Towards "NextStep Userguidance" in a Mechanized Math Assistant

Bakkalaureate Thesis
at IICM,
Graz University of Technology

Markus Kienleitner
mkienleitner@student.TUGraz.at

April 24, 2012

Contents

List of Figures

Abstract

This Baccalaureate Thesis is joint work with [Kob12]. Cooperation was enforced by the comprehensiveness of the task to reorganize the dialog of ISAC which is under development at TUG. The goal of the reorganization is to allow non-programmers to adapt learners' interaction with 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 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 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 first tool, the rule-engine. After a careful analysis of the existing dialog architecture and of the implementation in ISAC the rule-engine has replaced the key-functionality of ISAC's dialog. Both steps, analysis and design as well as implementation are described in detail. The key-point in the dialog is now simplified such that an adaptive dialog can be developed at this point now looking like a 'tabula rasa'. So at this point complex code for rule-based adaptive dialogs can emerge with sufficient efficiency.

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 2002 [1]. 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

[1] `http://www.ist.tugraz.at/projects/isac/` and `http://www.ist.tugraz.at/isac/`

[2] THedu `http://www.uc.pt/en/congressos/thedu`

[3] eduTPS `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 [Kob12]. The latter thesis concerns one half of the solution, logging of high-level steps for approaching learning histories, this thesis describes the other half, a rule-based system for future design of dialogs.

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 ISAC in particular. §1.3 clarifies the advantages of ISAC's mathematics engine for dialog design. §1.4 discusses the general decision for what kind of tools to be integrated into ISAC; these two sections are in cooperation with [Kob12]. And §1.5 reports the findings during the extensive search for an appropriate rule-engine and the final decision for a certain product; this is individual work.

Chapter 2 documents the joint work on analysis and design for re-engineering ISAC such that continuation might benefit from the general views. §2.1 describes the status-quo in ISAC 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 [Kob12].

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, §3.4 for re-engineering the dialog and §3.5 for integrating the rule-engine. The last section §3.6 presents the use cases underlying the test-driven approach used in the ISAC 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 ISAC'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 ISAC's design is still up-to-date.

1.2.1 History and current state

At the time of ISAC'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

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 ISAC 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.

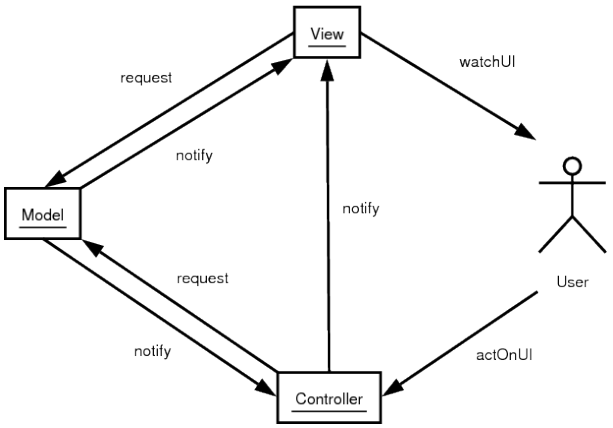

Figure 1.2: Interaction in the MVC Architecture

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, ISAC'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 ISAC's dialogs.

[4] http://en.wikipedia.org/wiki/Multitier_architecture

1.2.2 Confirmation of ISAC's original design

The original architectural design for ISAC [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 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.

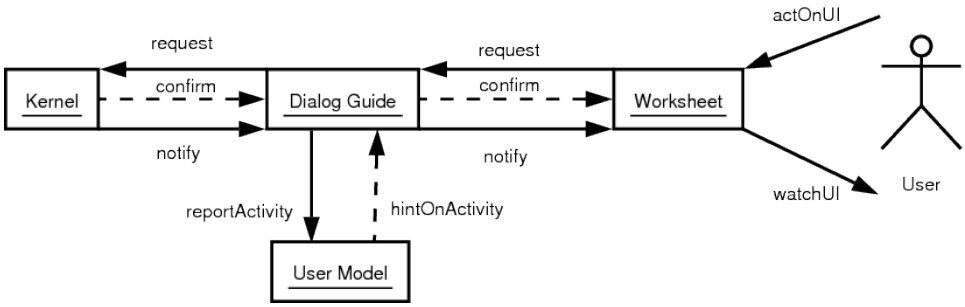

Figure 1.3: Basic ISAC architecture for calculations

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 haven 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 characterized 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 feed-back 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 $\mathcal{ISAC}$ 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 $\mathcal{ISAC}$?

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 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 straight forward.

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 [Kob12]. 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 Selection of a rule-based System

After a research about rule-based systems (RBS) in Java the follow table was created to figure out which system is the best for ISAC. The environment for testing was Linux (Ubuntu x64) Eclipse and NetBeans.

No.	Criterion	JRuleEngine	JESS	Drools Expert
1	**Support**			
1.1	mailing lists	forum only	in sourceforge	yes
1.2	responses	moderate	weak	good
1.3	documentation	moderate	moderate	large and good documentation
1.4	examples	good	good	very good
1.5	tutorial	short	good	good
2	**System features**			
2.1	tracing	print	print	print
2.2	statefulness	yes	yes	yes
2.3	rule syntax	xml	xml,lisp	mvel or java
3	**Source availability**			
3.1	Freeware	yes	only for academic	yes
3.2	Open source	yes	no	yes
3.3	last stable release	2008-04-16	2008-11-05	2011-06-23

looked up 2011-08-30

Very important for this project is, that the RBS is open source. So the first candidate JESS is eliminated because they have only academical licenses. After an intensive test period of JRuleEngine and Drools Expert the winner is determined.

It is Drools Expert; this RBS has a very good and large documentation with many examples. Another test criterion was the support/community, the response of questions. Drools Expert has a big community and a forum coupled with a mailing list. The response was very good and technical precise. The release cycles are about 6 months (compare last stable release of JRuleEngine).

One disadvantage of Drools Expert is, that only Eclipse is supported and $\mathcal{ISAC}$ is currently being developed on netbeans. There are some features that does not work for netbeans (e.g. graphical rule editor, decision tables), but this are only bonus features.

Chapter 2

A "Dialog Guide" for ISAC

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 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.

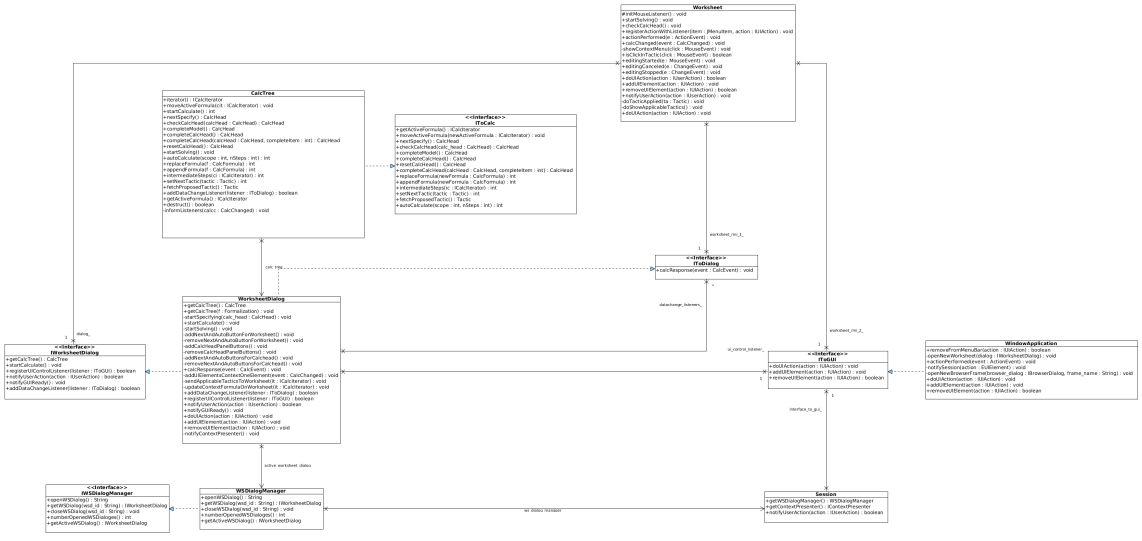

Figure 2.1: Overview in form of a classdiagram

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 $\longrightarrow$ presentation
- IToCalc: dialog $\longrightarrow$ application
- IWorksheetDialog:
- IToDialog: application $\longrightarrow$ dialog *and* dialog $\longrightarrow$ 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.

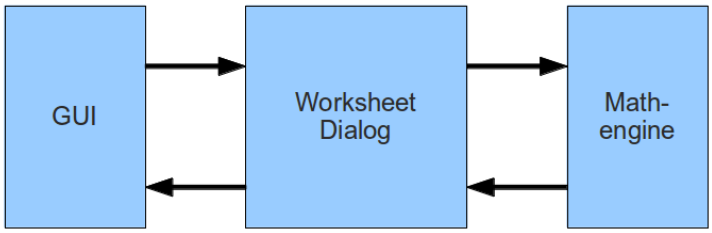

Figure 2.2: SCOPE of interaction diagrams

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);
```

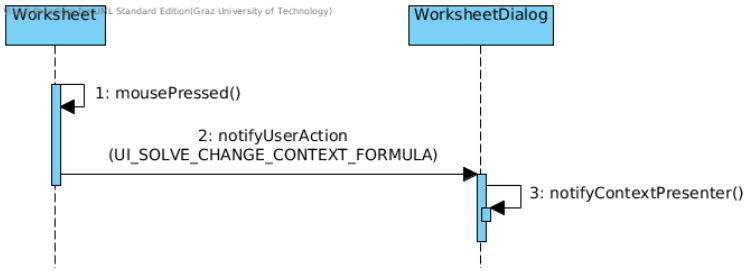

Figure 2.3: Interaction diagram for first click on a formula

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 *TSAC*-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=UI_CONTEXT_ONEELEMENT

ad 6 During the solving phase, notify that editing the currently active formula is finished. This implies a request for updating the CalcTree.

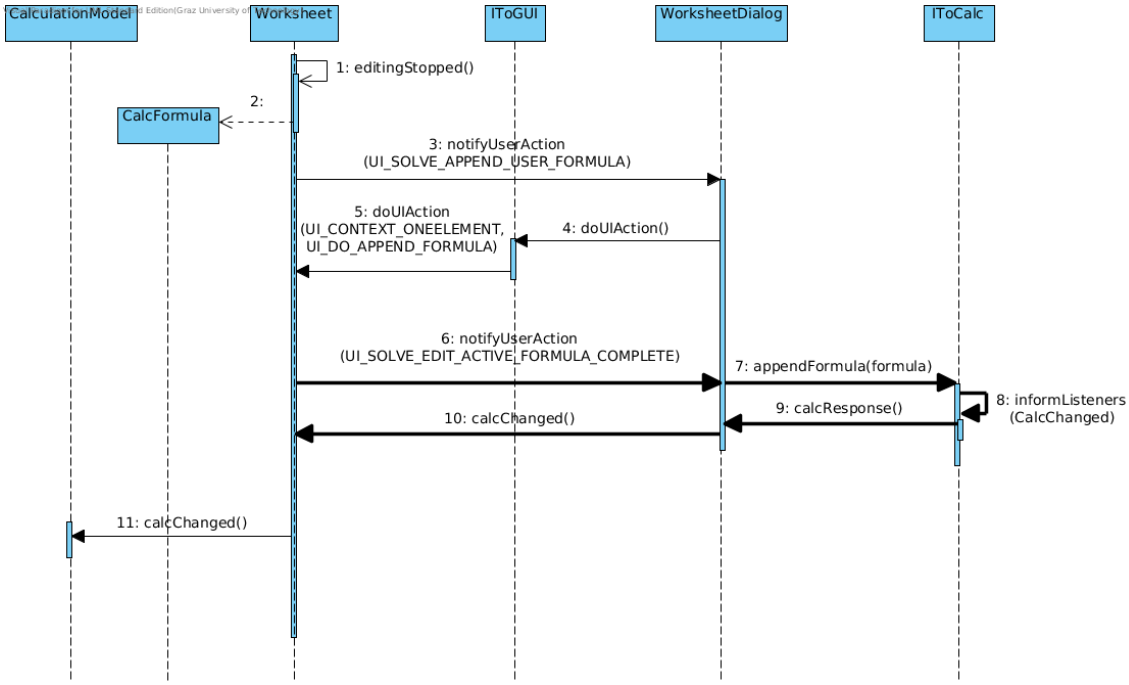

Figure 2.4: Interaction diagram for “append a formula”

ad 7 Checks if the formula is correct and calls informListeners with the calc changed event.

```
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()

```
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.

```
((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:

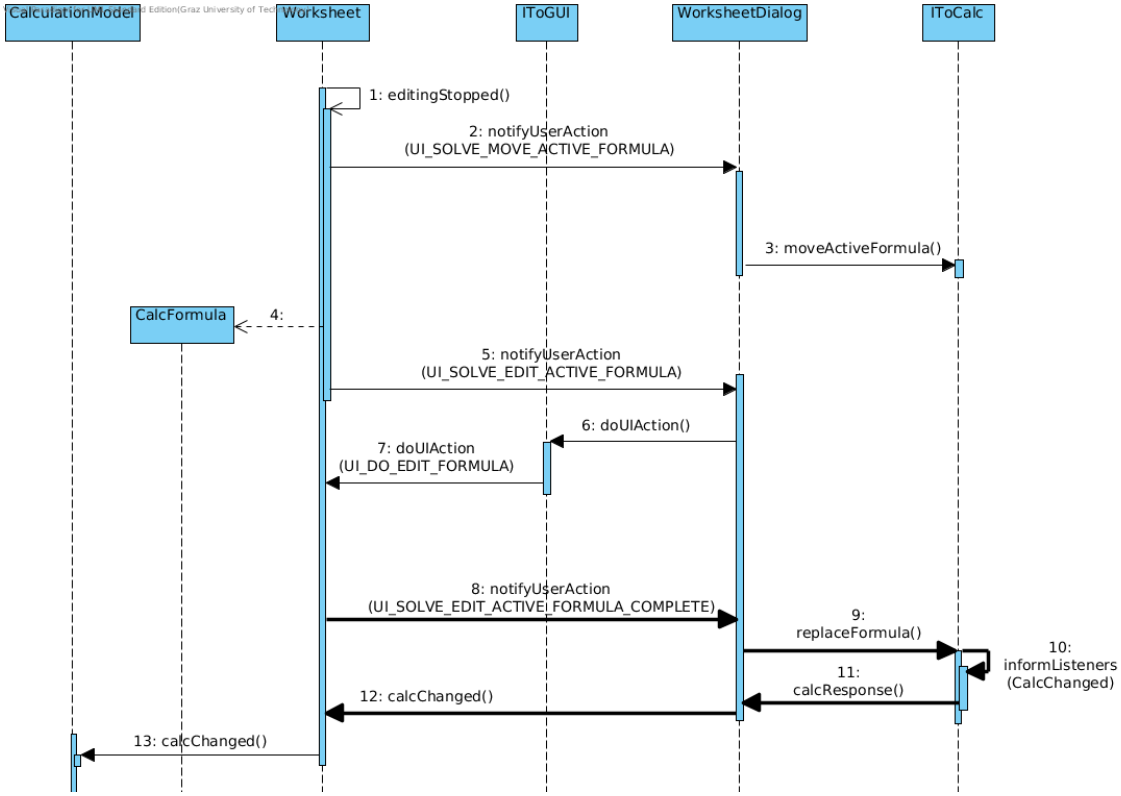

Figure 2.5: Interaction diagram for "replace a formula"

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

```
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.

```
((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.

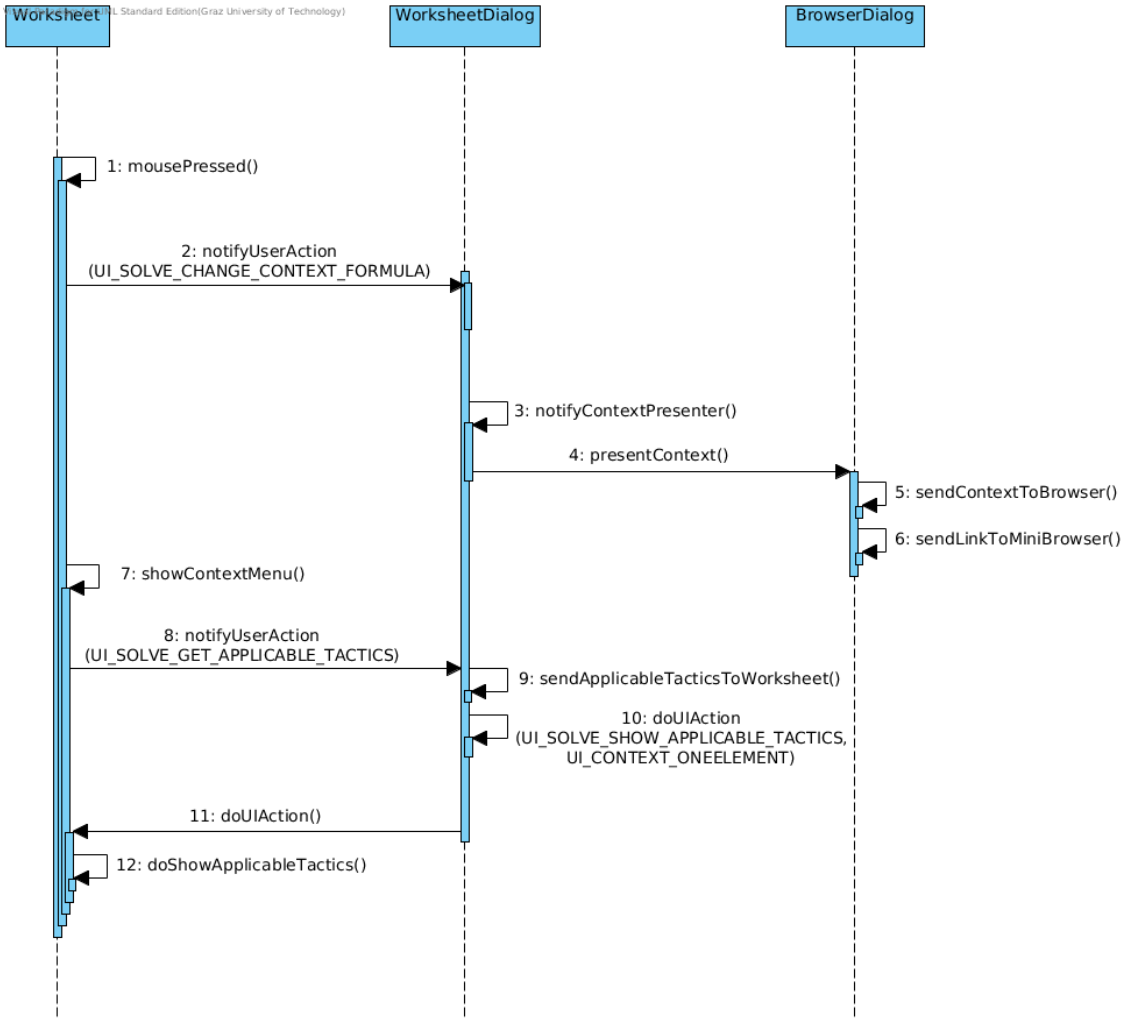

Figure 2.6: Interaction diagram for "Show tactic context"

ad 1 After clicking the tactic field (blank field under the formula field), the worksheet calls the function notifyUserAction with UI_SOLVE_CHANGE_CONTEXT_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!

```
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. ExampleDialiog, 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.

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

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

```
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 popup menu with the collection of applicable tactics.

```
case UI_CONTEXT_ONEELEMENT:
  switch (action.getAction())
    case UI_SOLVE_SHOW_APPLICABLE_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.

```
WorksheetDialog wd = this.ws_dialog_manager_.getActiveWSDialog();
if (wd != null) {
  wd.notifyUserAction(action);
}
```

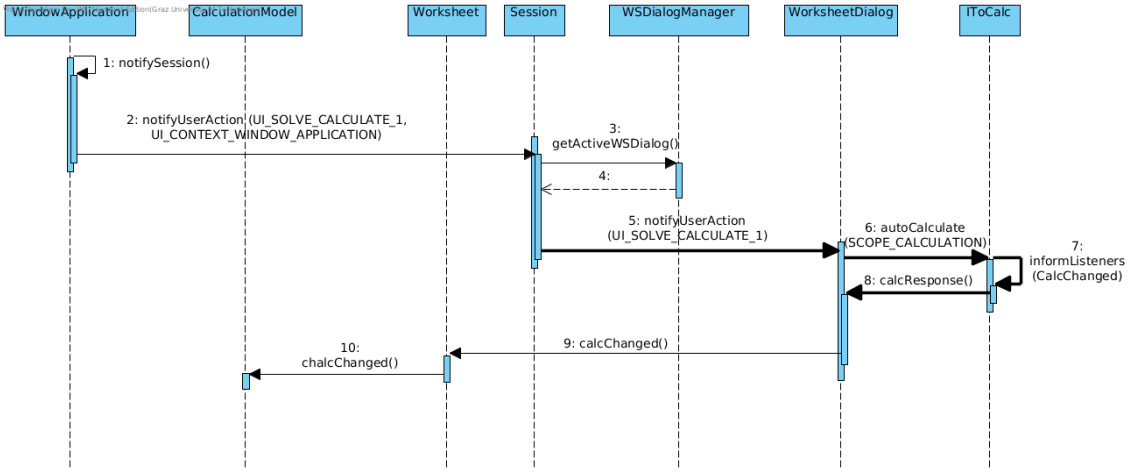

Figure 2.7: Interaction diagram for "Next Button"

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 math-engine the calculated step in form as a CalcChanged object and informs the listeners through the informListeners.

```
cc = math_engine_.autoCalculate(this.id_, scope, nSteps);
if (!(cc == null)) {
  transactionID = generateTransactionID();
  CalcChanged calcc = new CalcChanged(transactionID,true,
    new CalcIterator(this, cc.getLastUnchanged()),
    new CalcIterator(this, cc.getLastDeleted()),
    new CalcIterator(this, cc.getLastGenerated()));
  informListeners(calcc);
}
```

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

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

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

ad 10 Updates the nodes of the tree (CalculationModel)

Open Login Screen To start ISAC 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.

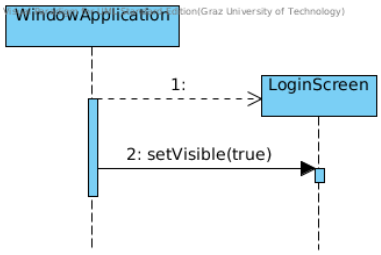

Figure 2.8: Interaction diagram for “Login Screen”

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

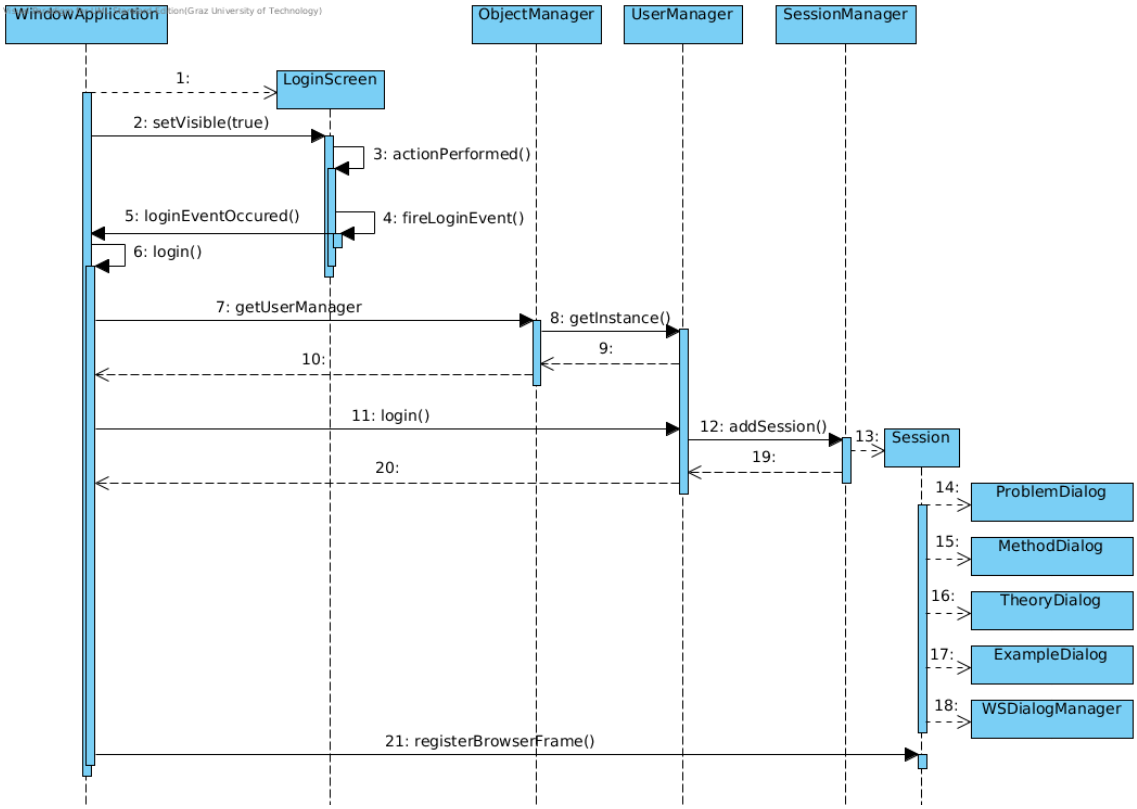

Figure 2.9: Interaction diagram for “Login”

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

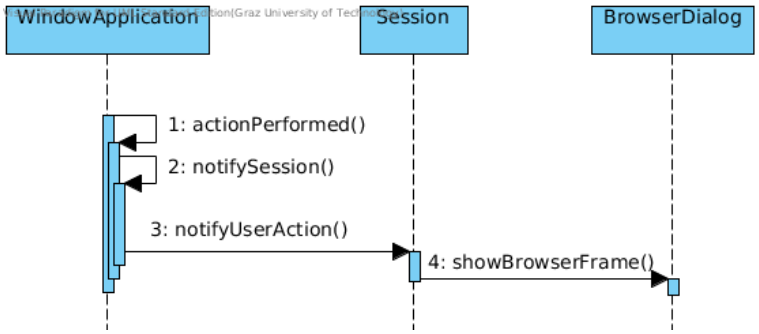

Figure 2.10: Interaction diagram for "Open BrowserDialog (Example, Method, Theory or ProblemBrowser"

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_METHO_BROWSER:
    method_dialog_.showBrowserFrame();
    break;
```

```
case UI_SHOW_PROBLEM_BROWSER:
  problem_dialog_.showBrowserFrame();
  break;
case UI_SHOW_THEORY_BROWSER:
  theory_dialog_.showBrowserFrame();
  break;
case UI_OPEN_WORKSHEET:
  ws_dialog_manager_.openWSDialog();
  break;
```

ad 4 Makes the selected browser frame visible.

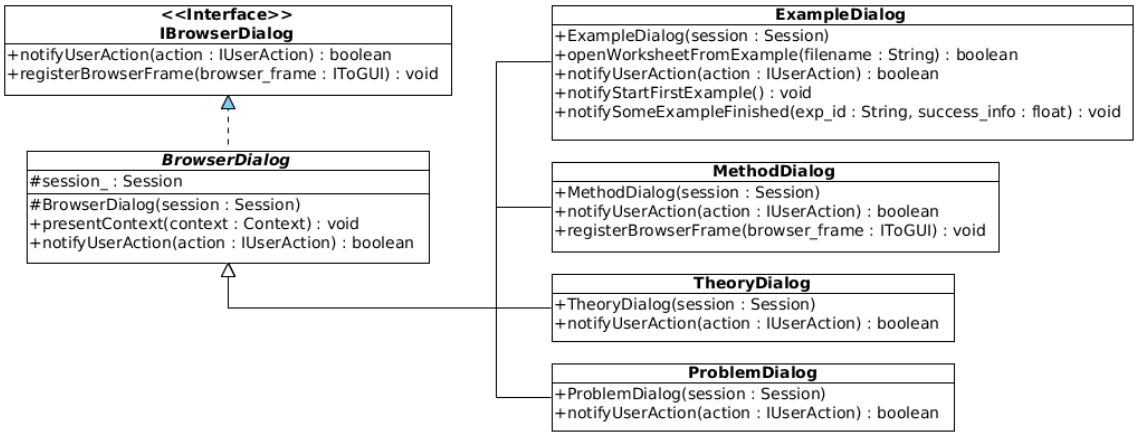

Figure 2.11: Interaction diagram for “Dependency of Browsers”

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 notifyUser-Action

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 frontend.

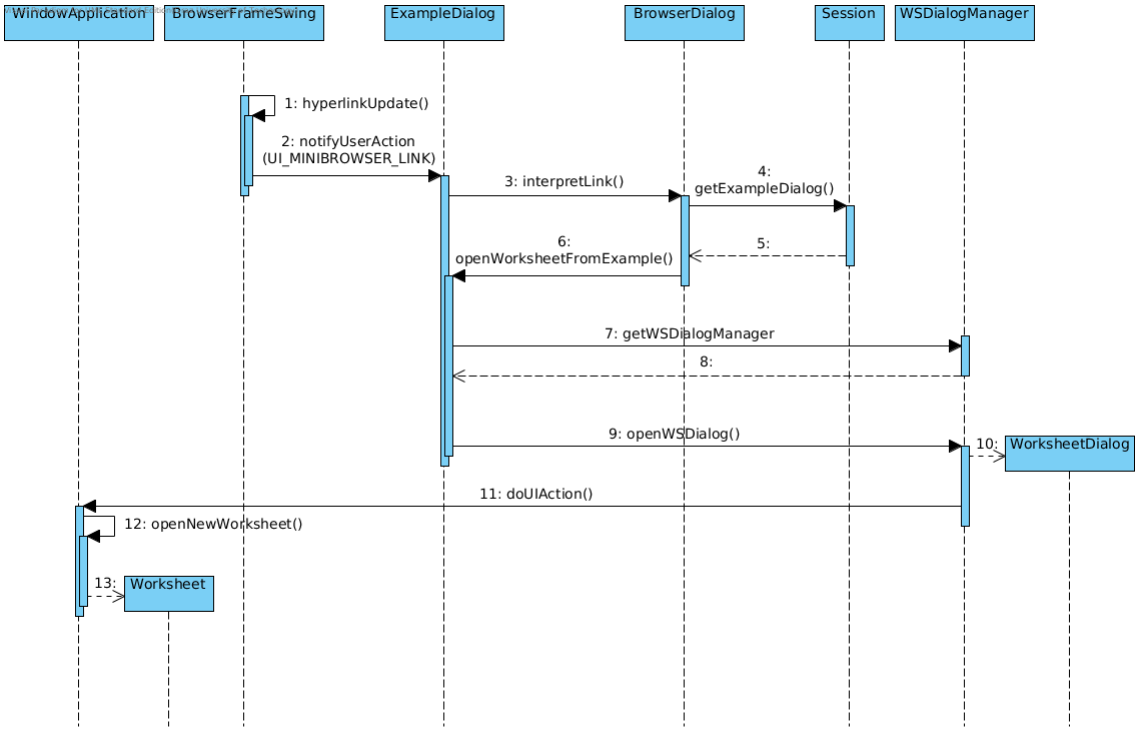

Figure 2.12: Interaction diagram for "Open an example"

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 *ISAC*'s principal architecture. After detailed analysis in §2.1 detailed design decisions can be made. Analyzed 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 *ISAC*'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 *ISAC* **can offer a list of actions known to the system.**

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

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

UR 2.2.4 *ISAC* **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 *ISAC* **can match a problem to a Model.**

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

UR 2.2.8 On request, *ISAC* **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 *ISAC* **records examples done by the user**

ISAC 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 *ISAC* **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 *ISAC* **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 *ISAC* **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 *ISAC* **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 *ISAC*.

2.3 Integration of a rule-based System

To integrate a rule-based System (RBS) has many advantages. At the beginning of the work there was a giant switch-case construction about 300 lines of code. It was very hard to change or expand something because it was very important that the given order remains. With rules is not less code but at a central point to change it.

Another important point is, that more rule-set can be used (e.g. an advanced learner has other rules as a beginner). A goal for the future is to disperse the <AUTO> and <NEXT> button, so the system will decide what the user should do. In this case more than one rule set is necessary. Thus we come to the borders of the switch-case.

For easy handling I decide to create a knowledge-base-creator helper-class, named RlbBuilder. It contains the knowledge creator and rules compiler.

One advantage of Drools is that this system has many features. For example it is possible to start a cron job [2] (timer function) - this can use for tests or time-limited assignments. [Ama12]

In combination with the new dialog should it be possible that the system creates independently exercises or tells a user he should try another calculation step. All that works, because the new dialog has all information of all dialogs and become the master mind of all dialogs.

[2] http://en.wikipedia.org/wiki/Cron

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 contend.

```
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.kestore.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

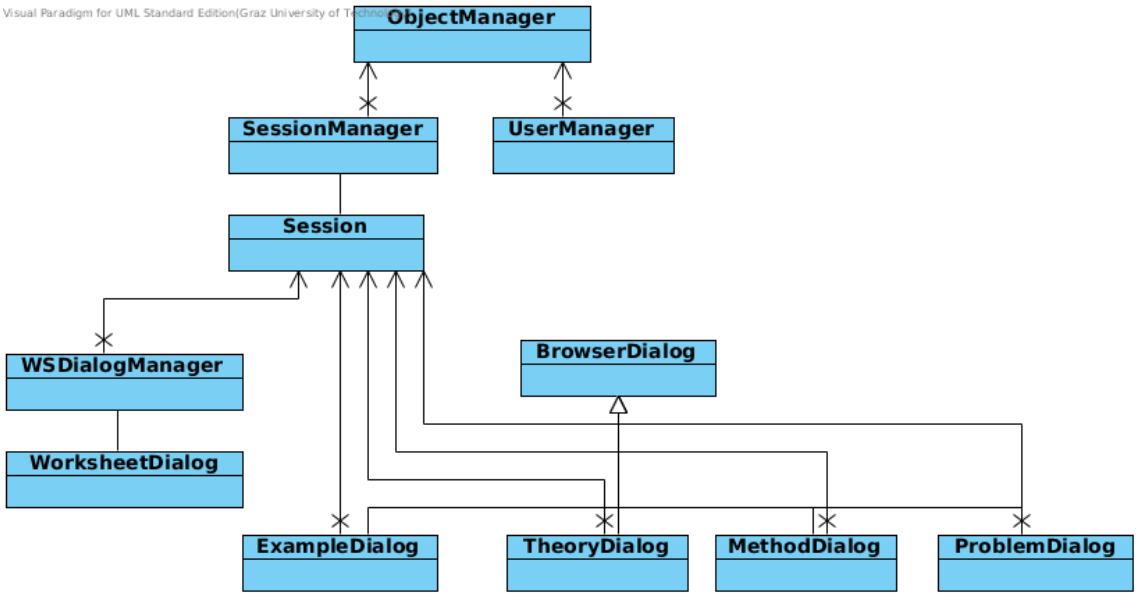

Figure 3.1: Overview dialog architecture

and a new user object would be create. 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 Selection of an appropriate rule-based system

Other tested rule-based systems:

- JRuleEngine
- JESS

All tested rule-based system build on the Java Rule Engine API JSR 94 [1].
There are many systems and each has its own advantages, but at the project time Drools had the best properties.

3.4 Detailed design of *ISAC*'s new dialog

The new dialog is the connection between Logitem (needed for UserLogger) and the whole dialogs. One DialogGuide for one user, like Session. Session handles technical issues for a user, whereas the DialogGuide handles learning issues. New dialog is the only place where

[1] `http://www.jcp.org/en/jsr/detail?id=94`

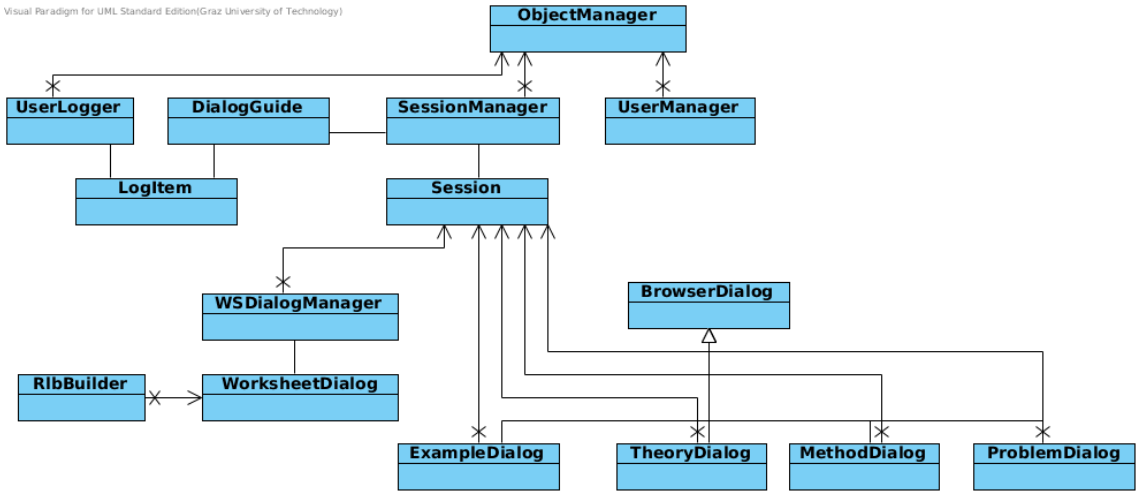

Figure 3.2: Design of new dialog

the UserLogger is written to. Its easy for an external application to get the LogItem because there is only one part in the code which handle that.

```
public void writeToDB(LogItem log_item)
{
  if(ObjectManagerPaths.LOGGER_DATABASE_ENABLED){
    user_logger_.saveLogItem(log_item);

  //add code here to expand
}
```

3.5 Integration of the rule-based system

On the drools homepage is a good documentation of the system. [Dro]

For this Project Drools 5.3.0 Final was used. In the download package of drools are many libraries for many special cases. So i figured out which are necessary for the core functions of drools.

1. antlr-3.3
2. antlr-runtime-3.3
3. core-3.4.2.v_883_R34x
4. drools-compiler-5.3.0.Final
5. drools-core-5.3.0.Final
6. drools-decisiontables-5.3.0.Final
7. knowledge-api-5.3.0.Final
8. mvel2-2.1.0.drools4
9. xstream-1.4.1

ad 1-2 is a parser generator
ad 3 is needed for netbeans (Java Development Tools Core Eclipse)
ad 4-6 drool specific tools
ad 7 build the knowledge base for rules
ad 8 is the dialect for the rules (not necessarily, but convenient to create rules).
ad 9 package to debug rules (write an output stream)

To use the RBS easily (knowledge base and rule compiler) a new class named isac.wsdialog.RlbBuilder was created. The RlbBuilder class need the class which fire the rules and the path of the rule file.

Constructor of RlbBuilder class:

```
RlbBuilder(Class<?> factoryclass, String rulepath)
```

After creating the Rule-Build (RLB) object, get access to knowledge base

```
public KnowledgeBase getKnowledgeBase()
{
    return this.kbase_;
}
```

So it looks in the WorksheetDialog

```
rlb_ = new RlbBuilder(WorksheetDialogRLB.class, "RlbLearnRLB.drl");
kbase_ = rlb_.getKnowledgeBase();
ksession_ = kbase_.newStatefulKnowledgeSession();
ksession_.insert(this);
ksession_.fireAllRules();
```

Create the RLB object (rlb_) and get the knowledge base (kbase_). With the knowledge base you can create a knowledge session (ksession_). Then its important to insert the object which should be applied with the rules. At least fire all rules (this tries to employ all applicable rules).
First the development of the rules running parallel in order that the other projects would not influenced. So the functionality of the WorksheetDialog was translated stepwise (start with little activities). The rules are in a separated file thus it's possible that rule-sets can changed dynamically.
At the end of intensive testing with testcases (generate from usecases) the old switch-case structure replaced with the new RBS. The clarity of the WorksheetDialog was improved.

A big problem was that drools can't access to private members or methods. For members use

```
ksession_.setGlobal("privateMember", private_member_);
```

to enclose. For the private methods it is not possible to access; can't access of an external knowledge to an internal object. There is only remains one possibility - public! All methods which are private and now public marked with:

```
/**
 * MKI DATE
 * *** DO NOT USE ***
 * Use this method only in WorksheetDialog!!!
 * Its public because this method is necessary for RLB
 * @deprecated
*/
```

3.6 Develop and test the usecases

Test-Driven Development
Test-driven development consists three main parts.[Bec03]

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.

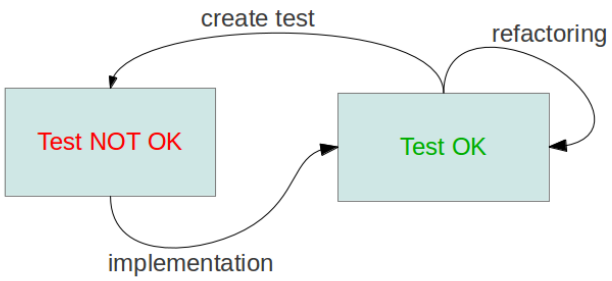

Figure 3.3: Schema of Test-Driven-Development

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 $\mathcal{ISAC}$ 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 $\mathcal{ISAC}$ Calculate the Next Formula
6. Having $\mathcal{ISAC}$ 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 Drools for dialog authors

We use the rule-based system Drools Expert for our experiments.
The experiments try to generalise *ISAC*s behaviour from a reactive system to an active system: At certain situations during stepwise problem solving *ISAC* shall take the initiative. One such situation is already clear: when the learner gets stuck, then the system shall offer help. Some early considerations about such help can be found here.

At the present Drools Expert is implemented in 'src/java/isac/wsdialog/WorksheetDialogRLB.java'. This implementation is parallel to WorksheetDialog.java and is able to run some tests in 'src/java-tests/isac/wsdialog/TestSolve.java'. The tests show that still some effort is required to replace WorksheetDialog by WorksheetDialogRLB.

Debug Drools Decomment the following lines in 'src/java/isac/wsdialog/WorksheetDialogRLB.java':

```
/* LOGGING RULE APPLICATION */
ksession_.addEventListener(new DebugAgendaEventListener());
ksession_.addEventListener(new DebugWorkingMemoryEventListener());
KnowledgeRuntimeLogger logger = KnowledgeRuntimeLoggerFactory.
           newFileLogger(ksession_, "./LOGFILE_Worksheetdialog" );
```

and

```
/* LOGGING RULE APPLICATION */
logger.close(); //close the stream
```

With these settings there is plenty output on the console, but little in './LOGFILE'. The latter does not accept a tilde in the path.

Be careful with identifiers and references

```
/* ATTENTION use the right class, e.g. 'WorksheetDialogRLB' and ruleset 'RlbLearnRLB' */
rlb_ = new RlbBuilder(WorksheetDialogRLB.class, "RlbLearnRLB.drl");
```

For further sources of errors compare 'src/java/isac/wsdialog/RlbLearnRLB.drl' and 'RlbLearn.drl'.

Replace WorksheetDialog with WorksheetDialogRLB

- mv WorksheetDialog.java WorksheetDialog-SAVE.java

- mv WorksheetDialogRLB.java WorksheetDialog.java
- rename identifiers of class and constructors
- and replace (very important)

```
rlb_ = new RlbBuilder(WorksheetDialogRLB.class, "RlbLearnRLB.drl");
```

with

```
rlb_ = new RlbBuilder(WorksheetDialog.class, "RlbLearn.drl");
```

Chapter 5

Summary and conclusion

This thesis tackled a practically relevant re-design and re-implementation in a joint effort with [Kob12]. 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 ISAC's dialog design from scratch and by reviewing the present state-of-the-art. The fundamental review confirmed the original decisions for ISAC's architecture. The extension of the dialog by a DialogGuide and a UserLogger was straight forward and did not affect ISAC'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 ISAC 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 ISAC. 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 feed-back 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 ISAC-project, however, is much more interested in learning than in grading.

Appendix A: Protocol of activities

Administrative work

Date	Hours	Description
04.07.2011 - 10.07.2011	10	Supervision: information about the ISAC-project, possible tasks ISACdocumentation, LaTeX
11.07.2011 - 17.07.2011	5	Supervision: LaTeX, proposal to Christian Gütl Working with netbeans
18.07.2011 - 24.07.2011	20	LatTeX netbeans, framework
20.04.2012 - 24.04.2012	23	editing Bakkalaureate Thesis in in LaTex

MS Installation

Date	Hours	Description
07.07.2011	5	Supervision: installation of the ISACsystem
11.08.2011	3	Supervision: integration of log4j

MS Dialog Architecture

Date	Hours	Description
11.07.2011 - 17.07.2011	8	classdiagram *ISAC*documentation, LaTeX
18.07.2011 - 24.07.2011	10	Supervision: review of UserLogger Analysis design
25.07.2011 - 31.07.2011	11	Supervision: overview isac architecture, class diagram Analysis design
08.08.2011 - 14.08.2011	14	Supervision: overview isac architecture, class diagram Analysis design
15.08.2011 - 21.08.2011	13	Supervision: questions to dialog architecture,observer pattern
29.08.2011 - 04.09.2011	20	Debugging with Log4j Supervision: questions, interface java – SML
05.09.2011 - 11.09.2011	19	Analysis Design with Log4j debug output Supervision: interaction diagram: objects,methods for solve phase
12.09.2011 - 18.09.2011	16	Designed interaction diagram Supervision: interaction diagram: calculate one step
19.09.2011 - 25.09.2011	18	Designed interaction diagram Supervision: interaction diagram: append and replace formula
26.09.2011 - 02.10.2011	20	CalcMessage, Calchead locking, Logger Redesign interaction diagram for “replace and append a formula”
03.10.2011 - 09.10.2011	15	Supervision: discuss about the correctness of the current interaction diagrams

MS Select Rule-Based System (RBS)

Date	Hours	Description
08.08.2011 - 14.08.2011	12	Research of Rule-Based Systems Testing JESS, JRuleEngine
15.08.2011 - 21.08.2011	13	Supervision: demo of RBSs, assessment criteria of RBSs Testing Drools

MS Detailed Design of New Dialog

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

MS Integration of the RBS

Date	Hours	Description
06.02.2012 - 12.02.2012	20	transform switch-case to rules integration of rlb

MS Develop and Test the Usescases

Date	Hours	Description
26.09.2011 - 02.10.2011	10	eliminate ObjectManager reactivate ObjectManager
13.02.2012 - 19.02.2012	30	Develop Usecases and test the translated rules

Bibliography

[Ama12] Lucas Amador. *Drools developers cookbook: over 40 recipes for creating a robust business rules implementation by using JBoss Drools rules*. Birmingham, U.K. : Packt, 2012.

[Bec03] K. Beck. *Test-Driven Development by Example*. Addison Wesley, 2003.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Som merlad, , and Michael Stal. *Pattern-Oriented Software Architec- ture: A System of Patterns*. John Wiley & Sons, 1996.

[Dar00] Keith Darlington. *The Essence of Expert Systems*. Pearson Education, 2000.

[Dro] Drools. http://www.jboss.org/drools/documentation. last retrieved 29.Feb.2012.

[NK12] Kober Franz Neuper, Walther and Markus Kienleitner. Isac wiki, April 2012. http://www.ist.tugraz.at/isac/index.php/Development_Environment_for_ISAC_Front-end/.

[Eck95] Wayne W Eckerson. Three tier client/server architecture: Achieving scalability, performance, and efficiency in client server applications. *Open Information Systems*, 10, January 1995.

[Fow02] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[GMW79] M. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[GR05] Joseph C. Giarratano and Gary Riley. *Expert Systems, Principles and Programming*. Course Technology Ptr, 2005.

[HKN10] Florian Haftmann, Cezary Kaliszyk, and Walther Neuper. CTP-based programming languages ? considerations about an experimental design. *ACM Communications in Computer Algebra*, 44(1/2):27–41, 2010.

[iT02a] *ISAC* Team. *ISAC* – user requirements document, software requirements document, architectural design document, software design document, use cases, test cases. Technical report, Institute for Softwaretechnology, University of Technology, 2002. http://www.ist.tugraz.at/projects/isac/publ/appendices.ps.gz.

[iT02b] *ISAC* Team. *ISAC* appendices to the analysis and design documents. Technical report, Institute for Softwaretechnology, University of Technology, 2002. http://www.ist.tugraz.at/projects/isac/publ/appendices.ps.gz.

[iT02c] *ISAC* Team. *ISAC* software design document. Technical report, Institute for Softwaretechnology, University of Technology, 2002. http://www.ist.tugraz.at/projects/isac/publ/sdd.ps.gz.

[iT02d] *ISAC* Team. *ISAC* softwarerequirements document. Technical report, Institute for Softwaretechnology, University of Technology, 2002. http://www.ist.tugraz.at/projects/isac/publ/srd.pdf.

[iT02e] *ISAC* Team. *ISAC* use cases. Technical report, Institute for Softwaretechnology, University of Technology, 2002. http://www.ist.tugraz.at/projects/isac/publ/use.ps.gz.

[iT02f] *ISAC* Team. Userrequirements document. Technical report, Institute for Softwaretechnology, University of Technology, 2002. http://www.ist.tugraz.at/projects/isac/publ/urd.pdf.

[Jac98] Peter Jackson. *Introduction to Expert Systems*. Addison Wesley, 1998.

[Kie12] Markus Kienleitner. Towards "nextstep userguidance" in a mechanized math assistant. Master's thesis, IICM, Graz University of Technology, 2012. Bakkalaureate Thesis.

[KN08] Alan Krempler and Walther Neuper. Formative assessment for user guidance in single stepping systems. In Michael E. Aucher, editor, *Interactive Computer Aided Learning, Proceedings of ICL08*, Villach, Austria, 2008.

[Kob12] Franz Kober. Logging of high-level steps in a mechanized math assistant. Master's thesis, IICM, Graz University of Technology, 2012. Bakkalaureate Thesis.

[Kre05] Alan Krempler. Architectural design for integrating an interactive dialogguide into a mathematical tutoring system. Master's thesis, University of Technology, Institute for Softwaretechnology, Graz, Austria, March 2005. http://www.ist.tugraz.at/projects/isac/publ/da-krempler.pdf.

[Neu] Walther Neuper. Announcement for an upcoming generation of tp-based educational mathematics assistants. submitted to eduTPS at CADE2012.

[Neu06] Walther Neuper. Angewandte Mathematik und Fachtheorie. Technical Report 357, IMST – Innovationen Machen Schulen Top!, University of Klagenfurt, Institute of Instructional and School Development (IUS), 9010 Klagenfurt, Sterneckstrasse 15, 2006. http://imst.uni-klu.ac.at/imst-wiki/index.php/Angewandte_Mathematik_und_Fachtheorie.

[Neu07] Walther Neuper. Angewandte Mathematik und Fachtheorie. Technical Report 683, IMST – Innovationen Machen Schulen Top!, University of Klagenfurt, Institute of Instructional and School Development (IUS), 9010 Klagenfurt, Sterneckstrasse 15, 2007. http://imst.uni-klu.ac.at/imst-wiki/index.php/Angewandte_Mathematik_und_Fachtheorie_2006/2007.

[Neu12] Walther Neuper. Automated generation of user guidance by combining computation and deduction. In Pedro Quaresma and Ralph-Johan Back, editors, Proceedings First Workshop on *CTP Components for Educational Software,* Wrocław, Poland, 31th July 2011, volume 79 of *Electronic Proceedings in Theoretical Computer Science*, pages 82–101. Open Publishing Association, 2012.

[NR08] Walther Neuper and Johannes Reitinger. Begreifen und Mechanisieren beim Algebra Einstieg. Technical Report 1063, IMST – Innovationen Machen Schulen Top!, University of Klagenfurt, Institute of Instructional and School Development (IUS), 9010 Klagenfurt, Sterneckstrasse 15, 2008.
http://imst.uni-klu.ac.at/imst-wiki/index.php/Begreifen_und_Mechanisieren_beim_Algebra-Einstieg.

[Pfa85] G. E. Pfaff, editor. *User Interface Management Systems: Proceedings of the Seeheim Workshop*, Berlin, 1985. Springer Verlag.

[Ram00] Ariel Ortiz Ramirez. Three-tier architecture. *Linux Journal*, 75(3508), July 2000.

[SG96] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[SL06] Kirk P. H. Sullivan and Eva Lindgren, editors. *Computer Keystroke Logging and Writing: Methods and Applications*. Elsevier, 2006.