Prototyping “Systems that Explain Themselves” for Education

Alan Krempler
MMM-Kommunikation
Graz, Austria
alan.krempler@mmm-komm.at

Walther Neuper
University of Technology
Graz, Austria
wneuper@ist.tugraz.at

“Systems that Explain Themselves” appears a provocative wording, in particular in the context of mathematics education — it is as provocative as the idea of building educational software upon technology from computer theorem proving. In spite of recent success stories like the proofs of the Four Colour Theorem or the Kepler Conjecture, mechanised proof is still considered somewhat esoteric by mainstream mathematics.

This paper describes the process of prototyping in the *ISAC* project from a technical perspective. This perspective depends on two moving targets: On the one side the rapidly increasing power and coverage of computer theorem provers and their user interfaces, and on the other side potential users: What can students and teachers request from educational systems based on technology and concepts from computer theorem proving, now and then?

By the way of describing the process of prototyping the first comprehensive survey on the state of the *ISAC* prototype is given as a side effect, made precise by pointers to the code and by citation of all contributing theses.

1 Introduction to a Never Ending Story . . .

. . . a story, where no end is in sight for realising specific ideas how to support learning and teaching mathematics. In the early nineties of the last century visionary minds were required to come up with the idea of using concepts and technologies from (computer) theorem proving (TP)[1] to build educational math software; these minds were Dines Bjørner[2], Peter Lucas[3] and Bruno Buchberger[4], who jointly planned such a project within the framework of UNU/IIST[5]. TP work done in Europe and in the US since the fifties was only known to a few specialists. But it was considered trustworthy enough to state the main idea for an R&D project: if mathematics is the science of reasoning (and not only of calculating), then technology implementing such reasoning must be used as a base. Presently there are two exceptions underpinning their educational tools for formal mathematics with TP, MathToys [3] and 4ferries [1], the former just an experiment and the latter already a commercial firm (tools for geometry, where some of them also integrate TP, are not mentioned here).

Funding of the UNU/IIST project failed and respective material was used to start the *ISAC*-project. At that time computer algebra systems became readily available, and *ISAC*’s decision for TP has been

[1]In this paper TP abbreviates the academic discipline as well as the products this discipline develops, proof assistants and automated provers frequently included in the former.

[2]`https://en.wikipedia.org/wiki/Dines_Bjoerner`

[3]`https://de.wikipedia.org/wiki/Peter_Lucas_(Informatiker)`

[4]`https://en.wikipedia.org/wiki/Bruno_Buchberger`

[5]`http://www.iist.unu.edu/`

questioned from beginning and respective argumentation became more pinch-hitting over time. Taking apart Isabelle was much easier as is today; there was no Isar proof language [49], all commands could be executed on the command line and the Emacs, driven by a specific interface [8], was considered insurmountable. However, numerals looked strange (`#1, #2, ...`)[6], so even work on reals in floating point representation and complex numbers was started and removed again later[7]— one of the several efforts made superfluous by Isabelle's rapid development.

In the early phases of the *ISAC* project efforts required for developing a usable tool have been drastically underestimated. Early field tests at technical schools were successful [33, 38, 39] in that they confirmed the design principles; but the test also showed clearly that very much work would be required to arrive at a system, which does not distract students from learning. From the very beginning much code came from students' diploma theses, master's theses and projects. Contribution of students mostly were inspiring, and those from their supervisors invaluable: the latter now form an interdisciplinary network [6] ready to constitute competent project teams.

Of course, there were various attempts at fund raising; most of them failed[8]: Austrian FWF rejected proposals as to far off basic research, SparklingScience rejected proposals because referees doubted that yet another computer software would improve math education; FP7 rejected an internationally well staffed proposal, because 2/3 of the money was planned for development and only 1/3 for pedagogical evaluation (instead the other way round). The problem is that experts in engineering and mathematics education still don't know TP and thus cannot judge respective promises.

The paper roughly follows the timeline of *ISAC*'s development. The mathematics engine §2 was developed first; §2.1 shows how much *ISAC* benefits from Isabelle while §2.2 introduces an original contribution of *ISAC* (enabling the system to propose a next step when the student gets stuck). §2.3 introduces *ISAC*'s universe of mathematics knowledge and §2.4 explain why Isabelle's great simplifier is *not* used. The front-end's development §3 has been started later, following a thorough design phase in 2002/2003. §3.1 describes how *ISAC*'s front-end communicates with the mathematics engine based on Isabelle, §3.2 and §3.3 explains why *ISAC* can be called self-explanatory (by meeting users' expectations and by specific dialog guidance). Most recent requirements analysis at technical faculties led to specific support for a specification phase §3.4. §4 collects current issues in R&D, in particular front-end technologies §4.1 and transition to professional development §4.2. The final conclusions are given in §5, particularly justifying the aim towards "systems that explain themselves".

2 The Mathematics Engine

The mathematics engine's (abbreviated to math-engine in the sequel) repository has been separated from the front-end, because respective developments run separated after the interface in between had stabilised[9]. The math-engine is designed following concepts of TP and implemented re-using technology from TP.

[6]`http://www.ist.tugraz.at/projects/isac/publ/mat-eng-de.pdf`

[7]`https://intra.ist.tugraz.at/hg/isa/rev/ab57fbfcfffd`

[8]*ISAC*'s web page `http://www.ist.tugraz.at/isac/` reflects lack of funding and only serves internal information for developers.

[9]`https://intra.ist.tugraz.at/hg/isa/file/2f1b2854927a/src/Tools/isac/Interpret/mathengine.sml`

2.1 Isabelle's Components Used in *ISAC*

ISAC adopts as much of the TP Isabelle's [41] concepts as appropriate for engineering mathematics, and uses as much of Isabelle's code as well. This is in more detail:

- terms of simple typed λ-calculus, type inference and parsing of terms — the basis of modelling mathematics.
- logical contexts; it took a long time until the "everything local" principle pervaded Isabelle (and this process is still not finished); in 2004 Isabelle's tactics started to understand contexts, *ISAC* took up this concept much later [28]. The most significant benefit for *ISAC* is, that formulas input by the user need not be complicated by type annotations, because types are inferred from the context.
- typed matching for simplification; *ISAC*, however, has a specific simplifier for a certain reason: traces of simplification are surprisingly long, too long to "explain" to a student what is going on. But if one groups the rewrite rules, one can get steps fairly close to hand-written calculations, for instance when simplifying fractions [14].
- automated provers for three tasks:
  1. Check pre-conditions of formal specifications for problems and of guards for methods
  2. Derive a formula input by a student from the logical context (or reject the input); for this purpose Isabelle's provers combined with proof reconstruction by Metis [10] seem appropriate. Construction of most solutions for engineering problems is simple forward reasoning. Since this is mostly within normalising term rewriting, correctness of input is decidable in most cases.
  3. Check post-conditions upon completion of calculations (also in sub-problems).

  Only the first two tasks are implemented; they still use only the simplifier.
- knowledge management by theories; engineering students are *not* expected to create *new* mathematics knowledge, rather they are expected to use it efficiently and with understanding. For that purpose *ISAC* tries to support investigative access to theories; these are views on theories are additional to Isabelle's, see §2.3
- all theories imported by multivariate analysis; this was the appropriate knowledge base for prototyping, which will be extended to various engineering disciplines. Actually, *ISAC* is able to make domains, prepared by Formal Methods, interactively accessible, for instance, interactions under security protocols by rewriting [45]. The wide range of applications will enforce to re-organise theory imports (which are presently blocked by the interface, see §4.1 below)

The major difference between Isabelle and *ISAC* is in implementation of proofs versus calculations. The reason is that problem solving in engineering mathematics, in particular as done in academic education, is very different from proving; we will elaborate on this in §3.2 below. However, since such calculations are constructed by simple forward reasoning, results of calculations in *ISAC* are correct by construction.

ISAC holds calculations in a `Ctree`[10] with two kinds of nodes: (1) for steps in forward reasoning together with formal justification[11], and (2) for formal specifications[12],. The root of the tree is a specification, specifications in leaves hold sub-problems. The `type calcstate`[13] is given by a `ctree` paired with a pointer to the current position in the tree.

This design is very different from Isabelle's `structure Proof_Node` and `datatype state`; respective design decisions of *ISAC* need to be revised when *ISAC* is going to adopt Isabelle/PIDE as

[10]`https://intra.ist.tugraz.at/hg/isa/file/2f1b2854927a/src/Tools/isac/Interpret/ctree.sml`

[11]`https://intra.ist.tugraz.at/hg/isa/file/2f1b2854927a/src/Tools/isac/Interpret/ctree-basic.sml#l41`

[12]`https://intra.ist.tugraz.at/hg/isa/file/2f1b2854927a/src/Tools/isac/Interpret/ctree-basic.sml#l27`

[13]`https://intra.ist.tugraz.at/hg/isa/file/2f1b2854927a/src/Tools/isac/Interpret/calchead.sml#l89`

discussed in §4.1 below.

2.2 The Lucas-Interpreter Extends TP

In principle, there is no general method to find proofs in TP [13]. Nevertheless, the development of *ISAC* has been started with the requirement that an educational system must model a process of problem solving such that student and system cooperate on equal terms: both have some knowledge (and tell it on request), both can check the partner's steps, both know how to do a next step — and both can change roles any time (as, for instance, possible with chess software).

So in 2001 *ISAC*'s development started with a programming language and a respective interpreter[14]. At that time there was no function package [24] in Isabelle, so this has been developed from scratch under supervision of Peter Lucas[15] and named "Lucas-Interpreter" later. For instance, the program guiding user-interaction of the example shown in Fig.3 on p.97 is the following.

```
"Script Biegelinie (l::real) (q::real) v::real) (b::real=>real) (s::bool list) =" ^
"  (let (funs::bool list) =                                                 " ^
"            (SubProblem (Biegelinie, [vonBelastungZu, Biegelinien],        " ^
"                          [Biegelinien, ausBelastung])                     " ^
"                          [REAL q, REAL v]);                               " ^
"       (equs::bool list) =                                                 " ^
"            (SubProblem (Biegelinie, [setzeRandbedingungen, Biegelinien],  " ^
"                          [Biegelinien, setzeRandbedingungenEin])          " ^
"                          [REAL l, BOOL_LIST funs, BOOL_LIST s]);          " ^
"       (cons::bool list) =                                                 " ^
"            (SubProblem (Biegelinie, [LINEAR, system], [no_met])           " ^
"                          [BOOL_LIST equs, REAL_LIST [c, c_2, c_3, c_4]]); " ^
"       B = Take (lastI funs);                                              " ^
"       B = ((Substitute cons) @@                                           " ^
"             (Rewrite_Set_Inst [(bdv, v)] make_ratpoly_in False)) B        " ^
" in B)                                                                     "
```

The program text is parsed into an Isabelle term according to §2.1. Several syntactical details are imposed by the parser, for instance the capital letters of `REAL` or `LINEAR` avoiding name clashes with identifiers in the parser's (still global!) name space. The program's arguments `l...s` get the values from data prepared behind the example (no.`7.70` in the `Example browser` left in Fig.3 on p.97) checked by the precondition of a formal specification (shown in detail on p.100). The program body breaks down the solving process into three `SubProblems`. A sub-problem is determined by two arguments, (1) pointers into *ISAC*'s knowledge-base (see §3.4 below) and (2) the list of arguments. Further details can be found in [34, 36].

The interpreter works on the program term like a debugger: at each tactic[16] (e.g. `SubProblem` ...`Substitute` in the program above) control is handed over to the user[17]. [35] gives an example for investigating programs, but this concept works for mathematics as well. The challenge for the Lucas-Interpreter is to find a derivation from the previous step and the logical context, in case the user inputs a step[18]. Derivation preserves logical consistency. For engineering mathematics this works surprisingly well: a method can be broken down into parts which are evaluated by normalising term rewriting systems.

Up to now Isabelle's function package has been well elaborated and programming is convenient — and convenient programming is important for *ISAC*, because wide-spread usage can only be expected, if

[14] https://intra.ist.tugraz.at/hg/isa/file/2f1b2854927a/src/Tools/isac/Interpret/script.sml
[15] https://de.wikipedia.org/wiki/Peter_Lucas_(Informatiker)
[16] https://intra.ist.tugraz.at/hg/isa/file/2f1b2854927a/src/Tools/isac/Interpret/tactic.sml#l59
[17] https://intra.ist.tugraz.at/hg/isa/file/2f1b2854927a/src/Tools/isac/Interpret/script.sml#l12
[18] https://intra.ist.tugraz.at/hg/isa/file/2f1b2854927a/src/Tools/isac/Interpret/script.sml#l14

everyone interested in interactive course material can implement such content with efforts comparable with programming in Mathematica. So, what is called `Script` above shall become `partial_function` (termination can*not* be proven for whole classes of input data) and evaluation shall re-use mechanisms of Isabelle's code generator [17].

2.3 Distinguished Knowledge

Software tools in formal methods, including TP, are designed to *develop* formal models and to *prove* properties of these. In contrast to that, *ISAC* is designed to interactively specify appropriate models from a given collection, to make them transparent, to interactively construct solutions, and *not* to develop new knowledge or to prove anything. Therefore *ISAC* provides a specific structure of knowledge and distinguishes three aspects of knowledge.

The example on p.92 shows how programming is related to this structure, particularly how `SubProblems` are determined by three different aspects. For the first

```
SubProblem (Biegelinie, [vonBelastungZu, Biegelinien], [Biegelinien, ausBelastung])
```

there is the *theory* `Biegelinie`, the *problem* referenced by `[vonBelastungZu, Biegelinien]` in a tree and the *method* referenced by `[Biegelinien, ausBelastung]` in another tree. So there are three aspects of knowledge separated into three different data structures:

Theories concern the *deductive aspect* exactly as provided by Isabelle. Although Isabelle theories represent a directed acyclic graph (DAG), *ISAC* provides a tree view for the purpose of uniformity with the other structures. This shortcut needs to be revised as soon as respective views will become available in Isabelle/jEdit.

Problems concern the *aspect of application*, which is represented by formal specifications, i.e. an assembly of input data, pre-conditions on these, ouput data and a post-condition relating input and output. Specifications are collected in a tree, which allows a simple kind of problem refinement down along branches, first applied to classes of equations for assigning the right methods solving specific classes [27].

Methods concern the *algorithmic aspect* represented by the programming language introduced above by an example. A method associates a program with a guard, which has the same structure as a model (intoduced in §3.4 below). Methods are preliminiarily stored in a tree, but in contrast to problems here is no specific reason for this data structure.

ISAC uses Isabelle's theory management also for defining problems and methods. However, the structure of problems and methods is different from theories. For instance, the problem (i.e. the formal specification) of a linear equation is polymorphic for integers (i.e. rings), for rational, real and complex numbers (i.e. fields), for vector spaces, etc. Such a problem can be defined somewhere in Isabelle's DAG of theories as soon as a predicate `is_linear` is defined and a post-condition can be formulated.

So storing problems and methods in respective trees proceeds independently from theory evaluation. Originally a global reference variable stored these trees, later Isabelle enforced to use `Unsynchronized.ref` and finally parallelisation of theory evaluation enforced to use the functor `Theory_Data` [53] in Isabelle/ML.

The above separation of knowledge appears to anticipate current R&D in formal methods [40]: at least a distinction between definition of language elements (as in theories) and definition of re-usable specifications (as in problems) becomes apparent. Users' interactions on the three aspects of knowledge will be discussed in §3.4 below.

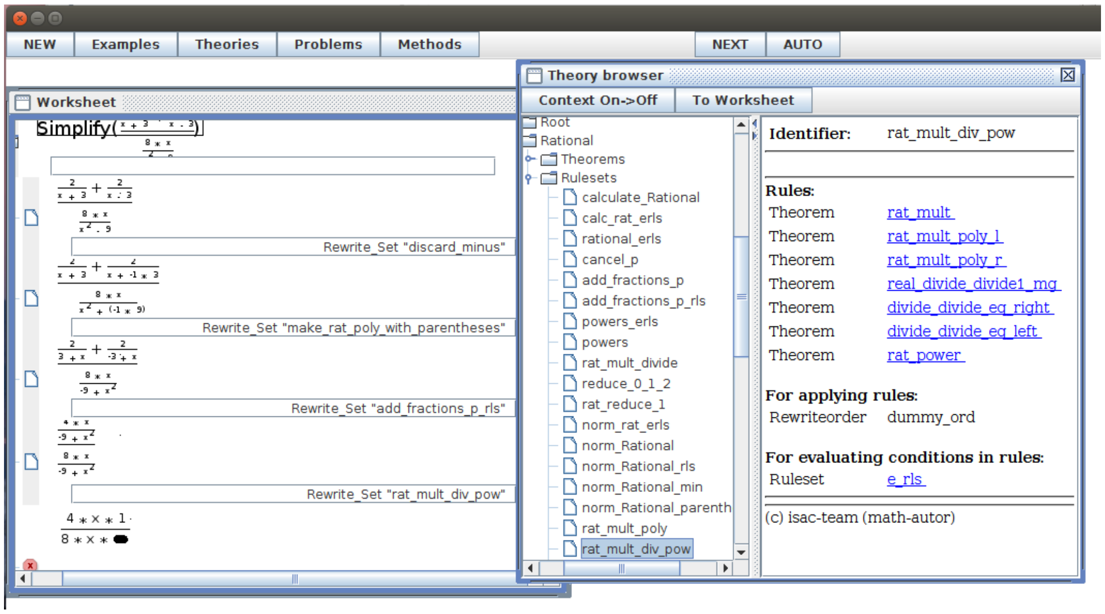
2.4 A Specific Rewrite Engine

Isabelle’s simplifier was a highly elaborated and complicated component already in 2000. So it was hard to decide, how to adapt it to *ISAC*’s requirements. When early design revealed, how much Isabelle’s and *ISAC*’s requirements differ, the decision was for developing the simplifier from scratch and for re-using only Isabelle’s `Pattern.match`. Presently *ISAC* requires the simplifier for three purposes.

1. **Evaluate programs** during Lucas-Interpretation (§2.2), in particular code concerned with list processing. Isabelle provides an elaborated collection of functions for this purpose, which need to be evaluated efficiently. *ISAC* would have gotten this for free if Isabelle’s code generator [17] would have existed at the time of development.

2. **Evaluate preconditions** and postconditions of specifications after instantiation with the data of a particular example. *ISAC* uses Isabelle’s λ-terms (§2.1), i.e. deep embedding of mathematical formulas and implementation by ML functions. Evaluation of such functions is alien to Isabelle’s simplifier; *ISAC*’s simplifier has to evaluate such functions embedded into terms with logical connectives.

3. **Simplify mathematical expressions** as close to traditional work by paper and pencil as possible. An example is given by Fig.1: A fraction is simplified in collaboration between system and user; the latter inputs the next step (the cursor is the black spot bottom left). The user had asked *ISAC* for justification of steps, shown at the right margin. In the last but one step the Lucas-Interpreter has decided (by a program similar to the one on p.92) to apply a group of seven theorems `rat_mult` ...`rat_power` assembled under the identifier `rat_mult_div_pow`. These are shown in the `Theory browser`, a specific view on theories (§2.3), in the right window of Fig.1.



Simplify(...)

$\dfrac{\frac{2}{x+3}+\frac{2}{x-3}}{\frac{8*x}{x^2-9}}$

Rewrite_Set "discard_minus"

$\dfrac{\frac{2}{x+3}+\frac{2}{x+-1*3}}{\frac{8*x}{x^2+(-1*9)}}$

Rewrite_Set "make_rat_poly_with_parentheses"

$\dfrac{\frac{2}{3+x}+\frac{2}{-3+x}}{\frac{8*x}{-9+x^2}}$

Rewrite_Set "add_fractions_p_rls"

$\dfrac{\frac{4*x}{-9+x^2}}{\frac{8*x}{-9+x^2}}$

Rewrite_Set "rat_mult_div_pow"

$\dfrac{4*x*1}{8*x*}$

Root
Rational
Theorems
Rulesets
calculate_Rational
calc_rat_erls
rational_erls
cancel_p
add_fractions_p
add_fractions_p_rls
powers_erls
powers
rat_mult_divide
reduce_0_1_2
rat_reduce_1
norm_rat_erls
norm_Rational
norm_Rational_rls
norm_Rational_min
norm_Rational_parenth
rat_mult_poly
rat_mult_div_pow

Identifier: rat_mult_div_pow

Rules:

Theorem	rat_mult
Theorem	rat_mult_poly_l
Theorem	rat_mult_poly_r
Theorem	real_divide_divide1_mg
Theorem	divide_divide_eq_right
Theorem	divide_divide_eq_left
Theorem	rat_power

For applying rules:
Rewriteorder dummy_ord

For evaluating conditions in rules:
Ruleset e_rls

(c) isac-team (math-autor)

Figure 1: *ISAC*’s close to hand-written calculations.

With the experiences gained by *ISAC*’s simplifier so far the question has to be re-raised, whether the proprietary component would be better replaced by native Isabelle components, Isabelle’s simplifier and evaluation machinery of the code generator. However, such grouping is not trivial with respect to confluence and termination [9], and compiling terminating and confluent groups might not preserve these two properties indispensable in rewriting. So specific tools like [23] might be considered to assist mathematics authors in compiling terminating and confluent term rewriting systems (called `Rulesets` here).

A fundamental design idea of *ISAC* is to smoothly extend traditional customs and notation of engineers to a more rigorous level. The simplifier is relevant for this idea, because engineering problems can be broken down to sub-problems where most of them are just rewriting. And experience with simplification, equation solving, differentiation, etc has shown that such grouping of theorems can achieve traces fairly close to traditional work by paper and pencil.

3 The Front-End

The design for an *ISAC*-front-end was an adventure. The first attempt [12] went straight into implementation alongside the implementation of the math-engine — this was a flop, the user requirements were completely unclear: What can users request for learning from a novel kind of math-engine (with features not as clearly expressed as in §2 above)?

So in 2002 an outstanding team of four students [15, 16, 18, 25] was formed and led by an expert[19] in thorough engineering of user requirements. This resulted in a substantial documentation [46] and a design guiding development up to the date of this article. However, the documentation could not be kept up to date by students, rather documentation is scattered across thirty diploma theses now[20], which are also cited in this paper (thus the lengthy bibliography). Recently a second round of user requirements engineering started in collaboration with staff of engineering faculties [47], which is under construction and already led to novel ideas for support in the specification phase.

Before turning to a description of design and implementation for the front-end (accompanied with a short description on expectations for users with respect to “systems that explain themselves”) interfaces between Isabelle’s and *ISAC*’s front-ends and respective back-ends are considered as follows.

3.1 The Interface Math-Engine — Front-End

The sustainable evolution of Isabelle over the years not only influenced *ISAC*’s mathematics engine, but also the interface between the latter and *ISAC*’s font-end. In 2000, when *ISAC* started development, Isabelle was still used via tactics on the command line. So the standardised streams of Unix, `stdin` and `stdout` were the interface between *ISAC*’s math-engine in SML and the *ISAC* front-end in Java. When the proof language Isar [49] superseded Isabelle’s tactical language, also the more advanced user-interface Isabelle/jEdit [51] replaced the Emacs-interface. Also Isabelle/jEdit motivated a new kind of interface Isabelle/PIDE [52] as shown in the top area of Fig.2 on the next page.

Isabelle/PIDE came along with dropping `stdin` and `stdout` in Isabelle2013-1 — *ISAC* was in danger to lose connection between front-end and back-end. But there was good luck: The well-known verification system Leon was dependent on Isabelle’s `stdin` and `stdout` as well, so there were resources to develop the `libisabelle` [19] interface as a slim version of Isabelle/PIDE and to maintain it reliably[21] — and *ISAC* now re-uses this interface as shown in the bottom area of Fig.2.

[19]Klaus Schmaranz `http://www.consonya.com`.

[20]`http://www.ist.tugraz.at/isac/Publications_and_Theses#Theses`

[21]`https://lars.hupel.info/libisabelle/`

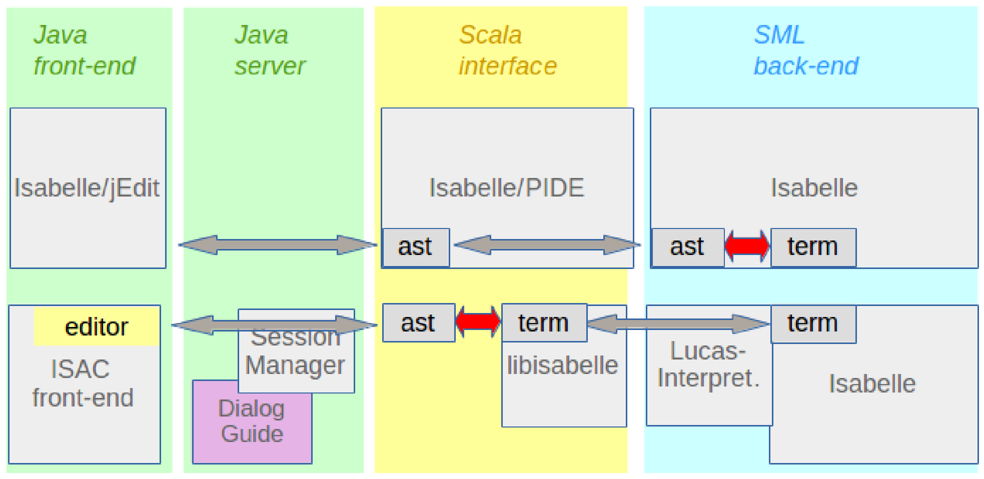


Figure 2: Interfaces of Isabelle and of *ISAC*.

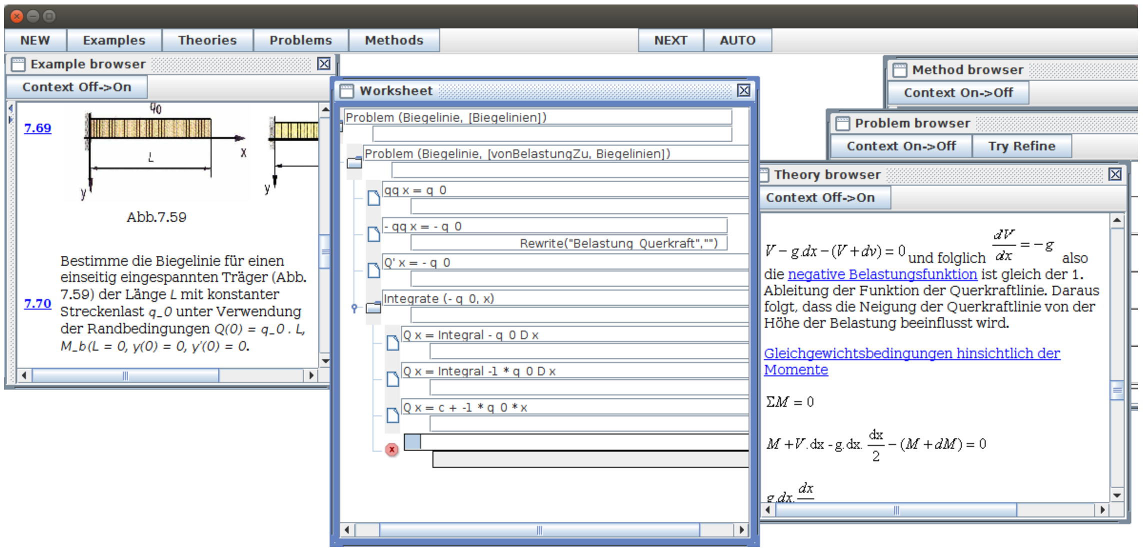
Recently an experimental formula editor has been implemented in *ISAC* [31], already shown in the left window of Fig.1. This editor is not only written in Scala (indicated by yellow background in Fig.2), it also raises architectural issues: An editor in the *ISAC* front-end uses Java Swing and the underlying data structure is an “annotated syntax tree” (AST) close to visual representation. However, `libisabelle` transfers Isabelle’s λ-terms as Scala trees (which nicely integrate with Java). So Isabelle’s translation between `terms` (appropriate for mathematics) and ASTs (appropriate for presentation) had to be transferred from `SML` to `Scala`, indicated by red arrows in Fig.2. The question, whether this transfer is another detour in *ISAC*’s prototyping process, will be discussed at the end of the paper. `DialogGuide` and `SessionManager`, located on the *ISAC* server, will show up again in Fig.4 below and will be discussed there. The `Lucas-Interpreter` in the back-end has been introduced in §2.2.

3.2 A Self-Explanatory System

From students’ perspective the main idea of *ISAC* is to support learning math by doing math on a *model of math* implemented in software. An analogy is good chess software which is used by champions to explore new strategies as well as by novices to learn how to play chess. Such software is a *complete* (with support for the complete game from opening to final) and *interactive* model of chess, where moves are analogous to steps in solving problems in engineering math. But chess software is not a *transparent* model: watching cascades of valuation functions at work millions of times is not informative.

So a fundamental design idea of *ISAC* is to provide a *complete, interactive and transparent model of math* — the system not only explains itself at the user-interface (like a mobile), but also explains itself by exhibiting it’s internal structure (down to a certain level): The *ISAC* prototype is *complete* in the sense that is supports all phases of problem solving: Fig.3 shows the phase of solving the `Problem` given by example 7.70 in the electronic textbook on the left (entitled `Example browser`). Behind the example the input data, the theory `Biegelinie` and suggestions of a specification `[Biegelinien]` is hidden — design of interactions on these during phases of modelling and specifying will be described in §3.4.

The `Worksheet` in the middle resembles a sheet of paper used for constructing a solution of the problem. §2.4 mentioned that *ISAC* should adapt to engineers’ customs and notations as much as possible – which is better accomplished by the experimental editor in Fig.1 on p.94 than in Fig.3. The `Worksheet`



7.69

Abb.7.59

7.70 Bestimme die Biegelinie für einen einseitig eingespannten Träger (Abb. 7.59) der Länge *L* mit konstanter Streckenlast *q_0* unter Verwendung der Randbedingungen *Q(0) = q_0 . L, M_b(L = 0, y(0) = 0, y'(0) = 0*.

```
Problem (Biegelinie, [Biegelinien])
  Problem (Biegelinie, [vonBelastungZu, Biegelinien])
    qq x = q 0
    - qq x = - q 0
                    Rewrite("Belastung Querkraft","")
    Q' x = - q 0
    Integrate (- q 0, x)
      Q x = Integral - q 0 D x
      Q x = Integral -1 * q 0 D x
      Q x = c + -1 * q 0 * x
```

$V - g.dx - (V + dv) = 0$ und folglich $\frac{dV}{dx} = -g$ also die negative Belastungsfunktion ist gleich der 1. Ableitung der Funktion der Querkraftlinie. Daraus folgt, dass die Neigung der Querkraftlinie von der Höhe der Belastung beeinflusst wird.

Gleichgewichtsbedingungen hinsichtlich der Momente

$\Sigma M = 0$

$M + V.\mathrm{dx} - \mathrm{g.dx}.\frac{\mathrm{dx}}{2} - (M + dM) = 0$

$g.dx.\frac{dx}{}$

Figure 3: *ISAC*'s front-end.

shows the student at input for simplifying an integral. In addition to the functionality of a sheet of paper the prototype already provides various kinds of support:

- The system will check the input reliably according to Pt.2 on p.91.
- If the student gets stuck, the system can provide a `next` (the button in Fig.3) step due to Lucas-Interpretation (§2.2); how the latter is used for adaptive user guidance, this will be shown in §3.3.
- In case the system does steps, the student can ask for explanation by requesting a derivation: in the step from $-qq\ x = -q_0$ to $Q'\ x = -q_0$ this is only the theorem `Belastung_Querkraft` and `Rewrite` is the tactic applying this theorem.
- For each theorem respective Isabelle proofs can be inspected. However, lookup of underlying definitions by mouse click like in Isabelle/jEdit is not yet realised.
- Each theorem can be accompanied by multimedia data with explanations; an example for theorem `Belastung_Querkraft` is given in the `Theory browser` at the right.
- The specification and the method underlying the problem solution can be inspected in the `Problem browser` and `Method browser` respectively.

So, an interested student can find all kinds of explanations within all three aspects of mathematics, the deductive, applicative and algorithmic aspect according to §2.3. Field tests, performed so far [33, 38, 39], indicate that digital natives indeed experience such a system as self-explanatory.

Furthermore, TP-technology offers various self-explanatory features not yet adopted in the prototype: For instance, Isabelle has a counterexample generator [10], or [44] can generate specific examples even for implicit definitions.

3.3 Adaptive User Guidance

Section §3.2 introduced an analogy between learning chess and learning mathematics by software as well as the fundamental design idea of *ISAC* to be a complete, interactive and transparent model of math — a clear and unadorned model. Therefore the main aim of *ISAC*'s user guidance is to make the experience with the self-explanatory system more efficient, more smooth and motivating, without addressing extrinsic motivation. We do not go for mimicking a human tutor.

A `DialogGuide`[22] is responsible for implementing the requirements regarding processes of problem solving introduced in the first paragraph of 2.2. In doing so, it implements the Dialog Control component of the Seeheim Model [42], which was chosen as a template for *ISAC*'s user interaction architecture. The structure of the `DialogGuide`, its operating infrastructure and the relation of the architecture to the Seeheim Model are illustrated in Fig.4; the `DialogGuide` has already be mentioned in Fig.2 on p.96. The

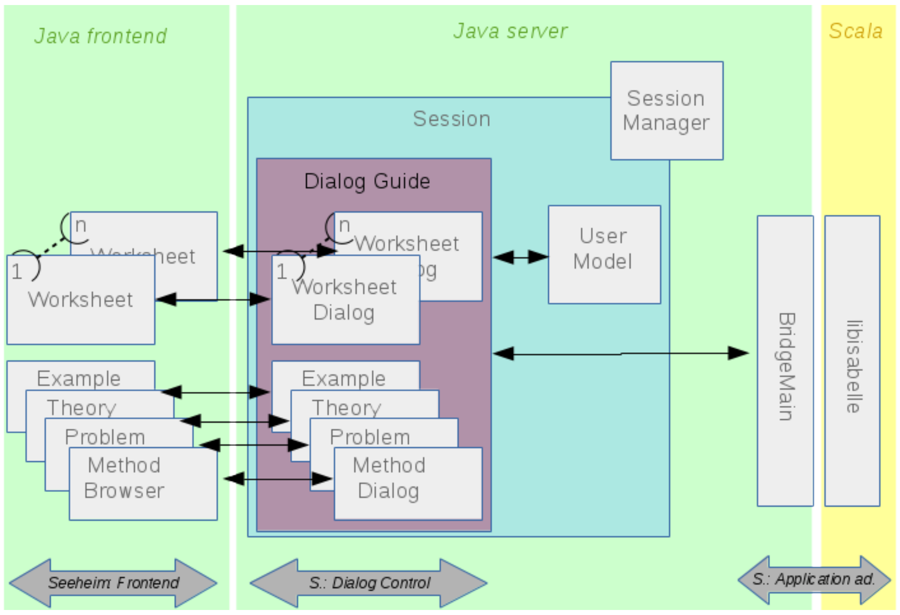


Figure 4: User interaction and its relations to code and system architecture.

left part of Fig.4 depicts all GUI elements introduced in Fig3 on p.97: `Worksheet`, `ExampleBrowser`, `TheoryBrowser`, `ProblemBrowser` and `MethodBrowser`; they correspond to the Frontend component of the Seeheim Model. In the central part, the `SessionManager` assigns one `Session` per user which contains a `DialogGuide` proper and a `UserModel`[23]. The `DialogGuide` coordinates a couple of dialogs: one `WorksheetDialog`[24]. per `Worksheet` (so several (variants of) calculations can be done in parallel) and one `*Dialog`[25]. per `*Browser`. Note that *Dialog* in this context does not denote widgets on screen, but an internal component moderating the flow of user interaction.

The `UserModel` represents the system's knowledge about a user which can be used by the `DialogGuide` to adapt its behaviour towards this specific user. As finding the relevant parameters for a `UserModel` is subject to further interdisciplinary research, a testbed under the name of `UserLogger`[26] has been developed [22] to assist in research and prototyping.

On the far right, lastly, `BridgeMain` and `libisabelle` connect to the math-engine and constitute the Application adapter of the Seeheim Model.

[22] `https://intra.ist.tugraz.at/hg/isac/file/b2fd3773f54b/isac-java/src/java/isac/session/DialogGuide.java`

[23] `https://intra.ist.tugraz.at/hg/isac/file/b2fd3773f54b/isac-java/src/java/isac/users/UserModel.java`

[24] `https://intra.ist.tugraz.at/hg/isac/file/b2fd3773f54b/isac-java/src/java/isac/wsdialog/WorksheetDialog.java`

[25] `https://intra.ist.tugraz.at/hg/isac/file/b2fd3773f54b/isac-java/src/java/isac/browserdialog/BrowserDialog.java`

[26] `https://intra.ist.tugraz.at/hg/isac/file/b2fd3773f54b/isac-java/src/java/isac/users/UserLogger.java`

In order to prepare the `DialogGuide` for meeting the challenges of expected complexity, user interactions are not determined by Java code[27], but by a rule-based system [21].

In the prototype implementation of this architecture, the `DialogGuide` does not interfere with the flow of communication as the `UserModel` does not yet differentiate users, both awaiting a major development effort in collaboration with experts in user modelling and with experts in psychology of mathematics education.

Anticipating such collaboration conceivable interventions of the `DialogGuide` include:

Sharing knowledge in the sense of communicating steps in a calculation done by the user or the math-engine to the other partner for inspection. Information about such a step taken includes the resulting formula but in addition to that may include the tactics applied to connect the formula to the previous one.

Deciding which knowledge to share, as not every detail known to the TP is necessarily of interest to every user on every occasion. This decision includes the question how much knowledge to share. In addition to depending on a particular user, this decision can depend on the context of the calculation, eg. whether done to solve a problem, to playfully explore or to take an exam.

Deciding when to share knowledge, which can depend on how far into a calculation we have proceeded as well as how much time has passed.

Asking for explanations which is equivalent to giving reasons for steps taken in a calculation. In *ISAC*'s architecture this would mean requesting to make transparent which tactics were employed in achieving a result

Asking for activity such as selecting a method to solve the problem at hand, computing the next step or proposing the next tactic to apply.

Deciding what to ask for basically involves the same questions as deciding which knowledge to share. With the requirement that the `DialogGuide` treat both partners on equal terms §2.2, it is just a matter of point of view whether something passed on by the `DialogGuide` has been requested by one partner or offered by the other.

Deciding when to ask for activity or for explanations.

Choosing the right level of abstraction by grouping rewrite-rules as mentioned in §2.1 or selecting specific explanatory material.

Switching roles of the user and the math-engine boils down to exchanging the actions of sharing and requesting information. Again, the `DialogGuide` does not make any difference between math-engine and user.

Such interventions of the `DialogGuide` are supported by *ISAC*'s software architecture.

Given the experience from *ISAC*'s prototyping, collaboration with experts in user modelling and in psychology of mathematics education will follow these guidelines:

The `DialogGuide` implements its interventions by presenting only part of the calculation tree to the respective partners and by blocking requests or inserting additional requests into the communication.

Adaptive user guidance requires that the above decisions be not hard-coded, but be made to fit a specific situation as closely as possible. How to arrive at the right decisions will depend on the user, her knowledge, experience and aims on one hand and on the task being tackled on the other hand. We

[27]`https://intra.ist.tugraz.at/hg/isac/file/b2fd3773f54b/isac-java/src/java/isac/wsdialog/WorksheetDialog.java#l699`

can assume that the specifics of the task can be derived from the problem and the method of the current calculation, both of which are known. Users can be categorised on a very general level as "beginner" or "expert". A more refined model of user expertise could take into account whether a user is "familiar with", "aware of" or "completely new to" specific theories or even individual rewrite rules. Even personal traits such as endurance or resistance to stress could make for a better learning experience if properly considered. Finding relevant parameters for a `UserModel` is subject to research in fields other than computer science.

Finally, the `UserModel` need not remain static. Given that web sites can adapt to users by observing and analysing their browsing behaviour, the `UserModel` could get to know the user more closely from experience gained in past interactions.

There are high expectations in user guidance by *ISAC*; some ideas how to exploit the potential are: support for rule-application [26] and handling of error-patterns [11].

The architecture described above has the major benefit that authoring mathematics (by writing programs, see §2.2) is strictly separated from authoring dialogues (by using the rule-based system [21]), thus disentangling expertise often unpleasantly mixed in authoring educational systems.

3.4 Specific Support for a Specification Phase

Support for formal specification has been a concern of prototyping from the beginning up to the present, when usability at engineering faculties is being considered [47]. From the beginning it was clear that solving problems in engineering mathematics starts with translating observations from the real world into formulas. In order to support the student in this effort, *ISAC* hides minimal data behind each example, for instance no.`7.70` in Fig.3 has this hidden formalisation:

$$\begin{array}{l}[\ (\ [\ \textit{Traegerlaenge } L,\ \textit{Streckenlast } q_0,\ \textit{Biegelinie } y,\\ \quad \textit{Randbedingungen } [Q\,0 = q_0 \cdot L,\ M_b\,L = 0,\ y\,0 = 0,\ \frac{d}{dx}y\,0 = 0],\ \textit{FunktionsVariable } x\ \ ]\\ \ (\textit{"Biegelinie"},\ [\textit{"Biegelinien"}],\ [\textit{"IntegrierenUndKonstanteBestimmen2"}\ ]\ )\ )\ ]\end{array}$$

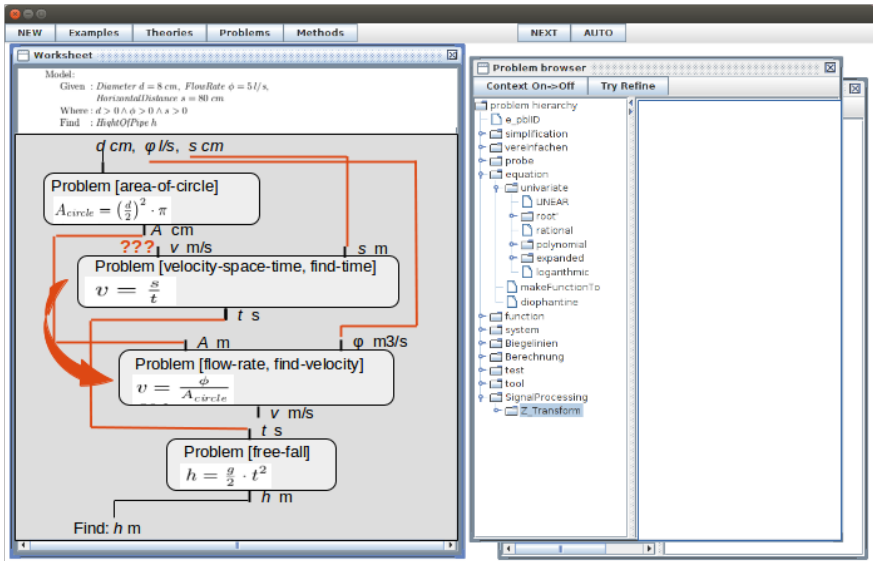
The first two items are input data (symbolic in this case), the second line starts with (a sub-term of) the post-condition and the third line is a triple referencing theory, problem and method, respectively (see §2.3).

These data are used by *ISAC* to help students with input to the formal specification, which is the following for the example in Fig.3:

```
01 Problem (Biegelinie, [Biegelinien])
02   Specification:
03     Model:
04       Given    : Traegerlaenge L, Streckenlast q_0
05       Where    : q_0 ist_integrierbar_auf [0,L]
06       Find     : Biegelinie y
07       Relate   : Randbedingungen [Q 0 = q_0 · L, M_b L = 0, y 0 = 0, d/dx y 0 = 0]
08     References:
09       Theory   : Biegelinie
10     x Problem  : ["Biegelinien"]
11     o Method   : ["IntegrierenUndKonstanteBestimmen2"]
12   Solution:
```

The `Specification` consists of a `Model` and `References`, where the "x" and "o" indicate selection whether the model is shown for the `Problem` (as specification) or for the `Method` (as guard). Preconditions given in `Where` are checked by an automated prover (§2.4). The post-condition, partially given by `Relate`, should be checked as soon as a result has been constructed (this check is not yet implemented in the prototype). The `Solution` is the calculation shown in the `Worksheet` of Fig.3 after the window for specification has been closed.

Field tests showed that high-school students (and teachers!) are not interested in formal specification. While "Formal Methods" are still on the fringes in most engineering studies, academic staff is well aware that problem solving involves not only assigning methods to sub-problems (as described above) but also appropriate selection of problems and respective sequencing. Fig.5 shows an example for what is in the pipeline toward improved support for the specification phase. In the example the `Problems` have already


Model:
Given : $Diameter\ d = 8\ cm,\ FlowRate\ \phi = 5\ l/s,\ HorizontalDistance\ s = 80\ cm$
Where : $d > 0 \wedge \phi > 0 \wedge s > 0$
Find : $HightOfPipe\ h$

Problem [area-of-circle] $A_{circle} = \left(\frac{d}{2}\right)^2 \cdot \pi$
Problem [velocity-space-time, find-time] $v = \frac{s}{t}$
Problem [flow-rate, find-velocity] $v = \frac{\phi}{A_{circle}}$
Problem [free-fall] $h = \frac{g}{2} \cdot t^2$

Figure 5: Sequencing of sub-problems ahead of solve-phase.

be selected from the `Problem browser` right and placed into a panel on the left. Here the problems can be moved around until a sequence is found such that each problem gets appropriate input from a problem above (which means that this problem has to be solved first). Respective operations are shown dynamically on a slide movie[28]. Such sequencing also would be helpful for the `SubProblems` listed in the example program on p.92.

This panel for selecting and sequencing sub-problems is not yet implemented in the prototype. However, implementation seems straight forward: Isabelle's type system will be helpful in matching the input and output data from sub-problems.

4 Present Issues and Future Work

The never-ending process of prototyping a new TP-based generation of tools for engineering education has a stopover in this paper such that the next steps become clear. With the survey on design, on architecture and benefits for students given in the previous sections, specific technical issues were clarified. Before an account of specific tasks and estimates on respective development efforts are given, a big open question needs to be tackled ...

[28] `http://www.ist.tugraz.at/projects/isac/publ/movie-sub-problems.pdf`

4.1 Where shall Front-Ends Go?

Isabelle is well underway towards a "Prover's Integrated Development Environment (PIDE)" [51]. PIDE's document model [50] is powerful such that it can drive front-of-the-wave IDEs like MicroSoft VSCode[29], while connecting with HTML5 for interaction on standard browsers [29] received little attention. A look ahead into the future was made in [54] towards distributed version control and multi-user session management; however, the respective FP7 proposal was rejected as mentioned in §1. Also ideas about collecting formal mathematics into wikis [48] are around for some time, but not yet realised.

On the other hand, *ISAC*'s design envisaged a web-based system from the very beginning [46, p.13]. An intermediate experiment with launching *ISAC*'s front-end via Java Webstart [55] is promising. But there are good reasons to replace *ISAC*'s proprietary front-end (Fig.2 on p.96) with some standard component in order to reach professional usability with reasonable effort. As an educational tool *ISAC* shall go in direction of handhelds. But respective technology still appears not sufficiently settled for investing major development efforts.

So the front-ends of Isabelle and of *ISAC* go opposite directions: the former towards a sophisticated workbench for engineers running on a workstation and the latter towards an easy-to-use tool running on a tablet or even on a mobile. However, the current state of Isabelle/jEdit appears a good compromise to start adopting it as front-end for an educational tool in introductory courses at engineering faculties.

The issues in adopting Isabelle/jEdit as front-end for *ISAC* are challenging: Differences on the side of the math-engine have been discussed in §2.1. Even more challenging is a controversial architecture: Isabelle is a heavy tool for one engineer with, in an optimal case, a server farm running automated provers in parallel — while *ISAC* is a client in the cloud, for participants of courses served by a central session management (see §3.3). The yellow area in the middle of Fig.2 on p.96, the interfaces between front-ends and math-engines raises serious questions to be tackled in cooperation with the Isabelle team.

4.2 Towards a Professional Tool

Students' projects (more than 30 in number[30]) were appropriate for experimenting with ideas and technologies. But *ISAC*'s code base has become too complex (45957 lines of code (LOC) production code, 18465 LOC test code for the front-end, 33573 LOC production code, 42384 LOC test code for the mathematics-engine) and the planned development tasks are demanding such that the next phase of development cannot be carried out without full-time software professionals. The following tasks have to be accomplished towards a professional tool for engineering education at universities:

1. Shift *ISAC*'s programming language to Isabelle's function package according to §2.2.
2. Adapt Isabelle's document model to the needs of *ISAC*: introduce session management in order to allow for user guidance (§3.3), adapt parsers to *ISAC*'s calculations (instead of Isar proofs), adapt the interface of *ISAC*'s math-engine to PIDE's document model. Adoption of this model solves several problems, e.g. explicit theory imports in the specification phase (presently blocked by libisabelle as mentioned in §2.1).
3. Improve the graphical formula editor; clarify location of ast-translations according to Fig.2 and embed the editor into Isabelle/jEdit.
4. Provide authoring tools:
   (a) for dialogue authoring according to §3.3

[29]`http://sketis.net/2017/isabellevscode-1-0-in-isabelle2017`
[30]`http://www.ist.tugraz.at/isac/Credits`

   (b) for adding explanations in HTML 5.0 to math knowledge according to §2.3
   (c) for debugging programs using the Lucas-Interpreter, see §2.2
5. Implement a graph-based tool for the specification phase as shown in Fig.5. Consider partial evaluation of physical units only according to [47, p.27].
6. Replace the only non-open-source library [21] in *ISAC* by proprietary code using Scala's `match`. Further shifts from Java to Scala can be expected (the editor [31] is already written in Scala).
7. Implement tools for collaboration as envisaged already by *ISAC*'s architecture: tutors remotely inspect students work, lecturers demonstrate a calculation on students' computers, students construct solutions collaboratively according to §3.3.
8. Make *ISAC* water-proof for written exams (where the system's help functionality and transparency are reduced to "exam mode" by the `DialogGuide` §3.3).
9. Extend *ISAC*'s calculations in a way that they can hold incorrect steps. Unlike proofs, where only incomplete steps are handled (by `oops` in Isabelle), for students also incorrect steps might be instructive: What can happen when I integrate over poles, or when I allow division by zero?

The efforts for implementing the above points can be estimated with ten man-years.

5 Conclusions

After almost two decades of prototyping is clear, where the *ISAC* software and respective innovations in teaching and learning are most helpful: in mathematics education at technical faculties. The original conception provided a "complete", transparent and interactive model of mathematics (§3.2, §3.3). As such it is appreciated most by academic lecturers, who have respective expertise and freedom for innovation; here *ISAC*'s ability to connect basic courses to advanced labs as well as to connect intuitive application with abstract mathematics come into effect most usefully and most effectively.

Long-term prototyping definitely paid off; the user requirements evolved over time and time could not have been shorter for finding out, what users (students and teachers) can request from a new generation of systems based on TP technology. A recent requirements analysis in cooperation with universities of applied science in Upper Austria confirmed the original conception and contributed novel ideas (§3.4). Experience with various technologies settled, the system interfaces stabilised. Tasks for further integrating *ISAC*'s math-engine (§2) into Isabelle are clarified. As mentioned in §4.2, the original design stood the test and the code quality remained such that improving the prototype towards a professional system appears more efficient than re-implementing from scratch.

The variety of contributions of more than thirty students not only balanced the system, this work also involved various kinds of expertise from the respective advisors. This way a network of interdisciplinary expertise arose, a perfect requisite to constitute competent developer teams for large development projects. The tasks to be performed in order to develop a professional tool are well defined and the required efforts can be reliably estimated (§4.2).

The estimated effort of ten man-years, however, is difficult to finance. So it is good to know that the *ISAC*-project is a free-rider on the successfully ongoing propagation of formal methods, including TP, which is enforced by increasing complexity in technology as addressed by "internet of things", "industry 4.0", "systems of systems" and the like.

Once formalisations of engineering disciplines are elaborated, *ISAC* can add interactive problem solving with little effort; and who knows, "systems that explain themselves" might even be useful for knowledge management in development units of specific companies.

Acknowledgements

The remarkable origins of the *ISAC*project were described in §1. The continuation is supported by hosting code repositories and administrative data at Graz University of Technology, the institutes IST and IICM and at Johannes Kepler University Linz, the institutes RISC and IIS. The evolvement of the project was and is driven by interdisciplinary expertise, where particularly warm thanks go to the advisors of theses within *ISAC* (orderd by time of first involvement): Franz Wotawa, Clemens Heuberger, Stephan Dreiseitl, Denis Helic, Bernhard Aichernig, Christian Gütl, Jens Knoop and Wolfgang Schreiner.

References

[1] *4ferries*. `https://fourferries.com/en/home/`.

[2] *Isac-project*. `http://www.ist.tugraz.at/isac/History`.

[3] *Mathtoys*. `http://mathtoys.org/`.

[4] *Publications in the ISAC-project*. `http://www.ist.tugraz.at/isac/Publications_and_Theses`.

[5] *Slide movie on specifying sub-problems and determining a sequence*. `http://www.ist.tugraz.at/projects/isac/publ/movie-sub-problems.pdf`.

[6] *Team of the Isabelle project*. `https://www21.in.tum.de/`.

[7] *Team of the ISAC-project*. `http://www.ist.tugraz.at/isac/Credits`.

[8] David Aspinall (2000): *Proof General: A Generic Tool for Proof Development*. In: *Tools and Algorithms for the Construction and Analysis of Systems*, *LNCS* 1785, TACAS, pp. 38–43, doi:10.1007/3-540-46419-0_3.

[9] Franz Baader & Tobias Nipkow (1998): *Term Rewriting and All That*. Cambridge University Press, Cambridge, doi:10.1017/CBO9781139172752.

[10] Jasmin Christian Blanchette, Lukas Bulwahn & Tobias Nipkow (2011): *Automatic Proof and Disproof in Isabelle/HOL*, pp. 12–27. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-24364-6_-2.

[11] Gabriella Daróczy (2013): *Cognitive Aspects of Designing Dialogues in Theorem-Prover Based Mathematics Assistants; Implementation of Error-Patterns Guiding Dialogues in ISAC*. Master's thesis, ME:CogSci, Vienna, Austria. `http://www.ist.tugraz.at/projects/isac/publ/gdaroczy-ma.pdf`.

[12] Thomas Maximilian Fink (2001): *Benutzerschnittstelle für ein Mathematik-Lernsystem im WWW*. Master's thesis, University of Technology, Institute for Softwaretechnology, Graz, Austria. `http://www.ist.tugraz.at/projects/isac/publ/tf-dipl.pdf`.

[13] Kurt Gödel (1986): In Feferman et al., editor: *Collected Works I. Publications 1929-1936*, Oxford University Press, Oxford, pp. 144–195. Translation from: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I, 1931.

[14] Matthias Goldgruber (2003): *Algebraische Simplifikation mittels Rewriting in ISAC*. Master's thesis, University of Technology, Institute for Softwaretechnology, Graz, Austria. `http://www.ist.tugraz.at/projects/isac/publ/DA-M02-main.ps.gz`.

[15] Richard Gradischnegg (2004): *Eine Java/SML–Schnittstelle für ISAC auf Basis von XML*. Master's thesis, University of Applied Sciences, Dpt. Software Engineering, Hagenberg, Upper Austria. `www.ist.tugraz.at/projects/isac/publ/da-gradischnegg.ps.gz`.

[16] Andreas Griesmayer (2003): *Architecture and Knowledge-Represenation of the Web-based Math-Learning-System ISAC*. Master's thesis, University of Technology, Institute for Softwaretechnology, Graz, Austria. `www.ist.tugraz.at/projects/isac/publ/da-griesmayer.pdf`.

[17] Florian Haftmann (2017): *Code generation from Isabelle/HOL theories*. Theorem Proving Group at TUM, Munich. Available at `http://isabelle.in.tum.de/dist/Isabelle2017/doc/codegen.pdf`. Part of the Isabelle distribution.

[18] Mario Hochreiter (2004): *Design and Implementation of a Graphical User Interface for the Math-Learning-System ISAC*. Master's thesis, University of Applied Sciences, Dpt. Software Engineering, Hagenberg, Upper Austria. `www.ist.tugraz.at/projects/isac/publ/da-hochreiter.ps.gz`.

[19] Lars Hupel & Viktor Kuncak (2016): *Translating Scala Programs to Isabelle/HOL*, pp. 568–577. Springer International Publishing, Cham, doi:10.1007/978-3-319-40229-1_38.

[20] Natalie Karl (2016): *Developing an Inclusive Approach for Representing Mathematical Formulas*. Master's thesis, Hagenberg University of Applied Sciences, Linz, Austria. `http://www.ist.tugraz.at/projects/isac/publ/Masterthesis_NatalieKarl.pdf`.

[21] Markus Kienleitner (2012): *Towards "NextStep Userguidance" in a Mechanized Math Assistant*. Master's thesis, IICM, Graz University of Technology. `http://www.ist.tugraz.at/projects/isac/publ/mkienl_bakk.pdf`.

[22] Franz Kober (2012): *Logging of High-Level Steps in a Mechanized Math Assistant*. Master's thesis, IICM, Graz University of Technology. `http://www.ist.tugraz.at/projects/isac/publ/fkober_bakk.pdf`.

[23] Martin Korp, Christian Sternagel, Harald Zankl & Aart Middeldorp (2009): *Tyrolean Termination Tool 2*, pp. 295–304. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-02348-4_21.

[24] Alexander Krauss (2009): *Automating Recursive Definitions and Termination Proofs in Higher-Order Logic*. Ph.D. thesis, Technische Universität München.

[25] Alan Krempler (2005): *Architectural Design for Integrating an Interactive Dialogguide into a Mathematical Tutoring System*. Master's thesis, University of Technology, Institute for Softwaretechnology, Graz, Austria. `http://www.ist.tugraz.at/projects/isac/publ/da-krempler.pdf`.

[26] Alan Krempler & Walther Neuper (2008): *Formative Assessment for User Guidance in Single Stepping Systems*. In Michael E. Aucher, editor: *Interactive Computer Aided Learning, Proceedings of ICL08*, Villach, Austria. `http://www.ist.tugraz.at/projects/isac/publ/icl08.pdf`.

[27] Richard Lang (2003): *Elementare Gleichungen der Mittelschulmathematik in der ISACWissensbasis*. Master's thesis, University of Technology, Institute of Software Technology, Graz, Austria. `http://www.ist.tugraz.at/projects/isac/publ/da-rlang.ps.gz`.

[28] Mathias Lehnfeld (2011): *Verbindung von 'Computation' und 'Deduction' im ISAC-System*. Master's thesis, Institut für Computersprachen, Technische Universität Wien. Bakkalaureate project.

[29] Christoph Lüth & Martin Ring (2013): *A Web Interface for Isabelle: The Next Generation*, pp. 326–329. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-39320-4_22.

[30] Roman E. Maeder (2012): *Programming in Mathematica*, 3rd edition. Addison-Wesley, Reading, Mass.

[31] Marco Mahringer (2018): *Formula Editors for TP-based Systems. State of the Art and Prototype Implementation in ISAC*. Master's thesis, University of Applied Sciences, Hagenberg, Austria. `http://www.ist.tugraz.at/projects/isac/publ/mmahringer-master.pdf`.

[32] Walther Neuper (2001): *Reactive User-Guidance by an Autonomous Engine Doing High-School Math*. Ph.D. thesis, IICM - Inst. f. Softwaretechnology, Technical University, A-8010 Graz. `http://www.ist.tugraz.at/projects/isac/publ/wn-diss.ps.gz`.

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

[34] Walther Neuper (2012): *Automated Generation of User Guidance by Combining Computation and Deduction*. In Quaresma & Back [43], pp. 82–101, doi:10.4204/EPTCS.79.5. `http://eptcs.web.cse.unsw.edu.au/paper.cgi?THedu11.5`.

[35] Walther Neuper (2014): *GCD — A Case Study on Lucas-Interpretation*. In: *Joint Proceedings of the MathUI, OpenMath and ThEdu Workshops and Work in Progress track at CICM*, Coimbra, Portugal. `http://ceur-ws.org/Vol-1186/paper-17.pdf`.

[36] Walther Neuper (2016): *Lucas-Interpretation from Users' Perspective*. In: *Joint Proceedings of the FM4M, MathUI, and ThEdu Workshops, Doctoral Program, and Work in Progress at the Conference on Intelligent Computer Mathematics*, Bialystok, Poland, pp. 83–89. `http://cicm-conference.org/2016/ceur-ws/CICM2016-WIP.pdf`.

[37] Walther Neuper (2017): *Formal Abstraction in Engineering Education — Challenges and Technology Support*. *Acta Didactica Napocensia* 10(1), doi:10.24193/adn.10.1.1. With sect.no. `http://www.ist.tugraz.at/projects/isac/publ/sys-explain-eng-edu.pdf`.

[38] Walther Neuper & Christian Dürnsteiner (2007): *Angewandte Mathematik und Fachtheorie mithilfe adaptierter Basis-Software*. Technical Report 683, IMST – Innovationen Machen Schulen Top!, University of Klagenfurt, Institute of Instructional and School Development (IUS), 9010 Klagenfurt, Sterneckstrasse 15. `https://www.imst.ac.at/imst-wiki/images/f/f9/683_Kurzfassung_Neuper.pdf`.

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

[40] Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock & Jan Peleska (2015): *Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions*. *ACM Comput. Surv.* 48(2), pp. 18:1–18:41, doi:10.1145/2794381.

[41] Tobias Nipkow, Lawrence C. Paulson & Markus Wenzel (2002): *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. *LNCS* 2283, Springer, doi:10.1007/3-540-45949-9.

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

[43] Pedro Quaresma & Ralph-Johan Back, editors (2012): Proceedings First Workshop on *CTP Components for Educational Software (THedu'11)*. 79, Open Publishing Association.

[44] Wolfgang Schreiner, Alexander Brunhuemer & Christoph Fürst (2018): *Teaching the Formalization of Mathematical Theories and Algorithms via the Automatic Checking of Finite Models*. In Walther Neuper & Pedro Quaresma, editors: *ThEdu'17 Postproceedings*, this volume of *Electronic Proceedings in Theoretical Computer Science*, Open Publishing Association, Johannes Kepler University, Linz.

[45] Christoph Sprenger & Ivano Somaini (2017): *Developing Security Protocols by Refinement*. *Archive of Formal Proofs*. `http://isa-afp.org/entries/Security_Protocol_Refinement.html`, Formal proof development.

[46] *ISAC* Team (2002): *ISAC – User Requirements Document, Software Requirements Document, Architectural Design Document, Software Design Document, Use Cases, Test Cases*. Technical Report, Institute for Softwaretechnology, University of Technology. `http://www.ist.tugraz.at/projects/isac/publ/isac-docu.pdf`.

[47] FH-Design Team (2017): *ISAC-Project: User Stories, User Requirements Document, Use Cases Document*. `http://www.ist.tugraz.at/projects/isac/publ/isac-doc2.pdf`.

[48] Josef Urban, Jesse Alama, Piotr Rudnicki & Herman Geuvers (2010): *A Wiki for Mizar: Motivation, Considerations, and Initial Prototype*, pp. 455–469. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-14128-7_38.

[49] Makarius Wenzel (2007): *Isabelle/Isar — a generic framework for human-readable proof documents*. In R. Matuszewski & A. Zalewska, editors: *From Insight to Proof — Festschrift in Honour of Andrzej Trybulec*, *Studies in Logic, Grammar, and Rhetoric* 10, University of Bialystok, pp. 277–298.

[50] Makarius Wenzel (2011): *Isabelle as document-oriented proof assistant*. In: *Proceedings of the 18th Calculemus and 10th international conference on Intelligent computer mathematics*, MKM'11, Springer-Verlag, Berlin, Heidelberg, pp. 244–259. Available at `http://dl.acm.org/citation.cfm?id=2032713.2032732`.

[51] Makarius Wenzel (2012): *Isabelle/jEdit a Prover IDE within the PIDE framework*. In J. Jeuring et al., editors: *Conference on Intelligent Computer Mathematics (CICM 2012)*, *LNAI* 7362, Springer, pp. 468–472, doi:10.1007/978-3-642-31374-5_38.

[52] Makarius Wenzel (2014): *Asynchronous User Interaction and Tool Integration in Isabelle/PIDE*. In: *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pp. 515–530, doi:10.1007/978-3-319-08970-6_33.

[53] Makarius Wenzel (2017): *The Isabelle/Isar Implementation*. With contributions by Stefan Berghofer, Florian Haftmann and Larry Paulson.

[54] Makarius Wenzel & Burkhart Wolff (2012): *Isabelle/PIDE as Platform for Educational Tools*. In Quaresma & Back [43], pp. 143–153, doi:10.4204/EPTCS.79.9.

[55] Thomas Zillinger (n.y.): *A General Web-Interface of an Assessment-Engine Instantiated for a TP-based Mathematics Assistant*. Master's thesis, University of Technology, IICM, Graz, Austria. `http://www.ist.tugraz.at/projects/isac/publ/tzilling-master.pdf`.