From MIT SketchML to InkML or There and Back Again

Rui Hu
Computer Science Department
University of Western Ontario
London, Canada
rhu8@uwo.ca

Stephen M. Watt
Computer Science Department
University of Western Ontario
London, Canada
Stephen.Watt@uwo.ca

***Abstract*—The MIT Sketch Markup Language (SketchML) and the Ink Markup Language (InkML) are both used for sketch data representation. Techniques for exchanging sketch data between the two formats, however, are currently not readily available. In this article, we present a data binding solution for two-way conversion between SketchML and InkML. We show how to transform SketchML files to InkML archiving and streaming style. This allows sketch data to be used in collaborative environments where real-time sharing is desired. In the reverse conversion, we bind InkML elements to SketchML elements. This makes sketch data that is represented by InkML available to existing SketchML applications. We have tested these ideas in a shared whiteboard application and found them to perform well.**

***Keywords*-Pen computing, InkML, MIT SketchML**

I. Introduction

While once somewhat exotic and specialized, pen and touch enabled devices are now ubiquitous. It is therefore becoming increasingly important to be able to exchange drawing and writing input between platforms and applications. Various vendors record pen or touch input either in their own proprietary formats, or simply as images that lose important information. Various formats for digital ink have been proposed earlier, including Unipen [1] and Jot [2], but these have either been restricted to special uses or have not received wide acceptance. Presently two data formats stand out as sufficiently general for wide-spread use: InkML [3] and MIT SketchML [4], which we refer to as SketchML for simplicity. Both of these are XML-based data formats, similar in that they both record trace information, but otherwise quite different. This paper examines what is involved in converting between these formats.

InkML is an open and up-to-date standard proposed by the W3C to represent digital ink. It supports both archival (off-line) and streaming (on-line) ink data. Of interest to us here, it supports digital ink streaming between participants based on the concept of a context containing various information, including canvas, canvas transformation, trace format, ink source properties, brush properties and time stamps. *SketchML* is another open XML-based format providing a collection of elements to represent sketches as well as other meta information (such as the study and the domain for which the sketch was created). SketchML is useful in sketch data representation and annotation [5]. However, it lacks support for sketch sharing which is important in collaborative environments, such as distance education and work-group meetings. These environments involve multiple participants and are more complex than the single-user case.

By converting from SketchML to InkML, we make SketchML data acceptable by digital ink applications that support InkML. We also exploit the InkML streaming features so that sketch can be shared in real time between participants, who may be in the same room or across the planet. By converting from InkML to SketchML, we make sketch data that is represented by InkML compatible to existing SketchML applications.

II. SketchML to InkML

SketchML can be used by applications to store and manipulate sketch data. The `<sketch>` element is the root element of any SketchML file instances. It contains a set of metadata elements as well as sketch data elements. The metadata elements includes `<sketcher>`, `<mediaInfo>`, `<study>` and `<domain>`, which in turn specifies the person who created the sketch, the referenced external file (e.g. audio file), and the study and the domain that the sketch was created for. The sketch data elements, including `<point>` and `<shape>`, are used to represent a series of strokes. Each stroke consists of a sequence of contiguous ink points. To support stroke grouping and annotation, multiple strokes are allowed to exist within a named `<shape>` element. The `<shape>` element may also contain other metadata such as color, pen tip, raster and so on. The UUIDs that are assigned to each point in SkethcML are encoded as additional channels in the InkML trace format.

To convert from the SketchML format to the InkML format, the metadata elements can be easily represented by InkML `<annotationXML>` elements. An example of using InkML `<annotationXML>` element to represent SketchML metadata element is shown in Listing 1.

```
<annotationXML>
  <sketcher>
     <id>86bf5a7b-b71b-4912-a3aa-e686f5abdf1b</id>
     <dpi x="96" y="96" />
  </sketcher>
</annotationXML>
```

Listing 1. SketchML <sketcher> as InkML annotation

A. SketchML to Archival InkML

InkML archiving typically handles strokes that have been collected over some span of time and may re-organize them so that preferably the strokes be state-free. Therefore, the associated contextual information can be stored apart from the strokes. It is usually represented by the <context> elements. Each of them is assigned an identifier using the xml:id attribute. References to these contextual elements are made using the contextRef attributes of each stroke.

To accommodate the InkML archiving style, we represent each SketchML <shape> element by a pair of an InkML <traceGroup> element and a <context> element. The <traceGroup> contains the shape's coordinates. The <context> element contains contextual information. Reference to the <context> element is made by the contextRef attribute of the <traceGroup> element.

B. SketchML to Streaming InkML

InkML streaming delivers strokes in sequential time order. Initially, each ink collaboration participant sets up a default context and listens to context changes. This is similar to an event-driven model in which context changes are made when contextual elements are received. Whenever a new contextual element is received, it simply updates old values.

Converting from the SketchML to the InkML streaming style adopts the similar approach as to InkML archiving style. Each SketchML <shape> element is represented by an InkML <traceGroup> element along with a <context> element. However, two elements must be stored together and the <context> element must be sent prior to the <traceGroup> element. An example of conversion from SketchML to InkML streaming style is shown in Listing 2.

III. InkML to SketchML

Converting from InkML to SketchML is the reverse process. If the InkML was generated from SketchML, then all the meta data elements can be easily restored from the corresponding <annotationXML> elements. Strokes and their associated context will be together converted to SketchML <shape> elements.

IV. Implementation and Experiments

To test our ideas we have developed a software implementation, InkChat (http://www.orcca.on.ca/PenMath/downloads/InkChat/). InkChat is a cross-platform white-board application which allows conducting and archiving collaborative sessions that involve synchronized voice and sketch on a shared canvas. It accepts both the InkML and the SketchML formats and incorporates the technique presented for conversion. In particular, it allows any SketchML elements to be processed in the InkML streaming style so that it can be transmitted to other participants in real time.

```
<context traceFormatRef="TF0" xml:id="Ctx2">
   <brush xml:id="Brush2">
    <brushProperty name="transparency" value="255" />
        ...
   </brush>
</context>
<traceGroup contextRef="#Ctx2" xml:id="TG2">
  <annotationXML>
    <shape id="5a6b5e88-5274-404e-abb2-19c2e40351c9"
      type="SubStroke" width="150"
      author="86bf5a7b-b71b-4912-a3aa-e686f5abdf1b"
      height="1" name="stroke"/>
  </annotationXML>
  <trace xml:id="Trace2">
    3305.0 9966.0 1180113657038
      2758944186448628176 -6377473126013613401 23.0,
    3330.0 9952.0 1180113657046
      -7998112319767163665 -5372399576976323103 29.0,
    ...
  </trace>
</traceGroup>
```

Listing 2. Conversion from SketchML to streaming InkML

V. Conclusion

We have shown how to convert from SketchML to InkML archiving style and to InkML streaming style, each supports certain operations more directly. By converting to InkML archiving style, we can make SketchML file available to InkML applications. By converting to InkML streaming style, we can enable SketchML applications to be used in collaborative environments where sketch can be seen by other participants in real time. We have also shown how to conduct the reverse conversion, from InkML to SketchML. It makes sketch data that is represented by InkML compatible to existing SketchML applications. As a proof-of-concept, we have presented InkChat, a collaborative whiteboard application. It incorporates the conversion technique and shows good performance.

References

[1] I. Guyon, *Unipen 1.0 Format Definition*. http://www.unipen.org/dataformats.html, 1992.

[2] Slate-Corporation, *JOT - A Specification for an Ink Storage and Interchange Format*. http://unipen.nici.kun.nl/jot.html, 1996.

[3] Stephen M. Watt and Tom Underhill (Editors), *Ink Markup Language (InkML)*. http://www.w3.org/TR/InkML/, 2011.

[4] MIT-Design-Rationale-Group, *MIT SketchML Format*. http://rationale.csail.mit.edu/ETCHASketches/format/index.html.

[5] M. Oltmans, C. Alvarado and R. Davis, *ETCHA Sketches: Lessons Learned from Collecting Sketch Data*, 3rd ed. Proc. Making Pen-Based Interaction Intelligent and Natural, 2004.