Seminar/Projekt
ISAC-Bridge

Thomas Oberhuber

July 15, 2001
Abstract

This seminar/project is part of the ISAC-project, which is a mathematical tutoring system. The *bridge* is the connection between the SML based mathematical engine and a Java webfrontend. So the *bridge* has to fulfil two major tasks. The first one is to parse the text output, produced by the mathematical engine. The other one is to establish a connection to one or more frontends(clients).
# Table of Contents

1 Introduction ........................................... 2  
  1.1 Overview ........................................ 3  

2 Data Representation ................................. 4  

3 Interaction with the Mathematic Engine ............ 5  

4 Parsing ................................................. 6  

5 Networking ............................................. 7  
  5.1 Problem with Blocking Socket ................. 7  

6 User Management ..................................... 9  

A isac Interfaces Frontend — Kernel — Frontend .... 10  
  A.0.1 Interface Bridge — Kernel ................. 10  
  A.0.2 Interface Kernel — Bridge .................. 13  

B The two Representations of mathematical Expressions 14  

Index .................................................. 18
Chapter 1

Introduction

This project is one part of ISAC (Calulational proofs using ISAbelles knowledge). A cite from the ISAC homepage [1]: “The aim of the ISAC project is to do a reengineering of computer algebra systems, featuring a stepwise calculation and explicit modeling of (sub)problems and specification of methods - guided by the system.”

So ISAC is a system that should support students in doing math, rather than explaining or teaching it to them. An importing point in doing so, is to let the students choose their level of initiative. He/she can choose in a wide range, from “let the machine do the whole work”, to doing everything for his/her own, while the machine just checks the single steps. This is the major difference to other interactive notebooks like Mathematica.

The ISAC project consists of three parts. The most important one is the mathematical engine (short ME), which is based on Isabelle[3]. Isabelle is a theorem proving environment, implemented in the SML programming language, and therefore the ME is written in SML too. The ME is implemented by Walther Neuper, who was the ‘educational practitioner’ in the team, did the preliminary design of ISAC and developed some of the techniques employed. He coordinated the practical implementation of the prototype. The result of his work on ISAC, besides some other publications and the ISAC prototype, was his PhD-thesis “Reactive User-Guidance by an Autonomous Engine doing High-School Math”[4].

The second major component is the graphical user frontend (short FE), which is implemented in Java. Thomas Fink developed this “notebook” for his master thesis “Benutzerschnittstelle für ein Mathematik-Lernsystem im WWW”[5].

The basic task of this project/seminar was to implement an interface between the mathematical engine and the frontend. So it is easy to see that this work consists of two major parts. On the one hand this bridge (short BR) has to parse the textual output, produced by the ME, and build Java objects containing the extracted information(see Chapter 2 on page 4). On the other hand it has to establish connections to several clients. In addition to this the BR has to proceed queries from the FE, which it receives as Java objects again, and translate them back into a textual input for the ME.

Both sides of the communication require a translation of the mathematical expressions. The kernel (ME) uses some derivate of Isabelle’s mathematical notation, whereas the FE processes MathML[7], which is a W3C[6] standard.
To avoid conflicts while serving several clients, the ME creates userIDs. Also the BR uses this IDs to identify the users uniquely, to ensure that every single user receives the answers to his/her queries only.

As programming language to implement this BR, Java is chosen. The major reason for this decision was Java’s platform independence and that it provides powerful networking capabilities. Besides this JLex\[8\] and Cup\[9\] are rather comfortable tools for parser generation. As coding standard I followed the “Dinopolis Java Coding Convention”\[10\].

1.1 Overview

As described above, the bridge can be divided in two parts, the parsers and the networking unit. And so does the package structure (com.isac.bridge.parser and com.isac.bridge.client_server). The parser package contains three sub-packages, namely kernel2bridge, isacmath2mathml and mathml2isacmath. These are the three parsers. Each contains definition files for JLex and Cup. The client_server package consists of four classes. Client is instantiated by the frontend and establishes a connection to Server. ExecutionThread starts the kernel in its own thread and ClientList is used for user management.
Chapter 2

Data Representation

The syntax of the communication between the mathematical engine (short ME) and the bridge (short BR), defined by Dr Neuper, was subject of a permanent change while the duration of our work. It specifies both directions of the communication (e.g. input and output of the kernel). The current version of the specification in BNF-notation you can find in Appendix A on page 10.

To transfer the information to the frontend (short FE), java objects are generated, that hold the necessary data. For this purpose the whole syntax tree of the KE-BR communication is mapped to an class hierarchy. This has the tremendous advantage that no parsers are necessary within the clients, and the creation of the ME-input from client requests (Java object) can be realized by simple textual output methods (‘a la well known toString() methods).

The ME uses a derivation of Isabelle’s math syntax, called “isac math”. But to display the mathematical expressions in the FE, the BR has to convert these to MathML. Of course the MathML expressions the FE sends to the BR within some Java object has to be translated back to “isac math”.

A cite from the MathML homepage: “Mathematical Markup Language, or MathML is an XML application for describing mathematical notation and capturing both its structure and content. The goal of MathML is to enable mathematics to be served, received, and processed on the World Wide Web, just as HTML has enabled this functionality for text.”[7] Besides this, there is another reason why MathML is used in this project. Since MathML bases on XML, it is very facile to generate MathML expressions and these expression are rather effortlessly parseable too.

In Appendix B on page 14 you can find an example for these two mathematical representations.
Chapter 3

Interaction with the Mathematic Engine

To allow interaction with the ME, the kernel is started by the bridge itself. This is realized with the method `exec("command")` from the class `java.io.Runtime`, which starts an external process denoted by “command”. For this purpose a shell script called “start_kernel” is implemented to start SML [2] loading a pre-built image of the ME. Therefore it is easy to fetch input and output streams of the new process, that are necessary to communicate with the ME.

Using these streams the bridge can send commands to the kernel, and receive the corresponding answers. Since the kernel to bridge stream (henceforward called input stream) contains a lot of information, that is worthless for the bridge, the interesting data is embedded in pairs of “@@@@begin@@@@” and “@@@@end@@@@”. So all the uninteresting sections of the SML output can be filtered out. The data between these “parentheses” follows the syntax definitions, you can find in Appendix A.0.2 on page 13, and is parsed as described in chapter 4 on the following page.

As described in chapter 2 on the page before requests from clients are received as Java objects. To produce the input for the ME, which is written on the bridge to kernel stream (henceforward called output stream), the objects have a method `toSml()`. This method produces input strings, that are written to the output stream. If the object contains mathematical expressions, they are parsed and translated to “isac math” (see chapter 4 on the following page).
Chapter 4

Parsing

A major task while developing the bridge was to implement three parsers. The first and most powerful one was the \texttt{kernel2bridge} parser. As its name says it parses the kernel output. During the parsing it produces a Java object, that holds all the gained information. After this process the complete object is send to a frontend. The other two parsers are for converting mathematical notations. One translates \texttt{isac math} to \texttt{MathML}, whereas the other one does this in the opposite direction.

All three parsers are implemented using \texttt{JLex} (see \cite{8}) and \texttt{Cup} (see \cite{9}). \texttt{JLex} is a lexical analyzer generator. \texttt{Cup} is a parser generator, that reads a simple definition file. Since both tools are written for Java, you can provide Java code for all states, the parser can match. For this reason it is very easy to build a java object that holds all the parsed information, just while parsing the input. You can easily import your parser into a Java project, because \texttt{Cup} generates a Java source file implementing it.

\texttt{JLex} analyzes an input stream for known lexical elements, for which it produces a stream of tokens, which is processed by \texttt{Cup}. \texttt{Cup} interprets the tokens as terminals, which represent the “leaves” of a parse tree, whereas nonterminals define subtrees. Nonterminals provide a way to produce clear syntax definitions, because one can “split up” a grammar in smaller pieces. By trying to match the whole input to the defined terminals and nonterminals, \texttt{Cup} checks it for correct grammar. While doing so it can process appropriate Java code for each state of the parse process(e.g. building a Java object with the parsed information).

Both tools provide a rather simple method to define a syntax and therefore they were great help in developing this project.
Chapter 5

Networking

One aspect of the ISAC project was to build a distributed system, with a server the mathematic engine runs on, and several distributed frontends. The bridge provides this functionality.

There are several possibilities to implement network capabilities. I decided to do it the hard way (which is easy enough in Java:-), and realized it using pure sockets. So a thread was implemented, that listens on a port and forkes another thread to handle a user request. This thread fetches an outputstream from the ME to write all decoded requests to it. Java provides a comfortable way to send objects over sockets: ObjectOutputStream can write serializeable objects directly on the socket and ObjectInputStream can read them. So this wasn’t a problem at all. This thread also calls the toSml method to generate the kernel input.

When the client breaks the connection the kernel is notified that the user has finished the session and the corresponding thread dies.

5.1 Problem with Blocking Socket

There is a bug in the network communication. The answers to the clients are given directly from within the parser. So it could happen that a connection breaks down while the kernel or the parser are processing a request of the concerned user. If the answer object is too large to fit into the output stream’s buffer, the parser would be blocked. And because there is only one instance of the kernel to bridge parser, the whole bridge would hang up. (see Figure 5.1 on the next page)
Figure 5.1: Networking Problem
Chapter 6

User Management

One problem within a multi user system is to ensure to distinguish the users. In the ISAC project this is controlled by identifiers (userIDs), that are generated by the kernel while a user registration. The bridge has to know correlations between these IDs and the connections to the clients. This is necessary because the clients don’t know their identifiers and the bridge has to add them to the kernel input. The correlations are held in a separate class, that provides methods to get the corresponding counterpart.
Appendix A

isac Interfaces Frontend — Kernel — Frontend

In the Backus-Naur-Form used below the symbols of the meta language are ::= | ( ) * + ϵ as usual, whereas italic parenthesis ( ), braces { } and brackets [ ] belong to the object language. Terminal symbols start with an upper-case letter (reminding of SML datatype constructors), non-terminals start with a lower-case letter. The nonterminals int and string are as defined in SML (in particular: negative numbers are -1, ··· and not 1, -1, ···, strings are quoted by ” “), digit has the obvious meaning, and MAT is being defined elsewhere. Inserted comments (* ... *) do not belong to the object-language.

A.0.1 Interface Bridge — Kernel

FE-KE ::=  
StdinSML 0 0 0 0 New_User ; cr  
| StdinSML userID 0 0 0 End_User ; cr  
| StdinSML userID proofID acellID cellID dataFK ; cr  
userID ::= int  
proofID ::= int  
acellID ::= cellID  
cellID ::= int  
cr ::= carriage-return  

dataFK ::= New_Proof | End_Proof  
| Command user_cmd  
| RuleFK rule  
| FormFK MAT  
| PpcFK ( mat ppc )  

user_cmd ::= Accept | NotAccept | Example  
| YourTurn | Rules //MyTurn (†)  
| HowCome | DontKnow | WhatFor  
| Undo  
| ActivePlus | ActiveMinus (†)
| SpeedPlus | SpeedMinus (†) |
| Auto | Details (†) |

```
rule ::= Init_Proof ( mat_list , spec ) | Specify_Domain domID
| Specify_Problem pblID | Specify_Method metID
| Add_Given MAT | Del_Given MAT
| Add_Find MAT | Del_Find MAT
| Add_Relation MAT | Del_Relation MAT
| Refine pblID | Apply_Method metID
| Check_Postcond pblID | Free_Solve

| Rewrite_Inst subs thm | Rewrite thm
| Rewrite_Set_Inst ( subs , rls ) | Rewrite_Set rls (@)
| End_Ruleset | Calculate op
| Rewrite_Asm thm |

| Substitute subs | Apply_Assumption preds
| Take term | Take_Init term
| Group ( con , int_list ) | CAScmd term

| Split_And | Conclude_And
| Split_Or | Conclude_Or
| Begin_Trans | End_Trans
| Begin_Seq |
| Intersect | End_Intersect
| Check_elementwise pred | Collect_Trues (##)
| Mstep string |

mat_ppc:= ( ( Problem pblID )| ( Method metID )) (#)
```

```
Given mat_list
Where mat_list
Find mat_list
With mat_list
Relate mat_list
```

```
domID:= string
pblID ::= string_list
metID::= ( domID , string )
spec ::= ( domID , pblID , metID )
```

```
thm ::= ( thmID , MAT ) (* theorem *)
```

```
thmID:= string
term ::= MAT (* predicate *) (##)
pred ::= MAT (* predicates *)
preds ::= mat_list (* set of rewrite-rules *)
rls ::= string (* \( \wedge, \vee \) *)
con ::= \( \wedge, \vee \) (**) 
```

```
op ::= "plus" | "minus" | "times" (**)
    | "cancel" | "pow" | "sqrt" (**)
```

11
\texttt{subs ::= mat\_list} \hspace{1cm} (* substitutions as equalities *)

\begin{align*}
\texttt{int\_list ::=} & \quad \epsilon \mid (\text{int}, \text{int})^* \\
\texttt{string\_list ::=} & \quad \epsilon \mid (\text{string}, \text{string})^* \\
\texttt{mat\_list ::=} & \quad \epsilon \mid (\text{MAT}, \text{MAT})^*
\end{align*}
### A.0.2 Interface Kernel — Bridge

**KE-FE ::=**

1. `( userID , 0 , 0 , [ ] , New_User )`
2. `( userID , proofID , cellID , cellID_list , dataKF_list )`

**dataKF_list ::= [ dataKF ( , dataKF )* ]**

**dataKF ::= New_Proof | End_Proof**

- `( userID , proofID , cellID , cellID_list , dataKF_list )`
- `dataKF_list ::= [ dataKF ( , dataKF )* ]`
- `dataKF ::= New_Proof | End_Proof`
- `System string | Indicate ind_list`
- `Select [ rule ( , rule )* ]`
- `Active int | Speed int | Domain domID`
- `RuleKF ( edit , rule )`
- `FormKF ( cellID , edit , indent , nest , MAT )`
- `PpcKF ( cellID , edit , indent , nest , tag_ppc )`

**ind_list ::= [ ind ( , ind )* ]**

**ind ::= ( "guide", ( Safe | Unsafe | Helpless ) )**

1. `( userID , proofID , cellID , cellID_list , dataKF_list )`
2. `dataKF_list ::= [ dataKF ( , dataKF )* ]`
3. `dataKF ::= New_Proof | End_Proof`
4. `Request string | Message string | Error string`
5. `System string | Indicate ind_list`
6. `Select [ rule ( , rule )* ]`
7. `Active int | Speed int | Domain domID`
8. `RuleKF ( edit , rule )`
9. `FormKF ( cellID , edit , indent , nest , MAT )`
10. `PpcKF ( cellID , edit , indent , nest , tag_ppc )`
11. `ind_list ::= [ ind ( , ind )* ]`
12. `ind ::= ( "guide", ( Safe | Unsafe | Helpless ) )`
13. `Request string | Message string | Error string`
14. `System string | Indicate ind_list`
15. `Select [ rule ( , rule )* ]`
16. `Active int | Speed int | Domain domID`
17. `RuleKF ( edit , rule )`
18. `FormKF ( cellID , edit , indent , nest , MAT )`
19. `PpcKF ( cellID , edit , indent , nest , tag_ppc )`
20. `ind_list ::= [ ind ( , ind )* ]`
21. `ind ::= ( "guide", ( Safe | Unsafe | Helpless ) )`
22. `Request string | Message string | Error string`
23. `System string | Indicate ind_list`
24. `Select [ rule ( , rule )* ]`
25. `Active int | Speed int | Domain domID`
26. `RuleKF ( edit , rule )`
27. `FormKF ( cellID , edit , indent , nest , MAT )`
28. `PpcKF ( cellID , edit , indent , nest , tag_ppc )`
29. `ind_list ::= [ ind ( , ind )* ]`
30. `ind ::= ( "guide", ( Safe | Unsafe | Helpless ) )`
31. `Request string | Message string | Error string`
32. `System string | Indicate ind_list`
33. `Select [ rule ( , rule )* ]`
34. `Active int | Speed int | Domain domID`
35. `RuleKF ( edit , rule )`
36. `FormKF ( cellID , edit , indent , nest , MAT )`
37. `PpcKF ( cellID , edit , indent , nest , tag_ppc )`
38. `ind_list ::= [ ind ( , ind )* ]`
39. `ind ::= ( "guide", ( Safe | Unsafe | Helpless ) )`
40. `Request string | Message string | Error string`
41. `System string | Indicate ind_list`
42. `Select [ rule ( , rule )* ]`
43. `Active int | Speed int | Domain domID`
44. `RuleKF ( edit , rule )`
45. `FormKF ( cellID , edit , indent , nest , MAT )`
46. `PpcKF ( cellID , edit , indent , nest , tag_ppc )`
47. `ind_list ::= [ ind ( , ind )* ]`
48. `ind ::= ( "guide", ( Safe | Unsafe | Helpless ) )`
49. `Request string | Message string | Error string`
50. `System string | Indicate ind_list`
51. `Select [ rule ( , rule )* ]`
52. `Active int | Speed int | Domain domID`
53. `RuleKF ( edit , rule )`
54. `FormKF ( cellID , edit , indent , nest , MAT )`
55. `PpcKF ( cellID , edit , indent , nest , tag_ppc )`
56. `ind_list ::= [ ind ( , ind )* ]`
57. `ind ::= ( "guide", ( Safe | Unsafe | Helpless ) )`
58. `Request string | Message string | Error string`
59. `System string | Indicate ind_list`
60. `Select [ rule ( , rule )* ]`
61. `Active int | Speed int | Domain domID`
62. `RuleKF ( edit , rule )`
63. `FormKF ( cellID , edit , indent , nest , MAT )`
64. `PpcKF ( cellID , edit , indent , nest , tag_ppc )`
65. `ind_list ::= [ ind ( , ind )* ]`
66. `ind ::= ( "guide", ( Safe | Unsafe | Helpless ) )`
67. `Request string | Message string | Error string`
68. `System string | Indicate ind_list`
69. `Select [ rule ( , rule )* ]`
70. `Active int | Speed int | Domain domID`
71. `RuleKF ( edit , rule )`
72. `FormKF ( cellID , edit , indent , nest , MAT )`
73. `PpcKF ( cellID , edit , indent , nest , tag_ppc )`
Appendix B

The two Representations of mathematical Expressions

This is an example of a mathematical expression in “isac math” notation as well as its MathML representation:

"isac math" representation:
\( \sqrt{(9 + 4 \times x)} = \sqrt{x} + \sqrt{(5 + x)} \)

MathML:
<math>
<mrow>
  <msqrt>
    <mrow>
      <mo>(</mo>
      <mn>9</mn>
      <mo>+</mo>
      <mn>4</mn>
      <mo>\times</mo>
      <mi>x</mi>
      <mo>)</mo>
    </mrow>
  </msqrt>
  <mo>=</mo>
  <msqrt>
    <mrow>
      <mi>x</mi>
    </mrow>
  </msqrt>
  <mo>+</mo>
  <msqrt>
    <mrow>
      <mo>(</mo>
      <mn>5</mn>
      <mo>)</mo>
    </mrow>
  </msqrt>
</mrow>
</math>
<math>
  <mrow>
    <mo>+</mo>
    <mi>x</mi>
    <mo>)</mo>
  </mrow>
</math>
References

The ISAC homepage

Standard ML of New Jersey (abbreviated SML/NJ) is a compiler for the Standard ML '97 programming language with associated libraries, tools, and documentation. SML/NJ is free, open source software.

Isabelle is a popular generic theorem proving environment developed at Cambridge University (Larry Paulson) and TU Munich (Tobias Nipkow).


The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential as a forum for information, commerce, communication, and collective understanding.

MathML is an XML application for describing mathematical notation and capturing both its structure and content.

JLex is a lexical analyzer generator, written for Java, in Java. JLex was developed by Elliot Berk at Princeton University.

CUP Parser Generator for Java
        Dinopolis Java Coding Convention