数理解析研究所講究録
第1927巻 2014年 89-102

タブレット端末への数式処理システムの実装手法
How to develop a computer algebra system on tablets

藤本 光史
MITSUSHI FUJIMOTO
福岡教育大学
FUKUOKA UNIVERSITY OF EDUCATION *

Abstract

In 2010, we started a project to develop a Math e-Learning system *Mathellan* for pen-based mobile devices. We are planning to use AsirPad, a computer algebra system on Linux PDA, for a client of Mathellan. However, the mainstream of the current mobile devices is shifting from PDA to smartphones and tablet devices. Therefore, we needed a new development environment. AsirPad consists of two main components: a CAS engine and a handwriting interface. We used a cross-build environment by arm rootfs/QEMU/chroot to make an executable binary of Risa/Asir for the Android platform. We can build C/C++ source code in this environment as if we are in a self-build environment. It means that we do not need to modify source code for cross-build. Furthermore, we adopted a cross-platform application framework Qt to build a GUI. Qt can be used to build applications for various operating systems: Windows, MacOS X, Linux, Android and iOS. We can develop a GUI for various mobile devices with the same source code by Qt.

1 数式処理システムの教育への応用

筆者はこれまで数式処理システムの教育利用に関する様々な活動を行ってきた。この節では、そのいくつかを紹介する。2003年、Linux搭載のPDA Zaurusで動作する手書き数式インターフェイスを有した数式処理システム AsirPad [1] を開発し、2005年に中学校でこれを使ってRSA暗号の授業を行った。通常の電卓では巨大な数の割り算は実行できないため数式処理システムが必要だった。生徒達はAsirPadで計算しながらメッセージの暗号化と復号化の方法を学んだ。彼らにとってPDAも手書き数式の利用も初めての体験だったが、全く問題なく利用することができた。事前のトレーニングさえ不要だった [2]。

2008年、視覚に障害のある中高生のための科学イベント「科学へジャンプ・サマーキャンプ」において、ルービックキューブのワークショップを行った。UV印刷によって記号を印字した点字シールをルービックキューブに貼り、視覚障害者が遊べるように工夫した。ワークショップでは数式処理システム Gap を用いて組合せ総数やある操作の位数計算などを実行し、生徒達は数学がパズルに応用できることを学んだ [3]。これらの経験はモバイル端末や数式処理システムが教育に応用可能であることを我々に示唆してくれた。

2 タブレット端末と数式処理システム

教育用コンピュータに必要な要素として、(1) ポータビリティ、(2) 瞬時にオン・オフが可能、(3) 高解像度、(4) 操作が単純で直感的、が挙げられる。現在のタブレット端末はこれらを全て満たしており、教育

*fujimoto@fue.ac.jp

用コンピュータとして最適であると考える。既に多くの教育用アプリがタブレット端末用に開発され、数式処理システム（Computer Algebra System、以下 CAS と記述）も実装されている。現在入手可能な CAS アプリを CAS エンジンの利用形態で分類すると、以下の 4 つに分けることができる。

1. タブレット端末内の CAS エンジンを利用
2. 他のマシン上の CAS エンジンをネイティブアプリから利用
3. 他のマシン上の CAS エンジンを Web ブラウザから利用
4. ワークシート内に CAS カーネルを埋め込んで利用

現在のタブレット端末は CPU の周波数が 1GHz 以上であり、メモリも 2GB 有するものが主流であることから、1 や 4 の利用形態でも支障なく動作する。

3　サンプルアプリ Mobile CAS の仕様

本稿で実装するサンプルアプリ Mobile CAS の仕様は以下の通りとする。

- **ターゲットデバイス**：Android tablet
- **CAS エンジンの利用形態**：タブレット端末内の CAS エンジンを利用
- **CAS エンジン**：Risa/Asir
- **CAS エンジンとの通信方法**：File I/O
- **GUI**：QtQuick

すなわち、ネイティブアプリとしてタブレット端末に CAS を実装し、それと File I/O を用いて通信する GUI を作成する。GUI は以下の図のように、“Input field”、“Execute button”、“Output area”から成るシンプルなものとする。QtQuick を用いることにより、全く同じソースコードで Android や Windows タブレットという異なったプラットフォームに共通な GUI（図 1）を作成する。

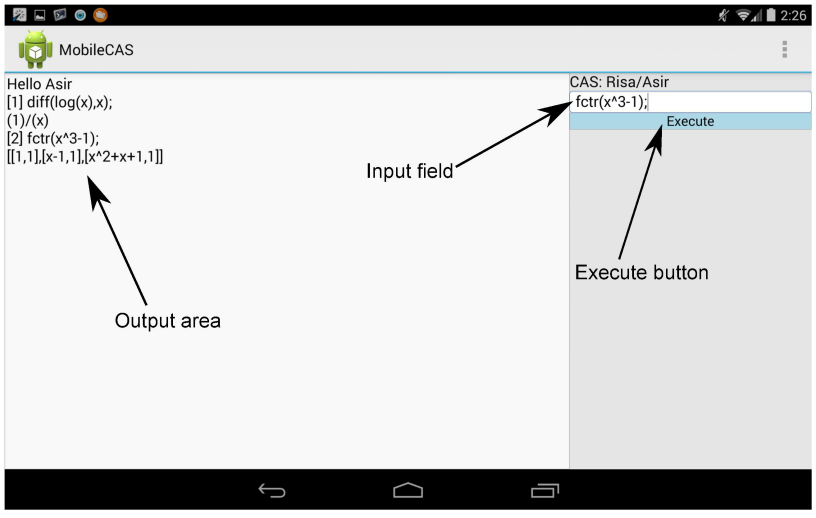


図 1: Mobile CAS の GUI

4 Androidへの既存数式処理システムのポーティング

既存のCASの多くは、UNIXベースのオペレーティングシステム上で動作するように開発されてきた。それらはCやC++などの言語で記述されており、LinuxなどのUNIX系OSを搭載した一般的なPCであれば、gcc/g++を用いて簡単にソースからビルドすることが可能である。AndroidはLinuxカーネルを使用するUNIX系OSであるため、既存のCASをポーティングすることが可能である。この節では、その方法について解説する。

一般的に、タブレット端末用のアプリはPC上のクロスコンパイル環境でビルドされる。Androidタブレットについても、Googleが2つのクロスコンパイル環境 Android SDK と Android NDK を提供している。Android SDK(Software Development Kit) は Java によるアプリ開発環境であり、通常のアプリはこれを用いて作成する。しかし、この場合はDalvikまたはARTというJava仮想マシンが実行時に必要となるため、高速な処理を必要とするアプリ開発には向かない。高速な処理が必要な場合は、Android NDK(Native Development Kit) を利用してCやC++で開発する。

ここで、誰もがAndroid NDKを利用すれば既存CASをポーティングできるのではないかと考えるだろう。しかし、ほとんどの既存CASはAndroid NDK上でビルドできない。それは、Android本体やAndroid NDKが使用するCライブラリが通常のglibcではなく、機能限定版のBionic libcであることが原因の一つにある[1)]。また、CASはガベージコレクション BoehmGC や多倍長整数演算 GMP などの外部ライブラリを必要とすることが多い。外部ライブラリの中には、クロスコンパイル環境でビルドされることを想定としていないものも存在する。これらのトラブルを避けるために、我々は arm rootfs・QEMU・chroot を用いたクロスコンパイル環境（以下、ARM仮想環境）を提案する。

4.1 ARM仮想環境の構築

Androidタブレットの多くがARM系のCPUを搭載している[2)]。そこで、Linux系ディストリビューションの一つである Debian/x86-64 が動作している PC に、次のように Debian/arm 用のルートファイルシステム (**arm rootfs**) を作成する。

```
$ sudo apt-get install debootstrap
$ mkdir armel_wheezy
$ sudo debootstrap --foreign --arch armel wheezy armel_wheezy \
> http://ftp.debian.org/debian/
```

次に、x86環境でARM用コードを実行可能なエミュレータ **QEMU** のスタティックリンク版を arm rootfs にコピーする。QEMUにはOS全体をエミュレートするフルシステムエミュレーションと一つのバイナリのみをエミュレートするユーザーモードエミュレーションがある。ここでは、ARM用バイナリを実行しようとした時に自動的にユーザーモードの QEMU が動いて実行するようにする。

```
$ sudo apt-get install qemu qemu-user-static
$ sudo cp /usr/bin/qemu-arm-static armel_wheezy/usr/bin/
```

最後に、再度 `debootstrap` を実行して arm rootfs 作成処理を完了する。

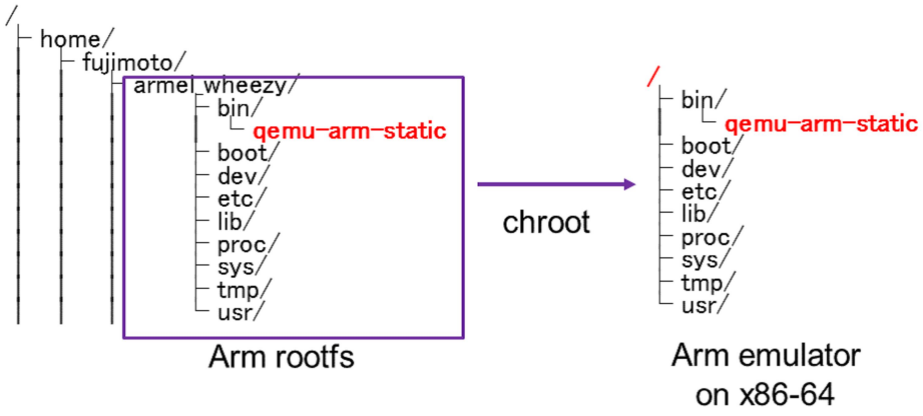
[1)] glibc ではなく Bionic libc のような独自ライブラリを利用することになった理由として、Android を搭載するスマートフォン及びタブレットが通常の PC と比較して非力なことや、GPL/LGPL/BSD といったライセンスの問題がある。

[2)] ARM 系の CPU と異なる Intel の x86 系 CPU が搭載された Android タブレットも存在するが、x86 用 Android には Binary Translator という ARM 用コードを実行する仕組みが搭載されている。

```
$ sudo chroot armel_wheezy /debootstrap/debootstrap --second-stage
```

chroot コマンドは、仮想的にあるディレクトリをルートディレクトリに変更する。arm rootfs のトップディレクトリ armel_wheezy 内で **chroot** コマンドを実行すると、armel_wheezy はルートディレクトリになる。

```
$ sudo chroot ./armel_wheezy
```



4.2 クロスコンパイル環境の調整

上で作成した ARM 仮想環境には、C コンパイラなどの開発ツールがインストールされていない。そこで、**chroot** コマンドで ARM 仮想環境に入ってから、以下のコマンドを実行してツールを整える。

```
$ apt-get install build-essential m4 bison file
```

Android は通常の Linux とディレクトリ構造が若干異なり、シェルの位置が /bin/sh ではなく /system/bin/sh になっている。そこで、これに合わせて ARM 仮想環境内のヘッダーファイル /usr/include/paths.h も以下のように変更する。

```
#if defined(ANDROID)
#define _PATH_BSHELL "/system/bin/sh"
#else
#define _PATH_BSHELL "/bin/sh"
#endif
```

また、ARM 仮想環境内での configure スクリプトを実行する際にシェルが呼び出される場合があるので、次のようにシンボリックリンクも作成しておく。

```
$ mkdir -p /system/bin
$ ln -s /bin/sh /system/bin/sh
```

4.3 ARM 仮想環境内での Risa/Asir のビルド

本稿で作成するサンプルアプリ Mobile CAS の CAS エンジンには Risa/Asir [4] を用いる。Android 上で動く Risa/Asir バイナリを作成するためには以下のようにする[3]。

```
$ sudo chroot ./armel_wheezy
# cd /home/devel/asir2000
# export CFLAGS="-O2 -Wall -D ANDROID -fsigned-char -static"
# ./configure --prefix=/data/data/com.spartacusrex.spartacuside/files/local
# make
# make install
# make install-lib
# file ./asir
./asir: ELF 32-bit LSB executable, ARM, version 1 (SYSV), statically linked,
for GNU/Linux 2.6.26, not stripped
```

このように、まるでセルフコンパイル環境を使っているかのようにソースコードからビルドすることができる。最後の file コマンドでは、生成された Risa/Asir バイナリが ARM バイナリであることを確認できる。そして、この Risa/Asir バイナリは以下のように ARM 仮想環境上で実行できる。

```
root@debian7:/home/devel/asir2000# ./asir
This is Risa/Asir, Version 20140224 (Kobe Distribution).
Copyright (C) 1994-2000, all rights reserved, FUJITSU LABORATORIES LIMITED.
Copyright 2000-2007, Risa/Asir committers, http://www.openxm.org/.
GC 7.2 copyright 1988-2012, H-J. Boehm, A. J. Demers, Xerox, SGI, HP.
PARI 2.0.17, copyright 1989-1999, C. Batut, K. Belabas, D. Bernardi,
   H. Cohen and M. Olivier.
[0] fctr(x^3-1);
[[1,1],[x-1,1],[x^2+x+1,1]]
[1]
```

後はこのバイナリを Android タブレットにコピーすればよい。この ARM 仮想環境を用いることにより、Android 用 Gap や Singular などの他の既存 CAS もビルドできる。

5 GUI の作成

ターミナルアプリで動作するコンソール版の CAS は CAS エンジンとして大変有用である。しかし、タブレット端末のターミナルアプリで CAS エンジンを使用するのは効率的ではない。やはり GUI が必要である。現在、いくつかのマルチプラットフォームな GUI フレームワークが存在する。その一つは WebView である。WebView は WebKit のコアクラスでありながら、通常のアプリの GUI としても利用可能となっている。Maxima on Android [5] はこれを用いて実装されており、MathJax [6] による数式表示が可能な GUI を有する。

---

[3] ここで作成したバイナリは、Android 用のターミナルアプリ TerminalIDE 内で利用することを想定したもので、http://www.fue.ac.jp/~fujimoto/asiroid/ からダウンロードできる。

もう一つのフレームワークとして、Qt（キュート）が挙げられる。Qt は 1994 年に Trolltech が開発したマルチプラットフォームなアプリケーションとユーザーインターフェイスのためのフレームワークであり、デスクトップ（Windows / Linux / Mac OS X）、組込システム（Windows Embedded / Embedded Linux / QNX / VxWorks）、モバイル（Android / iOS / Windows RT）の異なるプラットフォームに対して、共通の C++ソースコードで GUI だけでなくアプリ自体も作成することができる。現在、Qt は The Qt Company[4] によって開発とメンテナンスが行われており、C++に加えて QML(Qt Meta-Object Language) という CSS 風の独自言語を用いて簡単に GUI を作成できるようになった。この GUI の作成に特化したフレームワークは QtQuick と呼ばれている。この節では、QtQuick を使用した GUI の作成方法を解説する。

5.1 QtQuick を利用するための準備

QtQuick を使用して Android 用アプリを開発するには以下のツールが必要である[5]

- Java SE Development Kit (JDK) version 6 or later
- Apache Ant version 1.8 or later
- Android NDK
- Android SDK
- Qt 5.3

以上のツールをインストールした後、Qt の統合開発環境である Qt Creator を起動し、**Tools** メニューから **Options** をクリックして現れるダイアログボックスの **Android** の項目で各ツールの保存場所を入力する。

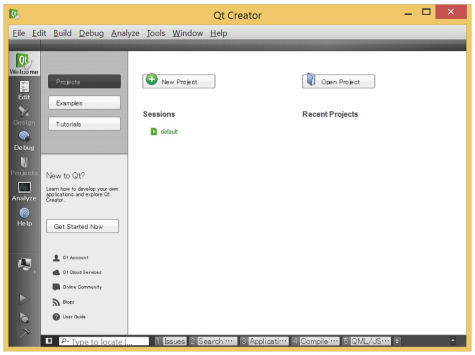


図 2: Qt Creator

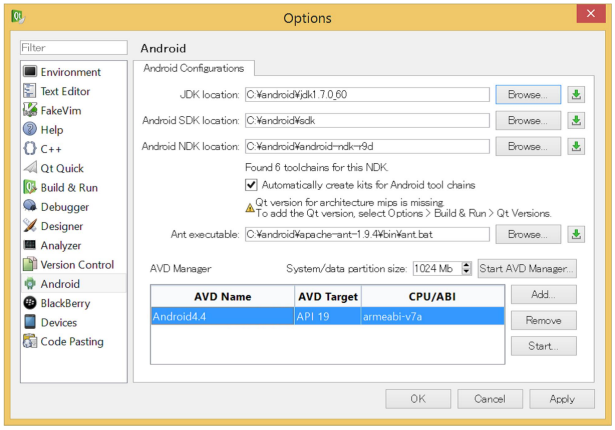


図 3: 各ツールの保存場所を入力

[4] http://www.qt.io

[5] これらのツールは以下のサイトから無償で入手可能である。
http://www.oracle.com/technetwork/java/javase/downloads/
http://ant.apache.org/bindownload.cgi
https://developer.android.com/tools/sdk/ndk/
https://developer.android.com/tools/sdk/
http://www.qt.io/download-open-source/#。

5.2 Qt アプリの作成

準備が整ったので、CAS エンジンを呼び出す Qt アプリの作成を行う。大まかな流れは以下の通りである。

1. QML による GUI の作成
2. QML で実現できない機能を C++プラグインとして用意
3. QML に C++プラグインをインポート

5.2.1 GUI の作成

Qt Creator を起動し、[New Project] → [Applications] → [Qt Quick Application] を選択する。プロジェクト名は「MobileCAS」、Qt Quick component set は「Qt Quick Controls 1.2」とする。これで MobileCAS プロジェクトが作られ、QML による GUI のサンプルとして以下の main.qml ファイルが自動生成される。

```
import QtQuick 2.3
import QtQuick.Controls 1.2

ApplicationWindow {
    visible: true
    width: 640
    height: 480
    title: qsTr("Hello World")

    menuBar: MenuBar {
        Menu {
            title: qsTr("File")
            MenuItem {
                text: qsTr("&Open")
                onTriggered: console.log("Open action triggered");
            }
            MenuItem {
                text: qsTr("Exit")
                onTriggered: Qt.quit();
            }
        }
    }

    Text {
        text: qsTr("Hello World")
        anchors.centerIn: parent
    }
}
```

デフォルトのWindowサイズが`width`と`height`に、Windowタイトルが`title`に指定されているので、適切なサイズと名前に変更する。一番下にある`Text`エレメントの部分がGUIのメインコンテンツである。これを以下の`Rectangle`エレメントに変更し、親Windowの7割の幅を持つOutput areaを左側に作成する。

```
Rectangle {
    id: outputArea
    anchors.left: parent.left
    width: parent.width * 0.7
    height: parent.height
    border.color: "DarkGray"
    border.width: 1
    TextArea {
        id: output
        width: parent.width
        height: parent.height
        font.pointSize: 18
        text: qsTr("Hello Asir")
    }
}
```

さらに、以下のコードを追加して、親Windowの右側にInput fieldとExecute buttonを作成する。

```
Column {
    id: inputArea
    anchors.left: outputArea.right
    width: parent.width * 0.3
    Text {
        font.pointSize: 18
        text: qsTr("CAS: Risa/Asir")
    }
    TextField {
        id: input
        width: parent.width
        font.pointSize: 18
        text: qsTr("input")
    }
    Button {
        id: executeBtn
        width: parent.width
```

```
            text: "Execute"
            style: ButtonStyle {
                label: Text {
                    verticalAlignment: Text.AlignVCenter
                    horizontalAlignment: Text.AlignHCenter
                    font.pointSize: 16
                    text: executeBtn.text
                }
            }
        }
    }
```

GUI の作成に必要なボタンやメニューなどの部品は「コントロール」と呼ばれる。一番最後の Button エレメントは QtQuick コントロールの一つであり、スタイルを変更するために ButtonStyle エレメントを使用している。これを有効にさせるには、QML ファイル main.qml の先頭部分に次のインポート文を追加する必要がある。

```
import QtQuick.Controls.Styles 1.2
```

5.2.2 C++プラグインの作成

上で作成した GUI から外部プログラムである Risa/Asir を CAS エンジンとして呼び出したい。しかし、QML には外部プログラムや外部ファイルにアクセスする機能がない。この問題をクリアするために、我々は C++プラグインを作成する。

Qt Creator で、[New Project] → [Libraries] → [Qt Quick 2 Extension Plugin] を選択する。プロジェクト名を「cmdlaunch」、クラス名を「Launcher」、URI を「org.inftyproject.launcher」とする。この URI はプラグインをインポートする際に必要となる。そして、Qt Creator 上で、ヘッダーファイル launcher.h とソースファイル launcher.cpp を次のように修正する[6)]。

[6)] これらのファイルは http://askubuntu.com/questions/288494/run-system-commands-from-qml-app の議論を参考にした。

launcher.h

```
#ifndef LAUNCHER_H
#define LAUNCHER_H

#include <QObject>
#include <QProcess>

class Launcher : public QObject
{
    Q_OBJECT
public:
    explicit Launcher(QObject *parent = 0);
    Q_INVOKABLE QString launch(const QString &program);

private:
    QProcess *m_process;
};
#endif // LAUNCHER_H
```

launcher.cpp

```
#include "launcher.h"

Launcher::Launcher(QObject *parent) :
    QObject(parent),
    m_process(new QProcess(this))
{
}

QString Launcher::launch(const QString &program)
{
    m_process->start(program);
    m_process->waitForFinished(-1);
    QByteArray bytes = m_process->readAllStandardOutput();
    QString output = QString::fromLocal8Bit(bytes);
    return output;
}
```

これを Qt Creator でビルドする際、「`No rule to make target ...`」というエラーが出る場合がある。このときは、設定ファイル cmdlaunch.pro 中の以下の部分を削除するとよい。

```
!equals(_PRO_FILE_PWD_, $$OUT_PWD) {
    copy_qmldir.target = $$OUT_PWD/qmldir
```

```
    copy_qmldir.depends = $$_PRO_FILE_PWD_/qmldir
    copy_qmldir.commands = $(COPY_FILE) \"$$replace(copy_qmldir.depends, /,
    $$QMAKE_DIR_SEP)\" \"$$replace(copy_qmldir.target, /, $$QMAKE_DIR_SEP)\"
    QMAKE_EXTRA_TARGETS += copy_qmldir
    PRE_TARGETDEPS += $$copy_qmldir.target
}
```

このプラグインをQMLから利用するために、上で得られた2個のファイル qmldir と libcmdlaunch.so を参照可能なフォルダにコピーする。Windows環境であれば次の場所に置けばよい。

```
C:\Qt\Qt5.3.2\5.3\android_armv7\qml\org\inftyproject\launcher
```

ここで作成した「外部プログラム呼び出しプラグイン」の他に、「PATH設定プラグイン」と「File I/Oプラグイン」が必要となる。これらのプラグインは `http://www.inftyproject.org/issac2014/android_plugins.zip` からダウンロード可能である。

5.2.3 C++プラグインのインポート

上で作成したC++プラグインをQMLファイルにインポートして、CASエンジンとFile I/Oによる通信ができるようにする。まず、QMLファイル main.qml の先頭部分に次のインクルード文を追加する。

```
import org.inftyproject.launcher 1.0
import org.inftyproject.fileio 1.0
import org.inftyproject.envset 1.0
```

次に、`ApplicationWindow` 内に各プラグインのエレメントを作成する。

```
    Launcher {
        id: launcher
    }
    FileIO {
        id: inFile
        source: "/sdcard/tmp/input.txt"
        onError: console.log(msg)
    }
    FileIO {
        id: outFile
        source: "/sdcard/tmp/result.txt"
        onError: console.log(msg)
```

```
    }
    EnvSet {
        id: pathEnv
    }
```

そして、外部プログラムへのPATHを設定するために、以下のようにColumnエレメント内のTextエレメントを修正する。

```
Text {
  font.pointSize: 18
  text: {
    pathEnv.setValue("PATH",".:/data/data/com.spartacusrex.spartacuside/files/bin:"
                     +pathEnv.value("PATH"))
    qsTr("CAS: Risa/Asir")
  }
}
```

さらに、Buttonエレメント内に以下を追加する。

```
  QtObject {
      id: inputNo
      property int i:0
      function next(){
          i++;
          return i;
      }
  }
  onClicked: {
      outFile.write("")
      inFile.write("R=" + input.text + "output(\"/sdcard/tmp/result.txt\")$R;quit;")
      output.append("[" + inputNo.next() + "] " + input.text)
      launcher.launch("asir -f /sdcard/tmp/input.txt")
      output.append(outFile.read())
  }
```

inFile.write(...)でRisa/Asir用のインプットファイルinput.txtを作成し、launcher.launch(...)で外部プログラムであるasirを起動してinput.txtを処理させている。計算結果はファイルresult.txtに書き込まれるので、それをoutFile.read()で読み取ってOutput areaに表示する。次の図はこの計算プロセスを表したものである。

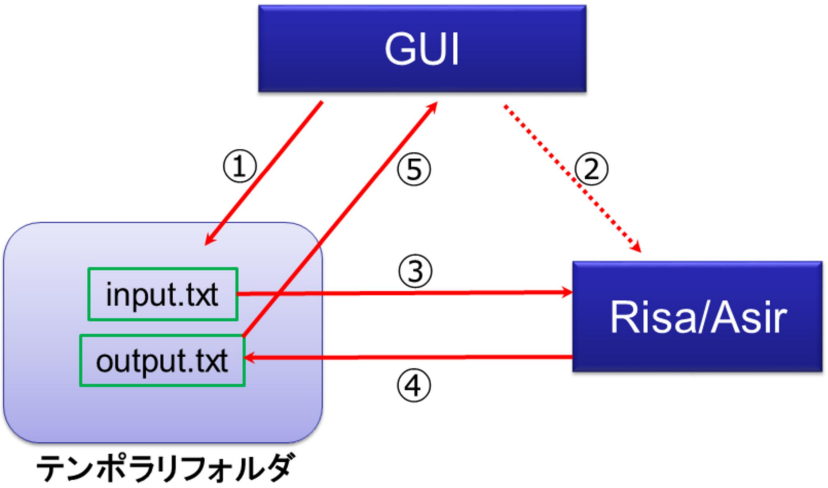


図 4: Mobile CAS の計算プロセス

6 残された課題 – Mobile CAS をどのように発展させるか

本稿ではタブレット端末に CAS を実装する手法として、Android タブレットを例に CAS エンジンと GUI を別々に作成する方法を解説した。GUI にはマルチプラットフォームな GUI フレームワークを使用しているので、CAS エンジンさえ用意できれば他のデバイスでも同じアプリが動作する。実際、筆者の研究室では Windows 8 タブレット、Mac OS X、Linux マシンで Mobile CAS が動いている。

このように、本実装手法はコンソール版の CAS を GUI 化するための手法としても有用である。しかし、Mobile CAS にはいくつかの課題が残されている。

1. **CAS エンジンとの通信方法の改善**

   Mobile CAS は CAS エンジンとの通信に File I/O を使用している。この File I/O による通信は最も簡単な実装方法であるが、計算毎にセッションを閉じてしまうので効率的ではない。計算途中で安全に中断する機能を実装するためにも、OpenXM プロトコル [7] のような通信手法を利用するべきであろう。

2. **CAS エンジンとの切替機能の実装**

   Mobile CAS は Risa/Asir を CAS エンジンと利用しているが、外部プログラムとして呼び出し可能なコンソール版 CAS であれば、どんな CAS も計算エンジンに利用できる。Mobile CAS の右上の「CAS: Risa/Asir」と表示している部分をドロップダウンリストに変更し、そこから様々な CAS エンジンを選択できるようにするべきと考える。

3. **手書き数式インターフェイスの実装**

   Mobile CAS における数式の入力はキーボードから行うが、タブレット端末のソフトウェアキーボードは英字と数字の切替操作が必要であり効率的とは言えない。筆者は AsirPad [1] において手書き数式インターフェイスの実装経験があり、この C++ソースコードを元に QML で手書き数式インターフェイスを実装中である。

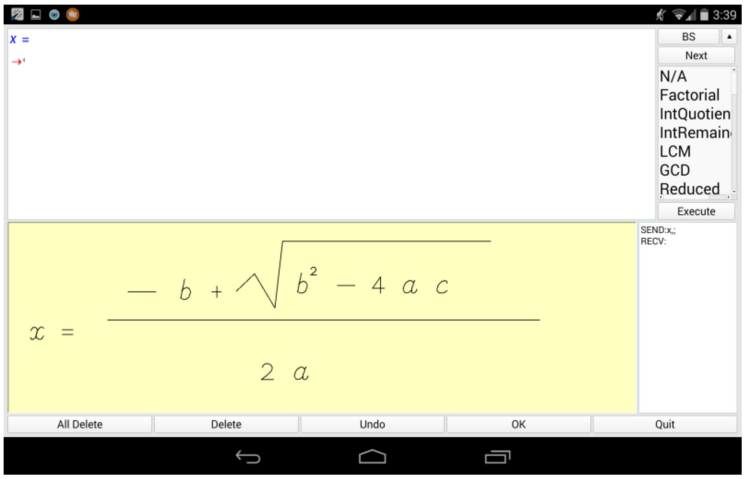


図 5: 実装中の手書き数式インターフェイス

参 考 文 献

[1] M. Fujimoto and M. Suzuki, AsirPad - A Computer Algebra System with a Pen-based Interface on PDA, Proceedings of the Seventh Asian Symposium on Computer Mathematics, Korea Institute for Advanced Study (2005) pp. 259–262.
Demo movie, `http://www.inftyproject.org/demo/AsirPad_Demo.zip`

[2] 藤本光史, 鈴木昌和, 金堀利洋, PDA と手書き数式インターフェースを用いた実践授業について, SSS2006 情報教育シンポジウム論文集, IPSJ Symposium Series Vol.2006, No.8 (2006) pp. 331–338.

[3] 視覚障害者の数式処理を用いた Rubik's Cube 解法学習の試み, 京都大学数理解析研究所講究録 1666, 「Computer Algebra – Design of Algorithms, Implementations and Applications」(2009) pp. 183–188.

[4] M. Noro, et al., *A computer algebra system Risa/Asir*, `http://www.math.kobe-u.ac.jp/Asir/asir.html`

[5] Y. Honda, Maxima on Android, `https://sites.google.com/site/maximaonandroid/`

[6] MathJax, `http://www.mathjax.org`

[7] M. Maekawa, M. Noro, N. Takayama, Y. Tamura and K. Ohara, The Design and Implementation of OpenXM-RFC 100 and 101, Computer Mathematics (Proceedings of the Fifth Asian Symposium on Computer Mathematics), World Scientific (2001) pp. 102–111.