Architectural Design for
Integrating an Interactive Dialog Guide
into a Mathematical Tutoring System

Alan Krempler

Master’s Thesis

IST - Institute for Software Technology
Graz University of Technology

TUG

Graz, March 2005

Supervisor: em. Univ. Prof. Dipl.-Ing. Peter Lucas
Project coordinator: Mag. Dr. Walther Neuper
Acknowledgements

This thesis as much as most other achievements in my life would not have been possible without very special people, who put trust in me waiting patiently without being indifferent, who gave me their commitment without meddling, in short, people who were present to give me their support in doing things my very own way.

While it is not possible to name everyone, I hope that if you are meant by these words, you will feel that and accept my gratitude for being with me.
Abstract

ISAC is an evolving mathematical tutoring system. One key feature of ISAC is a solving process which is made visible to the user and uses the same knowledge and concepts of problem-solving the user is intended to learn. One other key feature is interactive user guidance configurable to the changing needs of the individual user.

In the present thesis, possibilities for integrating interactive user guidance into a mathematical tutoring system are explored, based on requirements analysis in both technological and didactic aspects. The chosen system design is detailed down to the interface level.

The design for a component for interactive user guidance in problem-solving situations, called the Dialog Guide, is developed and experiences gathered in implementing a prototype are summarised.
Contents

I Introduction .................................................. 5
  0.1 The ZS4C Project ........................................ 6
  0.1.1 History, Scope and Goal of the ZS4C Project ...... 6
  0.1.2 Related Projects and Products ....................... 8
  0.1.3 Status of the Project ................................ 12
  0.2 Scopes and Goals ......................................... 12
    0.2.1 Goal of this Thesis ................................ 12
    0.2.2 Position of this Thesis within the ZS4C Project ... 13
    0.2.3 Structure of this Thesis ............................. 14
  0.3 Introduction to Terms Used throughout this Thesis ... 15
    0.3.1 System Components of ZS4C ......................... 15
    0.3.2 Abstraction Levels of Dialog Behaviour ............ 16
    0.3.3 Key Elements in Calculations ....................... 16
    0.3.4 ZS4C’s Concept of Calculations ..................... 17
    0.3.5 Mathematical Knowledge ............................ 18
    0.3.6 Describing a Calculation Task to ZS4C ............. 18

II Requirements Analysis ..................................... 20
  1 Use Cases .................................................. 22
    1.1 Initialising the Dialog ................................. 22
      1.1.1 Choosing a Starting Point ......................... 23
    1.2 Initialising a Calculation ............................. 23
      1.2.1 Modeling and Specifying .......................... 23
      1.2.2 Obtaining Help from ZS4C ......................... 24
      1.2.3 Alternate Views on Model and Specification .... 25
      1.2.4 Starting the Calculation .......................... 26
    1.3 Calculating a Result ................................... 27
      1.3.1 Moving the Active Formula ....................... 27
1.3.2 Taking Single Steps Interactively .................................. 27
1.3.3 Automatic Calculation .............................................. 28
1.3.4 Showing and Hiding Data ........................................... 29
1.3.5 Obtaining Help and Extra Information ............................ 31

2 Selected Requirements of the \textit{TSAC} System .................. 32
   2.1 Users of \textit{TSAC} .................................................. 32
   2.2 Calculations and Data Involved ..................................... 33
   2.3 Miscellaneous .................................................... 35

3 Requirements Specific to the Dialog Guide ....................... 36
   3.1 User Guidance ..................................................... 37
   3.2 User Profiling .................................................... 38
   3.3 Flexible Dialog Behaviour ........................................ 39
   3.4 Adaptation to Individual Users ................................... 40
   3.5 Restrictions ...................................................... 41

III Design Considerations ................................................. 42

4 User Interaction from a Didactic Point of View .................. 43
   4.1 Basic Theories about Learning and Teaching ..................... 43
       4.1.1 Behaviouristic Learning .................................... 43
       4.1.2 Cognitivistic Learning .................................... 45
       4.1.3 Constructivistic Learning .................................. 46
   4.2 Modelling Interaction With a Peer ................................ 48

5 User Interaction from a Technical Point of View ................. 50
   5.1 Describing a Dialog ............................................... 50
       5.1.1 Dialog Atoms ............................................... 50
       5.1.2 Sequencing ............................................... 51
       5.1.3 Knowledge Items ........................................... 51
       5.1.4 Situation of the User ...................................... 52
   5.2 Modelling a User ................................................ 52

6 System Environment ...................................................... 53
   6.1 Basic Concepts for Separable User Interfaces .................. 53
       6.1.1 The Seeheim Model ........................................ 53
       6.1.2 The MVC Architecture ..................................... 54
       6.1.3 Comparing the approaches .................................. 54
       6.1.4 Implications for \textit{TSAC} ................................ 55
   6.2 Distributed Components in a Heterogenous Environment ...... 57
       6.2.1 Location of the Dialog Guide ............................... 57
       6.2.2 Interfacing Java and SML ................................. 58
       6.2.3 Choosing a Means of Communication ...................... 58

2
6.2.4 The Dinopolis Middleware project .................. 59
6.2.5 Java-RMI .............................................. 60
6.3 The Architectural Design of the T$\overline{S}$AC System ........................ 61
   6.3.1 The Overall Design of T$\overline{S}$AC ..................... 61
   6.3.2 The Interfaces to the Dialog Guide Component .......... 65

7 Designing a Dialog Guide Component .......................... 67
   7.1 Controlling the Course of Interaction .................... 67
   7.1.1 Dialog Phases ........................................ 67
   7.1.2 Dialog Atoms .......................................... 69
   7.2 Sharing the Calculation with other Components ........... 70
   7.2.1 Representing the Specification ......................... 70
   7.2.2 Representing the Path to the Solution ................. 70
   7.2.3 Treating Subproblems ................................ 70
   7.2.4 Accessing Calculation Data ............................ 71
   7.2.5 Communicating Changes in the State of Calculation .. 72
   7.3 Configuring the User-Interface ........................... 73
   7.3.1 The Presentation Layer in Control ..................... 73
   7.3.2 The Dialog Guide in Control ......................... 74
   7.3.3 Splitting up Responsibilities and Providing for Inter-
        action ................................................... 74
   7.4 Obtaining and Storing Configuration Data ................ 75
   7.4.1 The User Settings .................................... 75
   7.4.2 Permissions and Security Issues ....................... 75
   7.4.3 The User Model ...................................... 75

IV Implementation .............................................. 77

8 Representing Calculation Data .................................. 78
   8.1 Storing Enumeration Types ............................... 78
   8.2 The Hierarchy of Mathematical Objects .................... 78
      8.2.1 CalcElement ....................................... 78
      8.2.2 Classes Derived from CalcElement .................. 79
      8.2.3 Subterms .......................................... 81
   8.3 Iterators for Navigating the CalcTree: ICalcIterator ...... 81

9 Communicating with Other Components .......................... 83
   9.1 Data Types Used for Communication ....................... 83
      9.1.1 Events ............................................. 83
      9.1.2 Exceptions ........................................ 84
   9.2 Interfaces used by the Dialog Guide ...................... 85
      9.2.1 Communicating Towards the Calculation: IToCalc .. 85
      9.2.2 Communicating Towards the User: IToUser .......... 86
9.2.3 The Presentation Layer as Seen from the Dialog Guide:  
  IT0Worksheet .................................................. 86
9.2.4 The Dialog Guide as Seen from the Presentation: DGuide 87
9.3 Communicating with the UserModel .......................... 88

V Summary, Project Reflection, Future Developments 90

10 Summary and Project Reflection ............................ 91
  10.1 System Architecture ..................................... 91
  10.2 Dialog Guide .............................................. 92

11 Possible Future Developments ............................... 93
  11.1 The ZS4C System ........................................ 93
  11.2 The Dialog Guide ....................................... 94

Bibliography .................................................... 95

VI Appendices .................................................. 100

A An example for reference .................................... 101
  A.1 Description, formalization and modeling phase ....... 101
  A.2 Knowledgebase and specification phase .............. 102
  A.3 Interaction on the worksheet and the browsers ..... 104
  A.4 The solving phase and subproblems .................. 106

B ZS4C’s tactics ................................................. 108

C Requests Recognised or Issued by the Dialog Guide .... 110
  C.1 Requests Recognised During the Specifying Phase .. 110
  C.2 Requests Recognised During the Solving Phase ..... 111
  C.3 Requests Issued by the Dialog Guide ................. 112
Part I

Introduction
0.1 The ISAC Project

0.1.1 History, Scope and Goal of the ISAC Project

The ISAC project originates from mathematical ideas of Bruno Buchberger [Buc82] and software-engineering expertise of Peter Lucas, which inspired Walther Neuper to develop the idea of a novel mathematics tutoring system, described in his PhD thesis [Neu01]. A substantial technological foundation is the theorem prover Isabelle [Pau94] and its high order logic HOL.

Let us begin the introduction with a quotation from the project’s homepage 1:

The ISAC-project is a research and development project at the Institute for Software Technology of the Graz University of Technology.

The ISAC-system establishes new base-technology for a novel kind of interactive and web-based transparent software for applied mathematics. The novelty is given by the human-readable knowledge base including Isabelle’s HOL-theories and by the transparently working knowledge interpreter (a generalization of ‘single stepping’ algebra systems).

Research within the ISAC-project goes alongside the development of the ISAC-system and concerns issues, which reach from technological ones like ‘math on the (semantic) web’ down to fundamental ones like a ‘rigorous formal semantics of tools for applied mathematics’.

The background to both, development and research, is given by actual needs in math education as well as by foundamental questions about ‘the mechanization of thinking’ as an essential aspect in mathematics and in technology.

The ISAC-system under construction comprises a tutoring-system and an authoring-system. The latter provides for adaptation to various needs of individual users and educational institutions and for extensions to arbitrary fields of applied mathematics.

At the end of the ongoing development process, ISAC is meant to enable the user to:

- specify calculations interactively,
- do calculations interactively on a worksheet resembling traditional pencil-and-paper work,

1http://www.ist.tugraz.at/projects/isac/
• choose to have parts of a calculation done automatically by the system
  and to do other parts by himself,
• ask for justifications for every single step in a calculation,
• edit and modify calculations in progress, exploring alternatives,
• browse the knowledge used by the system, augmented with descriptive
texts, figures and examples,
• configure the system to use specific knowledge,
• configure the system to use specific methods for solving problems,
• configure the system to use specific teaching strategies,
• be guided by a system adapting to the user’s experience,
• author examples, methods for solving problems and teaching strategies,
• exchange data with other users of TSAC as well as export data to other
  mathematics or publishing tools.

The work to be done to reach this ambitious goal is best described by a
quotation from an application for a FWF-Grant[Neu02a]:

The issue is to develop integrated tools for all aspects of doing
mathematics: tools for inventing - formalizing - exploring -
proving - managing - applying - publishing mathematics.

The prototype being developed aims at the level of teaching high-school
mathematics, but will be designed as general as possible to remain open to
future developments.

Much theoretical work has to be done to abstract the process of proving,
the processes involved in doing calculations and the processes involved in
teaching, explaining and exploring mathematics into formalisations or lan-
guages. While the abstraction of the logic of proofs into Isabelle’s HOL
has already been done by the Isabelle team, the development of TSAC is
concerned with the following:

• Abstracing the process of specifying a calculation into Problems
• Abstracing the process of doing calculations into Tactics and Methods
• Abstracing the processes of teaching, explaining and exploring into
  Dialog Atoms and Dialog Strategies
These abstractions will be explained in detail in section 0.3 below.

Many system designs fear embarrassing the user if the internal abstractions became visible, and consequently try to hide their interna. ISAC, on the other hand, regards exposing the system's reasoning and confronting the user as one key opportunity for exploring the world of mathematics and providing for constructivist learning experiences. Therefore, the guideline in structuring ISAC's interna is not algorithmic sophistication but rather modelling of human paths of thought.

As a convenient side-effect, ISAC will be extensible by authoring tools not requiring any expertise in computer programming but interacting with the author in terms from his area of expertise:

- Examples in terms of descriptive text and figures, formulas given and desired results
- Methods for solving mathematical Problems in terms of Tactics to be applied
- Teaching strategies in terms of asking questions, giving choices, leaving parts of formulas blank for the user to fill in or letting the user try for himself

0.1.2 Related Projects and Products

To further illustrate the concept of ISAC, a short comparison with related projects, products and technologies will be given, highlighting common grounds and substantial differences.

Educational Software

A wealth of learning software has become available over the past decades, ranging from simple flashcard-like programs to elaborate interactive systems. Overviews of readily available software can be found in the WWW\(^2\).

**Common grounds:** ISAC can be regarded as a math learning software insofar as we share the goal of extending the users' knowledge and ability to solve mathematical problems. We also share the methods of both presenting knowledge and practising examples in an interactive way.

**Major differences:** Most learning software works from pre-defined libraries of examples and lessons, and guides the user along didactically sophisticated, but fixed paths of interaction. ISAC is extensible in both respects, offering support and learning experience even for user's examples not previously known to the system. Moreover, ISAC's paths

\(^2\)http://www.educational-software-directory.net/math/
of interaction are designed to be composed of so-called Dialog Atoms (see section 5.1.1), providing for user guidance configurable by a supporting teacher or even automatically adapting itself to the learner’s behaviour.

A few programs will be mentioned for their special relationship with IS4C:

MathXpert ³ [Bee92] is remarkable for its flexible and interactive approach as opposed to pre-defined lessons. MathXpert guarantees correct results while leaving the way to the solution open to the user. Every step taken is justified by an explanatory statement.

ActiveMath [MAF+01] defines adaptation to the individual learner’s preference, knowledge and capabilities as a goal. It abstracts is experience with an individual user in an User Model.

Theorema [BJ98] differentiates the phases of specifying, exploring, proving, programming and writing up in solving mathematical problems. Especially the phase of exactly specifying the problem is a often-overlooked but central skill in doing math.

Computer Algebra Systems (CAS)

Common grounds: The user’s first impression of IS4C will most closely resemble a CAS. As with CAS, most user interaction will be calculating examples on a worksheet. We share the starting point of specifying a mathematical problem and the goal of eventually reaching a final result, after a number of transformations. Both CAS and IS4C guarantee mathematically correct results. In fact, IS4C can be used as a CAS.

Major differences: CAS traditionally concentrate on the goal of providing solutions to as wide a spectrum of computational problems as possible and providing these solutions fast. It is not likely that IS4C will reach a comparable speed or wealth of computational functionality any time in the near future. Superior computational power of CAS has been achieved by optimisation of algorithms, making the process of solving intransparent to the user. A quotation from an internet-forum illustrates the problem:

Q: Hi, Can Mathematica show the steps that make to arrive at the answer, in a integration

A: Nope, sorry. Mathematica doesn’t do integrals like humans do. You really don’t want to see what route it takes to get an answer.⁴

⁴http://forums.wolfram.com/student-support/topics/7921, nov-2004
Efforts to present intermediate results have been taken only recently. \textsc{Isac}, on the other hand, concentrates on making the process of solving transparent to the user. \textsc{Isac}'s mathematical knowledge is coded in the same rules and propositions used in teaching math and stored in a knowledge base browsable and comprehensible by the user. \textsc{Isac}'s computational algorithms are expressed using these rules and stored in so-called Methods (see 0.3.5), which are accessible to the user as well. This provides not only for transparency, but also the flexibility to extend the knowledge used in obtaining a result and even changing the very way how a solution is obtained. \textsc{Isac}'s interactive tutoring capabilities going far beyond mere presentation of examples are based on this flexible concept.

In contrast to most available CAS, \textsc{Isac} is a networked application, allowing for mathematical knowledge and example libraries being shared across the internet.

\textsc{Isac} puts more emphasis on specifying the initial problem than CAS do.

While CAS can be used for learning purposes insofar as they facilitate exploration by taking the burden of calculation off the user, \textsc{Isac} can actively influence user interaction to achieve pedagogic goals.

The following programs are mentioned for their wide-spread use in educational and scientific institutions:

\textbf{Mathematica}\textsuperscript{5} [Wol96] boasts extensive computational capabilities, both symbolic and numeric. Mathematica started adding user-transparency by offering display of intermediate results and even the rules applied for differentiation.

\textbf{Derive}\textsuperscript{6} [Sof94] is an intentionally compact program running on smaller machines and even hand-held calculators. Derive started adding access to intermediate results - called display step - as of version 6.

\textbf{MathCad}\textsuperscript{7} has its strengths in presentation and integrating a calculation with descriptive text and figures.

\textbf{Theorem Provers}

\textbf{Common grounds:} Theorem provers apply the abstract logic of proofs to a problem, in a step-by-step manner with a high degree of interactivity. \textsc{Isac} implicitly proves that result of a calculation is correct in the sense of the original problem. \textsc{Isac} does so by constructing the result of a calculation by applying theorems proven by the theorem prover Isabelle [Pau97].
**Major differences**: A theorem prover tries to answer the question 'Is the theorem true?'. A theorem prover's answer is yes or no. ISAC tries to solve a problem. ISAC's answer is a mathematical object called result, as with a CAS.

**Expert Systems**

**Common grounds**: ISAC's Knowledge Base could be regarded as a simple expert system for mathematical knowledge.

**Major differences**: Experts systems concentrate on abstracting, storing and retrieving knowledge, whereas ISAC emphasises application of the knowledge stored in its Knowledge Base. While expert systems are designed to deal flexibly with incomplete or even contradictory information, ISAC requires the knowledge in the Knowledge Base to be exact.

**Standards and Tools Used**

Several evolving technologies are used in the ISAC system, with their development closely observed and in some cases cooperatively influenced by ISAC:

**XML**\(^8\), an open standard for description of structured data, is used on all interfaces to the outside world. With limited resources of the ISAC project, open standards facilitate cooperation with other projects and the use of tools already available.

**MathML**\(^9\), a XML language for describing mathematical objects, is used both for processing and rendering of formulae. Technology for processing MathML is not widely available at the moment, but evolving rapidly.

**MoWGLI**\(^10\) [Neu03] aims at integrating and evolving existing standards to enhance accessibility and usability of mathematical information in the WWW.

**Dinopolis**\(^11\) [Sch02a], is a componentware framework for robust and secure distributed systems. While not being used in the current prototype, the concepts of Dinopolis influenced the design of ISAC's architecture and we hope to use Dinopolis for the integration of ISAC’s components in a future release.

**Isabelle** As stated before, the theorem prover Isabelle’s high-order logic is used in ISAC’s math engine.
0.1.3 Status of the Project

As a feasibility study a first prototype added a graphical, but non-interactive Worksheet by Thomas Fink [Fin01] and a network bridge by Thomas Oberhuber [Obe01] to the Math Engine developed by Walther Neuper [Neu01].

A second-generation prototype is under development with the aim to demonstrate the novel key features of TSAC's design. The demonstrations will aim at attracting interest of both, of opinion leaders in TSAC's application area (i.e. in education) and of researchers in several disciplines [Neu04a].

At the time of writing of this thesis the status of the second prototype was as follows:

New Theories, Problems and Methods have been added to the Math Engine by Stefan Karmel [Kar02], Richard Lang [Lan03] and Matthias Goldgruber [Gold03].

Andreas Griesmayr [Gri03] extended the system design to accomodate a Knowledge Base and the respective browser.

A first step to an interactive Worksheet making use of the new system design has been implemented by Mario Hochreiter [Hoc04], work on the Bridge to the Math Engine has been done by Richard Gradischnegg [Gra04].

0.2 Scopes and Goals

0.2.1 Goal of this Thesis

This thesis concentrates on the TSAC system being used for interactive automatic tutoring. The goal is designing a software component capable of managing the communication between a user and the Math Engine. Management of communication in this context is not only pertaining to technical aspects such as networking, communication protocols, and data integrity, but also and especially to high-level interaction between a user in a learning situation and a system having expertise in the field in question. Key functions of such a component include:

Retrieving information needed

Filtering superfluous, unneeded or confusing information

Distributing activity - and thus effort and responsibility for the result - between the user and the system

Offering help and additional information, when needed

To perform these tasks, the component must be integrated into the system with control over all information and activities relevant to the user.
Because learning is a very individual process, learning strategies and therefore requirements regarding the above functions vary with individuals, their learning goals and their previous knowledge. Under these circumstances, it seems impossible to design a universal learning strategy or even a sub-strategy for management of communication.

Consequently, the component's behaviour must be configurable to meet the needs of a specific situation. In addition to being configurable by the learner or a supervising teacher, we aim at making the component dynamically self-reconfigurable at runtime to react to events occurring during a session. The objects or types of information involved in a problem-solving session are well-known from a mathematical point of view and their relations can be described by the language of logic and mathematics. This makes these objects recognisable and the component's behaviour configurable. A formula, for example, can be distinguished from the transformations being applied to that formula and therefore the component can choose to display the transformations or hiding them.

In the field of human-computer-interaction, the situation is not so clear. A major task will be identifying objects involved in the dialogue between the user and the system and developing a language for describing their relations. This is a precondition for making the interactive behaviour of the system configurable as well. The scope of this task goes well beyond the scope of this thesis and will require further research involving not only computer science but also in the fields of human communication and didactics. While this cannot be accomplished with this thesis, the software component will be designed based on the assumption that such objects of interaction exist and the details of the objects will be filled in at a later time. A number of such objects, called Dialog Atoms, is described in [Neu02b] and will be incorporated into the prototype.

Finally, to deal with the individuality of users, an object abstracting the user's abilities, knowledge and preferences, called a User Model [LJS99], will be incorporated in the design.

The goal is the design of a component covering all these requirements, the implementation of the respective objects and the integration into the ISAC system. While the design will be kept as open and universal as possible, the implementation will comply with the specific requirements of ISAC.

0.2.2 Position of this Thesis within the ISAC Project

This thesis will deal specifically with guided user interaction while doing a calculation, the respective component will be called the Dialog Guide. Other parts of the system include:

- The already-implemented Math-Engine
• A bridge interfacing the SML-world of the Math Engine with the Java-world of the rest of the system

• A subsystem dealing with the Knowledge Base, comprising the Knowledge Base itself, a Knowledge Browser and the corresponding user interfaces

• A user-interface component for calculations, called the Worksheet

• Session and rights management

See also section 6.3 for an illustration of ZSAC’s system architecture. The Dialog Guide can be developed rather independent of the Knowledge Base subsystem and the Math Engine is abstracted by the bridging component. Integration with the ZSAC system will be designed with clear-cut interfaces so the system can run without the Dialog Guide or with the Dialog Guide replaced with a dumb stub component. Close cooperation with the team is required for interfacing with the Worksheet component and the bridge to the Math Engine. This will include the design of objects required throughout the entire system, such as mathematical objects or asynchronous events.

With the Math Engine already implemented, but little documentation or requirements analysis about other parts of the system available, requirements analysis in cooperation with other members of the project will be one of the first steps taken.

0.2.3 Structure of this Thesis

After an introduction, part II is dedicated to requirements analysis based on use cases in chapter 1 and a collection of derived user requirements in chapters 2 and 3.

Part III discusses design considerations, chapters 4 and 5 concentrating on user interaction, chapter 6 on overall system design and chapter 7 on detailed design of the Dialog Guide component.

Part IV highlights details of the prototype implementation.

Finally, part V summarises the work done in chapter 10 and gives an outlook on possible future developments in chapter 11.

Notational Conventions

Terms having a special meaning in ZSAC are Capitalised. Code or identifiers from the implementation are given in typewriter font.

Due to their short-lived nature, resources from the WWW are mentioned in footnotes rather than in the bibliography.
0.3 Introduction to Terms Used throughout this Thesis

In this section, terms describing basic concepts of TSAC are explained. Please refer to the example in appendix A for a practical view and to [Neu01] for in-depth explanations.

0.3.1 System Components of TSAC

Math Engine

The Math Engine takes care of all mathematical data analysis and processing. It is the only component aware of mathematical meaning of data. Several other components understand the structure of calculations, but are limited in their interpretation of data they are processing. This is due to the complexity of mathematics and the configurability of TSAC’s concept of mathematics. While future formula editors might share the Kernel’s syntactic knowledge, present components cannot check a formula for syntactic correctness or even distinguish a literal number from a variable name.

Knowledge Base

The Knowledge Base stores TSAC’s knowledge about mathematics. This knowledge is used by the Math Engine for calculations, but a human-readable copy of the same knowledge is intended to be accessible to the user for reference or for learning purposes.

Kernel

The Math Engine and the Knowledge Base together are referred to as the Kernel.

Knowledge Browser

The Knowledge Browser is the tool for searching and displaying knowledge from the Knowledge Base.

Worksheet

The Worksheet is an abstraction of the traditional paper-and-pencil interface to calculations. It is the user’s tool for viewing and manipulating calculations.
Dialog
From the user’s point of view, the Dialog is the didactic part of the system. It manages the user’s interaction with the system in a conceptual way. It makes decisions about which data to present and which options to offer to the user depending on the current learning situation.

User Model
The User Model is an abstraction of ISAC’s experience with an individual user based on the history of interaction. In particular, the User Model keeps records of knowledge viewed or used by the user, the Dialog Atoms used and the user’s success in this process. Based on these data, the Dialog makes assumptions about the user’s familiarity with mathematical concepts and guides him accordingly.

0.3.2 Abstraction Levels of Dialog Behaviour

Dialog Atom
Dialog Atoms are basic building blocks of user interaction when doing calculations in a learning context. Examples for Dialog Atoms are having the user calculate a step in a calculation, calculating a step automatically or asking questions.

Dialog Strategy
Dialog Strategies are sequences of Dialog Atoms guiding a user through the learning process, distinguished by different levels of Dialog Activity and different didactic backgrounds.

Dialog Activity
Dialog Activity denotes a measure for the system’s responsibility for the work being done. While in calculating a step automatically, ISAC takes all responsibility for the result, having the user enter the step leaves most of the responsibility, challenge and learning effect with the user.

0.3.3 Key Elements in Calculations

Formula
A Formula is a formal text describing mathematical objects. From a practical point of view, a Formula in ISAC is very much what you expect it to be.
Tactic

Generally speaking, a tactic is a step to be taken to transform a certain situation into another one. When doing calculations, \textit{\textsc{Is4C}} uses Tactics to transform one Formula into another Formula according to the rules of the Theory used, sometimes adding assumptions about the possible values of variables. Please note that Tactics resemble calculation rules, but on a different level of abstraction. Examples of \textit{\textsc{Is4C}}’s Tactics are \textit{rewriting a formula} or \textit{starting a sub-calculation}. Most calculation rules we learn in school, such as the law of distributivity, are parameters to the Tactic of \textit{rewriting a formula} using that specific rule. In \textit{\textsc{Is4C}}, whether applying the law of distributivity or the law of commutativity, the Tactic itself is merely \textit{rewriting a formula}, irrespective of the rule involved. The term Tactic stems from theorem proving, where a tactic maps one proof state to another, whereas \textit{\textsc{Is4C}}’s Tactics map calculation states one onto another. Refer to appendix B for an exhaustive list of \textit{\textsc{Is4C}}’s tactics.

0.3.4 \textit{\textsc{Is4C}}’s Concept of Calculations

Calc Tree

\textit{\textsc{Is4C}} organises calculations in Calculation Trees. The normal flow of a calculation alternates Formulas and Tactics transforming one Formula into the next one. This list-like structure branches where independent calculations have to be made to obtain intermediate results, such as finding the roots of the first derivative of a function to solve an optimisation problem or to obtain several cases in a case-split.

Subproblem

Independent calculations to obtain intermediate results are called Subproblems. Independent means that they are fully specified by a Calc Head and could be processed as calculations of their own.

Calc Head

A Calculation Header details all inputs, constraints and desired results necessary to start a calculation. Every Calc Tree has at least one Calc Head at its start, but may contain additional Calc Heads where Subproblems are started.

Specifying Phase

The process of specifying all information necessary to start a calculation is called the Specifying Phase. Apart from the necessity of providing the
information to the system, the Specifying Phase trains gaining insight into
the nature of a mathematical problem before searching for a solution.

**Solving Phase**

In the Solving Phase, Formulas are repeatedly transformed into other Form-
ulas by applying Tactics until eventually a solution is reached.

### 0.3.5 Mathematical Knowledge

Mathematical knowledge stored in \( \mathcal{L}_4 \)'s Knowledge Base is categorised as
follows:

**Problem**

A Problem describes the abstract common properties of a class of examples.
A Problem describes the goal - what are we looking for - and which infor-
mation has to be given to find a solution. Finding the roots of a polynomial
equation is an example of a Problem, finding the maximum of a function is
another. Please note that when using the term \textit{Problem} in the context of
\( \mathcal{L}_4 \), we are not talking about a specific problem (e.g. solving a specific
equation) but a class of problems (e.g. solving root-equations). Problems
are contained in a hierarchy \( \mathcal{L}_4 \) uses for mechanical refinement of a Prob-
lem (e.g. a type of equation, say 'univariate equation') to a more appropriate
Problem (say 'rational univariate equation').

**Theory**

Theories contain the deductive part of knowledge such as all theorems derived
from the axioms and the rules of inference for a given realm of numbers.

**Method**

A Method is an instruction which Tactics to use in which sequence in order
to find a solution to a Problem.

### 0.3.6 Describing a Calculation Task to \( \mathcal{L}_4 \)

**Example**

An Example is very much the same as examples found in traditional math
textbooks. In addition to an Example’s Description, \( \mathcal{L}_4 \) stores one or more
Formalizations and one or more Specifications prepared by the author of the
Example to aid in the process of guiding the student during the Specifying
Phase. Examples are organized into example collections.
**Formalization**

A Formalization is a description of a real-world problem, which is normally stated in natural language, expressed in purely mathematical terms. A Formalization could consist of a formula, a indication of the solution sought and optionally additional formulas indicating constraints applying to valid solutions.

**Model**

A Model is an abstract Problem instantiated by concrete data as obtained from the Formalization of an Example or from user input.

**Specification**

Given a Formalization, a Specification specifies the Problem to be solved, the Theory where to search for solutions and a Method to be used in solving the example. If a student recognises two examples as being "essentially the same thing to calculate, only the numbers are different", it is the Specifications he recognised as being "the same" and the Models as being "different".

**Description**

In the context of ZSAC, the term Description applies only to descriptions of Examples, comprising text, formulas and figures, such a given in traditional math textbooks.

**Explanation**

An Explanation is data, such as text or figures, giving additional information about an item in the Knowledge Base. Imagine a Formula being transformed into the next Formula by rewriting using the law of distributivity. An Explanation linked to the Tactic "rewriting using the law of distributivity" could be a short text or a link into the Example Collection illustrating the nature of distributivity. The Explanation is not used by the system in the process of calculating a solution, it rather serves informational and didactic purposes for the user.
Part II

Requirements Analysis
The Dialog Guide is embedded in a complex and heterogeneous system. At the start of the design phase, barely any specifications were available, apart from the abstract idea described in [Neu01] and the implemented Math Engine and first-generation prototype for reference. Requirements analysis was led by ideas from [Jac92] and [Coc00] without going into too fine detail.

This part documents the requirements analysis process. In chapter 1, selected use cases are listed. Requirements of the system and particularly the Dialog Guide as a starting point for design are derived in chapters 2 and 3. The contents are based on the work-in-progress-style internal documentation of the ZS4C project [GKN02d, GKN02e, GKN02c], analysis of the first-generation prototype and internal communication of the ZS4C team.

Wherever appropriate, explanations of design decisions will refer to the use cases and requirements listed here. On the other hand, many requirements will not be met at the early stage of development of ZS4C covered in this thesis.

Please see section 0.3 in the introduction for definitions of terms or [Neu01] for in-depth explanations.
Chapter 1

Use Cases

In the sections below, all actions conceived for ISAC and relevant to the design of the Dialog Guide component are listed.

Some of these actions may be blocked depending on the specific example being solved, the user's preferences or restrictions in the user’s rights. In this case, nothing happens, as if the action had not been requested.

The specific way how actions are requested is not specified. At the discretion of the designers of the GUI, this could happen by selecting from a menu bar or a context menu, by clicking buttons or by keyboard shortcuts.

1.1 Initialising the Dialog

UC 1.1.1 Identifying the User

The user has to identify himself so ISAC can retrieve his personal preferences, his performance history and restrictions in his access rights.

On success: The user is identified and continues by choosing a starting point as described in 1.1.1.

On failure: The identification process starts over again.

UC 1.1.2 Contacting a Math Engine

The Math Engine does not run on the same machine as the Dialog Guide. The network address of the Math Engine has to be entered manually or read from a configuration file. This could happen before or after the user login.

On success: Continue with user identification (UC 1.1.1) or choosing a starting point (1.1.1).

On failure: Failure is critical, calculations cannot be done. The program has to be aborted. As an alternative, the user can be prompted for a network address even if he would not enter it manually under normal circumstances.
1.1.1 Choosing a Starting Point

UC 1.1.3 Doing an Example from an Example Collection

The user can choose to do an example from a collection of prepared examples.

On success: The complete Model and Specification of the example are known to IS4C, so IS4C can offer help filling in the fields of Model and Specification.

UC 1.1.4 Starting a Calculation from Scratch

The user can choose to start an example from scratch, entering the Model and Specification manually into an initially empty form.

On success: The user is free to do calculations not contained in the example collection. On the other hand, in this case IS4C cannot offer any help in completing the Model and Specification, only checks for completeness and consistency. See also UC 1.1.3 and UC 1.2.2.

UC 1.1.5 Starting a Calculation from a Problem

The user can choose to start an example beginning from a Problem, entering the Model and Specification manually. After choosing a Problem by browsing the Problem hierarchy, modelling and specifying will start with a form which is initially empty except for the Problem chosen.

In the future, IS4C could offer extra help by offering a choice of Methods known to match the Problem or by help texts specific to the Problem.

On success: The user is free to do calculations not contained in the example collection. On the other hand, in this case IS4C cannot offer any help in completing the Model and Specification, only checks for completeness and consistency. See also UC 1.1.3 and UC 1.2.2.

1.2 Initialising a Calculation

1.2.1 Modeling and Specifying

UC 1.2.1 Entering the Model and Specification

A complete Model and Specification comprise all mathematical data necessary for starting a calculation. The user enters the data into a form with fields labelled accordingly.
1.2.2 Obtaining Help from T5AC

UC 1.2.2 Having T5AC Check the Model and Specification

The user can have the Model and Specification checked for consistency and completeness. On each of the fields in the Model and Specification, feedback about syntax, completeness and consistency is returned. No fields of the Model or Specification will be changed.

On success: The Model and Specification are complete and correct and ready for starting a calculation as in 1.2.4. All fields are marked as "ok".

On failure: Fields offending criteria for completeness and correctness are marked and offer feedback on their status. The user can continue modelling or specifying. T5AC does not correct incorrect items; however, the user can remove the items marked as incorrect and let T5AC provide the correction as described in other use cases in this section.

UC 1.2.3 Having T5AC Complete the Specification

When doing a prepared example, the user can have the Specification completed by T5AC with data from the example collection.

On success: The Specification is complete and correct and ready for starting a calculation as in 1.2.4.

On failure: If the example was started from scratch (UC 1.1.4), the data entered so far might not suffice to fill in all fields or find the best match. The user can still continue specifying manually.

UC 1.2.4 Having T5AC Complete the Model

When doing a prepared example, the user can have the Model (fields Given, Find, Relate) completed by T5AC with data from the example collection.

On success: The Model is complete and correct.

On failure: If the example was started from scratch (UC 1.1.4), the data entered so far might not suffice to fill in all fields or find the best match. The user can still continue modelling manually.

UC 1.2.5 Having T5AC Complete One Field of the Model

When doing a prepared example, the user can have one field of the Model completed by T5AC with data from the example collection.
On success: The selected field of the Model is complete and correct.

On failure: If the example was started from scratch (UC 1.1.4), the data entered so far might not suffice to fill in the field or find the best match. The user can still continue modelling manually.

UC 1.2.6 Having ZSAC Complete One Field of the Specification

The user can have one field of the Specification (fields Theory, Problem, Method) completed by ZSAC from data entered so far.

On success: The selected field of the Specification is complete and correct.

On failure: If the example was started from scratch (UC 1.1.4), the data entered so far might not suffice to fill in the field or find the best match. The user can still continue specifying manually.

UC 1.2.7 Refining the Problem Manually

Specifying a Problem matching the Model as closely as possible is crucial for applying the optimal Method. The user can choose to browse the hierarchy of known Problems (as described in [Neu01]) to pick a Problem. The Problem picked is checked for consistency with data entered so far.

On success: The Problem matches other data entered so far. The Problem field is marked as “ok”. Other fields are marked as “ok” or “superfluous”.

On failure: The Problem field is marked as in UC 1.2.2, offering feedback on its status. The user can continue modelling or specifying.

UC 1.2.8 Having ZSAC Refine the Problem Automatically

The user can have ZSAC refine the Problem to a best match of the data entered into the Model so far.

On success: The Problem matches other data entered so far.

1.2.3 Alternate Views on Model and Specification

UC 1.2.9 Entering a Calculation Like into a Traditional Algebra-System

As an alternative to entering all fields of the Model and Specification manually, the user can choose a form-based input format resembling traditional algebra systems, like:

\[
\begin{align*}
\text{solve: } & x^3 + x^2 + x + 1 = 0 \quad \text{solve for: } [x] \\
The label “solve” on the form implies the Problem to be solved and thus the interpretation of the input and the contents of the Model and Specification.
\end{align*}
\]
**On success:** If the data entered into the form can be interpreted matching the semantics of the form, the Model and Specification are completed by 7S4C.

**On failure:** The user can continue modelling or specifying.

**UC 1.2.10 Changing the View on the Model and Specification**

The user can choose to display and edit the Model and Specification in detail or in a simplified view resembling traditional algebra systems like in UC 1.2.9.

**On success:** The Model and Specification are displayed in the desired view.

**On failure:** If the data in the Model and Specification cannot be mapped into one of the available forms, the view remains unchanged.

**1.2.4 Starting the Calculation**

**UC 1.2.11 Starting Interactive Calculation**

Having entered a Model and a Specification, the user can start solving the example interactively.

**On success:** In a worksheet window, the first formula in is displayed. Calculation continues as in 1.3.

**On failure:** If the Model and Specification are not error-free, complete and consistent, modelling and specifying continues. Status information on the fields of the Model and Specification is updated.

**UC 1.2.12 Starting Automatic Calculation**

Having entered a Model and a Specification, the user can request the calculation being done by 7S4C.

**On success:** In a worksheet window, the final result is displayed. The user can still analyse and modify the calculation as described in 1.3.

**On failure:** If the Model and Specification are not error-free, complete and consistent, modelling and specifying continues. Status information on the fields of the Model and Specification is updated.
1.3 Calculating a Result

1.3.1 Moving the Active Formula

Most actions during the Solving Phase refer to a specific point in the course of calculation, the currently active formula. All editing takes place at the currently active formula or the Tactic being applied to it and calculation continues from that point. Most of the time, the last formula entered or calculated will be the active one.

UC 1.3.1 Moving the Active Formula

In order to edit a part of the calculation, the user has to move the active formula to the desired location first.

On success: The referenced formula becomes the new active formula. Actions refer to this position now.

1.3.2 Taking Single Steps interactively

UC 1.3.2 Entering a Tactic Manually

The user can enter the Tactic, i.e. step of calculation, to be applied to the active formula manually. The Tactic is recognised by its name. The Tactic is not applied immediately, but will be applied the next time a step in a calculation (1.3.3) is requested.

On success: The Tactic is recorded and will be applied to the active formula the next time a step of calculation is requested.

On failure: If the Tactic entered is not known to $\text{LS4C}$ or the Tactic is not applicable to the currently active formula, the user is notified but nothing else happens.

UC 1.3.3 Picking a Tactic from a List of Known Tactics

The user can pick the Tactic to be applied to the active formula from a list of all Tactics known to $\text{LS4C}$. The Tactic is not applied immediately, but will be applied the next time a step in a calculation (1.3.3) is requested.

On success: The Tactic is recorded and will be applied to the active formula the next time a step of calculation is requested.

On failure: If the Tactic is not applicable to the currently active formula, the user is notified but nothing else happens.

UC 1.3.4 Picking a Tactic from a List of Applicable Tactics
The user can pick the Tactic to be applied to the active formula from a list of Tactics applicable to the current situation. This list is prepared by $\mathcal{TSAC}$. The Tactic is not applied immediately, but will be applied the next time a step in a calculation (1.3.3) is requested.

**On success:** The Tactic is recorded and will be applied to the active formula the next time a step of calculation is requested.

**UC 1.3.5 Entering a Formula Manually**

The user can enter a proposal for the next formula in the course of calculation manually.

If a Tactic has been chosen for this step, the input must match the result of this Tactic.

If no Tactic has been chosen, there must be a sequence of Tactics which derive the formula entered from the currently active formula.

**On success:** The entered formula is entered into the calculation and becomes the currently active formula. Additional steps may be added automatically to reflect the derivation of the formula entered.

**On failure:** The formula is rejected if it is not the result of the Tactic chosen for this step or - if no Tactic has been chosen - no derivation confirming the correctness of the formula can be found. A new formula can be entered.

**UC 1.3.6 Editing and Replacing a Formula**

The user can edit and modify the active formula, which need not be the last formula in the calculation. Changing a formula potentially invalidates all subsequent steps, which are removed from the calculation. There must exist a sequence of Tactics which derive the formula entered from the preceding formula.

**On success:** The edited formula is entered into the calculation. The display is updated as to reflect the derivation of the entered formula.

**On failure:** The formula is rejected if no derivation confirming the correctness of the formula can be found. A new formula can be entered.

**1.3.3 Automatic Calculation**

**UC 1.3.7 Having $\mathcal{TSAC}$ Propose the Next Tactic**

In addition to choosing a Tactic manually (UC 1.3.2), $\mathcal{TSAC}$ can propose the next Tactic based on its knowledge of the Method solving the current Problem.
On success: A Tactic for the next step has been chosen.

UC 1.3.8 Having \textit{\textsc{zsac}} Calculate the Next Formula

In addition to entering a formula manually (UC 1.3.5), \textit{\textsc{zsac}} can calculate the next formula by applying the chosen Tactic to the active formula. If no Tactic has been chosen, \textit{\textsc{zsac}} uses its knowledge about the Method solving the current Problem to choose a Tactic (UC 1.3.7)

On success: The entered formula is entered into the calculation and becomes the currently active formula.

UC 1.3.9 Having \textit{\textsc{zsac}} Calculate until the End of the Current Subproblem is Reached

\textit{\textsc{zsac}} can calculate any steps needed to finish the current subproblem automatically.

On success: Intermediate steps have been added to the calculation to solve the current subproblem. The result of the subproblem becomes the currently active formula.

UC 1.3.10 Having \textit{\textsc{zsac}} Calculate until a Final Result is Reached

\textit{\textsc{zsac}} can calculate any steps needed to finish the current calculation automatically.

On success: Intermediate steps have been added to the calculation to reach a final result. The result becomes the currently active formula.

1.3.4 Showing and Hiding Data

UC 1.3.11 Hiding a Category of Information

The user can choose not to see certain categories of information (e.g. Tactic, Assumptions). In any case, formulas cannot be hidden and remain displayed.

On success: The respective category of information is removed from the worksheet.

On failure: The worksheet remains unchanged.

UC 1.3.12 Showing a Category of Information

The user can choose to see categories of information currently hidden.
On success: The respective category of information is displayed on the worksheet.

On failure: The worksheet remains unchanged.

UC 1.3.13 Choosing the Displayed Nesting Depth of Details

For the sake of better overview, the user can choose the depth of nested subproblems or subcalculations which are displayed. Hidden details are indicated by symbols to facilitate display on demand. If the currently active formula is below the displayed level of detail, it is moved up to the nearest visible formula.

On success: The respective nesting levels are removed from or added to the worksheet. Symbols indicate hidden levels of detail. After hiding levels, the active formula may have moved.

On failure: The worksheet remains unchanged.

UC 1.3.14 Hiding Parts of the Calculation

For the sake of better overview, the user can choose to hide several steps of the calculation. Hidden steps are indicated by symbols to facilitate display on demand. If the currently active formula has been hidden, it is moved up to the nearest visible formula.

On success: The respective steps are removed from the worksheet. Symbols indicate hidden steps. After hiding steps, the active formula may have moved.

On failure: The worksheet remains unchanged.

UC 1.3.15 Hiding Tactics on Behalf of ZSAC

ZSAC can decide to hide steps of the calculation from the user. Such decisions can be based on records of the user’s experience with specific Tactics or on the user’s preferences.

On success: The respective steps are not displayed on the worksheet. Symbols indicate hidden steps.

UC 1.3.16 Showing Hidden Parts of the Calculation

Parts of the calculation indicating hidden steps can be expanded, adding one nesting level of detail at a time. It does not matter whether the steps have been selected for hiding (UC 1.3.14) or have been hidden because of their nesting depth (UC 1.3.13) or have been hidden by ZSAC (UC 1.3.15) in the first place.

On success: The respective steps are added to the worksheet.

On failure: The worksheet remains unchanged.
1.3.5 Obtaining Help and Extra Information

UC 1.3.17 Displaying the Assumptions Holding at a Specific Point in the Calculation

The user can have the Assumptions (e.g. restrictions on possible values for a variable) holding at a specific point in the calculation displayed. Different from actions changing the calculation, the user need not move the active formula to the spot of interest.

On success: The Assumptions holding at the referenced formula are displayed.

UC 1.3.18 Displaying the Origin of Assumptions

The user can have the origin of an Assumption, that is the Tactic creating the assumption, indicated on the worksheet.

On success: The Tactic where the assumption originated is indicated.

UC 1.3.19 Getting Background Information from the Knowledge Base

Especially with Tactics, extra background information may be available from ZSAC’s Knowledge Base. In an extra window, ZSAC’s Knowledge Base can be browsed, with information on the current item as a starting point.

On success: A browser window is opened showing information on the current item.

On failure: If no information matching the current item can be found, browsing starts from the root of ZSAC’s Knowledge Base or from information on the current Problem.
Chapter 2

Selected Requirements of the \textit{ISAC} System

In this chapter, requirements for the design of the overall system, not necessarily specific but pertaining to the Dialog Guide will be listed.

2.1 Users of \textit{ISAC}

**UR 2.1.1** \textit{ISAC} is a multi-user system

**UR 2.1.2** The users access \textit{ISAC} via internet.

The computing resources needed to run such a complex application exceed the resources presently available to the average user. Moreover, organisation of centralised courses and curricula suggests separation of application and user-interface. An additional requirement is to keep expenses and effort for the average user at a minimum, in terms of computing power needed and installation effort. Ideally, a standard web browser would suffice.

**UR 2.1.3** \textit{ISAC}’s data storage supports simultaneous access

Data storage has to support locking and versioning.

**UR 2.1.4** Users access \textit{ISAC} with different roles

Several possible roles when accessing \textit{ISAC} dictate different rights and access to different modules of \textit{ISAC}.

Visitor (Besucher): occasionally drops into an \textit{ISAC}-site and browses the respective mathematical knowledge base and the example collection. A Visitor may calculate examples.
Learner (Lernender): uses ZS4C for learning and exercising, i.e. primarily calculates examples in the example collection by use of the math knowledge base. As a member of courses the learner is called a student.

Math Author (Mathematik-Autor): is an expert in computer mathematics who adapts and extends the mathematics knowledge base.

Dialog Author (Dialog-Autor): is an expert in learning theory who adapts and extends the Dialog Guide.

Course Administrator (Kurs-Administrator): is a person administering the use of ZS4C for learning within a group of learners.

Course Designer (Kurs-Designer): adapts and extends the example collection which can be solved by a given math knowledge base, and adds explanations to items in the knowledge base. These tasks do not require special knowledge in computer mathematics.

Administrator (Administrator): this role combines the system administrator installing the software, and the person who implements the overall design of an ZS4C-site (introductory pages etc.) on behalf of the site-owner.

**UR 2.1.5 Learners can be grouped into courses.**

There are groups of learners in order to support the administration of courses. The membership w.r.t. these groups determines the selections of examples in the example collection (UR 3.5.2), the selection of explanations in the Knowledge Base (UR 3.5.3) and the initial setting of the Dialog Guide as captured in UR 3.3.4 and UR 3.3.3.

**UR 2.1.6 One learner may be member in different groups**

One learner may be member in different groups but the settings for a session depend on exactly one group. This implies that multiple memberships have to be resolved at login time.

### 2.2 Calculations and Data Involved

**UR 2.2.1 ZS4C’s Math Engine is given as an already-implemented module.**

The Math Engine is already implemented in the same program language as Isabelle, SML. The Math Engine is described in [Nen04b]. Many of the following requirements may seem too specific for an early stage in development. This is in part due to the Math Engine being already implemented and its interfaces being already known.
UR 2.2.2 ĪŚ4C’s Knowledge Base stores 4 kinds of mathematical knowledge

īŚ4C’s Knowledge Base stores Theories, Problems, Methods, Examples as explained in section 0.3.5 in the introduction. See [Neu01] for a detailed explanation.

UR 2.2.3 A Formalization is a list of formulas.

The formulas are organised into groups called Find, Given, Where and Relate.

UR 2.2.4 A Specification is a triple of Theory, Problem and Method.

UR 2.2.5 Each example carries a list of pairs of a Formalization and a Specification respectively.

The pairs of Formalization and Specification are used for user guidance while specifying a calculation and are remain to users in a learning situation.

UR 2.2.6 Explanatory data can be embedded into examples

An example can be described by verbal text, by formulas, by figures and possibly by movies. Additionally, each example contains data hidden from the visitor and the learner.

UR 2.2.7 An example may contain Error Schemes

An Error Scheme modelling typical errors and providing explanations and specific user guidance for resolving the errors may be stored with an example.

UR 2.2.8 A Calculation undergoes 2 Phases

A calculation undergoes two phases: the Specification Phase and the Solving Phase, where the latter may contain these phases recursively (see the demonstration example in appendix A).

UR 2.2.9 Calculations are created interactively in steps.

A step is initiated by the input of a learner: input of a Tactic, of a formula or of a request that ĪŚ4C take over the calculation. A Tactic is applied to a formula and generates another (the derived) formula.

UR 2.2.10 In the Solving Phase, every formula is justified by a Tactic.

UR 2.2.11 ĪŚ4C uses Tactics as listed in appendix B
UR 2.2.12 A calculation has a tree-like structure

UR 2.2.13 \textit{TSAC} guarantees correct results.

\textit{TSAC}'s display of a Calculation reflects the state of the Math Engine. The Math Engine does not produce any inconsistent state of calculation. Thus everything displayed by \textit{TSAC} (apart from items being edited by the user right now) is proven to be consistent with the Specification and the Knowledge Base. User input will be accepted only if it can be proven to be correct by a check with the underlying Math Engine.

UR 2.2.14 A Tactic may contain Error Schemes

UR 2.2.15 There are fill-in patterns for Tactics.

In addition to a Tactic being applied manually by the user or automatically by \textit{TSAC}, a Tactic can be presented with parts left blank to be filled in by the user.

UR 2.2.16 There are fill-in patterns for items of the Model.

2.3 Miscellaneous

UR 2.3.1 \textit{TSAC} supports internationalization.

UR 2.3.2 \textit{TSAC} supports cross-linking calculations and knowledge

Calculations may contain links into the Knowledge Base, e.g. to the definitions or proofs of Tactics applied in the course of calculation or into underlying theories. The Knowledge Base may contain links to examples illustrating theoretical concepts.

UR 2.3.3 \textit{TSAC} is open to data exchange with other tools.

To facilitate interfacing with other tools, \textit{TSAC}'s objects support being exported to and imported from open data formats. XML formats are preferred.

UR 2.3.4 Several users can watch the progress of one calculation.

Several users may watch the progress of a calculation, but there is exactly one user controlling the calculation and taking actions. This can be useful for instruction situations, especially teleteaching.

UR 2.3.5 \textit{TSAC} supports export of its display for editing with other tools

Editing in \textit{TSAC} is limited by UR 2.2.13. For publishing purposes, \textit{TSAC}'s display can be exported for editing with standard publishing tools.
Chapter 3

Requirements Specific to the Dialog Guide

In this chapter, requirements specific to the design the Dialog Guide will be listed. Many of these requirements affect the Math Engine as well, which is already implemented.

UR 3.0.6 TSAC provides User Guidance if the problem to be solved has been specified.

A problem known to the system implies that the system knows a method to solve the problem. Thus TSAC can solve the problem automatically or propose the next step to be done.

UR 3.0.7 TSAC can assist in calculating examples unknown to the system

Without knowing the Problem, TSAC cannot propose steps or solve the Problem automatically. Still, TSAC can apply Tactics chosen by the user to a formula. With a theory specified, TSAC can check formulas input by the user for correctness and consistency with previous steps in the calculation. At all times, TSAC ensures the calculation displayed is consistent and error-free as detailed in UR 2.2.13.

UR 3.0.8 A worksheet can be opened with pre-defined examples from an example collection

UR 3.0.9 A worksheet can be opened with pre-defined examples from the Knowledge Base

This is to illustrate knowledge from the Knowledge Base by typical examples stored with the knowledge.
3.1 User Guidance

UR 3.1.1 T$\S$AC can offer a list of actions known to the system.

UR 3.1.2 T$\S$AC can offer a list of actions applicable to the current situation.

UR 3.1.3 T$\S$AC can propose the next action to be taken.

UR 3.1.4 T$\S$AC can do one or more steps automatically.

UR 3.1.5 The user gets immediate feedback on data entered into the Model.

UR 3.1.6 T$\S$AC can match a problem to a Model.

UR 3.1.7 T$\S$AC can refine a problem to match a Model more closely.

UR 3.1.8 T$\S$AC’s use of its Knowledge Base can be watched by the user.

The user can look up the parts of the Knowledge Base currently used by T$\S$AC. This includes showing

• Tactics used in the calculation in their context in the Knowledge Base.

• Methods being applied to solve the current Problem with indication of the Tactic being currently applied.

• Problems currently being solved in their context in the hierarchy of Problems. This particularly helpful just before and after having T$\S$AC refine a Problem.

UR 3.1.9 On request, T$\S$AC provides additional information on parts of the calculation.
Additional information in a calculation can be provided at any time on request of the learner. This feature comprises more detailed views onto the proofstate, as well as explanations according to the following table:

<table>
<thead>
<tr>
<th>'detail' on</th>
<th>element</th>
<th>yields</th>
</tr>
</thead>
<tbody>
<tr>
<td>whole formula</td>
<td>//</td>
<td>generating tactic</td>
</tr>
<tr>
<td>whole formula</td>
<td>//</td>
<td>associated assumptions</td>
</tr>
<tr>
<td>formula</td>
<td>function-constant</td>
<td>definition in the theory</td>
</tr>
<tr>
<td>formula</td>
<td>floatingpoint-no</td>
<td>precision of this no</td>
</tr>
<tr>
<td>evaluated predicate</td>
<td>//</td>
<td>derivation</td>
</tr>
<tr>
<td>evaluated assumption</td>
<td>//</td>
<td>derivation</td>
</tr>
<tr>
<td>tactic</td>
<td>theorem</td>
<td>theorem instantiated</td>
</tr>
<tr>
<td>theorem instantiated</td>
<td>//</td>
<td>animation of matching</td>
</tr>
<tr>
<td>tactic</td>
<td>ruleset</td>
<td>intermediate steps</td>
</tr>
<tr>
<td>tactic</td>
<td>subproblem</td>
<td>intermediate steps</td>
</tr>
<tr>
<td>error 'not matching'</td>
<td>//</td>
<td>animation of matching</td>
</tr>
<tr>
<td>model</td>
<td>theory</td>
<td>file of the respective theory</td>
</tr>
<tr>
<td>model</td>
<td>problem</td>
<td>model instantiating this problem</td>
</tr>
<tr>
<td>model</td>
<td>problem</td>
<td>inherited assumptions</td>
</tr>
<tr>
<td>model</td>
<td>method</td>
<td>model instantiating the guard</td>
</tr>
<tr>
<td>model</td>
<td>method</td>
<td>script of the respective method</td>
</tr>
</tbody>
</table>

**UR 3.1.10 With items in the Knowledge base, Descriptions can be stored.**

Every item in the Knowledge Base can provide a description, illustrating its meaning, giving theoretical background information, referencing related topics or giving examples of use.

**UR 3.1.11 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.

### 3.2 User Profiling

**UR 3.2.1 *$\mathcal{LSAC}$ records examples done by the user**

*$\mathcal{LSAC}$ 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 3.2.2 *$\mathcal{LSAC}$ records items in the Knowledge Base viewed by the user.**
This information can be used to base the Dialog Guide's behaviour on information supposedly known to the user.

**UR 3.2.3** *ISAC records the user's success and errors.*

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

**UR 3.2.4** *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 3.2.5** *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.

### 3.3 Flexible Dialog Behaviour

**UR 3.3.1** *ISAC's Dialog behaviour is constructed from Dialog Atoms*

We hope that it is possible to develop a language which allows to define Dialog Patterns as combinations of Dialog Atoms already implemented and Dialog Strategies sequencing these atoms. By means of such a language learning strategies could be described, and this description could be interpreted in reaction to a dynamic dialog state and according to a knowledge profile.

To do such 'dialog programming' is considered a comprehensive task, which in general exceeds the knowledge of a course designer or a course admin. On the other hand, a course admin can be expected to associate courses with dialog profiles, and a course designer can be expected to select Dialog Strategies within process of time in a course.

**UR 3.3.2** *The Dialog's behaviour can be configured.*

The Dialog's behaviour in terms of Dialog Atoms to be used can be preset by the course designer and the user to match the requirements of the situation in style and complexity. The probability of asking the user a question is an example of such a preset. DIALOG:ADAPT:history

**UR 3.3.3** *The number of calculation steps taken at a time can be configured.*

This extends to taking several steps at a time and doing whole rule sets or subproblems in one step.
3.4 Adaptation to Individual Users

3.4.1 The activity of the Dialog Guide adapts to the learner.

In a learning situation, active participation of the student is one key to acquiring and consolidating knowledge and skills. The Dialog Guide will support this by asking questions or letting the user decide what to do in the next step. On the other hand, with growing expertise, once thrilling questions become trivial and boring. The Dialog Guide adapts his strategy by dropping challenges the user has already mastered a few times. Whenever possible, the dialog adapts its behaviour, i.e. the choice of Dialog Atoms, to challenge the user without frustrating him. The choice is based on the present and past actions of the individual user.

3.4.2 The Dialog adapts the amount of information displayed to the learning situation.

As with the Dialog’s behaviour, the amount of information displayed adapts to the requirements of the current situation.

3.4.3 The Dialog regards the performance of the user.

The performance is measured by response times, errors, difficulty of examples done, requests into the Knowledge Base and Dialog Activity.

3.4.4 The Dialog regards the knowledge touched by the user in the current session.

3.4.5 The Dialog regards the history of the user.

In addition to the of performance and knowledge touched in the current session, the history of the user’s previous sessions is regarded as well.

3.4.6 The learner can override the Dialog behaviour chosen by the system.

This includes the settings for dialog activity, stepwidth and display filtering rules.
3.5 Restrictions

Restrictions will apply when using ZSAC as a learning or tutoring tool, especially during exams. When being used as a calculation tool, restrictions are relaxed to gain access to the full power of the Math Engine.

UR 3.5.1 Restrictions are individual to a user or group.

UR 3.5.2 Groups of examples may be invisible

Parts of the example collection may be inaccessible for specific groups of learners.

UR 3.5.3 Access to items in the Knowledge Base can be restricted.

UR 3.5.4 The amount of User Guidance may be restricted.

UR 3.5.5 The use of Dialog Patterns may be restricted.

UR 3.5.6 Restrictions may be overridden.

Depending on the settings provided by the course-designer and the user’s access rights, some restrictions may be overridden by the user. Overriding e.g. allows to look at explanations and examples for other courses.

UR 3.5.7 Restrictions may apply within time limits.

Some restrictions may apply only within certain time limits, e.g. during an exam or during class. Time limits can be given by start and finish or by their duration.

UR 3.5.8 Restrictions may depend on the user’s learning progress.

Some restrictions may apply until certain examples have been solved or the user has mastered certain aspects of knowledge.
Part III

Design Considerations
Chapter 4

User Interaction from a Didactic Point of View

In addition to technological issues, developing educational software poses a number of questions about the underlying theory of learning [BP99]. In this chapter, we will explore the most common learning theories and their implications for the design of ISAC.

4.1 Basic Theories about Learning and Teaching

4.1.1 Behaviouristic Learning

Process of learning: Associating stimulus and reaction

Goal of learning: Using the appropriate behaviour in specific situations

Behaviouristic Theory of Learning

Behaviourism founds its theory on the correlation of an observable behaviour with an observable situation or stimulus, disregarding internal mental processes. The process of learning is explained as reinforcing the connection between stimulus and reaction in the case of success and weakening the connection in the case of failure. It is of little importance where the desired behaviour originated in the first place, as long as it is associated with the stimulus and reinforced whenever it occurs in the proper context. Behaviouristic learning is based on drill and repetition. The goal of behaviouristic learning is to change the behaviour toward a more appropriate behaviour.

Behaviouristic Teaching Strategies

- Embed the items to be trained into a well-structured environment
• Make key features of the item to be learned - the stimulus - easily recognisable by associated colours or sounds

• Give hints to produce the desired behaviour

• Give immediate, unambiguous feedback about success or failure

• Exercise over and over again

• Repeat the process until the desired result is achieved

**Advantages and Drawbacks**

+ Once learned, items are applied automatically, thus fast and easily. Most people do not *think* about calculations involving natural numbers in the range up to 10 or 20 - they simply *know* the answer. They have been drilled to know it in school.

+ Being a rigorous, mechanistic process, behaviouristic learning strategies are easily implemented in software.

+ Learning progress can easily be evaluated.

+ The low degree of freedom in the process reduces uncertainties and fear of the new. Simplistic yes-no, good-evil models take the stress out of situations otherwise too complex.

- The low degree of freedom in the process provides little intrinsic motivation and tends to become boring rather soon.

- Based on a mechanistic view, behaviouristic learning does not take individual potentials or weaknesses into account.

- Behaviour is produced independent of insight, which is crucial for advanced mathematical skills.

- Most mathematical problems are too complex for a simple yes-no rating of behaviour.

**jSAC’s Support for Behaviouristic Learning**

• The Math Engine offers immediate feedback even on intermediate steps taken by the learner.

• Several important notions are conceptually separated in *jSAC*, such as the clear separation of formula and tactic, or specifying and solving phase. As such they can be presented in easily distinguishable ways.
• Fill-in patterns for Formulas resulting from the application of Tactics
train automatic application of calculational skills.

• The User Model tracks steps not mastered by the learner, so the Dialog
Guide can repeat them if so desired.

4.1.2 Cognitivistic Learning

Process of learning: Information processing interconnecting new and fa-
miliar knowledge

Goal of learning: Insight into the way knowledge is interconnected and
the how-and-why of doing things

Cognitivistic Theory of Learning

Cognitivism researches the internal processes of acquiring, understanding,
processing and sorting knowledge and tries to develop learning strategies
making use of the internal structures of human reasoning. Cognitivistic
teaching focuses on cognition, insight into complex issues and problem-
solving strategies by supporting the processes mentioned above. While be-
vaviourism teaches facts, cognitivism teaches the interrelations of facts.

Cognitivistic Teaching Strategies

• Explain goals and motivations
• Explain the background of the process
• Describe and justify the single steps of the process
• Emphasise context, not details
• Give examples
• Guide, give role models
• Exercise simulations with choices of options
• Analyse and explain the outcome

Advantages and Drawbacks

+ Insight is emphasised, which is considered essential to complex topics
like mathematics.

+ Gaining insight may be a tedious process, but once gained, insight
reduces the complexity of extensive topics and allows to remember
more than by merely storing facts.

45
+ In addition to just repeating a specific situation, insight enables to deal with new situations which bear structural similarities to already-known situations.

+ Being guided learning, cognitivistic strategies offer the security of standards and coherence.

- Being guided learning, cognitivist strategies do not make full use of individual potentials and abilities.

- Analysing, describing and organising an expert’s insight in order to transport it to the learner is a tedious process.

TSAC’s Support for Cognitivistic Learning

- Every step taken in a calculation is justified by a Tactic.

- The Knowledge Base is visible to the user and can be augmented by additional Explanations and Examples illustrating the concepts.

- During the Solving Phase, the interconnection between Problem and Method is emphasised.

- TSAC’s Methods can be made transparent to the user.

4.1.3 Constructivistic Learning

Process of learning: Exploration and interpretation of the environment

Goal of learning: Building an usable abstraction of relevant environments

Constructivistic Theory of Learning

Constructivism regards learning as the process of building an internal representation of the surrounding environment by self-organisation of mental processes. What we seem to know about the world is not knowledge about real objects but abstractions of experiences we have made with objects surrounding us. Learning is regarded successful if these abstractions enable us to interact with the world surrounding us, regardless of the abstractions’ similarity to reality, which cannot be directly perceived. With our knowledge being abstractions of experiences and learning the adaptation and rework of these abstractions, learning becomes a very individual, autonomous process where notions of right or wrong lose their relevance. Without common definitions of right or wrong, teaching in the conventional sense of the word becomes meaningless. Still, there are strategies to support the individual process of exploration. See [Mon64] for implications for teaching.
Constructivist Teaching Strategies

- Offer interaction instead of presenting facts or guiding the right way
- Offer a complex, but consistent environment to explore
- Offer many different aspects of a topic
- Offer opportunities to experience the topic in many different ways. For extra experience about the making of this thesis mail to alan.kremler@mmm-komm.at.
- Confront with consequences
- Offer help
- Refrain from rating the learner's success

Advantages and Drawbacks

+ Individual and autonomous learning makes the most of an individual’s potentials and previous experiences
+ Holistic learning experiences provide for effective and lasting results
- Constructivist learning environments tend to overstrain many learners and make them feel helpless and disoriented
- It is hard to program a learning environment which is rich and diverse enough to offer sufficient opportunity for exploration

TSAC’s Support for Constructivist Learning

- The Knowledge Base can be browsed and explored by the user.
- Provided a Tactic is applicable to the current situation, Tactics can be applied by the user in arbitrary order, not limited to the order predefined by TSAC’s Methods.
- The user can modify and rework existing calculations.
- The Math Engine offers immediate feedback even on intermediate steps taken by the learner, thus easing the exploration of alternatives.
- Flexible user guidance and adjustable Dialog Activity make it possible to balance the freedom to explore and the comfort of user guidance in a way appropriate to the individual user.
4.2 Modelling Interaction With a Peer

Interaction with a peer mathematician or student can be used as a guideline for designing ISAC’s behaviour. Most students feel more comfortable interacting with a human peer than reading books, giving better and more effective results when interacting with humans. Therefore, the goal is to answer any objection like "When learning with a peer, I can X" with "You can X with ISAC as well".

For the design of the Dialog Guide, this means being structurally prepared for every conceivable interaction in learning situations. Being structurally prepared means being able to moderate the interaction. If we take asking a question as an example, the Dialog Guide has to provide the structures for recognising a question as such, routing the question to a component which can answer it, retrieve the answer and route the answer to the presentation layer.

It is the responsibility of other components to provide the answer (the content) - most probably the Math Engine or the Knowledge Base. It is the responsibility of the Presentation Layer to present the answer in a consistent and optically pleasant way.

Now which are the interactions used by humans in learning situations? The following list is by no means complete or systematic, it is intended to indicate a direction for modelling human interaction in software:

- asking questions,
- answering questions,
- giving hints,
- giving examples,
- pointing out references,
- pointing out alternatives,
- demanding activity from the student,
- insisting,
- repeating,
- checking for correctness,
- confirming results,
- providing solutions.
All of the above interactions can be found in existing educational software. Nevertheless, most humans prefer interaction with humans and describe software as fascinating in the beginning but boring and limiting in the long run. This might be founded in the fact that existing software uses these building blocks in fixed scripted sequences or even hardcoded into the program.

So another important feature of human interaction is the flexibility to act in various ways depending on the situation and the goals to be achieved.

Software trying to model human interaction needs the ability to adapt to several aspects:

**Theory of learning:** Depending on the underlying theory of learning, educational software would choose specific interactions from the list above and use them in specific sequences.

**Problem to be solved:** In the context of mathematics, some problems can be solved by learning to apply a simple algorithm, while others may demand deeper insight or theoretical background. As a result, every problem to be solved would require a specific sequence of interactions.

**Experience of the student:** Depending on the student’s experience, new knowledge demands more attention than knowledge already mastered, therefore interaction specific to the individual solving a problem and the pieces of knowledge involved.

**Present situation:** Most importantly, humans can react to their peer’s present levels of concentration, motivation or frustration, which can change rapidly.
Chapter 5  

User Interaction from a Technical Point of View

Based on the analysis in the previous chapter, let us explore possibilities to model human tutoring in software. While we cannot expect software to reach levels of human intelligence or empathy in the near future, our vision is to provide for near-human levels of flexibility of behaviour.

5.1 Describing a Dialog

To make behaviour of software configurable, a means of abstracting behaviour will be needed. A wide variety of programming languages provide abstractions of a computer’s possible behaviour. Developing a sort of scripting language for interaction in tutoring will require further research in the fields of both computer science and didactics. IS4C could serve both as a prototype and as a testing platform for research efforts to come. We are well aware that this section will pose more questions than it can answer.

5.1.1 Dialog Atoms

Taking the list of interactions from the previous chapter as a starting point, choosing a proper level of abstraction is an issue. The single interactions to build IS4C’s behaviour from are called Dialog Atoms.

The following proposal for possible atomic interactions when taking a step in a calculation during the Solving Phase is quoted from [Neu01]:

The dialog atoms are the following, ordered by descending ‘activity’ of the learner: All atoms concern a step from the current formula \( f \) applying a tactic \( tac \) which yields the resulting formula \( f' \) (the derivation of \( f \)), i.e. \( f \rightarrow_{tac} f' \).

1. given \( f \), input the next formula \( f' \)
2. given a partial \( f \) (supplied by \( \text{ISAC} \)), complete \( f \) such that it is a derivation of \( f \)

3. given \( f \), input a tactic \( tac \) to be applied to \( f \)

4. given \( f \), select \( tac \) from a list (supplied by \( \text{ISAC} \)) to be applied to \( f \)

5. given \( f \) and a partial \( tac \), complete the \( tac \) (i.e. a theorem, a substitution, etc.) such that it can be applied to \( f \)

6. given \( f \), \( tac \), and a partial \( f' \), complete \( f' \) such that it is the result of applying \( tac \) to \( f \)

7. given \( f \) and \( f' \), input \( tac \) such that \( f' \) is the result of \( f \) applying \( tac \)

8. given \( f \) and \( f' \), select \( tac \) from a list (supplied by \( \text{ISAC} \)) such that \( f' \) is the result of \( f \) applying \( tac \)

9. given \( f \), \( f' \) and a partial \( tac \), complete \( tac \) such that \( f' \) is the result of \( f \) applying \( tac \)

### 5.1.2 Sequencing

Once a set of Dialog Atoms is implemented, these can be sequenced to build interactional patterns. This might suffice to provide different behaviour depending on the theory of learning used, as suggested in section 4.2.

On a different level of detail, sequencing Dialog Atoms could provide specific behaviour matching the problem to be solved as well. Such patterns would be stored as additional information per example in an example collection. Even the user’s level of expertise could be taken into account, if the sequences can be adapted by a supervising teacher.

Existing educational software may already be based on libraries of Dialog Atoms sequenced by simple scripts.

If using sequenced Dialog Atoms both for implementing a learning theory and for guiding through a specific example, the additional problem of reconciling both sequences might arise.

### 5.1.3 Knowledge Items

In a highly interactive system such as \( \text{ISAC} \) (UC 1.3.2, UR 2.2.9), fixed sequences are bound to fail because the user is free to choose his own way of solving a problem and depart from the pre-programmed path.

To handle this situation, which cannot be foreseen before runtime, a suitable scripting language would have to provide conditional statements. Conditions would have to recognise the knowledge involved in the present interaction, which calls for Knowledge Atoms recognisable by the system. \( \text{ISAC} \) will use Tactics as preliminary Knowledge Atoms for further research.
5.1.4 Situation of the User

The biggest challenge is to take the user's individual situation into account. Little is known about categories of users' situations and it may be doubted that atomic decomposition of the properties of a user makes any sense at all.

On the other hand, one could argue that usable assumptions may be derived from which knowledge the user has already used and how much effort or time it took him. It cannot be foreseen whether consistent data about the user will become available, but a dialog scripting language should be prepared to make use of them.

5.2 Modelling a User

As stated before, information about the user is desirable to adapt the Dialog's behaviour accordingly. How could such information be obtained and processed to provide the system a usable abstraction of a user, i.e. to build a User Model?

We have identified

- the Dialog Atom used,
- the knowledge touched,
- the success in doing so
- and the identity of the user himself

as important parameters of an interactional situation. It is hoped that if these parameters are observed over a large number of interactions, they provide significant information about situations already encountered, which can be used if similar situations occur again [LJS99].

Some sort of processing to detect correlations will be necessary, be it with statistical methods, neural networks or other technology.

One critical point is to obtain a measure for a user's success in using knowledge in the context of a Dialog Atom. This will require expertise and further research in the field of didactics, which is out of the scope of this thesis. ZSAC will use the time spent in the interaction and whether the user's input could be confirmed by the Math Engine as a starting point for further experiments.
Chapter 6

System Environment

In this chapter, the environment of the Dialog Guide component will be discussed. While an effort was made to design the Dialog Guide as universally usable as possible, there are still aspects of the system environment to be considered. Beyond a mere description, relevant decisions of the ZSAC team on the overall system will be discussed as well.

6.1 Basic Concepts for Separable User Interfaces

Apart from the benefits of structuring complex systems, separable user interfaces provide adaptability of look-and-feel without the need of reworking the entire system. For our analysis, we will concentrate on the Seeheim and Model-View-Controller (MVC) basic architectures.

6.1.1 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, keypresses 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 Dialogue Controller defines the structure of the interaction between user and system. Typical tasks of the Dialogue 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 Dialogue 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.

![Interaction in the Seeheim Architecture](image)

Figure 6.1: Interaction in the Seeheim Architecture

Note that in figure 6.1, the messages are named "notify" and "request" from the user’s point of view. From the Dialogue Controller’s point of view, the messages are distinguished by their direction in or out of the Dialogue 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 Dialogue Controller’s state and logic. It is the semantic in the Dialogue Controller’s logic that makes a difference between user and application, if any.

6.1.2 The MVC Architecture

As opposed to the Seeheim Model, which structures the system as a whole, the MVC architecture [BMR+96] is grouped around single data objects as follows:

The Model is any data object in the application requiring user interaction.

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.

In a complex system, the link between application and user can contain several Model-View-Controller-triples, each grouped around a specific data item.

6.1.3 Comparing the approaches

- Without the need to pass the Dialogue Controller on every interaction, the MVC model tends to be faster in runtime. This is particularly
important for giving the user immediate feedback about his current actions and options.

- Being built around smaller units of single objects, MVC is more flexible and easier to develop and extend.

- On the other hand, MVC lacks the clear separation between application and presentation layer, spreading the functionality across a multitude of interacting MVC triples. While this eases development, the resulting complex dependencies make it harder to understand or debug the system as a whole.

6.1.4 Implications for ZSAC

The design of ZSAC is based on the Seeheim Model, for the following reasons:

- A clear separation of Application, Dialogue Controller and Presentation layer is a design goal because:
  
  - At present, the Application itself is written in SML, whereas the rest of the system is being implemented in Java. The necessity to interface the different worlds of different programming languages implicitly separates Application from User Interface.
  
  - The already-implemented Math Engine with the Isabelle system in the background uses more computing resources than a typical consumer machine can provide at present. This and the goal to centralise mathematical knowledge and example collections for groups of users suggests running the Application on a dedicated server.
- To minimise effort and expenses on the side of the user, the part of the system running on the user’s machine should be kept as small as possible, a Java-capable web browser being the aimed-at minimum. An ideal design would leave only the Presentation Layer running on the user’s machine.

- The goal to adapt ZSAC’s behaviour to the situation of the individual user’s situation asks for a well-defined, configurable and exchangeable Dialogue Controller.

- As it seems foreseeable that the design of ZSAC’s Dialogue Controller will pose questions going well beyond the scope of a single master’s thesis, a separable Dialogue Controller component will ease independent research. We expect the development of new approaches to dialogue description languages as practical experience with a ZSAC prototype becomes available.

• The major drawback of the Seeheim Model, the difficulty to provide immediate feedback to the user, is not relevant to the design of ZSAC, as the complex mathematical knowledge involved requires consultation of the Math Engine for even basic feedbacks. With the present speed of the Math Engine, any additional delays from the use of the Seeheim Model will not be perceivable by the user.

Any operations not involving mathematical knowledge can be handled locally by the Presentation Layer, in a MVC manner, if desired.

• The aforementioned goal of adapting ZSAC’s behaviour to the situation of the individual user can be easily addressed by augmenting a centralised Dialogue Controller with a User Model. The User Model would store preferences set by the course designer and the user himself. The Dialog Guide would report activities to the User Model, which would store these reports and process them into an abstraction of the user’s preferences, experience and behaviour. The other way round, the User Model would be queried by the Dialog Guide for clues how to behave in a specific user-interaction situation.

Based on these considerations, the top level design of ZSAC looks like this:

**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 centralised 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 Dialogue 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.

![Diagram](image)

Figure 6.3: Basic ISAC architecture for calculations

### 6.2 Distributed Components in a Heterogenous Environment

Several questions arise from the fact that the various components of the ISAC system reside on different machines and at least two programming environments - Java and SML - are used.

#### 6.2.1 Location of the Dialog Guide

With at least two machines involved - the user’s computer with the Worksheet and the server with the Math engine - the question where to put the Dialog Guide remains. The Dialog Guide accesses the Math Engine, the Worksheet and the User Model frequently. For simplicity, mobility, security and centralisation reasons, the User Model cannot reside on the user’s machine. The same is true for the Dialog Guide. The Dialog Guide with the persistent data of the User Models could run on the server together with the Math Engine or on an other server of his own. The Dialog Guide is designed with the ability to run on a machine of its own in mind. The final decision
on the location of the Dialog Guide will be based on tests with the prototype implementation.

6.2.2 Interfacing Java and SML

The problem of interfacing the SML and Java environments has been delegated to a separate component, the Bridge [GHJV95]. The Bridge communicates with the Math Engine via text pipes, parses the data and represents relevant data as Java objects. The fact that there is another system involved, which operates in a non-object-oriented manner and communicates via text pipes is completely hidden from the Java part of ISAC. Additionally, the bridge could use intelligent strategies, e.g. using the Proxy pattern [GHJV95], to address the bottleneck problem of several Worksheets communicating with the Math Engine through a single text pipe. For an in-depth discussion of the Bridge see [Gra04].

6.2.3 Choosing a Means of Communication

For integration of the various components across several machines an off-the-shelf solution was sought with the following criteria in mind:

- **Speed**: Many of the user’s requests originating on the Worksheet require processing in the Math Engine. With ISAC being an interactive system, we need response times near the response times in normal human interaction.

- **Security**: With many of the features of ISAC depending on the identity of the user and the enforcement of access rights, basic security features are a necessity.

- **Mobility**: Users might want to use ISAC at work, in class or at home, even on computers they use only occasionally. This asks for communication robust against changing addresses of client and server computers, but still secure.

- **Conformity to Standards**: is especially important for the Worksheet, which runs on the user’s machine. An ideal solution would build on standard resources available on the user’s machine without the need of additional installation. If installation is required, it must be kept as simple as possible.

  With the goal of being open to cooperation with the tools of other projects, a standard solution would ease integration of the different systems.

  In addition to that, a standard solution is preferred with the limited resources of the ISAC project in mind.
Several options were reviewed to find a suitable means of communication, among them Java-RMI \(^1\), XML-RPC \(^2\), SOAP \(^3\) and Dinopolis \(^4\).

While the Dinopolis approach looked most promising with respect to the aforementioned criteria, it was not yet available in a stable implementation. Java-RMI was chosen for the implementation of the TSAC prototype. The strongest arguments for Java-RMI were the seamless integration into the Java programming environment and the widespread use and accepted stability.

### 6.2.4 The Dinopolis Middleware project

Dinopolis \(^{[Sch02a]}\) is an object management middleware system aiming at object and component retrieval, security and run-time polymorphism. While not part of the current implementation of TSAC, Dinopolis still played an important role during the design phase.

The main features of Dinopolis include:

**Component and object management** Extending the traditional data-and-operations view, Dinopolis objects have the following aspects, all handled by the middleware:

- **Content** is the traditional data content.
- **Meta-Data** can be used for administrative purposes which are not necessarily reflected by the content.
- **Interrelations** associate several objects, of equal or different types, in a mn:n manner.
- **Operations** manipulate the content of an object.
- **Services** are GUI building blocks offered by an object to facilitate data manipulation by the user. Services can be viewed as the View and Controller parts of a MVC triple.

All of these aspects are handled transparently by the Dinopolis middleware according to the current run-time context, which not only takes burden from the programmer, but also allows for run-time dynamic typing as a key feature.

Objects and components are accessed through Globally Unique Handles (GUH), a mechanism allowing for unambiguous retrieval of objects independent of their physical locations \(^{[Sch02b]}\).

**Role-based security** Dinopolis takes control over access to components, objects, their data and operations. Access control is based on the user

\(^1\)http://java.sun.com/products/jdk/rmi/
\(^2\)http://www.xmlrpc.com/spec
\(^3\)http://www.w3.org/TR/SOAP
\(^4\)http://www.dinopolis.org/
and his role as opposed to the user's identity alone. With Dinopolis, the same user would have different access rights and therefore see different aspects and different behaviour of objects depending on whether he accesses ISAC as administrator, teacher or student. All security-relevant operations are handled transparently by the Dinopolis middleware, so the only task left to ISAC is to identify the user. As far as objects are concerned, the transparent security system may be considered a special case of dynamic typing described above.

**Ability to embed existing systems** Existing systems can easily be integrated into Dinopolis by writing embedders for existing objects such as databases. This could ease integrating ISAC with other projects without the need to define common data structures. In addition to that, the SML part of ISAC could easily be integrated this way.

**Platform-independency** At present, Dinopolis is being implemented in C++ and Java, but the protocols and algorithms are open to arbitrary programming languages.

The features mentioned above make Dinopolis ideally suited for the needs of ISAC, for communication of the various distributed components as well as for security and role-dependent behaviour of the system. Unfortunately, Dinopolis is still being developed and was not finished in time to base the current prototype of ISAC on. The ISAC team still hope to use the features of Dinopolis in future versions.

### 6.2.5 Java-RMI

Java Remote Method Invocation provides for communication of processes running in different Virtual Machines, even if the VMs run on different computers. Basically, RMI provides invoking methods on remote objects and passing parameters to remote methods. RMI cannot be compared directly to Dinopolis, because RMI's purpose is communication between remote objects, but not object retrieval, system security or dynamic typing.

To invoke methods on a remote object, the remote object has to implement a public remote interface extending `java.rmi.Remote`. Every method in the remote interface has to declare to throw `RemoteException`. The remote object may implement other methods in addition to the remote interface, but only the methods in the remote interface can be invoked remotely.

Any objects implementing the `java.io.Serializable` interface can be passed as parameter. Unlike other distributed component systems, this is the only condition imposed on parameters.

Apart from the non-restrictive conditions listed, remote objects can be used like local objects once a connection is established.
This tight integration into the Java programming environment is the key advantage of Java-RMI and one of its few drawbacks, as it is limited to Java as a programming language.

6.3 The Architectural Design of the TSAC System

In this section, the specific environment for this implementation of the Dialog Guide component will be discussed, with a few notes on the overall design of the TSAC prototype. In addition to the references given in the text, more details on the overall design can be found in [GKN03, GKN02b].

6.3.1 The Overall Design of TSAC

In addition to the components discussed so far, the TSAC system contains several other components, which are not so closely related to the Dialog Guide component and the topic of this thesis.

Browsing the Knowledge Base

In contrast to most currently available algebra systems, TSAC bases its calculations entirely on rules and knowledge visible to the user. For every step done in a calculation, there is a justification in TSAC’s Knowledge Base, which can be displayed on request. Even more importantly, these justifications are meant to be understood by the user, as they are expressed in terms of human mathematical reasoning, not in sophisticated optimised algorithms.

As TSAC’s Knowledge Base can be understood by humans, it can be used as a reference or even as a learning tool. Interaction with the Knowledge Base is moderated by a Dialogue Controller, in a way similar to interaction with the Math Engine in an ongoing calculation. Early designs of TSAC had a single Dialog Guide component controlling both interaction with calculations and with knowledge.

Figure 6.4: The first sketch for TSAC’s architecture
As mathematical knowledge is of rather static nature, browsing the Knowledge Base is independent of ongoing calculations. This led to two subsystems which can be used separately, each with a Dialogue Controller and a Presentation Layer of its own. Even the respective Application Interfaces have little in common.

The two subsystems interact in just a few points:

• Both Dialogue Controllers share a common User Model, for a inventory of knowledge supposedly known to the user, be it from browsing the respective knowledge item, be it from having used specific knowledge in a calculation.

• When browsing the Knowledge Base, an example illustrating the presented concept can be calculated.

• When doing a calculation, items from the Knowledge Base justifying the correctness of the calculation can be displayed.

For an in-depth discussion of the Knowledge Browser, see [Gri03].

![Diagram of subsystems](image)

Figure 6.5: Design based on the Seeehm model and showing the separation of browsing the knowledge and calculating

**Authoring Tools**

Specialised tools are needed for editing the collections of static information stored in the system, such as

• Methods

• Problems
• Examples

• Explanations

The implementation of authoring tools has been postponed to a later time, when experience with a running prototype will be available. These tasks can be easily separated from the current design and implementation process, as they affect only static data and do not depend on the running system. For the time being, this information is entered directly into ZSAC’s storage, which resides in human-readable text files. While human-readable, the format of these files is not easily understood without knowledge of ZSAC’s inner structure. For thoughts on the structure of future authoring tools, see [Gri03].

Logging into the System and Bootstrapping

As a distributed system, ZSAC is started in several steps. For the following discussion, we will differentiate ZSAC’s components into centralised static components and components created dynamically on a per-session or per-user basis.

Static, centralised components:

• Math Engine
• Knowledge Base
• Example Collection
• Object Manager

Components created per session or per user:

• Session Dialog
• Presentation Layers
• Dialog Guide
• Browser Dialog
• User Model

As for the centralised components running on dedicated servers, we will assume they have been started by their respective administrators and their services are available at login time of an individual user.

For initializing the components of ZSAC running on the user’s machine, several actions have to be performed:

• Identify the user
• Start components individual to a user or session
• Retrieve the user’s User Model from centralised storage
• Connect the components - centralised and individual - to each other

One of the main problems in designing this part of IS4C is finding a bootstrapping procedure capable of connecting all the relevant components, which essentially means making sure that every component can address any component it needs to cooperate with.

Whereas the always-present, centralised components can easily be addressed by entering the network address of their respective servers or storing the addresses in configuration files, the dynamically created per-user and per-session components are harder to localise. In principle, only the instance which created a component can know how to address it in the first place. As a solution, a Session Dialog component was introduced to initialise and keep track of all dynamically created components.

![Diagram](image)

**Figure 6.6: Adding session management and a shared user model**

Another problem is communicating with web browsers running on the user’s machine unless new content has been requested by the browser. This makes it impossible to notify the user of a change in the state of one of the centralised components until the browser polls for new content. While this does not hinder browsing the Knowledge Base which presents data of rather static nature, watching a calculation being modified by the Math Engine requires a minimal software component capable of handling requests from the network running on the user’s machine.

Once it was clear that some sort of active software on the user’s machine would be required, parts of the Session Dialog functionality were moved onto
the user’s machine along with the login procedures and a basic GUI coordinating the display requirements of the various components. The component running on the user’s machine is simply referred to as GUI.

It is hoped that features and stability of the startup process will greatly improve once the Dinopolis middleware system becomes available.

For details on the design of the bootstrap process and session management see [Gri03, Hoc04].

Figure 6.7: The overall design of the ISAC system

6.3.2 The Interfaces to the Dialog Guide Component

Data exchanged at the interfaces of the Dialog Guide component include:

Examples to be started When initialising the Dialog to moderate a process of calculation, the starting point can be an empty worksheet or an Example from the example collection. In case of starting from an Example, the Example has to be passed to the Dialog.

Notifications about updates in a calculation With present technology, calculations done by the Math Engine may take longer than the average user would wait. Moreover, response times are not easily predictable, so waiting for a call to return would block the Dialog Guide - hence user interaction - for too long a period of time. Therefore calls to the Math Engine return immediately, with asynchronous notifications being sent when the Math Engine completes a request. In addition to continuous attention to the user, this approach allows for several users watching one and the same calculation on their respective Worksheets and being even notified of updates in the calculation requested.
by other users. For efficiency reasons, the update notifications contain hints about which parts of the Calc Tree may be affected by the update. These notifications are passed from the Math Engine to the Dialog and from the Dialog to the Presentation Layer.

**The calculation itself** The Dialog needs access to the Calc Tree stored in the Math Engine and passes a filtered version of the tree to the Worksheet for display. The Dialog cannot understand the mathematical meaning of Formulas, but is interested in identifying Tactics. It is the Tactics which the user is learning to apply and the Dialog Guide has to provide appropriate user guidance for.

**Calc Head** As with the Calc Tree during the Solving Phase, during the Specifying Phase a Calc Head has to be shared between Math Engine, Dialog Guide and Worksheet.

**Notifications about user requests** The Dialog has to be informed about actions the user triggers on the Worksheet. The Dialog in turn translates the user actions into internal state changes or requests to the Math Engine.

**Requests to the Math Engine** As the Math Engine stores the only instance of the Calc Tree significant to further processing, all manipulations of the tree have to be done by the Math Engine. Request to edit the calculation originating from the user are processed by the Dialog Guide and execution is requested from the Math Engine.

**Information touched** Records of the user’s interaction with ZSAC’s knowledge are kept in the User Model and abstracted to ZSAC’s view of the user’s knowledge and abilities. The User Model is informed about every interaction of the user with the calculation or the Knowledge Base. The User Model’s abstraction is in turn queried by the Dialog Guide to decide on details of user guidance.

**Dialog Atoms** Information about the Dialog Atoms involved in user interaction is passed to the User Model to record not only the fact that the user interacted with certain parts of knowledge but also the nature of the interaction.

**User settings** The user’s preferences about the way he wishes to be guided have to be communicated to the Dialog Guide, whereas preferences about the visual appearance of the GUI are communicated directly to the Worksheet.
Chapter 7

Designing a Dialog Guide Component

In this chapter, design specific to the Dialog Guide component will be dealt with. We will start with a closer look at the Dialog Guide’s internal organisation and behaviour. In the remaining sections, we will concentrate on how the Dialog Guide deals with data or components shared with other parts of the system.

Relationships to design patterns described in [GHJV95] will be pointed out where appropriate. If needed, refer to section 6.3 or 0.3 in the introduction for explanations of terms.

7.1 Controlling the Course of Interaction

The Dialog Guide is responsible for guiding the user the way to obtaining a solution to a problem.

7.1.1 Dialog Phases

On a coarse level, interaction goes through several phases with a fixed sequence, independent of the particular problem being solved (UR 2.2.8). These so-called Dialog Phases have been modelled on a state machine with well-defined states and transitions between the states. During each of these phases, the Dialog Guide behaves differently and reacts to different requests. Regardless of mathematical context, the Dialog Phases provide a certain degree of error-robustness by recognising out-of-order events.

With the Dialog Phases and their relationships becoming more complex in future development, providing separate sub-classes for the Dialog’s behaviour in different states may be appropriate, as described in the State pattern [GHJV95].
Initialising

To start interaction, the Dialog Guide has to establish connections with the components it interacts with, a Worksheet representing the Presentation Layer and a Math Engine representing the application (UC 1.1.2). In addition to that, the Dialog Guide needs information about the user it deals with, to be able to adapt its behaviour accordingly (UC 1.1.1, UR 2.1.1). Only after being provided with a CalcHead to act upon, optionally filled in with a Formalization of a pre-defined Example (UC 1.1.3), the Dialog can enter the Specification Phase.

Specifying

The goal of this phase is to gather enough - and consistent - information to start solving (UC 1.2.1). During this phase, the user can add information to a CalcHead, in arbitrary order. After every item added, the CalcHead is checked with the Math Engine for consistency and completeness (UC 1.2.2, UR 3.1.5).

Requests for help entering items cannot be answered by the Dialog Guide because of lacking mathematical knowledge. Such requests are passed to the Math Engine, if allowed by the user’s settings. The Specifying Phase can be finished only after the CalcHead is confirmed being complete and consistent by the Math Engine (UC 1.2.4).

Solving

To enter the Solving Phase, a valid CalcHead is required. Consequently, the mathematical situation initially described by the CalcHead is transformed towards a situation called result. The transformation is performed in steps (UR 2.2.9) which are recorded in a CalcTree (UR 2.2.12). As opposed to the Specifying Phase, the steps are not cumulative, but sequential. This implies that while it is possible to change steps already taken, such an action is likely to render subsequent steps invalid (UC 1.3.6). The transformations are not performed by the Dialog Guide itself but by the Math Engine. Requests to take a step are passed to the Math Engine and steps entered by the user are checked by the Math Engine (UR 2.2.13). Note that the Dialog Guide does not know anything about mathematics, it knows only about the structure of interaction in problem-solving. As stated before, transformations can be done entirely by the user or by the Math Engine, or with combined effort of both. This opens up a spectrum of interactional possibilities how to take a step and the various possibilities are described as Dialog Atoms (see section 5.1.1).

With the concept of a state machine in mind, additional phases can be added easily in the course of future development to handle more complex
sequences.

Solving Subproblems

Subproblems are calculations within calculations. In principle, they do not differ from top-level problems, but during testing of the prototype implementation the necessity to treat the specification of subproblems differently might turn up. This can be achieved by adding another Dialog Phase. In the present implementation, interactive specification of subproblems is skipped and taking this step is always left to the Math Engine.

![State machine for Dialog Phases](image_url)

Figure 7.1: A state machine for the Dialog Phases

7.1.2 Dialog Atoms

As opposed to the high-level Dialog Phases, Dialog Atoms are basic building blocks of system-user interaction at the level of a single interaction. For configuring the Dialog Guide’s interactional behaviour (UR 3.3.2), we aim at developing an abstract language with Dialog Atoms (UR 3.3.1) as part of the vocabulary. The Dialog Guide could contain an API for programming its behaviour, with a lower-level interface implementing an Interpreter pattern [GHJV95] for Dialog Atoms and a high-level interface implementing the Strategy pattern.

Let us quote [Neu01] again:

The dialog atoms are the following, ordered by descending 'activity' of the learner: All atoms concern a step from the current formula \( f \) applying a tactic \( tac \) which yields the resulting formula \( f' \) (the derivation of \( f \)), i.e. \( f \longrightarrow_{tac} f' \).

1. given \( f \), input the next formula \( f' \)
2. given a partial \( f \) (supplied by \( \mathcal{L}S4C \)), complete \( f \) such that it is a derivation of \( f \)
3. given \( f \), input a tactic \( tac \) to be applied to \( f \)
4. given \( f \), select \( tac \) from a list (supplied by \( \mathcal{L}S4C \)) to be applied to \( f \)
5. given \( f \) and a partial \( tac \), complete the \( tac \) (i.e. a theorem, a substitution, etc.) such that it can be applied to \( f \)
6. given \( f \), \( tac \), and a partial \( f' \), complete \( f' \) such that it is the result of applying \( tac \) to \( f \)
7. given \( f \) and \( f' \), input \( tac \) such that \( f' \) is the result of \( f \) applying \( tac \)
8. given \( f \) and \( f' \), select \( tac \) from a list (supplied by \( \text{TS4C} \)) such that \( f' \) is the result of \( f \) applying \( tac \)
9. given \( f \), \( f' \) and a partial \( tac \), complete \( tac \) such that \( f' \) is the result of \( f \) applying \( tac \)

Note that exchanging the parts of the user and \( \text{TS4C} \) in the above proposal yields another set of Dialog Atoms, which can be treated as equivalent from the Dialog's point of view. Taking atom 1 as an example, it is essentially the same whether \( f \) is supplied by \( \text{TS4C} \) and \( f' \) is expected to be input by the user or the user asks \( \text{TS4C} \) to derive \( f' \) from \( f \). In abstract terms, in both cases one part provides \( f \) and the other part is expected to supply \( f' \). For this reason, the Dialog Guide tries to provide symmetric Dialog Atoms and make use of this symmetry in the implementation.

### 7.2 Sharing the Calculation with Other Components

#### 7.2.1 Representing the Specification

A Formalization (UR 2.2.3) storing lists of formulas called Given, Find, Where, Relate and a Specification (UR 2.2.4) storing identifications of a Theory, a Method and a Problem comprise all data necessary to specify a calculation to the Math Engine. In the \( \text{TS4C} \) system, this information is called a CalcHead, indicating that every calculation has a header specifying a starting point. Once the Solving Phase of a calculation is started, this information does not change any more.

#### 7.2.2 Representing the Path to the Solution

The path to the result is represented by a tree-like structure (UR 2.2.12), alternating formulas and tactics (UR 2.2.10). There is always exactly one Tactic being applied to a formula. In the course of calculating a result, the structure grows as new formulas are added by the user or the Math Engine.

#### 7.2.3 Treating Subproblems

A subproblem is a calculation within a calculation. As such, every subproblem is preceded by a CalcHead. Once specified by a CalcHead, a subproblem could be treated as a calculation of its own and solved independently of the
enclosing calculation. Two possibilities for treating subproblems and feeding their results back into the main calculation were explored.

**Independent CalcTrees**

Every subproblem could be stored in an independent CalcTree. This would emphasise the fact that a subproblem can be solved independently of the enclosing problem and reflect that fact in the data structure used. As an advantage, every calculation would have exactly one CalcHead and exactly one CalcTree associated with it, representing the data involved in the Specifying Phase and the Solving Phase, respectively. This would allow for clearly separating the CalcHead from the CalcTree thus reflecting the separation of the Specifying Phase from the Solving Phase in the storage of data. On the other hand, such an approach would complicate feeding back the results of a subproblem into the enclosing calculation.

**One Common CalcTree**

As an alternative, all data of a calculation, including associated subproblems, could be stored in a single data structure, subproblems being stored as branches of the tree. While this approach reflects the fact that a subproblem is part of the enclosing calculation, the distinction between Specifying Phase and Solving Phase becomes blurred, as the CalcHeads specifying subproblems must be stored within a CalcTree. Having subproblems tightly integrated into the enclosing calculation eases using their results.

This approach was chosen for implementation, in part due to the fact that the already-implemented Math Engine stores calculations in a single tree.

**7.2.4 Accessing Calculation Data**

**CalcHead**

With the CalcHead having a fixed number of fields, all members can be accessed directly. Wherever components share a CalcHead, the object itself is referenced or passed.

**CalcTree**

For accessing data in the dynamically growing CalcTree, the Iterator pattern [GHJV93] was chosen for its main advantage of hiding the internal representation of the data accessed. Several reasons suggested hiding the internal representation:

- An Iterator can serve the additional purpose of referencing elements of a calculation.
• An Iterator is likely to be a much smaller object than the calculation it points into.

• Several components residing on different machines imply having to pass information about the calculation across the network. Using compact Iterators instead of the entire calculation would make efficient use of network bandwidth and save computing time needed for serializing large objects.

• At an early stage in design, the final structure of a calculation as stored in the Java-implemented part of $ZSAC$ was not yet decided upon. Using Iterators made it possible to start development of other components without knowing which data structure would be eventually implemented.

• There was much debate about runtime efficiency versus ease of development in representing a calculation. Hiding the internal representation would allow for implementing efficient data structures at a later time without having to redesign the entire system.

• Neither the Dialog Guide nor the Presentation Layer need to know how a calculation is actually stored. On the other hand, both components are interested in the structure of a calculation as presented to the user. Using Iterators would allow traversing a calculation in a user-oriented manner independent of the actual implementation.

7.2.5 Communicating Changes in the State of Calculation

As a component sitting between the Application and the Presentation Layer, one of the tasks of the Dialog Guide is to propagate information about changes or events in one component to the other. The Dialog Guide intercepts the flow of information and modifies it by implementing $ZSAC$'s logic of user-interaction.

Wrapper-based Design

One approach is wrapping the objects representing the calculation - the Calculation Tree and associated objects - into objects with the same interfaces but different behaviour, following the Decorator pattern [GHJV95]. This way, the Dialog Guide can filter information considered not appropriate for being presented to the user by simply removing these data from the representation accessible to the Presentation Layer. For an example, if the user is not interested in the Tactics transforming one formula into another, the Tactics simply do not show up in the representation of the calculation seen by the Presentation Layer. This has the advantage of simplicity - the Presentation Layer and the Application need not consider filtering taking place or even
know about filtering at all. Moreover, the same interface can still be used if one would want a system without the intervention of a Dialog Guide for direct communication between the Application and the Presentation Layer.

**Event-driven Design**

In addition to data representing the state of calculation, there is data representing changes in time. Many of these changes occur asynchronously at unpredictable intervals - such as interactions of the user - or with considerable unpredictable delay after the event that triggered them - such as results of a lengthy transformation becoming available. This sort of changes is communicated through event messages, with objects interested in such notifications registering as Observers [GHJV95] with sources of events. The main sources of events are the Presentation Layer for user actions and the Application for new information about the calculation becoming available. The Dialog Guide intercepts and filters these messages using the Mediator pattern [GHJV95].

### 7.3 Configuring the User-Interface

The Dialog Guide and the Presentation Layer have to cooperate closely in user interaction. Consider a button on the screen triggering some action. It is the Presentation Layer’s responsibility to render the button and to notice the user clicking it. It is the Dialog Guide’s responsibility to decide whether the user is allowed to request such action and to trigger appropriate action in the Application.

The division is not always so clear-cut. Consider internationalisation of the user-interface (UR 2.3.1): Is it the Presentation Layer, which is responsible for rendering in general and the language environment, that decides which text to set on the button? Or is it the Dialog Guide, which knows the meaning of the action triggered by the button? For the following considerations, we will stick to the example of the button.

#### 7.3.1 The Presentation Layer in Control

If the Presentation Layer controls every aspect of a button, such as visual appearance, placement on screen and the actions triggered, everything seems easy. Problems arise if we consider buttons which are needed only in special contexts. A button asking for the next Tactic to be applied to a formula does not make sense during the Specifying Phase, where no Tactics occur.

We could show the button all the time, with the Dialog Guide simply ignoring requests when not appropriate. This has the disadvantage of confusing the user with lots of buttons which can be clicked but make no sense in the current context.
We could show buttons only if clicking them makes sense. If the Presentation Layer were to solve this problem, it would need information about the current phase of the dialog. While this may seem feasible, in other situations the applicability of the button might depend on the user’s role or privileges, or on the user’s level of expertise, which in turn might change even during a session. Especially if buttons depend on didactic strategies, this involves knowledge which has nothing to do with presentation but is clearly part of the Dialog Guide.

7.3.2 The Dialog Guide in Control

If we put the Dialog Guide in control of the buttons, it becomes easy to solve problems with context, but this would require the Dialog Guide to care about internationalisation and visual appearance, which is out of interactional context and should be left to the Presentation Layer.

7.3.3 Splitting up Responsibilities and Providing for Interaction

It seems best to have the various aspects of a button controlled by the component which possesses the information necessary to do so.

While the Presentation Layer should control every visual aspect of a button such as text, shape and placement on screen, the Dialog Guide should control the context in which the button appears and the action it triggers. See [SB001] for the discussion of a related problem.

First attempts aimed at providing means for the Dialog Guide to enable or disable buttons otherwise controlled by the Presentation Layer. In the meantime, the goal changed to having the Dialog Guide control the creation and destruction of elements of user-interaction as well.

The Dialog Guide creates a element of user-interaction by providing an identification of the action it triggers. The presentation layer need not even understand the meaning of the action, it uses the identification merely for notifying the Dialog Guide which action has been triggered by the user. In addition to that, the Dialog Guide provides the Presentation Layer with hints about the context of the user interaction, such as whether the action relates to a single formula or the user’s session as a whole. The Presentation Layer can use this information to choose an appropriate visual representation. Moreover, the the Dialog Guide does not even request the trigger to be a button. It requests that a means for the user to trigger a request be created and it is left to the design of the Presentation Layer to offer a button, a menu item or both. This approach bears similarities to the Factory pattern [GHJ95], but the created object remains with the Presentation Layer and is not passed back to the Dialog Guide.
7.4 Obtaining and Storing Configuration Data

Much of the Dialog Guide’s behaviour can be parameterised (UR 3.3), and many of these parameters are individual to a user (UR 3.4). Moreover, some of the parameters are modified by the system itself during a session (UR 3.4) based on data collected (UR 3.2).

7.4.1 The User Settings

By user settings we denote preferences on aspects of the system set by the user (UR 3.3.2, UR 3.3.4), such as amount of data shown, levels of difficulty or customisations of the visual appearance of the program. As these settings do not pertain only to the Dialog Guide but also to other parts of the system, they will be managed outside the Dialog Guide. It is to be noted that this information does not change very often, so efficient processing is not an issue.

7.4.2 Permissions and Security Issues

With ZS4C being developed as groupware (UR 2.1.5), special attention has to be paid to the fact that settings will be set not only by the individual user, but also by privileged persons such as course administrators (UR 2.1.4). In addition to that, changing some of the settings may be subject to restrictions (UR 3.5.1) depending on the role of the user. It is assumed that reconciliation of contradictory settings and security issues have already been resolved by user management and that the Dialog Guide has access to the settings in effect for the current session.

7.4.3 The User Model

As required in UR 3.4, the Dialog Guide will adapt to the assumed knowledge and abilities of the user. Decisions about which information to show and which actions to take will be based on an internal abstraction of the experience the system had with the user, the User Model.

The User Model is notified about every interaction between user and mathematical knowledge and information about the knowledge involved (UR 3.4.4). The user’s performance (UR 3.4.3) is recorded together with information about the context of the interaction, i.e., the Dialog Atom used in the interaction. For efficiency reasons, data is stored as statistical digest rather than as log. The User Model is accessed frequently, at least once per interaction both for query and for recording the outcome, and logging every event would grow the amount of data processed unmanageable very soon.

Note that the User Model gathers and processes the data, but is not aware of their meaning - knowledge items and Dialog Atoms are processed as identifying numbers. Interpretation of the data is left to the components...
which use them, i.e. the Dialog Guide and, in the future, the Knowledge Browser.

Abstractions on a higher level than the presently used statistics could be added in the future, as cooperation of ISAC in the field of didactics could provide abstract measures e.g. for a user’s familiarity with a topic. In any case, the User Model provides descriptive information about the user and decisions about further actions are always taken by the Dialog Guide.

As the user’s history has to be regarded as well (UR 3.4.5), the User Model will be stored across sessions along with the user’s settings.
Part IV

Implementation
Chapter 8

Representing Calculation Data

8.1 Storing Enumeration Types

In many contexts in the implementation, variables or parameters hold information about one out of a finite set of alternatives.

Such values are often stored as integer numbers with every alternative being represented by a number. Throughout the IS4C system, named constants (public static final int) are used to represent alternatives.

While this approach improves readability and the names of the constants are scoped, the values themselves are still indistinguishable from other integers and thus cannot be type-checked or range-checked for proper use at compile time.

In the future, these integers will be replaced by the Typesafe Enum pattern [Blo01] to prevent unintentionally wrong use of enumerated values. In this pattern, every enumeration has a class of its own with a private constructor and all possible values are public constants initialised internally using the private constructor.

8.2 The Hierarchy of Mathematical Objects

8.2.1 CalcElement

CalcElement, accessible through the interface ICalcElement, is the common base class for all objects stored in a calculation. Every node in a CalcTree is a CalcElement or derived from it. CalcIterators (see 8.3 below) reference ICalcElements.

Data Stored

Apart from serving as a common reference, ICalcElement defines the attributes:
Type is the concrete type of the element, one of CALCCEL_FORMULA, CALCCEL_TACTIC, CALCCEL_CALCHEAD or CALCCEL_ASSUMPTION.

Visibility hints at whether to display the CalcElement to the user:
DISPLAY_VISIBLE_DEFAULT, DISPLAY_VISIBLE_HIGHLIGHT, DISPLAY_HIDDEN (invisible but can be displayed on request), DISPLAY_INVISIBLE (remaining invisible).

ViewStyle hints at how detailed the element will be displayed to the user. ViewStyles have been defined for CalcHead and will be added as alternative levels of detail for the other derived types are developed. The ViewStyle is chosen by the Dialog Guide to match the user’s level of expertise.

Rating stores the result of the Dialog Guide’s estimation of the user’s familiarity with the CalcElement in question, obtained from evaluating a query to the User Model. As estimating the user’s familiarity with the CalcElement is expected to be a complicated process, the results, once computed, are stored for later reference during the session.

It is debated whether the attributes Visibility, ViewStyle and Rating will stay with a data type used throughout the system or they will be moved to a derived type visible only to the Dialog Guide and the Presentation Layer.

Conversions
Regardless of the internal representation, a CalcElement provides the methods toSMLString and toMathML for delivering external representations of the data stored for the Math Engine and the Presentation Layer, respectively. As soon as appropriate parsers are implemented, the methods fromSMLString and fromMathML will be added. For the time being, the SML representation of the Math Engine is used internally for prototyping purposes and is likely to be replaced by a more efficient structured representation in the future.

8.2.2 Classes Derived from CalcElement
Apart from CalcHead, the classes derived from CalcElement are implemented basically as mere stubs, as the Dialog Guide does not know the mathematical meaning of the data intended to be contained, hence does not use or manipulate them.

Formula
Formula is presently implemented storing its internal data as a String holding the Math Engine’s internal representation.
Figure 8.1: The hierarchy of mathematical objects

At a later stage in development, the internal structure of a Formula will be made accessible to the user. To provide for this, a method copyTerm is added to the class to extract subterms (see section 8.2.3) for substitutions or similar purposes.

CalcHead

As a consequence of the decision to integrate subproblems into the main CalcTree (see section 7.2.3), CalcHead has to be derived from CalcElement.

According to UR 2.2.3 and UR 2.2.4, the CalcHead has the following additional attributes used during the Specifying Phase (see section 7.1.1):

Specification, consisting of Method, Problem and Theory, each being an identification referencing the Knowledge Base.

Formalization, consisting of Given, Relate, Where, Find: CalcHeadItem-Lists, which are basically lists of Formulas.

Every CalcHeadItem is a formula associated with a Status for feedback from the Math Engine. During the Specifying Phase, the Math Engine checks the CalcHead and provides specific feedback on every CalcHeadItem labelling it as one of CALCHEAD_CORRECT, CALCHEAD_INCORRECT, CALCHEAD_SYNTAXERROR, CALCHEAD_INCOMPLETE, CALCHEAD_MISSING or CALCHEAD_SUPERFLUOUS.
Again, the Dialog Guide does not know the mathematical meaning of the attributes of a CalcHead, but it uses the Status feedback to provide user guidance and decide about the completion of the Specifying Phase.

The ViewStyles VIEWSTYLE_FULL, VIEWSTYLE_SINGELINE, VIEWSTYLE_IN_CALC are introduced for the CalcHead.

Tactic
A Tactic is presently implemented as a stub, adding the attributes:

Name : To the user, every Tactic is identified by a String representing its name.

Description : Descriptive text according to UR 3.1.10.

and, as many Tactics use a theorem for rewriting:

TheoremSymbolic : A Formula as a symbolic representation of the theorem used for rewriting.

TheoremName

TheoremDescription

Assumption

Assumptions are presently implemented as a mere stub.

8.2.3 Subterms

Subterms of a Formula can be regarded as Formulas themselves. To navigate a Formula and select different subterms, the stub of a TermSelector class with methods goRoot, goLeft, goRight, goDown and goUp is provided for reference.

8.3 Iterators for Navigating the CalcTree: ICalcIterator

Methods offered by iterators include the following:

moveRoot moves an iterator to the start of the calculation, i.e. the root of the CalcTree. Note that the root element of a CalcTree is always a CalcHead (see section 7.2.3).

moveCalcHead moves the iterator to the CalcHead specifying the part of the calculation the iterator is presently referencing. Note that this is not necessarily the root CalcHead, as subproblems are integrated into one single CalcTree (see section 7.2.3).
moveUp and moveDown move the iterator to the previous or next element in the tree, respectively.

moveLevelUp and moveLevelDown move the iterator one nesting level out of one level deeper into the tree. These methods might seem redundant to moveCalcHead, but they are not as there may be mathematical structures other than subproblems branching the CalcTree.

getElement returns the CalcElement referenced by the iterator.

getTactic returns the Tactic being applied to the currently referenced Formula. getTactic supersedes the previous idea of a moveTactic method, as it has been decided to couple Formula and Tactic more tightly and to reference Tactics by referring to the Formula they are applied to.

All move methods return a boolean being true on success and false if there is no appropriate element to move to.
Chapter 9

Communicating with Other Components

Please note that naming is still very inconsistent due to the fact that the implementation evolved over a longer period of time involving cooperation with several members of the ISAC team. As development has stabilised in the meantime, a next step will include cleaning up the interfaces and implementing consistent naming.

9.1 Data Types Used for Communication

9.1.1 Events

Events are used whenever a component needs to notify other components of something happening without knowing which component will eventually handle the situation and without having to wait until processing the event has finished. Apart from receiving requests from the user, events are used for notifications about changes in the state of the CalcTree. Events are sent to components having registered as being interested in notifications using the Observer pattern. Events as used by ISAC resemble the Command pattern [GHJV95].

Changes in a Calculation: CalcChangedEventArgs

A CalcChangedEventArgs is sent by the Math Engine every time the calculation represented by a CalcTree changes to notify other components to look for updates. Normally, the Dialog Guide listens to CalcChangedEventArgs from the Math Engine and passes them on to the Presentation Layer, if appropriate. The Dialog Guide might decide not to pass the events if the changes affect only details of the calculation filtered away by the Dialog Guide.

Methods offered by CalcChangedEventArgs:
getLastUnchangedFormula returns an iterator pointing to the last Formula in the CalcTree not affected by the change to avoid examining parts of the tree which have not changed anyway.

isCompleted returns false, if the Math Engine has been requested to calculate more than one step to indicate that the CalcTree has changed but the request is not completed yet.

**Intervention by the User: UserAction**

Every time a user request cannot be handled by the Presentation Layer alone, a UserAction event is sent to registered listeners, which is usually the Dialog Guide. See appendix C for a list of requests recognised by the Dialog Guide.

getActionID returns an int identifying the user's intervention or request.

Several classes are derived from UserAction if additional information has to be passed:

UserActionOnInt carries an additional int, and is presently used for changing the ViewStyle of the CalcHead during the Specifying Phase.

UserActionOnIterator carries an iterator, e.g. to move the active formula.

UserActionOnCalcElement carries any CalcElement, e.g. to replace the active formula by a version edited by the user.

UserActionOnCalcHeadCompoundID refers to a part of a CalcHead during the Specifying Phase, e.g. to have a part of the CalcHead completed by the Math Engine.

**Controlling the Presentation Layer**

Although the naming is not adequate, UserAction objects are reused to send requests from the Dialog Guide to the Presentation Layer. Requests sent by the Dialog Guide include having the user edit a formula and activating or deactivating buttons on the user interface (see appendix C.3).

**9.1.2 Exceptions**

**Exceptions Handled by the Dialog Guide**

Being embedded into a distributed system, RemoteExceptions from Java-RMI have to be handled by most components, including the Dialog Guide.

NotInSpecificationPhaseException is thrown by the Math Engine and probably more exceptions originating from the Math Engine will be added in the future.
Exceptions Thrown by the Dialog Guide

DialogProtocolException is thrown if a request arrives out-of-order, e.g. calculating without having finished the Specification Phase. Every DialogProtocolException carries additional information about which UserAction caused the error and which Dialog Phase the Dialog Guide was in when the error occurred. Several exceptions have been derived from DialogProtocolException to indicate special situations:

DialogMathException is thrown if the Dialog Guide cannot recover from an exception thrown by the Math Engine.

DialogNotImplementedException is thrown if a requested feature is not yet implemented by the Dialog Guide.

DialogUnknownActionException is thrown if the Dialog Guide has been passed an unknown UserAction.

9.2 Interfaces used by the Dialog Guide

The interfaces IToCalc and IToUser are used for operating on the calculation. IToCalc is implemented both by the Math Engine and the Dialog Guide and IToUser is implemented both by the Dialog Guide and the Presentation Layer. This makes the Dialog Guide a wrapper for the Math Engine as seen from the Presentation Layer and a wrapper for the Presentation Layer as seen from the Math Engine. This approach made it possible to develop and test the Math Engine and parts of the Presentation Layer independent of the actual implementation of the Dialog Guide.

The interfaces DGuide and IToWorksheet are used for guided interaction with the user and are implemented by the Dialog Guide and the Presentation Layer, respectively.

9.2.1 Communicating Towards the Calculation: IToCalc

This interface contains two methods for accessing a calculation.

addDataChangeListener registers a component implementing the IToUser interface as listener for CalcChangedEvents to be notified of changes in the calculation.

iterator returns an iterator for navigating a calculation.

Other methods of this interface provide direct manipulation of a CalcTree. These methods are intended to be used by the Dialog Guide but not directly by the Presentation Layer, as user access to these operations is abstracted in the requests listed in appendix C. Therefore, calling these methods on the
Dialog Guide is deprecated. This part of the interface should be extracted to a separate interface implemented by the Math Engine only.

`getActiveFormula` returns a `CalcIterator` indicating the position in a calculation where modifications take place. This includes editing as well as the starting point for automatic calculation.

`moveActiveFormula` sets the position where modifications to the calculation will be applied.

`appendFormula` appends a new, empty `Formula` to a calculation.

`replaceFormula` replaces the active formula, if the correctness of the new `Formula` can be confirmed.

`fetchProposedTactic` asks for the next `Tactic` the Math Engine would apply.

`setNextTactic` sets the next `Tactic` to be applied.

`modifyCalcHead` applies modifications to a `CalcHead` and asks for feedback from the Math Engine.

`completeCalcHead` completes a `CalcHead` automatically.

`autoCalculate` calculates a number of steps automatically.

### 9.2.2 Communicating Towards the User: IToUser

This interface is implemented by the Dialog Guide to be able to be notified by the Math Engine about changes in the calculation as well as by the Presentation Layer to be able to be notified by the Dialog Guide.

The only method `calcChanged` is passed a `CalcChangedEvent` as parameter. In the present implementation, the Dialog Guide passes the `CalcChangedEvent` to all registered listeners without filtering or other processing taking place.

### 9.2.3 The Presentation Layer as Seen from the Dialog Guide: IToWorksheet

This interface is implemented by the Presentation Layer to provide a means for the Dialog Guide controlling the user interface as described in section 7.3.3. In addition to that, the Dialog Guide can request actions from the user, such as editing a formula.

The only method `doUIAction` is passed a `UserAction` to be processed by a user interface as parameter.
Both events and iterators are intercepted and modified by the DialogGuide.

**9.2.4 The Dialog Guide as Seen from the Presentation:**

The `DialogGuide` starts calculation by providing an identification of the user, a `Formalization` and `STARTFROM_EMPTY_WORKSHEET`, `STARTFROM_PROBLEM` or `STARTFROM_EXAMPLE` to indicate how much information will be available for user guidance during the Specifying Phase. Note that the user does not interact with this Dialog Guide until he chooses to start a calculation, so use cases UC 1.1.1 are handled outside the Dialog Guide.

The identification of the user is used to retrieve the appropriate `UserSettings` and `UserModel`. The `Formalization` - if non-empty - is passed on to the Math Engine during initialisation.

After successfully contacting a Math Engine, the Dialog Guide enters the Specifying Phase.
registerUIControlListener registers a component implementing the IToWorksheet interface as listener for UserActions controlling a user interface.

notifyUserAction is the central method for communicating with the Dialog Guide. A UserAction as listed in appendix C is passed to the Dialog Guide. The return value is true if the request has been accepted and will be processed and false if processing the request is denied. Processing a request can be denied according to the user’s privileges or the user’s experience, a decision the Dialog Guide takes after consulting the UserSettings and the UserModel. Note that denying a request is not an error condition from the Dialog Guide’s point of view. An error condition would be a request impossible due to the Dialog Guide’s state, such as not matching the current Dialog Phase. In this case, there has been an error in the communication between the Dialog Guide and the Presentation Layer and a DialogProtocolException is thrown.

Once a request is accepted, further action resulting from the request is communicated through UserActions sent by the Dialog Guide or CalcChangedEvents originating in the Math Engine.

```
+registerUIControlListener(IToWorksheet)
+notifyUserAction(UserAction)
```

```
+doUIAction(UserAction)
```

Figure 9.3: The Dialog Guide and the Presentation Layer communicating user interaction

### 9.3 Communicating with the UserModel

On the level of items in the Knowledge Base, such as Tactics, and the context of Dialog Atoms, the UserModel provides statistical data about previously recorded interactions with the user.

getTouchedCount returns the number of previously recorded interactions.

getSuccessCount returns the number of previously recorded successful interactions.
getTouchedLast returns the timestamp of the last interaction recorded.

getSuccessLast returns the timestamp of the last successful interaction recorded.

getTimeSpentAvg returns the average time the user spent with the interaction.

g getTimeSpentMax returns the maximum time the user spent with the interaction.

Statistical data are gathered as the Dialog Guide announces the start and end of every interaction between user and Knowledge Base.

startInteraction is called with identifications of the Knowledge Base item and the Dialog Atom used.

d endInteraction is called with an int indicating the success of the interaction. As there is no appropriate measure for the success of an interaction presently available, the values 0 and 1 will be used for prototyping.
Part V

Summary, Project Reflection, Future Developments
Chapter 10

Summary and Project Reflection

This chapter gives a summary of the work on this thesis, the achieved goals and remaining issues.

Working on a project so complex in both technological as well as co-operational respects was a major learning experience underestimated in the beginning.

With little specification available to start with, requirements analysis turned out to be a tedious but very creative and fruitful process.

While discussing in a fluctuating team of up to five persons put much stress on each teammember, the results proved to be worth the effort, the slowly evolving culture of documentation remaining the only hindrance.

The design of an overall system architecture with clear-cut interfaces provided a stable base for division of tasks, which allowed for independent work on specific fields.

Detailed development of the interfaces to the Math Engine and the Presentation Layer proceeded in close cooperation with the responsible teammembers.

In the last six months, a running prototype integrating all components proved feasibility of the concept and provided a testing platform.

10.1 System Architecture

The system architecture described in chapter 6.1.4 proved stable and is believed to cover most requirements of the evolving system.

Integration of all system components except for the not-yet-implemented authoring tools succeeded without major changes to the original concept. Java-RMI proved a suitable means for communication.

The design of the CalcElements Formula, Tactic and CalcHead is used successfully in the prototype, although without the capability of using MathML
representations of data and the handling of Assumptions in its beginning.

Cleanup of the interfaces, consistent naming and an identification scheme for CalcElements remain to be accomplished.

10.2 Dialog Guide

The design of the Dialog Guide proved suitable for integration into the system and all data flows relevant to solving a calculation pass the Dialog Guide. The Dialog Guide at the present stage of development reacts to user input at a high level of abstraction and distributes activity between the Math Engine and the Presentation Layer. Communication with the Presentation Layer has been tested successfully for the Solving Phase and more detailed development and testing will be needed for the Specifying Phase. The proper sequence of Dialog Phases is implemented with accurate checks for consistency. Although passing most data unaltered, the Dialog Guide's ability to intervene transparently was tested by filtering Tactics out of a Calculation Tree.

On the other hand, the inner structure of the Dialog Guide lacks some detail. A few of the actions listed in appendix C remain to be implemented. While they provide a useable set of building blocks for Dialog Atoms, there is no prototypical Dialog Atom implemented as yet. Developing a comprehensive set of Dialog Atoms is left to cooperations of $\mathcal{TSAC}$ in the field of didactics.

The ambitious goal of developing a language for describing Dialog Strategies as flexible sequences of Dialog Atoms remains out of reach until an identification scheme for CalcElements, a basic set of Dialog Atoms and a measure for the user's success in using a Dialog Atom become available.

Although provided for in the design and the available interfaces, the implementation of the Dialog Guide configuring the user interface as described in section 7.3.3 is only in its beginning. Seizing the some of the suggestions in [SBCO01] looks promising.
Chapter 11

Possible Future Developments

In this chapter, possible directions for future development of the ISAC system and the Dialog Guide are presented. Some may be implemented in future development, while others may remain utopic.

11.1 The ISAC System

• At present, knowledge used by the Math Engine for calculating results and knowledge accessible to the user through the Knowledge Browser is duplicated in the system. While being essentially the same in the sense of being compiled from the same source files, modifying both components to use the same data objects could help in further integration of the system.

• Exploring options for access to one calculation through several Presentation Layers simultaneously could open up new possibilities for group teaching and collaboration. The foundation is laid by the system designed to run the Presentation Layer and Dialog Guide components on different machines and communication with the Presentation Layer structured according to the Observer pattern.

• Moving the logic of modifying the appearance of objects according to the needs of individual users from the Dialog Guide to the objects themselves could make the system more manageable. This approach is inspired by Dinopolis’ concept of dynamic typing (see 6.2.4).

• For the Java part of ISAC, granularity of representation of mathematical data reaches the level of formulas. Modelling terms, operators and operands as objects could make them accessible to the user and open up new possibilities for interaction and learning.
11.2 The Dialog Guide

- Developing a language for describing concepts of user-interaction remains the next major goal involving research in fields of both computer science and didactics.

- The concept of the Dialog Guide could be extended beyond the scope of IΣ4C. The Dialog Guide knows nothing about mathematics but deals with user interaction in teaching of problem solving. Problem solving strategies based on Dialog Phases and Dialog Atoms could find applications even outside the field of mathematics.

- While IΣ4C’s system design clearly differentiates the roles of the user and the Math Engine, the Dialog Guide need not necessarily. Based on the concept of symmetrical Dialog Atoms, a Dialog Guide moderating interaction with learning taking place on both sides could be imagined.
Bibliography


[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.


Software technology, Graz, Austria, Oct 2003.


Part VI

Appendices
Appendix A

An example for reference

This example intended to illustrate interaction with IS4C is quoted from [Neu99].

IS4C features a new kind of calculations in applied mathematics, and it is an issue to make the novel functionality of this software as clear as possible. In order to meet this issue, an example is given to be referenced by the use-cases within this document. The example is taken from a calculus course at high schools; thus it should not pose problems to understand the underlying mathematics. Nevertheless the example covers all major features offered by IS4C.

A.1 Description, formalization and modeling phase

The description of an example\footnote{These terms are defined in the appendix of [GKN02a] 'List of Terms used in the IS4C-project'} may consist of text, formulas and figures:

\textit{Given a circle with radius }r = 7, \textit{inscribe a rectangle with length }u \textit{ and width }v. \textit{Determine }u \textit{ and }v \textit{ such that the rectangles area }A \textit{ is a maximum.}

![Figure A.1: Figure for the maximum example](image)

The initial step in solving such an example is, to construct a model from the description. The respective model looks like this, if all items are input:
given : [ Constants $r = 7$ ]
where : $0 \leq 7$
find X : [ Maximum $A$ ]
AdditionalValues $[u, v]$ ]
with : $A = 2uv - u^2 \land \left(\frac{u}{2}\right)^2 + \left(\frac{v}{2}\right)^2 = r^2 \land \forall A', u', v'. A' = 2u'v' - (u')^2 \land \left(\frac{u'}{2}\right)^2 + \left(\frac{v'}{2}\right)^2 = r^2$
relate : \[
\begin{align*}
A &= 2uv - u^2, \\
\left(\frac{u}{2}\right)^2 + \left(\frac{v}{2}\right)^2 &= r^2
\end{align*}
\]

The boxes mark the items meant for input by the user, whereas the surrounding information is provided by the system and serves user guidance. If the model is perfect, \textsc{Is4c} can solve the example autonomously.

In order to provide user guidance already in the model phase, each example is accompanied by a formalization prepared by an author and normally hidden from the user:

\[
F_I \equiv ( \{ r = 7 \}, \{ A, [u, v] \}, \{ 0 \leq \frac{u}{2} \leq r, \{ A = 2uv - u^2, \left(\frac{u}{2}\right)^2 + \left(\frac{v}{2}\right)^2 = r^2 \} \) \\
F_{II} \equiv ( \{ r = 7 \}, \{ A, [u, v] \}, \{ 0 \leq \frac{v}{2} \leq r, \{ A = 2uv - u^2, \left(\frac{u}{2}\right)^2 + \left(\frac{v}{2}\right)^2 = r^2 \} \) \\
F_{III} \equiv ( \{ r = 7 \}, \{ A, [u, v] \}, \{ 0 \leq \alpha \leq \frac{\pi}{2}, \{ A = 2uv - u^2, \frac{v}{2} = r \sin \alpha, \frac{u}{2} = r \cos \alpha \} \)
\]

In this case the formalization comprises three variants, $F_I, F_{II}, F_{III}$, which presumably cover all possibilities students would consider in a particular course. All of such formalizations for one example together are called ‘formalization’ in the sequel.

Given such a formalization and a specification (see below), \textsc{Is4c} can solve an example autonomously and in stepwise interaction down to the result.

If \textsc{Is4c} is being used to solve an example unknown to the system (i.e. without a hidden specification and formalization prepared by an author) \textsc{Is4c} cannot provide user guidance at the beginning of this phase (see UC 1.1.4). In particular, the items Constants, Maximum and AdditionalValues have to be found in the theory Descript.thy.

### A.2 Knowledgebase and specification phase

The knowledge base comprises three parts, theories, problems and methods.

Theories contain the knowledge deduced from axioms and definitions by formal proof (done by the interactive theorem prover Isabelle). For the example at hand knowledge is prepared like the following:

<table>
<thead>
<tr>
<th>Theory Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descript.thy</td>
<td>Knowledge base for the example</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is4cExample</td>
<td>Example problem</td>
</tr>
</tbody>
</table>

102
theory 'calculus'

consts
d_d :: "[real, real] => real"

rules
diff_sum  "d_d bdv (u + v) = d_d bdv u + d_d bdv v"
diff_prod  "d_d bdv (u * v) = d_d bdv u * v + u * d_d bdv v"
diff_quot  "Not (v = 0) ==> (d_d bdv (u / v) =
             (d_d bdv u * v - u * d_d bdv v) / v ^ 2)"

Problems capture the aspect of application of knowledge.

The example at hand shall be described by the problem {calculus, optimization} (note the reverse order w.r.t. the hierarchy above, which seems more usual in many cases, e.g. {linear, univariate, equation}), and will be broken down into the subproblems {make, function, on_interval, for_maximum,
differentiate, function] and [tool, find_values]. The root-problem of the example looks like:

Solve_problem [maximum, calculus, optimization]
given : [ Constants fix ]
where : map ($0 \leq $) fix
find : [ Maximum m ]
        AdditionalValues vs
with : let $x_1 = \{ m \} \cup \{ vs \} \cup (\text{Vars } rs)$;
     $x_2 = \text{map primed } x_1$;
     in map (op\&\&)$ rs \_ \& \_ \$
     \forall $x_2 \$. ($\lambda x_1 \$. map (op\&\&)$ rs \$) \_ \$ $x_2 \$
relate : rs

This problem is matched with the formalization and yields the model shown above.

Methods describe the algorithms solving the problems. The method solving the example calls the subproblems mentioned:

Script Maximum_value (fix ::bool list) (m ::real) (rs ::bool list)
        (v ::real) (itv ::real set) (err ::bool) =
         (let
           e_ = (hd o (filter (Testvar m_))) rs_;
           t_ = (if #1 < Length rs_ then (Subproblem (Reals,[make, function],no_met) [m_, v_, rs_])
                 else (hd rs_));
           mx_ = Subproblem (Reals,[on_interval, for_maximum, differentiate, function], maximum_on_interval) [ t_, v_, itv_ ]
in (Subproblem (Reals,[tool, find_values],find_values)
    [ mx_, (Rhs t_), v_, m_, (dropWhile (ident e_), rs_)])

An example is given a specification ('it is specified') by three pointers into each of the three parts of the knowledge base, i.e. a pointer to a theory, to a problem and to a method. For the example this is (Differentiate, [calculus, optimization], Maximum_value).

A.3 Interaction on the worksheet and the browsers

Within a calculation the centre of interaction with the user is a so-called worksheet. At certain points the user may want to view the knowledge base in a so-called browser-window and/or select some knowledge, or the dialog presents such a window.
In the modeling phase, the user inputs formulas on the worksheet, and on the worksheet they get most of the feedback. After a while the worksheet could look like this:

\[
\text{Solve}_\text{problem} [\text{maximum, calculus, optimization}]
\]

\textbf{Model}

\begin{itemize}
  \item \textit{given:} \ [\text{Constants } r = 7 ]
  \item \textit{where:} \ [0 \leq r ]
  \item \textit{find:} \ [\text{Maximum, AdditionalValues } [u, v] ]
  \item \textit{relate:} \ [A = 2uv, \ (\frac{u}{2})^2 + (\frac{v}{2})^2 = r^2, \ \frac{v}{2} = r \sin \alpha ]
\end{itemize}

where \textit{Solve}_\text{problem} [\text{maximum, calculus, optimization}], \textit{Constants}, \textit{Maximum}, \textit{AdditionalValues}, and \(0 \leq r\) have been supplied by the system, and several items are marked: \textit{Maximum} with 'missing', \(A = 2uv\) with 'incorrect', and \(\frac{v}{2} = r \sin \alpha\) with 'superfluous'. Regarding this kind of feedback users may successfully complete modeling; if they are not capable to do it, they just hit a 'go-on' button and \textit{TSSC} gets the correct model from the hidden formalization and specification. For the following let us assume, that the model is completed correctly:

\[
\text{Solve}_\text{problem} [\text{maximum, calculus, optimization}]
\]

\textbf{Model}

\begin{itemize}
  \item \textit{given:} \ [\text{Constants } r = 7 ]
  \item \textit{where:} \ [0 \leq r ]
  \item \textit{find:} \ [\text{Maximum, AdditionalValues } [u, v] ]
  \item \textit{relate:} \ [A = 2uv - u^2, \ (\frac{u}{2})^2 + (\frac{v}{2})^2 = r^2, \ \frac{v}{2} = r \sin \alpha ]
\end{itemize}

\textbf{Specification}

\begin{itemize}
  \item \textit{Theory:}
  \item \textit{Problem:}
  \item \textit{Method:}
\end{itemize}

The superfluous item \(\frac{v}{2} = r \sin \alpha\) does not matter, below there are the specification-fields to be determined (i.e. input by the user) within the specification phase.

The specification phase requires data from the knowledge base in general. In the case of the examples model above the decision is required, which kind of problem the given model can be matched with: \textit{[linear, optimization]}, \textit{[calculus, optimization]}, or some other problem. The information necessary for this decision can be found in the hierarchy of problems.

Learners can browse the hierarchy of problems (theories, methods) and they can apply the problem (theory, method); the latter transfers the problem (theory, method) to the respective field below the model on the worksheet.

For instance, applying the problem \textit{[linear, optimization]} would cause this display on the worksheet:
Solve problem [maximum, calculus, optimization]

Model

given: [Constants \( r = 7 \)]

where: \( 0 \leq r \)

find: [Maximum, Additional Values \([u, v]\)]

relate: \[ A = 2uv - u^2, \left(\frac{u}{2}\right)^2 + \left(\frac{v}{2}\right)^2 = r^2, \frac{v}{2} = r \sin \alpha \]

Specification

Theory:

Problem: [linear, optimization]

Method: [make_fun, by_elimination]

where some items in the model would be marked with some error-feedback. For instance, applying a theory even might cause the feedback 'syntax-error' when the model contains function constants not defined in the theory or in one of the theories parents.

Browsing problems (when started from a model on the worksheet) matches the problem selected with the model on the worksheet, i.e. on the browser-window the same feedbacks are given as for the model on the worksheet described on p.105.

A.4 The solving phase and subproblems

Before we describe LS4Cs features for the solving phase, we fix one correct model and one specification (out of several possible ones) as follows:

Solve problem [maximum, calculus, optimization]

Model

given: [Constants \( r = 7 \)]

where: \( 0 \leq r \)

find: [Maximum \( A \), Additional Values \([u, v]\)]

relate: \[ A = 2uv - u^2, \left(\frac{u}{2}\right)^2 + \left(\frac{v}{2}\right)^2 = r^2, \frac{v}{2} = r \sin \alpha \]

Specification

Theory: Reals

Problem: [maximum, calculus, optimization]

Method: [make_fun, by_elimination]

The following worksheet shows the whole calculation without the respective models and specifications; this short-presentation could stem from an interaction, where the user is merely interested in the result and let the system calculate autonomously:

Solve problem [maximum, calculus, optimization]

1. SubProblem (DiffAppl, [make, function])

1. solve_univariate \( \left(\frac{u}{2}\right)^2 + \left(\frac{v}{2}\right)^2 = r^2 \) \( u \)

1'. \( L = \left[ u = 2\sqrt{r^2 - \left(\frac{u}{2}\right)^2}, u = -2\sqrt{r^2 - \left(\frac{u}{2}\right)^2} \right] \)
2. \( L_1 = \left[ u = 2\sqrt{r^2 - \left(\frac{v}{2}\right)^2} \right] \)

1'. \( A_1 = 2 \cdot 2\sqrt{r^2 - \left(\frac{v}{2}\right)^2} \cdot v - v^2 \)

2. Sub Problem \((\text{DiffApp}, \text{for interval}, \text{for maximum, differentiate, function})\)

1. \( \frac{d}{dv} \left( 2 \cdot 2\sqrt{r^2 - \left(\frac{v}{2}\right)^2} \cdot v - v^2 \right) = \)

1'. \( A'_1 = 2\sqrt{r^2 - \left(\frac{v}{2}\right)^2} - \frac{v^2}{2\sqrt{r^2 - \left(\frac{v}{2}\right)^2}} - 2v \)

2. \( \text{solve univariate} \left( 2\sqrt{r^2 - \left(\frac{v}{2}\right)^2} - \frac{v^2}{2\sqrt{r^2 - \left(\frac{v}{2}\right)^2}} - 2v = 0 \right) v \)

2'. \( L = [v = 234.567] \)

3. Sub Problem \((\text{DiffApp}, [\text{find values, tool}])\)

3'. \([u = 123.456, v = 234.567]\)

\([u = 123.456, v = 234.567]\)
Appendix B

ISACs tactics

Init Proof_Hid (dialogmode, formalization, specification) transfers the arguments to the math engine, the latter two in order to solve the example automatically. The tactic is not intended to be used by the student; it generates a proof tree with an empty model.

Init_Proof generates a proof tree with an empty model.

Model_Problem problem determines a problemtype (eventually found in the hierarchy) to be used for modeling.

Add_Given, Add_Find, Add_Relation formula inputs a formula to the respective field in a model (necessary as long as there is no facility for the user to input formula directly, and not only select the respective tactic plus formula from a list).

Specify_Theory theory, Specify_Problem problem, Specify_Method method specifies the respective element of the knowledgebase.

Refine_Problem problem searches for a matching problem in the hierarchy below 'problem'.

Apply_Method method finishes the model and specification phase and starts the solve phase.

Free_Solve initiates the solve phase without guidance by a method.

Rewrite theorem applies 'theorem' to the current formula and transforms it accordingly (if possible – otherwise error).

Rewrite_Asm theorem is the same tactic as 'Rewrite', but stores an eventual assumption of the theorem (instead of evaluating the assumption, i.e. the condition)

Rewrite_Set ruleset similar to 'Rewrite', but applies a whole set of theorems ('ruleset').
**Rewrite Inst (substitution, theorem), Rewrite Set Inst (substitution, ruleset)** similar to the respective tactics, but substitute a constant (e.g. a bound variable) in 'theorem' before application.

**Calculate operation** calculates the result of numerals w.r.t. 'operation' (plus, minus, times, cancel, pow, sqrt) within the current formula.

**Substitute substitution** applies 'substitution' to the current formula and transforms it accordingly.

**Take formula** starts a new sequence of calculations on 'formula' within an already ongoing calculation.

**Subproblem (theory, problem)** initiates a subproblem within a calculation.

**Function formula** calls a function, where 'formula' contains the function name, e.g. 'Function (solve 1 + 2x + 3x^2 = 0 \( x \))'. In this case the modelling and specification phases are suppressed by default, i.e. the solving phase of this subproblem starts immediately.

**Split And, Conclude And, Split Or, Conclude Or, Begin Trans, End Trans, Begin Sequ, End Sequ, Split Intersect, End Intersect** concern the construction of particular branches of the proof tree; usually suppressed by the dialog guide.

**Check elementwise assumptions** w.r.t. the current formula which comprises elements in a list.

**Or to List** transforms a conjunction of equations to a list of equations (a questionable tactic in equation solving).

**Check postcond**: check the current formula w.r.t. the postcondition on finishing the respective (sub)problem.

**End Proof** finishes a proof and delivers a result only if 'Check postcond' has been successful before.
Appendix C

Requests Recognised or Issued by the Dialog Guide

This appendix lists constants identifying requests recognised or issued by the Dialog Guide in user interaction. These represent the present base for the development of Dialog Atoms.

C.1 Requests Recognised During the Specifying Phase

UI_SPECIFY_TRY_MATCH : try to find a Problem matching the items of the Model entered so far.

UI_SPECIFY_TRY_REFINE : try to find a more specific Problem.

UI_SPECIFY_CHANGE_VIEW : change the view on the CalcHead.

UI_SPECIFY_COMPLETE_CALCHEAD : have the CalcHead completed by the Math Engine.

UI_SPECIFY_COMPLETE_METHOD : have the Method field of the CalcHead completed by the Math Engine.

UI_SPECIFY_COMPLETE_THEORY : have the Theory field of the CalcHead completed by the Math Engine.

UI_SPECIFY_COMPLETE_PROBLEM : have the Problem field of the CalcHead completed by the Math Engine.

UI_SPECIFY_COMPLETE_GIVEN : have the Given field of the CalcHead completed by the Math Engine.

UI_SPECIFY_COMPLETE_FIND : have the Find field of the CalcHead completed by the Math Engine.
UI_SPECIFY_COMPLETE_RELATE : have the Relate field of the CalcHead completed by the Math Engine.

UI_SPECIFY_CHECK_CALCHEAD : get feedback about the CalcHead’s completeness and consistency.

UI_SPECIFY_CALCULATE_1 : leave the Specifying Phase, enter the Solving Phase and calculate the first step of the Method.

UI_SPECIFY_CALCULATE_ALL : leave the Specifying Phase, enter the Solving Phase and calculate until a final result is reached.

C.2 Requests Recognised During the Solving Phase

UI_SOLVE_CALCULATE_1 : calculate one more step.

UI_SOLVE_CALCULATE_ALL : calculate until a final result is reached.

UI_SOLVE_CALCULATE_SUBPROBLEM : calculate until the current subproblem is solved.

UI_SOLVE_EDIT_ACTIVE_FORMULA : request editing the currently active formula.

UI_SOLVE_EDIT_ACTIVE_FORMULA_COMPLETE : notify that editing the currently active formula is finished. This implies a request for updating the CalcTree.

UI_SOLVE_APPEND_USER_FORMULA : request inserting a new formula after the currently active formula, making the new formula the currently active formula. At present, this does not imply editing the formula and must be requested separately.

UI_SOLVE_MOVE_ACTIVE_FORMULA : request making the referenced formula the currently active formula.

UI_SOLVE_GET_PROPOSED_TACTIC : ask for the Tactic the Math Engine would apply to the active formula.

UI_SOLVE_GET_APPLICABLE_TACTICS : ask for a list of Tactics applicable to the active formula.

UI_SOLVE_SET_NEXT_TACTIC : set the Tactic to be applied to the active formula.

UI_SOLVE_HELP_ENTERING_FORMULA : when entering a formula during the solving phase, request help on entering the formula.
UI_SOLVE_SHOW_ASSUMPTIONS : show which assumptions are made at the current point of calculation.

UI_SOLVE_SHOWDETAILS : request more detailed information about the currently referenced element.

C.3 Requests Issued by the Dialog Guide

UI_D0_EDIT_FORMULA : request editing the currently active formula.

UI_D0_APPEND_FORMULA : request appending a new, empty formula.

UI_D0_ACTIVATE : activate a UI element, such as a button or menu item.

UI_D0_DEACTIVATE : deactivate a UI element.

UI_D0_DETACH : the Dialog Guide has detached the listener for UI control and will not send any more requests. Because being registered as a Data Change Listener is independent of that, CalcChangedEvents will still arrive!