Techniques for Transformation and Exchange of Standardized Digital Ink

(Spline Title: Transformation and Exchange of Digital Ink)
(Thesis Format: Monograph)

by

Birendra Keshari

Graduate Program in Computer Science

Submitted in partial fulfillment
of the requirements for the degree of
Master of Science

Faculty of Graduate Studies
The University of Western Ontario
London, Ontario, Canada

THE UNIVERSITY OF WESTERN ONTARIO
FACULTY OF GRADUATE STUDIES

CERTIFICATE OF EXAMINATION

The thesis by
Birendra Keshari

entitled:
Techniques for Transformation and Exchange of Standardized Digital Ink

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

Date: April 15, 2008

Chair of Examining Board
Dr. John Barron

Abstract

Easy availability of various pen based devices, such as Tablet PCs, PDAs and smartphones, has created many opportunities to be explored in the area of pen-based computing. The diverse nature and settings of pen-based applications demands a platform-neutral representation for digital ink supporting a wide range of operations. Ink Markup Language (InkML), an XML-based language, provides such a representation that allows a variety of operations in a flexible and efficient manner.

InkML provides support for a range of pen-based application through two styles - *streaming* and *archival* - which are semantically equivalent but have different properties. In this thesis, we present solutions for doing transformations between these two styles and performing operations on them. Using streaming InkML, we present different techniques that can be used for exchanging digital ink in a heterogeneous collaborative environment. The main challenges for such a task stem from factors such as differences in channel properties of the ink sources with different capabilities and screen resolutions. We give an analysis and comparison of the various ink sharing techniques.

As two use cases, we discuss a mathematical symbol recognition server and an InkML based protocol to provide support for doing mathematics in a collaborative inking environment.

Keywords: InkML, streaming ink, archival ink, whiteboard, symbol recognition

Acknowledgments

I would like to thank my supervisor Dr. Stephen M. Watt for his support, motivation and guidance, without which this work would have not been possible. I would also like to thank all the members of the ORCCA lab for their help and support. I would especially like to thank Dr. Elena Smirnova for her helpful discussions and motivation.

I would like to thank my internship advisor, Dr. Sriganesh Madvanath, for his helpful discussions and guidance during a summer internship at HP Labs, Bangalore, India.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Thesis Subject

Pen-based input methods provide intuitive and convenient ways to interact with computer systems in various circumstances. For example, it is more convenient to input mathematical formulas, musical notation or drawings in handwritten form using a digital pen than with traditional keyboard-based input methods. There has been a lot of work done in the area of pen-based computing such as handwriting recognition, writer identification, ink messaging, electronic form filling, authentication and so on. The easy availability of various pen-based devices such as Tablet PCs, PDAs and Smartphones has created even more opportunities to be explored in this area.

In a pen-based application, the movement of a digital pen on the digitizer generates digital ink which may contain various information including x and y coordinates, time, pressure and orientation of the pen tip. The digital ink thus generated has to be represented in some format so that it can be interpreted. Moreover, for easy interoperability between the software packages it is very important that the digital ink be stored in a standard and open format. Most of the existing pen-based applications either use their own proprietary digital ink format or some of the popular, but limited, digital ink formats such as UNIPEN [Guy94] and Jot [Cor93]. Since the nature of pen-based applications is so diverse, it is also important that the digital ink format capture various requirements and provide support for different operations such as streaming, sharing digital ink, annotation and so on.

UNIPEN is very focused on handwriting recognition and therefore it may not be suitable for other types of pen-based applications that require operations such as real-time streaming of digital ink, sharing of digital ink and so on. Jot is a proprietary format and it also doesn't provide support for such operations. ISF (Ink Serialized Form) [Mic04] is a binary digital ink format developed by Microsoft that is also proprietary. Ink Markup Language (InkML) is an open standard digital ink format being developed under the W3C Multi-Modal Interaction Group and it provides various features required by different pen-based applications. Being XML based, it provides greater flexibility to application developers *e.g.* application-specific information can be added easily to digital ink files. Besides several other benefits, it provides support for streaming and sharing digital ink.

Pen-based application can be broadly categorized into streaming and archival types. InkML provides support for both the types through two forms or styles of markup called "streaming" and "archival ink". Both of these are semantically equivalent but each provides support for certain operations more directly. For example, streaming style provides better support for streaming digital ink in an incremental order with lower overhead for transmission and archival style provides better support for operations such as search, retrieval, annotation and so on. Applications that operate in both modes may benefit from conversion of digital ink from one format to another. To the best of our knowledge, no such algorithms or tools exists to date. In this thesis, we present algorithms to do such conversions. These are also used with the digital ink sharing techniques, which we discuss later, to archive the shared ink which is collected in streaming form.

Communication using digital ink is one of the most important applications of pen-based technology. Whiteboard sharing, which allows several users to interact and put digital ink on a shared canvas using pen-devices, is an ink-based communication method that can be very useful in several practical scenarios such as classroom teaching, distance education, workgroup meeting, collaborative document annotation and so on. Whiteboard sharing becomes more useful and interesting, but at the same time more challenging, when the collaborative environment is heterogeneous. The challenges stem not only from differences between operating systems and platforms but also from the differences in characteristics of the pen and the display devices, such as screen size, screen resolution, ink channels and so on. In this thesis, we present various techniques for exchanging digital ink in heterogeneous collaborative environments. We do an in depth analysis of each technique and compare them.

In order to provide support for doing mathematics using a digital pen in a collaborative inking environment, it is important to provide pen-based mathematical user interfaces. A mathematical symbol recognizer is the core component of such interfaces and is usually developed using machine learning algorithms. This requires large amount of training data and also high processing capability. Therefore, it might not be feasible to implement a full-fledged mathematical handwriting recognizer in smaller resource constrained devices such as PDAs, Smartphones and so on. We present the concept of a symbol recognition server and an InkML-based protocol to allow communication between the recognition server and the clients for tasks such as sending the recognition requests, receiving the results and so on. This when combined with the whiteboard sharing, can provide a starting infrastructure for doing mathematics in a collaborative environment.

1.2 Related Work

Digital ink formats other than InkML do not have the concepts of streaming and archival forms and no algorithms or tools to do conversion between these two forms of InkML exist till date. In this thesis, we present such conversion algorithms for the first time.

Various whiteboard sharing applications have been developed in the past. However, interoperability was not the main focus in those systems. Tivoli [PMMH93] is one of the most popular and earlier system developed by IBM for informal workgroup meeting. The *Learning Experience Project* [BCH$^+$04], an initiative of Microsoft Research's Learning Sciences and Technology group, explores collaborative learning space by developing the ConferenceXP research platform. The platform provides a whiteboard sharing feature which has been further explored by several educational institutions: InkBoard [NWSS05], developed at MIT, is a collaborative sketching application based on the ConferenceXP platform and designed for Tablet PCs which enables design teams to interact with each other by using real-time strokes. The ConferenceXP uses Microsoft's proprietary Ink Serialized Format (ISF) [Mic04] for streaming the digital ink and therefore, this limits its use to Windows environment where ISF is natively supported.

A peer-to-peer collaboration technique using InkML has been discussed in [ASM08]. Both the clients taking part in the ink communication agree upon the common channels in an initialization phase. Once initialization is done, both starts sending ink to each other in data transfer phase. This system however, does not support multi-party ink communication.

XEP-0113 [FJe03], an extension protocol to XMPP [osc99], is one of the solutions aimed at supporting interoperability across platforms and devices by using

SVG (Scalable Vector Graphics) to represent digital ink for ink messaging. However, it doesn't provide support for other channels except x and y space.The RiverInk Framework [NG07] proposes the use of a subset of InkML to represent ink for interoperability. It proposes sending digital ink in both InkML and PNG image format for interoperability between heterogeneous devices (including both pen and non-pen devices). The bulky nature of the data makes it less suitable for resource constrained devices in mobile networks.

Lenaghan and Malyan propose XPEN, an XML-based format for distributed online handwriting recognition [LM03], which is based on the UNIPEN format. In our work, we use InkML to represent the ink traces and we also describe the InkML-based protocol for communication between the recognition server and clients. We show how most of the information can be put within the ink itself using annotation XML. We also show how our technique provides support for creating a collaborative environment for doing mathematics.

1.3 Thesis Organization

Chapter 2 introduces the various concepts of InkML that are required to understand the work presented in later chapters. It discusses the semantics and use of different InkML elements with examples. It also explains the concepts of the streaming and archival form of InkML, the shared canvas concept and annotation.

Chapter 3 starts by discussing the need for streaming to archival and archival to streaming conversions. It analyses the conversion problems and then presents the conversion algorithms. It also discusses the optimization algorithm which can be applied after streaming to archival conversion.

Chapter 4 presents different InkML-based techniques for sharing digital ink in heterogeneous collaborative environments. We do in-depth analysis of each technique and also compare them.

Chapter 5 discusses a framework for doing mathematics in a collaborative environment. It presents various use-cases which should be fulfilled by an InkML-based communication protocol for this task and then it shows how the protocol fulfills them. It also describes the symbol recognition component which resides in the recognition server.

Chapter 6 provides details about the implementation of the algorithms, techniques and components discussed in Chapters 3, 4 and 5.

The thesis concludes with Chapter 7 by summarizing our work and by discussing the future directions and possibilities.

Chapter 2

InkML Concepts

2.1 Introduction

This chapter introduces some of the concepts of InkML ([CFW06], [Wat07]) that are needed to understand the work presented in later chapters.

The movement of a digital pen on the surface of a digitizer produces a sequence of points containing values that describe the trajectory of the pen as the user writes. In InkML, a continuous sequence of such values (from pen-down to pen-up or *v*ice-versa) is represented by a `<trace>` element. Figure 2.1, taken from InkML specification [CFW06], shows an example of traces that contain X and Y values. The rendered version of this digital ink (also taken from InkML specification) is shown in Figure 2.2.

The points within a trace are separated by a commas and each coordinate within a point is separated by white space(s). Each coordinate of the point in a trace can be considered as a separate "channel" that provides values over time.

```
<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>
```

Figure 2.1: Example of traces (from [CFW06])

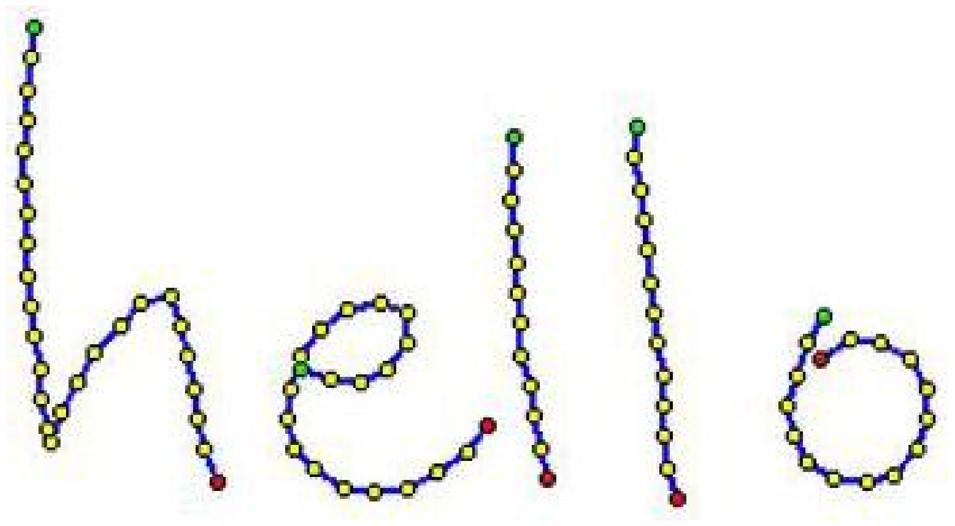


Figure 2.2: Traces in rendered form (from [CFW06])

2.2 Trace Format

In general, an ink source can have several channels (*eg.* X, Y, Force) associated with it and an ordered sequence of such channels is known as the "trace format" of the ink source. Such channels can be "regular" (always appearing) or "intermittent" (may or may not appear). In InkML, a trace format is represented by `<traceFormat>` and it can have multiple `<channel>` elements and optional `<intermittentChannel>` elements as children. The order of `<channel>` and `<intermittentChannel>` inside `<traceFormat>` defines the order of channel values of a point inside `<trace>`. For example, in a trace format definition, if channel X appears first and it is followed by channel Y then an InkML interpreter would interpret the first value of a point in a trace to be X and the second value to be Y.

2.2.1 Channel

In InkML, a channel is represented by `<channel>` element and may have several attributes that can be used to set properties of the channel. Some of these attributes which we shall use to describe our work are listed in Table 2.1.

Attribute Name	**Meaning**
name	Name of the channel (*e.g.* X, Y)
type	The data type of the point values for this channel (*e.g.* integer or boolean)
min	The lower boundary for the values of this channel
max	The upper boundary for the values of this channel

Table 2.1: Channel Attributes (from [CFW06])

The required attribute *name* specifies the interpretation of the channel in the trace data and InkML provides some reserved channel names which are listed below in Table 2.2.

An intermittent channel, represented by `<intermittentChannel>`, can enclose one or more `<channel>` elements as children. It lists those channels whose value may optionally be recorded for each sample point. For example, channel side button states(B_i) could be used as intermittent channels (value may be recorded only when the button is pressed).

2.2.2 Channel Properties

Channel properties are used to represent the properties of the channels of a trace format. In InkML, they are represented by `<channelProperties>` and it can enclose multiple `<channelProperty>`s as children. A channel element can have the attributes *channel* (channel name), *name* (name of the property of device or ink source), *value* (value of named property) and *units* (units used for value). A channel property declaration sets the value of a particular property *name* of a channel *channel* to *value.*

Channel Name	**Dimension**	**Meaning**
X	length	X coordinate.
Y	length	Y coordinate.
Z	length	Z coordinate. Height of the pen position above writing surface.
F	force	pen tip force.
S		tip switch state (touching/not touching the writing surface).
B1 ... Bn		side button states.
OTx	angle	tilt along the x-axis.
OTy	angle	tilt along the y-axis.
OTz	angle	tilt along the z-axis.
OA	angle	azimuth angle of the pen (yaw).
OE	angle	elevation angle of the pen (pitch).
OR	angle	rotation (rotation about pen axis).
C		color value (device-specific encoding).
CR, CG, CB		color values (Red/Green/Blue).
CC, CM, CY, CK		color values (Cyan/Magenta/Yellow/Black).
W	length	stroke width (orthogonal to stroke).
T	time	time (of the sample point).

Table 2.2: Reserved Channel Names in InkML (from [CFW06])

InkML specification provides a list of reserved property names, resolution being one of them. Resolution may be expressed as fractions of unit, *eg.* 1/2000 in (inches), 0.5 mm or it may be expressed in inverse units *eg.* 1500 points per inch. Figure 2.3 shows an example of channel properties.

```
<channelProperties>
    <channelProperty channel = "X" name = "resolution" value = "0.01"
    units="mm"/>
    <channelProperty channel = "Y" name = "resolution" value = "100"
    units = "1/mm"/>
</channelProperties>
```

Figure 2.3: An Example of Channel Properties

2.3 Trace and Trace Group

A trace, represented by `<trace>`, is a sequence of sampled points that represent the trajectory of the pen tip as user writes. The values of each point of a trace are recorded according to the format specified by the trace format of the ink source. InkML has the notion of the "current context" (discussed later in Section 2.5) and the trace format of a trace is either determined implicitly by the `<traceFormat>` associated with the current context. Alternatively, a trace can make an explicit reference to a context (through a `contextRef` attribute) and the trace format associated with that context becomes the current trace format. Figure 2.4 shows an example of a trace whose trace format contains channels X, Y and F.

```
...
<context>
  <traceFormat>
       <channel name="X" min="0" max="2034"/>
       <channel name="Y" min="0" max="2034"/>
       <channel name="F" min="0" max="255"/>
  </traceFormat>
</context>
<trace> 1000 1003 200, 1003 1007 198, 1004 1022 176, ... </trace>
...
```

Figure 2.4: An Example of Trace

The point values can also be recorded in terms of velocity (using ' as a prefix) or acceleration (using " as prefix). This helps to reduce the size of the data. For example, the trace data in Figure 2.4 can be equivalently represented as shown in Figure 2.5.

A group of traces can be enclosed within a `<traceGroup>` element. This can be useful for example to logically group multiple traces. A trace group can also contain multiple trace groups and this allows to group ink data in a more flexible way. An

```
...
<context>
  <traceFormat>
       <channel name="X" min="0" max="2034"/>
       <channel name="Y" min="0" max="2034"/>
       <channel name="F" min="0" max="255"/>
  </traceFormat>
</context>
<trace> 1000 13 200, '3 '4 '-2, '1 '15 '-22, ... </trace>
...
```

Figure 2.5: An Example of Trace using Velocity

example of `<traceGroup>` is shown in Figure 2.4.

```
...
<traceGroup xml:id='eqn1'>
   <traceGroup xml:id='sym1'>
      <trace>...</trace>
      <trace>...</trace>
   </traceGroup>
   <traceGroup xml:id='sym2'>
      <trace>...</trace>
      <trace>...</trace>
      <trace>...</trace>
   </traceGroup>
</traceGroup>
...
```

Figure 2.6: An Example of Trace Group

2.4 Ink Source

An ink source (`<inkSource>`) allows one to represent various specifications of a hardware device (digitizer) such as model, manufacturer, trace format, sample rate, active area and channel properties. Manufacturer and model are specified through `manufacturer` and `model` attributes of `<inkSource>` whereas trace format and channel properties are specified by making `<traceFormat>` and `<channelProperties>` direct children to `<inkSource>`. An example of ink source is shown in Figure 2.7.

```
<inkSource xml:id = "tabletxyz"
   manufacturer = "xyz.com"
   model = "XYZTab 2000 USB"
   specificationRef="http://www.xyz.com/xyz-tab/2000usb.html">

   <traceFormat href="#tf1"/>

   <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="5500" units="1/in"/>
      <resolution channel="Y" value="5500" units="1/in"/>
      <resolution channel="F" value="522" units="dev"/>
   </channelProperties>
</inkSource>
```

Figure 2.7: An Example of InkSource

2.5 Context

In InkML, various details about the context in which the ink is recorded is expressed through `<context>` element. A context has different aspects: `<traceFormat>`, `<inkSource>`, `<brush>`, `<timestamp>`, `<canvas>` and `<canvasTransform>`. These aspects can appear as a direct child to `<context>` or these can be referenced through corresponding referencing attributes of `<context>` (*eg.* `brushRef`).

InkML has the concept of "current context" which is the context at any particular instant of time. Current context can change due to an event such as a change in brush.

A new `<context>` element can alter the current context by overriding the current values of aspects with the new ones. To allow reusability, it is allowed to reference a previously defined context from a new `<context>` through `contextRef` attribute. In such cases, all the aspects from previous context are inherited and the new aspects overrides the inherited aspects.

```
<context xml:id="context1" brushRef="#penA" traceFormatRef="#format1"
  canvasRef="#can1"/>
...
<context xml:id="context2" contextRef="#context1" brushRef="#penB"/>
```

Figure 2.8: An Example Showing Context Reference

For example, in the InkML snippet shown in Figure 2.8, `context2` inherits all the aspects of `context1` through `contextRef`. However, it points to a separate brush `penB` and therefore `penB` overrides `penA`. This is similar to the concepts of overriding and inheritance in object oriented paradigm. This concept has been illustrated in Figure 2.9.

2.6 Shared Canvas

InkML provides built-in support for applications that require sharing of digital ink coming from different ink sources by means of the `<canvas>` and `<canvasTransform>` elements, both aspects of current context. A canvas has an associated trace format specified either as a child element or referred to by its `traceFormatRef` attribute.

In a collaborative environment, all the ink sources should agree upon a common canvas *i.e.* current context of each sender (ink source) should point to the shared canvas. The `<canvasTransform>` element can specify two child elements known as the *forward canvas transform* and the *inverse canvas transform*. In the whiteboard

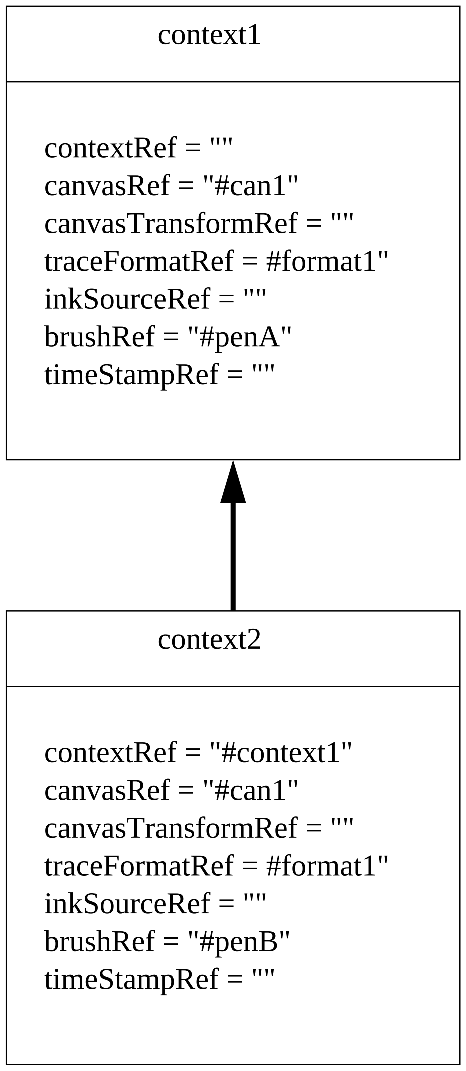


Figure 2.9: Inheritance and Overriding of Context through Reference ([KW07b])

sharing scenario, a *forward canvas transform* contains the mapping information required to map the ink data from an ink source trace format to the canvas trace format, and an *inverse canvas transform* contains information about the inverse mapping. If the *inverse canvas transform* is not specified and the `invertible` attribute of `<canvasTransform>` is `true`, it implies that the forward mapping is invertible i.e. the *inverse canvas transform* can be determined automatically. Each ink source participating in the ink communication can establish its current canvas transform by sending out a `<canvasTransform>` element. Figure 2.10 shows an example of shared canvas (CT denotes forward canvas transform and iCT denotes the inverse).

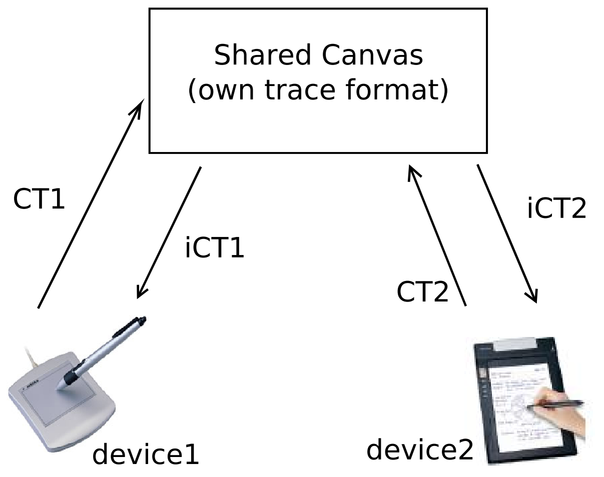


Figure 2.10: An Example of Shared Canvas (from [KMA+08])

2.7 Annotation

InkML allows arbitrary textual and XML-based annotation of digital ink in an easy way and these allow to attach semantics to the digital ink. Simple textual descriptions in the ink markup can be inserted using `<annotation>` elements. The category of textual annotation is expressed by setting attribute `type` to the predefined values in InkML or it can also be application defined. An example of textual annotation is shown in Figure 2.11.

XML-based annotation can be done using `<annotationXML>` elements. This provides greater flexibility as it allows arbitrary XML (*eg.* MathML, RDF, XHTML) to annotate the digital ink to provide richer semantics. Figure 2.12 illustrates an example where XML-based annotation is used to express the bounding box of a trace using XML.

```
<traceGroup xml:id="tg1">
   <annotation type="truth">Hello</annotation>
   <trace> ... </trace>
   ...
</traceGroup>
```

Figure 2.11: An Example of Textual Annotation

```
<traceGroup xml:id="tg1">
   <annotationXML>
      <boundingBox>
         <x>100</x>
         <y>200</y>
         <width>1200</width>
         <height>3200</height>
      </boundingBox>
   </annotationXML>
   <trace> ... </trace> ...
</traceGroup>
```

Figure 2.12: An Example of XML-based Annotation

2.8 Definitions

Reusable contents such as brush, canvas, canvas transform, context, ink source , trace format, trace and others can be defined in the `<definitions>` section of InkML. This section usually appears at the beginning of the ink markup and the elements defined within it have a unique id so that they can be later referenced from outside. Such definitions do not alter or establish any context and these are only for the purpose of reuse. This may not be used heavily in streaming applications as the contextual elements and traces may not be known in advance but it can be very useful for archival applications. We discuss the two styles of InkML (streaming and archival) and use of definition in archival form in following Section 2.9.

2.9 Streaming and Archival Styles

Pen-based applications can be broadly classified into streaming and archival types. A streaming application sends/receives digital ink to/from another streaming application. An example of such applications is shown in Figure 2.13. On the other hand, an archival application deals with storage of digital ink for future retrieval and processing. Figure 2.14 shows an example of such application.

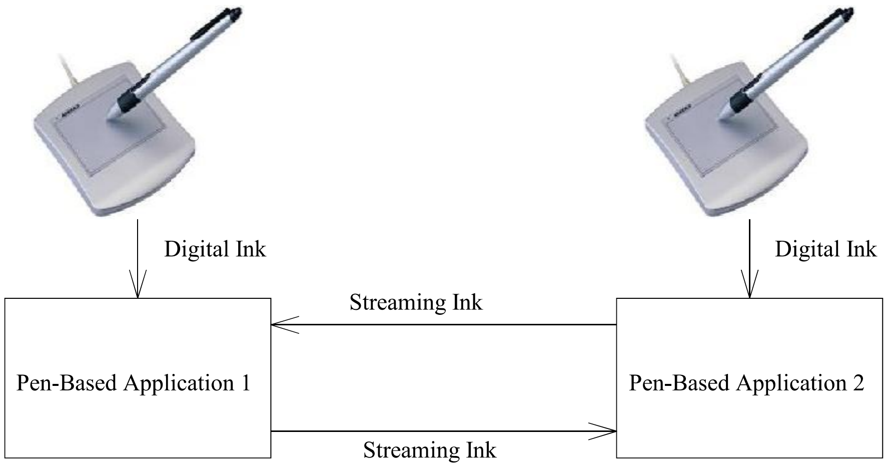


Figure 2.13: Streaming Applications (from [KW07b])

InkML provides support for both types of applications through the notion of two styles of ink markup: *streaming* and *archival*. These are semantically equivalently but each provides direct support for certain operations. For example archival style provides more direct support for operations such as search, retrieval, annotation and so on and streaming style provides support for transmission of digital ink in an incremental order resulting in lower overhead for transmission.

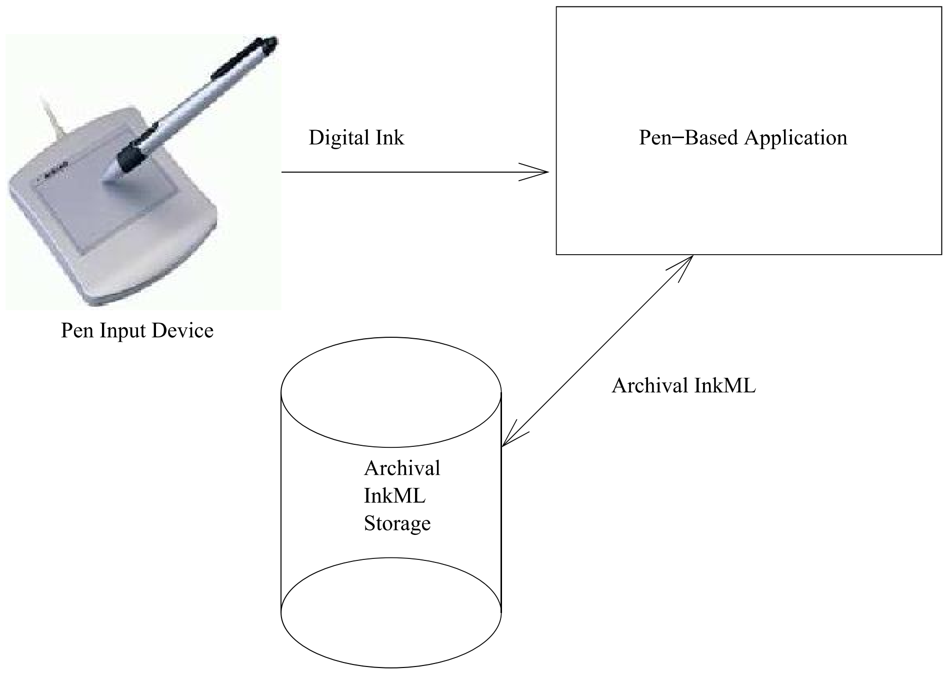


Figure 2.14: Archival Application (from [CFW06])

2.9.1 Streaming Style

Streaming style can be used in a scenario where an application transmits the digital ink to another application (or sometimes to itself). This style is based on the concept of *current context* which is the context associated with the ink being generated at a particular instance of time. The current context has various aspects such as `<inkSource>`, `<traceFormat>`, `<brush>`, `<timestamp>`, `<canvas>` and `<canvasTransform>`. Initially, all of these contextual element has a default value (*Default Context*) and these values can be altered by sending a `<context>` element. An event that occurs in the sender application (*eg.* change in brush) can be directly mapped to one of the relevant contextual elements. Ink data may be sent after its context has been established. Thus, stream of ink data interspersed with contextual elements may be transmitted in an incremental order.

With this model, each receiver can easily maintain the current context of the sender. Whenever a new contextual element is seen, its values suitably modify (or override, as appropriate) the old values. Contextual elements are sent only when there is a change in context and this helps to reduce the data on the wire. For instance, if the current brush's color is red and a red trace is scribbled then it is sufficient to only send the trace and not the brush color information, as the receiver can know the sender's current brush color from the current context it maintains. This idea has been illustrated in Figure 2.15 (trace "t4" is sent without any brush information).

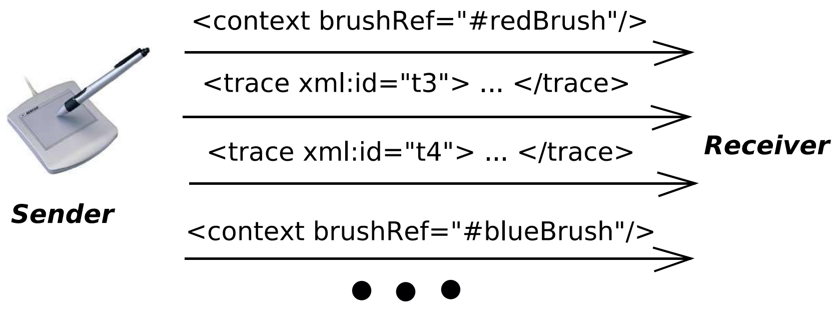

<context brushRef="#redBrush"/>
<trace xml:id="t3"> ... </trace>
<trace xml:id="t4"> ... </trace>
<context brushRef="#blueBrush"/>
Sender
Receiver

Figure 2.15: An example of Streaming Style InkML (from [KMA+08])

2.9.2 Archival Style

In archival style, all the contextual elements such as `<traceFormat>`, `<inkSource>`, `<brush>`, `<timestamp>`, `<canvas>`, `<canvasTransform>` and `<context>` are defined within `<definitions>` section. Traces and trace groups make direct references to these contextual elements from outside the `<definitions>` to establish their context. Such structure helps to know the context information directly from the `<definitions>` alone. Thus, it directly supports the search and retrieval operations by saving time which is important for archival applications. An example of archival style is shown in Figure 2.16.

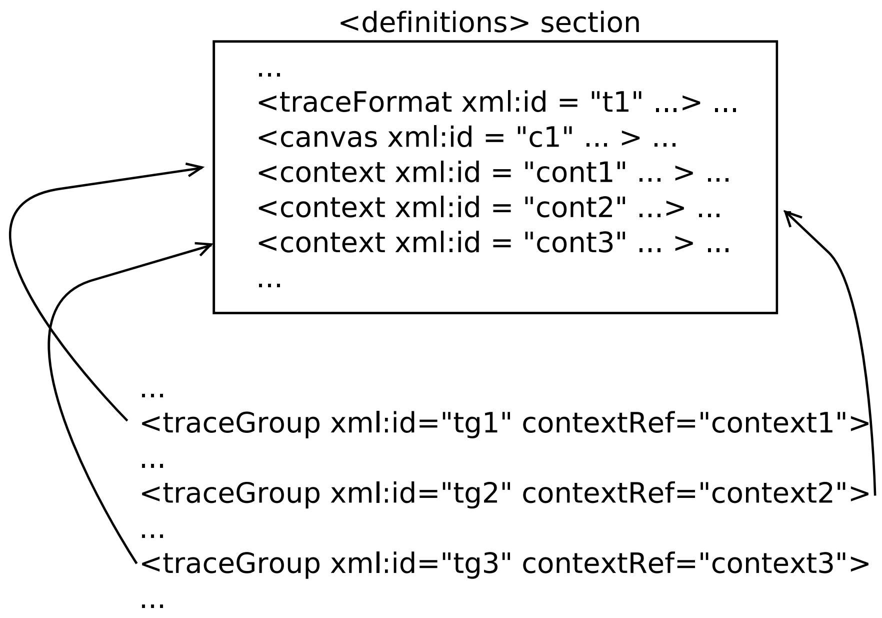


Figure 2.16: An Example of Archival Style InkML

Chapter 3

Streaming-Archival Conversion Techniques

In this chapter we highlight the importance of streaming-archival conversion, discuss the nature of the conversion problem and then provide algorithms for doing these conversions. This chapter is based on our paper [KW07b] published in the proceedings of the International Conference on Document Analysis and Recognition, 2007.

3.1 Need for Conversion

Since streaming style InkML provides direct support for streaming digital ink, pen-based applications that require ink to be sent and received from other applications as it is generated will use the streaming style of InkML to exchange digital ink. The feature to save the digital ink collected from various ink sources for future retrieval and processing can increase the usefulness of such pen-based applications. For example, saving an ink chat session or a collaborative design session can be very useful for later use. The structure of the ink markup in the archived form should facilitate operations such as search, retrieval, annotation and so on. Streaming style doesn't provide support for such operations. Therefore, it may not be efficient to save the digital ink in the same form as it was generated and received. Archival style provides better support for such operations. Hence, conversion from streaming to archival style is very important in such situation.

The ability to stream the digital ink archived in the past can be another important feature of a pen-based application. The streaming can occur within the same application or between two separate applications on different machines/devices that are able to communicate with each other. For example, it can be very useful if an ink chat application can play an ink chat session from an archived form that was saved in the past. Similarly, it can also be useful if an application can stream an ink conversation in archival form saved in the past to another application on different machine. Since streaming style provides mechanism to directly map events to contextual elements, it is more suitable for the ink interpreter in the receiver application. Therefore, conversion from archival to streaming form can be very useful in such scenarios.

3.2 Conversion Problem

Both streaming and archival styles of InkML carry exactly the same information. However, the organization of the InkML primitives within the two structures are different. Hence, they impose different requirements for markup processor and generators and give different computational complexities for certain operations. In a streaming form, contextual elements appear in-line to the ink data and contextual elements can make references to previously appearing contextual elements resulting in a very complex chain of references. In order to find the current context at a particular point in the markup, the previously occurring chain of references has to be resolved.

In archival form, all the contextual markups are enclosed within `<definitions>` section. Each trace/trace group can make explicit reference to the contextual elements defined in this section to establish its context. Such structure allows the context of a trace/trace group to be known directly from `<definitions>` section.

The conversion problem is to transform a streaming structure to archival structure and *vice versa* without any loss of information in efficient ways (Figure 3.1). In an abstract way, the transformation problem can be seen as rearrangement and update of the InkML primitives by preserving the semantics.

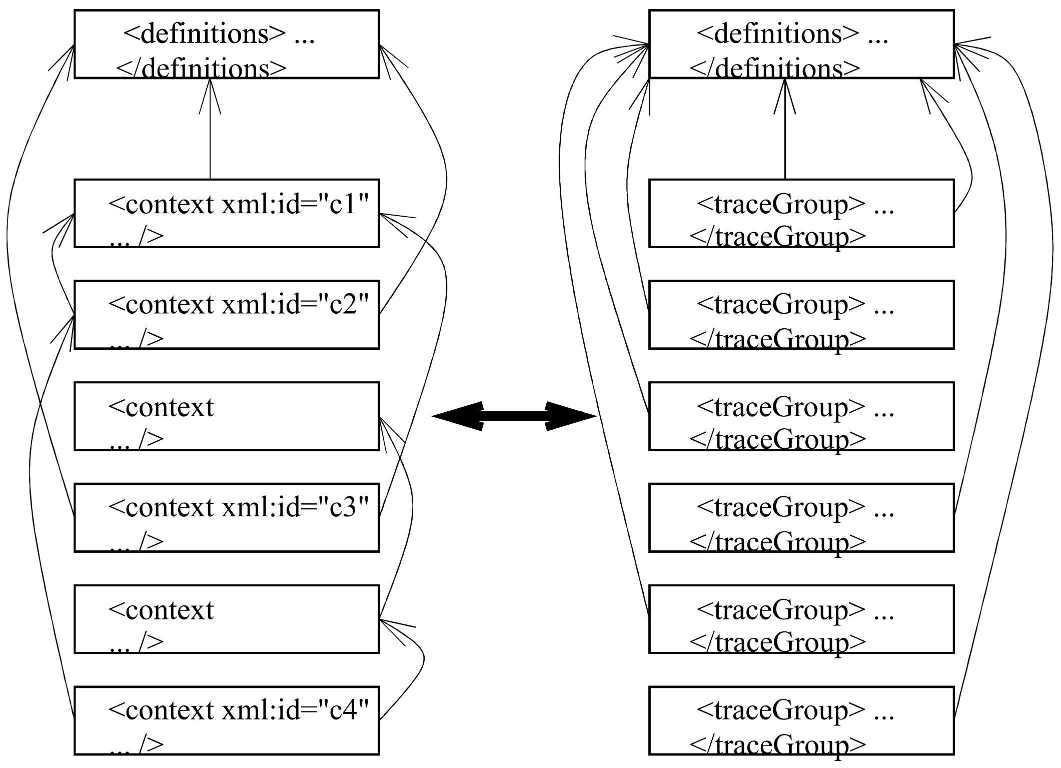


Figure 3.1: Streaming (left) and Archival (right) Ink Markup Structure Conversion (from [KW07b])

3.3 Streaming to Archival Conversion

The central idea behind streaming to archival conversion is to resolve the context of each trace in the markup. These contextual elements are then put in the `<definitions>` section and all the traces that share the same context are grouped using `<traceGroup>`. Each `<traceGroup>` points to its context defined in `<definitions>` section through its `contextRef` attribute.

A detailed overview of the algorithm is presented in activity diagram shown in Figure 3.2. An XML parser is used to parse the ink markup and obtain a DOM (Document Object Model) [Gro] tree. All the elements defined within a `<definitions>` section are saved for later use. We will call it *oldDef*. The XML elements in the DOM tree is scanned sequentially and in order to keep track of the different aspects of the current context of each trace in the markup a data structure with fields *id*, *brush*, *traceFormat*, *inkSource*, *canvas*, *canvasTransform* and *timestamp* is maintained.

Whenever a `<context>` element is encountered, a test is made to check whether the changes due to this element results in the same context as current context or different. If the changes results in the same context as current context then it is simply ignored. Otherwise, the current value of *id* is saved in a temporary variable (let's call it *tempID*) and the aspects of the current context are updated. The field *id* is set to the value of property `id` of the `<context>` if present, otherwise an id of the form "contextgN" is automatically generated by the algorithm, where 'N' is an integer that starts from 1 and gets incremented by one each time a new id is generated. A new context element with its id set to *id* and `contextRef` set to *tempID* (if `<context>` doesn't have this attribute already set) is created and put in the `<definitions>` section. Contextual elements (*eg.* `<brush>`, `<traceFormat>` and so on) that are enclosed within `<context>` are also put in the definition. All traces

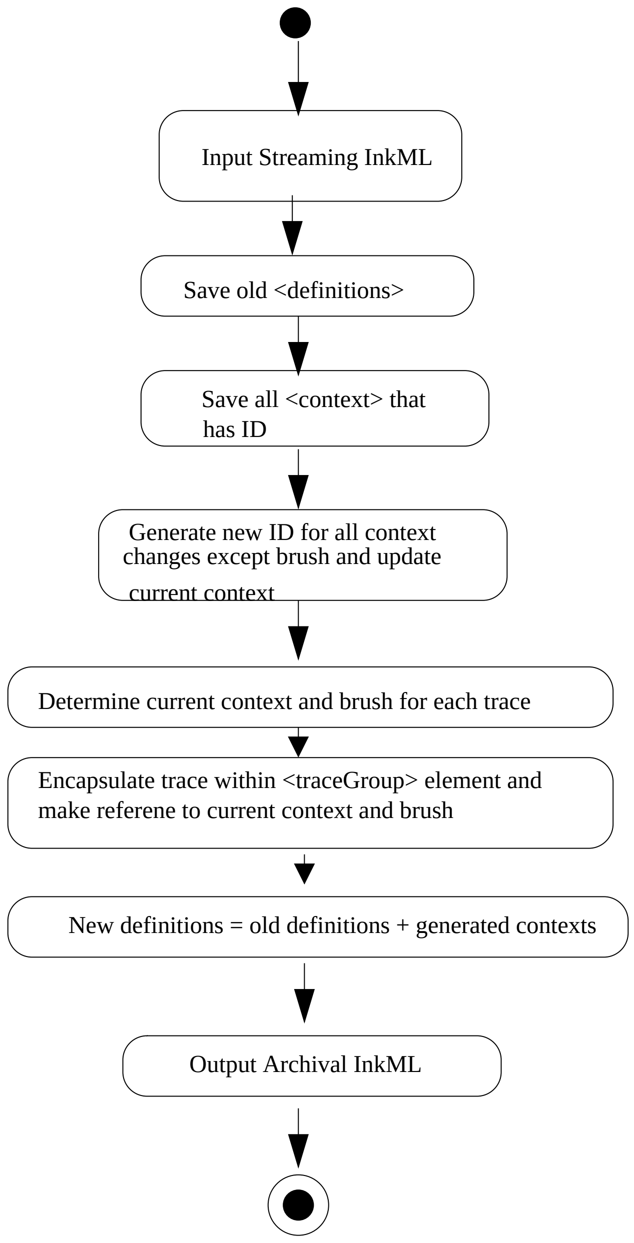

Input Streaming InkML
Save old <definitions>
Save all <context> that has ID
Generate new ID for all context changes except brush and update current context
Determine current context and brush for each trace
Encapsulate trace within <traceGroup> element and make referene to current context and brush
New definitions = old definitions + generated contexts
Output Archival InkML

Figure 3.2: Streaming to Archival Translation (from [KW07b])

sharing same context are grouped together by enclosing them within a `<traceGroup>` and its `contextRef` attribute points to the newly created context in definition section. Since `<traceGroup>` has an attribute `brushRef`, if the context change is only the brush change then no new context is put in the definition section. The `brushRef` property of `<traceGroup>` points to the appropriate brush and `contextRef` attribute points to the id of the current context. Finally, the *oldDef* is merged with the existing definitions in the `<definitions>` section.

Figure 3.3 shows an example of streaming markup input to the conversion algorithm and Figure 3.4 shows its transformed archival form.

3.4 Optimization

As the context changes over a period of time, the context at a particular point of time can happen to be the same as the context at any previous time. Therefore, during transformation of InkML structure from streaming to archival form, there are possibilities of duplicate context ids being generated. Although this doesn't have any effect on the semantics, it does increase the size of the archival ink. A situation where duplicate context ids can be generated is shown in Figure 3.5. The conversion algorithm (previous section) will generate an id for each context change. The two contexts *C1* and *C4*, shown in the Figure 3.5, happen to be the same (since both have 't1' as trace format and 'b1' as brush). Therefore, *C4* should be removed from the definition section and all the references to *C4* should be updated to *C1*. This can help to reduce the size of the markup.

In a collaborative inking scenario, if digital ink is collected from different ink sources then each ink source may change the context to its current context before sending its ink data even if its context is not changed, so that the receiver knows

```
<ink>
   <definitions>
      <brush xml:id='penA'/>
      <canvas xml:id='can1'>...</canvas>
      <canvas xml:id='can2'>...</canvas>
      <canvasTransform xml:id='trans1'>...</canvasTransform>
      <traceFormat xml:id='format1'>...</traceFormat>
   </definitions>
   ...
   <context xml:id='c1' canvasRef= '#can1'
       canvasTransformRef='#trans1' traceFormatRef='#format1'/>
   <trace>...</trace>

   <context><traceFormat xml:id='format2'>...</traceFormat></context>
   <trace>...</trace>

   <context canvasRef='#can2'/>
   <trace>...</trace>

   <context canvasRef='#can1'/>
   <trace>...</trace>

   <context brushRef='#penA'/>
   <trace>...</trace>

   <context traceFormatRef='#format1'/>
   <trace>...</trace>
   <trace>...</trace>

</ink>
```

Figure 3.3: A Streaming Style Markup (Input)

```
<ink>
   <definitions>
      <brush xml:id='penA'/>
      <canvas xml:id='can1'>...</canvas>
      <canvas xml:id='can2'>...</canvas>
      <canvasTransform xml:id='trans1'>...</canvasTransform>
      <traceFormat xml:id='format1'>...</traceFormat>
      <traceFormat xml:id='format2'>...</traceFormat>
      <context xml:id='c1' canvasRef= '#can1'
         canvasTransformRef='#trans1' traceFormatRef='#format1'/>
      <context xml:id='contextg1' contextRef='#c1'
         traceFormatRef="#format2"/>
      <context xml:id='contextg2' contextRef='#contextg1'
         canvasRef='#can2'/>
      <context xml:id='contextg3' contextRef='#contextg2'
         canvasRef='#can1'/>
      <context xml:id='contextg4' contextRef='#contextg3'
         traceFormatRef='#format1'/>
   </definitions>
   ...
   <traceGroup contextRef='#c1'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg1'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg2'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg3'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg3' brushRef='#penA'>
      <trace>...</trace>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg4'>
      <trace>...</trace>
   </traceGroup>
</ink>
```

Figure 3.4: An Archival Style Markup (Output)

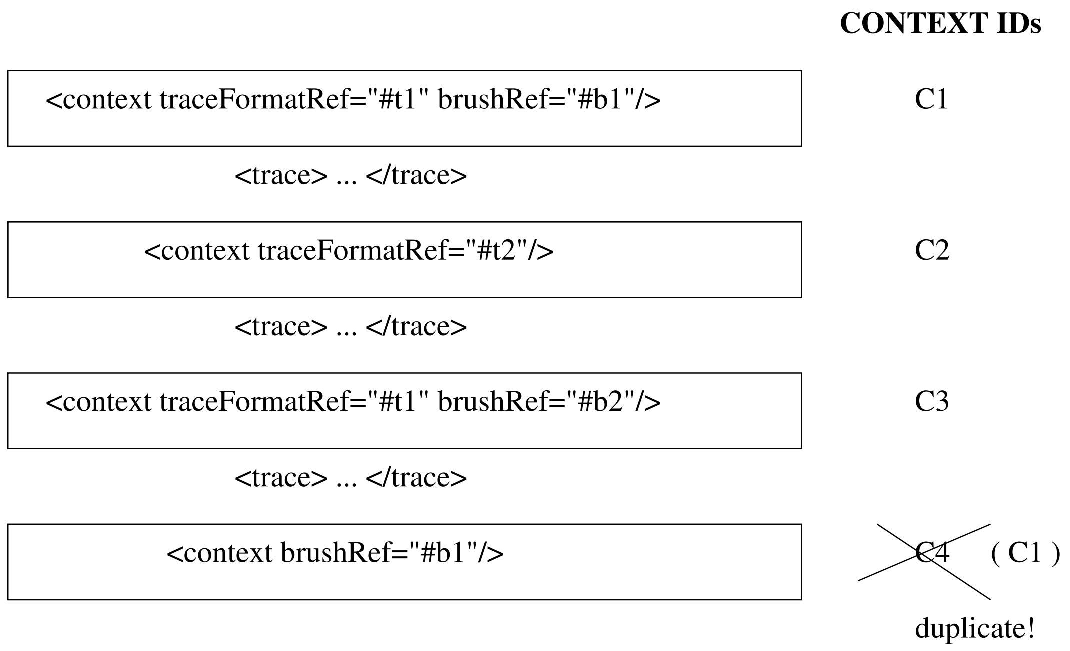

CONTEXT IDs
<context traceFormatRef="#t1" brushRef="#b1"/>
C1
<trace> ... </trace>
<context traceFormatRef="#t2"/>
C2
<trace> ... </trace>
<context traceFormatRef="#t1" brushRef="#b2"/>
C3
<trace> ... </trace>
<context brushRef="#b1"/>
C4
(C1)
duplicate!

Figure 3.5: An Example of Duplicate Context (from [KW07b])

where the ink is coming from. In such situation, the chances of getting duplicate context ids generated can be high and hence, optimization can be very beneficial.

The output of the conversion algorithm presented in previous section can be further optimized using the technique shown in Figure 3.6. The optimization algorithm basically scans all the contexts in the `<definitions>` section and detects those that are duplicates. Two vectors: *vect* and *duplicateVect* are used to store the original and duplicate vectors respectively and these are initially empty. A data structure called *currentContext* with fields *traceFormat*, *inkSource*, *brush*, *canvas*, *canvasTransform*, *timestamp* and *original* is maintained to keep track of the current context as `<context>` elements are being scanned. The field *original* is used to store the reference to the original context when the context itself is a duplicate. Whenever a new context is found, a test is made to check whether it is already present in *vect* vector or not. Comparison is done by comparing the absolute values of each aspects of the two contexts. If the test fails then it is appended to *vect*. Otherwise, the `original` field of current context is set to the id of the context in *vect* and the context is appended to *duplicateVect*. The process continues until all the `<context>`s have been processed. In the end, all the duplicate contexts and their corresponding originals is found in *duplicateVect*. These are removed from the definitions and all the references made to these contexts from outside of definitions are replaced by the id of the original context.

In Figure 3.4, a closer observation at the result obtained by applying streaming to archival technique reveals that the context *contextg4* generated during the conversion is a duplicate of context *c1*. The application of optimization algorithm on this output produces the optimized version which is shown in Figure 3.7. In this output, the duplicate context has been removed and all the references to it has been replaced by the id of original context (*c1*).

3.5 An Alternate Approach to Streaming to Archival Conversion

The streaming to archival conversion algorithm described in the Section 3.3 preserves the chain of context references within `<definitions>` section in the archival form. This might be useful for some applications to know how the contexts were changed. An alternative approach to streaming to archival conversion is to resolve all the aspects of the context of each trace group and make direct references to the aspects (brush, trace format, canvas, canvas transform, time stamp and ink source) and not to other contexts, thus, removing the chain of references. Although this removes the information about how different contexts were inherited, this approach

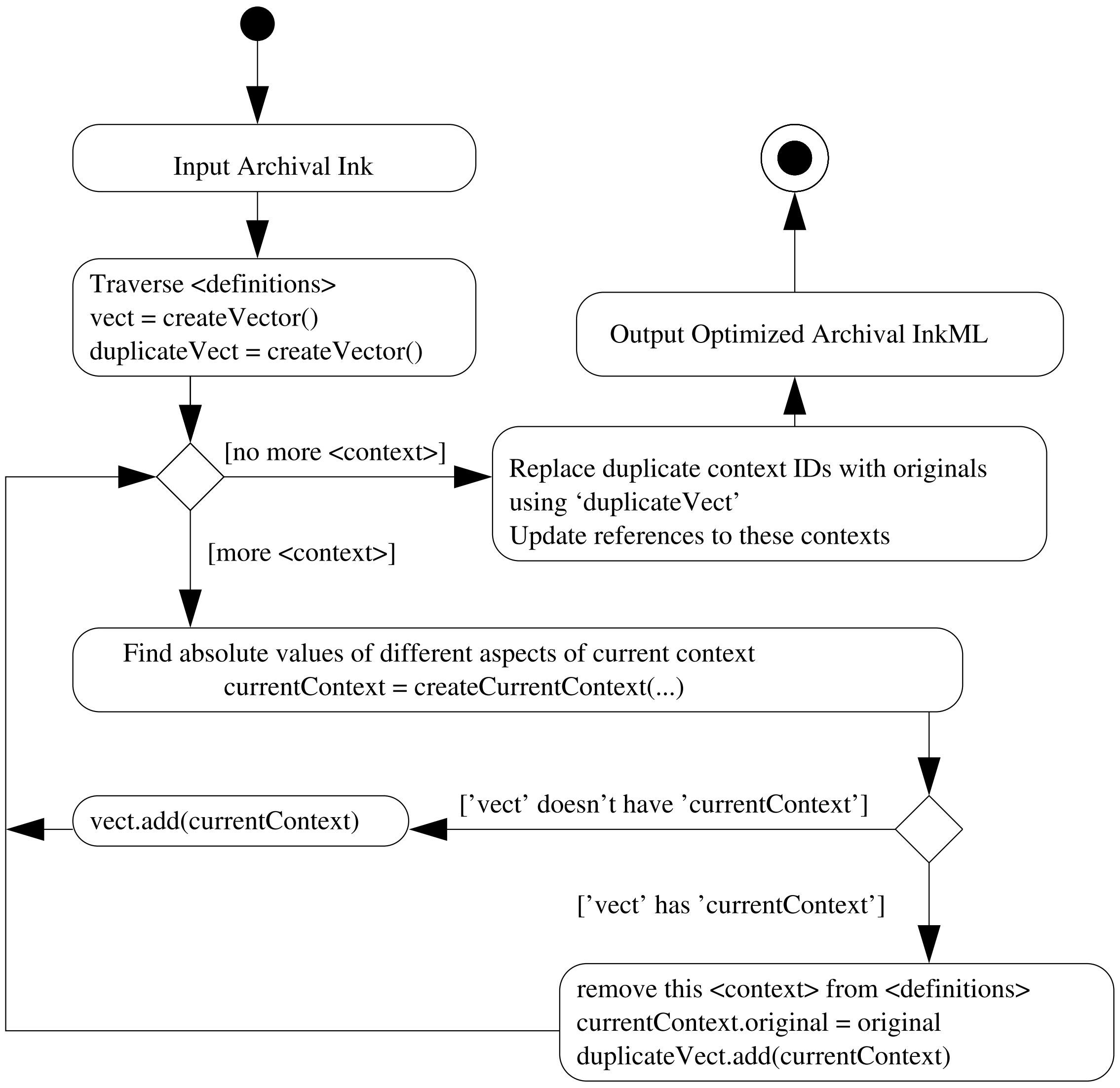

Input Archival Ink
Traverse <definitions>
vect = createVector()
duplicateVect = createVector()
[no more <context>]
[more <context>]
Replace duplicate context IDs with originals
using ‘duplicateVect’
Update references to these contexts
Output Optimized Archival InkML
Find absolute values of different aspects of current context
currentContext = createCurrentContext(...)
[’vect’ doesn’t have ’currentContext’]
vect.add(currentContext)
[’vect’ has ’currentContext’]
remove this <context> from <definitions>
currentContext.original = original
duplicateVect.add(currentContext)

Figure 3.6: Optimization of Archival Ink (from [KW07b])

```
<ink>
   <definitions>
      <brush xml:id='penA'/>
      <canvas xml:id='can1'>...</canvas>
      <canvas xml:id='can2'>...</canvas>
      <canvasTransform xml:id='trans1'>...</canvasTransform>
      <traceFormat xml:id='format1'>...</traceFormat>
      <traceFormat xml:id='format2'>...</traceFormat>
      <context xml:id='c1' canvasRef= '#can1'
         canvasTransformRef='#trans1' traceFormatRef='#format1'/>
      <context xml:id='contextg1' contextRef='#c1'
         traceFormatRef="#format2"/>
      <context xml:id='contextg2' contextRef='#contextg1'
         canvasRef='#can2'/>
      <context xml:id='contextg3' contextRef='#contextg2'
         canvasRef='#can1'/>
   </definitions>
   ...
   <traceGroup contextRef='#c1'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg1'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg2'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg3'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg3' brushRef='#penA'>
      <trace>...</trace>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#c1'>
      <trace>...</trace>
   </traceGroup>
</ink>
```

Figure 3.7: Optimized Archival InkML

may be more beneficial when speed is of higher priority and the application doesn't need the information about how context changes happened. This may save time as the different aspects of the context of a trace group can be known directly.

The algorithm to do such conversion is very much similar to the earlier streaming to archival conversion algorithm (Section 3.3). A new `<context>` generated in the earlier algorithm would not make a reference to another context, instead, it would make references to different aspects defined in the `<definitions>` using the corresponding referencing attributes such as `traceFormatRef`, `canvasRef` and so on. These aspects can be determined from different fields that are used to keep track of different aspects of the current context. A Result of applying such technique to the streaming InkML shown in Figure 3.3 is shown in Figure 3.8.

The same optimization algorithm discussed in Section 3.4 can be applied to the result shown in Figure 3.8 and this produces the output shown in Figure 3.9.

3.6 Archival to Streaming Conversion

For the conversion from archival to streaming form, it is required to capture the events (*eg.* change in brush) and reflect them in the markup in-line to the trace outside the `<definitions>`. A technique to do such conversion is shown in Figure 3.10.

The first step in the conversion process is to remove all the `<context>` definitions in the `<definitions>` section from the DOM tree and save it temporarily. A data structure similar to the one used in previous techniques is used to keep track of the current context. This data structure has these fields: *id*, *brush*, *traceFormat*, *inkSource*, *canvas*, *canvasTransform* and *timestamp*. Each `<traceGroup>` element outside the `<definitions>` is processed one by one. The `contextRef` attribute of a `<traceGroup>` is used to determine its context and then it is compared with the current context.

```
<ink>
   <definitions>
      <brush xml:id='penA'/>
      <canvas xml:id='can1'>...</canvas>
      <canvas xml:id='can2'>...</canvas>
      <canvasTransform xml:id='trans1'>...</canvasTransform>
      <traceFormat xml:id='format1'>...</traceFormat>
      <traceFormat xml:id='format2'>...</traceFormat>
      <context xml:id='c1' canvasRef= '#can1'
         canvasTransformRef='#trans1' traceFormatRef='#format1'/>
      <context xml:id='contextg1' canvasRef='#can1'
         canvasTransformRef='#trans1' traceFormatRef="#format2"/>
      <context xml:id='contextg2' canvasRef='#can2'
         canvasTransformRef='#trans1' traceFormatRef="#format2"/>
      <context xml:id='contextg3' canvasRef='#can1'
         canvasTransformRef='#trans1' traceFormatRef="#format2"/>
      <context xml:id='contextg4' canvasRef= '#can1'
         canvasTransformRef='#trans1' traceFormatRef='#format1'/>
   </definitions>
   ...
   <traceGroup contextRef='#c1'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg1'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg2'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg3'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg3' brushRef='#penA'>
      <trace>...</trace>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg4'>
      <trace>...</trace>
   </traceGroup>
</ink>
```

Figure 3.8: Result of Applying Alternate Approach

```
<ink>
   <definitions>
      <brush xml:id='penA'/>
      <canvas xml:id='can1'>...</canvas>
      <canvas xml:id='can2'>...</canvas>
      <canvasTransform xml:id='trans1'>...</canvasTransform>
      <traceFormat xml:id='format1'>...</traceFormat>
      <traceFormat xml:id='format2'>...</traceFormat>
      <context xml:id='c1' canvasRef= '#can1'
         canvasTransformRef='#trans1' traceFormatRef='#format1'/>
      <context xml:id='contextg1' canvasRef='#can1'
         canvasTransformRef='#trans1' traceFormatRef="#format2"/>
      <context xml:id='contextg2' canvasRef='#can2'
         canvasTransformRef='#trans1' traceFormatRef="#format2"/>
      <context xml:id='contextg3' canvasRef='#can1'
         canvasTransformRef='#trans1' traceFormatRef="#format2"/>
   </definitions>
   ...
   <traceGroup contextRef='#c1'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg1'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg2'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg3'>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#contextg3' brushRef='#penA'>
      <trace>...</trace>
      <trace>...</trace>
   </traceGroup>
   <traceGroup contextRef='#c1'>
      <trace>...</trace>
   </traceGroup>
</ink>
```

Figure 3.9: Result of Applying Alternate Approach and Optimization

No action is taken if both are same. Otherwise, their difference is found using the following formula:

$$\begin{aligned} D &= C1 - C2 \\ D_i &= \begin{cases} Ci & \text{if } C1_i \neq C2_i \\ empty & \text{Otherwise} \end{cases} \end{aligned}$$

where D is the difference between two contexts $C1$ and $C2$ and X_i denotes the i^{th} aspect of context X. The difference between the contexts is used to reflect the change in event using a `<context>` element and it is put in-line to the `<trace>` or `<traceGroup>` element. The current context is updated to the context of the `<traceGroup>`. Traces that are enclosed within a `<traceGroup>` with InkML annotations in the archival ink are encapsulated within the same `<traceGroup>` with `contextRef` attribute removed. In the absence of annotation, `<traceGroup>`s are removed. Stream of digital ink is output after each `<traceGroup>` is processed. Thus, streaming InkML is generated iteratively (digital ink sharing the same context is generated in each iteration).

Figure 3.11 shows the result of applying archival to streaming technique on the archival sample shown in Figure 3.4.

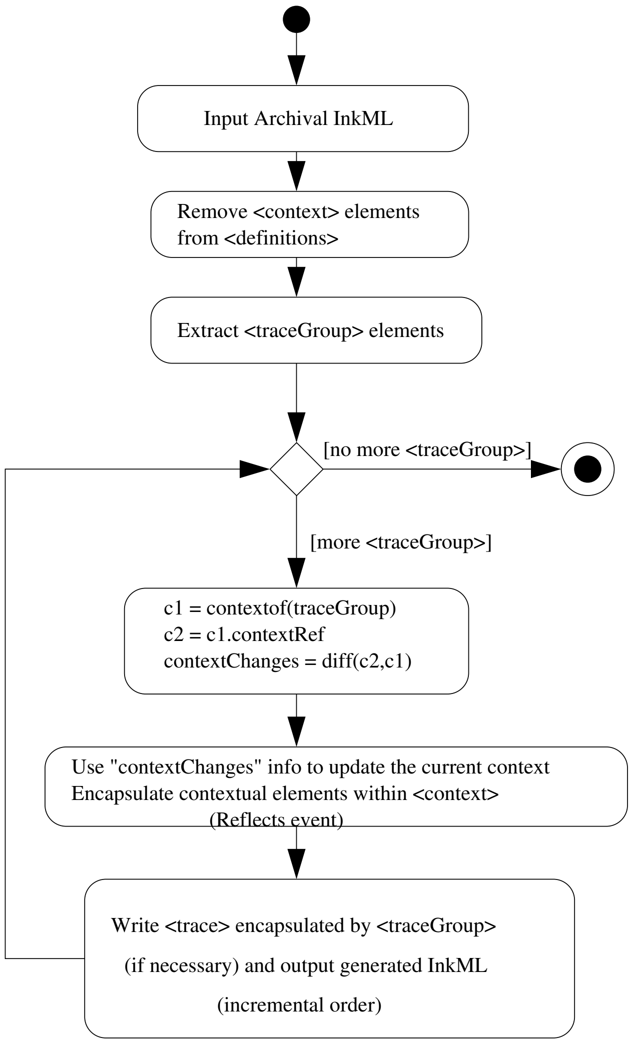

Input Archival InkML
Remove <context> elements from <definitions>
Extract <traceGroup> elements
[no more <traceGroup>]
[more <traceGroup>]
c1 = contextof(traceGroup)
c2 = c1.contextRef
contextChanges = diff(c2,c1)
Use "contextChanges" info to update the current context
Encapsulate contextual elements within <context>
(Reflects event)
Write <trace> encapsulated by <traceGroup>
(if necessary) and output generated InkML
(incremental order)

Figure 3.10: Archival to Streaming Translation (from [KW07b])

```
<ink>
   <definitions>
      <brush xml:id='penA'/>
      <canvas xml:id='can1'>...</canvas>
      <canvas xml:id='can2'>...</canvas>
      <canvasTransform xml:id='trans1'>...</canvasTransform>
      <traceFormat xml:id='format1'>...</traceFormat>
      <traceFormat xml:id='format2'>...</traceFormat>
   </definitions>
   ...
   <context canvasRef= '#can1'
       canvasTransformRef='#trans1' traceFormatRef='#format1'/>
   <trace>...</trace>

   <context traceFormatRef='#format2'/>
   <trace>...</trace>

   <context canvasRef='#can2'/>
   <trace>...</trace>

   <context canvasRef='#can1'/>
   <trace>...</trace>

   <context brushRef='#penA'/>
   <trace>...</trace>

   <context traceFormatRef='#format1'/>
   <trace>...</trace>
   <trace>...</trace>

</ink>
```

Figure 3.11: A Streaming Style Markup (Output)

Chapter 4

Techniques for Sharing Digital Ink

In this chapter we discuss different InkML-based schemes that allow to share digital ink in heterogeneous collaborative environments. We analyze each technique and compare them. Part of this work was done at HP Labs at Bangalore, India during a summer internship under the supervision of Dr. Sriganesh Madhvanath and is the basis of an article which is to appear in the International Conference on Frontiers in Handwriting Recognition (ICFHR), 2008 ([KMA+08]).

4.1 Using Shared Canvas

To allow multiple clients (ink sources) share digital ink, there should be some method that allows the conversion of digital ink from each source's trace format to each target's trace format. This is achieved using the `<canvas>` and `<canvasTransform>` elements discussed in Chapter 2. The element `<canvas>` provides a shared virtual space for cooperation of ink applications. The main advantage of such an intermediate representation is that only $(m+n)$ conversions are required to convert from m sources to n targets instead of $(m * n)$, as illustrated in Figure 4.1.

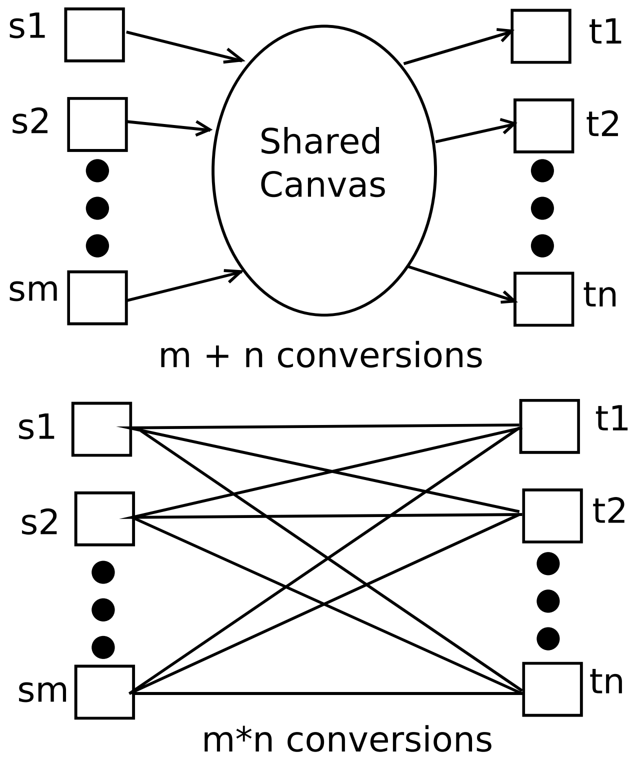


Figure 4.1: Sharing ink with shared canvas(upper) and without it(lower)

The conversion of digital ink from a source trace format to target trace format involves two steps: (i) convert the digital ink from source trace format to shared canvas trace format using forward canvas transform of the source and (ii) convert the digital ink in shared canvas trace format to the target trace format using inverse canvas transform of the target. For example, in Figure 2.10, in order to convert the digital ink from *device1*'s trace format to *device2*'s trace format, we can apply *CT1* first and then *iCT2* to the ink data.

From an implementation standpoint, InkML provides greater flexibility in the sense that the canvas transformations (forward and inverse) can be applied at different locations (client or server) depending upon the requirements. In the following subsections, we discuss different possible schemes and highlight their advantages and disadvantages. Depending on the location at which the transformations are applied, we may categorize these schemes as (i) client-oriented (ii) server-oriented and (iii) hybrid. In all these schemes, we assume that there are several clients participating in the ink communication coordinated by a central server, and all communication is always via the server (Figure 4.2). Also, the choice of the trace format of the shared canvas is application dependent.

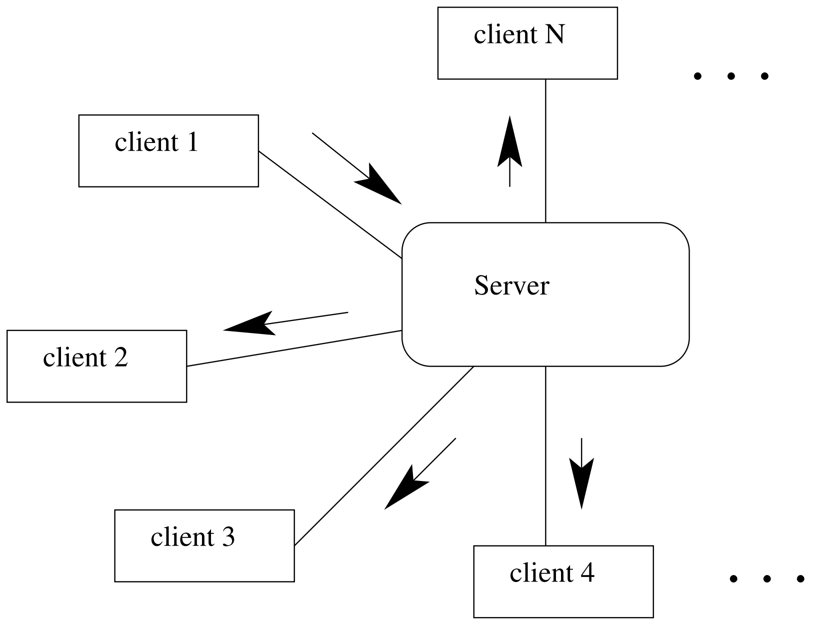


Figure 4.2: Collaborative Environment

4.2 Client-oriented Schemes

In this scheme, each client is responsible for keeping a record of information (trace format and canvas transform) about other clients which is required to compute transformations and also for applying the transformations in both directions (i.e. to and from the shared canvas format). This scheme can be further categorized into two variants depending on whether the forward transformation is applied before sending the ink data to the server, or after.

4.2.1 Scheme I

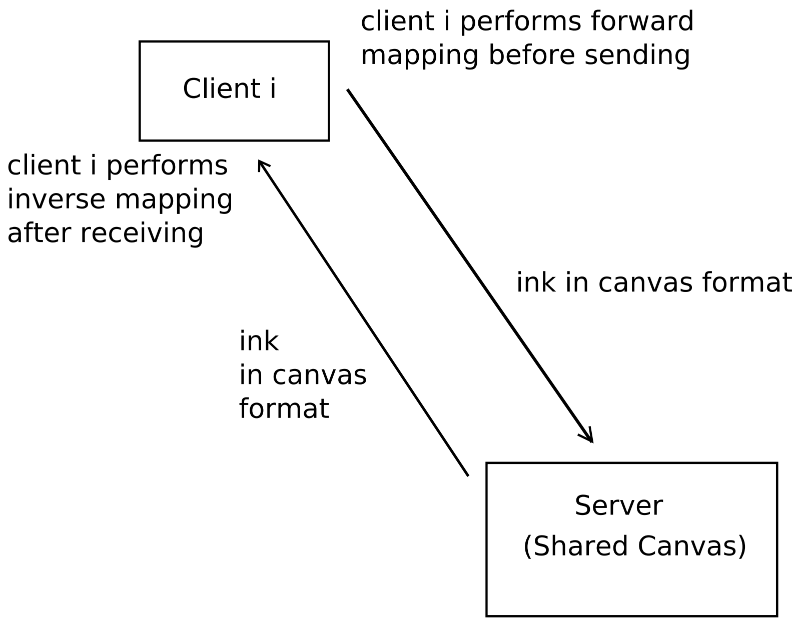

client i performs forward mapping before sending
Client i
client i performs inverse mapping after receiving
ink in canvas format
ink in canvas format
Server (Shared Canvas)

Figure 4.3: Client-oriented Scheme I

In this scheme, client applies forward transformation before sending the ink to the server. The following sequence of activities occurs whenever a client c_i joins the session or when its pen-device changes:

- The server sends shared canvas information to the client c_i in the following format:

```
<ink>
   <definitions>
      <canvas xml:id='sharedCanvas'>
         <traceFormat xml:id='canvasFormat'>
         ...
```

```
            </traceFormat>
        </canvas>
    </definitions>
    <context traceFormatRef='#canvasFormat'/>
</ink>
```

The server alters the current context by establishing current trace format to be the canvas trace format. This is because later, all the ink broadcast by the server is in the canvas format.

- By comparing its own trace format with the trace format of the shared canvas, client c_i computes its canvas transform and saves it for later use. Every time the ink source of client c_i generates ink, forward canvas transformation is applied to it and then the transformed trace data is sent to the server.

- The server broadcasts the ink received from client c_i to all clients (may be except client c_i in which case it is generated internally).

The above technique guarantees that each client receives the digital ink in the trace format of the shared canvas. Since each client saves its canvas transformation, it can apply the inverse canvas transformation to the ink that it receives from server to transform the ink to its own format.

The trace format of the ink during the entire session doesn't change. However, other contextual elements (e.g. change in brush) can change and are therefore maintained globally at the server. Whenever the context at a client changes, the server compares the effect of the change with the global current context. The context change is broadcast if the change is different from the current global context, otherwise the change is ignored. This also helps to reduce data on the wire. For example, if client c_i changes the brush color to red and the previous color the server broadcast was blue, the current brush at server is changed to red. When another client c_j changes its brush color to red again, there is no effect. If client c_j changes its brush to blue then the global context at the server is updated to blue.

Advantages:

- The server application is very simple as it simply broadcasts whatever it receives.
- Since switching of context is not required when the sender changes, the size of the data on the wire is smaller.

Disadvantages:

- Performing ink mapping in both directions may not be feasible for resource constrained devices such as PDAs, Smartphones and so on.
- There could be loss of precision due to approximation as a result of transformation.
- The server has to maintain the global context.

4.2.2 Scheme II

In this scheme both the transformations are done by a client similar to the previous scheme. However, the forward transformation is applied only after the client receives the ink from the server. The following sequence of activities occurs whenever a client c_i joins the session or when its pen-device changes:

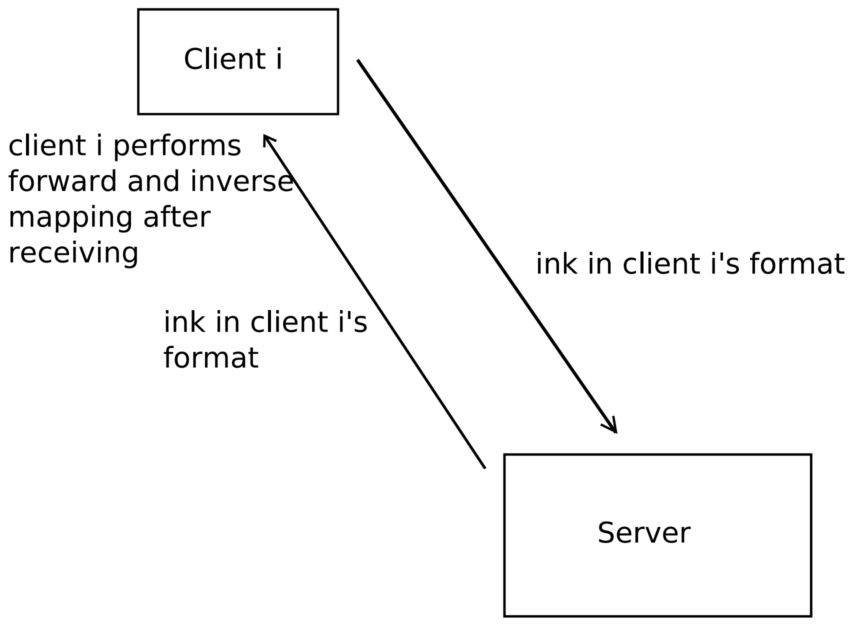


Figure 4.4: Client-oriented Scheme II

- The server sends shared canvas information to the client c_i in the following format:

```
<ink>
   <definitions>
      <canvas xml:id='sharedCanvas'>
         <traceFormat xml:id='canvasFormat'>
         ...
         </traceFormat>
      </canvas>
   <definitions>
</ink>
```

  - By comparing its own trace format with the trace format of the shared canvas, client c_i computes its `<canvasTransform>` and then it sends its ink source definition followed by the canvas transform to the server in the following format:

```
<ink>
   <definitions>
      <inkSource xml:id='ci'>
         ...
```

```
            <traceFormat xml:id='ciTF'>
            ...
            </traceFormat>
            <channelProperties>
            ...
            </channelProperties>
            ...
         </inkSource>
         <canvasTransformation xml:id='ciCT'>
         ...
         </canvasTransformation>
      </definitions>
   </ink>
```

- The server broadcasts the definitions received from client c_i to all the clients (may be except c_i in which case the client generates them internally).

The above technique allows each client to know the canvas transformation of all other clients. In order to send the ink to the server, a client c_i sends the contextual element `<context traceFormatRef='#ciTF' canvasTransformRef='#ctCT'/>` first and then the trace data. The server broadcasts the trace data received from client c_i to all the clients. Each client c_j receiving the ink can thereafter perform forward mapping (from client c_i to canvas) as well as inverse mapping (from canvas to client c_j) to get the ink in its own format.

Advantages:

- This technique allows digital ink to be collected from all ink sources in its original form, without any modification or loss of precision due to approximation as a result of transformations.

- The server application is very simple as it simply broadcasts whatever it receives just like the previous scheme.

- The server doesn't need to maintain global context.

Disadvantages:

- There are a number of context switches as each client alters the context before sending its trace data. This can increase the amount of data transmitted.
- Similar to previous approach, performing ink mapping in both directions may not be feasible for resource constrained clients such as PDAs, Smartphones and so on.

4.3 Server-oriented Scheme

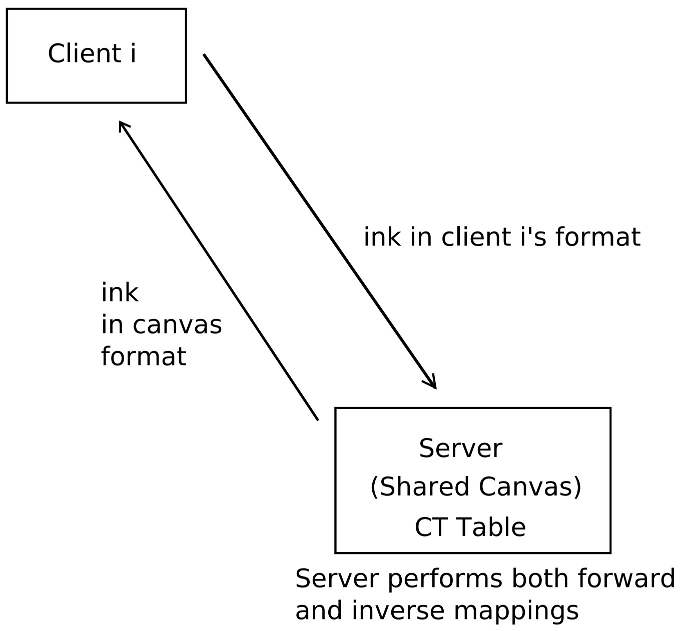


Figure 4.5: Server-oriented Scheme

This scheme puts the burden of record keeping and transformation on the server, and relieves the clients of these tasks. The following sequence of activities occurs whenever a client c_i joins the session or when its pen-device is changed:

- The client sends its ink source definition in the following format:

```
<ink>
   <definitions>
      <inkSource xml:id='ci'>
         ...
         <traceFormat xml:id='ciTF'>
         ...
         </traceFormat>
         <channelProperties>
         ...
         </channelProperties>
         ...
      </inkSource>
   </definitions>
</ink>
```

- The server computes the canvas transformation for client c_i and keeps a record of it (does not broadcast to other clients). It thus constructs a *Canvas Transformation Table* (CT Table).

When the server receives trace data from the client c_i, it finds the forward canvas transformation of client c_i from CT Table and then applies it to the trace data to convert it to the shared canvas format. The shared canvas is private to the server and clients are completely unaware of it. In order to send the ink data to a particular client c_j, it uses CT Table again to find the client's corresponding inverse mapping, applies it and sends the transformed ink data to the client. Thus, it applies transformation selectively for each client and then sends the transformed ink data to each client separately. In addition, this scheme also requires the server to maintain a global context similar to the one described in client-oriented scheme I and this helps to reduce the data on the wire to some extent as context change from a client is broadcast only when the change can later the current global context.

Advantages:

- Since there are no frequent context switches, data on the wire is reduced.

- There is no burden on the client, which makes this scheme particularly suitable for resource constrained devices.

Disadvantages:

- Simple broadcast by the server is not possible, since selective transmission to each client is also required.
- The ink data collected by the client is the transformed version, possibly resulting in loss of precision.
- The server has to maintain the global context.

4.4 Hybrid Scheme

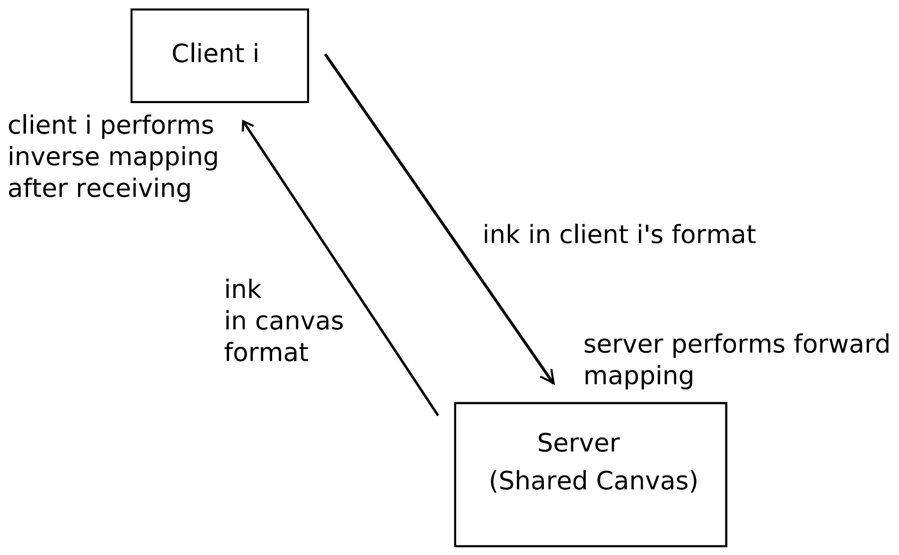


Figure 4.6: Hybrid Scheme

This scheme can be thought of as a blend of the two previous schemes. It helps retain some of the benefits of the previous approaches and strikes a better balance of server load and complexity on one hand, and support for resource constrained clients on the other. The central idea is to perform half of the transformation (forward transformation) at the server and the other half (inverse transformation) at the client. Under this scheme, the following sequences of activities can occur whenever a client c_i joins the session or when its pen-device changes:

- The server sends the shared canvas information to client c_i in the following format:

```
<ink>
   <definitions>
      <canvas xml:id='sharedCanvas'>
         <traceFormat xml:id='canvasFormat'>
         ...
         </traceFormat>
      </canvas>
   </definitions>
</ink>
```

- By comparing its own trace format with the trace format of the shared canvas, client c_i computes its `<canvasTransform>` and then it sends its ink source definition followed by the canvas transform in the following format:

```
<ink>
   <definitions>
      <inkSource xml:id='ci'>
         ...
         <traceFormat xml:id='ciTF'>
         ...
         </traceFormat>
         <channelProperties>
         ...
         </channelProperties>
         ...
      </inkSource>
      <canvasTransformation xml:id='ciCT'>
      ...
      </canvasTransformation>
   </definitions>
</ink>
```

- The server keeps a record of canvas transformation obtained from client c_i in the CT Table and then it sets the current trace format to be the shared canvas trace format by sending following markup to client c_i:

```
<ink>
   <context traceFormatRef='#canvasFormat'/>
</ink>
```

The channels in the trace format of the canvas should be a superset of the channels supported by all the clients. When the server receives trace data from client c_i, it finds it's forward mapping information from the CT table and applies it to the trace data. This also includes adding null values for unreported channels. Then the server does a simple broadcast of the ink data to all the clients. As with the client-oriented scheme I and server-oriented scheme, this scheme also involves the server maintaining a global current context. A change in context is broadcast only if it has resulted in a change to the global current context.

Advantages:

- Simple broadcasting by the server is sufficient.
- The size of the data on the wire is lower than in the client-oriented schemes.

Disadvantages:

- The size of the data on the wire is relatively greater than the server-oriented scheme.
- There could be loss of precision when the server applies forward canvas transformation.
- The server needs to maintain the global context.

Scheme	Data size on the wire	Processing load on client	Processing load on server	Server maintains global context?
Client I	Low	Medium	Low	Yes
Client II	High	High	Low	No
Server	Low	Low	High	Yes
Hybrid	Low	Low	Medium	Yes

Table 4.1: Comparison of the Schemes

4.5 Comparisons

In the previous sections we discussed different schemes and showed how each has its own advantages and disadvantages. Table 4.1 shows a comparison of the schemes.

The client-oriented schemes are suitable in the situations where the client devices have good processing capability and we want the server to be simple (broadcasting) with less overhead on it. Out of the two schemes, scheme I seems to be better than scheme II in terms of data size on the wire and processing overhead on the client. The Server-oriented scheme puts all the overhead on the server and hence is seems to be very suitable for resource constrained devices such as PDAs, Smartphones and so on. The only major disadvantage of this technique is the requirement of selective transmission of data by the server to the client. This makes it difficult to reuse a standard server such as the XMPP server [osc99]. The hybrid approach seems to create a balance between the other two schemes. Since half of the transformation is applied on the server and only the other half is applied at clients, the overhead on the clients is medium. At the same time, the server can simply broadcast the trace data and no selective transmission is required.

As a conclusion, we can say that the choice of the scheme is dependent mainly on the bandwidth of the network and nature of the clients and the server involved in the communication for the purpose of sharing digital ink. Hybrid and Client I schemes are suitable when a compromise is required.

4.6 Comparisons with Other Digital Ink Formats

Other popular digital ink format such as UNIPEN [Guy94] and Jot [Cor93] do not provide better support for streaming and exchanging digital ink. Hence, they may not be used for exchanging digital in a collaborative environment in an efficient manner.

Ink Serialized Form (ISF) [Mic04] may be used for streaming digital ink. Tablet PC SDK [Mic04] provides API to convert stroke objects to ISF and *vice-versa*. But ISF doesn't support transmission of ink in an incremental order. For example, to send a red stroke it is required to send the stroke with color information everytime, even if the color of the current stroke is same as the previous one.

In order to support exchange of digital ink, Tablet PC SDK provides an intermediate representation in HIMETRIC unit (each unit is 0.01 millimeter). The API has built-in functions for doing conversion between ink space and pixel space of the display. The concept of ink space is analogous to the concept of shared canvas in InkML. This technique of representing the digital ink in an intermediate form using HIMETRIC unit is very similar to the client-oriented scheme I discussed in section 4.2.1. Since this requires both the transformation to be done by the clients it may not be suitable for resource constrained devices like PDAs, Smartphones and so on. Also, being proprietary format, ISF lacks the benefits which an open standard can offer.

Chapter 5

Support for Collaborative Mathematics

One useful application of the digital ink sharing techniques discussed in the previous chapter can be doing mathematics in collaborative environments. For a full-fledged collaborative mathematical environment, mathematical handwriting recognition is required. In this chapter, we present a framework that supports the symbol recognition needs of clients (especially resource-constrained devices) using a central symbol recognition server. We use InkML for the purpose of different types of communications between the clients and the server. In this chapter, we discuss the use of InkML to develop the communication protocol and also describe our symbol recognition component. The symbol recognition part of this chapter is based on our work [KW07a] published in the International Conference on Document Analysis and Recognition, 2007.

5.1 Motivations

In order to support a full-fledged collaborative mathematical environment, a mathematical handwriting recognizer is required. A core component of such handwriting recognizer is a symbol recognizer which requires a large amount of training data to build a model and also good processing capability for the prediction of new symbols. In resource constrained devices such as PDAs, Smartphones and so on, it might not be feasible to store the training data and train the recognizer. One could train the recognition system and build the model in a separate machine with good processing capabilities but sometimes the model can still be larger. This method would also not allow the system to adapt or evolve in future. Hence, a dedicated server for the task of recognition can be very useful. For this, there should be some techniques by which a client can send various kinds of requests (*eg.* recognize, retrain and so on) to the recognition server and the server can respond accordingly.

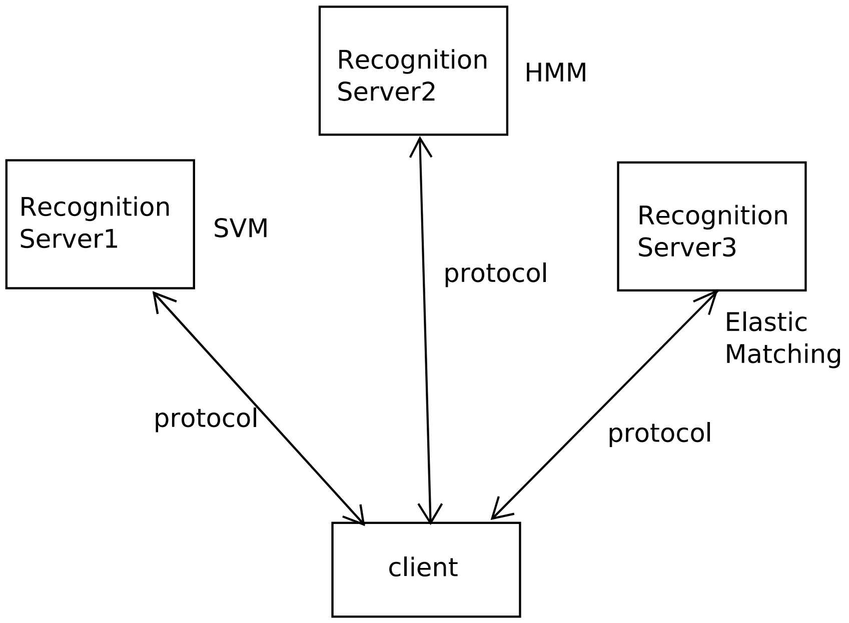


Figure 5.1: Results from Recognition Servers can be combined

Another advantage of the client-server architecture is that that recognition results from various servers can be combined to obtain better results. Figure5.1 shows a client combining results from three different recognition servers that use three different recognition algorithms (SVM, HMM and Elastic Matching). The combination of classifiers have been shown to be fruitful in various works such as [KW07a]. We discuss the InkML-based protocol used for the communication between the recognition servers and the clients in the following sections.

5.2 Use Cases

The use cases of the InkML-based protocol used for the communication between the recognition server and the clients are shown in Figure5.2. Each use case is described below.

- **Login**: The protocol should allow the clients to log-in into the server by providing a valid client name and a password. The communication should start only after the client is authenticated.
- **Send Recognition Result**: The client should be able to send recognition request containing information such as the trace data to be recognized, writer info and so on.
- **Send Writer Info**: The protocol should allow the client to send writer information such as age, gender and so on. Such information can be helpful to the recognition module.
- **Send Training Data**: The protocol should allow the client to update the training data at any time.

- **Send Control Information**: The protocol should allow the client to send control information such as to retrain the system after training data is updated.
- **Send Recognition Result**: The protocol should allow the server to send the recognition result to the clients. This may contain probabilities of the recognized symbols.
- **Send Error Messages**: The protocol should allow the server to send error messages whenever an error occurs (*eg.* recognition error, malformed XML error).

5.3 The InkML-Based Protocol

We describe how the protocol captures each use case in the following sub-sections.

5.3.1 Login

In order to start communication with the server, client provides a valid user name and a password in the following format:

```
<authenticate>
   <username>...</username>
   <password>...</password>
</authenticate>
```

5.3.2 Client Recognition Request

Client sends the recognition request in the following format:

```
<recognitionRequest xml:id='request1'>
<ink>
   <traceGroup>
      <annotationXML>
         <writerID>bkeshari</writerID>
```

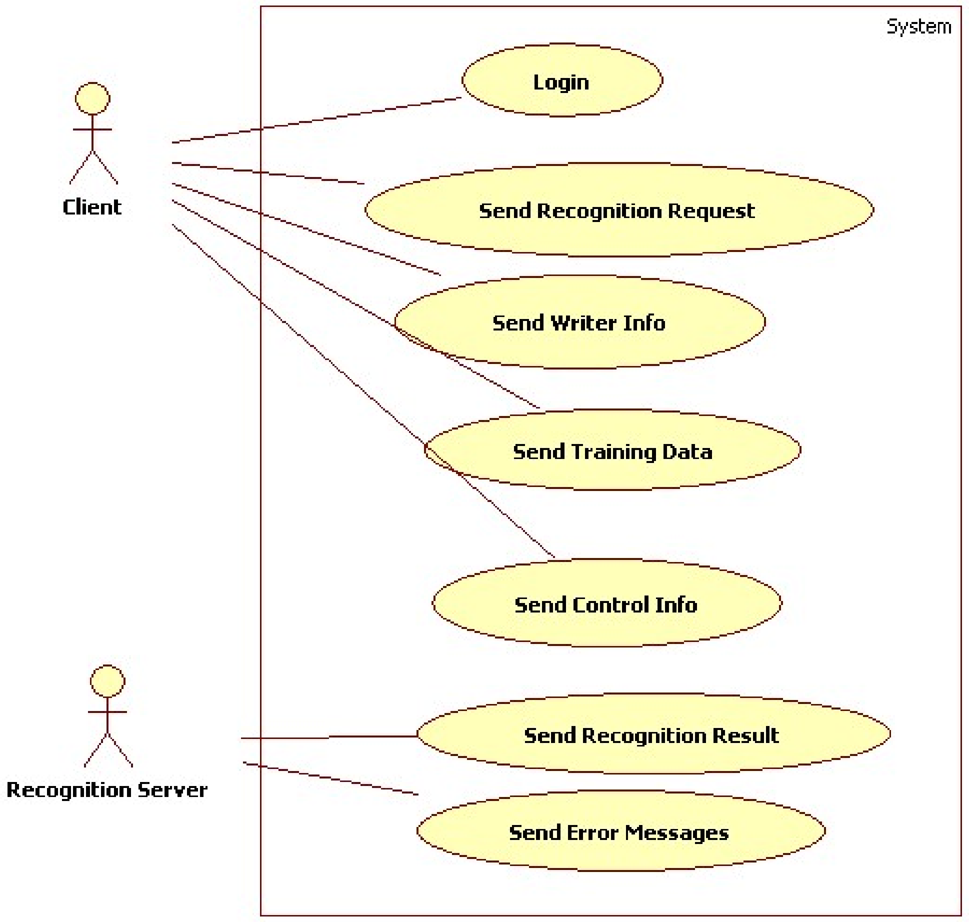


Figure 5.2: Use Case Scenarios

```
      </annotationXML>
      <traceGroup xml:id='symbol1'>
         <trace xml:id='t1'>x1 y1, x2 y2, x3 y3, x4 y4 ...</trace>
         <trace xml:id = 't2'> ... </trace>
      </traceGroup>
      <traceGroup xml:id='symbol2'>...</traceGroup>
      </traceGroup>
   </ink>
</recognitionRequest>
```

The `id` attribute of `<recognitionRequest>` must be unique for an entire session. Annotation XML may be used to put the writer id in the request to instruct the server to use the model of a particular user. However, this is not mandatory. If no writer id is provided, the server tries to use the model of the user whose name matches with the login name. Otherwise, writer independent model is used.

In the above example, trace grouping has been performed by the client. Client may also send traces without grouping and in such cases, server will do the trace grouping.

```
<recognitionRequest xml:id='request1'>
   <ink>
      <traceGroup>
         <annotationXML>
            <writerID>bkeshari</writerID>
         </annotationXML>
         <trace xml:id='t1'>x1 y1, x2 y2, x3 y3, x4 y4 ...</trace>
         <trace xml:id = 't2'> ... </trace>
         ... (other traces)
     </traceGroup>
    </ink>
</recognitionRequest>
```

5.3.3 Server Recognition Response

The server responds to the client recognition request using `<recognitionResult>`. The attribute `recognitionRequestRef` identifies the request which is being responded. All the recognition results are enclosed within a `<symbolRecoResult>` element. Its `traceGroupID` identifies the group of traces that were recognized as a single symbol. Each result is expressed using a `<result>` element and server may provide top N choices with probabilities, where the value of N is controlled by the client. The default value is 1 which means the server doesn't include probability. The result contains name of the recognized symbol (value of element `<symbolName>`), Unicode value of the recognized symbol (value of element `<unicode>`) and it may also contain the estimated probability of correctness (value of element `<probability>`).

```
<recognitionResult recognitionRequestRef='#request1'>
   <symbolRecoResult traceGroupID='symbol1'>
      <result>
         <symbolName>sigma</symbolName>
         <unicode>u0187</unicode>
         <probability>0.56</probability>
      </result>
      <result>
         <symbolName>d</symbolName>
         <unicode>u0107</unicode>
         <probability>0.26</probability>
      </result>
    </symbolRecoResult>
    ...
</recognitionResult>
```

In the above example the trace grouping was already done by the client. In the cases where the grouping is not already done by the client, the server uses the `<symbolSegment>` element to provide the grouping information to the client. For example in the following response, trace `t1` and `t2` has been grouped and recognized as `sigma` with a probability of 0.56 and `d` with a probability of 0.26.

```
<recognitionResult recognitionRequestRef='#request1'>
   <symbolRecoResult>
      <symbolSegment traceID='t1'/>
      <symbolSegment traceID='t2'/>
      <result>
         <symbolName>sigma</symbolName>
         <unicode>u0187</unicode>
         <probability>0.56</probability>
      </result>
      <result>
         <symbolName>d</symbolName>
         <unicode>u0107</unicode>
         <probability>0.26</probability>
      </result>
   </symbolRecoResult>
   ...
</recognitionResult>
```

5.3.4 Sending Writer Info

Client may send writer information in the following format:

```
<writer id='bkeshari'>
   <name>Birendra Keshari</name>
   <age>24</age>
   <gender>male</gender>
   <rightHanded>true</rightHanded>
</writer>
```

5.3.5 Sending Training Data

A client may send training samples to the recognition server and the server may store it for training in future. The training data should contain the name of each symbol and also its unicode. It may also contain an optional writer id and if it is not specified, the current login name is assumed to be the writer id. All this information is put in the InkML using `<annotationXML>`.

```
<trainingData>
   <ink>
      <traceGroup>
         <annotationXML>
            <writerID>bkeshari</writerID>
         </annotationXML>
         <traceGroup>
            <annotationXML>
            <symbolAnnotation>
               <symbolName> pi </symbolName>
               <unicode> U0078 </unicode>
            </symbolAnnotation>
            </annotationXML>
               <trace xml:id = 't1'> ... </trace>
         </traceGroup>
         <traceGroup> ... </traceGroup>
      </traceGroup>
</ink>
</trainingData>
```

5.3.6 Exchange of Control Information

- A client may request the server to build the model for a particular user in the following way:

  ```
  <trainingRequest xml:id='tr1'>
     <writerID>bkeshari</writerID>
  </trainingRequest>
  ```

  The server will respond a `<trainingRequest>` by a `<trainingRequestResult>` which encloses training error information in the following way.

  ```
  <trainingRequestResult trainingRequestRef='#tr1'>
     <trainingError>9</trainingError> (Error in percentage)
  </trainingRequestResult>
  ```

- Client may instruct the server to provide top N results of symbol recognition in the following format.

  ```
  <topNRecognitionResult value='3'/>
  ```

5.3.7 Error Handling

Server may send an error message using `<error>` element whenever an error occurs. The `code` attribute is set to a decimal value which specifies the error code. Various situations that may cause the server to send error messages to the client are listed below.

- The Server may send an error message in response to `<recognitionRequest>` if recognition fails. An example is given below.

  ```
  <recognitionResult recognitionRequestRef='#request1'>
     <error code='1'/> (none of the traces were recognized)
  </recognitionResult>
  ```

Error Code	Meaning
0	Unknown Error
1	Requested user's model data not found for prediction
2	Model data not found for prediction
3	Recognition failed during feature extraction
4	Recognition failed during prediction
5	Training failed during feature extraction
6	Training failed while building the model
7	Malformed XML

Table 5.1: Error Codes

```
<recognitionResult recognitionRequestRef='#request1'>
   <traceRecoResult traceID='t1'>
      <error code='1'/>
   </traceRecoResult>
   ... (others are recognized)
</recognitionResult>
```

- The Server may send error message in response to `<trainingRequest>` if training fails due to any reason. An example is given below.

```
<trainingRequestResult trainingRequestRef='#tr1'>
   <error code='0'/>
</trainingRequestResult>
```

Table 5.1 lists the error codes and their meaning used in the protocol.

5.4 Online Symbol Recognizer

We use support vector machines to train the symbol recognizer and also to predict new symbols. We describe support vector machines and the features used to train them in the following sub-sections.

5.4.1 Support Vector Machines

SVM is a supervised learning algorithm that has been widely and successfully used for pattern recognition in different areas. SVM is based on dual ideas of *VC (Vapnik-Chervonenkis) dimension* and *structural risk minimization principle* [Vap98]. The decision boundary in SVM is a hyperplane that separates the two classes leaving largest margin between the vectors of the two classes. However, in real life, problems can be linearly in-separable. To deal with this problem, a non-linear decision surface is obtained by using kernel that lifts the feature space into a higher dimension (can be infinite). A linear separating hyperplane is found in the higher dimensional space and this gives a non-linear decision surface in the original feature space. The decision function of SVM can be expressed by the following equation:

$$f(x) = \sum_i \alpha_i y_i K(x, x_i) + b \tag{5.1}$$

where y_i is the label of training pattern x_i and x is the pattern to be classified. Parameters α_i and b are found by maximizing a quadratic function subject to some constraints [Vap98]. Here $K(x, x_i) = \phi(x).\phi(x_i)$ is the kernel function and ϕ maps the feature vectors into a higher dimension inner product space. The most commonly used kernels are:

- $K(a, b) = \exp(-\gamma||a - b||^2), \gamma > 0$ (Radial Basis Function)
- $K(a, b) = (\gamma(a.b) + r)^d, \gamma > 0$ (Polynomial)
- $K(a, b) = \tanh(\gamma(a.b) + r)$ (Sigmoid)

Although SVM is primarily binary classifier,a multi-class classifier can be created by combining several binary classifiers. One-against-all, One-against-one and DAG

SVM are the popular techniques to combine binary classifiers to build multi-class SVM. Comparisons between different methods in [HL02] show that DAG and one-against-one are more suitable for practical use than the other existing methods.

The outputs from standard SVM are not calibrated probability estimates. More interesting tasks like post-processing and combining classifiers can be done with SVM when the outputs are probability estimates instead of labels. Various researchers have been working to output posterior probabilities with SVM. A good measure can be the distance of the pattern from the hyperplane. Platt [Pla99] developed a method to transform this distance into posterior probability by applying a sigmoid function on the outputs of the SVM as follow:

$$p(y = 1|f(x)) = \frac{1}{1 + \exp(Af(x) + B)} \tag{5.2}$$

where $f(x)$ is the output from SVM. Parameters A and B are obtained by minimizing the negative log-likelihood function on training data. Based on this work, an efficient method to compute the probability has been described in [WLW04] and implemented in `libsvm` [CL01] library.

5.4.2 Feature Extraction

5.4.3 Preprocessing

The following steps are involved in preprocessing:

- **Smoothing:** The following Gaussian smoothing is used to remove noise:

$$\begin{aligned} x_i &= 0.25 * x_{i-1} + 0.5 * x_i + 0.25 * x_{i+1} \\ y_i &= 0.25 * y_{i-1} + 0.5 * y_i + 0.25 * y_{i+1} \end{aligned}$$

- **Filling the intermediate points:** Linear interpolation is applied between every pair of consecutive points on each stroke to fill the intermediate points. This removes the time information but it was found that spatial alignment is more useful than temporal alignment of the points on the strokes. Hence, during re-sampling phase we re-sample the points at equal distance.

- **Re-sampling:** Since the number of points on each stroke is large after interpolation, re-sampling is required. Fewer number of points may miss useful information such as curvature, loops etc. and a larger number of points may increase the computation time. Therefore, a balance between the two is required. We determine the number of points heuristically. Each stroke is re-sampled to a fixed number of points (11).

- **Size Normalization:** Each stroke is scaled by $\frac{1}{max(h,w)}$, where h and w are height and width of the bounding box of the symbol. Strokes are then translated so that the whole symbol fit inside a square of unit length and their centers coincide.

Figure5.3 illustrates the preprocessing steps.

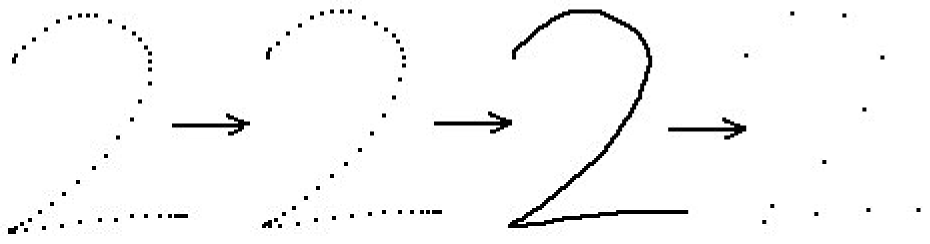


Figure 5.3: Processing: smoothing, interpolating and re-sampling (from [KW07a])

5.4.4 Online Feature Vector

An online feature vector for each stroke is extracted after the symbol is preprocessed. The feature vector consists of following features:

- Co-ordinates of each point on the stroke (x_i and y_i)
- Sines and cosines of the angles made by the line segments of the stroke
- Sines and cosines of turning angle between line segments. Figure5.4 shows an example of turning angle.
- Centre of gravity of the symbol calculated as $(\sum_i(x_i/N), \sum_i(y_i/N))$, where N is the total number of points on the stroke and x_i and y_i are the co-ordinates of each point.

Similar features have been used in [TR05]. It is found that co-ordinates and angle information are the most discriminating features. The decrease in error rate of the online system is very small when the size of the feature vector is increased by introducing new geometric features like relative length. However, the number of support vectors is reduced.

5.5 An Example

Figure5.5 shows an example of the protocol where client sends trace groups to the server and then server recognizes the symbols and sends the recognition result back to the clients.

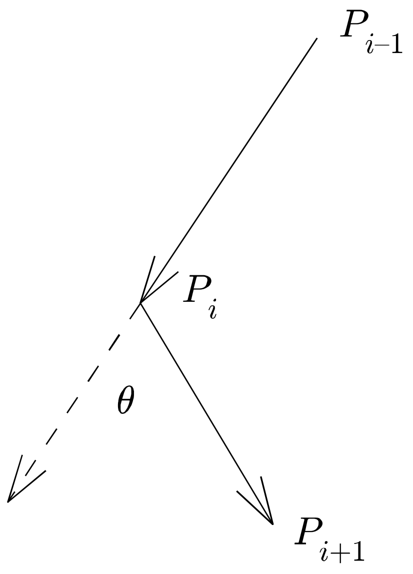
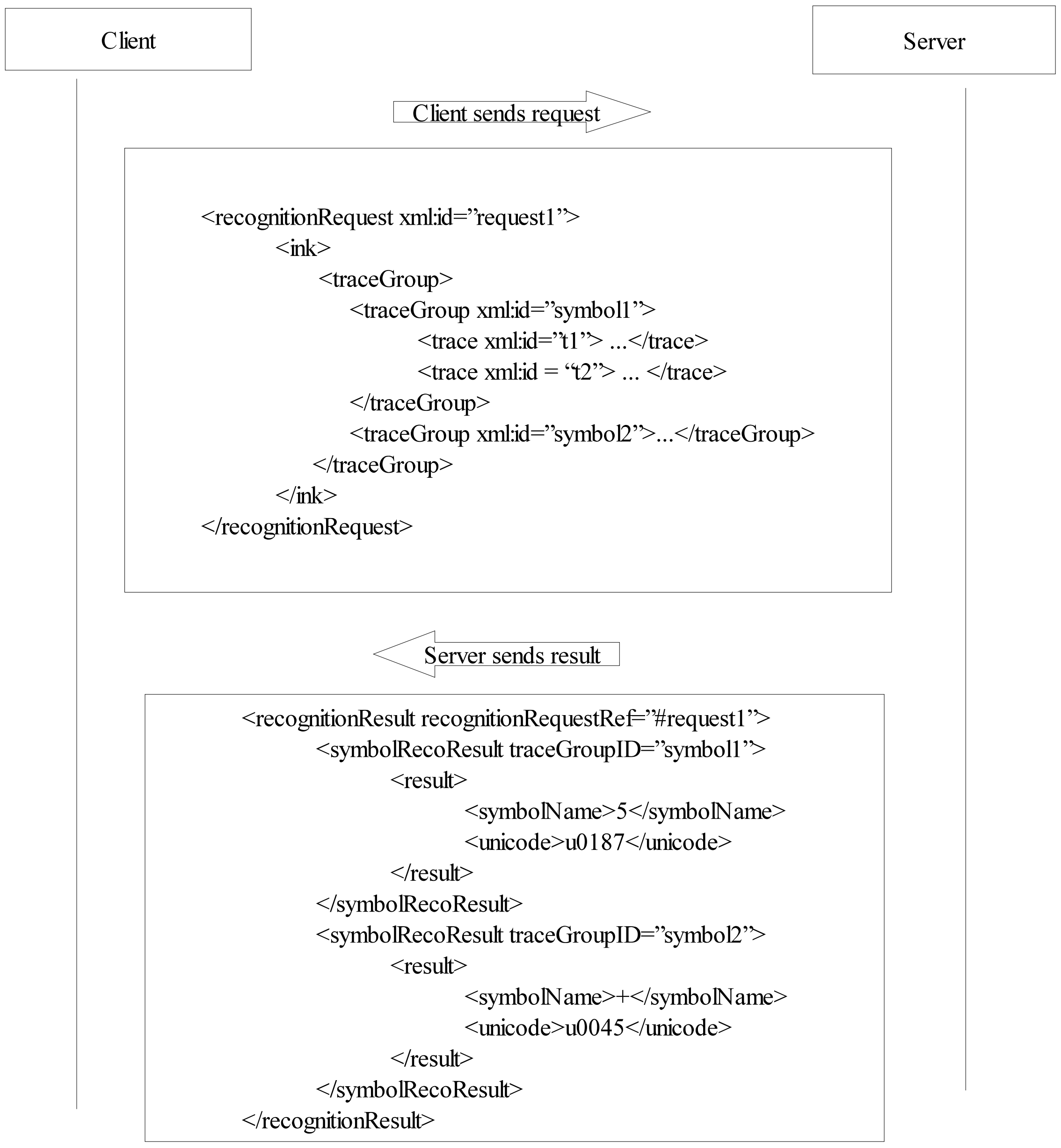


Figure 5.4: Turning Angle(θ) (from [KW07a])



```
<recognitionRequest xml:id=”request1”>
          <ink>
                <traceGroup>
                    <traceGroup xml:id=”symbol1”>
                              <trace xml:id=”t1”> ...</trace>
                              <trace xml:id = “t2”> ... </trace>
                    </traceGroup>
                    <traceGroup xml:id=”symbol2”>...</traceGroup>
               </traceGroup>
          </ink>
</recognitionRequest>
```

```
<recognitionResult recognitionRequestRef=”#request1”>
          <symbolRecoResult traceGroupID=”symbol1”>
                    <result>
                              <symbolName>5</symbolName>
                              <unicode>u0187</unicode>
                    </result>
          </symbolRecoResult>
          <symbolRecoResult traceGroupID=”symbol2”>
                    <result>
                              <symbolName>+</symbolName>
                              <unicode>u0045</unicode>
                    </result>
          </symbolRecoResult>
</recognitionResult>
```

Figure 5.5: An example

Chapter 6

Implementations

This chapter provides the implementation details of the techniques and components described in the previous chapters.

6.1 Streaming-Archival Conversion

The streaming-archival conversion algorithms have been implemented as an API in Java. The usage of the API has been illustrated below.

```
import ca.uwo.csd.orcca.penmath.inkml.saconverter.*;

...
   StreamingToArchival sa = new StreamingToArchival();
   sa.setStreamingFileName(fileName);
   sa.setOptimization(true);
   sa.setIndent("   ");
   String strArchival = sa.translate();
...
```

The archival InkML gets optimized only when the optimization flag is turned on using `setOptimization` method. The `setIndent` method is used to specify the string to be used for indentation.

The archival to streaming API is very much similar to the streaming to archival API and can be used in the following way.

```
import ca.uwo.csd.orcca.penmath.inkml.saconverter.*;

...
   ArchivalToStreaming as = new ArchivalToStreaming();
   as.setArchivalFileName(fileName);
   as.setIndent("   ");
   String strStreaming = as.translate();
...
```

Archival to streaming API can also be used in an iterative style as shown below.

```
import edu.uwo.csd.orcca.penmath.inkml.saconverter.*;

...
   ArchivalToStreaming as = new ArchivalToStreaming();
   as.setStreamingFileName(fileName);
   as.getStreamingDefinitions();
   while(as.hasMoreStream())
   {
      String streamingInkML = as.nextStream();
   }
...
```

The above style outputs all the traces that share the same context in the same iteration. This could be helpful for example to animate the ink traces.

We have also implemented a web interface for these translators using PHP and Java Script. PHP functions creates a translator process, sends the input to it, gets the result back and displays it in the browser. Figure 6.1 shows a screen shot of the web interface and an output window is shown in Figure 6.2.

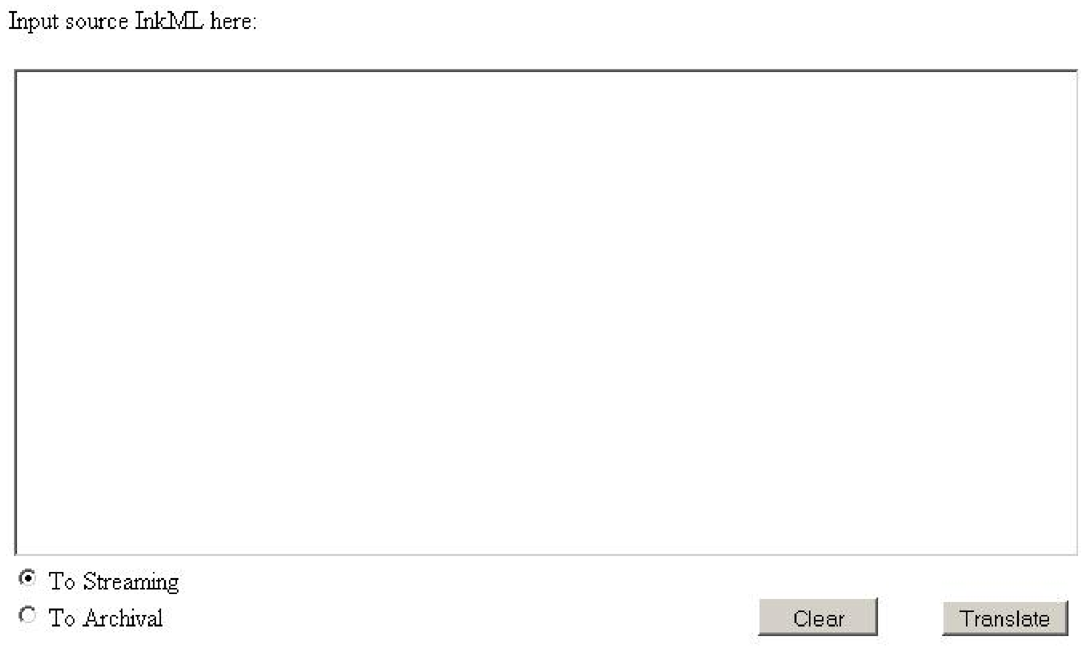
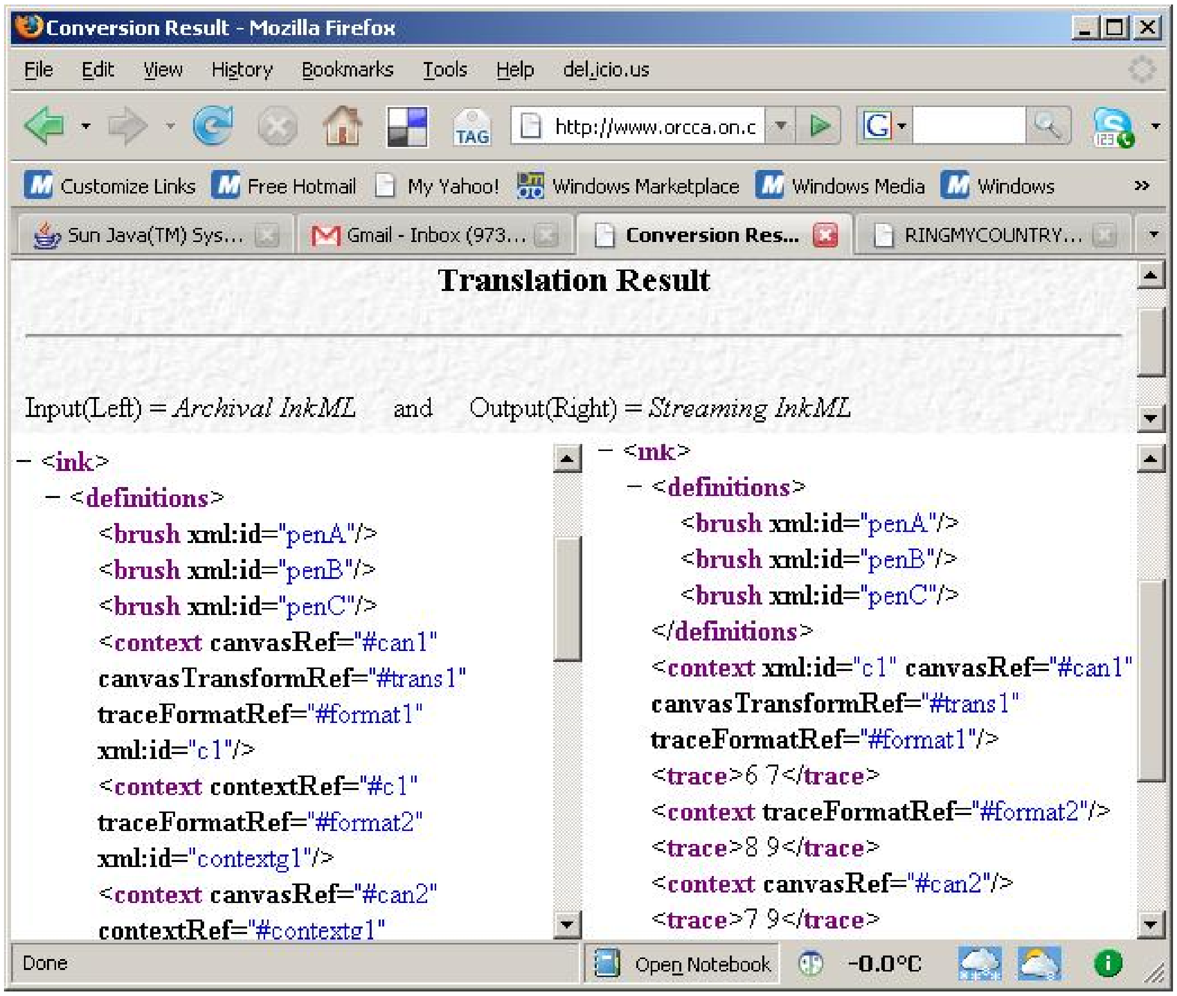


Figure 6.1: A Web Interface for Streaming-Archival Translators

6.2 Sharing Digital Ink

A collaborative inking application has been implemented using the hybrid scheme described in section 4.4 as it enjoys the benefits of both client-oriented scheme and server-oriented scheme and balances the processing load between the server and the client. The system is implemented in Java for better portability. Currently, the system supports mouse, graphics tablet and Tablet PC devices under Windows and Linux platforms.

The trace format of the shared canvas used in our implementation has X, Y and F (pressure) channels, which covers the channels of the devices we are currently using. The values of the channels X and Y are in absolute lengths units (millimeter) and we assume their resolution to be infinite. Channel properties of the ink source are used to compute forward and inverse canvas transform. For example, if channel X of a particular ink source has a spatial resolution of 10 ppi (points per inch), then the


Conversion Result - Mozilla Firefox
File Edit View History Bookmarks Tools Help del.icio.us
http://www.orcca.on.c
Customize Links Free Hotmail My Yahoo! Windows Marketplace Windows Media Windows
Sun Java(TM) Sys... Gmail - Inbox (973... Conversion Res... RINGMYCOUNTRY...
Done Open Notebook -0.0°C

Translation Result

Input(Left) = *Archival InkML* and Output(Right) = *Streaming InkML*

```
- <ink>
  - <definitions>
      <brush xml:id="penA"/>
      <brush xml:id="penB"/>
      <brush xml:id="penC"/>
      <context canvasRef="#can1"
      canvasTransformRef="#trans1"
      traceFormatRef="#format1"
      xml:id="c1"/>
      <context contextRef="#c1"
      traceFormatRef="#format2"
      xml:id="contextg1"/>
      <context canvasRef="#can2"
      contextRef="#contextg1"
```

```
- <ink>
  - <definitions>
      <brush xml:id="penA"/>
      <brush xml:id="penB"/>
      <brush xml:id="penC"/>
    </definitions>
    <context xml:id="c1" canvasRef="#can1"
    canvasTransformRef="#trans1"
    traceFormatRef="#format1"/>
    <trace>6 7</trace>
    <context traceFormatRef="#format2"/>
    <trace>8 9</trace>
    <context canvasRef="#can2"/>
    <trace>7 9</trace>
```

Figure 6.2: Output Window of Streaming-Archival Translators

forward canvas transform for that source would involve a multiplication factor of 2.54 (to convert from inches to millimeters).

The system provides a feature to set images of documents, maps and photographs as the canvas background. This allows to do several interesting and useful tasks such as collaborative document annotation and collaborative map annotation. Besides ink, our implementation can also send text and audio messages. The audio capability is added using the Smack Jingle API [Rea06a]. The canvas background image is saved in the ink using an annotation XML. The ink conversations which are in streaming InkML form can be translated to archival form using streaming to archival translator.

6.3 InkML-Based Protocol for Recognition

We have implemented a symbol recognition server that understands the protocol described in Chapter 5 in Python. The symbol recognition component uses support vector machines (machine learning algorithm) and it is implemented using `libsvm` library [CL01]. Features are extracted from the samples in the format understandable by libsvm. All the features are scaled between 0 and 1. Radial Basis Function (RBF) is used as the kernel after tyring several other kernels. For multi-class classification, we use one-against-one strategy and the probabilities are obtained through an optimization performed on the pairwise class probabilities [Pla99]. Best values of γ (RBF parameter) and C (SVM regularization parameter) are found by performing grid search with γ in the range 2^{-15}, 2^{-13}, ..., 2^{3} and C in the range 2^{-5}, 2^{-3}, ..., 2^{15}.

The dataset available at [dat] is used for training the recognition system. The training set consists of handwritten samples from 11 subjects (seven male and four female) collected using a Hewlett-Packard Laboratories (HP) Compaq tc1100 Tablet PC. The symbol set consists of 48 different symbols including *a*-*z*, 0-9, $\sum$, (,), $-$, $\surd$,

$\int$, $\{$, $<$, $>$, $+$, $\neq$ and *else*. The training dataset consists of each samples written 10 times by each subject. The data was converted to InkML format [CFW06] for better portability in the future.

A web-based client interface (Java Applet) which communicates with the recognition is shown in Figure 6.3. This was developed to test and debug the recognition server. It allows user to scribble a symbol on its ink canvas. It creates InkML markup from the traces, encloses it within a recognition request markup and sends them to the recognition server for recognition. The server recognizes the symbol and sends back the result to the client. The client parses the recognition result markup and displays the recognition result to the user.

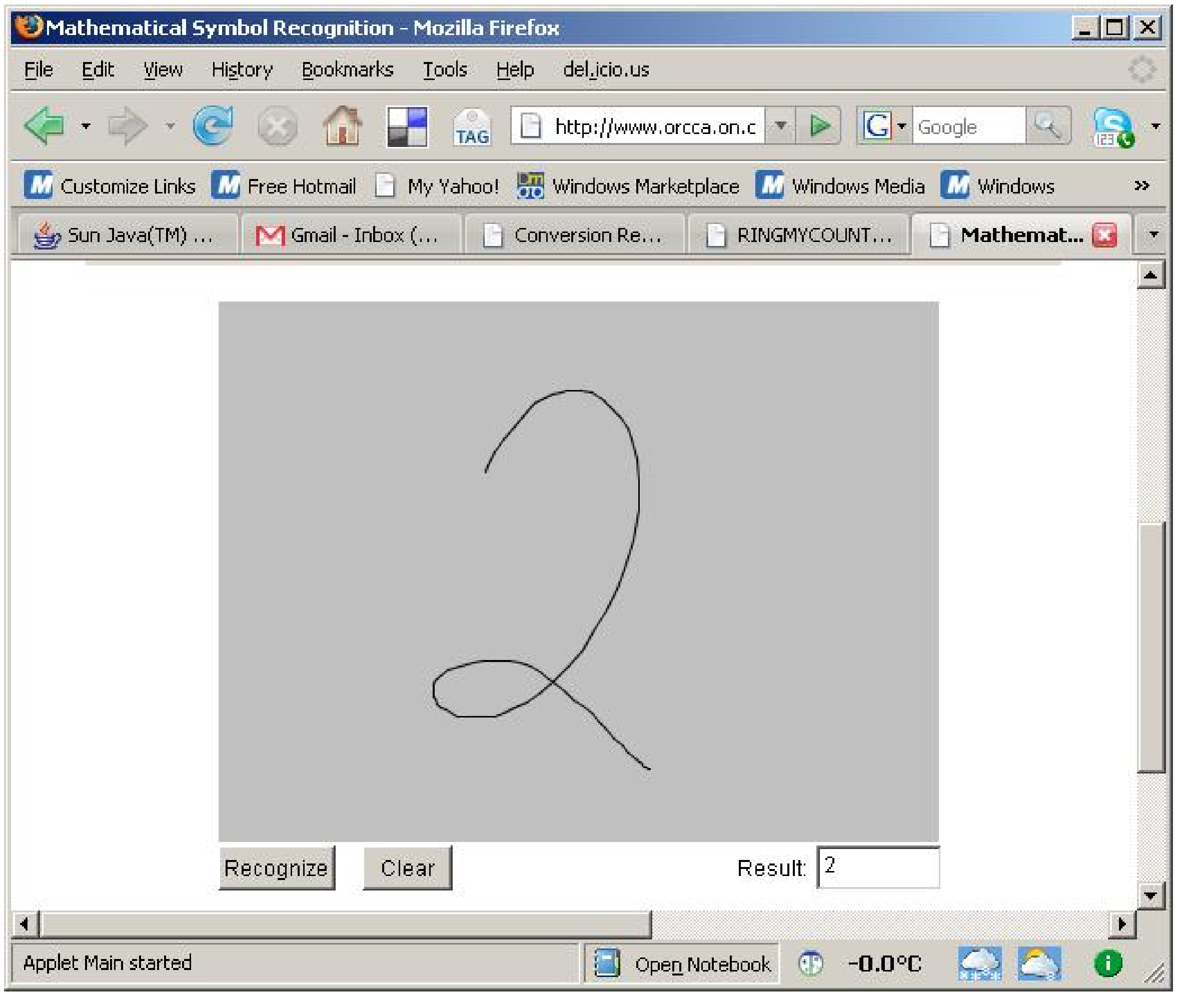


Figure 6.3: Web-based Symbol Recognition Interface

Chapter 7

Conclusion

7.1 Summary

In this work, we have developed algorithms and techniques for exchanging digital ink in heterogeneous collaborative environments and for transforming digital ink from one form to another depending upon the need. We have also presented an InkML-based protocol for allowing mathematical symbol recognition which can be useful in a collaborative environment. In doing so, we hope to have contributed something in the area of pen-based computing and also demonstrated various proofs of concepts of InkML.

We introduced various concepts of InkML including streaming and archival styles and shared canvas. We presented algorithms for doing conversion from streaming style to archival style and *vice-versa*. We also used these translators in a white-board sharing application to archive the ink conversation.

We discussed four different InkML-based techniques (client-oriented I, client-oriented II, server-oriented and hybrid) for sharing digital ink in a heterogeneous collaborative environment. We presented an in-depth analysis of all these techniques and also compared them. We found hybrid technique to be more suitable in practical scenarios as it enjoys the benefits of both client-oriented and server-oriented techniques and thus balances the load between the client and the server. We also compared InkML with other digital ink formats and showed how it is more flexible and efficient. We implemented a white-board sharing application using the hybrid technique.

We presented an InkML-based protocol for doing mathematical symbol recognition and a support vector machines based symbol recognition system. This can be integrated with the white-board sharing application to provide mathematical user interfaces.

7.2 Future Directions

There are a number of possible extensions that can be added to our system. One could add gesture commands such as delete ink traces, copy ink traces and so on to the system. For this, a set of symbols can be fixed as gesture symbols and these can be added to the training database. Necessary actions can be taken if the recognition result from the server is determined to be a gesture.

A structure recognition system can be added to the recognition server which when combined with the current symbol recognition module can form a complete mathematical handwriting recognition system. The protocol can be extended to support representation of mathematical structure in the recognition result so that the client can interpret the mathematical structure recognition result. The client could do computations by sending the recognized result to a computer algebra system such as Maple [CFG$^+$84] server and display the result in the ink canvas (may be broadcast to all other users sharing the white-board). For example, users could perform several operations like expanding an expression, simplifying an expression and so on.

This would provide a complete environment for doing mathematics in a collaborative environment which could be very helpful for example in a classroom teaching and distance education settings.

The current InkML-based protocol doesn't provide control over the symbol recognition algorithms. In future, this feature can be added at application level so that client can decide which algorithm to use for recognition. This would make the recognition system more flexible and user can try different algorithms depending upon their choice. The InkML-based protocol can also be used to combine results from several recognition servers. This can help to improve the accuracy of predictions.

Video communication capability can be added on top of the existing system to make it more useful. Currently, only the digital ink and the canvas background image are saved in InkML format. This can be extended by adding the feature to save all three mediums: digital ink, audio and video in a synchronized way.

Bibliography

[ASM08] Manoj Prasad A, Muthuselvum Selvaraj, and Sriganesh Madhvanath. Peer-to-peer ink messaging across heterogeneous devices and platforms. In *Compute 08: Proceedings of the 1st Bangalore Annual Compute Conference*, 2008. Article no. 25.

[BCH+04] Jay Beavers, Tim Chou, Randy Hinrichs, Chris Moffatt, Michel Pahud, Lynn Powers, and Jason Van Eaton. The learning experience project: Enabling collaborative learning with conferencexp. Technical report, Microsoft Research, Redmond, WA, USA, 2004. MSR-TR-2004-42.

[CFG+84] Bruce W. Char, Gregory J. Fee, Keith O. Geddes, Gaston H. Gonnet, Michael B. Monagan, and Stephen M. Watt. On the design and performance of the maple system. In *Macsyma Users' Conference*, pages 199–219, Schenectady, New York, General Electric Corporation, 1984.

[CFW06] Yi-Min Chee, Max Froumentin, and Stephen M. Watt (editors). *Ink Markup Language (InkML)*, October 2006. `http://www.w3.org/TR/InkML/` (valid on April 2, 2008).

[CL01] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. `http://www.csie.ntu.edu.tw/~cjlin/libsvm` (valid on April 2, 2008).

[Cor93] Slate Corporation. *JOT — A Specification for an Ink Storage and Interchange Format*, 1993. `http://unipen.nici.kun.nl/jot.html` (valid on April 2, 2008).

[dat] `http://www.graphics.cs.brown.edu/research/pcc/` (valid on April 2, 2008).

[Dur04] Kevin Durdle. Supporting mathematical handwriting recognition through an extended digital ink framework. Master's thesis, The University of Western Ontario, London, ON, Canada, 2004.

[FJe03] Jon Ferraiolo, Fujisawa Jun, and Dean Jackson (editors). *Scalable Vector Graphics (SVG)*, 2003. `http://www.w3.org/TR/SVG` (valid on April 2, 2008).

[Gro] W3C DOM Working Group. *Document Object Model Specification*. `http://www.w3.org/DOM/` (valid on April 2, 2008).

[Guy94] Isabella Guyon. Unipen 1.0 Format Definition. The Unipen Consortium, 1994. `http://www.unipen.org/dataformats.html` (valid on April 2, 2008).

[HL02] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, March 2002.

[KMA$^+$08] Birendra Keshari, Sriganesh Madhvanath, Manoj Prasad A, Muthuselvum Selvaraj, and Stephen M. Watt. Sharing digital ink in heterogeneous collaborative environments. In *International Conference on Frontiers in Handwriting Recognition*, 2008. (to appear).

[KW07a] Birendra Keshari and Stephen M. Watt. Hybrid mathmematical symbol recognition using support vector machines. In *Proceedings of International Conference on Document Analysis and Recognition*, pages 859–863, 2007.

[KW07b] Birendra Keshari and Stephen M. Watt. Streaming-archival inkml conversion. In *Proceedings of International Conference on Document Analysis and Recognition*, pages 1253–1257, 2007.

[LM03] Andrew P. Lenaghan and Ron R. Malyan. Xpen: An xml based format for distributed online handwriting recognition. In *Proceedings of International Conference on Document Analysis and Recognition*, pages 1270–1275, 2003.

[Map07] MapleSoft. *Maple 11 User Manual.* MapleSoft, Waterloo, Ontario, Canada, 2007.

[Mic04] Microsoft. *Microsoft Tablet PC SDK Documentation*, 2004. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tpcsdk10/lonestar/devcenter/tbidxindex.asp` (valid on April 2, 2008).

[NG07] Jonathan Neddenriep and William G. Griswold. Riverink-an extensible framework for multimodal interoperable ink. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, pages 258b–258b, 2007.

[NWSS05] Hai Ning, John R. Williams, Alexander H. Slocum, and Abel Sanchez. Inkboard - tablet pc enabled design-oriented learning. In *International Conference on Computers and Advancaed Technology in Education*, pages 154–160, 2005.

[osc99] Jabber open-source community. *XMPP standards foundation, Extensible Messaging Presence Protocol (XMPP)*, 1999. `http://www.xmpp.org` (valid on April 2, 2008).

[Pla99] John C. Platt. Probabilistic outputs for support vector machines and comparison to regularized methods. In A. Smola, P. Bartlett, B. Schlkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[PMMH93] Elin R. Pederson, Kim McCall, Thomas P. Moran, and Frank G. Halasz. Tivoli: an electronic whiteboard for informal workgroup meetings. In *Human Factors in Computing Systems, INTERCHI*, pages 391–398, 1993.

[Rat03] Eugene H. Ratzlaff. Methods, report and survey for the comparison of diverse isolated character recognition results on the unipen database. In *Proceedings of International Conference on Document Analysis and Recognition*, pages 623–628, 2003.

[Rea06a] Ignite RealTime. *Jingle Source Code*, 2006. `http://svn.igniterealtime.org/svn/repos/smack/trunk` (valid on April 2, 2008).

[Rea06b] Ignite RealTime. *Smack API Documentation*, 2006. `http://www.igniterealtime.org/builds/smack/docs/latest/documentation/index.html` (valid on April 2, 2008).

[SW06] Elena Smirnova and Stephen M. Watt. Combining prediction and recognition to improve on-line mathematical character recognition. Technical report, University of Western Ontario, 2006. `http://www.orcca.on.ca/TechReports/TR-06-06` (valid on April 2, 2008).

[SW08] Elena Smirnova and Stephen M. Watt. Context-sensitive mathematical character recognition. In *Proceedings of International Conference on Frontiers in Handwriting Recognition*, 2008. (to appear).

[TR05] Ernesto Tapia and Raul Rojas. Recognition of on-line handwritten mathematical expressions in the e-chalk system-an extension. In *Proceedings of International Conference on Document Analysis and Recognition*, pages 1206–1210, 2005.

[Vap98] Vladimir Vapnik. *Statistical Learning Theory.* John Wiley & Sons, New York, 1998.

[Wat07] Stephen M. Watt. New aspects of InkML for pen-based computing. In *Proceedings of International Conference on Document Analysis and Recognition*, pages 457–460, 2007.

[WLW04] Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 99(5):975–1005, August 2004.

[Wu04] Xiaojie Wu. Achieving interoperability of pen computing with heterogeneous devices and digital ink formats. Master's thesis, The University of Western Ontario, London, ON, Canada, December 2004.

VITA

Name:	Birendra Keshari
Born:	Kalaiya, Bara, Nepal, 1982
Education and Degree:	Kathmandu University Dhulikhel, Kavre, Nepal 2000-2004 B.Eng in Computer, CGPA: 3.83/4
Work Experience:	Internship, 2006 HP Labs, Bangalore, India
	Teaching and Research Assistant, 2006 - 2008 Dept. of Computer Science University of Western Ontario London, ON, Canada
	System Analyst, 2005 - 2006 Information and Language Processing Research Lab Dept. of Computer Science Kathmandu University Dhulikhel, Kavre, Nepal
	Software Developer (part time), 2005 - 2006 DASS Pvt. Ltd Kathmandu, Nepal

Publications:

- Birendra Keshari, Sriganesh Madhvanath, Manoj A Prasad, Muthuselvum Selvaraj, and Stephen M. Watt. Sharing digital ink in heterogeneous collaborative environments. In *International Conference on Frontiers in Handwriting Recognition*, Montreal, Canada, 2008 [to appear].

- Birendra Keshari and Stephen M. Watt. Hybrid mathematical symbol recognition using support vector machines. In *Proceedings of International Conference on Document Analysis and Recognition*, Curitiba, Brazil, 2007.

- Birendra Keshari and Stephen M. Watt. Streaming-archival inkml conversion. In *Proceedings of International Conference on Document Analysis and Recognition*, Curitiba, Brazil, 2007.

- Yogendra P. Yadava, Govinda Raj Bhattarai, Sanat Kumar Bista, Birendra Keshari and Jagannath Bhatta. Envisioning Machine Translation for the New Millennium: Outlines of Preliminary Steps in Nepal. *Contemporary Issues in Nepalese Linguistics, ed. by Yogendra P. Yadava et al.* pages 429-443, Kathmandu: Linguistic Society of Nepal, 2005.

- Birendra Keshari and Sanat Kumar Bista. UNL Nepali Deconverter. In *Proceedings of CALIBER*, Kochi, India, 2005.

- Birendra Keshari, Jagannath Bhatta and Sanat Kumar Bista. Nepali Part-of-Speech Guesser and its Application in Lexicon Building. In *Proceedings of International Conference on Natural Language Processing*, IIT Kanpur, India, 2005.