?

• Etc.

Such questions are not meant to be tackled by software engineers; serious approaches to such questions can only be expected by experts in cognitive and educational science. These experts have a rich set of authoring tools at their disposal for creating instructional software without expertise in software engineering. Such tools are widely used for various topics in academia and in schools, in business and in military. But there are no such authoring tools for stepwise problem solving in mathematics an in applications of mathematics to engineering problems. So the essential goal is formulated as follows:

*Analyse the present dialog of ISAC and re-engineer it such that non-programmers can start experimenting with dialog design.*

This goal has been tackled in a joint effort of two Baccalaureate Theses, this one and [Kie12]. The latter thesis concerns one half of the solution, a rule-based system for future design of dialogs, this thesis describes the other half, logging of high-level steps for approaching learning histories.

In order to document the challenges we met in this thesis we list the original task description as given at the beginning of the work:

1. Replace Java code by a rule-based system (RBS):
   • Identify relevant RBS features for the dialog
   • Select the most appropriate RBS
   • Re-implement the dialog with this RBS
2. Replace the Next and Auto buttons by
   • learner inputs / replaces a formula
   • system offers a list of theorems for application
   • system offers a partial theorem for application
   • system offers a partial formula for fill in
   • ...

Make interactions manageable for cognitive scientists!

3. Log high-level interactions of learners:
   • Log in a database
   • One record per learner input and system response
   • DB structure for retrieving individual performance

Create a history for analysis by cognitive scientists!

The thesis will subsequently report, which of these points have been accomplished, and which have been not (!) accomplished for good reasons also reported.

The structure of this thesis is as follows: the remaining sections in the introductory chapter concern general preparations. §1.2 reviews the history of dialog architecture in academia in general and in ZSAC in particular. §1.3 clarifies the advantages of ZSAC’s mathematics engine for dialog design. §1.4 discusses the general decision for what kind of tools to be integrated into ZSAC; these two sections are in cooperation with [Kie12]. And §1.5 reports the findings from analysis of the existing UserLogger; this is individual work.

Chapter 2 documents the joint work on analysis and design for re-engineering ZSAC such that continuation might benefit from the general views. §2.1 describes the status-quo in ZSAC before starting re-engineering; §2.1.1 presents the classes involved in the dialog and §2.1.2 respective interaction diagrams. Design for re-engineering is discussed in §2.2 and §2.3 already describes the integration of the rule-engine. The last section is individual work, the former sections are joint work with [Kie12].

Chapter 3 documents the practical work in this thesis. Practical work comprises the major part within the thesis as can be seen in the appendix; specific sections in this chapter also explain the reasons why certain points of the original task statement could not be accomplished. The documentation also tries to serve continuation of the work and might benefit with respect to implementation details. The details are given in §3.1 for installation, in §3.2 for analysis, §3.3 for design, §3.4 for integrating the rule-engine. The last section §3.5 presents the use cases underlying the test-driven approach used in the ZSAC project.

Chapter 4 provides some elements of future guidelines for dialog authors using the rule-engine and the final chapter 5 tries a summary and comments on future work.

1.2 State-of-the-art in dialog architecture

This thesis undertakes a major revision of ZSAC’s dialogs. So it appears appropriate to revise the state-of-the-art of how dialogs are modeled in software — even if it should turn out, that ZSAC’s design is still up-to-date.

1.2.1 History and current state

At the time of ZSAC’s initial architectural design in 2003 [Kre05] the decisions involved fundamental discussions on the architecture, because two completely different and incompatible models were available.
The Seeheim Model  The Seeheim Model [Pfa85] splits the entire system into three components as follows:

The Presentation Layer  is responsible for translation of physical representations, such as images, sounds, key-presses or mouse events into the logical concepts of the system and vice versa. Typical tasks of the Presentation Layer include rendering data on the display and parsing user input.

The Dialog Controller  defines the structure of the interaction between user and system. Typical tasks of the Dialog Controller include accepting events the user triggered on the Presentation Layer, routing events to appropriate destinations and making decisions whether and how to notify the user of changes in the state of the system. In other words, the Dialog Controller defines (and enforces the use of) a language for the interaction between user and application.

The Application Interface  is an abstraction of the application’s data and procedures from the user interface’s point of view. It maps objects and operations on the user interface to actual data objects and code in the application, thus representing the application’s functionality in a concise and consistent way.

![Figure 1.1: Interaction in the Seeheim Architecture](image)

Note that in Figure 1.1 on p.5, the messages are named "notify" and "request" from the user’s point of view. From the Dialog Controller's point of view, the messages are distinguished by their direction in or out of the Dialog Controller. Even more so, the Application and the Presentation Layer (representing the user) do not differ in a structural way. Both are merely objects generating events which might be of interest to other objects and have to be handled according to the Dialog Controller’s state and logic. It is the semantic in the Dialog Controller’s logic that makes a difference between user and application, if any.

The MVC Architecture  was the second model of choice. As opposed to the Seeheim Model, which structures the system as a whole, the MVC architecture [BMR+96, SG96, Fow02] is grouped around single data objects as follows:

The Model  is any data object in the application requiring user interaction. In a mathematics assistant like Zimac there are various data objects: the calculation, the specification of a problem, the collection of problems, the collection of methods, the collection of theories, which in turn contain definitions, theorems, etc.

The View  is an object providing a visual representation of the respective Model, thus enabling the Model to output its data.

The Controller  is an object accepting user input and notifying the Model or the View accordingly, thus providing the user with a means of controlling the Model.
Figure 1.2 on p.6 shows interaction in the MVC Architecture. It may be noted, that in a complex system, the link between application and user can contain several Model-View-Controller-triples, each grouped around a specific data item.

In the meanwhile, since 2003, the discussion about dialog architectures has continued. Interestingly, the 'old Seeheim Model' came up again under a new name and now integrates better with well-established concepts:

**Three Tier Client/Server Architecture (TTA):** This architecture\(^4\) is described as a design pattern in [Fow02] and has found wide-spread acceptance [Eck95, Ram00]. The tiers are the following:

**The Presentation Tier:** This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing, and shopping cart contents. It communicates with other tiers by outputting results to the browser/client tier and all other tiers in the network.

**The Application Tier** (business logic, logic tier, data access tier, or middle tier) The logic tier is pulled out from the presentation tier and, as its own layer, it controls an applications functionality by performing detailed processing.

**The Data Tier:** This tier consists of database servers. Here information is stored and retrieved. This tier keeps data neutral and independent from application servers or business logic. Giving data its own tier also improves scalability and performance.

As in the Seeheim Model, presentation and application are separated, while Seeheim didn’t relate to client/server architecture and didn’t say anything about the distribution of components over different machines. TTA clarifies the latter, and interestingly, *TSAC*’s architecture roughly follows also TTA’s distribution over different machines, although TTA was not available for decisions [Kre05]. Now we are ready to discuss the consequences of the general picture for re-engineering *TSAC*’s dialogs.

### 1.2.2 Confirmation of *TSAC*’s original design

The original architectural design for *TSAC* [Kre05] followed the Seeheim Model as shown in Figure 1.3 on p.7:

**Math Engine or Kernel:** In terms of the Seeheim Model, this is the Application. This component is already implemented in SML and is intended to run on a centralized

\(^4\)http://en.wikipedia.org/wiki/Multitier_architecture
dedicated server. All mathematical knowledge resides in this component, all calculations are done here. The SML system communicates via the standard input and output text streams.

**Dialog Guide and User Model:** In terms of the Seeheim Model, this is the Dialog Controller. This component is being implemented in Java. All user interaction is controlled by this component, and this is the only component aware of the individual user.

**Worksheet:** In terms of the Seeheim Model, this is the Presentation Layer. This component is being implemented in Java, with the additional goal of running in standard environments encountered on a consumer PC installations, as this component is intended to run locally on the user’s machine. The Worksheet is the only component aware of visual aspects of data, such as formatting, and the only component with direct user-interaction.

![Basic ISAC architecture for calculations](image)

Figure 1.3: Basic ISAC architecture for calculations

In subsequent years extensions of ISAC slipped into the MVC Architecture, mainly due to the necessity to keep development tasks within narrow boundaries: The extensions added windows for inspection and manipulation of problems, methods and theories to the window of examples. All these windows were associated with specific dialogs. However, all these dialogs were abstracted to a 'BrowserDialog' — a good starting point for re-engineering envisaged in this thesis, which aims at a 'Dialog Guide' handling all interactions of a learner. §2.2 discusses the detailed design which returns to a clean three-layered architecture.

### 1.3 ISAC’s high-level steps in doing mathematics

Characteristic features of TP-based educational systems have been described in §1.1 which carry over to ISAC’s prototype. The features’ impact on dialogs have also mentioned there.

Experience with ISAC’s dialogs [Neu06, Neu07, NR08] indicate additional features which are crucial for dialog design, which have been extensively discussed during the work on this thesis. But these have not yet been communicated publicly; here is a first trial to document these features.

Interactions with ISAC can abstracted to “high-level steps” in a straight forward manner, where “high-level” might be characterised as (1) “powerful” and (2) “substantial”:

1. An interaction is **powerful** for the learner, iff

   (a) it constructs or deconstructs a step in a calculation. Such an interaction which always is embedded into a logical context and thus involves more or less comprehension of mathematical concepts. An such an interaction is always checked by the system for logical consistence.
(b) it provides an item from the mathematics knowledge. The request addresses a comprehensive knowledge item, for instance a specification, or a respective pre/post-condition, a theorem or a proof, a whole method which might alternatively solve the problem at hand, etc. The returned item probably is selected due to some context or decorated with details from the context, both done mechanically by the system.

2. An interaction is **substantial** for analysis by the dialog designer: there is the assumption that such an analysis allows to relate “powerful” interactions to more or less comprehensive competences. Evaluation of competences is done mechanically: each of the “powerful” interactions includes feedback of the system, automatically generated by TP-based components.

High-level steps in that sense are evidently different from other interactions collected mechanically so far [SL06]. Experiences from the first dialog authors of ISAC shall check the above intuitions, put them in more concrete terms, related them to cognitive science and education science, and validate them.

The architectural considerations in §2 will build on the above assumptions looking forward to positive confirmation in the future.

### 1.4 The most urgent tools for dialog authoring in ISAC?

So far some evidence has been collected indicating considerable challenges in dialog design, if the addressed power of TP-based systems are to be exploited for the benefit of learners. This thesis assumes that these challenges are comprehensive and promising such that it is worth to specifically support meeting the challenges by a “development environment for dialog authors (Dialog-IDE)”.

**User requirements:** As a first approach to such an environment we try to specify the user requirements for dialog authors as follows:

**UR 1.4.1 The dialog author is an expert in pedagogy.**

Expertise in pedagogy can be given from Cognitive Science, Science of Mathematics Education, Science Education in general, Instructional Design, etc.

**UR 1.4.2 The dialog author is not an expert in programming.**

So a dialog author does not want to bother with loops and invariants, with nested if-then-else, with inheritance and other intricacies of programming languages.

**UR 1.4.3 Designing dialogs relies on trial and error.**

As long as there are no experiences and no models to learn from, dialog authors will proceed mainly by trial and error.

**UR 1.4.4 The effect of changes between different trials must be traceable.**

Testing in an interactive system causes fugitive sequences of interactions, a permanent trace of sequences relieves from (error-prone) hand-written traces.

**UR 1.4.5 Grouping interactions must be straightforward.**

Comprehensive dialogs adapt to learners’ levels, each level might be associated with a specific group of interactions. Different dialog modes (exploration, exercise, assessment, etc) also call for grouping of interactions.
Software requirements: As a second step the question arises: What are the tools, which put forward ISAC from the present state to a Dialog-IDE meeting the above requirements as soon as possible? — within the limited resources of two Baccalaureate Theses, this one and [Kie12]. Approaching the latter question we try to specify software requirements for dialog authors as follows:

SR 1.4.1 The tools are usable for non-programmers.

SR 1.4.2 The tools’ features meet the user requirements.

SR 1.4.3 The tools allow stepwise system specification.

Stepwise system specification means: At the state being there are so many uncertainties about a Dialog-IDE that too detailed specification runs high risk to require substantial revision soon. This would lead also to substantial re-engineering the code. A better choice is to decide for general tools which are open for more detailed specification and respective specialization of the software components in the future.

Decision for tools: Given the above user requirements and software requirements the decision was for only two tools:

1. A rule-engine of an expert system. Expert systems have been the tools to implement expert knowledge without involving expertise in programming [Jac98, Dar00, GR05]. The choice meets UR 1.4.2..UR 1.4.5 and also SR 1.4.1..SR 1.4.3.

2. An SQL database is the most standard tool to store and retrieve data; a database enforces clean data design and particularly meets UR 1.4.4 and all software requirements SR 1.4.1..SR 1.4.3.

SQL databases are standardized and freely available to an extent, which doesn’t require further discussion. The situation with expert systems is different, thus a discussion follows below.

1.5 Review of ISAC’s ’old’ User Logger

At the beginning of this thesis, ISAC already had a ’UserLogger’. This previous version had served in several field tests at schools [Neu06, Neu07, NR08] and had proved to be useful for simple analysis of learners’ performance.

This ’old’ UserLogger, however, exhibited severe violations of encapsulation and of ’information hiding’ principles. These violations, so the experience during and after development, were enforced by the ’old’ design decision for the UserLogger:

Make the UserLogger a comprehensive component, which relieves dialog authors by

• working with minimal information
• and requesting missing information from the dialogs.

At the time of this design decision rule-based systems were not under consideration. Now, with respect to rule-based systems, ISAC’s whole dialog architecture is under re-consideration. For this purpose the requirement “relieve dialog authors” has been strengthened already in §1.1.

And the design decision will now be changed to the opposite:

Make the ’new’ UserLogger as simple as possible and rely on a clear dialog architecture.
The ‘new’ UserLogger shall have only two public methods:

1. construct the record with the learners input together with administrative data identifying session, calculation, etc.

2. complete the record with the feed-back from the mathematics-engine and write it to the data-base.

This kind of simplicity relies on a clear dialog architecture, because the programmer needs to find the right location in the code, where the two methods suffice. We shall see in §2.2 that this location is found straightforward in rules.
Chapter 2

A “User Logger” for \(\text{ISAC}\)

\(\text{ISAC}\) is well documented \([iT02a, iT02e, iT02f, iT02d, iT02c, iT02b]\) by the students' theses working in the development\(^1\). In spite of these documents, the analysis with respect to dialogs was laborious to an extent not expected.

2.1 Review of \(\text{ISAC}\)’s existing dialog architecture

2.1.1 Classdiagram

The classdiagram in Figure 2.1.1 on p.11 represents a still simplified overview of important classes and methods. Only these objects are displayed which are concerned with the methods used in the interaction diagrams in §2.1.2 below.

\[\text{Figure 2.1: Overview in form of a classdiagram}\]

The classes and interfaces are related to the three elements identified in the above §1.2.1 as follows:

1. Presentation layer
   - WindowApplication: the container of all GUI elements

\(^1\)http://www.ist.tugraz.at/isac/index.php/Research
• Worksheet: the window for interactive calculation

2. Dialog layer

• WorksheetDialog: connects interaction on a single Worksheet with the math-engine, the application layer
• WSDialogManager: the component constructing and deconstructing Worksheet-Dialogs
• IWSDialogManager: the interface within this layer
• Session: the component managing the multi-user system

3. Application layer

• CalcTree

The interfaces abstract the interaction between the three layers as follows:

• IToGui: dialog \(\rightarrow\) presentation
• IToCalc: dialog \(\rightarrow\) application
• IWorksheetDialog:
  • IToDialog: application \(\rightarrow\) dialog and dialog \(\rightarrow\) presentation

### 2.1.2 Interaction diagrams

**SCOPE of interaction diagrams**

The following interaction diagrams specially treat the interaction between GUI, WorksheetDialog and math-engine; Fig.2.1.2 on p.12 shows an abstract view.

![Figure 2.2: SCOPE of interaction diagrams](image)

Mouse-event on a formula: This diagram shows the procedure of clicking on a position in the calculation. The following diagram shows the initial phase of all the subsequent interactions in this section, the first click on a position (which then is followed by different actions).

ad 2 The mouse-event in the Worksheet sends a notification to the WorksheetDialog with a UI:SOLVE:CHANGE:CONTEXT:FORMULA action.

ad 3 Notify the active context presenter that something happened. “Context” here concerns knowledge if shown in the browsers: the position selected by the mouse is related to specific knowledge items, to a specific problem, a specific method, a specific rule (applied at this position).
Relevant for the interaction discussed below is the fact, that Pt. ad 3 also determines the position for subsequent actions, append/replace a formula etc.

After the sequence of notifyUserAction the mouse-pressed method checks two different cases:

- Starts the procedure to show a context menu (right click)
  
  Show tactic context Fig.2.1.2 on p. 16
  
  if (e.getButton() == MouseEvent.BUTTON3 && selected_node_.getPosition() != null)
  showContextMenu(e);

- Editing a formula (double click)
  
  Append a formula 2.1.2 on p. 13
  Replace a formula 2.1.2 on p. 15
  
  if (e.getButton() == MouseEvent.BUTTON1 &&
      e.getClickCount() == 2)
  editingStarted(e);

![Interaction diagram for first click on a formula](image)

**Append a formula** is the most common interaction done on the worksheet. The interaction follows, as mentioned above, a double click on a position making it the “active formula” (see list of terms used in the 2S4C-project) !.

ad 1 This method is called after insert a new formula and press <Enter>

ad 2 Creates a new CalcFormula object and set the text from the input line for use in Pt.6

ad 3 The Worksheet notify the user action in the WorksheetDialog. UI_SOLVE_APPEND_USER_FORMULA → making the new formula the currently active formula)

ad 4 The WorksheetDialog informs with doUIAction() the listeners which implements the interface IToGUI.

ad 5 The WorksheetRMI which implements the IToGUI interface, calls the Worksheet-method doUIAction with an IUIAction. This action holds the user action information. In this case action UI_DO_APPEND_FORMULA context=UICONTEXT_OONEELEMENT

ad 6 During the solving phase, notify that editing the currently active formula is finished. This implies a request for updating the CalcTree.
ad 7 Checks if the formula is correct and calls informListeners with the calc changed event.

```java
ce = math_engine_.appendFormula(this.id_, f);
if (!(ce == null)) {
    if (ce instanceof CChanged) {
        CChanged cc = (CChanged) ce;
        hot_spot_ = new CalcIterator(this, cc.getLastGenerated());
        transactionID = generateTransactionID();
        CalcChanged calcc = new CalcChanged(transactionID, true, // completed
            new CalcIterator(this, cc.getLastUnchanged()),
            new CalcIterator(this, cc.getLastDeleted()),
            new CalcIterator(this, cc.getLastGenerated()));
        informListeners(calcc);
    }
}
```

ad 8-9 The interface IToCalc informs every listening WorksheetDialog (implements IToDialog) through calcResponse() 

```java
while (listn_it.hasNext()) {
    dg = (IToDialog) listn_it.next();
    dg.calcResponse(calcc);
}
```

ad 10 There is a remarkable design decision: unlike doUIAction, which goes via IToGui, calcChanged goes directly to the worksheet (and not via IToGui).
ad 11 The `calcChanged` method calls the `CalculationModel` which updates the tree from a `calcChangedEvent`.

```java
((CalculationModel) model).calcChanged(event);
```

**Replace a formula** already existing in the calculation is a variant of “input a formula”. Fig. 2.1.2 on p.15 shows this interaction:

![Diagram](image)

**Figure 2.5: Interaction diagram for “replace a formula”**

ad 1 The cursor is set on the formula to be edited. Via the interface `ICalcIterator` the position is handled as an iterator

```java
UserActionOnPosition uaop = (UserActionOnPosition) action;
Position activePosition = uaop.getPosition();
ICalcIterator iterator = new CalcIterator(calc_tree_, activePosition);
```

ad 2-3 The position is announced to `IToCalc`, so the next appropriate method on the ISAC-core will use this position.

```java
((IToCalc) calc_tree_).moveActiveFormula(iterator);
```

ad 4 The Worksheet reads the position and stores it in `CalcFormula`.

ad 5 A double-click on the (same!) formula tells the WorksheetDialog that the formula is going to be edited.
ad 6-7 IToGui allows the Worksheet to edit the formula (Presently the respective JPanel is not set read-only).

ad 8-9 <Enter> (editingStopped) finished editing and delivers an action (containing the formula) to the WorksheetDialog, which pass to the ISAC-core via IToCalc (replaceFormula).

ad 10-11 The ISAC-core informs all listeners (observer pattern) by informListeners with a calcChanged event. The Worksheet is among the listeners (calcResponse).

ad 12-13 The CalcChanged event tells the Worksheet, which formulas need to be deleted (because they cannot be derives from the replaced formula) and which formulas need to be inserted (input of a formula might cause several formulas between the lastUnchangedFormula and the lastGeneratedFormula). The formulas returned by getFormulaFromTo are inserted accordingly.

Show tactic context ISAC can display the next possible collections of tactics from the active formula.

Figure 2.6: Interaction diagram for “Show tactic context”
ad 1  After clicking the tactic field (blank field under the formula field), the worksheet calls the function notifyUserAction with UI_SOLVE_CHANGECONTEXT_FORMULA Action in the WorksheetDialog.

ad 2 The WorksheetDialog checks the position of the current action (clicking event). Keep in mind that in this case the position should not NULL!

```java
UserActionOnPosition uaop = (UserActionOnPosition) action;
Position activePosition = uaop.getPosition();
if (activePosition != null) {
    context_formula_ = new CalcIterator(calc_tree_, activePosition);
    notifyContextPresenter();
}
```

ad 3 After checking the position the function notifyContextPresenter is called. This method checks in which dialog (e.g. ExampleDialog, ProblemDialog,...) the click occurs.

ad 4-6 presentContext is implemented in the BrowserDialog (each BrowserDialog is responsible for the communication to the WorksheetDialogs via the ContextProvider-interface) which shows the changes from the context in the browser.

```java
sendContextToBrowser(context);
sendLinkToMiniBrowser(context, context_type_, "");
```

ad 7 At this point the program checks if the right mouse button was clicked

```java
if (e.getButton() == MouseEvent.BUTTON3
&& (selected_node_.getPosition() != null)) {
    showContextMenu(e);
}
```

ad 8 Before the Worksheet calls the notifyUserAction Method, it will be check if the click was in the tactical field or not.

```
Every tree cell has the formula and tactic part. This method checks whether the given click is in the tactic part or the formula part of the cell.
```

ad 9-11 Sends the applicable collection of tactics to the Worksheet.

ad 12 Finally shows the pop up menu with the collection of applicable tactics.

```java
switch (action.getAction())
    case UI_SOLVE_SHOWAPPLICABLE_TACTICS:
        doShowApplicableTactics();
        break;
```

Next Button  Calculates the next step from the active formula and shows it on the worksheet.

ad 1-3 The diagram starts in the WindowApplication because the NEXT-button is not directly on the Worksheet (usually overloaded). In this way the NEXT-Button is for all WorksheetDialogs available and must be determined which WorksheetDialog is active. In order to get the active Dialog its necessary to use the WorksheetDialog-Manager through the Session.

```java
WorksheetDialog wd = this.ws_dialog_manager_.getActiveWSDialog();
if (wd != null) {
    wd.notifyUserAction(action);
}
```
ad 4-5 Now the active WorksheetDialog is established in the session. So notifyUserAction method with UI\_SOLVE\_CALCULATE\_1 can be called. Before the WorksheetDialog can call autoCalculate must first be checked if the active formula is on calcHead. In that case it is needed to complete the calcHead first and fetch the the propose tactic. Unless the formula is not on CalcHead autoCalculate can be called immediately.

ad 6 autoCalculate gets from the mathengine the calculated step in form as a CalcChanged objekt and informs the listeners through the informListeners.

\[
\begin{align*}
\text{cc} &= \text{math_engine}_.\text{autoCalculate(this.id, scope, nSteps);} \\
\text{if} \ (!!(\text{cc} == \text{null})) \{ \\
& \quad \text{transactionID} = \text{generateTransactionID}(); \\
& \quad \text{CalcChanged} \ \text{calcc} = \text{new CalcChanged(transactionID, true,} \\
& \quad \quad \quad \text{new CalcIterator(this, cc.getLastUnchanged())}, \\
& \quad \quad \quad \text{new CalcIterator(this, cc.getLastDeleted())}, \\
& \quad \quad \quad \text{new CalcIterator(this, cc.getLastGenerated()));} \\
& \quad \quad \text{informListeners(calcc);} \\
\}
\end{align*}
\]

ad 7 The method informListeners triggers a calcResponse via the interface IToDialog and send an update event to the registered listeners.

\[
\begin{align*}
& \quad \text{while} \ (\text{listn.it.hasNext()}) \{ \\
& \qquad \text{dg} = (\text{IToDialog}) \text{listn.it.next();} \\
& \qquad \text{dg.calcResponse(calcc);} \\
& \}
\end{align*}
\]

ad 8-9 The active WorksheetDialog triggers in the calcResponse method calcChanged from the Worksheet.

ad 10 Update the nodes of the tree (CalculationModel)

Open Login Screen To start \texttt{TSAC} it is necessary to log in.

ad 1 The WindowApplication is started from the end-user. After starting the program, the WindowApplication is set on visible(false) and the LoginScreen were created.

ad 2 After the initialization of the WindowApplication, the LoginScreen set to visible(true) and the end-user can login with username and password.
Figure 2.8: Interaction diagram for “Login Screen”

Login  This diagram describes the initialization of objects after a correct login of an end-user.

Figure 2.9: Interaction diagram for “Login”

**ad 1-2** see above paragraph “Open Login Screen”

**ad 3-4** After the entry of the user dates, the Login button triggers actionPerfomed which calls fireLoginEvent.

**ad 5** The ILoginListener calls the loginEventOccured method which is implement in the WindowApplication.

**ad 6** If the type of the LoginEvent is LOGIN_TYPE_TRY_TO_LOGIN the system tries to login with the entered username and password otherwise the user pressed “exit” and
the GUI will be closed.

**ad 7-10** To check username and password the instance of UserManager (singleton) is needed and this goes through the ObjectManager.

**ad 11-13** Through the UserManager in the login method a new account will be created and read the properties of the given user. If the authentication is valid, a new user object will be build for the new session.

   Accounts acs = new Accounts(userpath);
   if (acs.authenticationValid(username, password)) {
      User user = new User(username, userpath);
      allLoggedInUsers.put(username, user);
      newSession = SessionManager.getInstance().addSession(user);
   }

**ad 14-18** In the Session the BrowserDialogs (Problem-, Method-, Theory- and ExampleDialog) and WSDialogManager will be created. Fig.2.1.2 on p.21

**ad 21** The BrowserDialog has to know his browser but the browser is made later than the dialog (in the WindowApplication). Therefore the browser has to be registered to the dialog. Furthermore the WindowApplication will be initialized. Finally the LoginScreen will be closed and the WindowApplication set visible(true).

**Open BrowserDialog (Example, Method, Theory or ProblemDialog)** Illustrate the process how a dialog is opened

Figure 2.10: Interaction diagram for “Open BrowserDialog (Example, Method, Theory or ProblemBrowser)”

**ad 1-2** By clicking on a element, an actionID will be returned and triggers the notifySession method with the actionID

**ad 3** In the current session the notifyUserAction activates the chosen BrowserDialog (Fig. 2.11 p. 21) from the user (actionID).

    switch (actionID) {
        case UI_SHOW_EXAMPLE_BROWSER:
            example_dialog_.showBrowserFrame();
            break;
        case UI_SHOW_METHOD_BROWSER:
            method_dialog_.showBrowserFrame();
            break;
    }
ad 4  Makes the selected browser frame visible.

Open an example  Open an example by clicking a hyperlink in the ExampleBrowser

ad 1-2  Get the clicked hyperlink and send it to the ExampleDialog through the notifyUserAction

ad 3  Interprets the given link in the BrowserDialog

ad 4  Get the ExampleDialog from the session.

ad 5-6  With the returned ExampleDialog object, openWorksheetFromExample can be called. This method prepare the example for the Worksheet.

ad 7-10  Fetch the WSDialogManager to open an new WSDialog.

ad 11  WSDialog sends a doUIAction to open the requested example for illustrating in a new worksheet.

ad 12-13  WindowApplication generates the Worksheet with the example and show it on the front end.

Figure 2.11: Interaction diagram for “Dependency of Browsers”
2.2 Design considerations: DialogGuide and UserLogger

This section documents the phase, where joint work with [Kie12] divorced into separated implementation work. However, detailed design particularly exhibited the necessity to do it together. DialogGuide and UserLogger cannot be designed without regarding each other; this will become clear in this section, too. The 'old' UserLogger suffered exactly from the fact, that respective design aimed at independence from the dialog [Kob12]. §1.2.2 confirmed state-of-the-art of ZSAC’s principal architecture. After detailed analysis in §2.1 detailed design decisions can be made. Analysed and new classes are being related as follows:

**SessionManager**: Handles the technicalities resulting from a multi-user system: constructing and deconstructing instances of dialogs for different learners and several calculations per learner.

**Session**: There is 1 session per learner.

**DialogGuide (NEW)**: There is 1 DialogGuide per learner in parallel to a Session; while the latter is concerned with technicalities, the tasks concerning pedagogy are separated and assigned to the DialogGuide. The DialogGuide is the “master mind” of all dialogs, the BrowserDialogs and the WSDialogs.

**BrowserDialog**: This is an abstraction of 4 similar dialogs guiding interaction with respective 'browsers' for ZSAC’s knowledge, theories, problems, methods and examples respectively.

**WSDialog**: For each calculation (displayed on a 'Worksheet') there is a WSDialog; so 1 user (i.e. 1 Session, 1 DialogGuide) can have 0..n WSDialogs.
**UserLogger** (NEW): There is 1 UserLogger per system, serving 0..n DialogGuides. The UserLogger relies on a clear architecture which makes recording simple for dialogs. Further specific design considerations are found in [Kob12].

Recalling §1.2, all these classes belong to the dialog layer. The interfaces to the presentation layer as well as to the application layer is not affected by the extension of the dialog layer by DialogGuide and UserLogger. A quick check shows that all the original user requirements can be met:

**User Guidance** cited from [Kre05]:

**UR 2.2.1** *TSAC* can offer a list of actions known to the system.

**UR 2.2.2** *TSAC* can offer a list of actions applicable to the current situation.

**UR 2.2.3** *TSAC* can propose the next action to be taken.

**UR 2.2.4** *TSAC* can do one or more steps automatically.

**UR 2.2.5** The user gets immediate feedback on data entered into the Model

**UR 2.2.6** *TSAC* can match a problem to a Model.

**UR 2.2.7** *TSAC* can refine a problem to match a Model more closely.

**UR 2.2.8** On request, *TSAC* provides additional information on parts of the calculation

**UR 2.2.9** The amount of user guidance is configurable.

The amount can be set by the user according to his preferences or by a course designer to match requirements of the course. For exam purposes, the amount of user guidance can be limited.

**User Profiling** cited from [Kre05]:

**UR 2.2.10** *TSAC* records examples done by the user

*TSAC* keeps a per-user record of examples done and the user’s performance in doing the example. The record is independent of the course the user has been logged into when doing the example.

**UR 2.2.11** *TSAC* records items in the Knowledge Base viewed by the user.

This information can be used to base the Dialog Guide’s behavior on information supposedly known to the user.

**UR 2.2.12** *TSAC* records the user’s success and errors.

This extends to application of single Tactics as well as whole examples or courses.

**UR 2.2.13** *TSAC* records the user’s time performance.

In the future, assumptions about the user’s familiarity with certain topics could be derived from these data.

**UR 2.2.14** *TSAC* records the user’s activity.

In this context, activity means the ratio of steps done by the user to the steps the user had done by *TSAC*.
2.3 Integration of the User Logger

In this section will be mentioned the prime points to design the UserLogger. We are also keep in mind that the generated data from Userlogger should as well readable for non-technical Persons. However, there almost only the technical aspects listed for the Design. Thereafter are the points what the UserLogger should fulfill:

- The logging process should not visible for end-user.
- The recorded data from the logging process should store in a database.
- The saved records in the database should also readable for non-technical persons.
- ISAC should also usable without the logging process.
- The access to the UserLogger should only be possible through special classes.
- The Userlogger should as few as possible accessible through getters and setters.
Chapter 3

Implementation and administration of the project

3.1 Installation of the experimental system

As a result of installation of the experimental system, a wiki was born. It contains the knowledge, experience and possible error sources. [NK12]

The following installation hints are only compatible with Linux or Mac as operating system.

In order to develop at ISAC-project, we need the following specific Software:

1. **Java 1.6**: The actual Java Development Kit taken from the oracle homepage.
2. **polyml 4.1.3**: The implementation language form Isabell (and ISAC core)
3. **NeatBeans**: The used IDE for development.
4. **Mercurial**: The Version Control System to organize the source code.

For some development task there also needed:

1. **log4j chainsaw**: Logging tool for distributed systems.
2. **XAMPP for Linux**: Local SQL Database to log the steps from the user.

**Configure and installed the experimental system with following steps:**

**Download ISAC**

The console commands for Linux or Mac are listed in *italic letter*.

1. Set up a new blank folder in your chosen directory.
   
   `mkdir proto3`

2. Clone in this new folder the ISAC repository.
   
   `proto3$ hg clone https://intra.ist.tugraz.at/hg/isac/ repos`

3. Download the ISAC knowledge in html and unpack it in your folder.
   
   `proto3$ tar -xzf kbase.tgz`
4. Set up a new blank folder in your directory and download the ISAC core binary and unpack it.
   
   ```
   proto3$ mkdir ml
   proto3$ cd ml
   proto3/ml$ tar -xzf HOL-Real-isac.tgz
   ```

5. Download and unpack this java library and overwrite it in your cloned repository "proto3/repos/lib" folder, sometimes they are corrupt.
   
   ```
   proto3$ mkdir ml
   ```

**Configure components**

1. Prepare a new folder in repos/src/java and copy the property files in the in this new folder, after copy this six files from the properties-template.linux folder, replace all MYUSERNAME by your user name.
   
   ```
   proto3$ cd repos/src/java/
   proto3/repos/src/java$ mkdir properties
   proto3/repos/src/java$ cp properties-templates.linux/* properties/
   ```

2. Create a .java.policy(invisible) file in your home directory with the following content.

   ```
   grant {
   permission java.net.SocketPermission ":*:1024-65535", "connect,accept";
   permission java.net.SocketPermission ":*:80", "connect";
   permission java.io.FilePermission "<<ALL FILES>>", "read, write, delete, execute", signedBy "ISAC";
   permission java.lang.RuntimePermission "setIO";
   permission java.lang.RuntimePermission "createClassLoader";
   permission java.util.PropertyPermission ":", "read,write";
   permission java.security.AllPermission;
   }
   ```

3. Reconfigure the Mercurial by editing proto3/repos/.hg/hgrc as follows:

   ```
   [paths]
   default = https://intra.ist.tugraz.at/hg/isac/
   [ui]
   username = n.n. <n@n.n>
   merge = meld
   editor = nano
   [extensions]
   hgext.fetch =
   [defaults]
   fetch = -m "merged"
   ```

**Configure NetBeans**

1. Create a new project in NetBeans from a existing sources with "proto3/repos" as project Folder.

   ```
   NetBeans-> File-> New Project-> Java Project with Existing Sources
   ```
2. Add the libraries to the new project form "proto3/repos/lib" folder.
   NetBeans "Project View" - right mouse click on project -> Properties -> Libraries
   - Add JAR/Folder

3. Define run configurations in NetBeans.
   NetBeans "Project View" - right mouse click on project -> Properties -> Run -> New

To adjust the 4 run configurations you need the following settings, paths are based on my system so should replace it with your paths:

**Configuration: BridgeMain**
Main Class: isac.bridge.BridgeMain
Arguments: /home/MYUSERNAME/proto3/repos/src/java/properties/BridgeMain.properties
VM Options: -Djava.rmi.server.hostname=127.0.0.1 -Djava.rmi.server.codebase=
            'file:///home/MYUSERNAME/proto3/repos/build/classes/
            file:///home/MYUSERNAME/proto3/repos/build/test/classes/
            file:///home/MYUSERNAME/proto3/repos/lib/log4j-1.2.11.jar'
            -Djava.security.debug=none

**Configuration: KEStore**
Main Class: isac.keystore.KEStore
Arguments: /home/MYUSERNAME/proto3/repos/src/java/properties/KEStore.properties
VM Options: -Djava.rmi.server.hostname=127.0.0.1 -Djava.rmi.server.codebase=
            'file:///home/MYUSERNAME/proto3/repos/build/classes/
            file:///home/MYUSERNAME/proto3/repos/build/test/classes/
            file:///home/MYUSERNAME/proto3/repos/lib/log4j-1.2.11.jar'
            -Djava.security.debug=none

**Configuration: ObjectManager**
Main Class: isac.session.ObjectManager
Arguments: /home/MYUSERNAME/proto3/repos/src/java/properties/ObjectManager.properties
VM Options: -Djava.rmi.server.hostname=127.0.0.1 -Djava.rmi.server.codebase=
            'file:///home/MYUSERNAME/proto3/repos/build/classes/
            file:///home/MYUSERNAME/proto3/repos/build/test/classes/
            file:///home/MYUSERNAME/proto3/repos/lib/log4j-1.2.11.jar'
            -Djava.security.debug=none

3.2 Dialog architecture: analysis

The Figure 3.2 on page 28 represents a simplified overview of the dialog architecture. Because of the clarity, methods and members have been omitted. Some of the showed classes are singleton that means, it exist exactly one object from the class. The following classes are singleton:

- SessionManager
- UserManager

The UserManager holds all logged-in users from the ISAC system and the SessionManager hold all sessions these would be create if a login was successfully. So we see in both cases is a singleton the right way, because if there more than one object form UserManager or SessionManager there got problems with the holding from sessions or users.

At launch from the ISAC system the ObjectManager will be create first. The ObjectManager creates the Session- and the UserManager object and hold it in parameter list.

If a new end-user logged-in in the system, the object from the UserManger would called
and a new user object would be created. This object becomes the SessionManager who creates a new session for the user and holds it in his parameter list too. During the building of the session the WSDialogManger and all BrowserDialogs (Method-, Example-, Problem and Theory) will be created. The BrowserDialogs turn into visible status, if one of the four buttons on the MainWindow will be clicked - otherwise they are invisible. The WSDialogManager is responsible for handling of one or more WorksheetDialogs (each Worksheet has one WorksheetDialog). In summary it can be said there are only one SessionManager and UserManager in the whole ISAC system, but there is for every single user who are logged-in, one specific session which hold the BrowserDialogs and the WSDialogManager.

### 3.3 Detailed design of the UserLogger

![UserLogger Design](image)

Figure 3.2: UserLogger Design

[Diagram of UserLogger]
The Figure 3.3 on page 28 displays how the new UserLogger is compound with the existing classes. The UserLogger is also a singleton and will create at launch of the Object-Manager. The DialogGuid is the link between the UserLogger and the existing Browser and WorksheetDialogs. The DialogGuid object will create per session and every Dialog(Browser, Worksheet) has this object as parameter in the constructor, so is the accessibility for every Dialog given. As previously mentioned the DialogGuid is the link between the Dialogs and the UserLogger, the LogItem is the recording object he will create at defined places at the Dialogs. If the record from the LogItem object completely it will through the DialogGuid the User-logger transmitted. The function of the UserLogger is to unpack the LogItem and save the collected data into the database.

3.4 Integration of the UserLogger

As previously mentioned the main part of the UserLogger is to unpack the given LogItem and save the stored data into the database. He is also responsible for connection establishment with the given database this is done in the following code snippet:

```java
String db_path = "jdbc:mysql://" + ObjectManagerPaths.LOGGER_DATABASE_PATH + ":" + ObjectManagerPaths.LOGGER_DATABASE_PORT + "/" + ObjectManagerPaths.LOGGER_DATABASE_NAME;
Class.forName("com.mysql.jdbc.Driver");
try {
    con = DriverManager.getConnection(db_path, "root", "");
} catch (SQLException ex) {
    System.err.println("SQLException: " + ex.getMessage());
}
```

As we can see the usage from ObjectManagerPaths, they are the read paths by the Object-Manager at startup. Before the connection with this code will established, it also check the parameter "LOGGER_DATABASE_ENABLED" from ObjectManagerPaths, if this set to false the connection will not be created.

The storing into the database from the received LogItem occurs in the following code snippet:

```java
    ;
pstmt = connection_.prepareStatement(query);
List<Object> db_list = new ArrayList<Object>();
db_list = log_item.generateListforDB();
pstmt.setString(1, String.valueOf(db_list.get(0)));
pstmt.setString(2, String.valueOf(db_list.get(1)));
pstmt.setString(3, String.valueOf(db_list.get(2)));
pstmt.setString(4, String.valueOf(db_list.get(3)));
pstmt.setTimestamp(5, Timestamp.valueOf(String.valueOf(db_list.get(4))));
pstmt.setString(6, (String.valueOf(db_list.get(5))));
pstmt.setString(7, String.valueOf(db_list.get(6)));
pstmt.setString(8, String.valueOf(db_list.get(7)));
pstmt.setString(9, String.valueOf(db_list.get(8)));
pstmt.setString(10, String.valueOf(db_list.get(9)));
```
The LogItem will defined at different Dialogs (Worksheet-, Example-, Method-, Problem or TheoryDialog). The collected data in the LogItem, will be stored through the constructor. If a calculation step in a Dialog concluded, the LogItem will completed with the method completeWithError() or completeWithChanged(). CompleteWithError() will called if a calculation step fails and completeWithChanged() will called if the calculation step delivers a result from the mathengine. After completing, the LogItem will transmit through the DialogGuid to the UserLogger.

3.5 Develop and test the usecases

Test-Driven Development [Bec03]

Test-driven development consists three main parts.

1. First of all you write a test for the actual part what you would change. This test provides the result what you would reach after the change from the code. If you run this test, it will fail because the current code delivers another result or the code didn’t exist.

2. Change the code part until the new test passed. The effort for the change should as little as possible.

3. Refactor the new code, so restructuring the existing body and revise the internal structure, without changing the external behavior. Run the test again if the test passed the new code part is finished.

The sense of test driven development is to get a better testability, furthermore it ensures that for every implemented functionality, a test exists and fulfilled. And the developer is able to see if the change effects on another functionalities.

![Test-Driven-Development Schema](image)

**Figure 3.3: Schema of Test-Driven-Development**

**Usecases**
To figure out if the rules working correct, we used defined usecases (designed by Alan Krempler [Kre05]) in form of testcases. The listed usecases below where in the test-directory (package isac.wsdialog). There are two versions for each test/usecases. One
version is for the original WorksheetDialog (without RLB) and the other for the new one (with RLB). So the very important functionality of ISAC can now being tested.

1. Picking a Tactic from a List of Applicable Tactics
2. Entering a Formula Manually
3. Editing a Formula
4. Replacing a Formula
5. Having ISAC Calculate the Next Formula
6. Having ISAC Calculate until a Final Result is Reached
7. Show the tactic applied to a formula
8. Show a list of tactics applicable to a formula
9. Show the intermediate steps leading to a formula
Chapter 4

Guidelines to UserLogger for dialog authors

This section displays the according on the database from some selected use cases. The logged information in the database are saved in one table, this table is divide in the following Columns:

1. **userlogger_id**: Is the primary key of the table.
2. **username**: The name of the logged user.
3. **session_id**: The ID from the session.
4. **dialog_type**: The dialog type(WORKSHEET, EXAMPLE, THEORY, PROBLEM, METHOD, SESSION, NONE) which actual active.
5. **worksheet_id**: The ID from the actual active worksheet
6. **time**: Time of writing into database.
7. **position**: Position within actual formula.
8. **formula_from**: The formula where the interaction is set on.
9. **step**: Actual selected step from the learner.
10. **step_arg_1**: Filename for start example, or UI_SOLVE_EDIT_ACTIVE_FORMULA_COMPLETE if the calculation complete
11. **step_arg_2**: Path from the selected example in the ExampleBrowser, otherwise empty.
12. **formula_to**: The formula resulting from the interaction.
13. **error**: Set to one if an error occurs otherwise empty.
14. **error_message**: Error message string if an error occurs otherwise empty.
Having *TSAC* Calculate the Next Formula

<table>
<thead>
<tr>
<th>user/session_id</th>
<th>username</th>
<th>session_id</th>
<th>dialog_type</th>
<th>worksheet_id</th>
<th>time</th>
<th>position</th>
<th>formula_from</th>
<th>step</th>
<th>step_arg_1</th>
<th>step_arg_2</th>
<th>formula_to</th>
<th>error</th>
<th>error_message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x 0</td>
<td>NONE</td>
<td></td>
<td></td>
<td></td>
<td>2012-02-23 22:42:47</td>
<td>nvl</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 x 0</td>
<td>EXAMPLE</td>
<td></td>
<td></td>
<td></td>
<td>2012-02-23 22:43:06</td>
<td>nvl</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 x 0</td>
<td>WORKSHEET</td>
<td>0,8</td>
<td></td>
<td></td>
<td>2012-02-23 22:45:13</td>
<td>nvl</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 x 0</td>
<td>WORKSHEET</td>
<td>0,8</td>
<td></td>
<td></td>
<td>2012-02-23 22:45:37</td>
<td>(P1)</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.1: Database record next step

Having *TSAC* Calculate until a Final Result is Reached

<table>
<thead>
<tr>
<th>user/session_id</th>
<th>username</th>
<th>session_id</th>
<th>dialog_type</th>
<th>worksheet_id</th>
<th>time</th>
<th>position</th>
<th>formula_from</th>
<th>step</th>
<th>step_arg_1</th>
<th>step_arg_2</th>
<th>formula_to</th>
<th>error</th>
<th>error_message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x 0</td>
<td>NONE</td>
<td></td>
<td></td>
<td></td>
<td>2012-02-23 22:08:00</td>
<td>nvl</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 x 0</td>
<td>EXAMPLE</td>
<td></td>
<td></td>
<td></td>
<td>2012-02-23 22:08:14</td>
<td>nvl</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 x 0</td>
<td>WORKSHEET</td>
<td>0,8</td>
<td></td>
<td></td>
<td>2012-02-23 22:08:26</td>
<td>nvl</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 x 0</td>
<td>WORKSHEET</td>
<td>0,8</td>
<td></td>
<td></td>
<td>2012-02-23 22:08:39</td>
<td>(P1)</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 x 0</td>
<td>WORKSHEET</td>
<td>0,8</td>
<td></td>
<td></td>
<td>2012-02-23 22:08:53</td>
<td>(P1)</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.2: Database record final result

Picking a Tactic from a List of Applicable Tactics

<table>
<thead>
<tr>
<th>user/session_id</th>
<th>username</th>
<th>session_id</th>
<th>dialog_type</th>
<th>worksheet_id</th>
<th>time</th>
<th>position</th>
<th>formula_from</th>
<th>step</th>
<th>step_arg_1</th>
<th>step_arg_2</th>
<th>formula_to</th>
<th>error</th>
<th>error_message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x 0</td>
<td>NONE</td>
<td></td>
<td></td>
<td></td>
<td>2012-02-23 22:29:36</td>
<td>nvl</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 x 0</td>
<td>EXAMPLE</td>
<td></td>
<td></td>
<td></td>
<td>2012-02-23 22:29:45</td>
<td>nvl</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 x 0</td>
<td>WORKSHEET</td>
<td>0,8</td>
<td></td>
<td></td>
<td>2012-02-23 22:29:55</td>
<td>nvl</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 x 0</td>
<td>WORKSHEET</td>
<td>0,8</td>
<td></td>
<td></td>
<td>2012-02-23 22:30:09</td>
<td>(P1)</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 x 0</td>
<td>WORKSHEET</td>
<td>0,8</td>
<td></td>
<td></td>
<td>2012-02-23 22:30:23</td>
<td>(P1)</td>
<td></td>
<td>LO_SESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.3: Database record picking a tactic
Chapter 5

Summary and conclusion

This thesis tackled a practically relevant re-design and re-implementation in a joint effort with [Kie12]. The re-engineering work reflects a turn of viewpoint to not yet existing task, dialog authoring.

Consequently the work on the thesis had to take into account a couple of uncertainties. These uncertainties are reflected in preliminary requirements for dialog authoring in §1.4.

The uncertainties were tackled by reviewing ZSAC’s dialog design from scratch and by reviewing the present state-of-the-art. The fundamental review confirmed the original decisions for ZSAC’s architecture. The extension of the dialog by a DialogGuide and a UserLogger was straight forward and did not affect ZSAC’s original architecture; in particular, the interfaces to the presentation layer and to the application layer did not require any changes.

A specific point of uncertainties were the estimates for the efforts required in the planned tasks. Even the task list cited in §1.1 changed during the project. Below there is an accountancy of the thesis’ goals accomplished and not accomplished, in a form which could be clarified at the end of the work:

Work accomplished by this Baccalaureate Thesis is:

- implementation of a rule-based dialog control: The central Java code guiding the interaction on a calculation has been replaced by code calling a rule-engine.
- 20 regression tests for most relevant interactions: Before the beginning of this thesis ZSAC had regression test for the functionality of the mathematics engine; a relevant selection from these tests has been doubled for tests on the dialog.
- logging a history of interactions in a database: Interaction is logged as high-level steps as defined in §1.3. The records comprise elements sufficient for starting experiments with dialog authoring without forestalling expertise in pedagogy.

Plans not accomplished by this Baccalaureate Thesis are:

- system’s intelligent offers for various interactions: The thesis implemented one set of rules which exactly resembles the behavior as found before starting the thesis. Implementation of further sets of rules and code selecting respective sets is up to the future.
- thus Next and Auto still required: Next is the only possibility to request a next step from ZSAC. Removing these buttons requires the systems ability to decide mechanically for the user-guidance – respective sets of rules are still missing.
- full coverage of regression tests: Not all interactions now handled by the rule-engine integrated into the dialog are covered by tests.
The overall conclusion is positive: in spite of interesting plans not accomplished, the major goals of the thesis have been achieved: The thesis provides prerequisites for non-programmers to exploit **rule-based dialog control** for

- extending the variety of interactions: the existing rule-set is an appropriate model for implementing further rule-sets. Cooperation with programmers is required until mechanisms for switching between rule-sets are implemented.

- grouping interactions into dialog patterns: interactions are controlled by rules, grouping of the rules models dialog patterns/modes.

- developing adaptive user guidance in step-wise solving mathematics problems: Switching between rule-sets adapts to varying learning situations and different levels of learner’s competences.

The thesis also provides prerequisites to exploit the **comprehensible history of interactions** for

- dynamically adapting rule-sets: If a learner gets much negative feedback from the mathematics-engine, then the dialog might adapt by increasing user-guidance; on the other hand the dialog might avoid mental under-load by reducing user-guidance — this now can easily achieved by switching rule-sets.

- abstracting sessions into a user-model: The abstraction of the dynamic adaption during a session seems not be difficult if approached with pedagogical expertise — the technical prerequisites are all implemented.

- evaluating the learning process (assessment): This is probably the simplest task for an expert in pedagogy. The **ISC-**project, however, is much more interested in learning than in grading.
# Appendix A: Protocol of activities

## Administrative work

<table>
<thead>
<tr>
<th>Date</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04.07.2011</td>
<td>10</td>
<td>Supervision: information about the ZSAc-project, possible tasks</td>
</tr>
<tr>
<td>10.07.2011</td>
<td></td>
<td>ZSAc-documentation, LaTeX</td>
</tr>
<tr>
<td>11.07.2011</td>
<td>5</td>
<td>Supervision: LaTeX, proposal to Christian Gütl</td>
</tr>
<tr>
<td>17.07.2011</td>
<td></td>
<td>Working with netbeans</td>
</tr>
<tr>
<td>18.07.2011</td>
<td>20</td>
<td>LaTeX</td>
</tr>
<tr>
<td>24.07.2011</td>
<td></td>
<td>netbeans, framework</td>
</tr>
</tbody>
</table>

## MS Installation

<table>
<thead>
<tr>
<th>Date</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>07.07.2011</td>
<td>5</td>
<td>Supervision: installation of the ZSAc-system</td>
</tr>
<tr>
<td>11.08.2011</td>
<td>3</td>
<td>Supervision: integration of log4j</td>
</tr>
</tbody>
</table>
**MS Dialog Architecture**

<table>
<thead>
<tr>
<th>Date</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.07.2011</td>
<td></td>
<td>classdiagram</td>
</tr>
<tr>
<td>17.07.2011</td>
<td>8</td>
<td>ZSAC/documentation, LaTeX</td>
</tr>
<tr>
<td>18.07.2011</td>
<td></td>
<td>Supervision: review of UserLogger</td>
</tr>
<tr>
<td>24.07.2011</td>
<td>10</td>
<td>Analysis design</td>
</tr>
<tr>
<td>25.07.2011</td>
<td></td>
<td>Supervision: overview isac architecture, class diagram</td>
</tr>
<tr>
<td>31.07.2011</td>
<td>11</td>
<td>Analysis design</td>
</tr>
<tr>
<td>08.08.2011</td>
<td>14</td>
<td>Supervision: overview isac architecture, class diagram</td>
</tr>
<tr>
<td>14.08.2011</td>
<td></td>
<td>Analysis design</td>
</tr>
<tr>
<td>15.08.2011</td>
<td>13</td>
<td>Supervision: questions to dialog architecture, observer pattern</td>
</tr>
<tr>
<td>29.08.2011</td>
<td>20</td>
<td>Debugging with Log4j</td>
</tr>
<tr>
<td>04.09.2011</td>
<td></td>
<td>Supervision: questions, interface java – SML</td>
</tr>
<tr>
<td>05.09.2011</td>
<td>19</td>
<td>Analysis Design with Log4j debug output</td>
</tr>
<tr>
<td>11.09.2011</td>
<td></td>
<td>Supervision: interaction diagram: objects,methods for solve phase</td>
</tr>
<tr>
<td>12.09.2011</td>
<td>16</td>
<td>Designed interaction diagram</td>
</tr>
<tr>
<td>18.09.2011</td>
<td></td>
<td>Supervision: interaction diagram: calculate one step</td>
</tr>
<tr>
<td>19.09.2011</td>
<td>18</td>
<td>Designed interaction diagram</td>
</tr>
<tr>
<td>25.09.2011</td>
<td></td>
<td>Supervision: interaction diagram: append and replace formula</td>
</tr>
<tr>
<td>26.09.2011</td>
<td>20</td>
<td>CalcMessage, Calchead locking, Logger</td>
</tr>
<tr>
<td>02.10.2011</td>
<td></td>
<td>Redesign interaction diagram for “replace and append a formula”</td>
</tr>
<tr>
<td>03.10.2011</td>
<td>15</td>
<td>Supervision: discuss about the correctness of the current interaction diagrams</td>
</tr>
</tbody>
</table>

**MS Select Rule-Based System (RBS)**

<table>
<thead>
<tr>
<th>Date</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>08.08.2011</td>
<td>12</td>
<td>Research of Rule-Based Systems</td>
</tr>
<tr>
<td>14.08.2011</td>
<td></td>
<td>Testing JESS, JRuleEngine</td>
</tr>
<tr>
<td>15.08.2011</td>
<td>13</td>
<td>Supervision: demo of RBSs, assessment criteria of RBSs</td>
</tr>
<tr>
<td>21.08.2011</td>
<td></td>
<td>Testing Drools</td>
</tr>
</tbody>
</table>
### MS Detailed Design of New Dialog

<table>
<thead>
<tr>
<th>Date</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>02.12.2011</td>
<td>10</td>
<td>Supervision: discuss about (dis)advantage rough design</td>
</tr>
<tr>
<td>03.12.2011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09.12.2011</td>
<td>9</td>
<td>Supervision: insert new class drawing ideas on board</td>
</tr>
<tr>
<td>10.12.2011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.12.2011</td>
<td>11</td>
<td>Supervision: beginning implementation of design</td>
</tr>
<tr>
<td>17.12.2011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.01.2012</td>
<td>15</td>
<td>Integrate new Dialog in the environment</td>
</tr>
<tr>
<td>22.01.2012</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### MS Integration of the UserLogger

<table>
<thead>
<tr>
<th>Date</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>06.02.2012</td>
<td>30</td>
<td>Integrate UserLogger with LogItem in WorksheetDialog</td>
</tr>
<tr>
<td>12.02.2012</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### MS Develop and Test the Use cases

<table>
<thead>
<tr>
<th>Date</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.09.2011</td>
<td>10</td>
<td>eliminate ObjectManager</td>
</tr>
<tr>
<td>02.10.2011</td>
<td></td>
<td>reactivate ObjectManager</td>
</tr>
<tr>
<td>13.02.2012</td>
<td>20</td>
<td>Test the develop Use cases with DB</td>
</tr>
<tr>
<td>19.02.2012</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Write Bachelors thesis

<table>
<thead>
<tr>
<th>Date</th>
<th>Hours</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04.07.2011</td>
<td>38</td>
<td>Write Bachelor Thesis</td>
</tr>
<tr>
<td>22.04.2012</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bibliography


