2009

Portable Implementation of Digital Ink: Collaboration and Calligraphy

Rui Hu
Western University

Portable Implementation of Digital Ink: Collaboration and Calligraphy

(Spine Title: Portable Implementation of Digital Ink)

(Thesis format: Monograph)

by

Rui Hu

Graduate Program
in
Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

THE UNIVERSITY OF WESTERN ONTARIO
THE SCHOOL OF GRADUATE AND POSTDOCTORAL STUDIES

CERTIFICATE OF EXAMINATION

Supervisor:	Examination committee:
Dr. Stephen M. Watt	Dr. Yuri Boykov
	Dr. Jacquelyn Burkell
	Dr. Mahmoud El-Sakka

The thesis by

Rui Hu

entitled:

Portable Implementation of Digital Ink: Collaboration and Calligraphy

is accepted in partial fulfillment of the
requirements for the degree of
Master of Science

Date	Chair of the Thesis Examination Board

Abstract

With the widespread availability of Tablet PCs and hand-held pen-based devices, digital ink applications are becoming increasingly popular across a variety of domains. These applications typically use Application Program Interfaces (APIs) and proprietary ink formats that are restricted to single platforms and consequently lack portability. In this thesis we explore the dimension of portability of both digital ink and digital ink handling programs. We examine how various APIs and data formats may be used to provide both low-level and high-level support for platform independence. We present a framework that collects digital ink on different operating systems and provides a platform-independent, consistent interface for digital ink applications. For data portability, we choose InkML to be the data representation as it provides platform-independent support for both data transmission and higher-level semantic representation for digital ink. For program portability we have developed a Java framwork that isolates applications from vendor APIs.

To test our ideas, we have developed two concrete problems. We present InkChat, a whiteboard application, which allows conducting and archiving portable collaborative sessions that involve synchronized voice and digital ink on a shared canvas. We also present Calligraphic Board along with two virtual brush models. The Calligraphic Board collects digital ink from a variety of platforms and renders it with calligraphic properties. Both the InkChat and Calligraphy Board implementations use our Java framework and use InkML as the medium to represent the digital ink, both for rendering it in real time and archiving it for later reference.

Keywords: Digital ink portability, Collaboration, Calligraphy, Pen-based computing

Acknowledgements

From the deepest of my heart, I am grateful to my supervisor, Dr. Stephen M. Watt, for the care to my life which led me through the early cultural struggling in Canada. I thank him also for his inspiration with new ideas which made me feel love of my work.

Sincere thanks to my colleagues in the Ontario Research Centre for Computer Algebra, they are all my friends, offering me advice and assistance which helped me a lot in the research, especially when I was depressed from failure.

Last but not least, I wish to thank my beloved parents, without their unyeilding support and endless encouragement, it would not have been possible to write this thesis. Thank you for the faith in me.

Contents

Certificate of Examination ii

Abstract iii

Acknowledgements iv

Table of Contents v

List of Figures ix

List of Listings xi

1 Introduction 1

1.1 Objectives 2

1.2 Related work 4

1.2.1 Inkscape 4

1.2.2 Inkwell 4

1.2.3 Windows Journal and OneNote 4

1.2.4 QuickSet 5

1.2.5 InkBoard 5

1.2.6 LiveBoard 5

1.2.7 Tivoli 6

1.2.8 Classroom 2000 6

1.2.9 Electronic Chalkboard 6

1.2.10 InkTracer 7

1.2.11 InkAndAudio Chat 7

1.3 W3C Multimodal Interaction Working Group 7

1.4 Data Representation 8

1.4.1 Digital Ink Standards 8

1.4.2 Multimodal Data Standards 10
1.5 InkML and Components 11
1.6 Thesis Organization 16
1.7 Acknowledgement 18

2 Portable Digital Ink Design Issues **19**
2.1 Digital Ink Devices 19
2.1.1 Wacom Tablet and Drivers 19
2.1.2 UC-Logic Tablet and Drivers 20
2.2 Digital Ink Platform APIs 20
2.2.1 Wintab for Windows 20
2.2.2 Windows XP Tablet PC SDK 21
2.2.3 Windows Presentation Foundation 22
2.2.4 XInput for Linux 22
2.2.5 The Linux Wacom Project 23
2.2.6 Cocoa for Mac OS 24
2.2.7 Palm OS 24
2.2.8 iPhone OS 25
2.2.9 Windows Mobile 26
2.3 Previous Works on Portability 27
2.4 A Cross-Platform Digital Ink Framework 28
2.4.1 Java Native Interface 28
2.4.2 Java ME CDC for Mobile Devices 32
2.4.3 A Cross-Platform Framework 32
2.5 Summary 33

3 Multimodal Collaboration Design Issues **35**
3.1 Multimodal Input 36
3.1.1 Voice Input 36
3.1.2 Pen Input 36
3.1.3 Voice and Pen Multimodal Input 37
3.2 Multimodal Collaboration 37
3.2.1 Voice Collaboration 37
3.2.2 Digital Ink Collaboration 38
3.3 Communication 41
3.3.1 Network Architecture 41
3.3.2 Skype 42

3.4 Summary 44

4 InkChat Architecture **45**
4.1 System Architecture 45
4.2 Ink Session Streaming and Archival 47
4.3 Ink Session Retrieval 48
4.4 Canvas Layers 49
4.5 Page Model 50
4.6 Conference Mode 51
4.7 Summary 52

5 Ink Rendering **53**
5.1 InkML Representation 54
5.2 Calligraphic Rendering 54
5.2.1 Round Brush 55
5.2.2 Tear Drop Brush 56
5.3 Summary 59

6 InkChat Implementation **61**
6.1 InkChat Components 61
6.2 Data Transmission 62
6.3 Ink Capture 62
6.4 Ink Erasing 63
6.4.1 Stroke-Wise Erasing 63
6.4.2 Point-Wise Erasing 63
6.5 Hit Testing 63
6.6 Ink Storage 65
6.7 Ink Rendering Objects 65
6.8 Ink Metadata 66
6.9 Summary 67

Conclusion **68**

A InkChat User Manual **73**
A.1 InkChat Features 73
A.2 Platform Availability 74
A.3 System Requirements 74
A.4 Using InkChat 74

Curriculum Vitae 77

List of Figures

2.1 Linux input subsystem and XInput handler 23
2.2 Java Native Interface 28
2.3 A Cross-Platform Framework for Digital Ink Applications 33

3.1 Digital ink streams 39
3.2 Client-Server network architecture 42
3.3 Peer-to-Peer network architecture 43
3.4 Skype network architecture 43

4.1 InkChat Architecture 46
4.2 Sending an InkML stream 48
4.3 Receiving an InkML stream 48
4.4 Loading an ink session 48
4.5 InkChat canvas layers 49
4.6 InkChat page model 50
4.7 The host initiates a stroke in conference meetings 51
4.8 The client initiates a stroke in conference meetings 52

5.1 A simple stroke of Chinese calligraphy. 55
5.2 The model of Round Brush 56
5.3 Filling the gap between two successive Round Brush ink points 56
5.4 The model of Tear Drop Brush 57
5.5 Tear Drop Brush: The case that the tail end remains. 58
5.6 Tear Drop Brush: The case that the tail end moves. 58
5.7 Example of a Chinese calligraphy using Tear Drop Brush. 59
5.8 The plot of Tear Drop Brush parameters 60

6.1 The system architecture of InkChat 62
6.2 The case that the pen tip hits the trace 64

6.3 The case that the pen tip does not hit the trace 64
6.4 Trace class diagram 66

List of Listings

1.1 Example of an EMMA file 10
1.2 Example of a SMIL file 11
1.3 Example of the <trace> and <traceFormat> elements 12
1.4 Example of the <traceGroup> element 13
1.5 Example of an InkML file 14
1.6 Example of the <definitions> element 15
1.7 Example of the <inkSource> element 16
1.8 Example of the <annotation> element 17
2.1 InkProvider.java 29
2.2 InkProvider.h 30
2.3 InkProvider.c 31
3.1 Brush changes 40
3.2 Trace format changes 40
3.3 Brush reference 41

Chapter 1

Introduction

Our primary objective of this thesis is to explore the dimension of digital ink portability. We place our emphasis on cross-platform viability and portable digital ink representation. We believe such capabilities would be extremely useful in developing multimodal collaboration applications.

Digital ink technologies have experienced a tremendous change over the past ten years. It has now been widely used by a variety of electronic devices including Tablet PCs, PDAs, digital pens, touch sensitive whiteboards, cameras and even cell phones. Such devices accept pen-based written input and pass it on to recognition software applications that can convert it into appropriate computer actions. Alternatively, they can organize it into documents or messages that can be stored for later reference or exchange with other collaboration participants.

Tied to digital ink is the notion of using a pen as means of input. Pen input is useful as it is complementary to other input devices. Compared to a keyboard, pen input is more expressive, it allows two dimensional handwriting and drawings that are hard to be generated by a keyboard. Compared to mouse, pen input can be natural as everyone learns to write in school and digital pens give higher maxim resolution. With the widespread availability of pen input devices, it has become popular to use pen input in classroom presentations, conference meetings or real-time distributed conversations.

1.1 Objectives

A number of digital ink applications have been developed in the past years following the emergence of digital ink. Yet most of these applications lack support for portability and are hence restricted to a single platform. Even though cross-platform portability is available, few of them provide many useful ink manipulations on the user interface. We examined the issues of these applications and found the underlying problems to be:

- **The use of platform-specific APIs**
  Applications such as Windows Journal retrieves digital ink by invoking the Windows XP Tablet PC APIs which are restricted to Windows platforms. Such applications can only work on Windows operating systems and are impossible to import to others.

- **The use of proprietary digital ink formats**
  The use of proprietary digital ink formats makes host applications lack portability and limits the ability to interoperate with other applications. For effective data representation and ease of interchange, portable applications should use an ink format standard that is not only neat and elegant but also platform-independent.

- **The complexity of portable capture of digital ink**
  Although pen input devices are able to generate digital ink on almost every individual platform, there has not been a common approach to capture digital ink across all these platforms. To implement a portable digital ink application, developers have to accommodate the detail of each platform. Obviously, this is costly in effort and will eventually affect the design of the user interface.

One of the possible solutions is to construct the application for each platform based on the platform-specific APIs and to interchange the data in a standard format. This, however, will raise other issues. First of all, it increases the time to develop this application, as it requires developers to understand the APIs for each platform, which is very costly in time. Secondly, the application is hard to maintain as it requires maintainers to have a multi-platform background. Last but not least, whenever a new platform comes out, the application has to be reconstructed, from the bottom to the top, based on the new platform's APIs.

Having recognized all these issues, the primary objectives of our work are to:

- explore the dimension of digital ink portability
- examine the question of how program code for digital ink applications can be made portable
- examine the question of which format is most suitable for digital ink representation
- test the suitability of our model of portability using suitably complex cases.

In particular, we wish to have digital ink application code be portable across multiple platforms. Today's platforms capture digital ink in various data structures and report it using different mechanisms. For example, on Windows platforms, digital ink is captured in Wintab packets or `Stroke` objects and reported by the Wintab interface or the Windows XP Tablet PC APIs. The same thing can be done using the `XInput` events and the Linux Input Subsystem on Linux platforms, or the `NSEvent` objects and the Cocoa Framework on Mac OS X. We describe all of these in some detail later. Obviously, portability among such a range of mechanisms requires a well-structured framework that can handle the details of each platform and model digital ink input in a platform-independent way. Applications that build on top of this framework could then collect digital ink on these platforms without knowing the underlying details. Chapter 2 examines the design issues of this framework.

At the same time, we also wish to have digital ink data be platform-independent. We explain why InkML is most suitable and how it works. Chapter 3 addresses this question of digital ink format.

In summary, we wish to understand what is required to have digital ink applications that we "write once and run anywhere".

1.2 Related work

1.2.1 Inkscape

Inkscape [2] is an open source vector graphics editor application. It can accept digital ink from a pen and save it using Scalable Vector Graphics (SVG) [16] format. SVG is the specifications of an XML-based file format for describing two dimensional vector images. Compared to bitmap images, SVG images have many advantages such as smaller file size, resolution independence and easy of resizing. But as a side-effect, SVG format limits the ability to modify the contents of images, especially for digital ink objects. For instance, when an eraser is used to erase a small part of an ink stroke. The remaining stroke may be in a curved shape that is hard to implement in an SVG format image. Also, an SVG image is not suitable for ink recognition which may require analysis of each ink point.

1.2.2 Inkwell

Inkwell, also known as Ink, refers to the handwriting recognition software developed by Apple Inc. It is restricted to Mac OS X. Inkwell allows one to write text in handwriting and then convert to typed text. Currently, it supports English, French and German writing. Inkwell can also be used to draw a sketch and insert it to any place that can display a picture. Inkwell uses platform-dependent APIs and thus cannot be exported to other operating systems.

1.2.3 Windows Journal and OneNote

Windows Journal is a note taking application developed by Microsoft. It has been integrated in Windows XP, Windows Vista and Windows 7. The user interface allows users to switch among pens, highlighters and erasers, move items around the page, insert original calendar information for meetings and save ink to files. As Windows Journal uses the Windows XP Tablet PC SDK [23], it is restricted to Windows platforms and thus cannot be exported to other operating systems. Windows OneNote is another windows application, which is similar to Windows Journal, but more elaborate and suffering the same portability limitations.

1.2.4 QuickSet

QuickSet [15] was a multimodal framework used by the US Navy and US Marine Corps to set up training scenarios and to control virtual environments. It accepts voice and pen input, communicating via a wireless LAN through an agent architecture to a number of systems. The system could recognize voice input with certain responses. If the voice interaction was not feasible, it could still analyze digital ink and then give several possible interpretations, such as platoon, mortar, fortified line, *etc.* This demonstrated that a multimodal interaction enabled a better and more efficient communication.

1.2.5 InkBoard

InkBoard was a whiteboard system which allowed for graphical collaboration and design, including network-shared ink strokes and audio/video conferencing capabilities [30]. It was developed by the MIT Intelligent Engineering System Lab and published in 2004. It integrated the Microsoft Conference XP research technology and thus limited itself to a Windows environment.

1.2.6 LiveBoard

LiveBoard [14] was developed by the Xerox Palo Alto Research Center in 1990. It aimed to provide a basis for slide presentations and group meetings. It was basically a large display system with interactive ability to communicate with a multi-state cordless pen. LiveBoard made use of an X11-based bitmap paint application to render the digital ink generated by the pen, and then projected the image to the large display. LiveBoard had some drawbacks. It could remember the drawing on the pages and recover them when necessary. But this only refers to the final version. Moreover, it was next to impossible to conduct collaborative operations at a stroke level.

1.2.7 Tivoli

Tivoli [31] was an extension of LiveBoard. It overcame the limitations of LiveBoard by making use of stroke objects, as opposed to pixel map images. These stroke objects were represented and manipulated by a proprietary Software Development Kit(SDK). In seeking to improve the collaboration capability, Tivoli can enable multiple cordless pens for multiple users.

1.2.8 Classroom 2000

The Classroom 2000 project [9] was developed at Georgia Tech. It was published by Gregory D. Abowd *et al.* [9] in 1998. Its primary purpose was to create an environment to capture as much of activity as possible from the classroom experience. It included tools to automate the production of lecture notes and to assist students in reliving the lecture. The recorded materials could be accessed via Web browsers. This application did not support real-time distributed collaboration, as the lecture materials had to be uploaded to a server before it could be accessed.

1.2.9 Electronic Chalkboard

In addition to the advantages of traditional blackboards, Electronic Chalkboard [17] integrates itself with distance education tool. Thus makes it possible to load images and interactive programs directly from a hard-drive or the Internet. It is also able to interact with computer algebra systems and to display computation results. Remote lectures or meetings are applicable using this system with a minimum requirement of a Java-enabled browser. Electronic Chalkboard allows geographically separated participants to see the objects as exactly same as they appear on the blackboard, and to hear the voice in a manner similar to local participants do. The lectures showing on the blackboard can be automatically recorded during the transmission and be viewed on the Web in real time. The architecture of the Electronic Chalkboard is quite simple. The server is running on a local machine which is connected to some pen input devices. And it is responsible for transmitting audio, video and images over time. On the remote machines, three applets are in charge of receiving streaming data and displaying on screen. However, Electronic Chalkboard has limitations. As the content on blackboard is saved as images, it is impossible to conduct stroke-wise

operations on client side and consequently makes the communication as an one way street.

1.2.10 InkTracer

InkTracer was developed by the Ontario Research Centre for Computer Algebra (ORCCA) [5]. It was a Java application designed for ink animation. It can parse an InkML file, retrieve stroke information, and then render it on a canvas. InkTracer can also convert ink strokes to an animated GIF image which records each phase of the rendering, from the beginning to the end.

1.2.11 InkAndAudio Chat

InkAndAudio Chat [32] was developed by the Ontario Research Centre for Computer Algebra (ORCCA) [5] in 2008. It was a collaborative whiteboard application that allowed conducting and archiving communication sessions involving audio and ink on a shared canvas. It uses WAV and InkML as the media to represent the data, and Skype, an application that handles voice calls over the Internet, as the backbone for data transmission. InkAndAudio Chat was a good attempt to use InkML for data representation, but a number of good features of InkML were not implemented by this application. These include brush change, trace format change, context change, context reference, ink metadata and ink annotation. It also lacked support for page navigation which requires InkML document parsing. InkAndAudio Chat is also platform-dependent. It was implemented using C# and the .NET technologies. As a result, it can only run on Windows platforms.

1.3 W3C Multimodal Interaction Working Group

The W3C Multimodal Interaction Working Group [18] was launched in 2002. As a part of the World Wide Web Consortium(W3C), it seeks to extend the Web and allows users to select the most suitable combination of interaction mode for their current needs, including any disabilities. It also aims to help developers provide an effective user interface for whichever modes users select [18]. With the effort of the

W3C Multimodal Interaction Working Group, several new standards such as InkML [11], EMMA [13], *etc.* were developed to meet the needs involved in multimodal interaction.

1.4 Data Representation

1.4.1 Digital Ink Standards

Ink format standards should provide useful features for note taking and sharing, real-time distributed conversations, conference meetings, and classroom presentations. Even more importantly, they should be platform-independent, which allows host applications to work on different operating systems. Digital ink technologies have evolved over years and are becoming more and more popular, as they provide a natural mode other than keyboards and mice for computer input. These technologies have been used by a number of devices, such as Tablet PCs, PDAs and GPS receivers. These devices share a common point of portability which require digital ink to be represented in a flexible way.

Various ink format standards do exist today and some of them have been widely used. We describe these briefly below.

Jot

The Jot [12] standard was established in 1992 as a joint work of Slate, Lotus, GO, Microsoft, Apple, General Magic and other corporations. It was the first attempt to define a format for ink storage and interchange. As an initial specification, its primary purpose was to allow digital ink applications to run on small platforms such as PDAs and old Tablet PCs. These platforms in practice lacked memory space and consequently had limited ability for real time sharing. In addition, Jot provided little support for memory-intensive digital ink manipulations such as resizing strokes, modifying the color of strokes and deleting strokes.

Unipen

The Unipen [19] standard came into existence in 1994. It incorporated features of internal formats of several institutions, including IBM, Apple, Microsoft, Slate, HP, AT&T, NICI, GO and CIC. The primary purpose was for technical and scientific research use. As a result, it contained a lot of metadata to describe the ink captured from a digitizer, which would require large memory space and powerful processors. It was well suited to represent collection of handwriting samples, but not particularly well suited for other applications.

ISF

Ink Serialized Format (ISF) [22] is a Microsoft proprietary ink format that is mainly used on mobile devices such as PDAs, Tablet PCs and Ultra-Mobile PCs. ISF is restricted to Windows platforms and thus is not a good choice for applications made for portability.

SVG

Scalable Vector Graphics (SVG) [16] as an open standard has been underdeveloped by the World Wide Web Consortium(W3C) since 1999. It is an XML-based file format for describing two dimensional vector graphics. SVG has many advantages such smaller file size, resolution independence and ease of resizing. But as a side-effect, SVG format limits the ability to modify the contents of images, especially for digital ink objects. While it can represent curves, it is not well suited to represent the nuance of digital ink strokes.

InkML

Ink Markup Language (InkML) [11] is an open and up-to-date standard which is released by the W3C Multimodal Interaction Working Group [18]. It provides a standard format to represent digital ink generated by a stylus or an electronic pen. It also provides support for a wide range of hardware devices. It is a platform-independent format that can be used by applications on a variety of operating systems. A more detailed discussion can be seen in the section 1.5.

```
<emma:emma version="1.0"
    xmlns:emma="http://www.w3.org/2003/04/emma"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/04/emma
     http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
    xmlns="http://www.example.com/example">
  <emma:one-of id="r1" emma:start="1087995961542"
      emma:end="1087995963542"
      emma:medium="acoustic" emma:mode="voice">
    <emma:interpretation id="int1" emma:confidence="0.75"
    emma:tokens="flights from boston to denver">
      <origin>Boston</origin>
      <destination>Denver</destination>
    </emma:interpretation>
    <emma:interpretation id="int2" emma:confidence="0.68"
    emma:tokens="flights from austin to denver">
      <origin>Austin</origin>
      <destination>Denver</destination>
    </emma:interpretation>
  </emma:one-of>
</emma:emma>
```

Listing 1.1: Example of an EMMA file

1.4.2 Multimodal Data Standards

EMMA

Extensible MultiModal Annotation markup language (EMMA) [13] is developed by the W3C Multimodal Interaction Working Group. It aims to represent semantic interpretations of user input including speech, natural language text and digital ink. EMMA can be used as a standard data interchange format between components of a multimodal system. It defines means for recognizers to annotate application specific data with information such as confidence scores, time stamps, input mode (e.g. key strokes, speech or pen), alternative recognition hypotheses, and partial recognition results.

An example of EMMA file is shown in Listing 1.1, from [13].

In this example, there are two possible interpretations for the speech input. The confidence scores and tokens associated with this input are annotated using the `emma:confidence` and the `emma:tokens` attribute of each `<emma:interpretation>`

```
<smil>
  <body>
    <seq repeatCount="3">
      <img src="tree.jpg" dur="5s"/>
      <img src="flower.jpg" dur="5s"/>
    </seq>
  </body>
</smil>
```

Listing 1.2: Example of a SMIL file

element. Both interpretations have same timestamps and contain application-specific semantics that involves in the elements of <origin> and <destination>.

SMIL

Synchronized Multimedia Integration Language (SMIL) [10] is an XML-based language that allows authors to conduct interactive multimedia presentations. The latest edition is in version 3. It defines markups for timing, layout, animations, visual transitions and media embedding and allows presentations of media items including image, video, audio and text as well as links to other SMIL presentations. An example of SMIL file is shown in Listing 1.2, from [10].

This SMIL file plays each listed image for 5 seconds in the shown sequence with a repeat of 3 times. SMIL is useful. But it does not provide directly support for rendering digital ink. One can only use the <brush> element of the brushMedia module in SMIL, to paint a particular section of a window. W3C is, however, examining how to best use SMIL together with InkML.

1.5 InkML and Components

Ink Markup Language (InkML) [11, 36], introduced briefly in the previous section, is an XML format under development for pen-based applications that can store, manipulate and exchange digital ink. It provides a standard data format for representation of digital ink generated by a stylus or an electronic pen. InkML also provides support for a wide range of hardware devices. With the emergence of InkML as an open standard, a wide variety of applications have been using it for data representation. It

```
<traceFormat>
   <channel name="X" type="integer"/>
   <channel name="Y" type="integer"/>
   <channel name="F" type="integer"/>
</traceFormat>
<trace>
   1125 18432 257, 1128 18441 270, 1134 18452 278,
   1136 18459 275
</trace>
```

Listing 1.3: Example of the <trace> and <traceFormat> elements

is a platform-independent standard that can be used by applications on a variety of operating systems.

After considering various alternatives, we arrived at a design using InkML, not only because it is an open and up-to-date standard which supports portability, but also for it provides application-defined channels to support sophisticated ink representation.

The <trace> and <traceFormat> Element

The fundamental data element in an InkML document is the <trace> element. It is used to record the data captured by digitizers. The coordinates sequence inside a <trace> element represents a collection of contiguous ink points, which are encoded in terms of the specification given by the <traceFormat> element. In particular, the <traceFormat> element allows applications to define their own channels to support sophisticated ink representations. Each application-defined channel provides a coordinate value for each ink point. For example the ink point (x, y, f) can specify a three-channel trace format, applications may arbitrarily define the x and y to indicate the position of the ink point, and the f to represent the pen tip force.

Listing 1.3 shows an exmaple of the <trace> and the <traceFormat> elements.

The <traceGroup> Element

The <traceGroup> element is used to represent a collection of successive traces which share common characteristics, such as the same brush type, color, trace format, *etc.*

Listing 1.4 shows a collection of traces which share the same brush characteristics.

```
<definitions>
   <brush id="TearDropBrush">
        <color>blue<color>
        <width>4</width>
        <length>9</length>
   </brush>
</definitions>
<traceGroup brushRef="#TearDropBrush">
   <trace>35 41 27, 36 44 28, 39 45 30, 41 44 28</trace>
   <trace>150 141 78, 149 141 75, 146 143 70</trace>
</traceGroup>
```

Listing 1.4: Example of the <traceGroup> element

The <ink> Element

All InkML documents are well-formed XML documents that comply to the syntax rules. The <ink> element is the root element of any InkML document. All the content of an InkML document is contained within a single <ink> element. When combining InkML and other XML elements within applications, qualifiers may be used to disambiguate elements from different namespaces. Various allowed sub-elements are enclosed within the <ink> element. These allowed sub-elements include: <definitions>, <context>, <trace>, <traceGroup>, <traceView>, <annotation> and <annotationXML>. These can occur any number of times, in any order.

An example of an InkML file is shown in Listing 1.5, which represents five ink traces, from [11].

```
<ink>
   <trace>
      10 0, 9 14, 8 28, 7 42, 6 56, 6 70, 8 84, 8 98, 8 112,
      9 126, 10 140, 13 154, 14 168, 17 182, 18 188, 23 174,
      30 160, 38 147, 49 135, 58 124, 72 121, 77 135, 80 149,
      82 163, 84 177, 87 191, 93 205
   </trace>
   <trace>
      130 155, 144 159, 158 160, 170 154, 179 143, 179 129,
      166 125, 152 128, 140 136, 131 149, 126 163, 124 177,
      128 190, 137 200, 150 208, 163 210, 178 208, 192 201,
      205 192, 214 180
   </trace>
   <trace>
      227 50, 226 64, 225 78, 227 92, 228 106, 228 120, 229
      134, 230 148, 234 162, 235 176, 238 190, 241 204
   </trace>
   <trace>
      282 45, 281 59, 284 73, 285 87, 287 101, 288 115, 290
      129, 291 143, 294 157, 294 171, 294 185, 296 199, 300
      213
   </trace>
   <trace>
      366 130, 359 143, 354 157, 349 171, 352 185, 359 197,
      371 204, 385 205, 398 202, 408 191, 413 177, 413 163,
      405 150, 392 143, 378 141, 365 150
   </trace>
</ink>
```

Listing 1.5: Example of an InkML file

The <definitions> Element

The <definitions> element is a container which is used to make content reusable. The allowed sub-elements within the <definitions> elements include: <brush>, <canvas>, <canvasTransform>, <context>, <inkSource>, <mapping>, <timestamp>, <trace>, <traceFormat>, <traceGroup> and <traceView>. These elements can be referenced from other elements by using the attributes of brushRef, canvasRef, canvasTransformRef, contextRef,, inkSourceRef,, mappingRef, timestampRef and traceFormatRef.

Listing 1.6 shows an exmaple of the <definitions> element, from [11].

```
<ink>
   <definitions>
      <brush xml:id="redPen"/>
      <brush xml:id="bluePen"/>
      <traceFormat xml:id="normal"/>
      <traceFormat xml:id="noForce"/>
      <context xml:id="context1"
           brushRef="#redPen"
           traceFormatRef="#normal"/>
      <context xml:id="context2"
           contextRef="#context1"
           brushRef="#bluePen"/>
   </definitions>
   <context contextRef="#context2" traceFormatRef="#noForce"/>
   <context xml:id="context3"/>
</ink>
```

Listing 1.6: Example of the <definitions> element

The <inkSource> Element

The ink format and attributes vary from device to device. For applications communicating with different hardware it becomes necessary to record metadata about the ink format and quality. This is accomplished in the <inkSource> element. It records basic information about the hardware devices as well as the ink format. In particular, information such as a digitizer's dimension, sampling rate, latency as well as trace format and channel properties can be represented.

Listing 1.7 shows an exmaple of the <inkSource> element, from [11].

The <annotation> Element

The <annotation> element is useful as it provides a mechanism for inserting simple textual information in InkML documents. For instance, it may be used to give ground truth or recognition results or disambiguate ink traces from eraser traces. Or it may be used to inform remote clients that the local user is doing page navigation operations.

Listing 1.8 shows an exmaple of the <annotation> element, from [11].

```
<inkSource xml:id = "mytablet"
   manufacturer = "Example.com"
   model = "ExampleTab 2000 USB"
   specificationRef="http://www.example.com/products/
      exampletab/2000usb.html">
   <traceFormat href="#XYF"/>
   <sampleRate uniform="True" value="200"/>
   <activeArea size="A6" height="100" width="130" units="mm"/>
   <srcProperty name="weight" value="100" units="g"/>
   <channelProperties>
      <resolution channel="X" value="5000" units="1/in"/>
      <resolution channel="Y" value="5000" units="1/in"/>
      <channelProperty
         channel="Y"
         name="peakRate"
         value="50"
         units="cm/s">
      <resolution channel="F" value="1024" units="dev"/>
   </channelProperties>
</inkSource>
```

Listing 1.7: Example of the <inkSource> element

1.6 Thesis Organization

The primary objective of this thesis is to explore the dimension of digital ink portability. We investigate available APIs on a variety of platforms and examine the question of which format is most suitable to represent digital ink for both streaming and archival purposes. We propose a cross-platform framework that can collect digital ink from a variety platforms and provide a platform-independent, consistent interface to digital ink applications. InkChat and Calligraphy Board are the implementations to demonstrate our ideas.

The rest of this thesis is organized as follows:

In Chapter 2 we explore available APIs on a variety of platforms and propose a cross-platform framework for portable digital ink applications.

In Chapter 3 we investigate the design issues involved in multimodal collaboration and explain how we use InkML to stream digital ink.

In Chapter 4 we present the system architecture of InkChat. We focus on how InkChat sends and receives synchronized voice and digital ink as well as archives

```
<ink xmlns:dc="http://dublincore.org/documents/2001/10/26/dcmi
   -namespace/">
 <annotation type="description">A Sample of Einstein's
    Writings</annotation>
 <annotation type="writer">Albert Einstein</annotation>
 <annotation type="contentCategory">Text/en</annotation>
 <annotation type="language" encoding="ISO639">en</annotation>
 <annotation dc:language="en"/>

 <trace id="trace1">
    ...
 </trace>
 <traceGroup id="tg1">
    <annotation type="truth">Hello World</annotation>
    <traceGroup>
      <annotation type="truth">Hello</annotation>
      <trace> ... </trace>
      ...
    </traceGroup>
 <traceGroup>
    <annotation type="truth">World</annotation>
      <trace> ... </trace>
        ...
      </traceGroup>
    </traceGroup>

 <traceView href="#tg1">
    <annotation type="style">Cursive</annotation>
 </traceview>
</ink>
```

Listing 1.8: Example of the <annotation> element

these data. We also explain how each canvas layer works and how to support page navigation.

In Chapter 5 we present Calligraphy Board along with two virtual brush types. We explain our approach that makes use of InkML to represent sophisticated digital ink and to render it with calligraphic properties.

In Chapter 6 we describe the implementation of InkChat. We also explain how we capture digital ink as well as how we manage the data.

We conclude with a brief discussion and suggest directions for future work.

We also present a user manual in the appendix to help users to use InkChat.

1.7 Acknowledgement

The design of the two ink applications described in this thesis was done jointly with my supervisor, Dr, Stephen Watt. The implementation of InkChat was done in collaboration with Michael Friesen.

Chapter 2

Portable Digital Ink Design Issues

Digital ink technologies have evolved over past years. Its usage in form filling, free-hand input, document annotation and collaboration has been well recognized by more and more users. Today, pen input is almost everywhere, from big whiteboard systems to small PDAs, from portable Tablet PCs to smart iPhones, a number of known digital ink applications have been developed to run on these platforms. However, most of these applications use platform-specific APIs or proprietary ink formats and consequently lack portability.

One of our objectives of this thesis is to explore the dimension of digital ink portability. We focus on examining the availability of APIs on a variety of platforms and try to answer how to make digital ink applications viable across platforms. We believe such capability would be useful especially in developing multimodal collaboration applications.

2.1 Digital Ink Devices

2.1.1 Wacom Tablet and Drivers

Wacom [8] is one of the largest graphics tablet manufacturers in the world. It is well-known for its patented cordless, battery-free, pressure-sensitive and tilt-enabled digital pen. Wacom tablet samples X, Y coordinates, pressure, timestamp and tilt angle (if supported). Samples are collected at a high frequency, and hence this device

has been widely used in the fields of handwriting recognition, graphical design and so on.

In addition, Wacom tablet technology has been integrated by most Tablet PC leading manufacturers, such as Hewlett-Packet, Toshiba, Lenovo, Acer. With an Wacom supported PC, one can simply write on the screen and save the note in handwriting format or convert it to typed text.

Wacom currently provides official support of tablet drivers on two types of platform, Windows (Windows 98SE, Windows Me, Windows 2000, Windows XP, Windows XP x64, Windows Vista and Windows 7) and Mac OS (Mac OS 9 and Mac OS X). Wacom also provides tablet driver support for Linux through an open source 3rd party project, called The Linux Wacom Project [4].

2.1.2 UC-Logic Tablet and Drivers

UC-Logic [7] also offers graphics tablet models. The digital ink is generated by a wireless digital pen and then sent to a computer through a USB cable or wireless. UC-Logic provides official support of tablet drivers on Windows (Windows 98SE, Windows 2000, Windows XP and Windows Vista) and Mac OS (Mac OS X 10.2.6 or later).

2.2 Digital Ink Platform APIs

Pen input has been supported by most platforms. Each platform typically uses discrete APIs to collect digital ink. We will explore these APIs below.

2.2.1 Wintab for Windows

Wintab [29] is the industry standard for accessing graphics tablet features on Windows platforms. It provides a standardized programming interface to digitizing tablets, three dimensional position sensors, and other pointing devices. It has been defined by a group of leading digitizer manufacturers and application developers such as Wacom, Hitachi, GTCO, CalComp, Genius and OCE .

Wintab has been implemented using C++ and works as a low-level interface that directly communicates with tablet drivers. It claims many advantages including:

1. **Reliability**
   Wintab has been widely used for over 10 years for its good design and reliability by a number of well-known software products including Photoshop, Illustrator, *etc.*

2. **Easy Accessibility**
   Wintab's APIs are open to public and are accessible from Java applications through JNI.

2.2.2 Windows XP Tablet PC SDK

Windows XP Tablet PC SDK [23] provides both low-level and high-level support for digital ink. It enables input and output of handwriting data on a Tablet PC as well as interchange of these data with other computers. Windows XP Tablet PC SDK divided the APIs into three categories in terms of their functions:

1. **Tablet input**
   The Tablet input API manages pen-specific features, such as the various side-buttons on a pen, and collects digital ink and gestures. It contains two concrete classes, InkCollector and InkOverlay, to facilitate tablet input in the implementation.

2. **Ink data management**
   The Ink Data Management API handles ink manipulation and ink storage. It provides assistance on ink operations such as changing ink strokes and their properties, connecting and splitting strokes, rendering and erasing strokes. The Ink Data Management API stores digital ink in two types of object, stroke and ink. The stroke object is the fundamental block of an ink document which is represented by the ink object. The ink object may contain a collection of strokes and each of them cannot exist without an ink object as its owner.

3. **Ink recognition**
   The Ink Recognition API is concerned with tablet computing. It is targeted to suggest possible textual interpretations for digital ink. The Ink Recognition

API uses language-specific recognition engines that are designed for different countries. As a result, it is hard to evaluate its recognition accuracy in general. But compared to other existing technologies, the Tablet PC Platform PC Platform's English recognizer is fairly accurate when analyzing cursive English.

Windows XP Tablet PC SDK is also supported by the Microsoft .NET Framework.

2.2.3 Windows Presentation Foundation

The Windows Presentation Foundation (WPF) [25] is a graphical subsystem for rendering user interfaces on Windows platforms. That is part of the Microsoft .NET Framework. WPF is designed to provide user interface features such as transparency, gradients and transforms. It provides native support for digital ink. The tablet input, ink management and ink recognition APIs that originally contained in Windows XP Tablet PC SDK have been put together in the <`InkCanvas`> element with all of their functionalities.

2.2.4 XInput for Linux

The Linux platform makes use of a different approach to handle device input. Input devices, such as keyboards, mice, tablets, joysticks and a wide range of other devices that undertake the interaction between users and the command line or a graphical user interface, are protected and managed by the Linux kernel. Being part of the Linux kernel, the input subsystem can access these devices through special hardware interfaces such as serial ports, PS/2 ports, parallel ports and Universal Serial Bus. It provides a range of APIs that expose the device input to users in a device-independent way. The architecture of the Linux input subsystem is shown in Figure 2.1.

There are three components of the Linux input subsystem: drivers, input core and handlers. The drivers interact with low-level input hardware where they collect input events in real time. These events are then converted into standard format before being sent to the input core. In addition to registering and unregistering devices, the input core dispatches the standard events to appropriate handlers. The handlers convert received events into a format required by a particular API and eventually push them to appropriate applications in the user space.

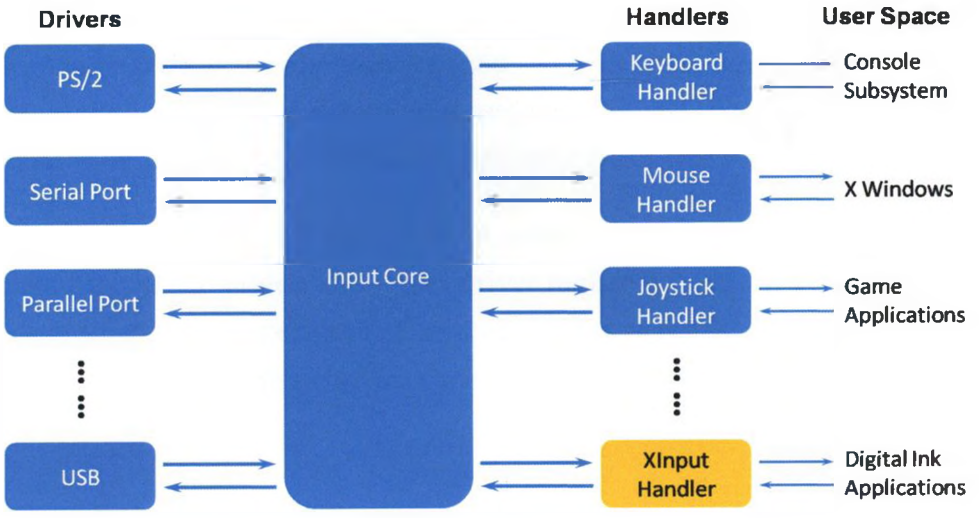


Figure 2.1: Linux input subsystem and XInput handler

The input subsystem includes almost all useful handlers. The keyboard handler interacts with command line, the mouse handler for manipulating a pointing device on X Window System, the joystick handler for game control, *etc.* There is also a special handler called XInput handler which pushes general input to user space applications. This handler is based on the idea of transparency and delivers XInput events directly to particular applications without converting to an API-specific format.

The capture of digital ink is undertaken by XInput handler. Tablet driver interacts with hardware, filters device input and sends it to the Linux input subsystem. The Linux input subsystem converts the input to XInput events and then directly delivers them to user space applications through the XInput handler. The XInput handler is useful as it makes user space applications close to the input source and hence allows more primitive operations on the raw data.

2.2.5 The Linux Wacom Project

The Linux Wacom project [4] is an open source project that manages the drivers, libraries and documentation for configuring and running Wacom tablets on Linux platforms. Current tablets fall into two categories, serial tablets including most Tablet PCs and USB tablets. The Linux Wacom Project provides driver support for both of them.

Serial tablet is connected to a COM port and can be directly accessed by opening an appropriate device, usually `/dev/ttyS0`. The Wacom tablet driver is responsible for finding the appropriate device and reporting input events to the Linux input subsystem. The Linux input subsystem will then push these events to the user space.

Unlike the serial tablet, USB tablet is connected to a Universal Serial Bus port. When user plug in the tablet, the tablet driver registers itself with the USB subsystem and then notifies the Linux input subsystem that it is ready for streaming data. The user applications can access the data events by opening an appropriate device such as `/dev/input/event0`.

2.2.6 Cocoa for Mac OS

Cocoa is one the five major available APIs dedicated to Mac OS X. Most Cocoa APIs are implemented in Objective-C, an object-oriented extension to C used extensively by Apple. It uses a dynamic runtime to execute application events [20].

In Mac OS X, there are two types of tablet event: `proximity` event and `pointer` event. The `proximity` events are generated whenever a pen is placed near tablet or removed from tablet. They contain information about which function is being used (the stylus or the eraser) and its capabilities. The `pointer` events may follow proximity events and contain current state of the pen including position, pressure, timestamp and tilt angle.

In Cocoa, tablet events are packed in `NSEvent` [20], a class that contains information about an input action. These events arrive via the event queue and will then be dispatched by the `NSApplication` class which manages the main event loop of an application. Each is sent to either a `proximity` event handler or a `pointer` event handler depending on the event type. Both handler process received events and make appropriate actions.

2.2.7 Palm OS

Palm OS is an embedded operating system initally developed by Palm Inc. It is normally used by Personal Digital Assistants (PDAs) which often come along with a pen. Devices using Palm OS have native support for pen input. Digital ink generated

by a pen will be converted to Palm OS pen input events. These events are categorized into three primitive types: `penDownEvent`, `penUpEvent` and `penMoveEvent`. The `penDownEvent` is fired by Palm OS event manager the first time the pen touches the digitizer. It contains window-relative position of the pen in pixels, i.e. number of pixels from left and top of the window boundary, and the number of taps received at this position. The `penMoveEvent` may follow the `penDownEvent` and passes pen movement information, including window-relative position of the pen in pixels, to its listeners. If the pen is lifted from the digitizer, the `penUpEvent` will be fired by Palm OS event manager. In addition to recording tap numbers, this event also contains two types of position information: display-relative position and window-relative position. The display-relative position describes the start point and the end point of current stroke. And the window-relative position represents the position relative to current window. By exploring the Palm OS SDK [26], we did not find it has API support for retrieving pressure and timestamp from a pen input event.

Palm OS APIs are implemented in the C language. They provide official support for Java Virtual Machine (JVM) which allows users to install and run Java applications on Palm PDAs. Taking advantage of the Java Native Interface, we can access pen input using Java, convert it to a platform-independent format and ultimately send it to portable digital ink applications.

2.2.8 iPhone OS

The iPhone OS is used on the iPhone and the iPod Touch devices from Apple Inc. Instead of input using a pen, it accepts finger input on a multi-touch screen. Finger input affords a different level of precision compared to pen input. If a finger taps on the screen, the contact area would approximately be an ellipse rather than a point as when using a pen. Also the tap pressure may be not uniformly distributed among that area. The iPhone OS analyzes all these information and computes a single touch point to represent the contact area.

The iPhone OS uses Cocoa Touch, a framework based on Cocoa [20], which provides direct support for handling user input. Just as with Cocoa, Cocoa Touch is built based on an event-driven architecture. If one or more finger taps are made on the screen, it will fire input events and route them to appropriate responders. Cocoa Touch currently supports two types of input event: `touch` event and `motion` event.

Both are represented by the `UIEvent` class which uses a type property to distinguish them.

The `touch` event contains information including the position of the finger, the number of taps received on the screen, the touch phases (down, moved and up) and the timestamp. Cocoa Touch manages active applications in a responder chain. It is a linked series of responder objects to which an event or action message is applied. When a `touch` event is generated, the iPhone OS traverses the responder chain to find an appropriate responder to handle it. The responder may render it on screen or keep it for later processing.

`Motion` events are fired when users move device in a certain way, such as shaking it. The iPhone OS evaluates and tests if it meets certain criteria. If so, it will interpret it as a gesture and create an `UIEvent` object to represent it. This `motion` event is then sent to the current active application for processing.

According to the iPhone OS SDK agreement 3.3.2, "An application may not itself install or launch other executable code by any means, including without limitation through the use of a plug-in architecture, calling other frameworks, other APIs or otherwise. No interpreted code may be downloaded or used in an application except for code that is interpreted and run by Apple's Documented APIs and built-in interpreter(s) [21]", since JVM allows Java applications to run without calling iPhone OS native APIs. It is obviously clear that neither Java nor Python are allowed on the iPhone OS. A possible solution for running Java applications on the iPhone OS is to translate Java code to Objective-C code through cross-compilation. This will be part of our future work.

2.2.9 Windows Mobile

Windows Mobile [24] is a compact operating system developed by Microsoft. It was designed for smartphones and mobile devices. Windows Mobile is based on Windows CE, an earlier Windows operating system for embedded systems. Most devices using Windows Mobile come with a pen and allow users to write on the screen instead of typing commands.

The current Windows Mobile is in its sixth version. It provides API support for digital ink. It offers a subset of the Windows XP Tablet PC SDK and its successor

Microsoft.Ink of the .NET Framework, including ink collection, data management, rendering, and recognition. It also provides ink controls to support note-taking scenario.

In seeking to interoperate with other applications running on Windows platforms, Windows Mobile natively supports Ink Serialized Format (ISF), a propriety format from Microsoft.

Windows Mobile pen input APIs are implemented in C and C++ language. Taking advantage of the Java Native Interface, we can collect digital ink from Windows Mobile devices and push it to high-level Java applications.

2.3 Previous Works on Portability

Previous work on digital ink portability has been conducted at the Ontario Research Centre for Computer Algebra. These include the theses of Xiaojie Wu [38] and Amit Regmi [32].

Wu's work focused on the conversions among UNIPEN, Jot and InkML in order to achieve data portability. Partial platform portability was also achieved in this work. It investigated two ink APIs, the IBM CrossPad API and the Microsoft Tablet PC SDK API, and developed an abstract API on top of them. This abstract API was partially implemented.

Regmi's work was based on the idea of portable multimodal collaboration. It developed a collaborative whiteboard application which provided cross-platform support for Windows, Linux and Mac OS X. It used InkML to exchange data. However, such portability still remained on the data level and was restricted to certain platforms. The implementation of the application varied from platform to platform. The whiteboard client for Windows was implemented in C#. It used the .NET Framework and was thus restricted to Windows platforms. The whiteboard client for Linux and Mac OS X was implemented in Python. Although Python supports cross-platform portability, the client was constructed using Linux-specific or Mac OS X-specific APIs and thus could not be ported to other platforms.

2.4 A Cross-Platform Digital Ink Framework

We suggest to develop portable digital ink applications using Java as it provides strong portability and applications can run on any Java Virtual Machine regardless of system architecture. For platforms that do not support Java directly, the Java code can be compiled to C or another native language.

2.4.1 Java Native Interface

The Java Native Interface (JNI) [34] is part of the Java platform and is designed to incorporate native code written in programming languages such as C, Objective-C and C++, together with code written in Java. Taking advantage of the JNI, programmers can make their Java applications interact with system-specific legacy code. Figure 2.2 illustrates how Java applications interact with native libraries. Java applications are written in the Java programming language and can be compiled to a machine-independent binary class format. The class can run on any JVM implementation in the host environment. Being part of JVM, the JNI can connect Java applications and native libraries and allow Java code to invoke native code and vice versa.

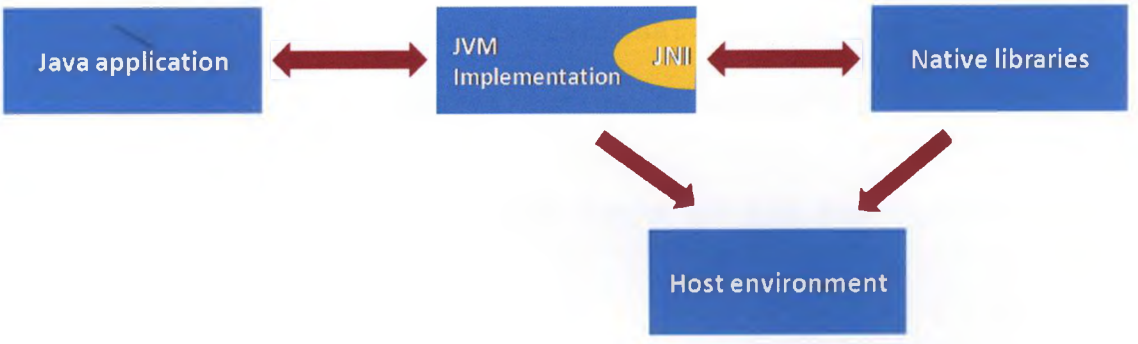


Figure 2.2: Java Native Interface

To use the JNI technology, one must first write a Java class which declares at least one native method. See Listing 2.1 for an example. The `InkProvider` class declares three native methods. Each native method's declaration contains a `native` modifier to indicate that this method will be implemented in another language. Before the native methods can be called, the `InkProvider` class must load the native library that implements these methods. This can be done by the static initializer of the `InkProvider` class.

```
public class InkProvider {

        /**
         * Load native libraray
         */
        static {
                System.loadLibrary("InkProvider");
        }

        /**
         * A native method that retrieves X coordinate
         * of the pen
         *
         * @return int
         *              The X coordinate of the pen
         */
        public static native int getPositionX();

        /**
         * A native method that retrieves Y coordinate
         * of the pen
         *
         * @return int
         *              The Y coordinate of the pen
         */
        public static native int getPositionY();

        /**
         * A native method that tests if the pen touches
         * the digitizer
         *
         * @return boolean
         *              If the pen touches the digiter,
         *              return true;
         *              Otherwise, return false
         */
        public static native boolean isPenDown();

        // Other methods
        ....
}
```

Listing 2.1: InkProvider.java

Having defined the `InkProvider` class, we need to compile the source file using the `javac` compiler. A `InkProvider.class` file will be generated in the current directory.

A JNI header file is useful when implementing the native methods in C language. This can be obtained by running `javah` on the `InkProvider.class` file.

```
javah -jni InkProvider
```

This command will generate a file named `InkProvider.h`, as shown in Listing 2.2. It provides three function prototypes for the native methods. Each prototype specifies its return value type and that it accepts two arguments. The first argument is the JNIEnv interface pointer which allows native methods to access data structures in the JVM. The second argument is a reference to the `InkProvider` object from which the method is invoked.

The native methods must be implemented complying with the prototypes specified

```
#include <jni.h>

/* Header for class InkProvider */
#ifndef _Included_InkProvider
#define _Included_InkProvider
#ifdef __cplusplus
extern "C" {
#endif

JNIEXPORT jint JNICALL Java_InkProvider_getPositionX
  (JNIEnv *, jobject);

JNIEXPORT jint JNICALL Java_InkProvider_getPositionY
  (JNIEnv *, jobject);

JNIEXPORT jboolean JNICALL Java_InkProvider_isPenDown
  (JNIEnv *, jobject);

  ....

#ifdef __cplusplus
}
#endif
#endif
```

Listing 2.2: InkProvider.h

```
#include <jni.h>
#include <stdio.h>
#include "InkProvider.h"

// Retrieves the X coordiniate
JNIEXPORT jint JNICALL Java_InkProvider_getPositionX(JNIEnv *,
    jobject) {
        // Get the X coordinate
        const int x = penInputEvent->x;

        return x;
}

// Retrieves the Y coordiniate
JNIEXPORT jint JNICALL Java_InkProvider_getPositionY(JNIEnv *,
    jobject) {
        // Get the Y coordinate
        const int y = penInputEvent->y;

        return y;
}

// Tests if the pen is down
JNIEXPORT jboolean JNICALL Java_InkProvider_isPenDown(JNIEnv
   *, jobject) {
        if(penDown is true) return true;
        else return false;
}
```

Listing 2.3: InkProvider.c

in the header file, as shown in Listing 2.3. For a proof-of-concept, the native methods get pen input data from the system and return it to the InkProvider class.

The last step is to build native libraries. Different operating systems support different ways to do this. A shared object library named `libInkProvider.so` will be created on Solaris and Linux machines. And a dynamic link library (DLL) `InkProvider.dll` will be created on Windows.

To assure that the `InkProvider` class works properly, the native library must be saved in one of the directories in the native library path. This can be accomplished by setting the `java` command line option, `-Djava.library.path`, to the location where the native library resides.

2.4.2 Java ME CDC for Mobile Devices

Java Platform Mirco Edition (Java ME) with Connected Limited Device Configuration (CLDC) [28] was developed by Sun Microsystems Inc. It only provides a base set of application programming interfaces and a virtual machine for resource-constrained devices like cell phones, pagers, and PDAs. Compared to Java Platform Standard Edition (Java SE), it provides limited feature support when developing Java applications. At the same time, it does not support the JNI, which means that it is impossible for Java applications to collect pen input data by this way.

Fortunately, Java ME defines an alternative, the Connected Device Configuration (CDC) HotSpot Implementation [27]. Its principal goal is to adapt Java SE technology from desktop systems to mobile devices. It uses the JNI as its native method support framework. Taking advantage of Java ME CDC, it becomes possible for Java applications to collect digital ink using the JNI.

2.4.3 A Cross-Platform Framework

Having explored available APIs on a variety of platforms including Windows, Windows Mobile, Linux, Mac OS X and Palm OS, we propose a framework that can capture digital ink across these platforms and provide a platform-independent, consistent interface to digital ink applications. This framework is illustrated in Figure 2.3.

This framework contains two layers, the platform layer and the JNI layer. The platform layer receives digital ink input from drivers and passes the data on to the upper interfaces, Wintab for Windows, XInput for Linux/Unix and Cocoa for Mac OS. As a rule, these interfaces push the data to user space in a platform-specific way and describe each data event inconsistently. For instance, Windows packs data events in Wintab packets which benefit Windows applications. On Linux or Unix platforms, the data events will be described using XInput events. On Mac OS platforms, the data events will be represented by the `NSEvent` objects which are restricted to Mac OS. Obviously, this puts all responsibility for event conformance on the shoulders of applications' implementation, which requires a huge effort. We desire that developers do not need to focus on the platform-dependence but on the functionality design. This leads to the design of the JNI layer.

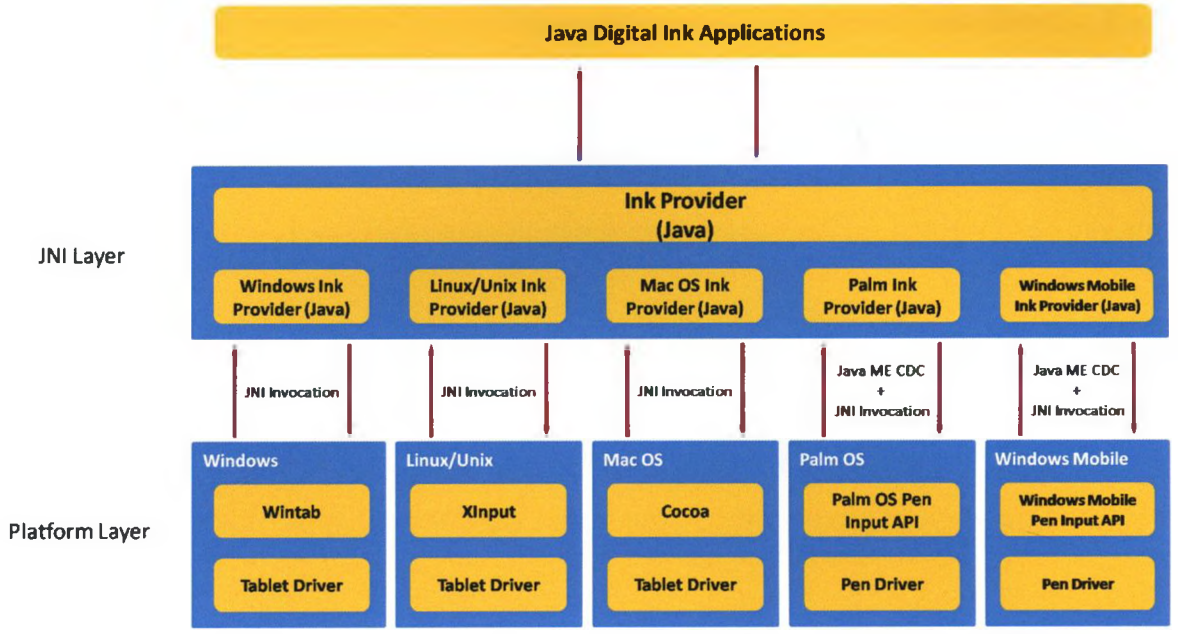


Figure 2.3: A Cross-Platform Framework for Digital Ink Applications

The JNI layer interacts with the platform layer and provides platform-independent, consistent APIs to digital ink applications. It is implemented using Java. As the Wintab, XInput, Cocoa, Palm OS pen input APIs and Windows Mobile pen input APIs are all implemented in C-like languages, the JNI layer can invoke them using the JNI. The JNI layer collects digital ink input from the platform layer, converts it to platform-independent events and then dispatches to digital ink applications. The benefit of doing so is that it saves developers from complex platform-dependence and allows them to focus on the functionality design, which eventually brings digital ink portability and useful manipulations.

2.5 Summary

In this chapter we have explored available digital ink input APIs on a variety of platforms including Windows, Windows Mobile, Linux, Mac OS X and Palm OS. We have identified their similarities as well as their differences. We have proposed a cross-platform framework that can collect digital ink in all these settings and provide a platform-independent, consistent interface for digital ink applications. We use the

JNI to invoke native libraries. For platforms that do not support Java directly, the Java code can be compiled to C or another native language.

Chapter 3

Multimodal Collaboration Design Issues

A number of applications have been developed in the past years to accommodate the needs of multimodal interaction. One of the most useful applications is the whiteboard system, where multiple users collaborate by means of voice, digital ink, text and video on a shared canvas. However, these systems normally use complex or proprietary formats for data exchange which consequently lead to poor portability and inter changeability. For instance, the use of proprietary ink formats such as Microsoft ISF(Ink Serialized Format) restricts applications to Windows environments. If another application wishes to interact with these applications but is running on a different platform or using a different data format, interoperability is not possible.

On the other hand, it would be useful for collaborative whiteboards to have strong portability, not just running on one operating system but many. There would also be a neat and elegant medium for flexible data exchange with other applications. Meanwhile, an improved rendering system along with several useful virtual brushes would eventually benefit users so that it looks as real as if it were written on a paper. It should also allow users to conduct various operations on the user interface, such as erasing by stroke, erasing by point, drag and dropping, copy and paste and so on.

Having identified this gap, the primary objective of this chapter were to analyze the design issues involved in multimodal collaboration.

3.1 Multimodal Input

Multimodal input is useful as it provides versatile means for users to interact with computers. These input modalities include keyboard, mouse, voice, pen, video and so on. We will focus on the voice input and pen input in this section, and explore their capabilities in collaborative environments.

3.1.1 Voice Input

Voice input is a fast and natural way to interact with computers. Most people speak faster than they can type or manipulate a mouse. Notably, certain people with physical disabilities prefer operating their computers simply by speaking. Voice input is hands-free, which is useful if one is driving. Also, voice input is flexible, one does not have to sit in front of computer: it is possible to use voice input while sitting, standing, or reclining. Although a headset microphone is typically required, it is still possible to use build-in microphones or other setups that don't require daily assistance.

3.1.2 Pen Input

With the widespread availability of pen-based devices such as Tablet PCs, PDAs and even cell phones, pen input starts playing an important role in human computer interaction. Pen input is a natural and powerful input modality since everyone learns to write in school. It is versatile as it provides more gestures and motions available compared to mouse and keyboard input. Pen input is also expressive. It is typically two dimensional and for some advanced digital pen it even supports three dimension (pressure sensitive). As a result, it can easily express symbols, signs and graphics. This is true especially in mathematics, as most of mathematical notations are two dimensional, with elements of both handwriting and drawing. These are hard to understand by the means of voice, keyboard or mouse, but could be easily expressed using a pen.

3.1.3 Voice and Pen Multimodal Input

We choose voice and pen as the input modalities for collaboration as together they add more benefits than either alone.

The advantages of voice and pen multimodal collaboration include:

1. **Portability**
   Most computing platforms support both voice and pen input.

2. **Easy to use**
   No extra assistance with setup is required to use voice and pen input.

3. **Complementary input modality**
   Both voice and pen input can work independently and simultaneously. One can write and speak at the same time, or separately.

4. **Adaptive to the environment**
   In some cases, one can singly use voice input as it is fast and requires less attention. If it is extremely noisy or requires expressive illustration, it becomes useful to use pen input.

3.2 Multimodal Collaboration

3.2.1 Voice Collaboration

With the emergence of VoIP and net phones in late 1990s, voice collaboration has become more and more prevalent as it provides users real-time communication services such as IP telephony, conference calls, voice controls and so on. Today, these services have been integrated into many known collaborative applications, such as IBM Lotus Sametime Unified Telephony, Google Wave and so on. Such applications help users easily find, reach and collaborate with each other.

Voice collaboration is not just concerned with collaborating with another person using voice, it is actually more than that. For instance, people with disabilities may find it difficult to operate a computer using a keyboard or a mouse. It would be

helpful if they could control the computer using voice. The computer would then try to recognize the voice and conduct appropriate operations.

We integrate voice collaboration along with digital ink collaboration into our whiteboard system, as we believe that they are complementary with each other in that the combination will provide a real-time, effective and expressive communication for our users.

3.2.2 Digital Ink Collaboration

As discussed earlier, pen input provides a natural and convenient way to interact with computers. One of the most useful applications is the collaborative whiteboard, where multiple users send and receive digital ink messages by using pen input devices to write and draw on a shared canvas. This shared canvas may be blank or already contained some shared information such as a map,an image or previous ink work.

The whiteboard is useful but at the same time the implementation is challenging, especially when the collaborative environments are heterogeneous. Different pen devices may have different properties. For instance, some digital pens only support two channels input, *i.e.* X and Y coordinate. While, advanced digital pens may allow three or even more channels input, *i.e.* X, Y, pen tip force, tilt angle and so on.

For portable representation and flexible interchange of digital ink, it is important to save it in a platform-independent and open standard format. Fortunately, InkML provides a wide range of features to support such capabilities. To achieve portability, InkML describes each pen device using an <inSource> element with a list of sub-elements, <traceFormat>, <sampleRate>, <latency>, <activeArea>, <srcProperty>, <channelProperties>, which in turn specifies the trace format, sampling rate, latency, resolution, additional ink source properties and channel properties. Digital ink generated by the pen will be represented by <trace> elements, conforming to the pen device properties. New traces will then be either streamed to other collaboration participants or archived in file system for later processing.

InkML streams digital ink based on the concept of context. The context is represented by the <context> element which contains various associated aspects, including <canvas>, <canvasTransform>, <traceFormat>, <inkSource>, <brush>, <timestamp>. Initially, each ink collaboration participant sets up a default context

and listens to context changes. This is similar to a event-driven model in that context changes will be made when <context>, <traceFormat> and <brush> elements are received. In practice, these elements will intersperse among digital ink streams, see Figure 3.1.

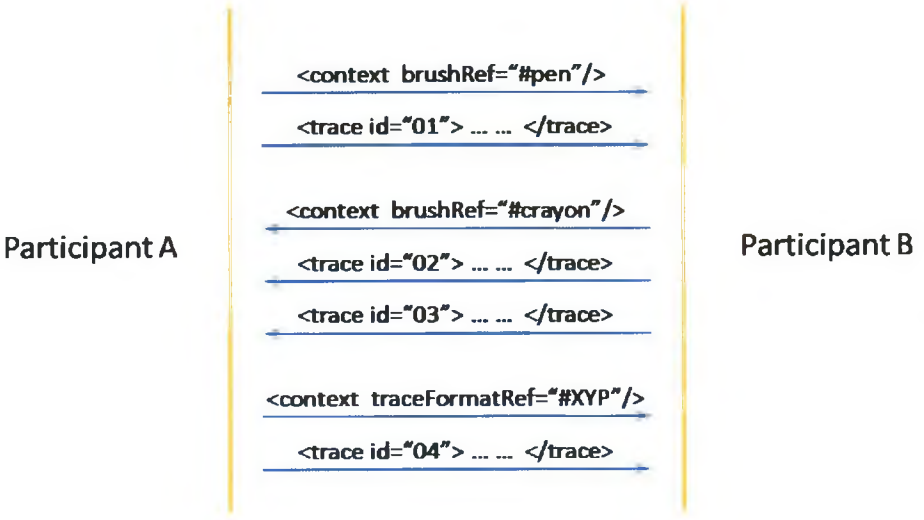


Figure 3.1: Digital ink streams

As each trace is associated with a context, the trace cannot be parsed properly until its associated context has been established with receivers by means of sending a <context>, <traceFormat> or <brush> element. This may increase transmission overhead when it only switches among a few favored contexts. See Listing 3.1 and Listing 3.2 for instance. Listing 3.1 shows a scenario of brush changes. It first changes current brush to "pen" and processes trace "01". It then changes the current brush to "crayon" as the <brush id="crayon"/> is received. Later, it switches the current brush back to "pen" and processes trace "03" and trace "04". Listing 3.2 shows a scenario of trace format changes. It switches from trace format "XY" to "XYP" and then switches back.

Obviously, sending the <brush id="pen"> and the <traceFormat name="XY"> element twice is unnecessary as receivers already know the "pen" and the "XY" trace format the first time they were used. A smarter way to do this is to save reusable elements in the <definitions> element. These reusable elements can be referenced by other elements using appropriate syntax. See Listing 3.3 for instance, the trace "01" refers to the "pen" brush and the trace "07" refers to the "crayon" brush. Initially, each ink collaboration participant can set up default definitions. If the asso-

```
        ......
        ......
<brush id="pen"/>
<trace id="01">45 29, 47 26, 50 26, 52 23</trace>
<brush id="crayon"/>
<trace id="02">37 22, 34 19, 33 19</trace>
<brush id="pen"/>
<trace id="03">21 41, 21 45, 23 44, 25 40</trace>
<trace id="04">30 30, 28 26, 28 26, 27 25</trace>
        ......
        ......
```

Listing 3.1: Brush changes

```
        ......
        ......
<traceFormat id="XY">
        <channel name="X" type="integer">
        <channel name="Y" type="integer">
</traceFormat>
<trace id="01">37 22 15, 47 26, 23 44, 25 40</trace>
<traceFormat id="XYP">
        <channel name="X" type="integer">
        <channel name="Y" type="integer">
        <channel name="P" type="integer">
</traceFormat>
<trace id="02">21 45 15, 34 19 18, 33 19 12</trace>
<traceFormat id="XY">
        <channel name="X" type="integer">
        <channel name="Y" type="integer">
</traceFormat>
<trace id="03">21 41, 21 45, 23 44, 25 40</trace>
<trace id="04">30 30, 28 26, 28 26, 27 25</trace>
        ......
        ......
```

Listing 3.2: Trace format changes

ciated context of a trace is already included in the definitions, instead of establishing the context with receivers by broadcasting a complete <context>, <traceFormat> or <brush> element, we can specify the context using the trace's `contextRef` and `brushRef` attributes. When the trace is received by other participants, the associated context can be set up by dereferencing the `contextRef` and `brushRef` attributes.

```
<trace id="01" brushRef="#pen">22 5, 23 7, 25 10</trace>
                  ......
<trace id="07" brushRef="#crayon">29 25, 30 25</trace>
```

Listing 3.3: Brush reference

3.3 Communication

3.3.1 Network Architecture

There are a number of network architectures available when considering the communication issues involved in whiteboard systems. We focus our analysis on the two most popular ones: Client-Server networks and Peer-to-Peer networks.

Client-Server networks

A Client-Server Network is a distributed network architecture that separates tasks and work loads between servers and clients, as shown in Figure 3.2. Each of the clients sends data requests to one or more connected servers. In turn, servers may reject these requests or accept them, process them, and return the requested information to the client. Standard networked functions such as email exchange, website and database access are based on this model.

The Client-Server's data storage is centralized. Only the server administrator has permission to update data. This may become bottleneck when designing collaboration applications. As opposed to keep data in one place, collaborative applications require that data and resources can be distributed and shared among participants. In collaborative applications, each participant should be able to work on a independent part or cooperate with others interactively.

The Client-Server networks also lack robustness. Should a critical server fail, the client's communication can be broken.

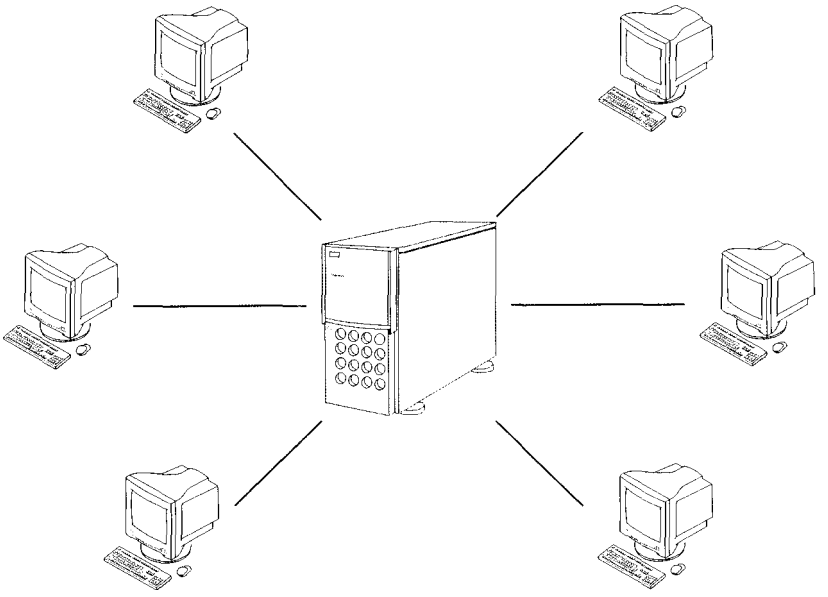


Figure 3.2: Client-Server network architecture

Peer-to-Peer networks

A Peer-to-Peer network, also known as P2P, is a distributed network architecture where each participant shares a portion of resources that are also available to the others. This is illustrated in Figure 3.3. In contrast to the Client-Server networks that keep data centralized on a server and only the administrator has permission to update it, the P2P networks allow each participant to make changes to his or her data copy and to directly share it with the others without communicating via a "server". This feature is very suitable for collaboration applications. Moreover, the distributed nature of P2P networks also increases robustness. It enables peers to directly find data without relying on a centralized index server.

3.3.2 Skype

Skype is a software application that allows user to communicate by voice and video as well as the text messaging. It has experienced a rapid growth in popular use since it was launched. It has become the present worldwide leader in VoIP. Skype uses an overlay Peer-to-Peer network which is a compromise between a pure Peer-to-Peer network and a Client-Server network. It organizes its nodes into two levels, normal node and super node, as shown in Figure 3.4.

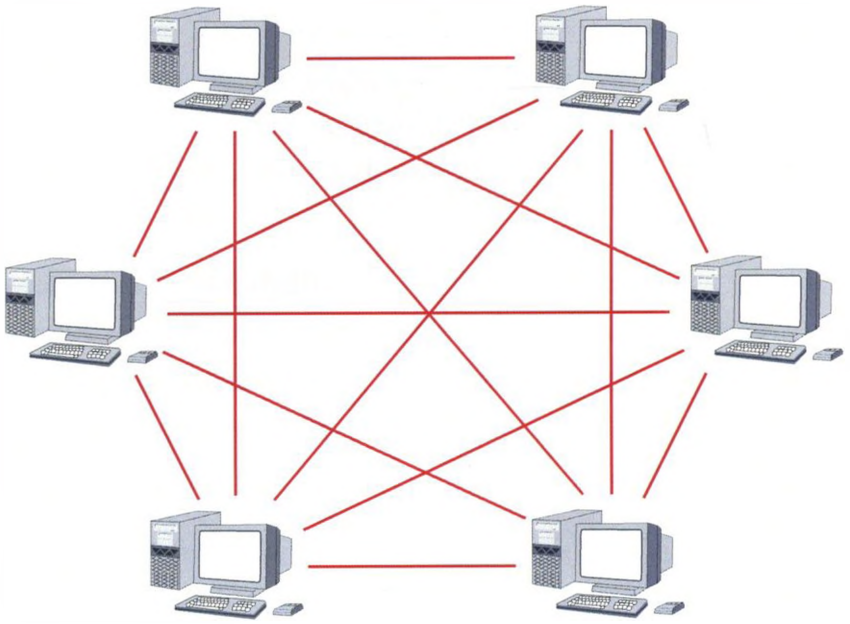


Figure 3.3: Peer-to-Peer network architecture

Any node with a public address having sufficient CPU, memory and network bandwidth can be a candidate to become a super node. This super node interconnected with the other super nodes that helps relay queries and acts as a proxy connection to normal nodes behind firewalls. Each normal node keeps an index of super nodes and can get involved by connecting to one of them. Such a decentralized distributed network is more robust than a traditional Client-Server network. Meanwhile, compared to a pure Peer-to-Peer network, the super nodes improve network scalability.

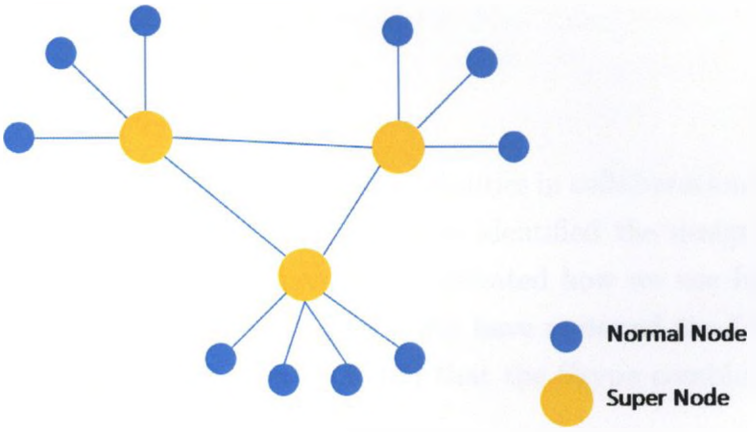


Figure 3.4: Skype network architecture

As discussed earlier, collaboration applications require that data can be easily collected and shared among its participants. With the distributed nature, Skype network is right for this job. We decided to use Skype as the backbone to transmit data over clients also based on the following considerations.

- **Cross-platform availability**
  Skype has a number of versions exist for different operating systems, including Linux, Linux-based Maemo, Mac OS X, iPhone OS, Microsoft Windows, Windows Mobile and even Sony's PSP.
- **API accessibility**
  The Skype APIs are publicly available and have been encapsulated in many programming languages including Java, Python and C#. These wrapper libraries make it possible to develop portable collaboration applications using Skype.
- **Cost effective**
  Skype client can be freely downloaded from its website and along with free IM, VoIP and file transfer functions. It only requires a computer with Skype installed and the Internet access to use its services.
- **Reliable and secure service**
  Skype provides reliable services even in miscellaneous networks. Moreover, Skype uses secure communication, any data transmitted between Skype clients will be encrypted. This encryption cannot be disabled and is invisible to users.

3.4 Summary

We have chosen voice and pen as our input modalities in collaboration as they together add more benefits than either alone. We have identified the design issues involved in digital ink collaboration. We have also illustrated how we use InkML to stream digital ink in heterogeneous environments. We have reviewed the Client-Server and Peer-to-Peer network types, and have found that the Skype combination is suitable for multimodal collaboration applications.

Chapter 4

InkChat Architecture

InkChat is a Java-based, portable collaborative whiteboard system which allows conducting conversations involving voice and digital ink on a shared canvas. This chapter illustrates InkChat's design and its architecture.

4.1 System Architecture

InkChat makes use of the cross-platform framework presented in the Section 2.4.3, whose primary purpose is to collect digital ink from a variety of platforms and to provide a platform-independent, consistent interface for digital ink applications.

Figure 4.1 illustrates the architecture of InkChat. The lowest layer is the platform APIs layer. One of the design goals of InkChat is to be portable. We expect that InkChat can interact with these platform APIs and capture digital ink regardless of its operating platform. As these APIs are typically implemented in C or C-like languages, we use the JNI to invoke their functionalities.

The Digital Ink Cross-Platform Framework consists of two types of components. The Ink Provider provides a platform-independent, consistent interface to upper digital ink applications. It also provides a prototype for its concrete, platform-specific Ink Providers, including Windows Ink Provider, Linux/Unix Ink Provider, Mac OS Ink Provider, Palm Ink Provider and Windows Mobile Ink Provider. Each of these Ink Provider classes implements the prototype. It works like an agent that invokes

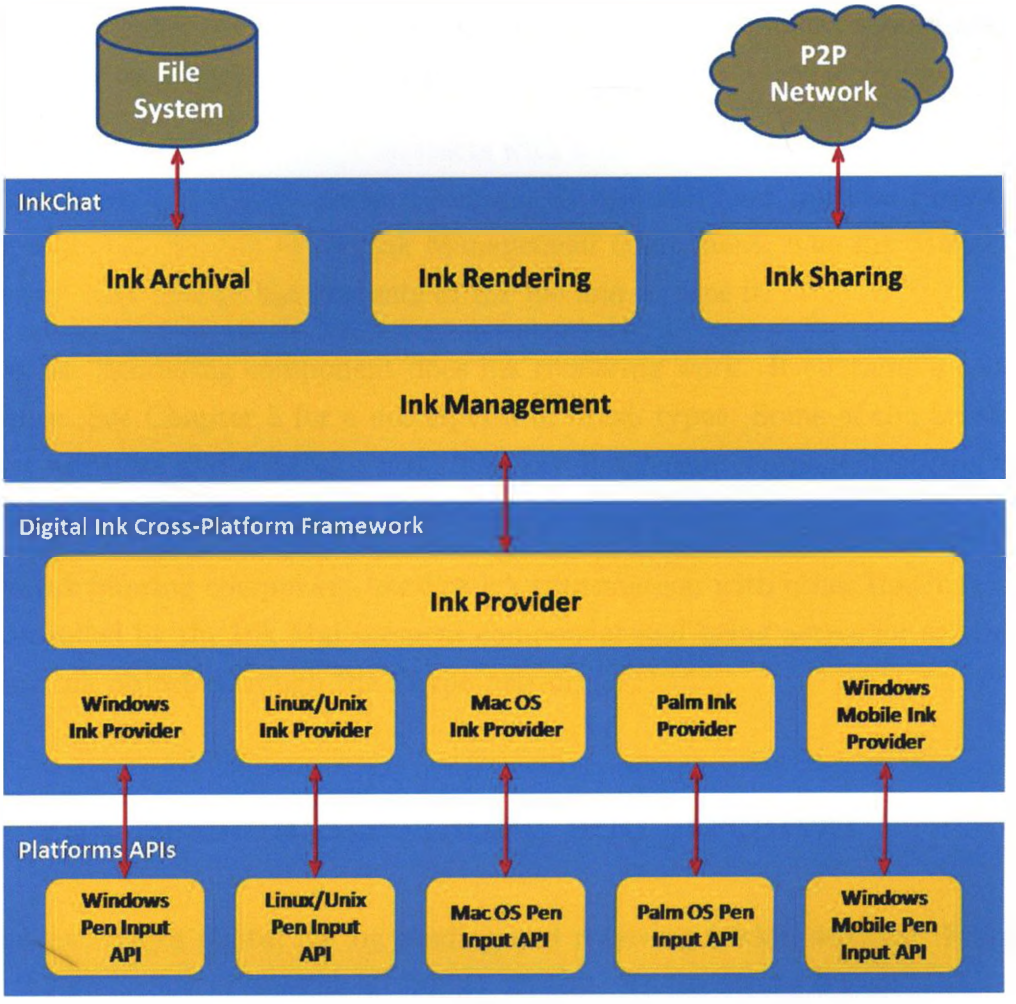


Figure 4.1: InkChat Architecture

platform APIs and responds to the application's requests. This whole layer is implemented using Java.

InkChat consists of four components and each of them is in charge of a particular responsibility. The Ink Management component accepts digital ink input from the cross-platform framework and converts it to an appropriate object for processing. This may involve noise removal, pen down, pen up, pen movement and non-contact pen pointing. It is also responsible for ink manipulations on the user interface. If an eraser is being used on the canvas, the Ink Management component continues detecting whether a target object is selected. If so, the object may be immediately removed from the canvas. In seeking to seamlessly work with other digital ink applications

that may use other ink formats, the Ink Management component should also have the flexibility to interchange the data.

The Ink Archival component interacts with a file system. It documents user's ink work and saves it using a standard format. It can also load ink files from the file system and pass data on to the Ink Management component. The Ink Management component may change the contents of the file and replace it.

The Ink Rendering component does ink rendering work. It contains a collection of brushes. See Chapter 5 for a description of brush types. Some of the brushes are pressure sensitive and some are not. The Ink Rendering component should render ink strokes in terms of the brush type.

The Ink Sharing component handles ink transmission with other InkChat clients. It is controlled by the Ink Management component and being active for sending and receiving ink objects through the Skype networks.

4.2 Ink Session Streaming and Archival

InkChat exchanges digital ink by sending and receiving InkML streams. Figure 4.2 shows the procedure of sending an InkML stream. An ink stroke is a collection of ink points. Each point consists of several coordinate values recording its position, pen tip force and so on. When an ink stroke is captured by the interface, the Ink Management component will encapsulate it as an InkML trace with additional information such as the current context and notify the Ink Sharing component that the InkML trace is ready to be sent. The trace will then be sent as an InkML stream to its destinations through the Skype networks by calling the Skype APIs. Meanwhile the Ink Archival component will also archive the InkML trace in the local file system. All ink strokes can be re-rendered by loading files from the file system when requested by the user interface.

Fig 4.3 shows the procedure of receiving an InkML stream. The Ink Sharing component passes any received InkML stream on to the Ink Management component. The Ink Management component will then parse the stream, retrieve its self-contained context and then convert it to an ink stroke for rendering. Meanwhile, these InkML streams will be saved in InkML files which are kept in the local file system. Such

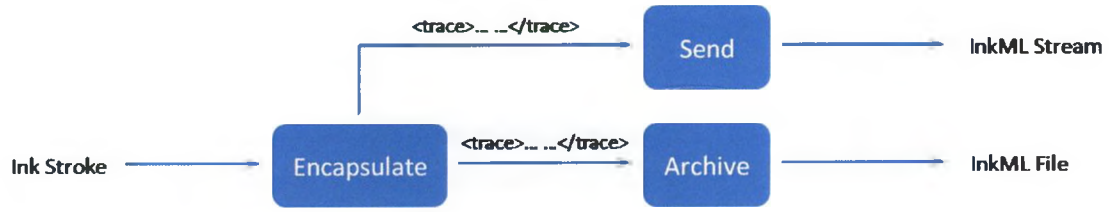


Figure 4.2: Sending an InkML stream

mechanism ensures that all participants can share consistent copies of the current ink session. This is helpful when users wish to pause and continue a conversation.

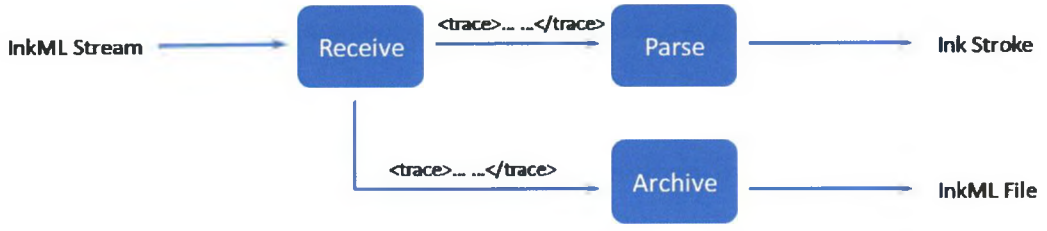


Figure 4.3: Receiving an InkML stream

4.3 Ink Session Retrieval

Users may wish to load a previously recorded ink session. This is implemented by loading an InkML file and retrieving all the information contained. As discussed earlier, contextual elements are defined within a <definitions> element. Figure 4.4 shows the procedure of retrieving an ink session from an InkML file. InkChat first loads the InkML file. The Ink Management component parses the <definitions> element and saves all contextual information to the current ink session. It then parses all traces and converts them to various ink stroke objects in terms of their corresponding contextual information. Meanwhile, these ink stroke objects will be processed at once.

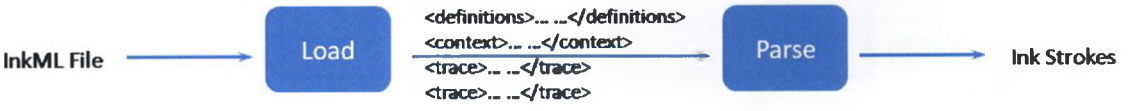


Figure 4.4: Loading an ink session

4.4 Canvas Layers

InkChat's canvas consists of three equal size layers, as shown in Figure 4.5. The bottom layer, the Image Canvas, is used to load images. Users can switch the conversation themes by loading various background images, such as calendars, music script paper and so on. The middle layer, the Ink Canvas, does the rendering work. All kinds of brushes, including erasers, work on this layer. These brushes draw ink points reported by the tablet driver or received from other participants. This layer is transparent and thus leaves the Image Canvas visible. The top layer is the Selection Canvas, it is like a sheet of glass over all the other layers. Users can select part of their ink work by drawing a selection free-hand with a stylus, while pressing it against the tablet (or, for a pointer, holding down the left mouse button). When lifting the stylus, the selection will be closed by connecting the current location to the start location with a straight line. All ink strokes contained by that area will be selected and passed on to the next operations, such as erasing, copy and paste, drag and drop and so on. The selection Canvas is useful as it also can be used to intercept input events. For instance, in teaching mode where only the teacher is allowed to write on the shared canvas while students can only view it, all input events on students' side can be intercepted by the Selection Canvas.

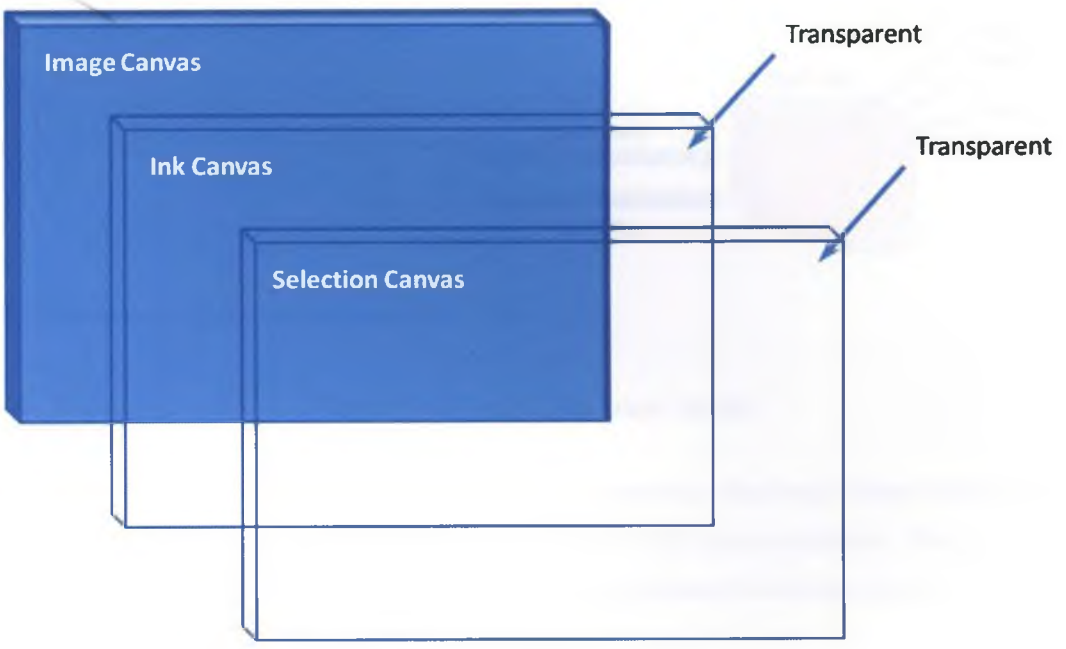


Figure 4.5: InkChat canvas layers

In addition to the three layers, InkChat also provides a flexible way to add more layers for application needs. When adding a new layer to the canvas, one must specify its depth as an integer. InkChat positions its layers in terms of their depth. The higher the number, the closer the component is to the "top" position. If two layers has the same depth, the relationship between the two layers is determined by which layer is added first. The first layer added will be overlapped by the ones added later.

4.5 Page Model

InkChat also supports a page model. The page model is useful when a user wishes to cover multiple topics in one session or to save and load in the middle of his or her document work. In both cases, the current page will first be saved to the file system as an InkML file. Then the Ink Canvas will send a page request to the file system to check if the next page is already available. If so, the Ink Canvas will load and parse the InkML file. It will then render the digital ink and allow the user to continue to work on that page. Otherwise, a new page will be created in the interface. Figure 4.6 illustrates the working procedure of the page model.

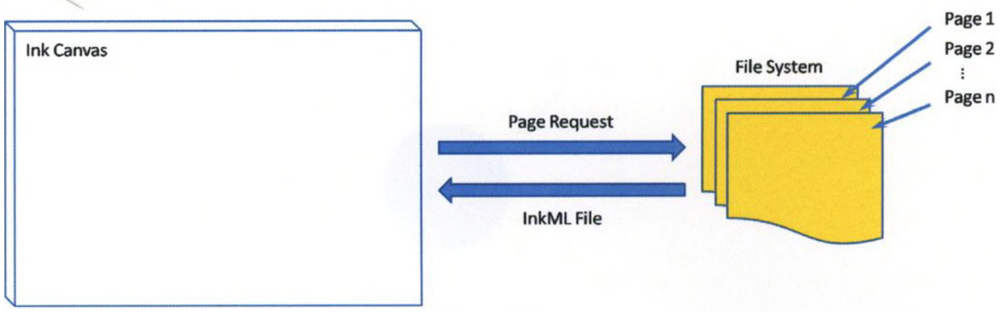


Figure 4.6: InkChat page model

If the user is involved in a multi-party conversation, the local client may send and receive InkML annotation streams to synchronize their page numbers. Such capability would be useful if InkChat is being applied in a distance learning environment.

4.6 Conference Mode

InkChat supports a conference mode where more than two participants can be involved in one conversation. The conference is initiated by the host which has a connection with every other participant. Audio routing to other participants is handled by the Skype client internally and is written to the corresponding WAV files using the Skype API. Digital ink routing shares the same mechanism that each ink stroke will be broadcast by the host to all participants except the initiator. Figure 4.7 and Figure 4.8 show the two cases when a host and when a client initiate a stroke.

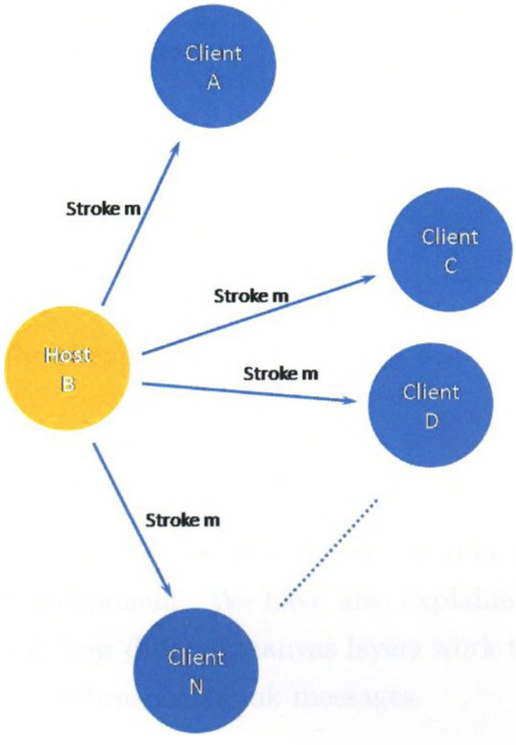


Figure 4.7: The host initiates a stroke in conference meetings

As shown in Figure 4.7, if a stroke is initiated by the host, it will be broadcast to all the clients.

As shown in Figure 4.8, if a stroke is initiated by a client, it will be sent to the host and then broadcast to all the other clients.

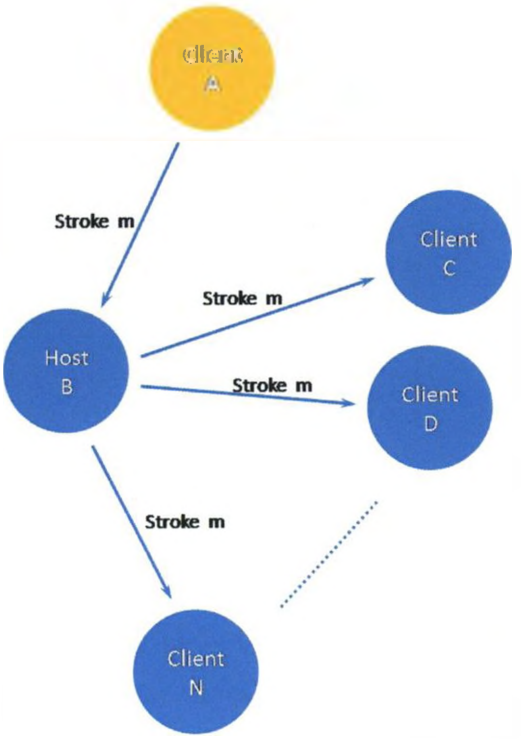


Figure 4.8: The client initiates a stroke in conference meetings

4.7 Summary

In this chapter we have presented the architecture of InkChat and have described the functionality of each component. We have also explained how InkChat handles ink streaming and archival, how different canvas layers work together to facilitate ink operations and how InkChat broadcasts ink messages.

Chapter 5

Ink Rendering

Although digital ink technologies have evolved over years, the "brushes" used by digital ink applications are still far simpler than real brushes. They are normally two dimensional, only recording X and Y coordinates. Few of them have support to record additional information such as pen tilt angle, pen tip force, timestamp *etc.* which are essential when simulating more complex writing instruments.

Various brush models have been developed and used on computers to simulate hair brush properties. One of the earliest attempts is that of Steve Strassmann [33]. It describes an investigation into a realistic model of painting. It simulated the behaviors of a brush with wet paint on paper. Wong and Ip [37] presented an approach to simulate the physical process of brush strokes creation using a parameterized model which captures the brush's 3D geometric parameters, the brush hair properties as well as the variations of ink deposition along a stroke trajectory.

Our work is similar to those in that we all keep track of each ink point and calculate its contour to simulate brush properties. Our work is also apart of those as we are using InkML, an open and up-to-date standard, to represent and render digital ink.

We present Calligraphy Board along with two virtual brush models in this chapter. The Calligraphy Board uses the framework proposed in the Section 2.4.3. It collects digital ink from a variety of platforms and renders it with calligraphic properties. We will leave the framework implementation to Chapter 6 and focus on the design of the two virtual brush types.

5.1 InkML Representation

After considering various alternatives, we arrived at a design using InkML, as it provides application-defined channels to support sophisticated ink representation. Each application-defined channel provides a coordinate value at each ink point, e.g. as for an InkML point (x, y, r, l, θ) which specifies a five-channel trace format. Applications may arbitrarily define x, y to indicate the position of an ink point, r, l to measure the brush head radius and the tail length, θ to represent the rotation angle of the brush tail. Such feature is extremely useful as it allows applications to flexibly represent ink strokes and to render them with various properties according to the context.

Taking advantage of InkML, we propose a approach to render sophisticated digital ink. we first capture digital ink and retrieve information such as coordinates, pen tip pressure, timestamp and tilt angle from pen-based devices. We then simulate the dynamic brush shapes by calculating the contour at each ink point. Finally we can render it in terms of the brush type.

To demonstrate our ideas, we have developed two virtual brushes. Both of them can render digital ink with calligraphic properties.

5.2 Calligraphic Rendering

With the widespread availability of pen-based devices, it becomes interesting to render digital ink with calligraphic properties. This is particular useful when aesthetic and decorative effect are desired. Calligraphy, especially Chinese calligraphy, cannot be aesthetic without stroke width variations. Therefore, rendering digital ink with calligraphic properties is not as easy as drawing a set of ink points and connecting them using simple lines or curves, but must embody the brush physical properties and express calligraphers' writing style.

We take Chinese calligraphy for an example. Since the brush is made of soft hairs, there are abundant width variations even in one simple stroke, as shown in Figure 5.1. A stoke normally gets fat at the beginning as calligraphers push hard when the brush tip drops on the paper. Then it gets thinner while the brush is moving. Once the brush reaches the stroke end, its tip suddenly turns back and the stroke gets fat again due to the delay of the brush tail.

Figure 5.1: A simple stroke of Chinese calligraphy.

We see that calligraphy cannot be rendered using simple lines or curves due to its stroke width variations. We thus use a different approach: instead of simply connecting ink points, we calculate the contour of each ink point to simulate the dynamic brush shapes while it is moving. Obviously, different brush types result in different two dimensional contours of an ink point. Even using the same brush, different pressures may cause contour size and shape variations. Therefore, after capturing digital ink from pen-based devices, we convert it to brush-oriented parameters and then render it in terms of the brush type.

We start from the Round Brush. We have also developed a more sophisticated one, the Tear Drop Brush, to simulate Chinese calligraphy. The following two subsections illustrate the two brushes individually.

5.2.1 Round Brush

The Round Brush, as its name suggests, draws each ink point as a filled circle. We use three parameters to model the Round Brush, as shown in Figure 5.2, x, y to indicate the position and r to measure the circle radius which is a function of pen tip pressure. Basically, the harder you press the brush, the larger the circle you would get. In our approach, the radius of the circle r is directly proportional to the pressure p, where k is a constant.

$$r = k \times p$$

Pen-based devices work with a certain sampling rate. A higher sampling rate indicates it samples more ink points in a unit time and leads to a greater chance that two successive circles would overlap. A lower sampling rate indicates it samples fewer ink points in a unit time and leads to a smaller chance that two successive circles

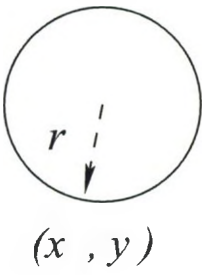


Figure 5.2: The model of Round Brush

would overlap. In addition, if the brush moves fast, the chance of overlapping would correspondingly decrease. Practically two successive circles do not overlap if the brush is moving. Even if they are overlapped, there is a great chance that they are not fully overlapped. As long as full overlap does not exit, there will be a gap between each pair of successive circles. In seeking to have a smooth rendering, we fill the gaps using the method shown in Figure 5.3.

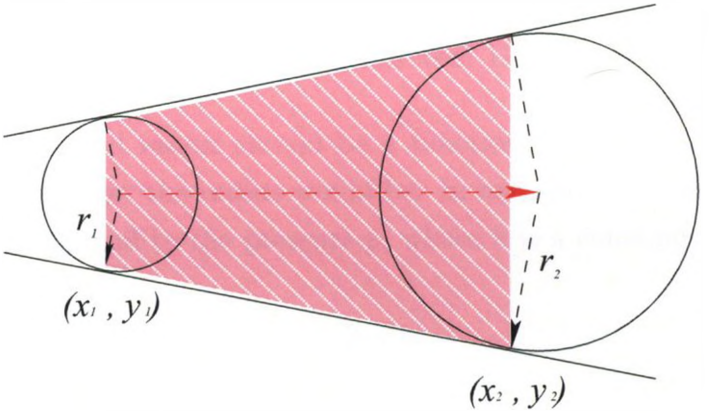


Figure 5.3: Filling the gap between two successive Round Brush ink points

As shown in Figure 5.3, as long as two successive circles do not fully overlap, we calculate their external tangents and color the area determined by the four tangent points. Otherwise, we do nothing as either circle is covered by the other.

5.2.2 Tear Drop Brush

The idea of the Tear Drop Brush comes from the Round Brush and it is more sophisticated. Holding a brush upright and pressing the tip on to paper, then lifting it up

quickly, would leave a round dot. If you then press the tip on to paper and drag it for a tiny distance, there would be a dot that looks like a tear drop left on the paper. Finally, if you press the brush very hard, the soft hair would spread out and the ink mark would become a fat round dot again. Based on this idea, we use five parameters to model the Tear Drop Brush, as shown in Figure 5.4, x and y indicate the position of the ink point, r represents the head radius, θ indicates the rotation angle of the tail, and l measures the tail length, the distance from the circle center to the tail end.

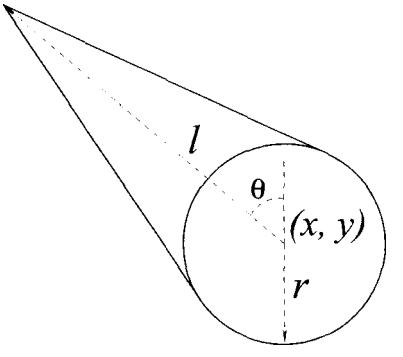


Figure 5.4: The model of Tear Drop Brush

In principle, the head radius should be a function of pressure. The harder you press the brush, the larger the radius would be. In our approach, we set the radius r to be directly proportional to the pressure p, where k is a constant.

$$r = k \times p$$

The tail length l ranges from r to L, the length of the brush hair. Its value depends on the distance between the old tail end and the new ink point.

We can describe this using what is called the "donkey wagon" model. Imagine the tail end of the tear drop shape is a wagon on a rope being pulled by a donkey at the center of the head. When the donkey changes direction the wagon follows the rope, which will be a straight line to the donkey's current position. If the donkey moves in a direction that makes the rope lose, then the wagon stays in the same place. Likewise, there are two cases when moving the Tear Drop Brush. If the distance between the old tail end and the new ink point is smaller than L, the new tail end would stay at its original place and we fill the area as shown in Figure 5.5. If the distance exceeds L, the new tail end would move to somewhere on the line determined by the old tail

end and the new ink point, while the tail length becomes L. And we fill the area as shown in Figure 5.6

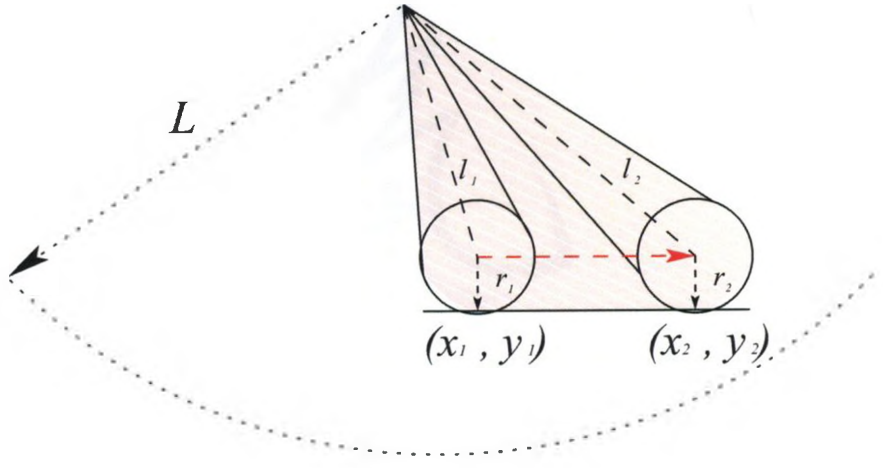


Figure 5.5: Tear Drop Brush: The case that the tail end remains.

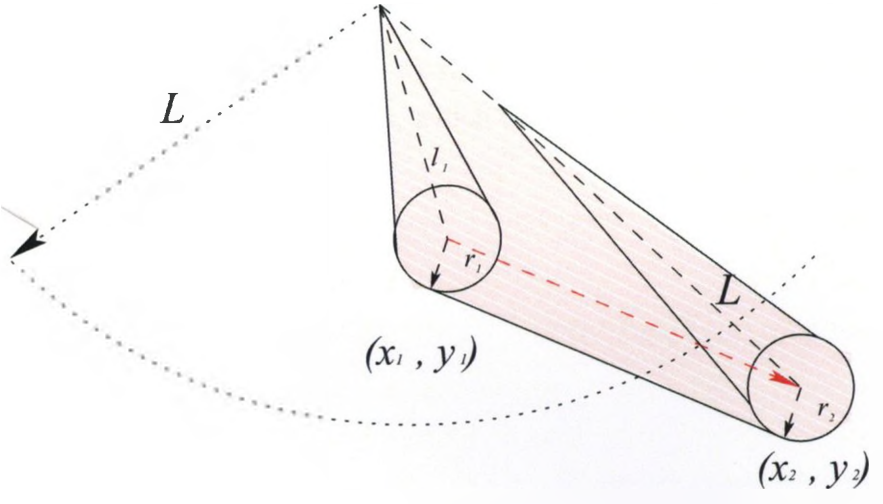


Figure 5.6: Tear Drop Brush: The case that the tail end moves.

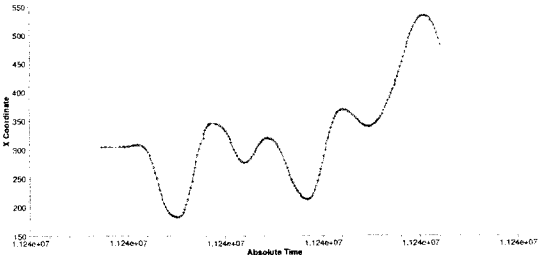
In the implementation, we retrieve coordinates and pen tip pressure at each ink point from a Wacom tablet running on Ubuntu 8.10. We convert it to the Tear Drop Brush parameters and then save them into InkML points (x, y, r, θ, l). An example image of the Tear Drop Brush calligraphy can be seen in Figure 5.7. Figure 5.8 shows the plot of the first stroke of Figure 5.7.

Figure 5.7: Example of a Chinese calligraphy using Tear Drop Brush.

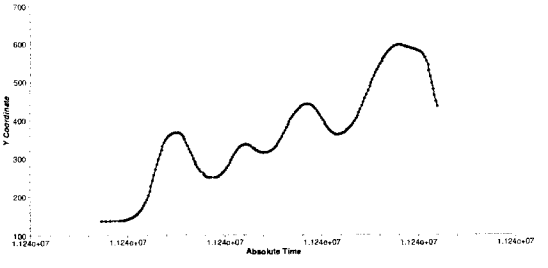
5.3 Summary

In this chapter we have described how to use InkML to represent sophisticated digital ink formats. To demonstrate this idea, we have presented two virtual brush types, the Round Brush and the Tear Drop Brush, which can render digital ink with calligraphic properties.

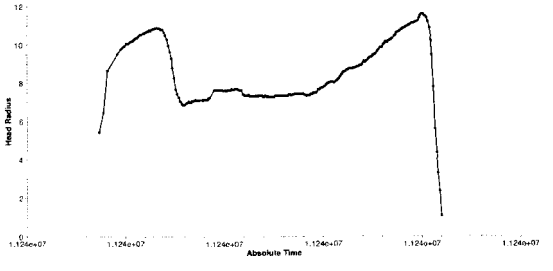
(a) The calligraphic ink stroke to plot



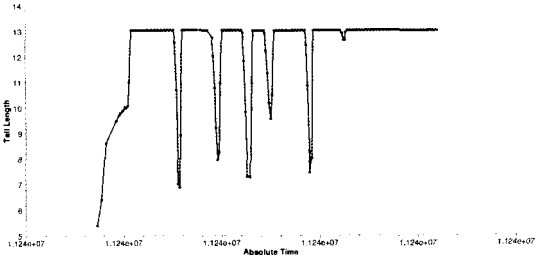
(b) The plot of x coordinates



(c) The plot of y coordinates



(d) The plot of the head radius



(e) The plot of the tail length

Figure 5.8: The plot of Tear Drop Brush parameters

Chapter 6

InkChat Implementation

6.1 InkChat Components

InkChat makes use of Skype as the backbone for sending and receiving data and thus provides reliable data exchange over miscellaneous networks. Figure 6.1 shows the framework of InkChat. Because InkChat handles data exchange using Skype, the underlying communication protocol is hidden so it does not matter whether App2App or Conference mode is being used. The Skype APIs can be accessed by calling the methods wrapped in Skype4Java [6], an open source library which enables developers to make add-ons on multiple platforms and to use good IDEs, libraries and framework.

The InkML Toolkit parses InkML streams as well as InkML documents. When an InkML trace is received by a client, it parses the InkML trace and converts it to an ink stroke for a real-time rendering. When loading ink is requested by the interface, the InkML Toolkit parses a complete InkML document, converts all InkML traces to ink strokes and records the InkML definitions and contexts for the following ink sessions.

The JPen [3] library is an open source library that works with different operating systems. It collects digital ink reported by a tablet driver. It was implemented on an event/listener architecture. Each digital ink or digital ink gesture that is reported by a tablet driver will be fired as an event to its listeners. The InkChat interface is responsible to handle these events and conducts corresponding operations, such as

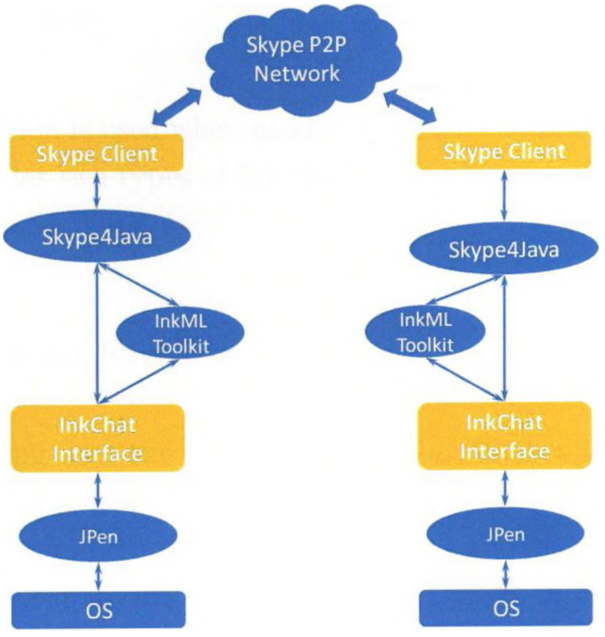


Figure 6.1: The system architecture of InkChat

brush switching, stroke-wise erasing, point-wise erasing, color and theme switching, ink saving and loading, page navigation, copy and paste, *etc.*

6.2 Data Transmission

As discussed earlier, the Skype network is most suitable for multimodal collaboration. InkChat uses Skype as the backbone to send and receive data over clients. Skype APIs can be accessed using Skype4Java, a wrapper library built on top of the Skype APIs. It handles the connection between an InkChat interface and a Skype client. It sends voice and digital ink using separate data channels to avoid transmission delay.

6.3 Ink Capture

InkChat makes use of the JPen library to capture digital ink across platforms. The JPen library is a Java-based library that uses the JNI to invoke platform-specific APIs. Currently, JPen can capture digital ink on Windows (Wintab), Linux (XInput) and Mac OS X (Cocoa).

6.4 Ink Erasing

The Ink Erasing function is used when users wish to edit their current or previous ink work. InkChat provides two types of ink erasing, stroke-wise erasing and point-wise erasing.

6.4.1 Stroke-Wise Erasing

The stroke-wise erasing uses the hit testing approach that will be discussed in the Section 6.5 to detect whether a stroke is selected. If so, it removes the target stroke from the canvas and re-renders the other strokes that may be affected. We re-render these strokes in their original sequence to keep color layers in line.

6.4.2 Point-Wise Erasing

The point-wise erasing erases part of a stroke instead of removing the whole stroke from the canvas. A stroke may be split into pieces when using point-wise erasing. This requires the application to detect at which part the stroke is broken up.

Point-wise erasing uses the hit testing approach as well, which returns a collection of ink points that need to be removed from a certain stroke. It then reforms the remaining ink points to compose new strokes and calculates the properties for each, including starting time and duration. The new strokes are then saved in a sequence of starting time.

6.5 Hit Testing

We call the procedure of detecting whether a pen is at or near an existing ink stroke "hit testing". Figure 6.2 and Figure 6.3 illustrate our approach for hit testing.

Each trace consists of a collection of ink points. We calculate its bounding box when a pen is writing on the canvas. We first detect whether the pen tip is contained by the bounding box of any trace. If so, we record that trace as being of interest. There may be several traces of interest. We then go through all the traces of interest

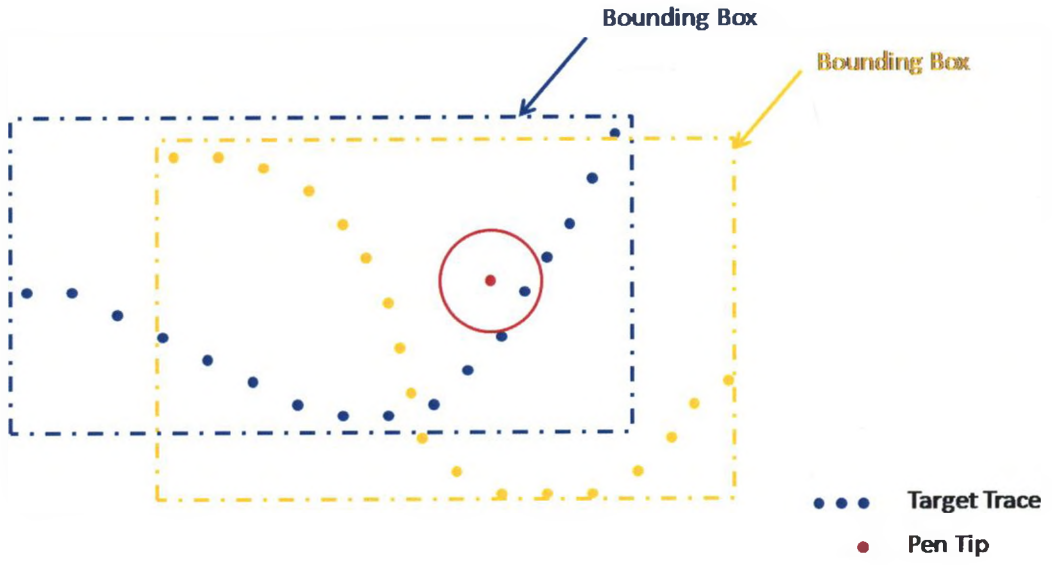


Figure 6.2: The case that the pen tip hits the trace

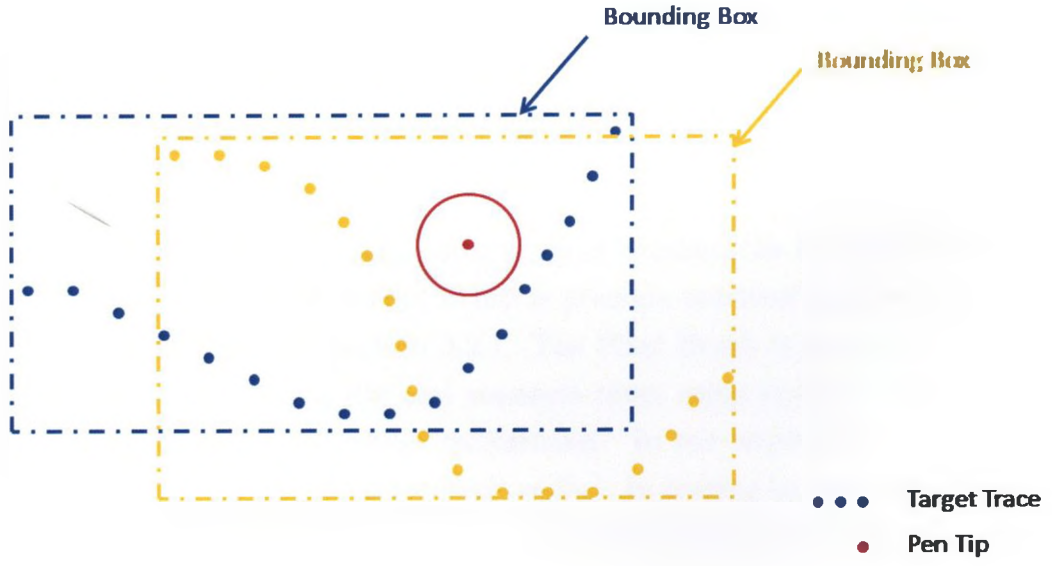


Figure 6.3: The case that the pen tip does not hit the trace

and detect whether any of their ink points are contained by the pen tip circle. The radius of the circle varies depends on the precision required. If so, we say that trace is affected.

6.6 Ink Storage

InkChat stores digital ink using InkML format. A trace is converted to an InkML stream once it is captured on the canvas. InkChat automatically saves it to the current ink session before sending to the other participants. Once the stream is received by a participant, InkChat immediately parses the stream and saves it to the current ink session. Once the conversation is finished or saving ink is requested by the interface, InkChat writes the current ink session into InkML files and saves them in the file system.

InkChat also supports loading digital ink from InkML files. It parses the files based on a InkML schema which validates the InkML files. InkChat makes use of the HP InkML Toolkit [1] to parse InkML fragments and InkML documents. InkML Toolkit is an open source library that was developed by HP Labs, India. It includes a set of tools to work with InkML data. We modified and refined this library to make it consistent with our InkChat application.

6.7 Ink Rendering Objects

InkChat renders digital ink using three types of brushes, the Round Brush, the Pixel Brush and the Eraser. The Round Brush is pressure sensitive and uses the rendering approach presented in the Section 5.2.1. The Pixel Brush is pressure insensitive. It renders each ink point as a dot and connects them using straight lines. The Eraser has the same color as the canvas background. In our implementation, we set the eraser shape to be a circle with a default radius. In erasing by point mode, the eraser moves across a stroke and may split the stroke into two or more pieces.

We classify ink strokes according to the brush types used to create them. Figure 6.4 shows the relationships between these classes.

The `Trace` class is an abstract class that defines a prototype of an ink trace. It contains several abstract methods which derived classes are required to implement. The `EraserTrace` class directly extends the `Trace` class. In addition to the attributes inherited, it defines `eraserSize` and `erasingMode` attributes to record the eraser size and the erasing mode. We use the `BasicTrace` class to represent a primitive trace that is generated by a pen. As InkChat is designed for collaboration, any of the

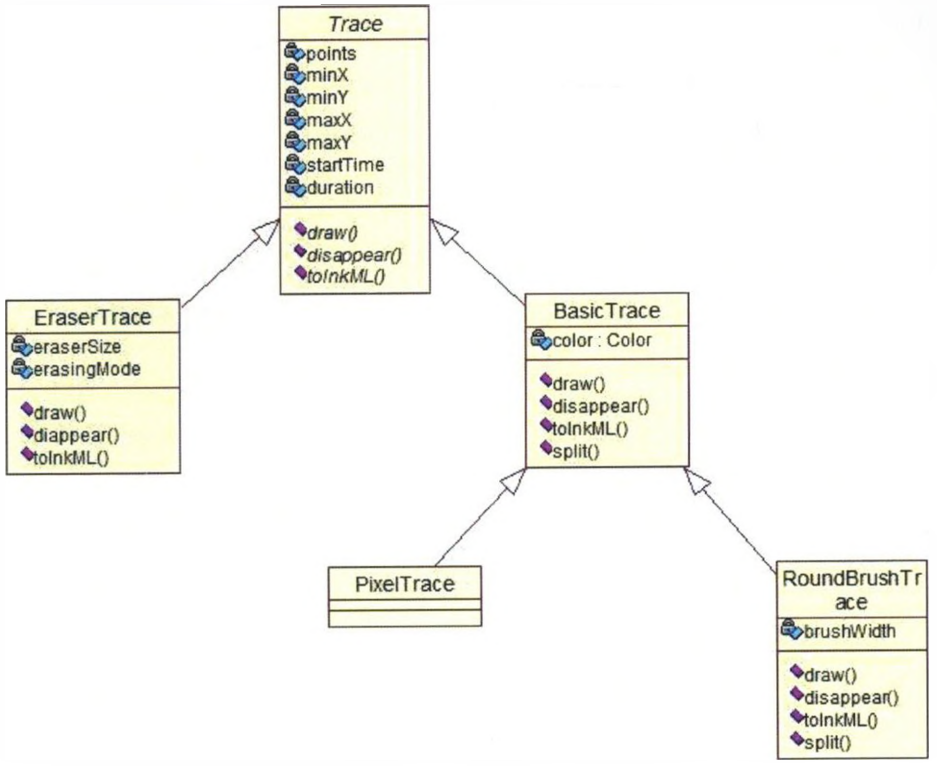


Figure 6.4: Trace class diagram

instances should contain enough information so that they can be processed without a context. The `RoundBrushTrace` class extends the `BasicTrace` class. It represents a trace rendered by a Round Brush. It uses `brushWidth` to record the physical width of the Round Brush. It also overrides the methods inherited from the `BasicTrace`. The `PixelTrace` class extends the `BasicTrace` class and represents a trace generated by a Pixel Brush.

6.8 Ink Metadata

Digital ink applications sometimes need a mechanism to represent metadata or semantics in addition to ink objects. This can be accomplished using the <annotation> element in InkML. In the InkChat implementation, we use the <annotation> element to synchronize user's page numbers as well as conversation themes.

InkChat currently recognizes 7 types of annotation. There are:

- `CLEAR`
  Clear the canvas.
- `PAGEUP`
  Go to the previous page.
- `PAGEDOWN`
  Go to the next page.
- `PAGE_NUM`
  Go to the page specified by the NUM.
- `LOADINK_FILENAME`
  Load ink from a file specified by the FILENAME.
- `THEME_FILENAME`
  Switch to the theme by loading the background image FILENAME.
- `DEFAULTHEME`
  Switch back to the default theme.

When the corresponding actions are conducted on the interface, InkChat sends these annotations to the other conversation participants. Once InkChat receives these annotations, it parses the markup language, retrieves attached information and conducts appropriate actions in the receiving client.

InkChat also provides extensibility to add more annotations. We plan to implement non-contact pointing, where the pen moves near the digitizer without touching it, so the other participants can see where it points at.

6.9 Summary

This chapter has provided the details of the InkChat architecture. We have gone through all its components and described how they are structured for portability. We have also explained how the user operations are performed on the interface, and how clients interact.

Conclusion

The primary objective of this thesis was to explore the dimension of digital ink portability, both of digital ink data and of digital ink application code. Our goal was to be able to have applications that we can "write once and run anywhere".

To support cross-platform viability, we have presented a framework that can handle the details of a variety of platforms and provide a consistent, platform-independent interface for digital ink applications. Applications that build on top of this framework can collect digital ink across all the supported platforms without knowing their details. InkChat and Calligraphy Board are two implementations we developed to demonstrate our framework's cross-platform viability. They both use the framework and work on a variety of platforms.

To support platform-independent data representation, we have evaluated several ink format standards including JOT, Unipen, ISF, SVG and InkML. We have chosen InkML as our data representation as it provides both digital ink streaming and archival support independent of platforms. Both InkChat and Calligraphy Board make use of InkML as the medium to represent digital ink. InkChat also implemented InkML context features to support ink collaboration in heterogeneous environments.

There are a number of interesting directions that can be pursued to pursue in the future. We plan to explore digital ink portability on platforms without Java such as the iPhone OS. We are also interested in issues that would arise in introducing video and text support.

Bibliography

[1] Inkml toolkit. http://inkmltk.sourceforge.net/. (valid on December 22, 2009).

[2] Inkscape. http://www.inkscape.org. (valid on December 22, 2009).

[3] Jpen library. http://sourceforge.net/apps/mediawiki/jpen/index.php?title=Main_Page. (valid on December 22, 2009).

[4] The linux wacom project. http://linuxwacom.sourceforge.net/. (valid on December 22, 2009).

[5] Ontario research centre for computer algebra. https://www.orcca.on.ca. (valid on December 22, 2009).

[6] Skype4java. https://developer.skype.com/wiki/Java_API. (valid on December 22, 2009).

[7] Uc-logic technology corp. http://www.uc-logic.com. (valid on December 22, 2009).

[8] Wacom co. ltd. http://www.wacom.com/index.html. (valid on December 22, 2009).

[9] Gregory D. Abowd, Jason Brotherton, and Janak Bhalodia. Classroom 2000: A system for capturing and accessing multimedia classroom experiences. In *CHI'98: CHI 98 conference summary on Human factors in computing systems*, pages 20–21, New York, NY, USA, 1998. ACM.

[10] Dick Bulterman, Jack Jansen, Pablo Cesar, Sjoerd Mullender, Eric Hyche, Marisa DeMeglio, Julien Quint, Hiroshi Kawamura, Daniel Weck, Xabiel Garca Paeda, David Melendi, Samuel Cruz-Lara, Marcin Hanclik, Daniel F. Zucker,

and Thierry Michel. Synchronized multimedia integration language (smil 3.0). http://www.w3.org/TR/SMIL3/. (valid on December 22, 2009).

[11] Yi-Min Chee, Max Froumentin, and Stephen Watt. Ink markup language (InkML). http://www.w3.org/TR/InkML/. (valid on December 22, 2009).

[12] Slate Corporation. Jot - a specification for an ink storage and interchange format. http://unipen.nici.kun.nl/jot.html. (valid on December 22, 2009).

[13] Michael Johnston (Editor). Extensible multimodal annotation markup language (EMMA). http://www.w3.org/TR/emma/. (valid on December 22, 2009).

[14] Scott Elrod, Richard Bruce, Rich Gold, David Goldberg, Frank Halasz, William Janssen, David Lee, Kim McCall, Elin Pedersen, Ken Pier, John Tang, and Brent Welch. Liveboard: A large interactive display supporting group meetings, presentations and remote collaboration. In *CHI'92: Proceddings of the SIGCHI conference on Human factors in computing systems*, pages 599–607, New York, NY, USA, 1992. ACM.

[15] P. R. Cohen *et al.* Quickset: multimodal interaction for distributed applications. In *MULTIMEDIA'97 Fifth ACM international conference on Multimedia*, pages 31–40. ACM, 1997.

[16] Jon Ferraiolo, Fujisawa Jun, and Dean Jackson(editors). Scalable vector graphics (svg) 1.1 specification. http://www.w3.org/TR/SVG/. (valid on December 22, 2009).

[17] Gerald Friedland, Lars Knipping, Raúl Rojas, and Ernesto Tapia. Teaching with an intelligent electronic chalkboard. In *ETP'04: Proceedings of the 2004 ACM SIGMM workshop on Effective telepresence*, pages 16–23, New York, NY, USA, 2004. ACM.

[18] The W3C Multimodal Interaction Working Group. http://www.w3.org/2002/mmi/. (valid on December 22, 2009).

[19] Isabella Guyon. Unipen 1.0 format definition. http://www.unipen.org/dataformats.html. (valid on December 22, 2009).

[20] Apple Inc. Cocoa: The objective-c programming language. http://developer.apple.com. (valid on December 22, 2009).

[21] Apple Inc. iphone sdk agreement. http://adcdownload.apple.com/iphone/iphone_sdk_3.1.2__final/ea0574_iphone_sdk.pdf. (valid on December 22, 2009).

[22] Microsoft Inc. Ink serialized format specification. http://download.microsoft.com/download/0/B/E/0BE8BDD7-E5E8-422A-ABFD-4342ED7AD886/InkSerializedFormat(ISF)Specification.pdf. (valid on December 22, 2009).

[23] Microsoft Inc. Microsoft windows xp tablet pc edition software development kit 1.7. http://www.microsoft.com/downloads/details.aspx?familyid=B46D4B83-A821-40BC-AA85-C9EE3D6E9699&displaylang=en. (valid on December 22, 2009).

[24] Microsoft Inc. Windows mobile 6 professional and standard software development kits refresh. http://www.microsoft.com/downloads/details.aspx?FamilyID=06111A3A-A651-4745-88EF-3D48091A390B&displaylang=en. (valid on December 22, 2009).

[25] Microsoft Inc. Windows presentation foundation. http://msdn.microsoft.com/en-us/library/ms754130.aspx. (valid on December 22, 2009).

[26] Palm Inc. Palm api guide. https://pdnet.palm.com/regac/pdn/PalmOSAPIGuide/index.html. (valid on December 22, 2009).

[27] Sun Microsystems Inc. Cdc runtime guide. http://java.sun.com/javame/reference/docs/cdc_runtime_guide.pdf. (valid on December 22, 2009).

[28] Sun Microsystems Inc. The connected limited device configuration (cldc) hotspot implementation. http://java.sun.com/products/cldc. (valid on December 22, 2009).

[29] LCS/Telegraphics. Wintab interface specification 1.1. http://www.wacomeng.com/devsupport/downloads/pc/wt1pt1.zip. (valid on December 22, 2009).

[30] Hai Ning, John R. Williams, Alexander H. Slocum, and Abel Sanchez. Inkboard - tablet pc enabled design oriented learning. In *CATE*, pages 154–160, 2004.

[31] Elin Ronby Pedersen, Kim McCall, Thomas P. Moran, and Frank G. Halasz. Tivoli: an electronic whiteboard for informal workgroup meetings. In *CHI'93:*

Proceddings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems, pages 391–398, New York, NY, USA, 1993. ACM.

[32] Amit Regmi. Supporting multimodal collaboration with digital ink and audio. Master's thesis, The University of Western Ontario, 2009.

[33] Steve Strassmann. Hairy brushes. In *SIGGRAPH'86*, pages 225–232, 1986.

[34] Inc. Sun Microsystems. The java native interface. `http://java.sun.com/docs/books/jni/download/jni.pdf`. (valid on December 22, 2009).

[35] Ltd. Wacom Co. Wacom enhanced graphics driver. `http://www.wacom.com/tabletpc/driver.cfm`. (valid on December 22, 2009).

[36] Stephen M. Watt. New aspects of inkml for pen-based computing. In *International Conference on Document Analysis and Recognition(ICDAR)*, pages 457–460, Curitiba, Brazil, September 2007. IEEE Computer Society.

[37] Helena T. F. Wong and Horace H. S. Ip. Virtual brush: a model-based synthesis of chinese calligraphy. In *Computers Graphics*, pages 24(1): 99–113, 2000.

[38] Xiaojie Wu. Achieving interoperability of pen computing with heterogeneous devices and digital ink formats. Master's thesis, The University of Western Ontario, 2004.

Appendix A

InkChat User Manual

This appendix provides a user manual of the InkChat application.

A.1 InkChat Features

InkChat provides the following features:

- Peer-to-Peer Conversation
- Conference Meeting
- Pressure sensitive brush
- Pressure insensitive brush
- Color Switching
- Stroke-Wise Erasing
- Point-Wise Erasing
- Theme Switching
- Voice Saving
- Ink Saving
- Ink Loading

- Page Navigation

A.2 Platform Availability

InkChat has been tested on the following operating systems:

InkChat Platform Availability

Operating System	Computer Architecture	Skype Version
Windows XP	x86	Skype 3.6.4.244
Windows Vista	x86	Skype 4.0.0.226
Windows Vista	x64	Skype 4.0.0.226
Windows 7	x86	Skype 4.0.0.226
Ubuntu 8.10	x86	Skype 2.0.0.72
Ubuntu 9.04	x86	Skype 2.0.0.72

Testing on Mac OS X is in beta.

A.3 System Requirements

InkChat requires a basic environment to run the application, including:

- Java Runtime Environment (JRE) 1.6 or above
- Skype client

On Windows platforms, one may need to install the Wacom Enhanced Graphics driver [35] to enable the pressure-sensitivity and additional side-switch functionality.

A.4 Using InkChat

To start and end a conversation:

1. Start and login to the Skype client

2. Start the InkChat application
   On Windows, run `run.bat`
   On Linux, run `bash run.sh`
3. Select one or more users from the available user list
4. Click the `Start chat` button to initiate a conversation
5. Click the `Hang up` button when done

To save the current page:

1. Go to the `File->Save Ink`
2. Specify a name and a path to save the file
3. Click the `Save` button to save

To load ink from an InkML file:

1. Go to the `File->Load Ink`
2. Specify the path of the InkML file
3. Click the `Open` button to load

To switch conversation themes:

1. Go to the `Edit->Canvas Background`
2. Select a theme from the list
3. Or click `Customize` to load a customized theme
4. Select `Default` to switch back to the default theme

To use a brush:

1. Select a brush from the brush list
2. Press the pen on the digitizer or click the left mouse button and move

To use an eraser:

1. Select an erasing type (Stroke-wise erasing and Point-wise erasing) from the eraser list
2. Keep the pen up-side-down or hold the right mouse button and cross the stroke being erased

To navigate pages:

1. Click the `Page Up` button to go to the previous page
2. Click the `Page Down` button to go to the next page or create a new page

To pick up a brush color:

1. Click the `Color` button to open an color palette
2. Pick a color and click `OK`