RiverInk – A Framework for Multimodal Interoperable Ink

Jonathan Neddenriep   William G. Griswold
Computer Science & Engineering
University of California, San Diego
{jneddenr,wgg}@cs.ucsd.edu

Abstract

Pen-based interfaces offer exciting opportunities in ubiquitous computing through the enabling of new hardware form factors and socially acceptable computing tasks. However, prevailing ink representations are not compatible across devices or even within vendors, compromising interoperability and hence true ubiquity. We propose a lossless multimodal multi-format XML ink framework called *RiverInk* that overcomes many of the interoperability issues faced using platform-specific standards. This paper motivates the interoperability problems created by ubiquity, and then describes the design of RiverInk's format, APIs, and ink controls. RiverInk's framework encompasses interoperable support for both archival and streaming ink-based applications, and a range of alternatives for platforms that do not currently support ink. Three RiverInk applications were developed within the ActiveCampus ubiquitous computing environment to demonstrate both the ease of adding interoperability to an ink-based application and the degree of interoperability provided.

1. Introduction

Pen computing has a rich history of innovative hardware and software solutions [1] that leverage pen-oriented modalities to provide new methods for both text and graphical input as well as user interaction. Recent advances in processor speed and hardware miniaturization, coupled with maturing software platforms, has brought usable tablets and handhelds to market at reasonable price points. Mass production of appropriate hardware components has enabled economies of scale in the handheld marketplace. Pervasive mobile wireless networks in conjunction with pen-based form factors are creating numerous opportunities for collaborative applications. The simplest of these is interactive ink messaging, or whiteboarding. A requirement of these applications is the ability to share ink data across all platforms and users.

Unfortunately there is no standard ink representation across platforms or even within a single vendor. For example, TabletPC ink is not only hardware and operating system specific, it is proprietary - making it impossible to work with on other platforms or form factors. PocketPC and Linux do not use standard formats for persistent ink storage. There are three possible solutions that come to mind.

This work supported by gifts from Microsoft Research and Hewlett Packard. A version of this paper submitted to UIST'05.

The first is an implementation of device-to-device native ink conversion functions [Figure 1]. From a storage perspective this is a very simple solution because a single native ink object is all that need be stored. Ink could be stored in any native format and could be turned into any other native format. This would require every platform to have knowledge of every other platform's ink. Although viable on a small scale, the growth potential of ink-enabled device platforms makes this an impractical solution with an *N^2* growth rate of conversion functions. Further, a new ink format would require all existing platforms to be updated with a new conversion function. All of this presupposes access to the ink specifications both from a technical and legal perspective. This is impossible in a proprietary world.

The second obvious solution is the implementation of a common intermediate format (CIF) [Figure 2]. This approach requires only *N* conversion functions (from each native format to the CIF and vice-versa). Additionally, each platform need only have knowledge about its own ink format and the CIF. Only one representation of the ink must be stored, making it a simple format for persistence. However, because of the wide variety of hardware device parameters and application-specific attributes, no conversion framework would produce a lossless copy of the original native format. In fact, on some platforms, programmatic stroke creation with complete point parameters is not a supported API call. For example, the early TabletPC SDK did not allow programmatically creating strokes with fully specified point-frequency sampled objects. Thus it was impossible to create lossless copies of TabletPC ink strokes from another platform (using a CIF or a native-to-native scheme).

Although conversion-only solutions fall short of the ideal, a third possible solution is a full, non-native, independent suite of ink controls implemented for all target platforms from scratch [2]. New controls for ink capture as well as text and gesture recognition would be required for all platforms. For example, ink controls for ink capture and persistent storage, using a common format, could be built for .NET, Java, GTK, Mac OS X, etc. Each platform would have a compatible control implemented for its native environment, but not using its native inking mechanisms. A suite of cross-platform controls like this would be compatible because they would use the same ink format. No conversions would be necessary, and all new platform support would only need to be implemented with an understanding of the common format. This suite could also be used to ink-enable platforms with no native ink support. However, such non-native controls would not take advantage of the inherent abilities and strengths of each platform's software or hardware. Further, the real-time performance of inking is a critical factor in user acceptance and usability. Non-native controls could not be optimized at the same level as native ones because inking performance often requires OS and hardware-specific optimization. Additionally, achieving a consistent look-and-feel and expected behavior for the user requires native controls. For example, copy-paste between applications of ink as a first-class data type requires native ink support. Most critically, a complete multi-platform implementation of new ink controls would require an enormous amount of duplicative effort, first in creating the suite from scratch, then porting the suite to every new platform, regardless of the ink support within the operating system.

In light of the limitations of each of these approaches we propose a hybrid approach incorporating elements of each, consisting of four main elements. To cope with incompatible standards, platform-to-platform ink conversions must be supported. Two, since many platforms do not have their own ink support, a platform-independent ink format and controls for creating and rendering ink are needed. Three, because some platforms cannot even create or render ink, at least with acceptable fidelity, alternative non-ink representations are required for them to display. Finally, because the design considerations for streaming ink applications differ so greatly from archiving ink applications, the framework requires special functionalities for each and the ability for an application designer to selectively opt out of parts of the framework that are ill-suited to their application.

In particular, the solution proposed is to use a standards-centric, multi-representational approach to enable multimodal, multi-platform applications that provide lossless ink persistence and communication. A client program creates and stores or transports up to four representations of the ink: its native representation, an intermediate format (InkML), an image of the ink, and a text representation of the ink from the native client's recognition engine (where available). If the resulting data is used on the same platform as the creator's, the native format can be used with no loss of fidelity. If a different platform is used, the intermediate format can be used in conjunction with an IL-to-native conversion function to provide native ink. On a non-inkable platform, such as the web, the image created by the client can be used. On devices limited to a text modality (such as a phone), the recognition results of the ink can be used. Speech-only devices can use this text as well. This multimodal scheme insures that a format appropriate to any device will be available, albeit with some loss of information if a non-native representation is used.

It should be noted that it is up to the application developer to determine the extent to which the framework is utilized for different modalities. For example, in the case of a whiteboard application, performance is the most important factor. In this case, only the CIF is streamed to other clients, in real-time. Although this results in loss of ink fidelity, it provides the performance necessary for the application by eliminating unimportant information and representations (in the whiteboard application outlined below, streaming is only done between ink-enabled devices, therefore ink images and text recognition results are not as important). Receiver-side framework libraries are able to handle any mixture of representations and use the most appropriate.

A scheme for representation and persistence of cross-platform ink data does not solve all interoperability problems, as some platforms do not even support ink *per se*. For this reason, the proposed framework includes a .NET control that uses our CIF as its native format and the multi-representational framework format directly as its persistence mechanism. Although not as full featured as native controls like TabletPC, it is easy to port and able to provide enough functionality for several useful applications. The proposed framework is thus a multilevel treatment of the interoperable ink problem, combining native ink persistence, seamless CIF conversion, and non-native universal controls with a compatible ink description format.

In the following, section 2 lays out the specific design and implementation details of the interoperable ink framework. Section 3 provides experimental results of developer usage of the framework in three ubiquitous multi-platform applications. Section 4 discusses the interoperability achieved, the extensibility and portability of the framework, and the framework performance.

2. Design and Implementation

RiverInk consists of three elements, a multi-format representation with corresponding APIs for conversion and archiving, a set of platform-independent controls for creating and rendering ink, and an API for streaming. We describe these elements bottom-up according to the architectural diagram shown in Figure 3. To date, three platform libraries have been developed, TabletPC, non-tablet Windows, and PocketPC. All three allow the developer to work in a managed .NET environment. The non-Tablet library consists of 1290 lines of code, while the TabletPC version contains 1660. This includes the representation classes, the ink control, and the streaming framework. .NET was chosen as the primary platform for the project because of existing target .NET applications (the ActiveCampus project) and because of the maturity of TabletPC controls for .NET.

2.1 Multi-format Representation

Our multi-format representation is the first element in addressing the four problems of interoperable ubiquitous ink: cross-platform ink incompatibility, lossiness, platforms that lack ink support, and modal incompatibilities. Three requirements for the framework include an intermediate format, an encapsulation mechanism for each ink representation, and seamless conversion management. An overview of the representational concerns of the API can be seen in Figure 4.

2.1.1 InkML as CIF

A non-proprietary, easily parseable representation for pen stroke data is a requirement for developing any interoperable mechanism for ink. InkML is a W3C standard for describing ink using XML [3]. It provides a means for describing twelve point-level properties and arbitrary stroke properties. This allows drawing and application specific attributes to be easily bound to the stroke data. Because of InkML's flexibility and its

XML basis, it was used as a primary building block of the interoperable ink representation. There is no standard, however, for specifying attributes that most applications have in common such as brush color, size, or transparency. This means that sharing data with outside applications is difficult. However, within the framework standard attributes were defined so that any platform using the framework could communicate this basic information. Currently, a lowest common denominator is used, with only x and y data being used in the CIF to simplify the conversion process and minimize unnecessary data (TabletPC is the only widespread platform that uses any other point-level properties). This would be a simple addition to the framework to add support for more properties.

2.1.2 XML to wrap representations

To provide the most open and parseable format possible, XML is used to wrap each representation within the interoperable ink object [Figure 5]. A representation can consist of any format (properly encoded to coexist within an XML document). In general it includes both ink data and image and text as alternative modalities. When an interoperable XML representation is read, any information understood by the platform specific library is kept while everything else is simply ignored. The output is contained in **`<InteropInk></InteropInk>`** tags. The InkML element is denoted with **`<InkML></InkML>`** and is not encoded within these tags. This allows a parser to directly read the InkML out of the RiverInk object. Image data is stored as a base64 encoded PNG denoted by **`<png></png>`** tags. MSInk is stored using a Microsoft binary encoding, ISF, between **`<MSInk></MSInk>`** tags. Finally, Unicode text output from the recognizer is recorded in **`<text></text>`** tags. Figure 6 shows a sample of the code needed to use the multiple representational storage approach of RiverInk instead of saving MSInk on its own. Only one additional line of code is needed to use RiverInk. Likewise, saving ink using RiverInk instead of the Microsoft Ink save functionality has the same small overhead cost.

2.1.3 Seamless conversion

To provide an infrastructure for ink conversion, an in-memory intermediate language (IL) representation was created for every interoperable ink object. This IL is an object based array of InkML elements. This consists of **`Traces`** (a single pen movement consisting of points, a stroke) with corresponding **`Brushes`** describing the attributes of the trace. **`Brushes`** are referenced by numeric id within the trace object. The most simple conversion function, and one built into every RiverInk library, is an IL-to-InkML function. This merely iterates through the trace and brush objects and concatenates them with XML tags. Each platform also implements its own IL-to-native conversion. TabletPC, for example, creates a **`Stroke`** from every IL trace and populates each **`Stroke`**'s **`DrawingAttributes`** with information from the IL's **`Brush`**. This conversion is done on the fly only when MSInk is requested by an application and there is no natively created MSInk object already in the interoperable ink object. This could occur, for example, if ink was created on a PocketPC (only InkML output) and was subsequently loaded on a TabletPC.

By abstracting the conversion information source through an IL, only *N* conversion routines must be written instead of *N^2*. In the example above, the InkML would be converted to the IL when loaded, and the MSInk creation function need only know how to translate from IL to Ink. This allows for the painless addition of additional supported data formats with only a constant increase in conversion functions. Further, these conversion functions are local to the new platform so other platform libraries do not need to be updated.

2.1.4 Multimodal representation

Many platforms used today are not inkable. Not only do they not support ink controls natively, they do not lend themselves well to handheld form factors coupled with natural ink input. Web applications, cell-phones, and speech-enabled platforms are just some examples of non-naturally-inkable platforms. Additionally, native clients can often render their own ink much better than another platform because they have full knowledge of the ink capture characteristics. Therefore, whenever possible, an image of the original ink rendering is included in the multi-representational format. This allows for display of the ink in a more accurate rendering when the ink is created on clients with advanced attributes. For example, pressure-sensitive ink strokes from a TabletPC retain their fading edges when shown on a PocketPC (without support for pressure sensitive ink rendering) by using a PNG of the TabletPC rendering. This must be done on the creator's platform for the same reason that the CIF must be generated on the creator – only the native platform knows how from both a legal and a technical perspective.

Many applications require a text-only modality, even in today's multimedia world. However, the majority of platforms do not contain ink-to-text conversion libraries. For this reason, the original client's text recognition results are stored as a representation in the multi-format scheme. This not only enables text only platforms, it provides for features such as searching and sorting of ink. Further, text-to-speech technology is a mature field so speech systems are also enabled by the textual representation.

Microsoft uses a similar multimodal full-fidelity approach to try and provide interoperability for ink data. MSInk allows a developer to save a GIF of ink that includes a binary encoding of the original ink data as metadata in the image. Although this allows some applications all the necessary support for multimodal ink support, it is lacking in several respects. First, the binary encoding of MSInk is still not readable on non-Windows platforms. Secondly, GIF is not as full featured as PNG and has been encumbered by patent issues. Finally, using GIF as the container for multiple representations drives the design of the format around a particular modality.

2.2 Controls and representation for non-ink platforms

Many platforms do not contain explicit support for ink. PocketPC does not use any mechanism for defining pen input as structured ink objects. In fact, there are no managed controls in the .NET Compact Framework for capturing, displaying, and saving pen input. Non-Tablet Windows does not ship with ink controls (for use with standalone input tablets, for example). Linux does not have ink capture controls or an accepted representation for tablets. Non-native controls with a corresponding ink format are needed.

Consequently, RiverInk includes a universal ink picture control for capture and storage of RiverInk. Its native ink format is InkML. The control is designed to run on the .NET Compact Framework, widening its accessibility to mobile devices. For any .NET device with basic Windows Forms support, ink capture, image backgrounds, streaming ink, and simple brush attributes are supported. To simplify porting and avoid performance problems on systems lacking ink-specific hardware, the rendering is simple, not supporting anti-aliasing, curve smoothing, pen pressure, or transparency. Since the control is not native, copy and paste of ink to other applications is not supported. Although Windows Forms support is required, the control abstracts the windowing toolkit to enable porting to other non-Windows .NET platforms. The control is just 326 lines of code.

2.3 Streaming

One of the identified requirements for ubiquitous ink-enabled applications is near real-time collaboration. To provide this functionality in a multi-platform environment, streaming ink was built into the interoperable ink libraries. Streaming performance was the most critical factor in the implementation. For this reason, only InkML was streamed. This pre-supposes that only ink-enabled devices take part in streaming sessions, as the image and text of the ink are not transmitted. The biggest challenge to address in the streaming implementation was a tight coupling between user interface elements and the streaming mechanism. To decouple the interoperable libraries from the platform specific user interface controls, a hybrid Mediator/Observer design pattern [4] is used [Figure 7]. A mediator is created that is attached to a control. In and out streaming objects are created with references to the mediator. The mediator is subclassed for each control from which streaming was supported. For example, for a TabletPC a **`TabletInkMediator`** was created. A Microsoft InkPicture could be passed into the mediator's constructor. The mediator registers for all stroke addition events, and sends the data from these (translated to InkML) through the attached out-stream. Likewise, the mediator receives events signifying incoming strokes and passes these to the native control in the native format after converting from InkML. Each stroke is sent with its corresponding InkML **`Brush`** element so that attributes are always attached to the correct incoming stroke. The InkML specification defines a streaming (versus archival) modality using XML element fragments to denote state changes (new strokes, brushes, etc). This specification is used in the streaming classes to provide the possibility to stream to third-party applications that also support the specification.

3. Experience

To assess the ease of developer usage and verify the claims presented above with regards to interoperability and multi-platform support, three ink-enabled projects were developed, all within ActiveCampus [5,6].

3.1 Digital Ink Graffiti

The first application was ink-based graffiti. Location-based digital graffiti is content that is fixed to a map location or an entity. Ink graffiti is digital graffiti created with a pen. Ink graffiti is more faithful to the graffiti metaphor because it lends itself well to pen-based form factors by enabling a user to tag while in a mobile configuration and because it better mimics spray-can graffiti. A user can use any color or image (either from the clipboard, file, or URL) as the background for a graffiti. Each user tag on a specific graffiti is layered on top of the previous. A slider allows a user to 'peel back' each layer to reveal the graffiti at a specific point in time [Figure 8].

The ability to tag from any pen-based platform and view on any other platform is a critical feature needed to aid adoption and provide true ubiquity in a heterogeneous university campus environment. The first author implemented this feature in ActiveCampus, so was completely familiar with the framework API. Integration with ActiveCampus was not difficult using an iterative development methodology. First, a TabletPC version was created, then support for the multi-representational data storage was included. Finally alternative platforms were ink-enabled by incorporating the RiverInk controls as an alternative. To enable non-TabletPCs with ink graffiti, it was necessary to change 51 lines of code from the TabletPC version (mostly to implement image compositing). An additional thirteen lines were added to detect and instantiate the correct version depending on platform. Complete functionality and interoperability from a user perspective was provided. Tablets seamlessly used TabletPC controls, while other platforms used the RiverInk .NET ink controls to provide inking on non-pen based Windows platforms. Finally, a PocketPC client was implemented for ink graffiti that used the .NET Compact Framework to provide a graffiti control. A user could tag on any platform on top of any previous graffiti and view the complete graffiti (i.e., all layers aggregated on top of the original background). If loaded on a TabletPC, each layer is either native MSInk, or an InkML-to-MSInk conversion is done. To the developer, the only call necessary was to access the MSInk attribute. This allowed all layers to be seen as MSInk in the native control, taking advantage of the advanced rendering (transparency, anti-aliasing, smoothing) found on the platform. If the layer was created on a TabletPC, it would be rendered from lossless data, otherwise some data could be lost in the conversion.

Because the PocketPC has minimal support for graphics rendering, the PocketPC client first combined all layers from their image representation in the interoperable ink data structure, and then allowed inking on top of this composite image.

This feature implementation demonstrates that the lossless storage of the original ink format, coupled with both an intermediate format and other non-ink modalities (image), provided sufficient data availability for all desired platforms and made a ubiquitous, multimodal application possible.

3.2 Whiteboard

The second feature, developed by another programmer, was a simple whiteboard application that mirrored the location-based chat functionality in ActiveCampus. A user is able to share a whiteboard with another user instead of a simple type/response conversation window [Figure 9]. The whiteboard is able to support multiple platforms, and render real-time streamed stroke data within a native control (if available). To do this, the streaming ink functionality of the interoperable ink framework was used, coupled with Jabber [7], the transport mechanism used in ActiveCampus. As outlined in Section 2.3, the streaming ink control uses the .NET Stream abstraction as its transport. This allows simple streaming over TCP/IP. However, Jabber uses an event model for its XML-based message passing. A transport layer was written (61 lines of code) to adapt for this architectural mismatch. This, coupled with a user interface for color, thickness, and chat management, was all that was needed. No further streaming infrastructure or control/streaming interfacing was needed beyond that included with the framework. A total of 24 lines of code were needed to set up the streaming outside the framework code. This included an interface abstraction providing different behavior for TabletPC and non-tablet. Not including this abstraction code, all streaming specific setup code (thirteen lines) was able to be copied from the sample application included with the framework.

3.3 ActiveClass

The third application to be ink-enabled using RiverInk, by yet another programmer, was ActiveClass, a tool for in-class collaboration [8,9]. It lets students ask questions in class and give feedback to professors and TAs. The ActiveClass project was initially web only, but a .NET-based client had recently been developed to better support real-time interaction. This provided an opportunity to integrate ink-based features. It allowed asking questions using ink directly on slides. A requirement of the ink features was to keep the web modality for viewing. Both the image and text representations of the interoperable ink object are extracted using server-side XML parsing in PHP. These two representations enable web-only clients to see questions on slides both as text (if coming from a TabletPC using recognition) [Figure 10] and as images overlaid on the slide [Figure 11]. This demonstrates the full multimodal support of RiverInk – by including both text and images, non-ink enabled users are still able to take advantage of the system and interact with those on ink-based platforms. To adapt the ActiveClass client from TabletPC-only to using the RiverInk framework for non-tablets took less than two hours for the developer, who was unfamiliar with the RiverInk API. The developer stated that he appreciated the framework because “I didn’t really have to write any code myself to get this functionality. I was able to just copy over the code…” After a refactoring that provided a pluggable user interface abstraction (around 230 lines were rewritten), the developer was able to add support for the RiverInk controls in an additional 96 lines of code (versus TabletPC support, which took 106 lines). The RiverInk control adapter class only differed on sixteen lines of code from the TabletPC adapter, allowing for easy incremental development.

4. Discussion

4.1 Interoperability

All three applications proved to be compelling usages of ink, according to users. The framework helped the developers achieve this by lowering the entry costs of adding interoperable support (both on a format and control level), as well as easily enabling alternative modalities. All three applications extended the reach of ink-based features by including, at a minimum, non-TabletPCs. Where possible within the ActiveCampus system, the PocketPC was also targeted for development as a natural platform for pen computing. This boosted the possible users from a small number of TabletPC owners on campus to anyone with a Windows device. Although it is possible to install the TabletPC runtime (using the TabletPC merge modules) on some non-tablet Windows machines, this does not always provide the level of ink capture desired and requires additional installation. Further, this is not possible on PocketPC or portable to Linux.

In the case of the whiteboard, a complete representation of ink was not used for performance reasons (see Section 4.3). Because only the CIF was used, ink was not transmitted with full fidelity. For example, pressure data is not currently included in the InkML representation. Therefore, even when transmitting between two TabletPCs there is a loss of data. This could have been avoided by using the full representation or by expanding the conversion functionality to include all twelve attributes of InkML. Both these would result in an increase in the size of InkML stored.

4.2 Portability

To be truly interoperable, the RiverInk framework must be portable to new platforms. RiverInk’s primary portability dependency is that the ink controls use Windows Forms. However, Windows-specific forms support is used in only eleven lines of code of the control. Porting RiverInk to a non-Microsoft .NET platform would require modifying these eleven lines to introduce an alternative windowing toolkit. A GTK [10] version of the control is possible and would be a simple task given the UI decoupling in the RiverInk controls. A Factory design pattern [4] at the application level could manage the selection of the appropriate control implementation. This, coupled with the portable Mono runtime for .NET, would significantly widen the number of available platforms for RiverInk, including Linux and Mac OS X.

4.3 Extensibility

To be truly interoperable, the RiverInk framework must also be extensible to new ink standards. For example, suppose support for Apple's InkWell [11] technology was desired. Assuming portability issues were already resolved, to be able to read and write the interoperable ink format, a new class, **`InteropInkMacOS`**, targeted at Mac OS X would need to be written. This would consist of changing approximately 130 lines of code using the **`InteropInkTablet`** class as a template. It would only need to be aware of the CIF (InkML) and the Apple ink objects. Persistent output would consist of the Apple format, InkML, PNG, and text from the InkWell recognizer. No other platform changes would need to be made. If this persistent XML file were to be loaded on a TabletPC, MSInk would be created from the InkML, ignoring the native Apple format as unrecognized.

To complete support in the framework for an additional platform, a streaming mediator must be written that takes events from a native ink control and streams them as InkML and vice versa. This would consist of approximately 120 lines of code. Once this is done, an additional platform can be used seamlessly in both static and streaming ink applications.

4.4 Performance

An inherent drawback of multi-representational storage is the overhead in disk space, memory, and bandwidth required. For example, graffiti users sometimes noticed a delay when reading heavily tagged graffiti, since each layer generally included all four formats. A RiverInk object generated on TabletPC can be nearly an order of magnitude larger than binary MSInk. For example, a normal graffiti layer as seen in Figure 8 consists of about 4k of MSInk data, 14k of InkML, 6.5k of PNG data, and 40 bytes of text. A possible feature to mitigate this is to add a storage layer that uses an open compression format before streaming or archiving. This would add compression complexity for any applications that use the output without the corresponding library. Currently, only a simple XML parser is needed to extract a particular representation whereas an XML parser and a compression library would be needed if a transport/storage layer were added. Yet, with the increase in cheap memory, prolific disk space, and high speed networks, storage space of this magnitude is not a big concern. As a point of contrast, in the ink graffiti application, the disk size of the graffiti backgrounds were often an order of magnitude larger than the ink layered on top.

Although not critical for user acceptance of graffiti, performance was a primary concern for whiteboarding and streaming in general. Considering the cost of not only generating but also sending each representation, the streaming mechanism uses only the CIF for transport. This decision was validated by the performance of the whiteboarding application. Initial users were impressed with the speed of the ink transport. What lag there is in the application (about a quarter second) is due to thread interactions between the UI and the Jabber transport layer. Also, pen strokes are not sent until the completion of the stroke, which more directly influences the user's perception of performance. Neither affects the usability of the system, however. Based on the user experience with graffiti, if each stroke had generated and transmitted a multi-representational format, real-time performance would degrade significantly. This ability to choose the level of usage of framework features is an important factor for developers in achieving performance goals while balancing interoperability concerns.

5. Conclusion

Pen-based computing is becoming commonplace. However, the lack of mature and complete solutions for interchange between platforms, as well as ink collection and display on non-ink enabled platforms has hampered the use of ink in ubiquitous software applications. RiverInk provides a hybrid solution consisting of a conversion framework, a multimodal, multi-format representation, and ink controls for use on platforms with no native ink controls. This approach provides a highly portable and truly interoperable framework for ink that takes advantage of native ink features and controls when possible. Lossless ink storage is achieved by never throwing away the original native format ink. Conversion to other native formats is provided by using a CIF (InkML). Multimodal support of ink is enabled by including an image of the ink and its textual representation. Because of differing requirements for archival versus streaming applications, a streaming framework that dealt with both UI and transport issues is included.

Three applications were developed with RiverInk within the ActiveCampus ubiquitous computing environment. The applications used the multi-format representation coupled with conversion functionality to demonstrate the abilities of RiverInk's data interchange. They also leveraged the multimodal features of the representation to provide ink data across differing user modalities (for example, web browsing). Platforms that are not natively ink-enabled are supported through the use of the RiverInk controls. Finally, real-time streaming of interoperable data is demonstrated. As a whole, the ink-enabled features of ActiveCampus make a compelling case for the relevance of ink within a ubiquitous computing environment, while the RiverInk framework provides the functionality necessary to deploy these features in a heterogeneous environment with minimal developer effort.

Acknowledgments

This project is supported in part by Microsoft, with additional assistance from Hewlett Packard. The authors would like to thank Christopher Schulte, Tony Chen, Steven Brown, Charles Lucas, and Adriene Jenik.

References

[1] Rob Jarrett, Philip Su, “Tablet Computing Comes of Age”, *Building Tablet PC Applications*, Microsoft Press, 2003

[2] SATIN, A Toolkit for Informal Ink-based Applications, http://guir.berkeley.edu/projects/satin/

[3] Ink Markup Language, W3C Multimodal Interaction Activity, http://www.w3.org/TR/InkML

[4] E. Gamma, R. Helm, J. Vlissides, and R. E. Johnson. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995.

[5] W. G. Griswold, P. Shanahan, S. W. Brown, R. Boyer, M. Ratto, R. B. Shapiro, and T. M. Truong, ``ActiveCampus - Experiments in Community-Oriented Ubiquitous Computing'', *IEEE Computer*, To Appear.

[6] W. G. Griswold, R. Boyer, S. W. Brown, T. M. Truong, E. Bhasker, G. R. Jay, and R. B. Shapiro, ``ActiveCampus - Sustaining Educational Communities through Mobile Technology'', Technical Report CS2002-0714, Computer Science and Engineering, UC San Diego, July 2002.

[7] Jabber Software Foundation, http://www.jabber.org.

[8] M. Ratto, R. B. Shapiro, T. M. Truong, and W. G. Griswold, ``The ActiveClass Project: Experiments in Encouraging Classroom Participation'', *Computer Support for Collaborative Learning 2003*, Kluwer, pp. 477-486, June 2003.

[9] T. M. Truong, W. G. Griswold, M. Ratto, S. L. Star, ``The ActiveClass Project: Experiments in Encouraging Classroom Participation'', Technical Report CS2002-0715, Computer Science and Engineering, UC San Diego, July 2002.

[10] The GTK website, http://www.gtk.org.

[11] The Apple InkWell webpage, http://www.apple.com/macosx/features/inkwell/.

Figures

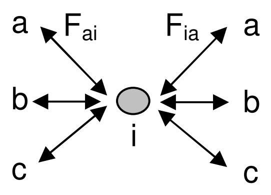


Figure 1. Common Intermediate Format Conversion: Every native format (a, b, c) requires an import from CIF and an export to CIF function.

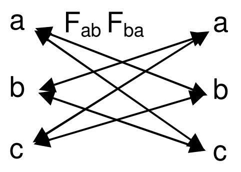


Figure 2. Device to Device Conversion: Every native format (a, b, c) requires an import and an export function to every other native format.

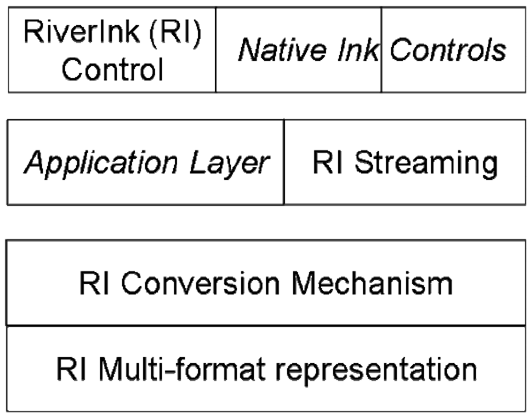


Figure 3. The RiverInk Architecture – The conversion mechanism is responsible for mediating between applications and the multi-format representation. The application layer is responsible for setting up appropriate ink controls and integrating with the interoperable ink objects. Streaming sits at the application level as it couples the UI and conversion functionality.

```
namespace InteropInk
{
  public class InteropInk
  {
    public ArrayList Traces;

    // multiple representations
    public virtual byte[] Image;
    public virtual string Text;
    public string InkML;

    public InteropInk();
    public InteropInk(string inInkML);

    public TraceBrush GetBrushByIndex(int index);
    public void Save(Stream inStream);
    public void Load(Stream inStream);

    // streaming support
    public virtual TracePacket AddTrace(ArrayList inPoints);
    public virtual TracePacket AddTrace(ArrayList inPoints, int inBrushIndex);
    public virtual void SetBrush(TraceBrush inBrush);
  }

  public class InteropInkTablet : InteropInk
  {
    // public attributes that can result in
    // a seamless conversion when accessed
    public Ink MSInk;
    public override byte[] Image;  // override for MSInk-based conversion
    public override string Text;   // override for MSInk-based conversion

    // constructors - if InkML is passed in, a conversion
    // will take place to get MSInk.
    public InteropInkTablet() : base(null);
    public InteropInkTablet(Ink inInk) : base(null);
    public InteropInkTablet(string inInkML) : base(inInkML);
  }
}
```

Figure 4. Representation and conversion API in the TabletPC library of RiverInk.

```
<?xml version="1.0" encoding="us-ascii"?>
<InteropInk>
 <InkML>
    <brush id="0">
    <height>1</height>
    <width>53</width>
    <transparency>0</transparency>
    <pentip>0</pentip>
    <color>-16777216</color>
    </brush>
    <trace brushRef="0">4445 2064 4445...</trace>
 </InkML>
 <png>iVBORw0KGgoAAAANSU...</png>
 <text>hello</text>
 <MSInk>base64:AD4cA...</MSInk>
</InteropInk>
```

Figure 5. Sample of multi-format XML representation used in RiverInk (content truncated).

```
// to load Microsoft Ink from a file
Ink msInk = new Ink();
byte [] inkData = loadDataFromFile("test.xml");
msInk.Load(inkData);

// to retrieve Microsoft Ink from the RiverInk representation
InteropInk riverInk = new InteropInk();
FileStream file = new FileStream("test.xml", System.IO.FileMode.Open);
riverInk.Load(file);
Ink msInk = riverInk.MSInk;
```

Figure 6. These code fragments demonstrate the difference between loading a Microsoft Ink object stored alone versus loading it from the RiverInk framework. Saving ink is also very similar using RiverInk instead of native TabletPC-only ink.

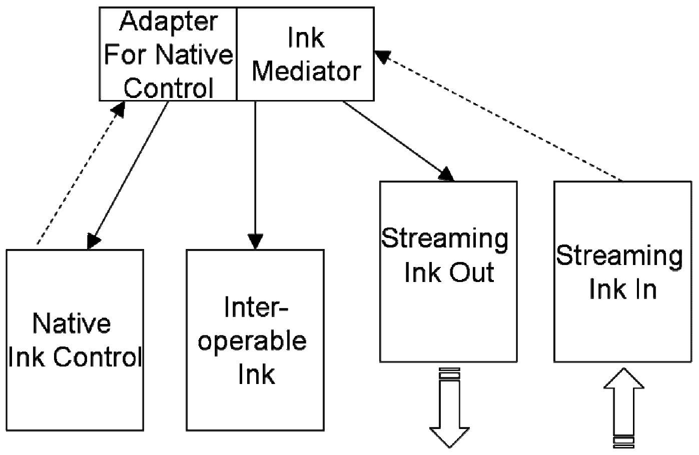


Figure 7 Streaming Architecture – Mediator decouples control UI from streaming transport components. An adapter is written for each supported ink control that subclasses the mediator creating a platform specific binding.

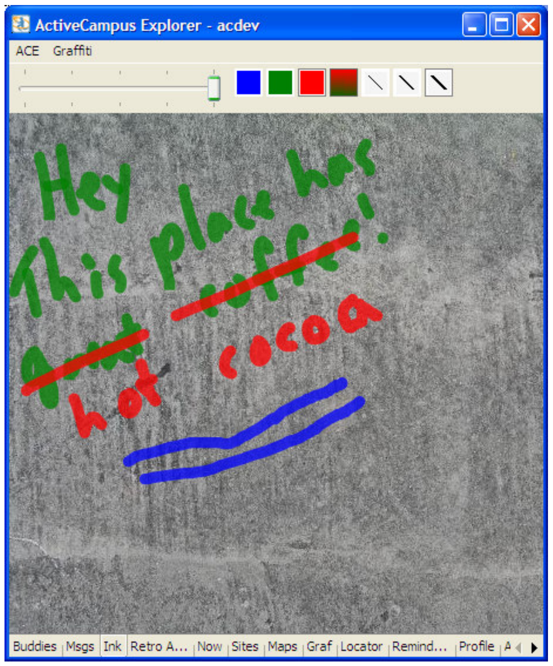
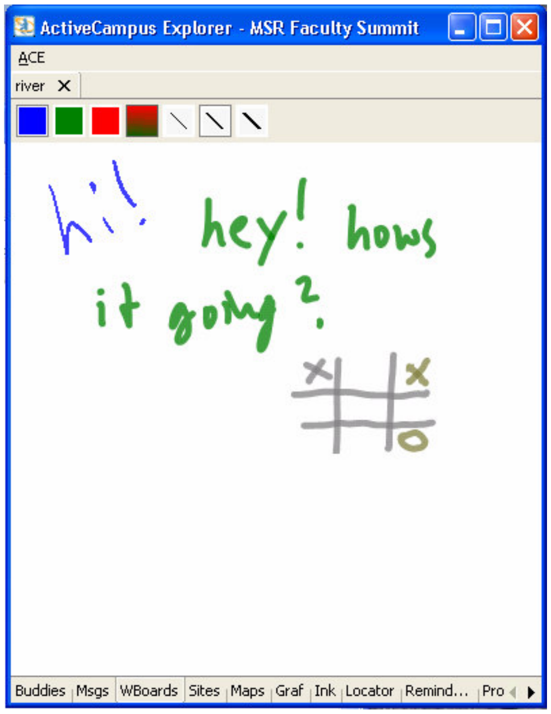


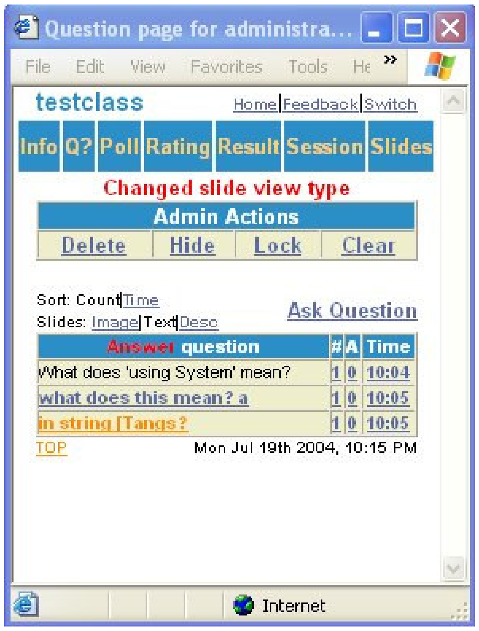
Figure 8. Ink graffiti within ActiveCampus. The slider at the top allows the user to go forward and backward in time to see the graffiti at different iterations.


ActiveCampus Explorer - MSR Faculty Summit
ACE
river
Buddies Msgs WBoards Sites Maps Graf Ink Locator Remind... Pro

hi!

hey! hows it going?

Figure 9. Whiteboard chat within ActiveCampus system.


Question page for administra...
File Edit View Favorites Tools He
Internet

testclass Home|Feedback|Switch

Info | Q? | Poll | Rating | Result | Session | Slides

Changed slide view type

Admin Actions			
Delete	Hide	Lock	Clear

Sort: Count|Time

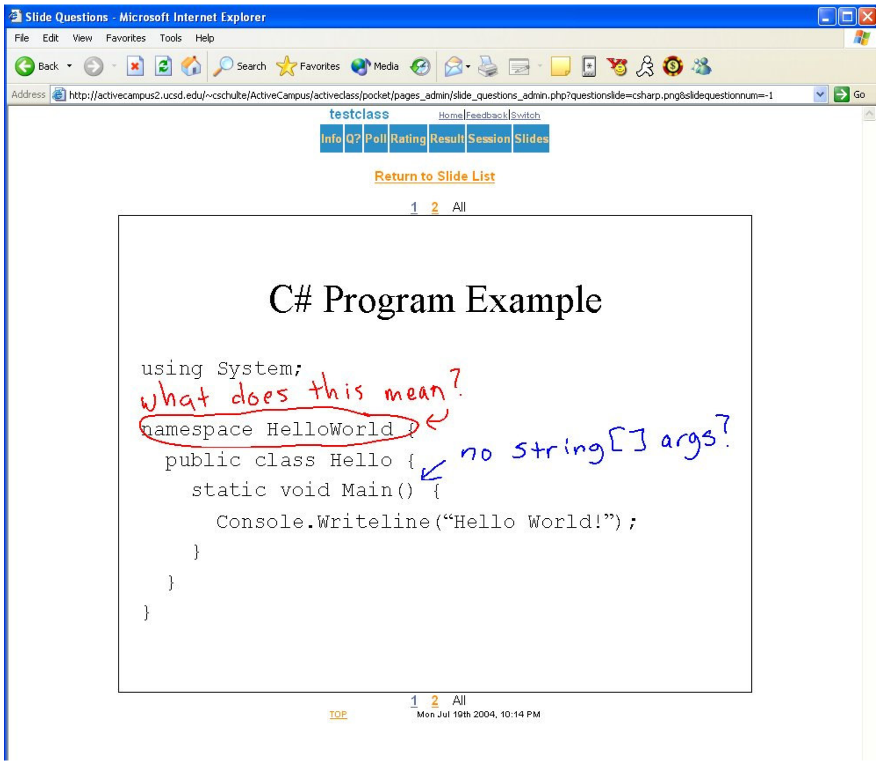
Slides: Image|Text|Desc

Ask Question

Answer question	#	A	Time
What does 'using System' mean?	1	0	10:04
what does this mean? a	1	0	10:05
in string [Tangs?	1	0	10:05

TOP Mon Jul 19th 2004, 10:15 PM

Figure 10. Textual list representation of the questions on the slide in ActiveClass. The recognition text is also able to be parsed from the multi-format XML RiverInk object server-side.


Slide Questions - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://activecampus2.ucsd.edu/~cschulte/ActiveCampus/activeclass/pocket/pages_admin/slide_questions_admin.php?questionslide=csharp.png&slidequestionnum=-1
Go
testclass
Home Feedback Switch
Info Q? Poll Rating Result Session Slides
Return to Slide List
1 2 All

C# Program Example

```
using System;
namespace HelloWorld {
  public class Hello {
    static void Main() {
      Console.Writeline("Hello World!");
    }
  }
}
```

what does this mean?

no string[] args?

1 2 All
TOP
Mon Jul 19th 2004, 10:14 PM

Figure 11. Ink annotations composited on a professor's slide in the ActiveClass system. The storage of a PNG of the ink within the interoperable ink format makes this web modality possible as the server only needs to be able to parse XML to get it.