Proceedings of the 6th AGILE
Dates – Lieu

A PORTABLE SVG-VIEWER ON MOBILE DEVICES FOR SUPPORTING GEOGRAPHIC APPLICATIONS

Thomas Brinkhoff

Institute of Applied Photogrammetry and Geoinformatics (IAPG),
Fachhochschule Oldenburg/Ostfriesland/Wilhelmshaven (University of Applied Sciences),
Center of Excellence for Geoinformatics (GiN)
Ofener Str. 16/19, D-26121 Oldenburg, Germany
e-mail: Thomas.Brinkhoff@fh-oldenburg.de
http://www.fh-oow.de/institute/iapg/personen/brinkhoff/

1. INTRODUCTION

An important development in the area of Geographic Information Science is the use of the *eXtensible Markup Language (XML)* [1] for representing, transmitting and visualizing spatial data. XML allows the definition of markup languages. A well-known example is the *Geography Markup Language (GML)* [2] as an XML encoding of geographic features. Current Geographic Information Systems (GIS) already support GML. Another – more general example – is *SVG*. *Scalable Vector Graphics* [3] is a standard of the W3C consortium for representing vector data.

Location-based services are another important development. The development of *mobile applications* is strongly influenced by the introduction of new mobile communication standards. Currently, the (intermediate) standard GPRS has been launched in Europe. By introducing the new mobile communication standard UMTS, this trend will be dramatically enforced. The appearance of mobile applications has also an impact on the devices used for presenting data: Instead of personal computers and workstations, *Personal Digital Assistants (PDAs)* or *mobile telephones* are used as Internet clients. However, the computing power of such devices is rather restricted compared to traditional computers. In addition, the speed and throughput of wireless networks are limited and are subject to large variations.

In this paper, the support of XML-represented simple features on mobile devices will be investigated. In order to be independent of the operating system of the mobile device, Java will be assumed as programming language. The paper starts with a presentation of existing XML encodings for the representation of geographic features (sect. 2). The variants and limitations of Java on mobile devices is topic of the next section (sect. 3). Then the processing of XML documents is discussed (sect. 4). The fifth section characterizes a corresponding object model. Observing this model and the limitations and differences of the Java versions, SVG parsers and SVG viewers are designed. The paper concludes with a summary (sect. 6).

2. XML FOR REPRESENTING GEOGRAPHIC FEATURES

This section describes the object models and specifications that are required for representing simple features using XML.

2.1 The Geographic Markup Language (GML)

For the XML-based representation of geographical features, the *Open GIS Consortium (OGC)* proposed the *Geography Markup Language (GML)* [2]. The object model of GML is based on the well-known *simple feature model* of the OGC [4]. In order to establish

Thomas Brinkhoff: „A Portable SVG Viewer on Mobile Devices for Supporting Geographic Applications“, Proceedings 6th AGILE Conference on Geographic Information Science, Lyon, France, 2003, Presses Polytechniques et Universitaires Romandes, pp. 87-96.

relationships between features and their geometry, a set of pre-defined properties are used by GML. The following example (fig. 1) shows an XML representation of a city. The city has the non-spatial properties `name`, `id`, and `population`. Furthermore, it has two spatial properties: `location` describes the point geometry of the city center and `extentOf` the polygon of the city boundary.

```
<?xml version="1.0"?>
<cities xmlns="http://www.geodbs.de/xml" xmlns:gml="http://www.opengis.net/gml">
    <city>
        <name>Oldenburg</name>
        <id>3403000</id>
        <population>153531</population>
        <gml:location>
            <gml:Point>
                <gml:coord>
                    <gml:X>8.2275</gml:X>
                    <gml:Y>53.1375</gml:Y>
                </gml:coord>
            </gml:Point>
        </gml:location>
        <gml:extentOf>
            <gml:Polygon>
                <gml:outerBoundaryIs>
                    <gml:LinearRing>
                        <gml:coordinates>
                            8.214289,53.087776, 8.277027,53.099326, 8.284276,53.121173,
                            8.305773,53.148083, 8.300405,53.162775, 8.299069,53.171417,
                            8.279517,53.191205, 8.226077,53.200909, 8.188254,53.19307,
                            8.173382,53.184449, 8.153453,53.165192, 8.164244,53.11119,
                            8.20185,53.115248, 8.204879,53.089616, 8.214289,53.087776
                        </gml:coordinates>
                    </gml:LinearRing>
                </gml:outerBoundaryIs>
            </gml:Polygon>
        </gml:extentOf>
    </city>
</cities>
```

Fig. 1 Example of GML.

For converting GML or other XML documents into another XML representation, the usage of *XSL transformations* is suitable. The transformation part of *XSL (Extensible Style Language)* [5] converts an XML document into another XML encoding by using XSL stylesheets.

2.2 Scalable Vector Graphics (SVG)

A large amount of spatial data is represented by vector data. However, there exists no general standard for representing vector data in the Internet. One typical solutions of this problem in the field of GIS is the usage of raster maps, which are dynamically generated on the server site. A restricted functionality of the client and a high data volume are the results. Both consequences concern especially mobile devices because they should work autonomously in order to avoid further data requests and because of the restricted throughput of wireless network connections. Another solution is the use of proprietary vector formats. The main disadvantage of such an approach is the missing standardization of those formats.

A data format for vector data is required that is flexible as well as standardized. Therefore, an obvious solution is using an XML-based data representation. A promising candidate is the graphical standard *SVG (Scalable Vector Graphics)*. The version 1.0 of SVG got the status as a W3C recommendation in September 2001 [3]. It has many features

that allow representing and visualizing cartographic information [7]. SVG supports not only zooming, panning, and the selection of objects, but also the manipulation of geometric objects. The following geometric shapes are supported by SVG:

- rectangles (`rect` element),
- circles and ellipses (`circle` and `ellipse` element),
- line segments and sequences of line segments (`line` and `polyline` element),
- polygons without holes (`polygon` element).

The `path` element allows a more general approach for defining shapes. A path consists of a sequence of connected or unconnected points. Connections between points may be line segments, quadratic and cubic Bezier curves, or elliptical arc curves. For examples, a path allows constructing multi-polygons with holes. For labels and annotations, the `text` element can be used. Fig. 2 shows a small SVG document. The visualizations of this and of another SVG document are depicted in fig. 3.

```
<?xml version="1.0"?>
<?xml-stylesheet href="svg.css" type="text/css"?>
<svg width="120" height="120">
    <g transform="scale(0.5)">
        <line x1="20" y1="20" x2="44" y2="54" style="stroke:red;"/>
        <rect x="60" y="10" width="24" height="34" style="stroke:rgb(0,0,0); fill:none;"/>
        <g id="contents" transform="translate(30,40)" style="stroke:red; fill:rgb(0%,75%,0%);">
            <rect x="30" y="12" width="24" height="34" visibility="normal"
                transform="translate(-30,-20)"/>
            <circle cx="50" cy="52" r="24" style="visibility:normal;"/>
        </g>
        <g id="group1">
            <rect x="70" y="52" width="24" height="24" rx="9" ry="9" style="fill:yellow;"/>
            <text x="0" y="70" style="font-size:7;">TEXT 1</text>
            <g>
                <line x1="30" y1="90" x2="100" y2="90" style="stroke:#0000ff;"/>
                <text x="30" y="90" style="alignment-baseline:auto; font-family:Arial;
                    font-size:16; font-style:italic; font-weight:bold; text-decoration:underline;
                    text-anchor:start;">
                        Hello
                </text>
            </g>
        </g>
        <ellipse cx="5" cy="55" rx="20" ry="40" class="green"/>
        <polyline points="50,50 75,50 90,75 120,80 90,60"/>
        <polygon points="15,5 30,1 35,15 15,10" transform="translate(30,0)"
            style="fill:aqua; stroke:maroon;"/>
        <path d="M 5 5 L 20 5 l 0 15 h -15 Z M 10 10 L 10 15 L 15 15 L 15 10 Z"
            style="fill:pink; stroke:teal;"/>
    </g>
</svg>
```

Fig. 2 Example of an SVG document.

For standard Internet browsers on PCs and workstations, it can be expected that they will soon support SVG without requiring additional plug-ins. However, for devices like PDAs or mobile telephones, additional software is required. Furthermore, the power of such “micro viewers” is too limited for supporting the complete definition of SVG. The SVG version 1.1 [8], which got the status as a W3C recommendation in January 2003, introduces so-called *profiles*. They allow defining restricted subsets of SVG. Currently, two profiles for mobile devices are proposed [9]: SVG Tiny for mobile phones and SVG Basic for PDAs. In this paper, we restrict SVG to those tags that are required for representing simple geographical features and that can be processed on current mobile devices.

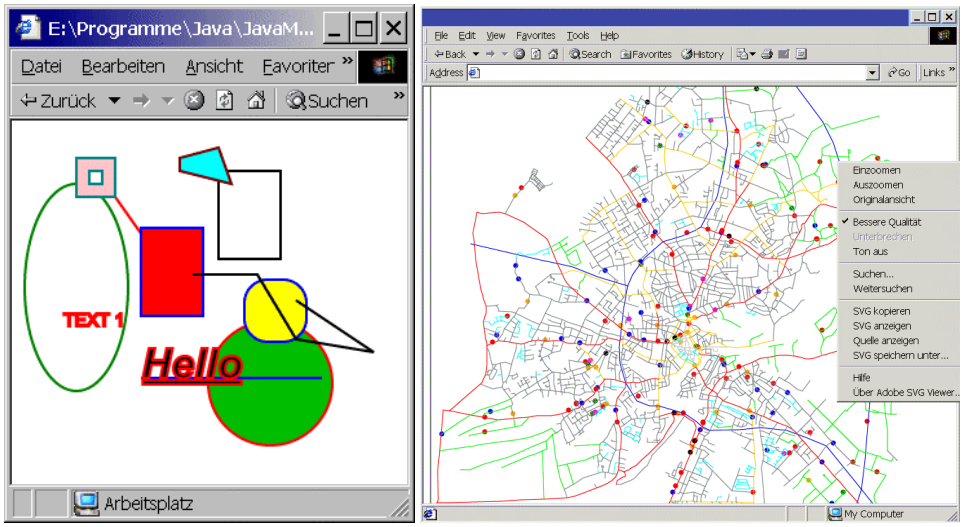

Fig. 3 Visualization of SVG documents by Adobe's SVG viewer [6].

3. JAVA ON MOBILE DEVICES

On PDAs we currently find three different operation systems: PocketPC (based on Windows CE), PalmOS and Epoc. On mobile phones, the situation is even more device-specific. Consequently, a programming language for such devices should be as much independent of the operating system as possible. The most common programming language fulfilling this requirement is Java. However, the following section will show that we have to use different versions of Java mainly depending on the characteristics of the mobile device.

3.1 PersonalJava

Especially for PDAs, Sun defined *PersonalJava*. PersonalJava uses Java 1.1.8 as its base with adding security features from Java 2 [10]. The class library is a subset of the standard library. The packages `java.io` for a file and stream handling and `java.net` for network support are included. The graphic library is of special interest for representing and visualizing geometries. It corresponds (with some exceptions) to the *AWT (Abstract Window Toolkit)* of Java 1.1.8. The class `java.awt.Graphics` allows drawing and filling rectangles, ovals, circular or elliptical arcs, and polygons (with holes) and drawing line segments, sequences of line segments, and texts. Bitmaps are supported. The coordinates are integer numbers and no affine transformations are provided. Classes representing geometric primitives as in Java 2 are missing.

3.2 The Mobile Information Device Profile for the Java 2 Micro Edition

For supporting devices like mobile telephones, two-way pagers, and wireless-enabled PDAs, Sun defined the *Mobile Information Device Profile (MIDP)* for the *Java 2 Micro Edition (J2ME)*. The version 1.0a [11] was published in December 2000. The new version 2.0 dates from November 2002 [12]. The MIDP is designed to operate on top of the *Connected Limited Device Configuration (CLDC)*. A *Personal Digital Assistant Profile (PDAP)* has not been specified, yet. Fig. 4 depicts the architecture of Java using CLDC and MIDP.

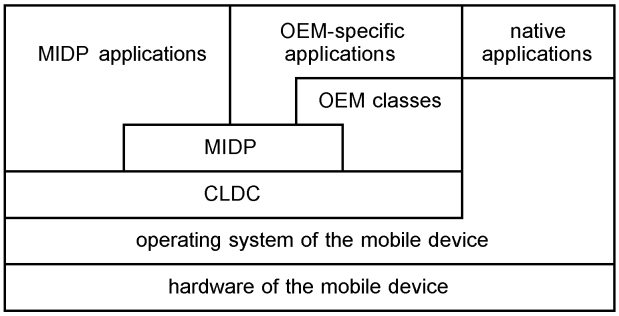

Fig. 4 Architecture of Java on mobile devices.

The minimum hardware requirements of the MIDP include a 96x54 display with an aspect ratio of approximately 1:1. “One-handed” keyboards like the keyboard of a phone, “two-handed” keyboards like a computer keyboard, and / or touch screens are expected. The minimum memory requirements are 128 KB (version 1.0) or 256 KB (version 2.0) of non-volatile memory for the MIDP components, 8 KB of non-volatile memory for application-created persistent data, and 32 KB (version 1.0) or 128 KB (version 2.0) of volatile memory for the Java runtime. A two-way, wireless networking with limited bandwidth is expected.

The class library is a very limited subset of the Java 2 class library. The package `java.net` is replaced by the package `javax.microedition.io` that supports network connections. The classes for the user interface are bundled in the package `javax.microedition.lcdui`. The included class `Graphics` allows drawing and filling rectangles, ovals, and circular or elliptical arcs, and drawing line segments, bitmaps, and texts – the filling of polygons is not supported. The MIDP version 2.0 added the support of filled triangles. The coordinates must be integer numbers – the floating-point types `float` and `double` are not supported by the MIDP! Thus, affine transformations are not provided. Again, classes representing geometric primitives as in Java 2 are missing.

4. THE PROCESSING OF XML ON MOBILE DEVICES

The processing of XML documents requires the parsing the documents. On a PC or a workstation, the parsing and further processing is relatively unproblematic. Necessary tools exist and most programming languages have libraries supporting XML. However, for location-based services, small mobile devices like PDAs and mobile phones are used. On such devices, the standard XML libraries of Java cannot be used. Therefore, the question should be investigated, how XML-represented geographic features can be processed on such devices.

4.1 Java API for XML Processing

Sun defined a lightweight API (application programming interface) for parsing XML documents. This *Java API for XML Processing (JAXP)* [13] allows the integration of concrete parser implementations. Two types of processing are supported: an event-driven approach, which is defined by the *Simple API for XML (SAX)*, and a tree-based approach that builds up the complete document in main memory. This second representation stores the *Document Object Model (DOM)*. An implementation of JAXP is, for example, the Xerces Java XML Parser provided by the open-source project Apache XML [14].

JAXP implementations have two severe disadvantages: First, they are based on the libraries of Java 2 and, second, their size must be measured in megabytes. Therefore, they cannot be used on mobile devices.

4.2 kXML

The open-source project kXML [15] [16] provides an XML parser suitable for all Java platforms including the Java 2 Micro Edition. Because of its small size (the compiled classes of version 1.21 need about 37kb), it is especially suited for applets or Java applications running on mobile devices like PDAs or MIDP enabled mobile phones. kXML was originally developed at the AI unit of the University of Dortmund. The kXML parser (version 1) supports two types of parsing: Using the request-oriented approach ("pull parsing"), a program must request the next tag or text of the document by itself – in contrast to SAX that calls user-defined methods in case of such an event. In addition, a DOM can be constructed.

5. OBJECT MODEL FOR XML-REPRESENTED SIMPLE FEATURES ON MOBILE DEVICES

For developing a Java object model for representing simple geographic features on mobile devices, we first have to consider which SVG elements are required for transforming GML into SVG (table 1).

GML geometry	SVG element
Box	`rect` (rectangle)
Point	There is no obvious counterpart for a point. Possible solutions are the use of rectangles (`rect` element), circles (`circle` element) or polygons (`polygon` element) for depicting symbols at the position of the point.
LineString	`line` (for line segments), `polyline` or `path` (for chains)
Polygon	`polygon` (for polygon without holes) or `path` (for polygons with holes)
MultiPoint, MultiCurve, MultiPolygon, MultiGeometry	The `g` element allows a definition of groups that can be considered as a layer or loosely coupled collection. A tighter coupling can be reached by one `path` element for representing a collection.

Table 1 Correspondence between GML and SVG elements.

Labels and annotations can be handled by the `text` element of SVG. Such an SVG document can be parsed using the kXML parser. We implemented an *SVG parser* that supports all SVG shapes (i.e. `rect`, `circle`, `ellipse`, `line`, `polyline`, and `polygon`), the `path` element with all straight connections and moves, the `text` element (without included style changes), and all transformations of the `transform` attribute. The SVG parser is based on the kXML parser (version 1.21) and works for PersonalJava as well as for the MIDP version.

Furthermore, a *spatial data structure* is necessary for organizing the resulting shapes according to their spatial properties. In our implementation, a partially ordered main-memory R-tree is used for this purpose [17]. The classes of this package are also independent of the Java version.

For representing the shapes, suitable Java classes are required. These classes should support at least the visualization of the shapes and their manipulation by affine transformations. In order to improve the comprehensibility, the structure and the methods of these classes are similar to the Java 2 geometry classes from the package `java.awt.geom`. In that package, the class `java.awt.Graphics2D` is responsible for displaying the geometries. In PersonalJava as well as in the MIDP, the `Graphics` class is predefined. Therefore, each shape class implements its visualization by using the restricted capabilities of the corresponding class `Graphics`. Another extension concerns the storage of the style properties (like color, visibility, or font name). In contrast to the Java 2 approach, each shape references an assigned style property object. The interfaces of these classes are identical in both Java versions. Fig. 5 depicts the corresponding package hierarchy.

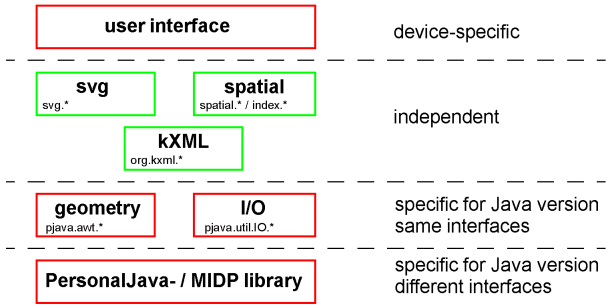

Fig. 5 Package hierarchy of the SVG viewer.

5.1 The PersonalJava Version

As mentioned before, PersonalJava uses the Java AWT. This restricts the visualization capabilities of an application considerably. For a shape, a color, a clipping rectangle and a simple paint mode can be set. For graphical texts, the font name, the font size and a simple font style can additionally be defined. By implementing some extra functionality, the attributes listed in table 2 are completely or partially supported. These attributes can be defined directly at a shape element using the `style` attribute and / or via CSS files referred by the element name and / or the `class` attribute. Undefined style attributes are inherited from the parent element.

Style attribute	Explanation	Support
`stroke`	Color of the border of a shape	All types of RGB definitions are supported. Many of the predefined color names are supported.
`fill`	The fill color.	see: `stroke`.
`visibility`	The visibility of the shape.	supported
`font-family`	The name of font.	supported – the existence of a font depends on the device.
`font-size`	The height of the font.	supported
`text-anchor`	The horizontal alignment of a text.	supported
`dominant-baseline` `alignment-baseline`	The vertical alignment of a text.	It is supported without distinguishing between `dominant-baseline` and `alignment-baseline`.
`font-style`	Specifies whether the text is to be rendered using normal or italic characters.	It is supported without distinguishing between `italic` and `oblique`.
`font-weight`	Specifies the boldness or lightness of the characters.	It is supported by using the font styles `PLAIN` and `BOLD`.
`text-decoration`	Describes decorations that are added to the text like underlining.	supported (but not the optional blinking)

Table 2 Supported SVG style attributes.

The SVG interpreter was integrated into a simple SVG viewer, which supports the map visualization, zooming, and panning. Fig. 6 shows PDAs executing this viewer.

Fig. 6 PDAs visualizing SVG documents.

5.2 The MIDP Version

The MIDP and its class `javax.microedition.lcdui.Graphics` are more restricted than PersonalJava and the corresponding class `java.awt.Graphics`. One major restriction concerns the non-existence of floating-point numbers. In the PersonalJava version, coordinates have the data type `double` like in the package `java.awt.geom`. For the MIDP, all coordinates are replaced by integer numbers. Therefore, the class `AffineTransform` has been completely re-implemented by interpreting the integers as rational numbers of limited precision.

An annoyance is the missing of a method for drawing filled polygons. At least, a method for filled triangles would be helpful – this method is first introduced by the MIDP version 2.0. Currently, this version has not been supported by mobile devices, yet. Assuming the existence of the method `fillTriangle`, triangulations of polygons have to be computed on the mobile device. A reduced performance for drawing filled areas would be the result. Another solution is the use of OEM-specific classes provided by the manufacturer of the device (fig. 4). However, the usage of such classes makes the application itself OEM-specific.

The implemented SVG parser supports the same attributes as the PersonalJava version. Some limitations are caused by MIDP restrictions like the support of only three fonts or of only three font sizes. The implemented SVG parser is used by a midlet. A *midlet* is a kind of applet in the context of the MIDP. This midlet realizes a simple SVG viewer that provides a visualization of the map. Zooming and panning are supported by processing key-pressed events. Fig. 7 shows the midlet running on a simulated PDA and mobile phone.

Fig. 7 Simulated PDA and mobile phone performing the midlet.

6. CONCLUSIONS

In this paper, the support of XML-represented simple features on mobile devices like PDAs and mobile phones was investigated. The approach assumes that a GML representation of the geographic features is transformed into the SVG format. For processing such SVG documents on a mobile device, a portable viewer is useful. In order to be independent of the operating system of the client, Java was selected as programming language. Two versions of Java are currently used on mobile devices: PersonalJava and the Mobile Information Device Profile (MIDP) for the Java 2 Micro Edition (J2ME). For the representation of the geometries, an object model was characterized. The implementation of this object model and of an SVG parser was directed by the limitations and differences of the two Java versions. On the top of this SVG parser, two SVG viewers were implemented.

7. REFERENCES

[1] World Wide Web Consortium, *EXtensible Markup Language, W3C Recommendation*, February 1998, http://www.w3.org/TR/REC-xml

[2] Open GIS Consortium, *Geography Markup Language (GML), Version 2.1.2*, *OpenGIS Implementation Specification*, September 2002, http://www.opengis.org/techno/implementation.htm

[3] World Wide Web Consortium, *Scalable Vector Graphics (SVG) 1.0 Specification, W3C Recommendation*, September 2001, http://www.w3.org/TR/SVG/

[4] Open GIS Consortium Inc., *OpenGIS Simple Feature Specification for SQL*, Revision 1.1, May 1999, http://www.opengis.org/techno/implementation.htm

[5] World Wide Web Consortium, *XSL Transformations (XSLT), Version 1.0, W3C Recommendation*, November 1999, http://www.w3.org/TR/xslt

[6] Abobe Systems Inc.: *SVG Zone*, http://www.adobe.com/svg/

[7] Neumann A., Winter A., „Vector-based Web Cartography: Enabler SVG“, *carto:net*, Version 2.0, October 2002, http://www.carto.net/papers/svg/index.html

[8] World Wide Web Consortium, *Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation*, January 2003, http://www.w3.org/TR/SVG11/

[9] World Wide Web Consortium, *Mobile SVG Profiles: SVG Tiny and SVG Basic, W3C Recommendation*, January 2003, http://www.w3.org/TR/SVGMobile/

[10] Sun Microsystems, Inc., *PersonalJava Application Environment Specification, Version 1.2a (Final)*, November 2000.

[11] Sun Microsystems, Inc., *Mobile Information Device Profile, JCP Specification Java 2 Platform, Micro Edition, Version 1.0a*, December 2000.

[12] Sun Microsystems, Inc., *Mobile Information Device Profile for Java 2 Micro Edition, Version 2.0*, November 2002.

[13] Sun Microsystems, Inc., *Java API for XML Processing, Version 1.1, Final Release*, February 2001, ftp://ftp.java.sun.com/pub/jaxp/89h324hruh/jaxp-1_1-spec.pdf

[14] Apache Software Foundation, *Xerxes Java Parser*, http://xml.apache.org/xerces-j/index.html

[15] Enhydra.org, *kXML, Version 1*, http://kxml.enhydra.org/index.html

[16] kObjects.org, *kXML, Version 2*, http://kxml.org/

[17] Brinkhoff T.: “Efficient Retrieval of Layered Data by Using R-trees”, submitted for publication, 2003.