GML2GML: Generic and Interoperable Round-Trip Geodata Editing – Concepts and Example

Matthias Merdes[1], Jochen Häußler[2], Alexander Zipf[3]
[1]EML Research gGmbH, Schloss-Wolfsbrunnenweg 33, Heidelberg, Germany
<firstname>.<lastname>@eml-r.villa-bosch.de
[2]European Media Laboratory GmbH, Schloss-Wolfsbrunnenweg 33, Heidelberg, Germany
<firstname>.<lastname>@eml-d.villa-bosch.de
[3]Dep. of Geoinformatics and Surveying, University of Applied Sciences, Mainz, Germany
zipf@geoinform.fh-mainz.de

SUMMARY

A range of tools offer possibilities for transforming geodata to SVG. We do not want to add yet another of such map generation projects, but investigate the general possibilities and restrictions for bidirectional XSL transformations between GML (Geographic Markup Language), SLD (Styled Layer Descriptor), and SVG. This is necessary in order to develop standards-based GIS applications supporting geodata-editing of geometry and attribute data. We explore the possibilities for a truly generic generation of these bi-directional transformations based on the GML application schema. This results in a cascade of several XSL transformations from the database schema to XSL code that finally transforms between GML data and SVG in both directions taking the style descriptions from SLD documents.

KEYWORDS: *SLD, WFS, GML, SVG, open standards, XSLT, round-trip transformation*

INTRODUCTION

A considerable range of spatial web service standards has been available for some time from the OpenGIS Consortium (OGC). On the other hand there are general standards by the W3C for developing web-based systems suggesting a potential for web-based geo-applications utilizing only OGC and W3C standards. But do these standards really fulfill all requirements of GI applications? The approach seems to work with simple mapping systems, but is it also possible to develop more sophisticated applications based completely on such standards? Can the combined application of OGC and W3C standards achieve synergy effects in more complex situations and specialized applications? What are current restrictions? We have investigated such issues within the framework of a generic application for mobile online editing of vector-based geodata. This includes the OGC Web Feature Service (WFS), GML, the OGC Filter Encoding, and the OGC SLD specifications and W3C standards like Scalable Vector Graphics (SVG), XSLT, and Cascading Style Sheets (CSS). In this context the applicability of XSLT to transform between various representations deserves major attention. It is of particular interest how GML can be transformed to SVG in a generic manner using only GML application schemas and SLD descriptions as knowledge sources which encapsulate the domain and application knowledge.

Rendering, Styling, Editing, and Writing Geodata online

Related work has so far concentrated on partial aspects of our approach, i.e. there are many projects that use XSLT to transform GML to SVG (Spanaki et al., 2003), or SLD for proprietary rendering mechanisms. Others focus on interoperability through OGC standards, e.g. (Boucelma et al., 2002), (Shekhar et al., 2001), (Misund & Lindh, 2004). A recent project that also handles web-based and mobile geodata-editing based on GML is presented by (Cheung, 2004). Our work however integrates the existing knowledge sources, namely application schemas and SLD styling descriptions to generate standards-based layout (SVG) and styling (CSS) specifications. This achieves a clean separation of concerns in the resulting products mirroring the separation of data (GML) and styling (SLD) concerns in the input sources. Additionally, we investigate the reverse transformation from SVG to GML

embedded in WFS transactions. It is thus necessary not only to handle purely geometric data but also attribute data. We will show that a client-side data management and WFS request generation is possible using SVG metadata and XSL transformations. We also investigate the issues connected with OGC filter encoding expressions embedded in SLD descriptions.

The aim of software engineering to model applications on an abstract conceptual level and derive applications from that has lead to concepts like Model Driven Architecture (MDA) applied to GI by (Grønmo, 2001). We now focus on the generic generation of SVG-based applications from GML application schemas. (Gnägi & Morf, 2002) as well as (Spanaki & Tsoulos, 2003) think in a similar direction, yet with different focus. While a lot of work has been done on generating SVG from GML for mapping purposes, we are not aware of systems or projects covering all of the mentioned aspects of our investigation into bi-directional generic and open standards-based vector-geodata-editing. Through the combination of the technologies used (XSL, SVG), the consistent use of open GI standards (GML, SLD, WFS), the support of reading operations as well as writing transactions using online geo-databases and the integration with a mobile geodata editor our achievements can be distinguished from other work. Furthermore, our work aims not only at web-based applications but supports (mobile) geodata-editing.

The remainder of this paper is structured as follows. We will describe the round-trip transformation form GML and SLD to SVG and CSS. Before presenting an extensive real-world example we will discuss the role of SLD in the SVG/CSS generation process.

TRANSFORMING FROM GML AND SLD TO SVG AND CSS

It is a goal for the map generation process to be completely parametrizable by an SLD document which contains styling information for the geodata and an application schema which contains the data model. With these two knowledge sources it should be possible to render the geodata queried from a WFS correctly as a map. A further requirement is the capability to process the data on the client side and write it back to the data source. The principal procedure is as follows:

1. Parameterization of the application by an SLD document and a data schema from a WFS
2. Query of the geodata from the WFS
3. Rendering of the geodata as a map while preserving the semantic representation of the data
4. Editing of the queried data or acquisition of new geodata
5.Writing back of modified data to the WFS with a WFS:Transaction request

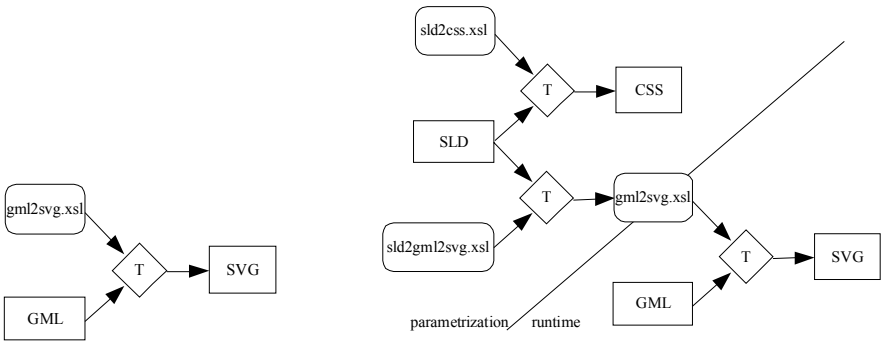

Figure 1: left: Static transformation of GML to SVG with a XSLT; right: Cascade of XSL transformations for the generation of SVG from GML with styling information from SLD.

Transformation between Data, Styling & Rendering Documents

The application of SVG implies that the geodata (GML) must be transferred into an SVG representation with, e.g. an XSL transformation or alternative technology (Jolif, 2003). In our case a WFS:GetFeature response containing a GML FeatureCollection must be transferred into an SVG

document. There are several examples of XSL style sheets (e.g., Spanaki, 2003) which transform GML to SVG (often called gml2svg.xsl). In the simplest case these are static scripts which transform a well-known GML structure into an SVG representation embedding a fixed presentation style as shown in Figure 1 (left part).

In the case of a generic application the gml2svg transformation script cannot be a static document because both the structure of the GML data as well as the styling of the presentation depend on the specific application. The script itself must rather be computed dynamically. As the presentation information is contained in an SLD document it is possible to generate the gml2svg script in another XSL transformation from the SLD document. As both, the CSS document and the gml2svg script are generated from the same SLD document instance it is possible to insert references to CSS classes into the SVG document. Figure 1 (right part) shows the three XSL transformations differentiating between 'runtime transformation' which is executed after a concrete data query and 'parametrization transformation' which can be executed at configuration time.

The non-geometrical attributes of features may be relevant in an SLD document in connection with attribute-dependent styling and are then parameters to a FilterEncoding definition in the SLD document. As an SLD document does not necessarily cover all FeatureTypes offered by the WFS, knowledge about the exact data model is required for the transformation script gml2svg in addition to the SLD styling information. This knowledge is contained in an XML schema from the WFS defining the FeatureTypes. Figure 2 shows the complete transformation cascade which must be executed in order to display geodata from a WFS as an SVG document based on an XML schema from the WFS and an application-specific SLD document in a completely generic way. This enables further processing of the geodata at a semantic level.

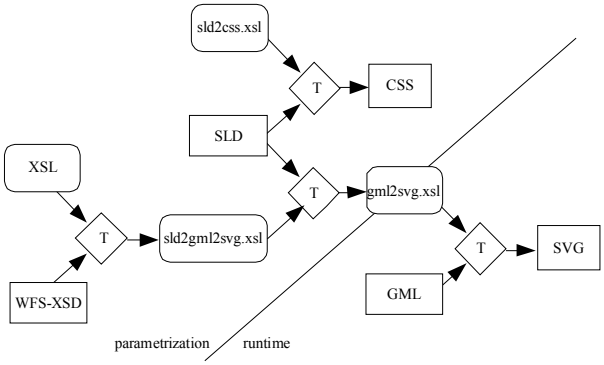

Figure 2: Complete generic parameterized GML2SVG transformation cascade.

According to the WFS specification clients query a WFS with GetFeature requests and receive a GML FeatureCollection. Thus, in order to transform a FeatureCollection to SVG all GML2 geometry types must be representable by SVG constructs. XSL scripts transforming GML documents of well-known structure to SVG have been available for some time. A generic transformation however must be able to handle all possibilities including varying formulations for various GML constructs. The definition of the presentation (as SLD) can partly be factored out into CSS. As the styling and hence the assignation of a CSS class to a feature may not only depend on its type but also on other geometrical and non-geometrical properties this assignation is non-trivial.

SVG Metadata as a Semantic Description of Geodata

As SVG is a generic format for vector graphics it does not provide built-in means to represent non-spatial data along with the geometric information. From a purely geometric point of view this additional attribute data can be seen as a form of metadata providing additional information beyond the

mere spatial representation of features. SVG does however provide a general-purpose element SVG:metadata which can also be used to store non-geometric data and is embedded in arbitrary geometric SVG elements. For other approaches see (Chuang, 2003). Once such a mechanism is in place it can also be used to store various kinds of other information such as editing status.

When storing GML data in metadata elements spatial and non-spatial data must be distinguished. Non-spatial attributes of features can safely be copied into SVG:metadata elements. Applying the same strategy to the geometric data of a feature leads to redundancy as the geometric data is already represented as an SVG geometry. This inflates the SVG document but makes the transformation from SVG back to GML easier since an unchanged geometry does not necessarily have to be transformed again. It has to be taken into account, that optional elements may not be present in all individual feature instances and therefore not in the SVG metadata either. This can constitute a problem if this metadata structure is used for other purposes such as editing of the attribute data (without explicit knowledge of the data schema). If further processing of the generated SVG document requires optional elements, the XSL transformation must be adapted more precisely to the structure of the various feature types. In this case the complete GML feature structure must be taken over into the SVG metadata element by element making the generation of the gml2svg transformation script more complex.

The Inverse Transformation

To complete the round-trip transformation we have to study the way back from the client to the server where all changes are stored persistently. This procedure is easier than the direction from GML to SVG as all styling information is irrelevant to persistence. As our approach includes a WFS as data source the data edited on the client has to be transformed into a WFS:Transaction request. It allows to insert, update, or delete schema-compliant features.

To include only the edited, deleted, or inserted features into a transaction request it is necessary to distinguish between edited and unchanged features. Thus, we must store information about the editing history of each feature - or even each feature property - within the metadata element, together with the feature properties which are nested under the metadata tag. When the changes are submitted by the user the SVG document can be examined for edited features. If the geometry of a feature has been edited or newly created this geometry has to be transformed to the corresponding GML geometry, e.g. by an XSL transformation, which is straightforward. Since the non-geometric feature properties are stored as original GML structures within the SVG:metadata element they can be used in the transaction without further transformation.

THE ROLE OF SLD IN THE SVG GENERATION PROCESS

Deriving Visible Structures from SLD

Deriving visible structures from SLD definitions includes two tasks:

1. Transforming simple style attributes from SLD to the appropriate CSS counterparts.
2. Expressing more complex SLD descriptions by creating SVG constructs, possibly factoring out stylistic information to CSS.

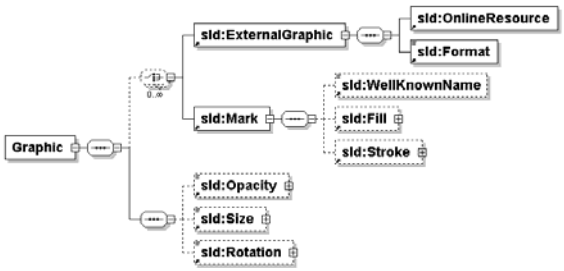

Figure 3: Schema Visualization of SLD:Graphic.

As an example of an SLD element which cannot be simply transformed to into a CSS class and a corresponding reference in the SVG document we describe the mapping of the SLD:Graphic element to a mixture of SVG and CSS structures. The SLD:Graphic element models a graphical symbol or icon as depicted in figure 3. It features a number of embedded elements for opacity, size, and rotation and can be defined by referencing external images. Alternatively, an SLD:Mark element may be used which contains a reference to a predefined so-called "well-known" symbol such as 'circle', 'star', 'square', 'triangle', 'cross' and 'x'.

Document type	Example document fragment
SLD (input)	<Graphic> <Mark> <WellKnownName>square</WellKnownName> <Fill>...</Fill> <Stroke>...</Stroke> </Mark> <Opacity>0.3</Opacity> <Size>16.0</Size> <Rotation>45.0</Rotation> <Graphic>
SVG (output)	<defs> <symbol id="square"> <rect width="16" height="16" /> </symbol> </defs> <use class="Class1" transform="rotate(45.0 X+8 Y+8)" x="X" y="Y" xlink:href="#square" />

CSS
(output)
	.Class1
	{
	//styles from <Fill> and <Stroke>
	opacity: 0.3;
	}

***Table 1**:* Example of corresponding SLD, SVG, and CSS fragments for SLD:Graphic.

The SLD fragment in Table 1 shows a SLD:Graphic instance containing the well-known symbol 'square' with opacity of 0.3, size of 16 pixels, and rotation of 45 degrees. Some of the source structures (symbol type, rotation, size) will appear directly in the resulting SVG document, whereas all CssParameters embedded in the SLD:Fill and SLD:Stroke elements must be transferred into the CSS document. The 'X' and 'Y' place holders in the SVG:use/@transform attribute are the two spatial coordinates from the underlying geometry defined in the GML file currently transformed.

OGC:Filters Embedded in SLD documents

SLD documents not only contain rendering specifications such as styles for lines or points but these specifications may also be restricted to certain selections of the geodata in question. For the specification of these selections the general-purpose mechanism of OGC:Filters is employed. Filters include Egenhofer, comparison, logical, arithmetic, and feature selection operators. In the following we briefly discuss the processing of such Filter elements within SLD documents.

OGC:Filter elements within SLD:Rule elements cannot be transformed to SVG/CSS structures directly, but have to be mapped to corresponding XPath expressions for match-template clauses in XSLT transformations which in turn must be applied to GML. The following example illustrates a non-spatial filter:

```
<ogc:Filter>
  <ogc:PropertyIsLessThan>
    <ogc:PropertyName>width</ogc:PropertyName>
    <ogc:Literal>2.0</ogc:Literal>
  </ogc:PropertyIsLessThan>
</ogc:Filter>
```

In order to generate different CSS classes and corresponding references in the SVG file the filter has to be transformed to an XPath condition, based on which certain features can be selected in the incoming GML data set by an XSLT script.

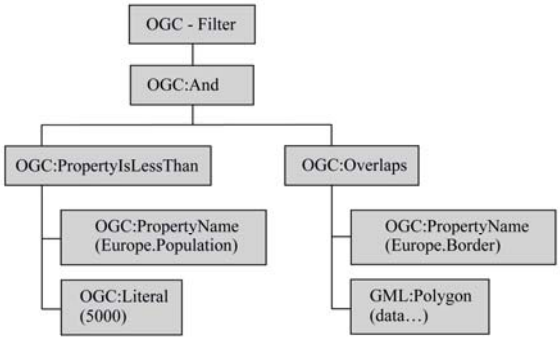

Figure 4: Example for a filter combining comparison and spatial operators.

Arbitrarily complex query conditions can be modeled by recursive nesting, resulting in a tree structure (see figure 4). For a working XSLT-based OGC:Filter evaluation engine a mapping between the mentioned filter operators and their semantically equivalent counterparts in XSLT is needed. This mapping is straightforward for comparison, logical, and arithmetic operators, and feature selection. Spatial operators however include spatial and topological computations typically executed by a WFS. While theoretically possible due to XSLT's Turing-completeness (Kepser, 2004), it is not desirable for a light-weight client-side OGC:Filter evaluation engine to reimplement spatial computation capabilities. The XSLT processing engine could be extended with a generic framework which converts the spatial operations within the filter expressions to WFS requests and executes them. This is possible with standardized hooks yet lacking performance. We can conclude that a powerful OGC Filter Encoding evaluation engine can in general be realized with XSLT which is particularly helpful for XSL transformation of SLD documents. While even spatial filters can be processed, a reasonable implementation is only practical for logical, comparison, arithmetic, and ID selection operators.

A REAL-LIFE EXAMPLE

With the following example from the research project "Advancement of Geoservices" (Breunig, 2003) we want to illustrate our conceptual investigations. In this project, a mobile data acquisition system for geodata is developed (Häußler, 2004). The application scenario is a road in southern Germany threatened by landslides. The mountain above the road is monitored by a sensor network registering sudden movements of the slope. The area is covered with clefts indicating the continuous movement of the whole slope. The measuring devices are located in some of the biggest clefts. If conspicuous extensions of a monitored cleft are registered, the road is closed immediately. An expert geologist uses online access to the geodata in the field.

The core client component, a Java editor for geodata, receives GML feature collections by a WFS with 15 feature types. The names of the feature types can be requested from the WFS with a WFS:GetCapabilities request. In our example we use a feature type for different types of ways and

streets named "Way". By sending a WFS:DescribeFeatureType request to the WFS, the following schema document will be received (the listing is only a part of the whole schema):

```
<xsd:element name="Way" substitutionGroup="gtp:_DistrictFeature">
  <xsd:complexContent>
    <xsd:extension base="gtp:AbstractGeoFeatureType">
      <xsd:sequence>
        <xsd:element name="multiCenterLineOf" type="gml:MultiLineStringPropertyType"/>
        <xsd:element name="width" nillable="true" type="xsd:float"/>
        <xsd:element name="category">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="asphaltierter Weg"/>
              <xsd:enumeration value="Fussweg"/>  …
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:element>
```

Here, the attributes of the feature type are defined: the name of the attribute containing the geometry is ‘multiCenterLineOf’, furthermore there are the non-geometric attributes ‘width’ and ‘category’. Since the element is an extension of the complex type gtp:AbstractGeoFeatureType (the namespace ‘gtp’ is project -specific) ‘gml:description’, ‘gml:boundedBy’ and ‘gml:name’ are inherited.

The second knowledge source that is required to parametrize the application for this scenario is the SLD document. The following listing shows part of the SLD for the feature type ‘Way’:

```
<StyledLayerDescriptor version="1.0.0">
  <UserLayer>
    <Name>LAYER_Way</Name>
    <LayerFeatureConstraints>
      <FeatureTypeConstraint>
        <FeatureTypeName>Way</FeatureTypeName>
      </FeatureTypeConstraint>
    </LayerFeatureConstraints>
    <UserStyle>
      <Title>WayClass</Title>
      <FeatureTypeStyle>
        <Rule>
          <LineSymbolizer>
            <Geometry>
              <ogc:PropertyName>multiCenterLineOf</ogc:PropertyName>
            </Geometry>
            <Stroke>
              <CssParameter name="stroke">blue</CssParameter>
              <CssParameter name="stroke-linejoin">round</CssParameter>
              <CssParameter name="stroke-width">0.4</CssParameter>
              <CssParameter name="fill">none</CssParameter>
            </Stroke>
          </LineSymbolizer>
```

```
            </Rule>
          </FeatureTypeStyle>
        </UserStyle>
      </UserLayer>
    </StyledLayerDescriptor>
```

This is the easiest possible form of a styling definition since the feature type is always styled the same way independently of attribute values or more complex filter encoding rules. As in this simple case the SLD definition maps to CSS only and not to a mixture of CSS and SVG structures the following simple CSS fragment can be generated from the SLD document:

```
g.WayClass { stroke-linejoin: round; stroke-width: 0.4; stroke: blue; fill: none }
```

Rules for the gml2svg transformation script can be derived from the SLD document that define the order of the different layers to be drawn and the CSS styles, markers, etc. to be used to render each feature type. The following listing shows a small part of the gml2svg script:

```
<xsl:if test="count(descendant::Way) > 0">
  <svg:g id="LAYER_Way" class="WayClass">
    <xsl:call-template name="makeSVGGroup4Feature">
      <xsl:with-param name="FeatureName">Way</xsl:with-param>
      <xsl:with-param name="ElementName_title">gml:description</xsl:with-param>
    </xsl:call-template>
  </svg:g>
</xsl:if>
```

The XSL:if element starts at the feature collection node of the WFS feature collection. If one or many features of type ‘Way’ are found in the feature collection, an element SVG:g representing a group of ‘Way’ features is generated. The values for the attributes ‘id’ and ‘class’ come from the SLD document. The generic template ‘makeSVGGroup4Feature’ is called with the feature name as parameter. This template generates SVG representation of GML features, detecting the name of the geometry attribute and other attributes itself. In our application, this generic template can be used because in the schema definition of the feature type ‘Way’ there is no optional element. If it is important for the application that the SVG document also contains the optional elements, a template would be required which generates also optional elements into the SVG document, even if a feature from a concrete feature collection doesn't contain this element.

The following listing shows a feature of type ‘Way’ in its GML representation, as received by a WFS:GetFeature request:

```
<Way fid="Way_10" >
  <gml:description">E74</gml:description>
    <gml:name"/>
    <gml:boundedBy"/>
    <multiCenterLineOf">
      <gml:MultiLineString>
        <gml:lineStringMember>
          <gml:LineString>
            <gml:coordinates cs="," decimal=".">3492638,5341516 …</gml:coordinates>
          </gml:LineString><
        </gml:lineStringMember>
        <gml:lineStringMember>  …  </gml:lineStringMember>
      </gml:MultiLineString>
```

```
        </multiCenterLineOf>
        <width"/>
        <category">Fussweg</category>
    </Way>
```

It is transformed by the gml2svg transformation to this SVG sub-tree:

```
<svg:g id="LAYER_Way" class="WayClass">
    <svg:g id="Way_10" class="originalFeature">
        <svg:title>Way - Way_10 - E74</svg:title>
        <svg:g id="Way_10_GEOMETRY">
            <svg:path d="M3492638.018,5341516.323L3492638.207,5341516.45 ..."/>
            <svg:path d="M3492637.985,5341517.046L3492638.159,5341517.164 ..."/>
        </svg:g>
        <svg:metadata>
            <eml:featureTypeName>Way</eml:featureTypeName>
            <eml:featureID>Way_10</eml:featureID>
            <eml:geometryElementName>multiCenterLineOf</eml:geometryElementName>
            <eml:geometryType>gml:MultiLineString</eml:geometryType>
            <eml:featureAttributes>
                <eml:featureAttribute>
                    <gml:description>E74</gml:description>
                </eml:featureAttribute>
                <eml:featureAttribute> <gml:name/> </eml:featureAttribute>
                <eml:featureAttribute> <gml:boundedBy/> </eml:featureAttribute>
                <eml:featureAttribute> <width/> </eml:featureAttribute>
                <eml:featureAttribute> <category>Fussweg</category> </eml:featureAttribute>
            </eml:featureAttributes>
        </svg:metadata>
    </svg:g>
</svg:g>
```

The SVG representation of the feature contains all information that is necessary for editing and transforming back to a WFS:TransactionRequests including GML. Due to space limitations we could only showed a small excerpt of this application example. The quantitative and qualitative complexity of the full scripts makes the described automatic generation of the transformation very valuable.

CONCLUSION

We have presented an analysis of the possibilities of a generic bi-directional conversion between GML&SLD and SVG&CSS using XSL transformations – thus based only on open standards by the OGC and W3C. Using the self-describing mechanisms of the services and the possibility to query the structure of the geodata schema using OGC Web Feature Servers, we have shown that it is possible to transform between geodata delivered by WFS and SVG in both directions without the necessity of application developers to write any application or domain specific code. This is a major improvement over the multiple transformations specialized for a particular database schema available so far, which also only realize the direction from GML to SVG. A further advantage of our approach is the clean separation of raw geodata (GML) and map styling information (SLD). Furthermore, in the SVG document as much styling information as possible is outsourced into CSS style sheets. We also do not mix code for transforming geometry and styling information but generate the appropriate code automatically from the appropriate descriptions. The rule-based grammar of XSL eases the definition of conversion rules compared to conventional programming that needs to traverse the XML tree.

Our work is being validated with the example of a mobile geodata editor, which supports online

geodata editing in the field (Häußler, 2004). While still some steps away from the proposals of (Fonseca et al., 2002) regarding the development of GIS from formal ontologies the overall concept of the presented work is similar in the sense that specific GI applications (i.e., maps and editors) are automatically generated from formal descriptions (application schemas and styling specifications) of geodata without prior knowledge of the data sources themselves. Of course there is still a far way to go from these schemas – as supported by today's standards – to actual domain ontologies. But from a standards-based interoperability point of view this is what we can achieve today.

ACKNOWLEDGEMENTS

This is publication no. GEOTECH-122 of the program GEOTECHNOLOGIEN of BMBF and DFG, Grant 03F0373E. The work described was funded by BMBF and the Klaus Tschira Foundation (KTS). We also thank the anonymous reviewers for suggesting improvements to the paper.

BIBLIOGRAPHY

Boucelma, O., Essid, M., Lacroix, Z.: A WFS-based Mediation System for GIS Interoperability. Agnès Voisard, Shu-Ching Chen (eds.) : ACM-GIS 2002, Proceedings of the Tenth ACM International Symposium on Advances in Geographic Information Systems, McLean, VA USA, ACM, 2002.

Breunig, M., Malaka, R., Reinhardt, W., Wiesel, J.: Advancement of Geoservices. Geotechnologien Science Report No. 2, Information Systems in Earth Management, Potsdam, 3750, 2003.

Cheung, I.: Modelling GML data for the spatial data presentation and editing on the web and mobile devices. GML and Geo-Spatial Web Services Conference 2004. Vancouver, Canada, 2004.

Chuang, T.-R., Chang, Y.: Embedding Domain Semantics in SVG. SVG Open 2003. Vancouver, Canada, 2003.

Fonseca, F., Egenhofer, M., Agouris, P. and Câmara, G.: Using Ontologies for Integrated Geographic Information Systems. Transactions in GIS, 6(3): 231-257, 2002.

Gnägi, R. and Morf, M.: Generating SVG from System Independent Conceptually Modeled Geodata. SVG Open 2002. July 15-17, 2002. Zurich, Switzerland, 2002.

Grønmo, R.: Supporting GI standards with a model-driven architecture. ACM-GIS 2001, Proceedings of the Ninth ACM International Symposium on Advances in Geographic Information Systems, Atlanta, GA, USA, :100-105, 2001.

Häußler, J., Merdes, M., Zipf, A.,: A Graphical Editor for Geodata in Mobile Environments. In: "Science Report" No. 4. Potsdam: Koordinierungsbüro GEOTECHNOLOGIEN: 55-58, 2004

Jolif, C.: Comparison between XML to SVG Transformation Mechanisms - The GraphML format use-case. SVG Open 2003. Vancouver, Canada, 2003.

Kepser, S.: A Simple Proof of the Turing-Completeness of XSLT and XQuery. In Tommie Usdin (ed.). Extreme Markup Languages, 2004.

Misund, G., Lindh, M.: Implementing the Open Mobile GeoWeb - Some Exercises. In: Jiang, B. and Zipf, A. (eds.): Special issue on "LBS and ubiquitous GIS" with the Journal of Geographic Information Sciences. CPGIS. Berkeley. California, 2005.

Shekhar, S., Vatsavai, R.R., Sahay, N., Burk, T. E. and Lime, S.: WMS and GML based Interoperable Web Mapping System. ACM-GIS 2001, Ninth ACM International Symposium on Advances in Geographic Information Systems, Atlanta, GA, USA:106-111, 2001.

Spanaki, M., Tsoulos, L.: A Holistic Approach of Map Composition Utilizing XML. SVG Open 2003. Vancouver, Canada, 2003.