Sharing Digital Ink in Heterogeneous Collaborative Environments

Birendra Keshari[*] *Muthuselvam Selvaraj*[+] *Manoj Prasad A*[+]
Sriganesh Madhvanath[+] *Stephen M. Watt*[*]
[*]U. Western Ontario, London Ontario, Canada
[+]Hewlett-Packard Laboratories, Bangalore, India
{muthuselvam.selvaraj,srig}@hp.com, {bkeshari,watt}@csd.uwo.ca

Abstract

We present techniques that allow sharing of standardized digital ink in a heterogeneous collaborative environment. We explore the use of W3C InkML (Digital Ink Markup Language) and show how it is more flexible and suitable than other digital ink formats for this purpose. We discuss different possible digital ink sharing schemes using InkML and describe pros and disadvantages of each. We present our implementation of a shared whiteboard collaboration system for sharing digital ink across different devices and platforms. We discuss several usage scenarios such as collaborative document annotation and map annotation etc in which our system can be used for sharing digital ink in an efficient manner.

Keywords: collaborative inking, shared canvas, InkML

1. Introduction

It has long been established that pen-based input methods provide more natural and convenient ways to interact with machines for certain tasks. One of the most useful and important applications of pen input is whiteboard sharing (digital ink messaging in a collaborative environment), wherein several users collaborate by using pen-enabled devices to write or draw on a shared virtual whiteboard which may be blank or may present some shared information such as a map, image or a document. Whiteboard sharing is very useful in scenarios such as class room teaching, distance education, work-group meeting, collaborative document annotation, amongst others. 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 devices, such as channel properties, screen resolution and so on.

In early electronic whiteboard applications such as Tivoli [1], which was developed for work-group meetings, interoperability was not an area of focus. A more recent example is InkBoard [2], a collaborative sketching application based on Microsoft's ConferenceXP research platform [5] designed for Tablet PCs. InkBoard enables design teams to interact with each other by using real-time strokes which are streamed in Microsoft's proprietary Ink Serialized Format(ISF) [12]. This limits its use to Windows environments where ISF is natively supported.

For flexible interchange of digital ink data between applications on different platforms, it is critical that digital ink be stored in an open and standard format. The best known open format, UNIPEN [3], is very focused on handwriting recognition and not optimized for operations such as real time digital ink transmission, rich ink annotation and so on. On the other hand, W3C InkML [4], a draft standard developed by the W3C Multi Modal Interaction(MMI) Working Group, is an open standard and provides robust structures for representing ink data and the inking environment. InkML is designed to support collaborative environments with heterogeneous devices and platforms. Being an XML based language, it offers greater flexibility to application developers, e.g. application specific information can be easily added to digital ink files. InkML facilitates complete and accurate representation of digital ink by allowing capture of recording information such as device characteristics, pen tilt, pen pressure and so on. Besides other benefits, it provides better support for streaming and sharing digital ink.

In this paper, we discuss techniques for sharing digital ink in a heterogeneous collaborative environment using InkML. This significantly extends initial work by some of the authors on peer-to-peer collaboration using digital ink [10]. Among solutions aimed at supporting interoperability across platforms and devices, XEP-0113 [7], an extension protocol to XMPP [11], proposes SVG [8] to represent ink messages. The RiverInk Framework [6] proposes the use of a subset of InkML (trace and brush) to represent Ink messages for interoperability. RiverInk proposes sending ink data simultaneously in PNG image and native Ink format along with InkML, for interoperabil-

ity between heterogeneous devices (including non pen-enabled devices), which may work well for LAN environments but may be too bulky for mobile networks. Further, these approaches focus only on the capture of X & Y channels of ink stroke data, and do not capture all relevant context information, or device capabilities and attributes such as additional channels, screen size and resolution.

We describe the streaming style of InkML in Section 2. Section 3 describes the *shared canvas* notion of InkML, which is central to our discussion. We present different techniques for sharing digital ink in Section 4, and our implementation in Section 5. Conclusions and future directions are presented in the final section.

2. Streaming InkML

InkML allows digital ink to be stored in two styles: *streaming* and *archival*. The two are semantically equivalent and can be converted from one to another without any loss of information. Algorithms to do such conversion have been presented elsewhere [9]. The only difference between these two styles is that each of these allow certain operations more directly, and hence more efficiently, than the other. For example, streaming InkML provides more direct support for streaming ink data resulting is lower overhead on the wire, whereas archival InkML provides more direct support for operations such as search and retrieval, annotations and so on.

Regardless of style, InkML models each ink capture device as an *ink source*, and specifies a *trace format* which in turn describes the format of samples captured by the digitizer, as a list of Channels such as X, Y and F (force) with dimension attributes. Digital ink is represented as a set of *traces* conforming to the trace format, each trace being a sequence of pen/finger position samples (typically at a uniform sampling rate) demarcated by consecutive pen-down and pen-up events.

Streaming InkML is based on the concept of "current context" which can be altered by `<context>` elements. At any particular instance in time, the ink application can maintain an associated current context which has various aspects such as canvas (`<canvas>`), canvas transform (`<canvasTransform>`), trace format (`<traceFormat>`), ink source (`<inkSource>`), brush (`<brush>`) and time stamp (`<timestamp>`). Initially, all of these contextual elements have a default value (*Default Context*). An event such as a change in brush that occurs as a result of one user's action can be mapped to the relevant contextual elements and sent to all others. Ink data may then be sent after its context has been established. Thus, a stream of ink data interspersed with context events may be transmitted in an incremental manner.

With this model, a receiver can easily maintain the current context of the sender besides maintaining its own local context. Whenever a new contextual element is received, its values suitably modify (or override, as appropriate) the old values. Context elements are sent only when there is a context change, which helps in the streaming scenario by reducing data on the wire. For instance, if the current brush's color is already red and a red trace in scribbled then it is sufficient to only send the trace data and not the brush color information, as the receiver is maintaining the sender's current context, and knows what the sender's current brush color is. This idea is illustrated in Figure 1.

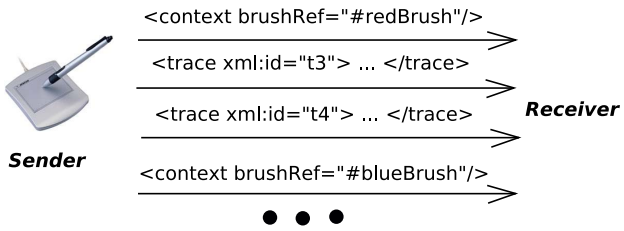

Figure 1. Ink and context changes communicated using streaming InkML

In InkML, a change in context can be expressed by making contextual elements a direct child of `<context>`. This has the effect of altering the current context as well as defining a contextual element which can be referenced subsequently. Another method to change a context is by making references to contextual elements using referencing attributes (e.g. `brushRef`) of `<context>`. This allows reuse and extension of previously defined contextual elements.

3. 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 sharing scenario, *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 `reverse canvas transform` contains information about the inverse mapping. If *inverse can-*

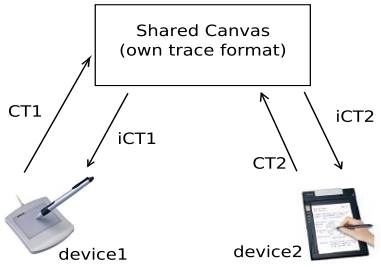

Figure 2. An example of shared canvas

vas 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 shows an example of shared canvas (CT denotes forward canvas transform and iCT denotes the inverse).

4. Sharing Digital Ink

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. In InkML, this is achieved using the `<canvas>` and `<canvasTransform>` elements discussed in the previous section. The element `<canvas>` provides a shared virtual space for cooperation of ink applications by supporting an intermediate representation of digital ink. The main advantage of such an intermediate representation is that only $(n + m)$ conversions are required to convert from *m* sources to *n* targets instead of $(n * m)$, as illustrated in Figure 3.

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, 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 pros and disadvantages. Depending on the location at which the transformations are applied, we may categorize these schemes as

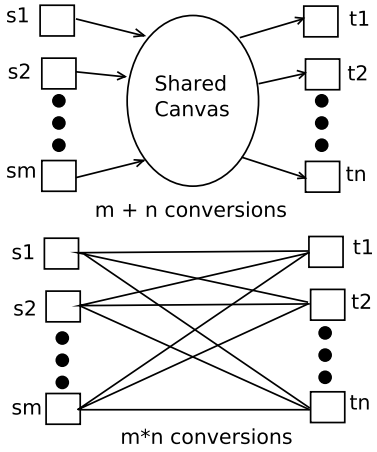

Figure 3. Sharing ink with shared canvas(upper) and without it(lower).

(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.

4.1. Client-oriented Scheme

In this scheme, the client is responsible for keeping record of information (trace format and canvas transform) about other clients and doing 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.

4.1.1. Scheme I

In this scheme, the forward transformation is applied only after the client receives the digital ink. The following sequence of activities occurs whenever a client c_i joins the session or when its pen-device is changed:

- Server sends shared canvas information to client c_i (`<canvas>` with `<traceFormat>` as a child to it.)
- Client c_i computes its `<canvasTransform>` by comparing its own trace format with the trace format of canvas. Client sends its ink source definition containing information about its trace format and channel properties (`<definitions>` containing `<traceFormat>` and `<channelProperties>` as children to `<inkSource>`) followed by the canvas transformation to the server.
- Server broadcasts these to all the clients (may be except c_i in which case it generates them internally).

The above process allows each client to know the canvas transformations of all other clients. A client c_i can send the contextual element

```
<context traceFormatRef='#tf-ci'
         canvasTransform='#ct-ci'/>
```

followed by its trace data, where 'tf-ci' and 'ct-ci' are respectively its trace format and canvas transform. This is then broadcast by the server to all clients. Each client c_j can thereafter perform forward mapping (from client c_i to canvas) as well as inverse mapping (from canvas to its own) of ink.

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 and it performs no other processing.

Disadvantages: There are a number of context switches as each client alters the context before sending its trace data. This can increase the data on the wire. In addition, performing ink mapping in both directions may not be feasible for resource constrained clients such as PDAs, smart phones and so on.

4.1.2. Scheme II

This is very similar to the previous scheme. The only difference is that each client applies the forward canvas transformation to its ink data before sending it to the server. Thus, the ink data broadcast by the server is always in the format of shared canvas.

Server maintains a global context to keep track of changes such as brush color. 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 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 still simple. Since changes in context do not occur very often, the size of the data on the wire is smaller.

Disadvantages: There could be loss of precision. As with the previous scheme, this scheme also may not be efficient for resource constrained clients.

4.2. 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:

- Client sends its ink source definition containing information about its trace format and channel properties (`<definitions>` containing `<traceFormat>` and `<channelProperties>` as children to `<inkSource>`).
- Server computes the canvas transformations 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). It sends `<traceFormat>` which is same as trace format of client c_i to client c_i.

When the server receives trace data from client c_i, it uses CT Table and finds its forward mapping information and then applies the transformation to convert the trace data to the shared canvas format. The shared canvas is private to the server and clients are completely unaware of it. In order to send ink data to a client, it uses CT Table again and finds the client's corresponding inverse mapping, applies it and sends the transformed data to the client. Thus, it applies transformations selectively for each client and then sends the transformed ink data to each client separately. Other contextual elements (e.g. change in brush) are maintained globally at the server as explained in 4.1.2.

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.

4.3. Hybrid Scheme

This scheme can be thought of as a blend of two previous schemes. It helps retain some of the the benefits of the previous schemes and strike a better balance of server load and complexity on the 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 occur whenever a client c_i joins the session or when its pen-device is changed:

- Server sends shared canvas information to client c_i (`<canvas>` with `<traceFormat>` as a child to it.)
- Client c_i computes its `<canvasTransform>` by comparing its own trace format with the trace format of the canvas. Client sends its

ink source definition containing information about its trace format and channel properties (`<definitions>` containing `<traceFormat>` and `<channelProperties>` as children to `<inkSource>`) followed by the canvas transformation to the server.

- Server keeps a record of canvas transformations obtained from each client in the CT Table. Server changes the current trace format by broadcasting a trace format identical to the trace format of the canvas.

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 its 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. This scheme also involves the server maintaining a global current context as explained in 4.1.2.

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 the forward canvas transform.

The advantages and disadvantages of all the schemes described are summarized in Table 1. The choice of scheme for a particular implementation depends on various factors such as the network bandwidth and the nature of the server and clients.

Table 1. Summary of the Schemes

Scheme	Data size on the wire	Context data size in client	Processing load on client	Processing load on server
Client I	High	High	High	Low
Client II	Low	Low	Medium	Low
Server	Low	Low	Low	High
Hybrid	Low	Low	Low	Medium

5. Implementation

We have implemented a system for collaborative inking using the ideas outlined above. Our implementation uses the *Hybrid Scheme* described in section 4.3 as it enjoys the benefits of the other two schemes and balances the processing load between the server and clients. It has lesser load on clients than *client-oriented scheme II* which is crucial for systems with resource constrained clients. For our implementation, we chose XMPP [11] (popularly known as Jabber), a open IETF standard as the base protocol for transporting InkML data over the network. We

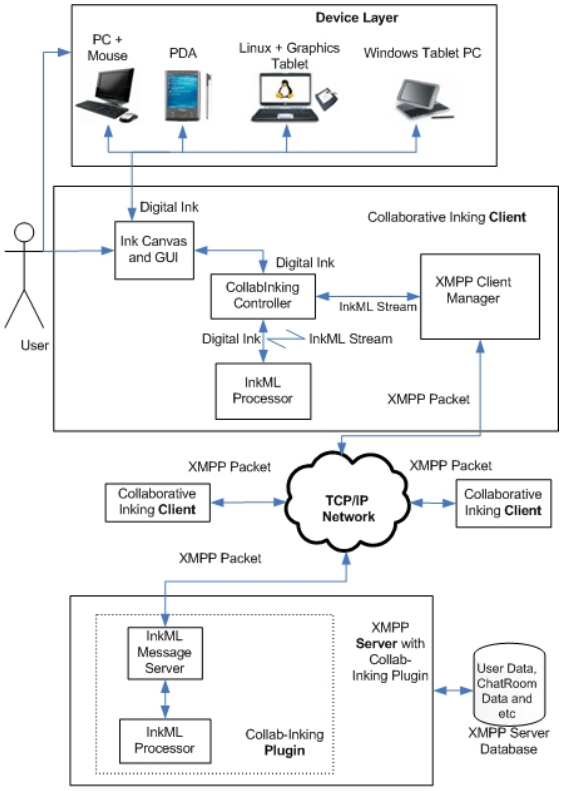

Figure 4. Solution architecture

have extended the XMPP server by developing an InkML plug-in and use it as the server component.

An InkML extension (XMPP Client Manager) developed using Smack API [13] is used by client applications to communicate InkML messages with the server. All of the platform independent code is written in Java, and platform dependent code in C/C++, and these are glued together using the Java Native Interface (JNI). Currently the application supports both Linux and Windows platforms. The solution architecture is depicted in Figure 4.

The trace format of the shared canvas used in our implementation has X, Y and F (pressure) channels, which covers the channel capabilities of the clients we are using presently. The values of channels X and Y of shared canvas are in absolute length units (millimeters) and we assume their resolution to be infinite. Channel properties of the ink source are used to compute the forward and inverse canvas transforms. For example, if channel X of a particular ink source has a spatial resolution of 10 ppi (points per inch), then the forward canvas transform for that source would involve a multiplication factor of 2.54 (to convert from inches to millimeters).

We have developed a generic cross-platform API to capture digital ink (in InkML format) that is loosely cou-

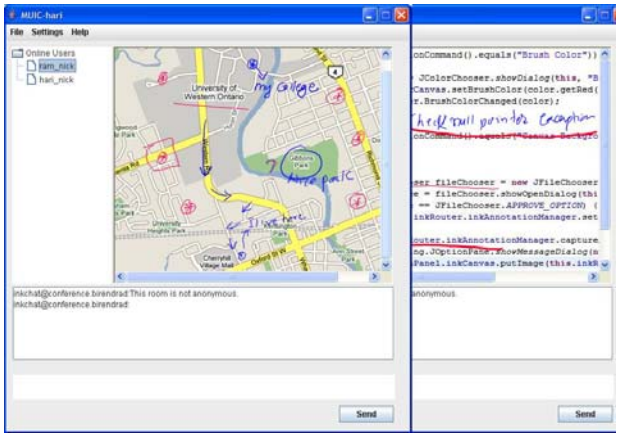

Figure 5. Annotating maps and document images using our client GUI

pled with the rest of the components. This API supports various pen devices such as Tablet PC, graphics tablets as well as mouse devices. It is used by the "Ink Canvas" component in the client application which captures and render digital ink on the client application GUI. The application GUI allows images of documents, maps and photographs to be set as the canvas background. This allows us to do several interesting and useful tasks such as collaborative document annotation, collaborative map annotation and so on (Figure 5). Besides ink, our implementation also features text and audio as additional modalities. Text messaging support is available by default in the core XMPP implementation. We have utilized the Smack Jingle API [14] for audio. The inking session can be saved in archival InkML format [9] for future reference. References to background images are recorded in archival InkML using InkML's built-in `annotation XML` elements.

6. Conclusions and Future Work

In this paper, we showed how InkML can be used for sharing digital ink in heterogeneous environments in different ways. We presented our implementation and discussed how it can be useful in various application scenarios. We compared InkML with other digital ink formats and showed how InkML is more efficient and flexible than other digital ink formats in general for the task of cross-platform whiteboard sharing.

There are a number of interesting directions we wish to pursue in the future. We plan to extend the coverage of ink sources to include Windows Mobile PDAs/smart phones, Smartboards and graphics tablets for Linux. We also intend to investigate collaborative inking in a WAN setting and using mobile devices, and related issues such as compression and encryption of InkML data, and timestamp-based synchronization across ink sources. We also plan to explore the use of collaborative inking in combination with other modalities including voice, video, and images for specific usage scenarios. The general space of classroom and distance education, and collaboration tools enabled by electronic whiteboards, and portable devices such as mobile phones is clearly a very rich area for explorations in collaborative inking across platforms.

References

[1] Elin Ronby Pederson, Kim McCall, Thomas P. Moran, Frank G. Halasz, "Tivoli: an electronic whiteboard for informal workgroup meetings", *Human Factors in Computing Systems, INTERCHI*, IOS Press, Amsterdam, Netherland, 1993, pp 391-398

[2] Hai Ning, John R. Williams, Alexander H. Slocum and Abel Sanchez, "InkBoard - Tablet PC Enabled Design-Oriented Learning.", *Advanced Technology for Learning*, 2005.

[3] Isabella Guyon, *Unipen 1.0 Format Definition*, The Unipen Consortium, 1994.

[4] Yi-Min Chee, Max Froumentin and Stephen M. Watt (editors), Ink Markup Language (InkML), October, 2006, `http://www.w3.org/TR/InkML/`

[5] Jay Beavers, Tim Chou, Randy Hinrichs, Chris Moffatt, Michel Pahud, Lynn Powers, Jason Van Eaton, *The Learning Experience Project: Enabling Collaborative Learning with ConferenceXP*, MSR-TR-2004-42, Microsoft Research, Redmond, WA, USA, 2004.

[6] Jonathan Neddenriep, William G. Griswold, "RiverInk–An Extensible Framework for Multimodal Interoperable Ink", *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, Computer Society Press, 2007, p. 258b

[7] Huib-Jan Imbens, XEP-0113: Simple Whiteboarding, the XMPP Standards Foundation, 2003, `http://www.xmpp.org/extensions/xep-0113.html`

[8] Jon Ferraiolo, FUJISAWA Jun and Dean Jackson (editors), Scalable Vector Graphics (SVG), January 2003, `http://www.w3.org/Graphics/SVG/`

[9] Birendra Keshari and Stephen M. Watt, "Streaming-Archival InkML Conversion", *International Conference on Document Analysis and Recognition*, Curitiba, Brazil, September, 2007.

[10] Manoj Prasad A, Muthuselvam Selvaraj and Sriganesh Madhvanath "Peer-to-peer Ink Messaging across Heterogeneous Devices and Platforms", *Compute '08: Proceedings of the 1st Bangalore annual Compute conference*, Bangalore, India, January, 2008.

[11] XMPP standards foundation, Extensible Messaging and Presence Protocol (XMPP), Jabber open-source community, 1999, `http://www.xmpp.org/`

[12] Microsoft, *Tablet PC SDK Documentation*, Microsoft, 2004.

[13] Ignite RealTime, *Smack API Documentation*, Ignite RealTime, 2006.

[14] Ignite RealTime, *Jingle Source Code*, Ignite RealTime, 2006.