Implementing Complex Polylines for use in GIS

Marko Kovačević[1], Aleksandar Milosavljević[2], Dejan Rančić[3]

Abstract – **This paper presents an approach for implementing visual portrayal of geographic features with polyline geometry (complex polylines) for appliance in GIS based applications. This approach relies on developed XML styling language for defining styling rules (decorations) of complex polylines. The structure of developed XML language was the foundation for designing the appropriate class library able to interpret XML definition of decorations and draw the desired complex polylines.**

Keywords – **Polylines, Style, GIS, XML, GDI+.**

I. Introduction

A geographic information system (GIS) is special type of computer-based information system tailored to store, process, and manipulate geospatial data [1]. The ability of GIS to handle and process both location and attribute data distinguishes GIS from other information systems. It also establishes GIS as a technology important for a wide variety of applications [2].

The fundamental information unit that GIS deals with is called a geographic feature. Geographic feature is an abstraction of a real world phenomenon associated with a location relative to the Earth [3]. Every feature may have a number of properties. One or more of the feature's properties may be geometric. Geometry provides the means for quantitative description of the spatial characteristics of features, including dimension, position, size, shape, and orientation. A geometric object is a combination of a coordinate geometry and a coordinate reference system. In general, a geometric object is a set of geometric points, represented by their coordinates. Basic geometric objects are points, polylines, and polygons [4].

The importance of the visual portrayal of geographic data in GIS cannot be overemphasized. The skill that goes into portraying data is what transforms raw information into an explanatory or decision-support tool. Fine-grained control of the graphical representation of geographic features is a fundamental requirement for any professional mapping community. Allowing user to define styling rules for visual portrayal of geographic features requires the existence of a styling language that the user and GIS application can both understand [5].

In this paper we present an approach for implementation of *visual portrayal of geographic features with polyline geometry* (further referred to as *complex polylines*). We developed XML styling language that enables simple and flexible way to define *styling rules* (further referred to as *decorations*) of complex polylines. The structure of developed XML language was the foundation for design and implementation of the appropriate class library aimed to interpret XML definition of decorations and draw the desired complex polylines.

Decorations include all graphic and text elements of a complex polyline. In this paper we focus on implementation of the following decorations:

- *Start cap* – graphic symbol at the start of the polyline.
- *End cap* – graphic symbol at the end of the polyline.
- *Pattern* – element set (includes graphic symbols, lines of different types (e.g. 2 pixels solid green line), empty spaces and text elements) that repeats itself along the polyline.
- *Label* – pattern like element set placed at defined positions on the polyline.

The paper is organized as follows: Section 2 presents XML language for defining decorations of complex polylines. Section 3 discusses the architecture and implementation issues of developed class library. Section 4 summarizes the achieved results.

II. XML Definition of Complex Polylines

XML language for defining decorations of complex polylines is specified using *XML Schema Definition Language*. A valid XML document contains a definition of decorations for a specific complex polyline, which can be interpreted using developed class library. Similar approach, based on defining styling language using XML Schema, is used by Open Geospatial Consortium in developing The Styled Layer Descriptor (SLD) Profile of the WMS [5].

Top level element that is used for specifying decorations is based on XML complex type named *ComplexLineType* (see Fig. 2). Bellow we briefly describe all containing elements of this XML type.

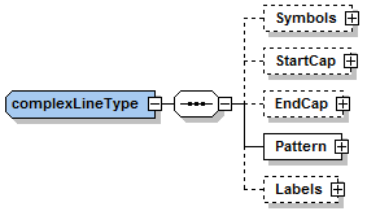

Fig. 2 - Structure of XML definition of complex polyline

Fig. 3 shows the structure of element *Symbols*. This element contains the description of every graphic symbol that is going to be used in decorations of the complex polyline. Contained element *SymbolLineWidth* is used to optionally setup the width

[1]Marko Kovačević is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: markko.marce@gmail.com

[2]Aleksandar Milosavljević is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: alexm@elfak.ni.ac.yu

[3]Dejan Rančić is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: ranca@elfak.ni.ac.yu

of the line used for drawing graphic symbols. There is also an array of elements *Symbol*. Element *Symbol* has two attributes: *id* and *src*. Attribute *id* is a unique identificator of the graphic symbol, used for referencing in other elements. Attribute *src* is the path to the SVG file that contains the shape description of the graphic symbol. Contained elements of the element *Symbol* are used to optionally setup the appearance of graphic symbol: its line and interior color (*LineColor* and *FillColor*, respectively), width of the symbol (*Width*) and displacement from the polyline segment (*Offset*).

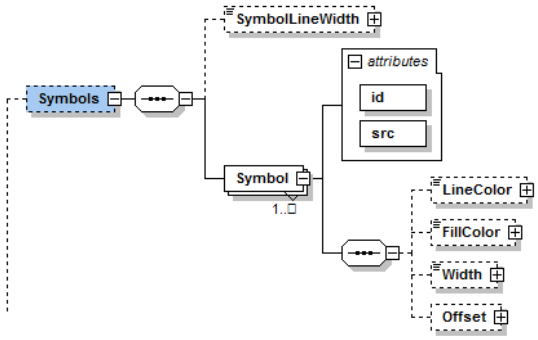

Fig. 3 - Structure of an XML definition of graphic symbols in complex polyline

Elements *StartCap* and *EndCap* are used to define graphic symbol at the start and at the end of the polyline, respectively. They only contain one attribute, *idRef*, a reference to the appropriate graphic symbol.

Fig. 4 shows the structure of an element *Pattern*, that is based on XML complex type *patternType*.

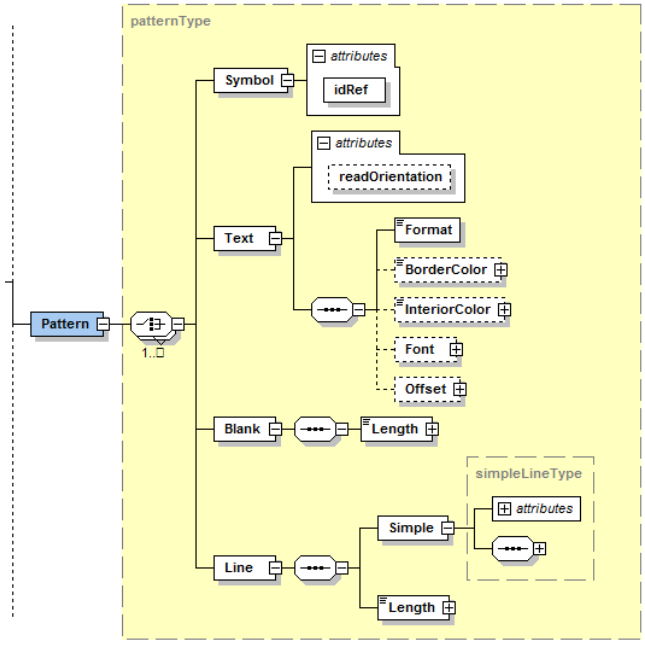

Fig. 4 – Structure of an XML definition of a pattern of complex polyline

Element *Pattern* is used to define a pattern of the complex polyline. Pattern is a set of graphic and text elements that repeats itself along the polyline. For creating the pattern, four contained elements can be used: *Symbol*, *Text*, *Blank* and *Line*. *Symbol* defines graphic symbol, referenced by the attribute *idRef*. *Text* defines text in pattern. Its attribute *readOrientation* is used to setup how the text will be oriented – in the direction of the polyline or in the way so it can be easily readable. First contained element *Format* holds the text. The rest of the contained elements of the element *Text* are used to optionally setup the appearance of text element: its line and shadow color (*InteriorColor* and *BorderColor*, respectively), font (*Font*) and displacement from the polyline segment (*Offset*). *Blank* defines the length (via element *Length*) of an empty spacing in pattern. *Line* defines length (via element *Length*) and properties (via element *Simple*) of the line in pattern.

Fig. 5 shows the structure of element *Labels*, which is used to define labels of the polyline. There are six types of labels, which correspond to the contained elements of element *Labels*. *PreStaringLabel* and *PostEndingLabel* define label before the start and label after the end of polyline, respectively. *StaringLabel* and *EndingLabel* define label at the start and label at the end of polyline, respectively. *EverySegmentLabel* defines label which will be in the middle of every segment of polyline. The array of elements *PositionLabels* defines labels at the specified positions on polyline.

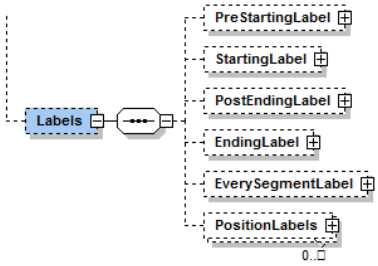

Fig. 5 – Structure of an XML definition of types of labels of complex polyline

All of these elements are based on XML complex type *labelType*, shown in fig. 6. Each label can consist of maximum 5 rows, which are defined by contained elements *RowAbove2*, *RowAbove1*, *Row0*, *RowBellow1* and *RowBellow2*. These elements are based on previously explained XML complex type *patternType*. The rest of the contained elements define the position of label (*Positions* and *Segment*), alignment of label (*Alignment*), orientation of label (*ReadOrientation*) and minimum distance between labels of the same type (*MinimumDistance*).

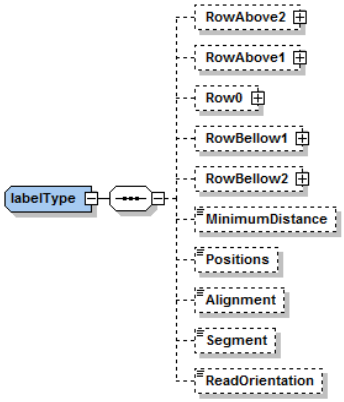

Fig. 6 – Structure of an XML definition of labels of complex polyline

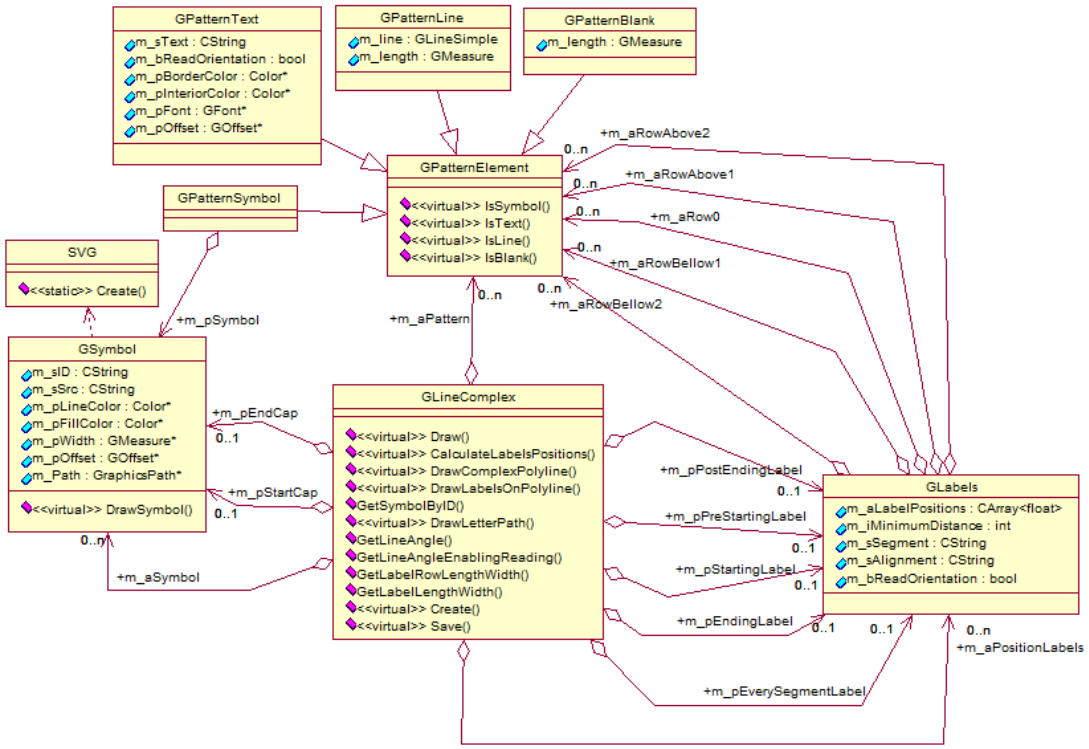

Fig. 7 – Logical model of class library for complex polylines

III. CLASS LIBRARY FOR COMPLEX POLYLINES

XML definition of decorations for complex polylines was the foundation for design and implementation of the class library aimed to interpret XML definition of decorations and draw the desired complex polylines. Similar approach, where XML definition and logical model of class library are closely related, is described in [6].

Fig. 7 shows UML logical model of the developed class library. The main tasks that this class library must perform are following:

- Reading data from a valid XML document and creating required objects.
- Calculating size, position and rotation of complex polyline decorations for submitted endpoints.
- Drawing complex polyline using GDI+ [7].

The class library for complex polylines is implemented in Visual C++. The main class is *GLineComplex*, which corresponds to XML complex type *ComplexLineType*. Attribute members of this class correspond to the contained elements of XML complex type *ComplexLineType* (Fig. 2). These members are set by calling the method *Create* that reads and interprets XML document with complex polyline definition. Method *Draw* does all necessary calculations and then draws complex polyline, which endpoints are passed to this method as one of the parameters.

Class *GSymbol* corresponds to the definition of XML element *Symbol* (Fig. 3). The graphic shape of each symbol is defined in appropriate SVG file [8]. Path to the SVG file is contained in the attribute member *m_sSrc*, which corresponds to XML attribute *src* of XML element *Symbol*. An attribute member *m_Path* contains GDI+ GraphicsPath object, obtained by calling the method *Create* of the class *SVG*. This method creates GDI+ GraphicsPath object from the submitted SVG file [7].

Abstract class *GPatternElement*, is a base class for four classes: *GPatternSymbol*, *GPatternText*, *GPatternBlank* and *GPatternLine*, which correspond to XML elements *Symbol*, *Text*, *Blank* and *Line*, respectively (Fig. 4). Class *GLabels* corresponds to XML complex type *labelType* (Fig. 5).

The main method of class *GLineComplex* is *Draw* method. Endpoints of polyline are passed to this method as one of its parameters. This method performs three essential tasks:

- Calculating *drawing parameters* (size, position and rotation) for defined decorations of complex polyline.
- Accepting decorations that satisfy certain conditions.
- Drawing complex polyline (with accepted decorations), using GDI+.

Fig. 8 shows pseudo code of *Draw* method. As it can be noticed, pattern of the polyline is drawn first. Drawing parameters for other decorations are calculated prior to drawing the pattern, but these decorations are drawn after drawing a pattern. In this way, pattern is drawn only where is visible (not where it is covered by other decorations), which in most cases gives better visual representation of polyline pattern.

Implemented approach also tackles the frequent problem of empty spaces in complex polylines. If the space left for drawing one of pattern's elements is slightly smaller than needed, it is better to draw this element than to leave this space empty. An implemented approach allows this element to be drawn without overlapping other, more important, decorations.

```
Set of decorations SI = {start cap, end cap, labels}.
Set of accepted decorations SA = {}.

foreach decoration D in SI
begin
   if (D exists)
   begin
      Calculate drawing parameters (params) of D.

      if ((Calculated polyline segment is big enough for D to be drawn on it) and
      (D can be drawn without overlapping with decorations from SA) and
      (if D is a label, minimum distance condition is satisfied))
      begin
         Put D in SA and memorize previously calculated drawing params of D.
      end if
   end if
end foreach

Calculate and draw pattern on parts of the complex polyline not reserved for
decorations from SA.
Draw decorations from SA, using previously calculated and memorized params.
```

Fig. 8 – Pseudo code for *Draw* method

We applied the class library presented in this paper to the GIS based application for graphical representation of military situation maps (see Fig. 9). All polylines (including polygons) shown in Fig. 10 are essentially complex polylines, drawn using the developed class library. Defining new types of polylines and modifying decorations of drawn polylines and polygons can easily be achieved due to the developed XML language. Part of the XML document that defines position and appearance of label "XX", of the polyline used to draw polygon in the top right corner of Fig. 9, is shown in Fig. 10.

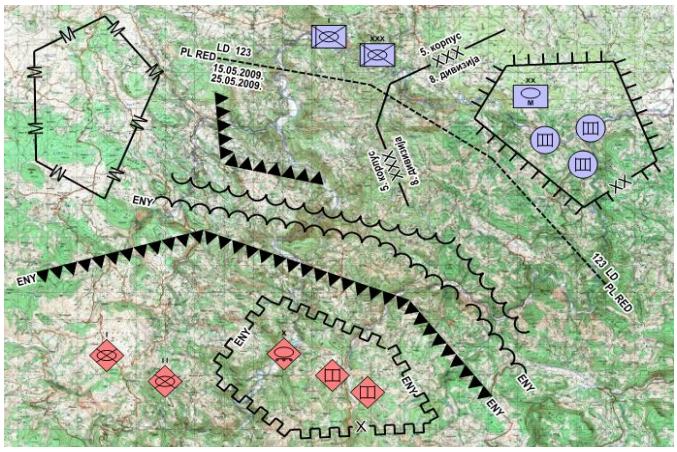

Fig. 9 – Using complex polylines for military situation maps

```
<PositionLabels>
   <Row0>
      <Text>
         <Format>XX</Format>
         <BorderColor>#FFFFFF</BorderColor>
         <InteriorColor>#000000</InteriorColor>
      </Text>
   </Row0>
   <Alignment>Center</Alignment>
   <Segment>South</Segment>
   <ReadOrientation>true</ReadOrientation>
</PositionLabels>
```

Fig. 10 –Defining label of complex polyline via XML

IV. Conclusion

Great importance of the visual portrayal of geographic features in GIS is undisputable. Implementing user-defined visual portrayal of geographic features requires the existence of a styling language that the user and GIS application can both understand.

In this paper we presented an approach for implementing visual portrayal of geographic features with polyline geometry (complex polylines). We developed XML styling language that enables simple and flexible way to define styling rules (decorations) of complex polylines. The structure of the this XML language was the foundation for designing the class library able to interpret XML definition of decorations and draw the desired complex polylines using GDI+.

In this paper we discussed implementation of the following decorations:

- *Start cap* – graphic symbol at the start of the polyline.
- *End cap* – graphic symbol at the end of the polyline.
- *Pattern* – element set repeated along the polyline.
- *Label* – element set located in defined locations on the polyline.

Element set specified for pattern and labels can contain one or more of the following elements:

- Graphic symbol (which shape is loaded from SVG file).
- Text with the specified properties.
- Line with the specified properties and length.
- Empty space with the specified length.

The core of the developed class library is an algorithm that calculates drawing parameters for specified decorations, eliminates some of the decorations in order to make the display distinctive and readable, and draws complex polylines.

References

[1] Worboys, M., and Duckham, M., *GIS: A Computing Perspective, Second Edition*, CRC Press, Boca Raton, FL, 2004.

[2] Chang, K., *Introduction to Geographic Information Systems, Third Edition*, McGraw-Hill, New York, NY, 2005.

[3] *The OpenGIS Abstract Specification, Topic 5: Features* (Version 5.0), document 08-126, Open Geospatial Consortium Inc., January 2009, http://www.opengeospatial.org/standards/as

[4] *OGC Reference Model* (Version 2.0), document 08-062r4, Open Geospatial Consortium Inc., November 2008, http://www.opengeospatial.org/standards/orm

[5] *Styled Layer Descriptor profile of the Web Map Service Implementation Specification* (Version 1.1.0), document 05-078r4, Open Geospatial Consortium Inc., June 2007, http://www.opengeospatial.org/standards/sld

[6] Milosavljević, A., Đorđević-Kajan, S., Stoimenov, L., *An Application Framework for Rapid Development of Web based GIS: GinisWeb*, Chapter 3 in Geospatial Services and Applications for the Internet (eds. J. T. Sample, K. Shaw, S. Tu, M. Abdelguerfi), Springer, 2008, pp. 49-72, ISBN: 978-0-387-74673-9.

[7] *Microsoft Windows GDI+*, Microsoft Corporation, http://msdn.microsoft.com/en-us/library/ms533798(VS.85).aspx

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