Peer-to-peer Ink Messaging across Heterogeneous Devices and Platforms

Manoj Prasad A, Muthuselvam Selvaraj and Sriganesh Madhvanath
HP Laboratories India, Bangalore
HPL-2007-202
December 20, 2007*

Digital Ink Markup Language, Ink Messaging, XMPP

Pen and touch interfaces for personal and shared devices are becoming increasingly relevant today, in the context of mobility and ease of use. A key capability enabled by pen-interfaces is that of messaging using handwritten, as opposed to text messages. Not only are ink messages easier to enter than text messages (especially when a full keyboard is not present), they allow the incorporation of other elements such as drawings and doodles into instant messaging. However since ink formats are typically platform-specific and proprietary, messaging across different platforms such as Tablet PCs and Linux-based PDAs poses an interoperability problem. In this paper, we show how Ink Markup Language (InkML), an open draft standard from W3C, can be used to address this problem. In particular, we propose an Ink messaging protocol, and a system architecture for implementing the protocol operations. We have implemented this protocol as an extension to the Extensible Messaging and Presence Protocol (XMPP), an open IETF standard.

Internal Accession Date Only
Approved for External Publication
To be published and presented at ACM COMPUTE 2008, Bangalore

Peer-to-peer Ink Messaging across Heterogeneous Devices and Platforms

Manoj Prasad A[o], Muthuselvam Selvaraj* and Sriganesh Madhvanath
OST/HP Labs India and HP GDAS*
Bangalore, India
{manoj.prasad, muthuselvam.selvaraj, srig}@hp.com

ABSTRACT
Pen and touch interfaces for personal and shared devices are becoming increasingly relevant today, in the context of mobility and ease of use. A key capability enabled by pen-interfaces is that of messaging using handwritten, as opposed to text messages. Not only are ink messages easier to enter than text messages (especially when a full keyboard is not present), they allow the incorporation of other elements such as drawings and doodles into instant messaging. However since ink formats are typically platform-specific and proprietary, messaging across different platforms such as Tablet PCs and Linux-based PDAs poses an interoperability problem. In this paper, we show how Ink Markup Language (InkML), an open draft standard from W3C, can be used to address this problem. In particular, we propose an Ink messaging protocol, and a system architecture for implementing the protocol operations. We have implemented this protocol as an extension to the Extensible Messaging and Presence Protocol (XMPP), an open IETF standard.

Categories and Subject Descriptors
H.4.3 [**Communications Applications**]: Messaging using handwritten messages.

Keywords
Digital Ink Markup Language, Ink Messaging, XMPP

1. INTRODUCTION
Instant Messaging has overtaken email as the most common application on the internet, and is rapidly becoming available on a number of portable devices and platforms. In addition to its social uses, messaging is also finding widespread use in domains ranging from customer support and remote healthcare, to active learning in classrooms. Digital ink is an important modality for messaging along with text, images and voice. The incorporation of digital ink in the popular IM application greatly enhances the power of the application in more ways than one. For instance, a user can communicate rapidly using handwritten messages in any language as well as drawings, without having to learn the text input mechanism of the specific device (Figure 1). Ink can also be overlaid on text and graphic content to enable new kinds of collaboration and social networking experiences. For example, one can imagine a group of friends using ink messages to interact over a city map on their GPS-enabled mobile devices, or over textbook content in a classroom.

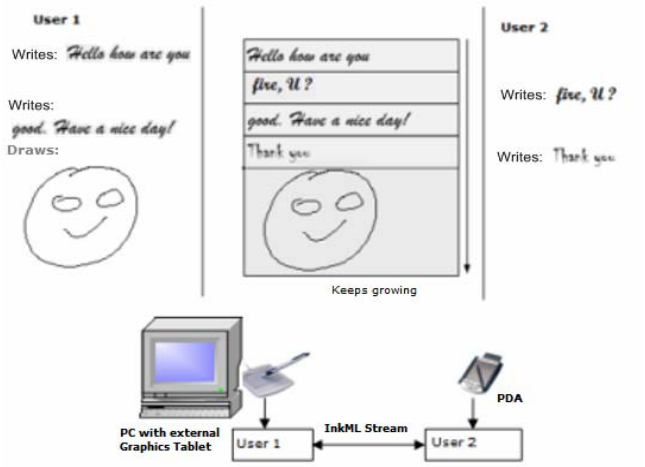

Figure 1: Peer-to-Peer ink messaging scenario across different devices

Unfortunately since ink formats are typically platform-specific and proprietary; messaging across different platforms such as Tablet PCs and Linux-based PDAs poses an interoperability problem. For example, User 1 may, have a Windows PC with an external graphics tablet as digitizer, while User 2 may use a Linux PDA with built-in digitizer (Figure 1). In order to provide digital ink-based instant messaging capability in a heterogeneous environment, one must necessarily address the following issues: (i) Representation of digital ink captured so that it may be understood by different ink-enabled platforms, (ii) Differences with respect to the constituent channels of ink data (such as X, Y, pressure, and so forth) captured by the devices, and differences in the resolution and range of their channel values, (iii) Differences in form factor and display size, and implications for rendering of ink, and (iv) An efficient protocol for messaging of ink messages that deal with the above differences.

The paper is organized as follows. Section 2 of the paper briefly introduces the two standards used in our solution: InkML and XMPP. Section 3 of the paper describes related work. Section 4 describes the ink messaging protocol we have devised.

Compute 2008, Jan 18-20, Bangalore, Karnataka, India. © 2008 ACM
ISBN 978-1-59593-950-0/08/01...$5.00

[o] Formerly intern at HP Labs India

Architectural and implementation details of our solution are provided in Section 5. The paper concludes with a discussion of next steps and future research directions.

2. INK MESSAGING USING INKML and XMPP

Our proposed solution for peer-to-peer ink messaging addresses issues identified earlier, by leveraging Digital Ink Markup Language (InkML) [1], a draft specification from W3C for the platform and device-independent description of digital ink. InkML is an XML based markup language for representing digital ink envisioned as an open alternative to the proprietary ink data formats from device vendors. InkML is easily extensible to meet application specific requirements. In addition to allowing accurate and platform-independent representation of the various "channels" or "dimensions" of digital ink such as position, pressure, color, width, and so on, InkML includes elements for grouping ink, transforming ink in various ways, and attaching metadata and semantic interpretation to ink. InkML also supports archival and streaming modes of using digital ink. The reader is referred to the most recent draft specification of InkML [1] for details of these, and the other core components of InkML.

For our implementation, we chose the Extensible Messaging and Presence Protocol (XMPP) [2] as the basic protocol for implementing our Ink messaging protocol. XMPP is an open XML communications protocol developed by the Jabber open-source community in 1999, formalized by the IETF in 2002-2004, and continuously extended through the standards process of the XMPP Standards Foundation. The core XMPP protocol provides the basic instant messaging and presence features. Beyond instant messaging, it provides a generalized, extensible framework for incrementally exchanging XML data which can be applied to develop a variety of distributed application services.

We have implemented our ink messaging protocol as an XMPP Extension Protocol (XEP) [3] to support InkML based Ink messages.

3. RELATED WORK

Messaging applications with support for digital ink messages may now be found on ink-enabled platforms such as TabletPC; however their realm of applicability is limited in scope by the proprietary formats of digital ink used for ink messages (e.g. Microsoft's Ink Serializable Format (ISF)), and the assumptions they make regarding the device's capabilities (such as support for .NET or the ability to capture certain channels of digital ink at a certain sampling rate and resolution. We believe that while ink as a data type has been investigated deeply in the context of Tablet PCs, the use of it for other devices and platforms is likely to grow.

Among solutions aimed at supporting interoperability across platforms and devices, XEP-0113 [4] is an extension to XMPP that uses the "path" element of Scalar Vector Graphics (SVG) to represent ink messages. It was aimed at providing basic whiteboard capabilities for XMPP based (chat) applications. This extension supports only X and Y channels and does not address differences in digitizer capabilities.

The RiverInk Framework [5] proposes the use of a subset of InkML [1] (trace and brush) to be used as the common intermediate ink data format for interoperability. It does not provide a messaging solution, but adopts a multi-part xml format containing (i) the ink data in PNG image format (suitable for rendering on non-ink aware platforms), (ii) native platform ink format (e.g. ISF in the case of Windows) as well as in (iii) InkML format, for interoperability between heterogeneous devices (including non pen–enabled devices). This solution may work well for LAN environments but may be too bulky for mobile networks given the multiple representations that need to be transmitted. Further, the approach focuses only on the capture of X & Y channels of ink stroke data, and does not capture all relevant contextual information, or device capabilities and attributes such as additional channels, screen size and resolution.

4. INK MESSAGING PROTOCOL

In this section, we briefly describe the protocol we have developed for peer to peer ink messaging. As mentioned earlier, our implementation uses XMPP, however a standard protocol such as HTTP, or a custom protocol may instead be used as the underlying protocol to transport ink messages. For reasons of brevity, we do not describe the common processes associated with messaging applications, such as user authentication and user state management.

The protocol has two phases of operation: Initialization Phase and Data Transfer phase. The protocol uses *<inkMLMessage>* elements to wrap the InkML data fragments corresponding to ink messages. The aggregation of all InkML data fragments for the entire messaging session including both of these phases is structured as a single InkML document.

4.1 Initialization Phase

The ink messaging application is intended to work across a wide variety of ink-enabled platforms such as PDAs, desktop PCs with graphics tablet peripherals, Tablet PCs and Smart phones. As described earlier, the digital ink generated by these devices differs with respect to the channels of ink captured by their respective digitizers, and the resolution and range of their channel values. This mandates careful transformation of the ink data from one device so that it may be consumed by another device. These differences are resolved in the initialization phase, as follows

4.1.1 Trace Format (Channel) Negotiation

To begin with, the initiator and the recipient of the messaging session exchange their <inkSource> information. Then, based on the <inkSource> information received, they exchange a <context> element containing a <traceFormat> having the common subset of channels supported by both. This <traceFormat> defines the format of the <trace> data to be exchanged in the Ink Messages. The idea here is to exchange only those channels of ink that both clients can capture and interpret. The <inkSource> and <traceFormat> elements are wrapped in <definitions> blocks and are persisted throughout the messaging session. Details of the ink messages exchanged in this phase are shown in Figure 2.

Initiator describes its Trace Format and supported channels:

```
<inkMLMessage>
 <definitions>
  <inkSource id = "src-A">
```

```
    <traceFormat>
      <channel name="X" type= "integer" max = "300" min ="0"
units="mm"/>
      <channel name="Y" type= "integer" max = "150" min ="0"
units = "mm"/>
      <channel name="F" type= "integer" max = "1024" min="0"
units="dev"/>
    </traceFormat>
  </inkSource>
 </definitions>
</inkMLMessage>
```

Recipient replies with its InkSource Definition:

```
<inkMLMessage>
  <definitions>
    <inkSource id = "src-B">
      <traceFormat>
        <channel name="X" type= "integer" max = "200" min ="0"
units="mm"/>
        <channel name="Y" type= "integer" max = "100" min ="0"
units = "mm"/>
        <channel name="T" type= "integer" max = "1000" min="0"
units="dev"/>
      </traceFormat>
     </inkSource>
 </definitions>
</inkMLMessage>
```

Initiator sends its context with the derived common TraceFormat:

```
<inkMLMessage>
  <context id = "ctx-A">
    <traceFormat>
      <channel name="X" type= "integer" max = "300" min ="0"
units="mm"/>
      <channel name="Y" type= "integer" max = "150" min ="0"
units = "mm"/>
    </traceFormat>
  </context>
</inkMLMessage>
```

Recipient sends its context with common TraceFormat:

```
<inkMLMessage>
  <context id = "ctx-B">
    <traceFormat>
      <channel name="X" type= "integer" max = 200" min ="0"
units="mm"/>
      <channel name="Y" type= "integer" max = "100" min ="0"
units = "mm"/>
    </traceFormat>
  </context>
</inkMLMessage>
```

Figure 2: Initialization Phase – channel negotiation

4.2 Data Transfer Phase

Ink data corresponding to a single message must contain the context of the sender, any brush changes (color and stroke-width) and a collection of <trace> elements. If the application performs layout analysis on digital ink data in order to group related traces into logical units such as a word or drawing unit, then the traces are grouped using <traceGroup> elements and the metadata information is captured using <annotationXML> elements. The Layout analysis is implemented using a heuristic algorithm, explained below.

The high shift in y-coordinate position of consecutive traces in different lines is used to detect the line break. The minimum and maximum x-coordinate position of each trace is found. The space between consecutive traces in the same line is found as the difference between the minimum x-coordinate position of the second trace to the maximum x-coordinate position of the first trace. The average value of the space between traces is used as the threshold to decide the space between words. Thus the related traces that belong to a word are placed in a <traceGroup>. The traces captured in drawing mode are simply grouped in to a single <traceGroup> without any layout analysis.

The structure of an example InkML ink message is shown in Figure 3.

```
<inkMLMessage>
    <context contextRef="ctx-A"/>
    <traceGroup id="group-1">
        <trace>12 23, '2 '2, 3 5, 6 7, 8 4 ... </trace>
        <trace>..... </trace>
        <trace>..... </trace>
        <annotationXML  type="diagram">
                <height>40</height>
                <width>50</width>
        </annotationXML>
    </ traceGroup>
    …
    <context>
        <brush id="red10Pen">
                <color> #FF0000 </color>
                <width> 10 </width>
        </brush>
    </context>
    <traceGroup id="group-n1">
        <trace>..... </trace>
                .....
        <annotationXML type="word">
                <height>40</height>
                <width>50</width>
        </annotationXML>
    </traceGroup>
    <traceGroup id="group-n2">
```

```
.....
</traceGroup>
<annotationXML>
    <messageId>Message05042007112244</messageId>
</annotationXML>
</inkMLMessage>
```

Figure 3: Example of an Ink message

At the receiving end, the InkML payload is parsed and rendered. The context (including the traceFormat) of the ink is first constructed with reference to the definitions set up during the Initialization Phase. Trace data is interpreted with reference to the implicit "current context", which is updated by brush events and other context changes encountered during parsing. The coordinates of the various ink channels are computed by taking into account the range and resolution of the source ink and the target display.

<traceGroup>s tagged as words are rendered by wrapping around the available column width of the application interface (we prefer a scrollbar only in the vertical direction). <traceGroup>s tagged as drawings are rescaled to fit within the column width while preserving their aspect ratio. Thus the third issue mentioned in the introduction is addressed in this phase of the protocol.

5. SOLUTION ARCHITECTURE

The implementation of the protocol involves the development of the Ink Processor, the logical component in each messaging client that implements the protocol described above, and an XEP to extend XMPP to support InkML messages as payload. The basic messaging and presence services of XMPP are utilized.

5.1 Ink Processor

From a conceptual standpoint, the Ink Processor needs to deal with ink messages captured locally, as well as received from its peer. The architecture of this component is shown in Figure 4, and described further below.

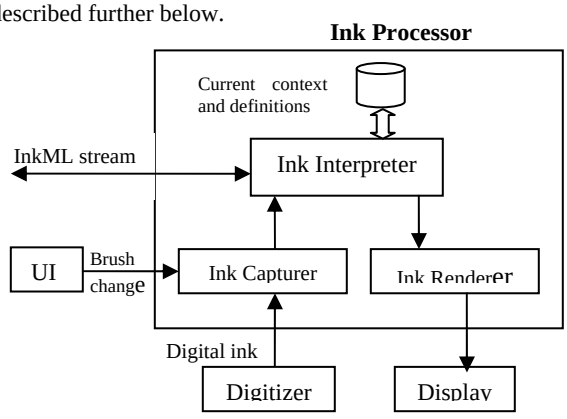

Figure 4: Ink Processor Architecture

5.1.1 Ink Capturer

The Ink Capturer component captures the ink strokes from the digitizer of the device and represents them using <trace> entities. It receives the brush change events from the Application User Interface (UI) and represents those using InkML <brush> entities. It also groups ink traces into logical units such as Words/Diagram based on explicit or implicit cues, and creates <traceGroup> entities to represent these logical units. It generates <ink> messages as shown in Figure 3 and sends them to the Ink Interpreter component.

5.1.2 Ink Interpreter

This component, as the name suggests, interprets InkML messages received from both the Ink Capturer (local ink) as well as from the peered messaging client. As it interprets the InkML messages, it constantly updates the Current Context and maintains Definitions. It also applies transformations to the digital ink based on the current context and sends ink in a renderable form to the Ink Renderer component.

5.1.3 Ink Renderer

The Ink Renderer component renders the digital ink data from the interpreter onto the display area of the device. In cases where the digital ink data does not fit the display area, the Ink Renderer makes scaling (drawings) and reflow (words) decisions based on the type of ink data received.

5.2 InkML Message XEP

This component is responsible for adding InkML messages to the XMPP message payload. It intercepts all the XMPP packets sent from the XMPP server to the client, extracts the InkML fragment data of the ink message and sends it to Ink Interpreter component. The client users use the unique Jabber Id (JID) to log into the XMPP server. This component also handles client login and messaging session management.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a solution to the problem of peer-to-peer ink messaging across heterogeneous devices and platforms. In particular, we have proposed the use of InkML as an interoperable digital ink format, and proposed a protocol for the use of InkML messages to support peer-to-peer ink-based messaging, and architecture for the Ink Processor component that implements these protocol operations over XMPP. We have implemented and tested peer to peer ink messaging solution across different ink platforms such as between a Linux desktop with an external graphics tablet, and a Windows PDA. Future work will focus on extending this protocol to support multiple clients, wherein a message from one client is broadcast to all others, and the participating devices could be either homogenous or greatly different, and studying performance in WAN settings. We also plan to explore the transmission of digital ink annotations of images and text documents along with the underlying content, and voice as an additional modality apart from ink.

7. REFERENCES

[1] Ink Markup Language(InkML), http://www.w3.org/TR/InkML/

[2] EXtensible Messaging and Presence Protocol (XMPP), http://www.xmpp.org/rfcs/.

[3] XMPP Extension Protocols (XEP), http://www.xmpp.org/extensions/

[4] XEP-0113: Simple Whiteboarding, http://www.xmpp.org/extensions/xep-0113.html

[5] Jonathan Neddenriep, William G. Griswold, "RiverInk--An Extensible Framework for Multimodal Interoperable Ink," hicss, p. 258b, 40th Annual Hawaii International Conference on System Sciences (HICSS'07), 2007.