A Context for Pen-Based Mathematical Computing

Elena Smirnova Stephen M. Watt

Ontario Research Centre for Computer Algebra
Department of Computer Science
University of Western Ontario
London Ontario, Canada N6A 5B7
{elena,watt}@orcca.on.ca

Abstract

We report on an investigation to determine an architectural framework for pen-based mathematical computing. From the outset, we require that the framework be integrated with suitable elements for pen computing, document processing and computer algebra. We demonstrate that this architecture is able to provide these interfaces and allows a high degree of platform independence for the development of pen-based mathematical software.

Keywords: Pen computing, computer algebra, Java, .Net, Maple, Microsoft Office.

1. Introduction

1.1. Motivation

With the recent widespread availability of pen-enabled devices (such as Pocket PCs, Tablet PCs and interactive whiteboards), there is an opportunity for a new generation of natural interfaces for mathematical software packages. The use of the pen to enter, edit and manipulate mathematical expressions can lead to a qualitative improvement in the ease of use of computer algebra systems. On the other hand, mathematical input on pen-enabled devices goes well beyond ordinary hand-written mathematics on paper or chalkboard, simply because it can enjoy the rich functionality of software behind the ink-capturing hardware.

We should point out from the beginning that pen-based software for mathematics has unique aspects that distinguish it from text-based applications and that these drive our software architecture with different considerations: First, the set of symbols used in mathematics is much larger than the usual alphabets or syllabaries encountered in natural languages. This is also different than the large character sets of ideographic Asian languages, because there the primitive elements are strokes that must be given in a well-defined order. Second, there is no fixed vocabulary for mathematics that can be used for context-based expression recognition. Third, mathematical notation has a complex two-dimensional structure, often with large symbols grouping parts of an expression. Mathematical handwriting thus ends up with aspects similar to both drawing and writing. These issues require a new approach for pen-based software solutions that handle handwritten mathematics. Ultimately we wish to have a pen-based platform for mathematical expression entry, mathematical editing and calculation.

1.2. Objectives

Many investigators have examined the question of mathematical handwriting recognition, including our group [3][4][5]. We address a different question in this paper: If a pen-based interface for mathematics is to be widely accessible, how should its architecture be framed? Earlier work on pen-based mathematics has not addressed this deployment question.

One of the first issues we face in starting a development project in pen-based interfaces for mathematics is to define the target audience. We have felt that it is too narrow to target solely the group of current computer algebra users. As computer algebra systems become more document-oriented, we find that it is a natural generalization to include the usual document processing audience as part of the scope. By taking this decision, we are led further to architect an environment where the mathematical components of a document can enjoy mathematically aware operations for reformatting, validity checking, etc, in a manner analogous to the natural language operations in high-end document processing software.

Taking into account our final target, we need to think not about developing a particular software solution that enables mathematical input, but rather about a general framework organization that will allow math interfaces for different applications on different platforms. These are to enable plugging ready-to-use pen-based mathematical software tools to hosting applications and systems. As will be discussed here, one of the key points in organizing such architecture we see in keeping a major part of pen-related functionalities invariant while carrying frameworks from one system to another.

The remainder of this article is organized as follows: First we outline the technologies that may be used as elements with an interface for pen-based mathematics. Then we discuss our portability objectives and their implications. Finally we describe the large-scale aspects of the architecture we have designed to satisfy these criteria and detail some of the implementation issues.

2. Overview of Existing Ink Technologies

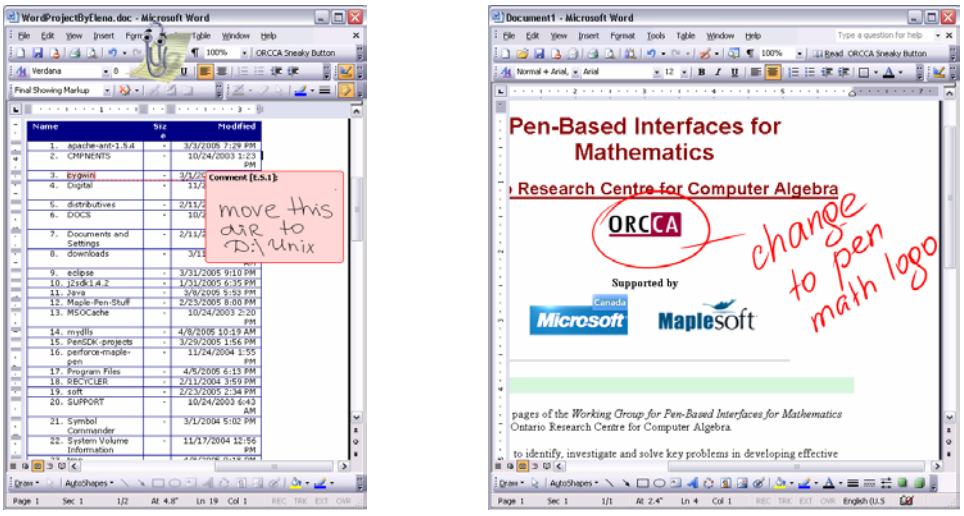
2.1. State Of The Art

Software providers in both the document processing and mathematical computing domains have made first efforts to enable pen-based input in their applications. For example, Microsoft Office 2003 already includes an inking feature, allowing the user to write over the document with a Tablet pen or stylus or to add handwritten pop-up comments to document elements, as shown in Figure 1a.

Maple 10 [8] has also made some first steps toward a pen-based interface, providing both scratchpad and glyph selector elements, as illustrated in Figure 1b. These use low-resolution mouse simulation for these two initial pen-oriented components, treating the tablet pen as a regular mouse-like pointing device and capturing motion events based on screen pixel resolution.

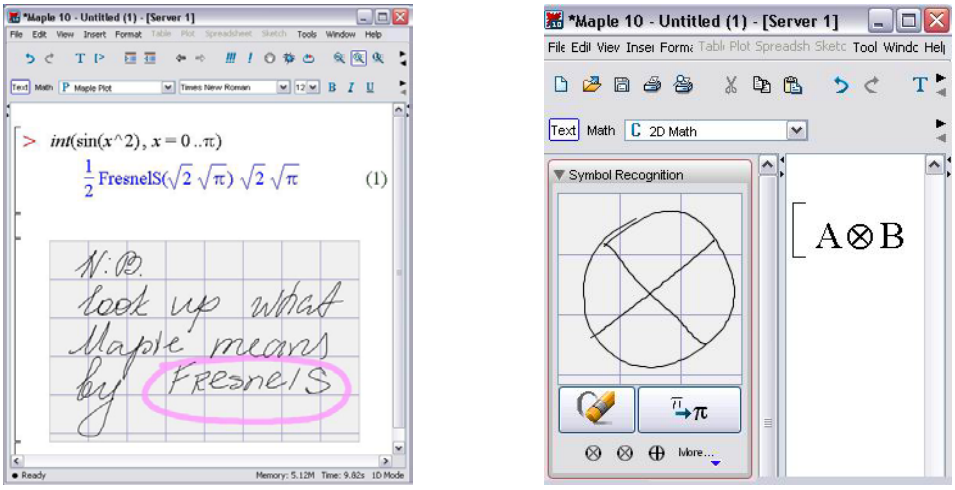
The advantages of having such facilities in Microsoft Office and Maple are obvious. On the downside, neither of these packages provides access to the collected ink, so it cannot be re-used in other applications. Furthermore, neither package exports an application programming interface (APIs) for its ink-management functionality so there is no way to extend the application's ink handling. Finally, neither so far provides ink support for handwritten mathematical expressions.

More generally, at present there is no pen-based support for handwritten mathematics that can be used with a range of applications. Thus we see a need to design interfaces for pen-based mathematical tools suitable for a wide range of applications, including document processing, mathematical computation, diagramming, etc.



ink comments overlay inking

Figure 1a. Ink features in Microsoft Word 2003



scratchpad character selector

Figure 1b. Ink features in Maple 10

2.2. Available Technologies

We examine the components and elements that may be used in a pen-based mathematics solution for our target audience. We will require that we can use some suitable combination of these technologies on the known present primary platforms. It is also important that we choose elements that allow us to achieve various measures of portability, a question we discuss in the next section.

2.2.1. Digitizer Device Drivers

One potential solution to capture pen information on Tablet PCs is the direct use of device-specific drivers. One very popular device is the WACOM digitizer. The WACOM driver provides a low-level packet-based interface to information about pen position in tablet coordinates, pen pressure and state of the pen buttons. It also has two channels reserved for pen tilt and rotation information, but these are not yet supported by most tablets. The lack of information about overall tablet geometry and resolution prevents reliable conversion between tablet and pixel coordinates in the cases where it is needed. Even though this option allows easier connection to general applications, e.g. though direct JNI calls, in general it works at too low a level for our purposes.

2.2.2. Tablet PC SDK

For ink applications our initial primary target platform is the Tablet PC. At present this runs Windows XP Tablet Edition as its operating system and is supported by the Tablet PC Platform SDK library [9]. This library provides access to high-resolution spatial information about pen strokes (about 15 times greater resolution than the mouse), as well as information about pen-pressure, (eventually) angle, and (indirectly) velocity. This library is accessible under the Microsoft .NET architecture [10]. We anticipate that a similar SDK will be available on future versions of Windows, including Longhorn [11]. The use of the Tablet PC SDK increases the opportunities for smooth interaction with other ink-based applications on the Windows platform. Provided with the Tablet PC SDK are language bindings for C#, among other .NET languages. The Tablet PC SDK also comes with COM Automation APIs, providing a link to unmanaged C++ code. These partially cover the functionality of the .NET APIs. Notably missing from the Tablet PC SDK is support for Java programming.

2.2.3. C#/.NET

Any architectural discussion for software to be deployed on Windows should consider C# as a potential implementation language. While C# would provide easy access to pen functionality on Windows, there are two reasons why we have chosen not to use it. First, and most importantly, this would add an extra layer of complexity in dealing with Maple objects. Secondly, while C# implementations and .NET support are becoming available for non-Windows platforms, we expect that these implementations will lag behind their Windows counterparts, and thus present portability problems. Finally, while it is technically possible to use C# within Microsoft Office [15][16], .NET controls may not in general be used in applications (such as MS Office) that are designed to be extended with ActiveX controls [17].

2.2.4. Maple

An interface for pen-based mathematics will be required to perform transformations on its input. While we might imagine pen-based mathematics component that incorporated all of its own mathematical manipulation capabilities, this would be short sighted. Just as high-end a document

processing application will make use of dictionaries and grammatical knowledge, a sophisticated interface for mathematical computing will need to make non-trivial mathematical transformations. Since even simply specified mathematical transformations, such as factorization or integration, require sophisticated mathematical libraries, it is inevitable that a pen-based mathematical framework will make use of a computer algebra system. While we wish our architecture to be sufficiently flexible to accommodate different computer algebra systems, we must be able to make effective use of at least one such system. For this we find Maple to be a suitable choice.

We conclude that, even though these technologies individually provide high-levels of functionality useful in ink applications, none is completely suitable for our needs. In particular these elements to not all work together. Our solution will need to combine them in such a way that the final architecture can provide high-quality ink capabilities while remaining portable across platforms and providing the flexibility of easy connection with applications.

3. Main Design Issues

3.1. Portability Criteria

We expect that in an interface for pen-based mathematics, the bulk of the application will be in dealing intelligently with mathematical objects and their transformation. We wish for this code to be portable both *longitudinally*, along the life of a given platform, and *laterally*, across platforms.

For longitudinal portability, we note that pen-computing is still a relatively immature discipline, compare to other areas of computing, and that even if we fix on one platform the APIs will evolve over time. For example, we would expect the Tablet PC SDK of 5 years from now to be at least as different from the present Tablet SDK as the present SDK is from that of the Pocket PC. Given that sophisticated mathematical software packages have lifetimes measured in decades, we therefore must structure our application to limit the impact of these SDK changes.

We have studied the question of lateral portability of ink applications elsewhere [1][2]. While the Tablet SDK presents the richest base environment, there exist base environments for other platforms that fill an analogous role, e.g. for Palm devices, for Mac OS X, and for Linux. We have developed an approach whereby an abstraction layer is used, consisting of abstract classes that are realized by platform-specific wrappers. On each platform some of the abstractions must be implemented *ab initio,* although this is least necessary on the Tablet PC. Access to the native platform recognition software can allow platform-specific conventions (e.g. Palm Graffiti) to be used, if desired. However, under normal circumstances the native character recognition will need to be supplanted with special-purpose mathematical recognizers that take into account the large symbol set and mathematical context.

The last, but not least is to ensure mathematical data portability. This may use MathML and/or OpenMath unified platform-independent formats for encoding mathematics.

3.2. Our Architectural Approach

Our portability criteria have thus moved us in a direction where the majority of our application code remains platform invariant. The "heavy-duty" components that are responsible for dealing with ink processing and mathematical structures manipulation should remain the same for different platforms as well as and their longitudinal evolution. These components are

A. the high-level mathematical object manipulation code
B. the low-level ink analysis code.

The parts that shall vary with the hosting system and platform are

1. platform-specific basic ink software to collect traces and support the abstract ink classes,
2. the inter-component interface code to link the low-level ink-processing module (B) and the high-level mathematical object manipulation part (A) with (1) and (3),
3. system-specific interface code for embedding within the host application.

This architectural approach is illustrated on Figure 2.

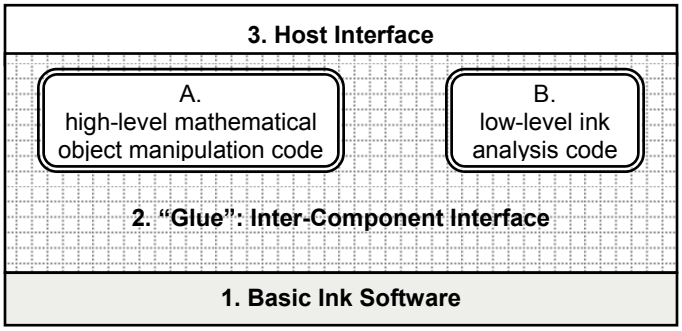


Figure 2. Framework determined by portability criteria

3.3. Implementation Languages

The last question is the choice of language for the bulk of the pen-based mathematical computing application. For the low-level, numerically intensive code the natural choice would be C++. This is used for ink analysis tasks such as character identification. C++, like all the efficient, lower-level programming languages, introduces some portability and component integration problems.

For the high-level code relating to mathematical structures, we have chosen to use Java. This choice ensures code portability and allows the most natural inter-operation with the Maple engine. Another possible choice would have been C#. While C# would make assembly of the low-level components considerably easier on a .NET platform, but it would make interaction with the symbolic math engine more difficult and will also result in portability restrictions.

With the choice of Java for (A) and C++ for (B) and given various possibilities for (2) depending on the platform our architecture ends up in following (Figure 3).

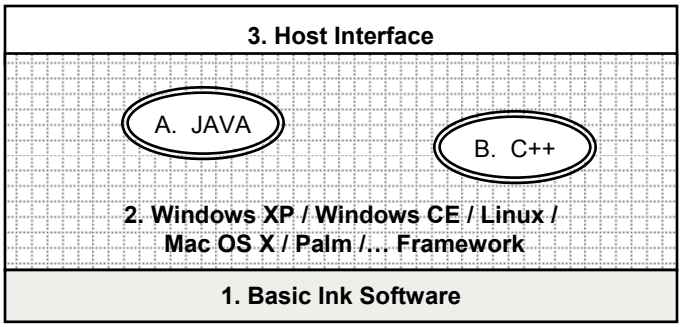


Figure 3. Framework determined by portability criteria and implementation languages

4. Some Implementation Issues

4.1. Instantiating the Architecture for the Tablet PC

Our fist instantiation of the portable architecture is for the Tablet PC. Our choice for (1.), the basic ink software to back the abstract ink classes, is the Tablet PC SDK. This choice gives the best ink handling for the platform [9]. The choice for (2.), the linkage mechanism among the components is the most complex question, which we discuss below. For now we simply note that the .NET architecture is one of the constituents used for linkage. The choice for (3.), the host interface mechanism depends on the application.

To test our framework architecture for the Tablet PC platforms, we have chosen Maple 10 to serve as an example of mathematical computational software and Microsoft Word to illustrate capacities of our solution for document processing application. These framework instantiations are illustrated conceptually in Figure 4.

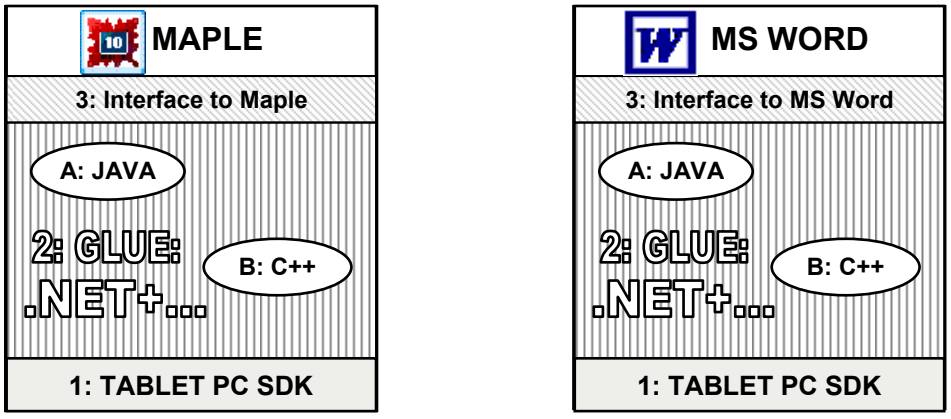


Figure 4. Instantiation of the framework for Maple and MS Word on the Tablet PC platform

After experimenting with different implementations of this test framework we arrived at similar versions of "glue" for each instance. Thus, even though they are implemented separately, they share a majority of their code, as we will show later.

4.2. Linkage for the Test Framework

To understand how the linkage for the framework should be organized, we have to define the core components of the architecture that will directly interact with each other in terms of common data structures, memory management etc. There are only three main components of the framework that need special linkage mechanisms; these are the SDK ink-collector (.NET), ink analyzer (raw C++) and math object manipulator (Java). The rest of the architecture modules can interact with the framework by directly connecting to one of these three main components. Our linkage will therefore consist of three essential tools binding Java, C++ and .NET.

The bi-directional connection between Java and C++ is mainly performed using JNI [6]. Even though this is a relatively mature technology, it still introduces considerable problems when dealing with multiple Java Virtual Machine instances running in one C++ process.

Communication between .NET and raw C++ components is feasible, but also requires some extra care. Programming in .NET languages, such as C#, produces only *managed* code. Managed code is executed by the Microsoft Common Language Runtime. This means that accessing raw (unmanaged) C++ cannot be performed directly. To achieve this, we use two techniques introduced by Microsoft: COM Interoperability [13] and Platform Invocation Services [14].

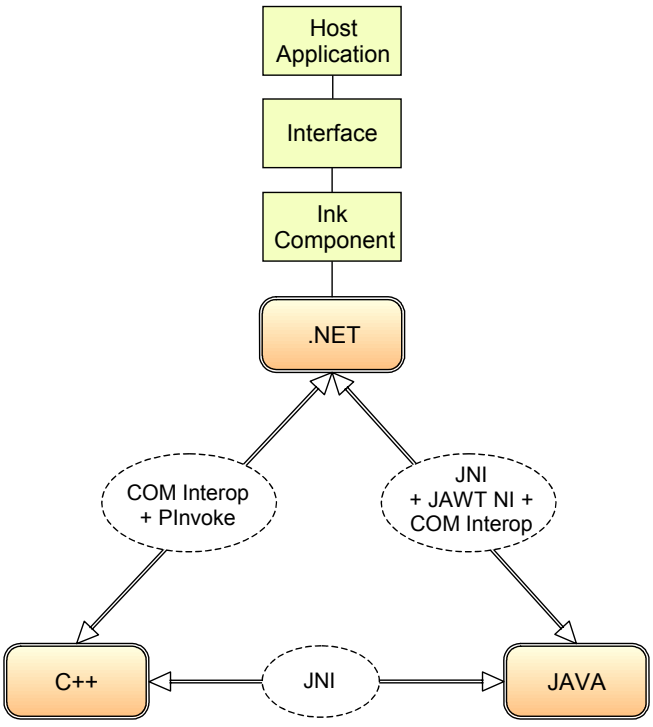


Figure 5. Linkage in the framework instantiated for Tablet PC.

Although interoperability between C++ and Java and between C++ and .NET can be supported by existing standard techniques, there is no straightforward way for Java code to access .NET objects. There are a number of alternatives to provide this linkage, none of them very elegant: One approach is to use certain 3rd party commercial products that provide wrappers for .NET

functionality to make it accessible via Java. This has the disadvantage that as the SDKs evolve it is necessary to use new versions of the 3rd party packages. A non-technical consideration is that these packages also have onerous licensing terms. A second approach is to use one of the 3rd party virtual machines that allow Java to run in a .NET environment. IKVM.NET (www.ikvm.net) is an example of such a solution. This appears to be a viable direction.

We have developed a solution requiring no 3rd party software and, while simple, is sufficient for our purposes. This solution comprises some .NET classes (in this case written in C#) to wrap the Tablet SDK functionality we require. Compiled C# code is passed as a .NET assembly to a special wrapper that is written in C++ and compiled as a COM object [12]. This wrapper is thus configured as unmanaged C++ code. It therefore need not obey all the .NET requirements and can be called from Java programs through Java Native Interface adaptors [6][7]. The main components of framework linkage mechanism are presented on Figure 5.

4.3. Interface to Hosting Applications

It is natural that the details of the host interface depend on the application, at least until suitable standards are developed. In order to allow Maple to access our pen-based framework via an embedded ink component, we provide a Java package that inherits from a general embedded Java Maple interface object (Figure 6a). Connection from this Java library to the Tablet PC SDK is driven by a combination of the Java Native Interface, the Java AWT Native Interface[7] (JAWT NI) and COM Interop [13] that enables hosting .NET components on a Java canvas.

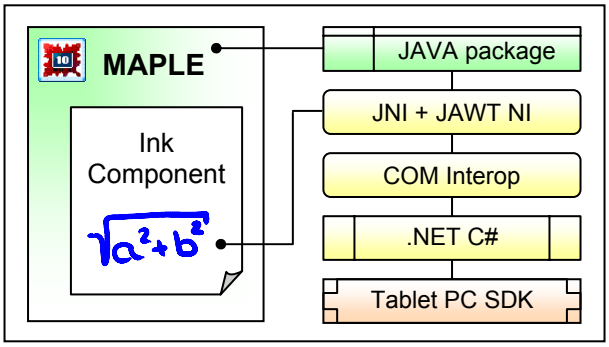


Figure 6a. Interface to Maple: Java library, accessing Tablet PC SDK ink objects

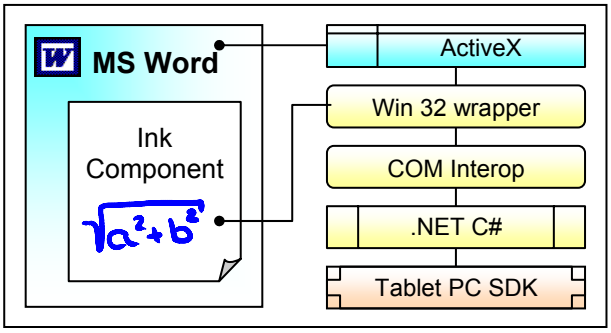


Figure 6b. Interface to MS Word: ActiveX control, accessing Tablet PC SDK ink objects

The host interface mechanism for a document processing application, in this case Microsoft Word, wraps the ink component in an ActiveX control. This ActiveX control, in turn, hosts a Win32 wrapper window that embeds a .NET ink collector overlay, accessible through COM Intreop [13] and PInvoke mechanisms [14] from managed and unmanaged C++ code (Figure 6b).

4.4. Framework Instance

After putting all of the essential components together, the whole framework implementation on the Tablet PC platform is instantiated for Maple and MS Word applications as shown in Figure 7.

The key framework components are labeled as follows:

(1) basic ink software,
(2) replaceable "glue" (includes a, b, c),
(3) interface to the host application.

The main "glue" ingredients are

(a) interface to Java (high-level math objects tool (A)),
(b) interface to C++ (low-level ink tool (B)),
(c) interface to .NET (basic ink-software level).

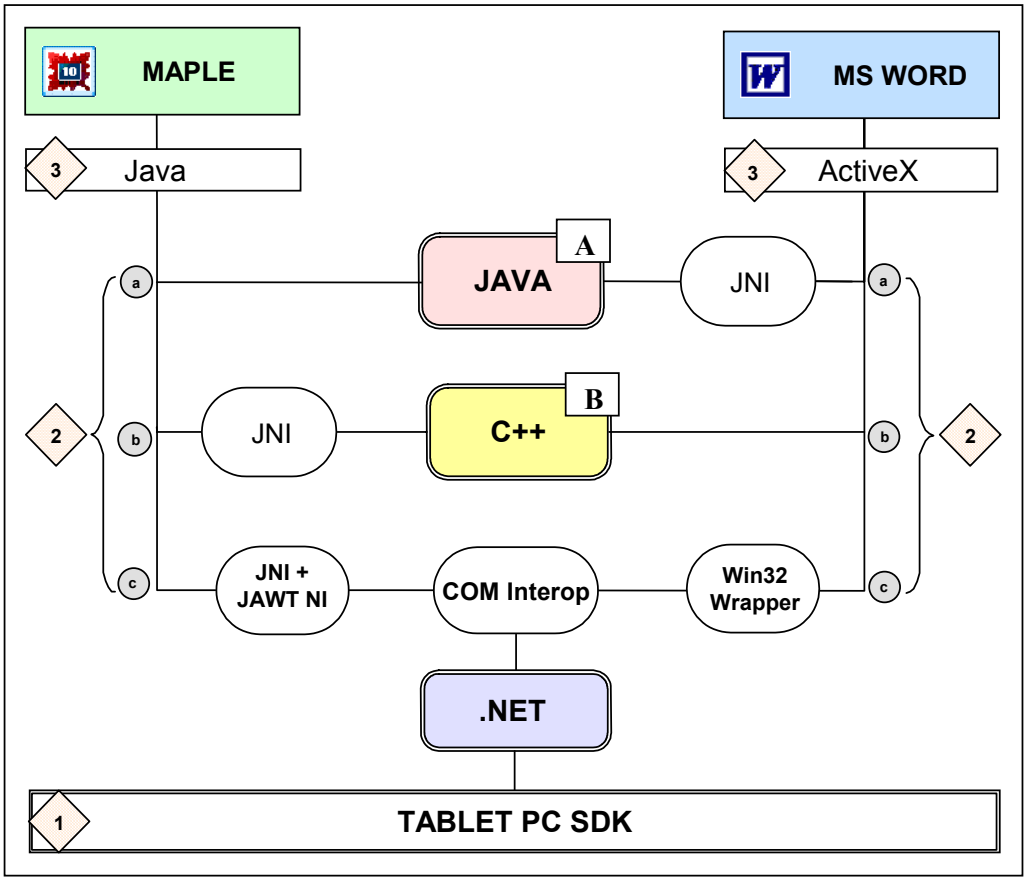


Figure 7. Implementation of the framework instantiated for Maple and MS Word applications on the Tablet PC platform

5. Status and Future Work

5.1. Current results

Our initial experiments show this direction to be a promising one. We have been able to provide versions of all the linkage code for our framework configuration, using a .NET control, owning the screen area inside a Java window, in Maple, or an ActiveX control, in Microsoft Office. This control can collect and manipulate high-resolution ink, and interact with the host application. These experiments are illustrated in Figures 8a and 8b.

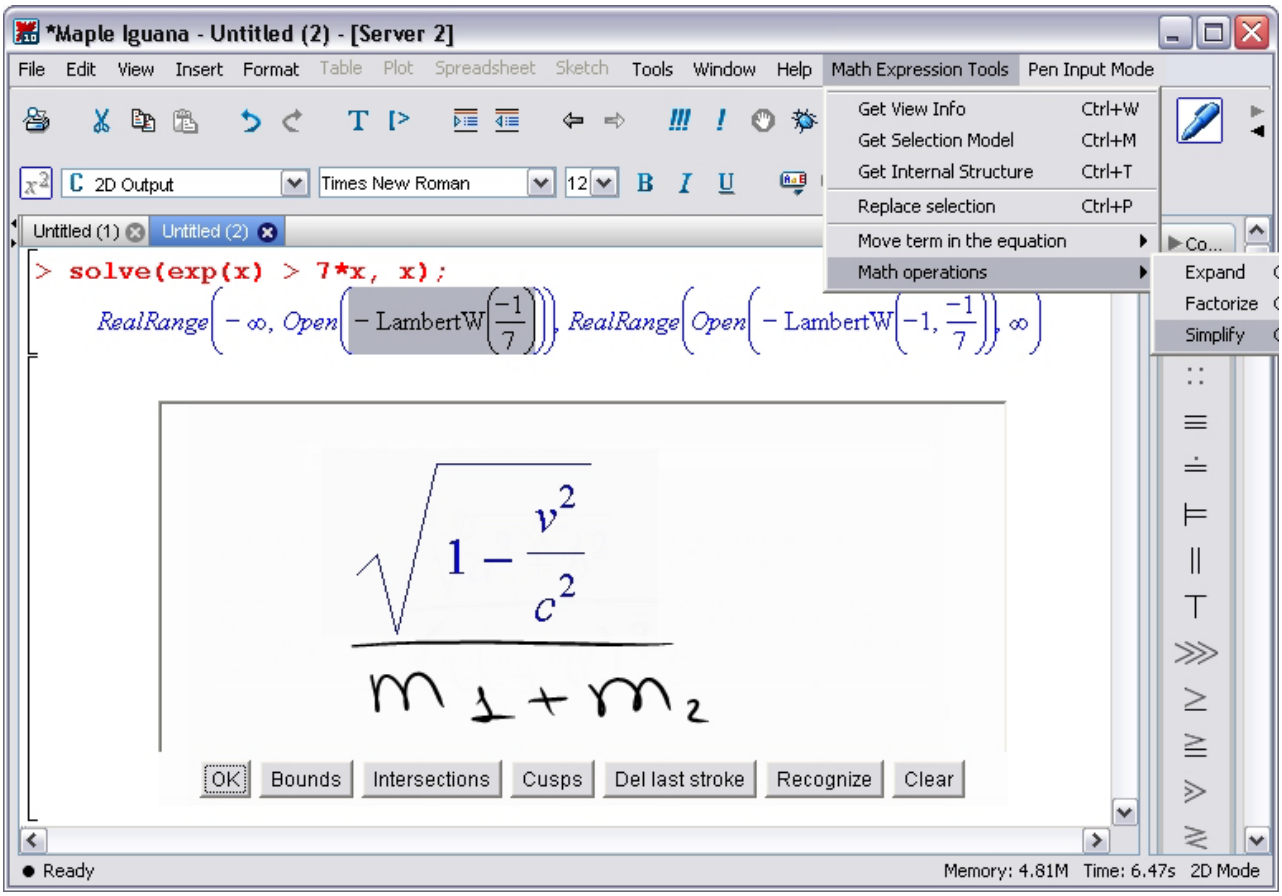


Figure 8a. A pen-based math component embedded in Maple

5.2. Putting all together

As next step, we seek to instantiate our approach for other platforms. In particular, we are interested in suitable modifications for our interconnection architecture for handheld devices and other operating systems.

Our PenMath group at the Ontario Research Centre for Computer Algebra is also working on enabling math-oriented pen-based features within this framework. These research directions include expression reorganization, direct expression manipulation, feature identification for

character classification in large symbol sets, analysis and exploitation of mathematical expression databases, and mathematical handwriting recognition [3][4][5]. With these tools available the whole pen-based architecture may instantiate as shown in Figure 9 without being bound to any specific ink-enabling platform or application.

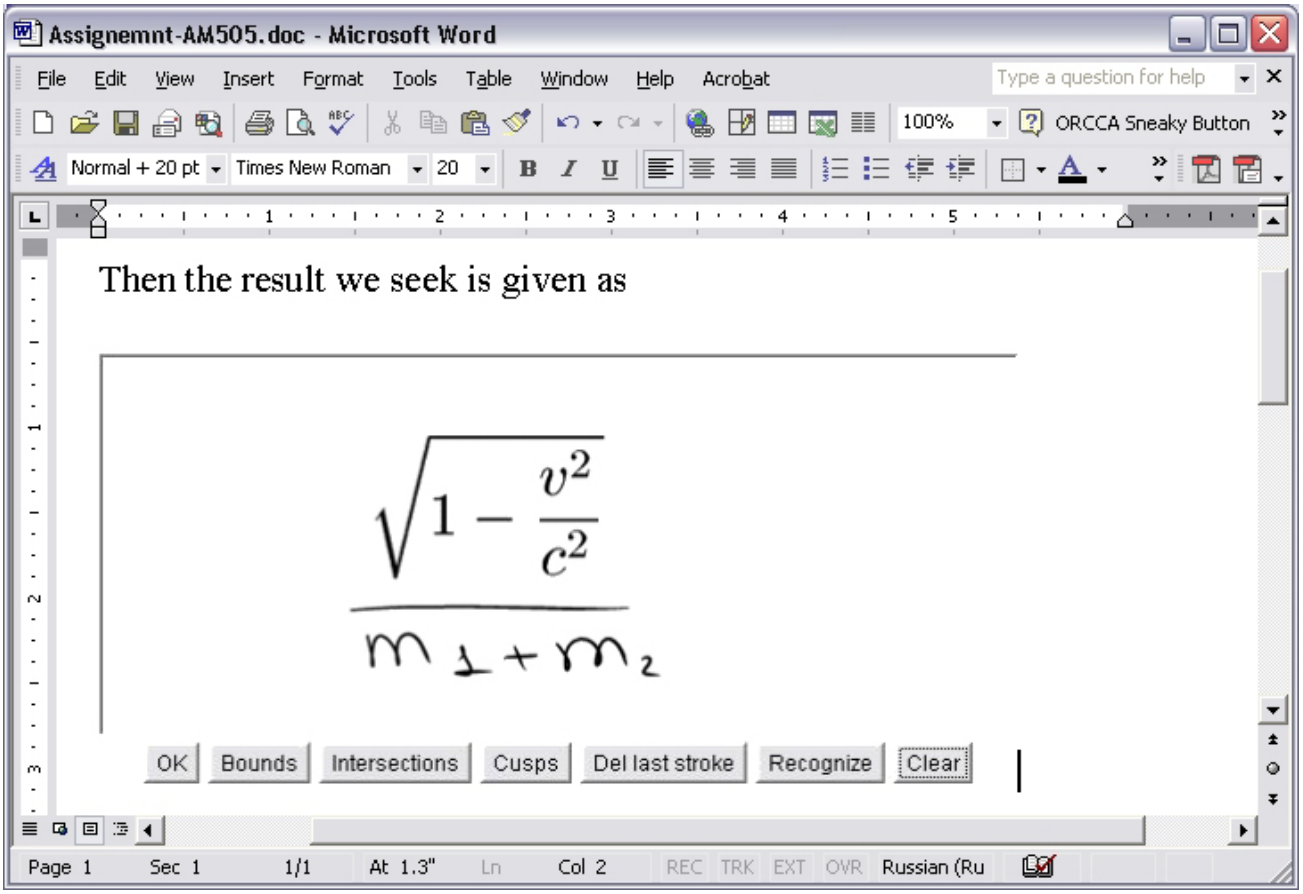


Figure 8b. A pen-based math component embedded in Microsoft Word

6. Conclusions

We have sought to design a framework that will allow wide-spread deployment of a pen-based mathematics interface suitable for both document processing and mathematical computing. We have required that the framework provide high-quality ink handling and easy access to a mathematical computing engine, in its first incarnation, while providing a path for future portability. We believe we have achieved these objectives, and that our solution can allow a more natural interface for mathematics in a variety of settings.

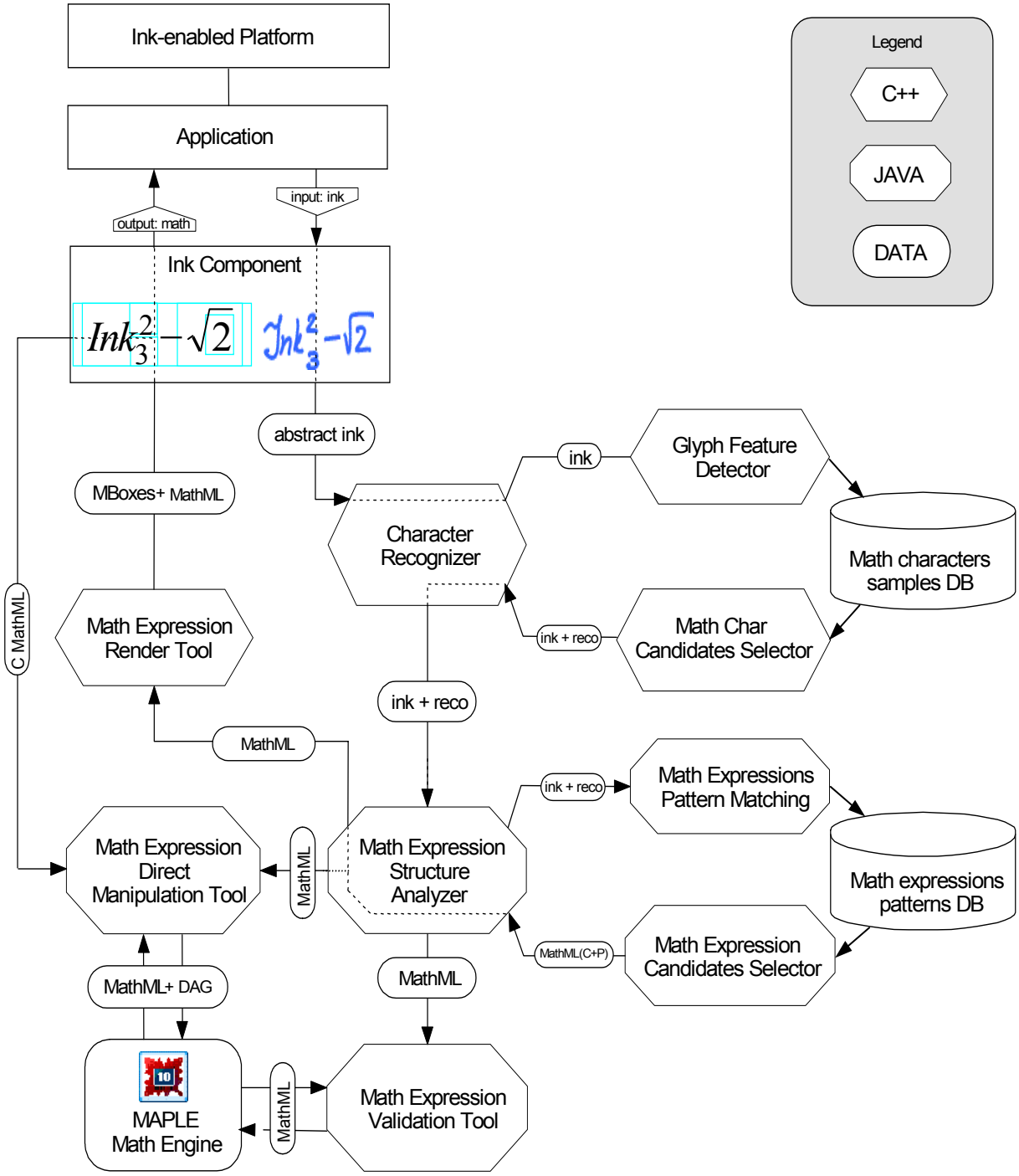


Figure 9. A complete framework for pen-based mathematical computing

References

[1] *Achieving Interoperability of Pen Computing among Heterogeneous Devices and Digital Ink Formats,* Xiaojie Wu, Masters Thesis, Department of Computer Science, University of Western Ontario 2004.

[2] *Supporting Mathematical Handwriting Recognition through an Extensible Digital Ink Framework,* Kevin Durdle, Masters Thesis, Department of Computer Science, University of Western Ontario 2004.

[3] *Implicit Grouping in Mathematical Expression Recognition,* Clare So, Bachelor's Thesis, Department of Computer Science, University of Western Ontario 2003.

[4] *Interactive Mathematical Handwriting Recognition for the Pocket PC,* Bo Wan and Stephen Watt, Proc. International Conference on MathML and Math on the Web (MathML 2002), June 28-30 2002, Chicago USA.

[5] *Handwriting Recognition for Mathematics Using a CrossPad Device,* Arthur Louie, Bachelor's Thesis, Department of Applied Mathematics, University of Western Ontario 2000.

[6] *Java Native Interface*, http://java.sun.com/j2se/1.4.2/docs/guide/jni/, 1999-2001.

[7] *The AWT Native Interface*, http://java.sun.com/j2se/1.4.2/docs/guide/awt/1.3/AWT_Native_Interface.html, 1999.

[8] *Maple*, Maplesoft, http://www.maplesoft.on.ca/

[9] *Building Tablet PC Applications*, Rob Jarrett, Philip Su, Microsoft Press, 2003.

[10] *Microsoft NET Framework*, http://msdn.microsoft.com/netframework.

[11] *Windows Longhorn Network,* http://www.windowslonghorn.net/

[12] *COM: Component Object Model Technologies*, http://www.microsoft.com/com/default.mspx.

[13] *COM Interoperability,* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vcoriCOMInteropTutorial.asp

[14] *Platform Invocation Services,*http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vcwlkplatforminvoketutorial.asp .

[15] How To Build an Office COM Add-in by Using Visual C# .NET, Microsoft Support Center http://support.microsoft.com/?kbid=302901, 2004.

[16] Understanding the Word Object Model from a .NET Developer's Perspective, Mary Chipman, MCW Technologies, LLC, 2003.

[17] ActiveX control containers that support .NET controls, Microsoft Support Center, http://support.microsoft.com/?id=311334