presented at: 7th International Symposium on Web and Wireless GIS (W2GIS 2007), Cardiff, UK, November 2007.
published in: Web and Wireless Geographical Information Systems, Lecture Notes in Computer Science, Vol. 4857, Springer, 2007, 76-93.

XFormsGI: Extending XForms for Geospatial and Sensor Data

Jürgen Weitkämper and Thomas Brinkhoff

Institute for Applied Photogrammetry and Geoinformatics (IAPG)
FH Oldenburg/Ostfriesland/Wilhelmshaven (University of Applied Sciences)
Ofener Str. 16/19, 26121 Oldenburg, Germany
{juergen.weitkaemper,thomas.brinkhoff}@fh-oow.de

Abstract. User interaction is mostly based on forms. However, forms are restricted to alphanumerical data – the editing of geospatial data is not supported. This statement does not only hold for established form standards like HTML forms, but also for the new W3C standard XForms. XForms defines a declarative framework for using forms and processing form data. It can be used with a variety of XML host languages. In this paper, we present an extension to XForms (called XFormsGI) that provides control elements for entering new geometries, for modifying the properties of existing objects as well as for selecting geometric entities. As XForms, XFormsGI is declarative, thus avoiding script extensions. XFormsGI supports coordinate systems. Additionally, it can be used for capturing and displaying sensor data like GPS or for the measurements of distances or temperatures. A prototypical implementation has been integrated into a mobile SVG client running on a PDA.

Keywords: XForms, SVG, Geographic Information Systems (GIS), Location-Based Services (LBS), User Interaction, Sensors.

1 Introduction

Nearly all software applications require human interaction for editing data or for displaying the results of a computation. The most important approach for such user interaction is based on *forms*. Forms are applied for depicting alphanumerical data as well as for requesting input from the user. They are not only used by desktop applications, but also by web and mobile applications. For web applications, *HTML forms* [10] are typically utilized.

As mentioned before, forms only offer display and input capabilities for alphanumerical data. In case of Geographic Information Systems (GIS) and other GI applications, however, it must be possible to display and capture geospatial data by graphical user interaction. Especially for web and mobile applications, it would be desirable to be able to describe the geospatial input facilities of the application declaratively in analogy to forms. Without such means, additional scripts and/or applets must be used for implementing the presentation and input of geospatial data.

The SPARK framework [3] is an example for such an approach that leads to a conceptional break between the handling of alphanumeric data and of geospatial data.

The new W3C standard *XForms* [14] allows describing the user manipulation of alphanumeric data in the context of XML documents. Like XHTML [16] should take the place of HTML, XForms is supposed to replace HTML forms. But in addition, it is able to be integrated into other XML standards. In case of graphical and geospatial XML documents, *Scalable Vector Graphics (SVG)* [12] and the *Geography Markup Language (GML)* [8] could be host languages for XForms. Like HTML forms, XForms does not support interaction with geospatial data. But XForms is suitable to be extended for geospatial data. In contrast to HTML forms, XForms separates the visual representation of the user interface from the underlying data model. A single device-independent XML form definition has the capability to work with a variety of standard or proprietary user interfaces.

In this paper, we propose an extension that is conform to XForms. It allows for editing geometric entities like georeferenced points, lines and areas. Our extension is called *XFormsGI*. Its main characteristics are:

- XFormsGI provides control elements for entering new geometries, for modifying the properties of existing objects as well as for selecting geometric entities.
- XFormsGI is declarative, avoiding complicated and confusing script extensions.
- XFormsGI takes spatial reference systems into account.

We will show in the following that the concept of XFormsGI is not only suitable for capturing geospatial data, but can also be used for *sensor data* like GPS positions or the measurements of distances or temperatures. This is especially useful for supporting mobile GIS applications and location-based services (LBS). In this case, metadata like quality, number of visible GPS satellites, the moving direction are processed in addition to the pure sensor data. For validating the concept of XFormsGI, a prototype has been implemented and integrated into a mobile SVG client running on a PocketPC PDA.

The rest of the paper is organized as follows: After presenting the essentials of XForms, the concept and control elements of XFormGI for geospatial data are introduced. The fourth section deals with supporting sensor data. Section 5 presents an application of our approach and Section 6 gives an overview on related work. The paper concludes with a summary and an outlook to future work.

2 The XForms Standard

In this section, we give a brief outline of the XML-based standard XForms.

XForms [14] has its origins in HTML forms and will replace the current forms in future versions of XHTML. The use of XForms is not restricted to (X)HTML but can be embedded in arbitrary XML applications. XForms overcomes several of the limitations of HTML forms. Notably, it separates clearly the purpose from the presentation of a form: "XForms are comprised of separate sections that describe what the form does, and how the form looks. (…) XForms is designed to gather instance data, serialize it into an external representation, and submit it with a protocol." [15].

The functionality of a form is described in one or more *model elements*, whereas its presentation is represented by a set of *control elements*.

2.1 The XForms Model

The *model element* describes several properties of the form:

- An XForms form collects its data – the so-called *instance data* – as child elements of its `instance` element. The instance data will be displayed by control elements.
- The structure of the instance data, i.e. the set of attributes, is defined by the `schema` element. The *schema* is also used to constrain the domain of data.
- The *data flow* to and from the server that is processing the form data is given by the `submission` element.
- An optional *model binding expression* (`bind` element) establishes the relation between the control elements and their instance data

Figure 1 gives an overview on the cooperation between the components of XForms.

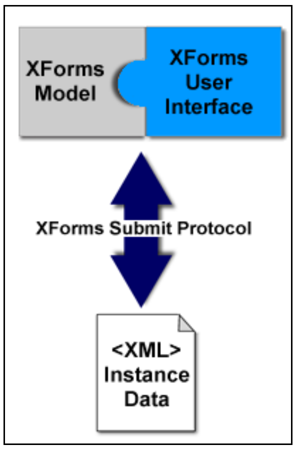


Fig. 1. Overall structure of XForms [15].

Figure 2 depicts the essential parts of an XForms model by a small example document.

```
<xforms:model id="form1" schema="xxx.xsd">
   <xforms:submission  method = "post"
                       id     = "send"
                       action = "http://..." />
   <xsd:schema>
      ...
      <xsd:simpleType name="status">
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="prof"/>
            <xsd:enumeration value="stud"/>
            <xsd:enumeration value="other"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:schema>
   <xforms:instance>
      <data>
         <name>Wacker</name>
         <firstname>Willi</firstname>
         <status>prof</status>
      </data>
   </xforms:instance>
</xforms:model>
```

Fig. 2. Example of the model element.

The URI `http://www.w3.org/2002/xforms` is the namespace of XForms. In our examples, we will use `xforms` as namespace prefix.

2.2 Controls

The form presented to a user is defined by a set of *control elements* defining the principal appearance and interaction of the form. They are embedded in a so-called *host language*. The host language is the XML application that contains the form. The most prominent example is XHTML. In case of graphical applications, the Scalable Vector Graphics (SVG) is a suitable host language.

In contrast to HTML forms, there exists no element which serves as container for all controls of one model. Instead, all controls referencing instance data of the same model element are considered a unit. They may be spread over the host document. There are several predefined controls like input fields, buttons, checkboxes, file upload elements, etc. Each control may have a *label element* as child which is placed automatically by the client.

Figure 3 depicts an example of an XForms `trigger` control with an embedded label usually displayed as a button.

```
<xforms:trigger>
       <xforms:label>Submit</xforms:label>
</xforms:trigger>
```

Fig. 3. Example of an XForms trigger.

Data Binding
The relation between a control element of the form and its instance data is established by the XForms binding mechanism. Each control refers to a model by a `model` attribute that may be omitted if the document only contains a single model. For specifying the element storing the data within the instance element, there exist mainly two alternatives:

- using the `ref` attribute to specify an XPath pointing to a child element of `instance` or
- using the `bind` attribute to reference a `bind` element in the model which in turn uses e.g. an XPath expression for defining the location of the data element.

In contrast to HTML, the XForms specification allows the definition of repeating structures. This permits the multiple insertion of input from a single set of controls as a sequence into the instance data.

Transmission of Data
An important functionality of forms is the transmission of data to a server that performs the processing of the input data or that provides data to be displayed. For this purpose, the `submit` element exists as a control element. When the *submit control* is activated by the user, the instance data is collected and transmitted. This is performed according to the protocol and address specified in the `submission` element of the corresponding model (see Figure 2).

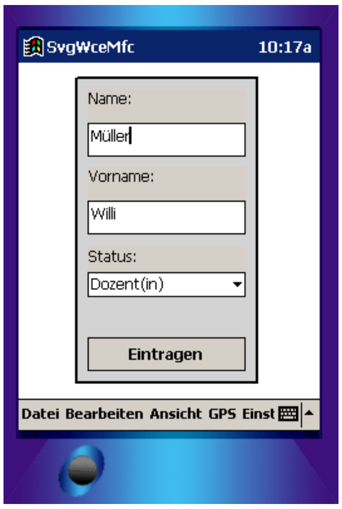
Embedding of XForms into SVG
As mentioned before, SVG is a suitable host language for graphical applications. The `foreignObject` element of SVG can be used for embedding XForms controls. Figure 4 depicts some XForms controls embedded into SVG.

```
<model id="form1" >
  ...
  <instance>
    <daten>
      <name>Müller</name>
      <firstname>Willi</firstname>
      <status>prof</status>
    </daten>
  </instance>
</model>

<foreignObject  x="0" y="110" width="100" height="20">
   <input model="form1" ref="/daten/name">
      <label>Name:</label>
   </input>
</foreignObject>
```

Fig. 4. Example of an XForms fragment that can be embedded in an SVG document.

Figure 5 shows the corresponding form displayed by the implemented prototype running on a PocketPC PDA.



Name:

Müller

Vorname:

Willi

Status:

Dozent(in)

Eintragen

Fig. 5. Display of an XForm by the PDA prototype.

2.3 Event Handling

XForms defines several *events* sent on state transitions of the document, controls or models. These events can either directly be routed to XForms actions like `reset` (resetting the form), `setfocus` (setting the focus to a specified control element), etc. or can be processed by *scripts*. XForms itself defines no method for script-based event handling. The definition of such facilities is a responsibility of the host language.

Figure 6 shows the usage of the action `reset` in order to model the behavior of the reset button known from HTML forms; such a control is not defined by XForms.

```
<xforms:trigger>
       <xforms:label>Reset</xforms:label>
       <xforms:reset ev:event="DOMActivate" model="themodel"/>
</xforms:trigger>
```

Fig. 6. Example of the usage of the action `reset`.

3 XFormsGI

In the following we describe an extension of XForms for capturing geospatial and sensor data. This extension will be called *XFormsGI*.

For XFormsGI we introduce the namespace URI `http://www.fh-oldenburg.de/xformsgi`. In the subsequent examples the namespace prefix `xfmgi` is used.

3.1 Controls

Several XFormsGI controls provide editing facilities for geospatial data. Geometric editing is usually performed by a pointing device like a mouse or track ball for desktop computers or a stylus for tablet PCs and mobile devices. Like in XForms, the details of interaction are not specified but left to the client implementation.

XFormsGI controls share most of the properties and behavior of XForm controls. One difference concerns the client visualization: To each XForms control, there exists a corresponding user interface element visualized by the client. For example, an XForms `input` element is usually represented as an edit box that also displays its instance data. In contrast, the main task of an XFormsGI control is to capture geometric data by pointing devices. To this end, it neither requires a direct visual representation nor the display of the corresponding data.

The controls introduced by XFormsGI are summarized in Table 1.

Table 1. XFormsGI controls.

Control	Description
`input`	Definition of a new base geometry.
`construct`	Like `input`. Additionally, from the base geometry a geometric entity of the host language is constructed based on a template.
`select`	Selection of elements for further processing.
`modify`	Modification of geometric properties of existing elements, mostly used in conjunction with a `select` control.
`sensor`	Definition of sensors like GPS (see Section 4).

State of XFormsGI Controls

Another difference between XForms and XFormsGI concerns the start and termination of an input. An XForms control like the text input element can be activated any time by the user to add, remove or modify one or more characters. If the user chooses to switch to another input field, the first one is not affected. In contrast, geometric interaction in general consists in a sequence of distinct steps that have to be performed in a certain order and that cannot be interrupted easily. Thus, an XFormsGI control needs to keep track of a *state* which consists of at least two possible values `stopped` or `running`. In order to start an input sequence, the state of the control has to be switched from stopped to running. This can be performed either by a trigger like a button or implicitly when the user picks an element. Selection and modification operations are typical examples where implicit activation is often used: Whenever an element is picked, the object is, e.g., added to the selection set or it is marked by a frame with "drag grapples" to allow stretching, rotation or displacement.

In order to represent the state, all XFormsGI controls have an attribute `state` which can have one of the following values:

- `stopped`, when the input is currently deactivated (default value of `state`),
- `running`, when input is currently active, and
- `start-implicit`, when the input sequence starts whenever the user picks certain elements.

This attribute can be modified by scripts in order to start an input sequence.

A further difficulty arises from the fact that some interaction sequences not only need a start trigger but also a stop trigger. Consider as example the construction of a line string with a previously unknown number of points. If a closed ring should be constructed, the user can pick the first point again to terminate the construction. If it needs not being closed, a separate trigger is required to terminate the input sequence.

In order to support different types of termination, two specific *XFormsGI events* are introduced in addition to the standard XForms events: `xformsgi-constructionstep` and `xformsgi-constructionterminated`. The first one is sent after every major construction step. In case of capturing a line string, it is sent any time a new point is defined. A further example is the input of a circle. After the construction of the center an `xformsgi-constructionstep` is raised to allow a special center mark to be set. During the processing of the `xformsgi-constructionstep` event, the geometric data already captured is available in the instance data.

The `xformsgi-constructionterminated` is raised after completion of the input sequence. Then, all data are stored in the corresponding instance data element.

Construction by the `input` Element

The `input` element allows capturing standard geometries. Its `type` attribute defines the geometry type. Possible values are `point`, `circle`, `line-segment`, `rect`, `polygon`, and `polyline`. For convenience, the `trigger` control activating the input can be placed as child to the `input` element.

Figure 7 illustrates an input control for capturing a line constructor.

```
<xfmgi:input
     type   = "line-segment"
     model  = "#geomodel"
     ref    = "#linedata "
     state  = "stopped">
   <xforms:trigger>
     <xforms:label>Start input</xforms:label >
   </xforms:trigger>
</xfmgi:input>
```

Fig. 7. Example of the usage of the `input` control.

In contrast to the start trigger, there is no direct support for a stop trigger. One possible solution would be to switch the trigger from “Start input” to “Terminate input” by a script. Alternatively a client implementation could add the “Terminate input” automatically to the menu or context menu when a geometric input is running.

Construction by the `construct` Element

The `construct` control is introduced as a convenience to facilitate the generation of geometrical entities of the host language by user input sequences. In contrast to the `input` element, which stores the input only in the instance data, the construction control generates a new entity after completion of the input sequence. The new entity is constructed according to a *template* specified by the `template` attribute and placed as new child to the element of the host language denoted by the `parent` attribute of `construct`.

Figure 8 illustrates this by using SVG as host language. The `template` attribute refers to the `image` element of SVG.

```
<defs>
   <image  id          = "img-template"
           width       = "60"
           height      = "40"
           xlink:href  = "tisch.svg" />
</defs>
<xfmgi:construct
   type       = "point"
   template   = "#img-template"
   parent     = "#new-elements"
   state      = "stopped">
   <xforms:trigger>
      <xforms:label>Start input</xforms:label >
   </xforms:trigger>
</xfmgi:construct>
```

Fig. 8. Example `construct` element referring to an SVG element as template.

How the constructed geometry data is applied to the newly generated geometric entity depends on the types of both, the `construct` and the template element. Consider the example of an SVG `image` element. If the `construct` element has the type `point`, the x and y coordinates of the image are set to the coordinates of the constructed point. For a `polyline`, the image is placed on every vertex, and finally, if it is of type `rect`, the image is scaled and positioned to fit inside the constructed rectangle.

Selection

The `select` control allows the selection of elements by the user. Subsequently the user may query information about the selected elements or modify them. Selected elements are usually highlighted by changing some of their style attributes or by adding decorations like a bounding box. Often a bounding box additionally displays some "grapples" where the user can drag the bounding box to change the size, position or orientation of the element. How the highlighting is performed, is – in correspondence to the XForms specification – out of the scope of XFormsGI.

By default, only elements having an `id` may be selected. It is task of the host language to control the selectivity of their elements. This can be done by (extensions to) the styling language (e.g. to Cascading Style Sheets) used in the host language like it is proposed in [2].

The main attributes of the `select` control are described in Table 2.

Table 2. Attributes of the `select` element.

Attribute	Description
`ref`	`ref` refers to the "selection set" in the instance data. The selection set is represented as a space-separated list of the identifiers of the selected elements.
`type`	`type` has alternatively the value `single` or `multiple`. In the first case, at most one element can be selected. In the second case, the selection set may contain an arbitrary number of elements.
`modifier`	Space-separated list of IDREFs referring to `modify` controls; explained in Section 0.

Figure 9 depicts an example for the `select` control.

```
<xfmgi:select
   type       = "single "
   model      = "#geomodel"
   ref        = "/data/selected-elements"
   modifier   = "#modify-placement #modify-delete
   state      = "start-implicit"
/>
```

Fig. 9. Example `select` element.

Modification

The `modify` control can be used either standalone or in conjunction with a `select` element to allow the modification of an existing graphical entity. Its main attributes are described in Table 3.

Table 3. Attributes of the `modify` element.

Attribute	Description
`type`	`type` defines the kind of modification that can be applied. It has one of the following values: `move-vertex`, `move-edge`, `move`, `rotate`, `resize`, `homogeneous-resize`, or `delete`.
`applyto`	`applyto` has alternatively the value `selected`, when the selected entity or entities should be modified, or, in the case of a stand-alone modifier, it consists of a space-separated list of IDREFs of the elements that are subject to the modification operation.
`description`	A description of the operation displayed to the user.
`ref`	Reference to the instance data element that stores the modified data.

Figure 10 shows an example of a selection control with three corresponding modify controls. The `modifier` attribute of the select control allows the direct "routing" of a selected element to one or more modification operations. A client may display the bounding box with the appropriate "grapples" (e.g., for scale and rotation) to give a hint to the user which modifications are possible. Figure 10 also depicts the interpretation of those controls by our prototype implementation. The `description` attributes are taken to be shown in a popup menu allowing the user to choose one of the three modifications.

```
<xfmgi:select
   type     = "single "
   modifier = "#modify-placement #modify-delete #modify-rotate"
   state    = "start-implicit"
/>

<xfmgi:modify id = "modify-placement"
   type        = "move-vertex"
   applyto     = "selected"
   description = "Verschieben"
/>
<xfmgi:modify id = "modify-rotate"
   type        = "rotate"
   applyto     = "selected"
   description = "Drehen"
/>
<xfmgi:modify id = "modify-delete"
   type        = "delete"
   applyto     = "selected"
   description = "Löschen"
/>
```

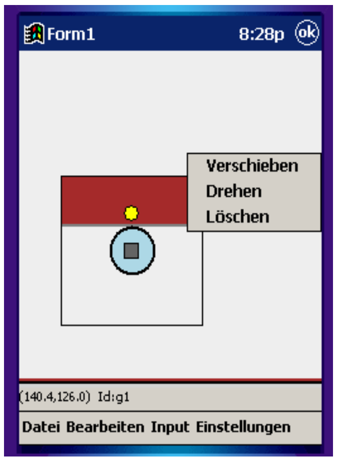


Fig. 10. Example of `selection` with corresponding `modify` elements.

3.2 Data Binding

XFormsGI uses the same data binding mechanism as XForms. Since geospatial data refers to spatial reference systems, we have to extend the XFormsGI data binding accordingly.

Coordinate Systems

In the context of GIS or LBS, we have to consider three coordinate systems:

- GPS-coordinates defining the current position (typically referenced by WGS'84 coordinates),
- the application reference system (e.g., a projected coordinate system), and
- the coordinate system of the host language.

The relation between the three systems can be described by two transformations T_1 and T_2 illustrated in Figure 11.

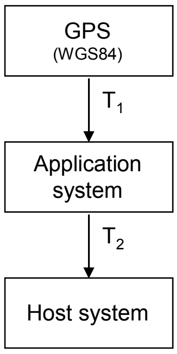


Fig. 11. Transformation chain.

The transformations are used in the following ways: GPS data (e.g., used to position a cursor depicting the current location) is mapped into the host system by first applying T_1 and then T_2. Visual construction or modification of an element by the user takes place in the host coordinate system. When coordinates are displayed numerically to the user or sent to the server for further processing, these coordinates have to be transformed to the application system by the inverse of T_2.

The SVG standard proposes to include information about the underlying geographic reference system into an SVG document by adding a metadata element containing an RDF description [11] of the coordinate reference system (see Section 7.12 "Geographic Coordinate Systems" in [12]). The coordinate system itself has to be defined by the "OpenGIS Recommendation on the Definition of Coordinate Reference Systems" [9]. This proposal leads to rather lengthy XML fragments. Instead, we introduced a simple XFormsGI element to describe the two transformations. The application system is identified by its EPSG number (see [5]) which establishes T_1. The mapping T_2 to host coordinates is described by a `transform` attribute which has the same functionality as the SVG `transform` attribute.

```
<xfmgi:coordinatereferencesystem
   crs-id     = "epsg:31467"
   transform  = "scale(1000,-1000) translate(-5308.5,-812)"
/>
```

Fig. 12. Example of the `coordinatereferencesystem` element.

Data Representation

If a `coordinatereferencesystem` element is present in the document, the instance data collects two sets of corresponding coordinate representations. The data is stored in two child elements of the reference element: the `hostcoords` and the `appcoords` element. An example is depicted in Figure 13. The data itself is stored as element content by a space-separated sequence of coordinates.

```
<data>
   <xfmgi:hostcoords>27.2 -157.4<xfmgi:hostcoords>
   <xfmgi:appcoords>8.19 53.14<xfmgi:appcoords>
</data>
```

Fig. 13. Example of coordinate data representing a single point.

The data representation according to the type of the geometric entity is straightforward, see Table 4.

Table 4. Data representation for selected geometric entities.

Type	Data	Description
`point`	`x y`	Coordinates
`line-segment`	x_0 y_0 x_1 y_1	Coordinates of start and end point
`circle`	`x y r`	Coordinates of center, radius

4 Sensors

Especially for mobile applications, the collection of data by sensors connected to the device becomes more and more important. For this purpose it is essential that XFormsGI supports this type of data capturing.

The `sensor` element is used to declare a “data source” which reads data from measuring equipment like GPS-sensors, compasses, velocity sensors, gas sniffers, barcode readers etc. The attributes of the `sensor` control are described in Table 5.

Table 4. Attributes of the sensor element.

Attribute	Description
`type`	Signifies the type of the sensor. The meaning of the other attributes depends on this value.
`source`	Either the name of the communication port used by the sensor – like e.g. COM1 and possibly further communication parameters – or the URL of a file containing emulation data. The source attribute may be omitted. In this case data is read from the standard port configured by the client.
`display-element`	Reference to host element used to visualize the readings
`cursor`	*For georeferenced sensors:* Optional. An IDREF to a host element used as a cursor to display the current location and track direction.
`speed-arrow`	*For moving georeferenced sensors:* Optional. An IDREF to a host element used to track speed (by scaling) and direction (may reference the same element as `cursor`).
`mode`	*For geo-sensors:* Optional. Any combination of following modes: `track-view`: keep the view centered on the current location; `track-angle`: cursor is rotated according to current heading; `track-velocity`: the speed arrow is scaled according to current velocity.

An example of a GPS sensor element is shown in Figure 14.

```
<xfmgi:sensor
   type        = "gps"
   ref         = "#gps-data"
   source      = "nmea2.txt"
   cursor      = "#gps-cursor"
   speed-arrow = "#gps-speed"
   mode        = "trackview trackangle"
   state       = "stopped"
/>
```

Fig. 14. Example of `sensor` element.

Additionally, metadata produced by the sensor may be placed in the instance data besides the coordinate information. For a GPS sensor, these may be height, date, and quality information like number of visible satellites, and so on. For the representation in the case of a GPS sensor, we choose the "way point element" `wpt` defined by the GPS exchange standard GPX, see [4]. Figure 15 shows an example of such instance data.

```
<xfm:instance>
   <gps-data>
      <xfmgi:hostcoords>8.19 -53.14</xfmgi:hostcoords>
      <xfmgi:appcoords>8.19 -53.14</xfmgi:appcoords>
      <gpx:wpt lat="8.190375" lon="53.132135">
         <gpx:time>4-04-07T15:23:32.809Z</gpx:time>
         <gpx:ele>62.100</gpx:ele>
         <gpx:sat>4</gpx:sat>
         <gpx:dhop>1.9</gpx:dhop>
      </gpx:wpt>
   </gps-data>
</xfm:instance>
```

Fig. 15. Example instance data for a GPS sensor.

5 Application

We outlined the concept of XFormsGI in the previous sections. In this section, we will shortly present a prototype, which serves to validate the concepts and which implements a specific XFormsGI user interface. The second objective is motivated by the fact that XForms leaves the details of the user interface to the implementation. The prototype has been developed as a mobile application running on a PocketPC PDA. SVG is used as host language of XFormsGI. Apart from the visualization of spatial data, the purpose of this application is the support for redlining and the input and editing of simple graphical elements.

The dominating input device for PDAs is the stylus. Therefore, the prototype is designed for supporting this type of input. The `input` and `construct` elements as described in Section 3.1 are implemented for the geometric primitives point, line segment, circle and polyline. Compared to the usage of a traditional mouse pointer on a desktop computer, the stylus has several restrictions. The prototype takes these specific characteristics into account. For example, the stylus lacks buttons and its position is only tracked when it touches the screen. As a consequence, a PDA is not able to distinguish between touches signifying for example the construction of a new point and simple display of the movement of the current track position.

In addition, the manipulation of existing geometric elements is addressed by the prototype. For example, it is possible to select one or more elements for deletion or geometric manipulation. At the moment dragging, insertion and deletion of vertices of a polyline are implemented. A GPS sensor element can be declared as described in Section 4.

Figure 16 shows a screenshot of a graphical editor defined by XFormsGI controls.

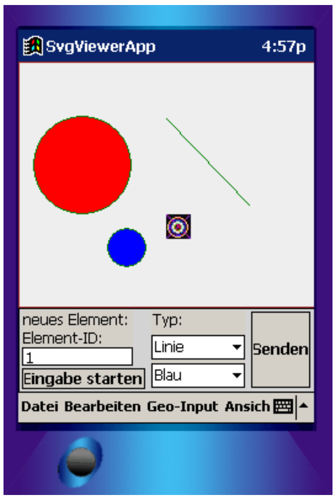


Fig. 16. Example of a complete graphical editor.

The integration of GPS sensors has been used for a tourist application: a historic city walk for the City of Oldenburg. This application offers the user historic (SVG and raster) maps of Oldenburg, corresponding textual information and shows the current position of the user by using the sensor element of XFormsGI. Figure 17 illustrates this application.

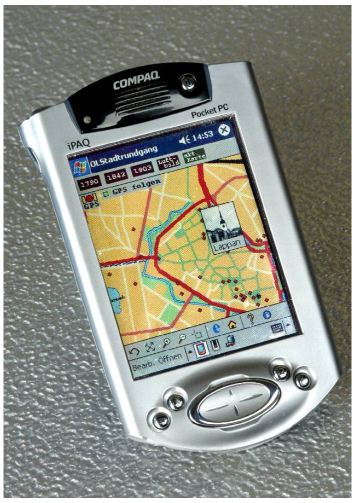


Fig. 17. Historic city walk application for the City of Oldenburg.

6 Related Work

The following three standards are the currently most prominent frameworks for developing form-based desktop and web applications:

- XUL is the *XML User-Interface Language* [7] developed by Mozilla. It allows the development of platform-independent graphical user interfaces.
- XAML (*Extensible Application Markup Language*) [6] is an XML application defined by Microsoft that allows the description of user interface similar to XForms for defining controls and to SVG for describing the visual aspects of the user interface. XAML is embedded into the broader framework .Net that provides the technical infrastructure for data exchange and visualisation. However, XAML is a proprietary standard and restricted to the next versions of the Windows operating system.
- *WebForms* [13] is a standard related to HTML forms and XForms. However, it is an extension to HTML forms that aims to simplify the task of transforming XForms into documents that can be rendered on HTML Web browsers that do not support XForms.

None of these standards support the input of geometric data. To the knowledge of the authors, the proposed XFormsGI is the only framework allowing descriptive definition for capturing geospatial and sensor data.

7 Conclusions

The presented XFormsGI framework allows the declarative definition of a graphical editor that needs no or only a small amount of scripting. Such an editor could be used, e.g., on a mobile device to collect field data. The representation by geographic coordinates is taken into account so that the seamless integration with a back-end application processing the captured data is possible.

For validating the concept of XFormsGI, a prototype has been implemented and integrated into a mobile SVG client [1] running on a PocketPC PDA.

A further feature of XFormsGI is the integration of sensors. In case of using SVG as host language for LBS, it is possible to make an SVG map location-aware by only adding two XFormsGI elements.

Several aspects of XFormsGI are still under development:

- In the current version, it is not possible to confine newly constructed points to predefined areas. This may lead to errors and inconsistent data. It would be desirable to have a means to define geometric input restrictions.
- In general, the construction of a new geometry is influenced by other geometries. As examples, consider the vertex of a building that has a certain distance from a given point, or the location of an accident on a road. In the simplest case, a new point could be snapped onto a given point. In analogy to the question of "selectibility" discussed in Section 3.1, this also requires extensions to the styling language used for the host document.
- In the context of sensors, a general method to map sensor measurements to element attributes of the host language is needed.

Acknowledgements
This work was funded by the Ministry for Research and Culture of Lower Saxony by the program "Praxisnahe Forschung und Entwicklung an niedersächsischen Fachhochschulen" and the German Federal Ministry for Education and Research by the program „FH^3 Angewandte Forschung an Fachhochschulen im Verbund mit der Wirtschaft". This support is gratefully acknowledged.

References

1. Brinkhoff, T., Weitkämper, J.: Mobile Viewers based on SVG$^{\pm geo}$ and XFormsGI. Proceedings 8th AGILE Conference on Geographic Information Science, Estoril, Portugal (2005) 599-604
2. Brinkhoff, T.: Towards a Declarative Portrayal and Interaction Model for GIS and LBS. Proceedings 8th AGILE Conference on Geographic Information Science, Estoril, Portugal (2005) 449-458
3. Fettes, A., Mansfield, P.A.: Creating a Graphical User Interface in SVG. Proceedings 3rd Annual Conference on Scalable Vector Graphics, Tokyo, Japan (2004)
4. GPX: The GPS exchange format. http://www.topografix.com/gpx.asp
5. International Association of Oil & Gas Producers, Surveying & Positioning Committee: EPSG Geodetic Parameter Dataset. http://www.epsg.org/
6. Microsoft MSDN: XAML Overview. http://windowssdk.msdn.microsoft.com/ library/default.asp?url=/library/en-us/wpf_conceptual/html/a80db4cd-dd0f-479f-a45f-3740017c22e4.asp
7. Mozilla.org: XML User Interface Language (XUL). http://www.mozilla.org/projects/xul/
8. Open Geospatial Consortium (OGC): OpenGIS Geography Markup Language (GML) Implementation Specification, Version 3.1.1, OGC 03-105r1 (2004)
9. Open Geospatial Consortium (OGC): Recommended XML/GML 3.1.1 encoding of common CRS definitions, OpenGIS Recommendation, OGC 05-011 (2005)
10. W3C: HTML 4.01 Specification, W3C Recommendation, 24 December 1999. http://www.w3.org/TR/html401
11. W3C: Resource Description Framework (RDF). http://www.w3.org/RDF/
12. W3C: Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation 14 January 2003. http://www.w3.org/TR/2003/REC-SVG11-20030114/
13. W3C: Web Forms 2.0, W3C Member Submission, 11 April 2005. http://www.w3.org/Submission/web-forms2/
14. W3C: XForms 1.0 (Second Edition), W3C Recommendation, 14 March 2006. http://www.w3.org/TR/2006/REC-xforms-20060314
15. W3C: XForms – The Next Generation of Web Forms. http://www.w3.org/MarkUp/Forms/
16. W3C: XHTML 1.0 The Extensible HyperText Markup Language (Second Edition), W3C Recommendation, 26 January 2000, revised 1 August 2002. http://www.w3.org/TR/xhtml1