A cross-application architecture for pen-based mathematical interfaces

Elena Smirnova* Stephen M. Watt
Ontario Research Centre for Computer Algebra
University of Western Ontario
London ON, Canada N6A 3L8
{elena,watt}@orcca.on.ca

Abstract

We address the problem of organizing pen-based environments for processing handwritten mathematics. Our approach is based on providing portable ink-aware mathematical interfaces to existing document processing software and mathematical packages. Our architecture includes components for on-line ink collection, mathematically-oriented recognizers, portability support, and interfaces to applications. We present a pen-based computing environment Mathink *and give an overview of its facilities for training, ink annotation and performance testing.*

1. Introduction

Determining the most effective human-computer interface for mathematics is a problem that has been addressed by various authors, including [4, 8], and one of the conclusions is that many users would prefer to input expressions using a pen instead of a keyboard. Furthermore, mathematical content entered with a digital pen can be processed in many interesting ways, including editing, validation and semantically-driven direct manipulation. To provide these functions, pen-based interfaces for mathematics must incorporate a number of capabilities, including collecting and processing of digital ink, recognition of handwritten expressions and connection to mathematical engines.

In this article we present our architecture for mathematical pen interfaces. In Section 3 we summarize the main features of our pen-based mathematical computing system, Mathink, which we have developed to validate our architectural approach. In Section 4, we consider two mechanisms for communicating between pen-based front-ends and mathematical computation engines. We also describe, in Section 5, the tools we have developed for system training and testing, and show how they can be used to improve performance in recognition.

*Present address: Texas Instruments, e-smirnova@ti.com

2. A Framework for Pen-Based Computing

We begin by summarizing certain principal purposes and key requirements of systems to be built for pen-based computing. We describe a framework architecture designed to implement such systems allowing high-quality handling of digital ink, while ensuring portability across platforms and applications.

2.1. Objectives

The objective of our work has been to identify and investigate the issues whose resolution will lead to effective pen-based mathematical computation. This has meant investigating a number of questions and approaches to different problems, from ink collection, to character recognition, expression analysis and manipulation. From the outset, we have recognized that in each of these areas there are important, difficult questions and that a full solution will require research in many domains.

From the beginning, we must emphasize that our goal was not to create a stand-alone mathematical recognizer nor to develop an application-specific plug-in. On the contrary, we have studied how to provide a uniform interface to many applications through one component. For concreteness, we have identified two classes of hosting environments: computer algebra systems and rich document editors.

Modeling a mathematical ink-handling component as one that may be used in many contexts places a number of constraints on system architecture design. We require that such a pen-based interface be portable across a range of platforms without sacrificing digital ink quality. In previous work [5], we have presented an architectural solution that allows one to embed device-specific support for digital ink and to ensure compatibility with hosting applications on various platforms. What we present here is based on this approach.

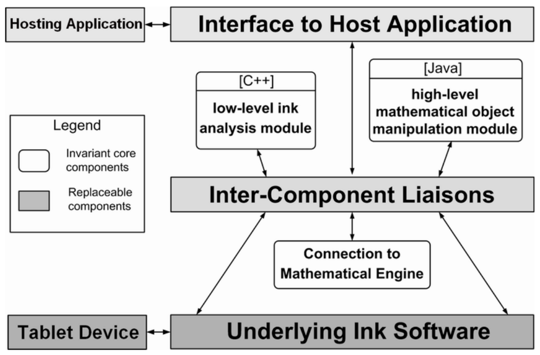
Hosting Application
Interface to Host Application
[C++]
low-level ink analysis module
[Java]
high-level mathematical object manipulation module
Legend
Invariant core components
Replaceable components
Inter-Component Liaisons
Connection to Mathematical Engine
Tablet Device
Underlying Ink Software

Figure 1. Top-level framework organization

2.2. Architecture Organization

The key point of our design is to separate the components responsible for the analysis of handwritten mathematics from the modules that provide connection to hosting applications and underlying platforms (Figure 1). In this way, core recognition components can remain invariant, while interfaces and adapters can be replaced for each combination of platform and hosting application. This allows re-use of the main capabilities of the pen-based framework in different environments without changing their internal organization.

Beyond being resource intensive, development and maintenance of recognition modules requires expertise in mathematical handwriting analysis. On the other hand, implementation of the "glue" components to plug recognition units into hosting environments does not demand any specific knowledge in mathematical recognition. Therefore, docking mechanisms can be developed independently by experts in the separate areas.

In our experiments with target platforms and applications, we have arrived at two core modules for character recognition and structural analysis. Our choice of implementation languages for these components had to satisfy two principal criteria: platform portability and support of the functionality required by the modules. For the character recognition unit, we chose C++ because of the high performance of it compilers for computation-intensive tasks. Structural analysis, however, involves less computation. It requires rich functionality for expression manipulation and sufficient XML support; therefore, for the second module we chose to use the Java language.

Another important component of the framework is an interface to mathematical engines. As well as recognizing ink input and translating it to a valid mathematical expression, we expect our pen-based component to allow further computation and manipulation with the formulae recognized. To provide this functionality we need to enable a mathematical back-end in our framework. Instead of building yet another symbolic computation system we have decided to make use of already existing powerful and well-developed symbolic computing packages. To satisfy the portability criteria, among the available computer algebra systems we consider only those which are supported on multiple platforms. We discuss connection mechanisms to mathematical software systems in Section 4.

3. Evaluating on Approach with Mathink

We have conducted experiments with our architecture on the Windows XP platform, using .NET Tablet PC SDK [6] for collecting and pre-processing high-resolution digital ink. In this section we introduce Mathink, an ink-aware mathematical component implementing an architectural approach we have described above. Mathink is designed both as mathematical pen-based plug-in and as an experimental environment to train and test mathematical recognizers.

3.1. Mathink system overview

Mathink provides a user interface allowing digital ink to be collected from a pen, mouse or other device. The system supports a range of ink sources, including both pressure-sensitive and touch-activated digitizers. We have successfully tested the system with Tablet PCs, SmartBoard, $\mathbf{e}^{3}_{\text{works}}$ Stylo USB tablet and Wacom digitizers.

The flow of control through the Mathink system is organized as follows: Ink glyphs are entered in the writing area is transferred to the system after an adjustable time delay. The writer thus is provided with immediate feedback from the recognizer for every character entered. Along with the best match, the recognizer offers other high ranked candidates. These are shown in a fixed location that allows rapid selection of alternatives, whose total number can be preset by the user.

Recognition results accepted by the user are displayed on top of or instead of the original ink. Having these two options allows users to switch between the ink input and typeset recognition results. When in typeset mode, the recognized content is drawn within the bounding box of original ink strokes. The best fit is calculated by the Mathink rendering module. This involves adjusting the size of the characters on a given baseline, while taking into account positioning of large grouping operators, fractions and radicals.

Layout of the expression is re-analyzed with each new character entered. Once the user has finished writing a formula the system finalizes and refines its structure. It is then parsed to a standard mathematical format, such as Presentation MathML. Once the mathematical content is constructed from a handwritten input the user can choose to send it to a symbolic computation back-end to perform further computation such as evaluation, simplification, solving, etc.

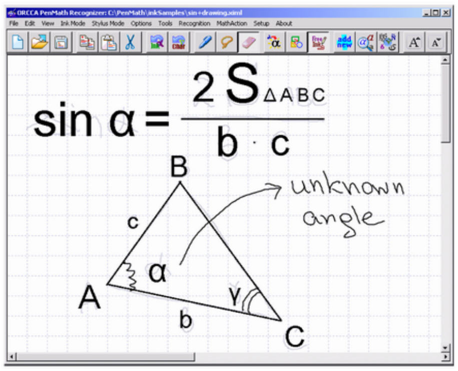
ORCCA PenMath Recognizer: C:\PenMath\inkSamples\sin+drawing.isf
File Edit View Ink Mode Stylus Mode Options Tools Recognition MathAction Setup About
$\sin\alpha = \frac{2 S_{\Delta ABC}}{b \cdot c}$
B
c
α
A
γ
b
C
unknown angle

Figure 2. Mathematical ink document with multiple recognition options

In addition to the main feature of mathematical expressions analysis, the Mathink system offers a drawing mode. In this mode ink input is recognized as basic geometrical shapes instead of mathematical characters. In addition, Mathink also allows to enter "free ink" that is not sent to the recognizer. Switching between the modes is permitted at any time, so a combination of recognized formulae, geometrical drawing, quick notes and arbitrary sketches can be placed in the same document, as shown in Figure 2.

As opposed to the approach taken in the InftyEditor [3], we do not discard collected ink after it has been recognized. The original strokes along with their annotations can be saved in Microsoft Ink Serialized Format (ISF) or using portable system-independent standard InkML [1]. Stored ink can be re-opened later in Mathink editor as a PenMath document or played back using built-in InkPlayer tool. This possibility allows evaluating of different combinations of recognizers and settings on the same hand-written example. In particular, we use this ability to test and optimize the performance of the system, as discussed in Section 5.

3.2. Exporting Mathink as plug-in control.

As we stated at the beginning, our main goal is to enable pen-based mathematical interfaces to computer algebra systems, rich document editors and other applications. To do this we explored methods for the integration of ink system with various software packages. Among the possible approaches, we studied methods of exporting the Mathink interface as a standard plug-in control compatible with target applications.

While the functionality exposed by a plug-in control should remain as invariant as possible across applications, the type and internal organization of a plug-in component in most cases is determined by the hosting system. We chose Microsoft Office as a suitable platform to evaluate our approach with document processors. As the second independent approach, we experimented with MAPLE 10 to demonstrate a pen-based mathematical interface in computer algebra packages. These choices led us to two forms of plug-ins: as an *ActiveX control* to be used with Microsoft applications and as *JavaBeans* to connect the Mathink interface in MAPLE.

Because the Mathink system involves a fair amount of managed `.NET` code, it cannot be directly compiled into *ActiveX control*, nor exported to Java. Therefore, first, we had to expose the Mathink `.NET` component via *COM Interop* as a *COM object*.

Further, to enable ink interface in an *ActiveX control*, we had to create an intermediate *Win32* component to host the `.NET` ink collector. Once compiled to an *ActiveX control*, the Mathink plug-in can be integrated to Microsoft Office applications, accessible either from standard toolbars or through the "Insert|Object.." menus.

Providing the Mathink interface as a *JavaBean* required connection between `.NET` and Java platforms. Existing commercial and open source packages providing a link between `.NET` and Java are known to lag behind one or both of the platforms as they evolve. Our approach uses no third party software. The connection mechanisms we have developed are based on standard protocols that are well defined and properly maintained for both platforms. While remaining simple, this solution suits our purposes.

Having Mathink control exported via *COM Interop*, the "unmanaged" nature of *COM* component allows it exposure to Java via the *Java Native Interface* (JNI). We use the *Abstract Window Toolkit Native Interface* to permit rendering to Java canvas from native code. Finally, we wrapped a JNI adapter within a Java package and exported its main classes as *JavaBeans*. Once the Mathink control is enabled on a Java platform, it can be incorporated into MAPLE worksheets. This not only allows the Mathink component to serve as a pen interface to the MAPLE computer algebra system, but also provides a direct connection to the MAPLE kernel, as we will discuss in the next section.

4. Connection to Math Software Packages

After digital ink has been collected and processed, and the mathematical expression is recognized and parsed to an internal format, the pen-based application moves to the next stage, where the user can manipulate the resulting expression. This manipulation will, in general, imply non-trivial mathematical transformation. This requires providing a connection to a mathematical engine from the handwriting recognition environment. In this section we present two approaches that allow our Mathink component to communicate with the MAPLE symbolic computation package. A similar approach would allow connection with other computer algebra systems (CAS).

4.1. Using the OpenMaple Interface

To send a request for computation to MAPLE from another application, we need to access the MAPLE kernel from outside of the CAS shell. Each request must be expressed as a valid Maple command with all of its arguments encoded in a Maple-compatible format. For this, we use supplementary tools offered by the MAPLE package: Starting with version 9.0, the OpenMaple interface allows access to the computer algebra system functionality via API calls from C++, Java or Fortran. Furthermore, a built-in parser for both Content and Presentation MathML allows importing mathematical data encoded in these formats into MAPLE.

Thus, once the handwritten expression is translated to MathML, it can be used in computation via OpenMaple API calls. To enable support this option in the Mathink environment we have developed a special CAS-communication module. It starts the MAPLE kernel as a background process, wraps the user's request in a MAPLE instruction and passes it along with MathML arguments using the OpenMaple protocol to the MAPLE kernel. The results of the computation are exported as Presentation MathML, sent back to the Mathink system and displayed.

This approach is similar to that used in the MathBrush [2] system, although Mathink has an independent implementation. Currently the Mathink interface allows transmission of recognition results for numeric evaluation, symbolic calculation, simplification and factorization. Nothing new would be required to support a variety of other operations. Mathink does not yet support equation solving, partial differentiation or other operations that require additional semantic analysis to detect variable instances in the the input. By identifying variable names, the user could be offered a choice of operations such as "solve for ..." or "differentiate with respect to ...."

4.2. Mathink Control as an Internal Maple Component

While using the OpenMaple API is relatively easy to implement and natural to use, it has certain restrictions. These are limitations in accessing MAPLE kernel from an outside application and ambiguities arising in conversion of mathematical notations to semantic content. Moreover, as a practical matter, the built-in MAPLE parser for Presentation MathML has a number of limitations.

To overcome these difficulties, we introduced an alternative architectural solution that allows the pen-based component to be run inside the MAPLE environment. As discussed in Section 3.2, it is possible to export the Mathink component as a Java object, which can then be integrated with MAPLE GUI code. From there, the Mathink component has direct access to the MAPLE kernel. This allows the

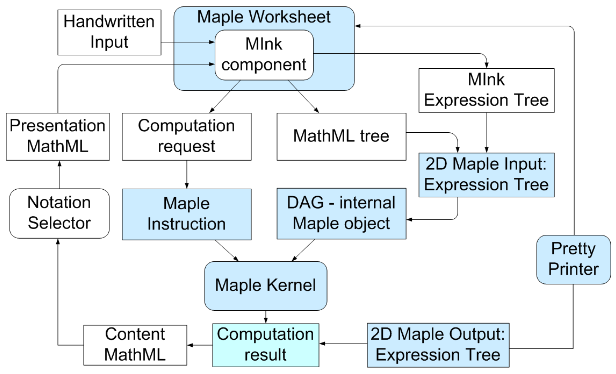

Figure 3. Mathink **accessing Maple kernel as an internal Maple component**

Mathink control to enjoy a larger range of computer algebra system functionality, as it can form more complex and detailed requests by using native MAPLE instructions. Furthermore, hosted inside MAPLE, Mathink can operate with mathematical objects at an internal level of the symbolic engine. This includes direct translation of recognized input to MAPLE structures, as shown in Figure 3.

5. Annotation and Performance Testing Tools

In addition to offering mathematical pen-based interface, the Mathink system provides testing environments to conduct experiments and adjust recognition performance. These include tools for collection of large sets of handwritten samples, facilities for ink annotation and automated performance testing. We have 300 tests that we use to verify or measure the Mathink system whenever it is changed. Automated testing of recognition performance requires a source of valid results to compare with the recognizer output. To do this we annotate collected ink with ground truth.

5.1. Ink Annotation Tool

To assign the ground true to each ink character in a test file, we annotate ink in a semi-automated process supervised by a human user. As discussed in [7], Mathink system uses a hybrid method of character recognition that involves use of local context. To eliminate possible errors in determining context during test runs, we also store the original context information for every ink glyph along with its character value.

The simplest way to provide ink annotation is to run the system recognizer on the whole expression, and then correct mis-recognized characters. Manual correction is done by selecting an incorrectly recognized character and entering its true value from the keyboard or using the Unicode palettes. In addition, the context information for each character in the expression should be verified and, if necessary, adjusted using context annotation control. This method of ink annotation is suitable only for expressions with a high rate of recognition accuracy. If the recognizer is wrong for more than a few characters in an expression, we can use a second method, which we call *annotation copying*.

The annotation copying tool transfers information from an already annotated expression to one being processed. A fully annotated expression is opened in a *master annotation window*, then the character values along with the context information are copied between matching ink entries. This method is especially useful for annotating a large collection of samples provided by users for a fixed set of formulae. In this case, for each formula in the questionnaire only one handwritten sample has to be manually annotated, the rest can be simply copied from the prototype. This process cannot be done completely automatically because there are different orders which may be used to write certain subexpression, such as double scripts, fractions, parenthesis and radical expressions, and these affect the context information.

5.2. Searching for Optimal System Settings

The character recognizer and the structure analyzer depend on a number of tunable parameters, such as threshold values used in baseline and script detection, minimum confidence for a character to be considered as a potential recognition candidate, maximum number of candidates to return from the recognizer to the structure analyzer, *etc*. For some of these parameters a default value can be relatively easily estimated and adjusted after a small number of test runs. However, there are parameters that are impossible to approximate without the results of practical testing on a large amount of data. Weight coefficients used in combination of results from several recognizers and the total recognition confidence with prediction certainty are examples of such parameters. We therefore search for optimal combinations of these coefficients experimentally.

To conduct these experiments, we added an automated testing facility to the Mathink environment. This allows the recognizer to run on handwritten samples, searching the parameter space. As shown in [7], using experimentally determined parameters allows to increase accuracy of on-line mathematical character recognition by up to 9.8%.

6. Conclusions

We have studied methods for organizing pen interfaces for mathematical computing that can be deployed in a variety of environments and have suggested an architectural approach that ensures portability of a pen-based frameworks across platforms and hosting applications. As an instantiation of this architecture, we have presented the Mathink software package for mathematical handwriting recognition. Mathink can be used as pen-based front-end to computer algebra systems and document editors.

There are many interesting questions in the area of pen-based computer interfaces for mathematics, and a number of design questions and configuration parameters are best decided based on empirical data. We have found that it is convenient to incorporate tools for experimental analysis into the Mathink interface for this purpose. As examples of this, we have described a ground truth annotation tool and a tool that searches configuration parameter spaces.

Although pen-based computer interfaces have been studied for decades, handling mathematics presents many domain-specific challenges. We have reached the stage where we can have useful interfaces, but there remain many areas of potential improvement. These range from low-level questions in software architecture to high-level questions using mathematical semantics. Much further experimentation is needed to properly address these questions, so we foresee an ongoing need for rapidly configurable experimental platforms.

References

[1] Y-M. Chee, M. Froumentin, and S.M. Watt (editors), *Ink markup language (InkML)*, World Wide Web Consortium, `http://www.w3.org/TR/2006/WD-InkML-20061023`, 2006.

[2] G. Labahn *et al*, *A preliminary report on the mathbrush pen-math system*, Maple Conference 2006, 2006, pp. 162–178.

[3] Mitsushi Fujimoto *et al*, *Infty Editor a mathematics typesetting tool with a handwriting interface and a graphical front-end to OpenXM servers*, Computer Algebra Algorithms, Implementations and Applications, RIMS, vol. 1335, 2003, p. 217226.

[4] R. Zanibbi *et al*, *Aiding manipulation of handwritten mathematical expressions through style-preserving morphs*, Graphics Interface, 2001, pp. 127–134.

[5] Elena Smirnova and Stephen M. Watt, *A context for pen-based computing*, Maple Conference 2005, Maplesoft, 2005, pp. 409–422.

[6] Rob Jarrett and Philip Su, *Building Tablet PC applications*, Microsoft Press, 2001.

[7] Elena Smirnova and Stephen M. Watt, *Combining prediction and recognition to improve on-line mathematical character recognition*, Tech. report, University of Western Ontario, `http://www.orcca.on.ca/TechReports/TR-06-06`, 2006.

[8] Lucy Zhang and Richard Fateman, *Survey of user input models for mathematical recognition: Keyboards, mice, tablets, voice*, Tech. report, University of California, 2003.