# DIGGS 1.0a Schema Evaluation

***Final Report***

Revision 1.0

Submitted by:


## Compusult Limited

40 Bannister Street
Mount Pearl, NL
A1N 1W1
Canada

Contact: Mr. Paul Mitten, Vice-President

Toll-free: 1-888-388-8180
Telephone: 709-745-7914
Fax: 709-745-7927
E-mail: marketing@compusult.net

Document No. 09012-003

August 6, 2009

Copyright © 2009 Compusult Limited

## DOCUMENT APPROVAL SHEET

| Approvals | Signature | Date |
|---|---|---|
| QA Manager | Wayne Emberley | August 6, 2009 |
| Project Manager | Paul Mitten<br>Vice-President | August 6, 2009 |

 Copyright © 2009 Compusult Limited 

## DOCUMENT REVISION HISTORY

| Revision | Reason for Change | Origin Date |
|---|---|---|
| 1.0 | Draft Document issued | August 6, 2009 |

 Copyright © 2009 Compusult Limited 

# Table of Contents

**1.0 INTRODUCTION** ... 1-1
1.1 XML Schema Objectives and Limitations 1-1
1.2 Document Layout 1-2

**2.0 CORE ISSUES** 2-1
2.1 Issue Rating Metric 2-1
2.2 Assessment Methodology 2-2
2.3 Recursion of DIGGS Objects 2-3
2.3.1 Summary 2-3
2.3.2 Recommendation 2-7
2.3.3 Rating 2-8
2.4 Table Object Implementation / Table Object Data Validation 2-8
2.4.1 Summary 2-8
2.4.2 Recommendation 2-11
2.4.3 Rating 2-14
2.5 Code Table Implementation 2-14
2.5.1 Summary 2-14
2.5.2 Recommendation 2-18
2.5.3 Rating 2-19
2.6 Key Fields / Use of Unique Identifiers 2-19
2.6.1 Summary 2-19
2.6.2 Recommendation 2-22
2.6.3 Rating 2-24
2.7 Creating Empty XML Files 2-24
2.7.1 Summary 2-24
2.7.2 Recommendation 2-25
2.7.3 Rating 2-25
2.8 GML Implementation 2-26
2.8.1 Summary 2-26
2.8.2 Recommendation 2-27
2.8.3 Rating 2-30
2.9 Flattening DIGGS Files 2-30

 Copyright © 2009 Compusult Limited 

2.9.1 Summary ... 2-31
2.9.2 Recommendation ... 2-32
2.9.3 Rating ... 2-33
2.10 Naming Convention for Related Elements ... 2-33
2.10.1 Summary ... 2-33
2.10.2 Recommendation ... 2-33
2.10.3 Rating ... 2-34
2.11 Inheritance of Objects ... 2-34
2.11.1 Summary ... 2-34
2.11.2 Recommendation ... 2-38
2.11.3 Rating ... 2-40

**3.0 PERFORMANCE REQUIREMENTS** ... 3-1
3.1 Ability to be Mapped to a Relational Database ... 3-1
3.2 Ability to Work with Leading XML Editing Tools ... 3-2
3.3 Ability to Generate Blank DIGGS XML Files Using Standard XML Editing Tools 3-3
3.4 Demonstrate Import / Export Between Vendors - Ability to Exchange Examples ... 3-4

**4.0 ASSESSMENT OF GML USAGE** ... 4-1
4.1 Were GML Standards, Conventions, and Best Practices Implemented Correctly? ... 4-1
4.2 Is DIGGS Best Implemented as a GML Application Schema? ... 4-4

**5.0 WEB-BASED COMMUNITY SCHEMA DEVELOPMENT** ... 5-1

**6.0 OTHER ISSUES** ... 6-1
6.1 Validation Issues in Oxygen ... 6-1
6.1.1 Summary ... 6-1

**7.0 ASSESSMENT OF SCHEMA ORGANIZATION AND COMPOSITION** ... 7-1
7.1 Assess Schema Organization and Composition Strategy ... 7-1

**8.0 SUMMARY OF RECOMMENDATIONS** ... 7-1

 Copyright © 2009 Compusult Limited 

# 1.0 INTRODUCTION

This document presents Compusult Limited's final report on the DIGGS version 1.0a Schema.

DIGGS version 1.0a is a Geographic Markup Language (GML) feature application schema based on GML version 3.1.1. The initial implementers identified various difficulties and issues with the schemas. The DIGGS online forums list several Core Issues that have been identified as the most important potential problems with the schemas. The DIGGS stakeholders have identified several Performance Requirements as critical objectives that must be achieved before the next version of the DIGGS Schema can be released.

This report provides the following:

- a description of our investigation into each Core Issue;
- a discussion of ways to achieve the Performance Requirements;
- an assessment of GML usage within the DIGGS 1.0a Schema;
- a discussion on the use of an online collaborative schema development tool; and,
- presentation of a subset of the DIGGS 1.0a Schema, based on the DIGGS Hole object, that we developed to support recommendations.

Our overall recommendation and conclusion is that the DIGGS schemas should reduce the current level of complexity as much as possible. The primary sources of complexity are the extensive use of inheritance within DIGGS. Additional complexity is added via the extensive inheritance from GML objects. Even though GML is essentially a secondary, supporting set of schemas for DIGGS, the implementation of DIGGS as a GML feature application schema adds considerable complexity, ties DIGGS extremely tightly to GML, and requires a significant understanding of GML to employ DIGGS. There is still a role for GML within DIGGS, but in less complex and more relevant aspects.

## 1.1 XML Schema Objectives and Limitations

XML Schema is used to define the structure of an XML document that is expressive enough to meet the needs of the application domain, while being restrictive enough to disallow anything that does not make sense for the application domain. The benefit is that a schema-valid XML document implies a logically-valid document for the application domain. General purpose XML Schema libraries are widely available to validate an XML document against a schema. Producers and consumers can schema-validate an XML instance to help ensure they have a logically valid and likely correct document.

In reality, XML Schema is not expressive enough to define all "business rules" of any particular application domain. It is possible to produce a schema-valid document that does not make sense for the application domain. For example, XML Schema does not support forcing the value of a

 Copyright © 2009 Compusult Limited 

given element or attribute to depend on the value of another element or attribute. Sometimes these limitations can be resolved via external facilities such as Schematron. However, this adds complexity and is not as widely supported as native XML Schema validation.

For schema designers, this results in a trade-off between a schema that is expressive enough to meet the potentially varied requirements of the application domain, but restrictive enough so that a schema-valid document is practically a logically-valid document.

The closer a schema-valid document is to a logically-valid document, the simpler the document format. In particular, the goal is to use the XML Schema document to understand what makes a logically-correct document for the application domain. Note that any rules that cannot be expressed in XML Schema must be formally documented outside the XML Schema document. XML Schema documents, and the XML Schema standard itself, have been criticized for being difficult to read and understand.

## 1.2 Document Layout

The remainder of this document is structured as follows:

- **SECTION 2.0 CORE ISSUES** - A discussion of the DIGGS 1.0a Core Issues from the DIGGS online forum.
- **SECTION 3.0 PERFORMANCE REQUIREMENTS** - A discussion of ways to help ensure the next version of the DIGGS Schema will meet the critical objectives.
- **SECTION 4.0 ASSESSMENT OF GML USAGE** - An assessment of the use of GML in the DIGGS 1.0a Schema.
- **SECTION 5.0 WEB-BASED COMMUNITY SCHEMA DEVELOPMENT** - A discussion of online schema collaboration.
- **SECTION 6.0 OTHER ISSUES** - A discussion of other issues associated with the DIGGS 1.0a Schema.
- **SECTION 7.0 ASSESSMENT OF SCHEMA ORGANIZATION AND ISSUES** - An assessment of the organization of the schemas in terms of the DIGGS goals.
- **SECTION 8.0 SUMMARY OF RECOMMENDATIONS** - A summary of the recommendations provided throughout this document.

 Copyright © 2009 Compusult Limited 

# 2.0 CORE ISSUES

The DIGGS online forum lists eleven Core Issues issues that have been identified as significant within the DIGGS 1.0a Schema. These are discussed in the following sub-sections, where each issue is rated according to the metric described in Section 2.1 and assessed according to the methodology described in Section 2.2.

## 2.1 Issue Rating Metric

The significance of the Core Issues was rated based on three components: Importance, Scope/ Difficulty of Change, and Time Estimate, as described below:

**Importance** - Indicates how important resolving the given issue is to the success of DIGGS. The importance of an issue is given one of three priorities:

1. **Very Important:** This issue must be addressed for the DIGGS schemas to be successful.
2. **Medium Importance:** Without addressing this issue, using the schemas will be more difficult, time-consuming and/or cumbersome.
3. **Minor Importance:** Minor changes, such as XML tagname spelling convention.

**Scope/Difficulty of Change** - Indicates the magnitude of the change to the schemas.

1. **Large Impact:** Radical changes to the schemas affecting the structure of the schema such that implementers will likely have to drastically rework any software that produces or consumes the schemas. High level design work may be required to re-engineer the change.
2. **Medium Impact:** Changes that will affect many schema files, but are mostly a mechanical change that may be implemented via a script.
3. **Low Impact:** Cosmetic changes, such as altering the naming convention of attributes. Such changes should not change the logic of any software that produces or consumes the schema.

**Time Estimate** - Approximately how much time (i.e., level of effort in days) would be required to change the schemas.

 Copyright © 2009 Compusult Limited 

## 2.2 Assessment Methodology

The DIGGS schema issues and proposed solutions were assessed using the following method:

- The issue was researched on the DIGGS website. In particular, any discussion on the DIGGS forums was assessed.
- Where applicable, the issue was assessed in the following XML IDEs:
    - Altova XMLSpy Enterprise Edition version 2009 sp1;
    - <oXygen/> XML Editor 10.2, build 2009051915; and,
    - Stylus Studio 2009 XML Enterprise Suite Release 2 Framework version: Build 1386e.
- The issue was summarized.
- One or more solutions were proposed, with examples, and pros and cons for each solution.
- The issue and proposed solutions were rated based on the issue assessment metric discussed in Section 2.1.

Additionally, these guidelines were followed when evaluating the issues and suggestions:

- The significance of the issue and proposed solutions were evaluated primarily from the perspective of users of the schemas; i.e., producers and consumers of XML instance documents defined by the schema.
- The focus was not to evaluate the schema's effectiveness in representing information in the DIGGS Geotechnical domain, but rather the generic schema issues and best practices.
- Issues and solutions were assessed based on what is considered right, as opposed to what may be easier to adopt, based on previous DIGGS versions and existing software implementations.
- A pragmatic approach was taken to issue resolution and suggestions. For example, if certain XML schema features were not well supported across the XML IDEs and tools, we simply suggested these features not be used.
- Generally accepted best practices were suggested where applicable.

 Copyright © 2009 Compusult Limited 

## 2.3 Recursion of DIGGS Objects

*Recursion of DIGGS objects results in XML parsing errors.*

### 2.3.1 Summary

This issue, as described on the DIGGS forums, is due to validation errors that occur when an OASIS XML Catalog file is not configured correctly for the DIGGS schemas.

The DIGGS Release Notes (DIGGS Release Notes.pdf - distributed with the schemas) and the DIGGS forums (http://www.diggsml.com/using-diggsml-catalog-file-cache-your-schemas-locally) describe how to configure an OASIS XML Catalog file for DIGGS.

OASIS XML Catalog files allow XML Schema tools to configure the location of externally referenced schema files.

The XML Schema external referencing mechanism is provided by the <include> and <import> elements. These elements allow an XML Schema to be designed in a modular fashion and for schemas to re-use objects defined in other schemas.

The <import> and <include> elements indicate the location of a referenced file by the schemaLocation attribute. The value of this attribute is of type anyURI, and, for example, can be an absolute HTTP URL, an absolute file location, or a relative location. XML Schema processors will retrieve the files referenced by the schemaLocation attributes when parsing a schema. Note that this implies that when an HTTP URL is used as a schemaLocation value, the file will be retrieved over the network.

```
<import namespace="http://www.opengis.net/gml"
  schemaLocation="http://schemas.diggsml.com/schemas/1.0a/gml/3.1.1/base/gml.xsd"/>
```

**Example 2-1: <import> element with absolute HTTP URL schemaLocation**

Example 2-1 uses the <import> element to reference a schema file located at the URL http://schemas.diggsml.com/schemas/1.0a/gml/3.1.1/base/gml.xsd.

```
<import namespace="http://www.opengis.net/gml"
    schemaLocation="../../gml/3.1.1/base/gml.xsd"/>
```

**Example 2-2: <import> element with relative schemaLocation**

Example 2-2 uses the <import> element to reference a schema file located at a relative location. The physical location is determined relative to the location of the file containing the <import> element.

 Copyright © 2009 Compusult Limited 

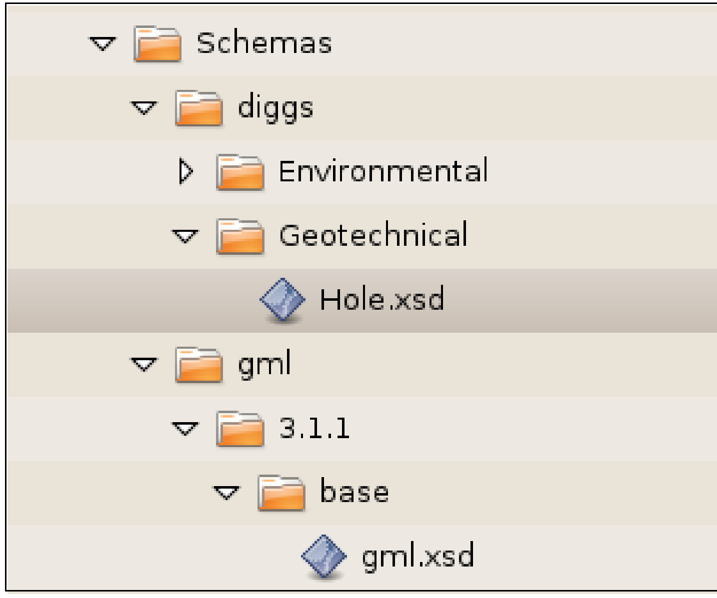


**Figure 2-1: Hole.xsd relative to gml.xsd**

Figure 2-1 shows the file **Hole.xsd** located at a file system location, and the file **gml.xsd** located at the relative location specified by the schema fragment in Example 2-2.

The DIGGS schemas always use absolute HTTP URL locations for the value of the schemaLocation attribute for the <import> and <include> elements. The DIGGS Catalog file is intended to direct XML Schema applications to override the HTTP URL specified in the schema files and use a local file system location instead.

The Catalog snippet in Example 2-3 below directs XML Schema applications to replace any schemaLocation values that start with http://schemas.diggsml.com/schemas/1.0a/ with file:///E:/Projects/DIGGS/1.0a/source/Schemas/.

```
<rewriteURI uriStartString="http://schemas.diggsml.com/schemas/1.0a/"
     rewritePrefix="file:///E:/Projects/DIGGS/1.0a/source/Schemas/"/>

   <rewriteSystem rewritePrefix="file:///E:/Projects/DIGGS/1.0a/source/Schemas/"
     systemIdStartString="http://schemas.diggsml.com/schemas/1.0a/" />
```

**Example 2-3: OASIS XML Catalog configuration for DIGGS**

By using absolute HTTP URL locations in this way, and if a Catalog file is not configured correctly, XML Schema parsers may indicate the schemas are not valid. For example, if one installs the DIGGS schema files on a local file system and opens any DIGGS schema file that defines a global schema object, schema parsers will indicate the file is not valid. Figure 2-2 illustrates this for the Hole.xsd file.

 Copyright © 2009 Compusult Limited 

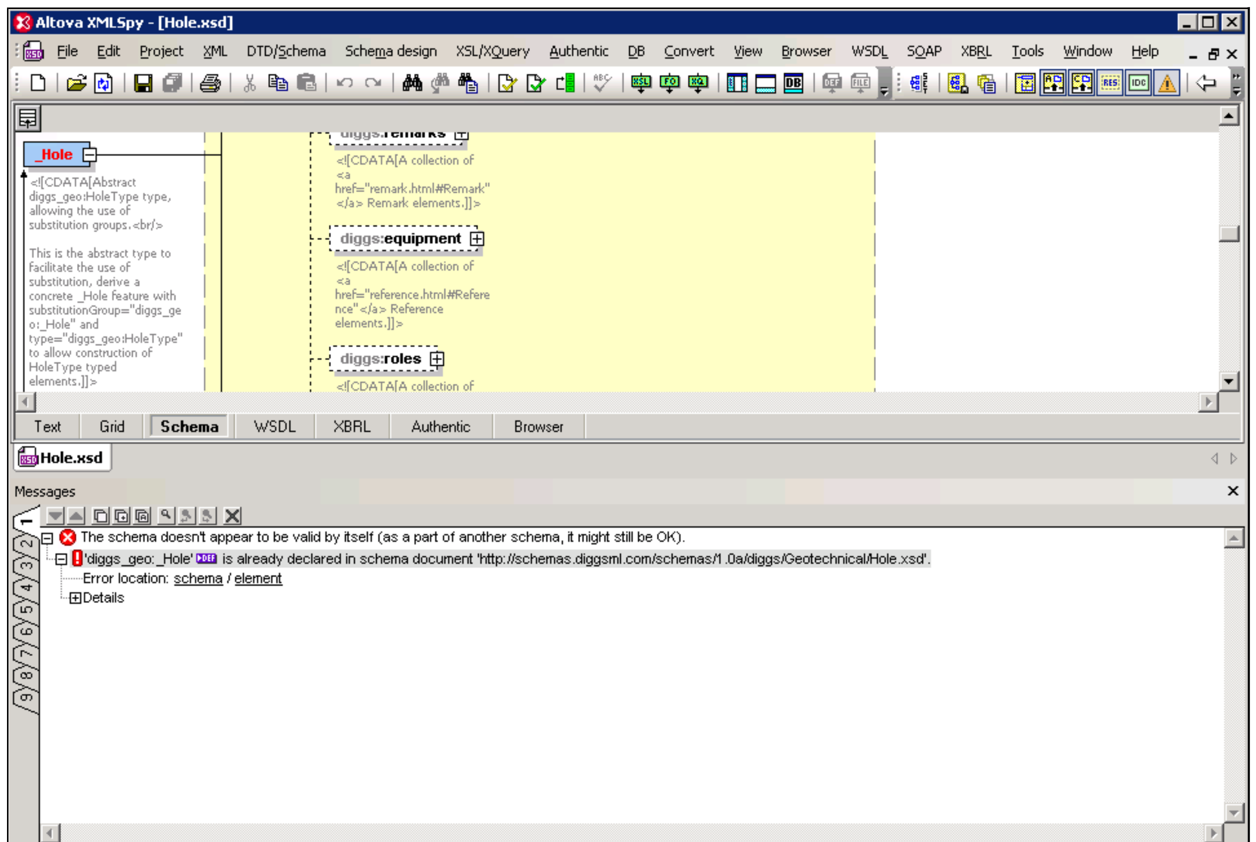


**Figure 2-2: Hole.xsd Validation Failure**

The Hole.xsd document includes the file located at the internet location http://schemas.diggsml.com/schemas/1.0a/diggs/Geotechnical.xsd (see Example 2-4).

```
<include schemaLocation=
  "http://schemas.diggsml.com/schemas/1.0a/diggs/Geotechnical.xsd" />
```

**Example 2-4: Internet Reference to Geotechnical.xsd**

The file Geotechnical.xsd includes the file Hole.xsd located at the internet location http://schemas.diggsml.com/schemas/1.0a/diggs/Geotechnical/Hole.xsd (see Example 2-5).

```
<include schemaLocation=
  "http://schemas.diggsml.com/schemas/1.0a/diggs/Geotechnical/Hole.xsd" />
```

**Example 2-5: Internet Reference to Hole.xsd**

 Copyright © 2009 Compusult Limited 

Schema validators will consider the objects defined in the locally-opened Hole.xsd file and the Hole.xsd file located at the Internet location (included via Geotechnical.xsd) as different objects with the same names, as shown in Figure 2-2.

The Catalog file resolves this issue by re-mapping the Internet location to the exact file system location where the files are installed locally.

The use of OASIS XML Catalog files is not required by XML Schema. As shown in Figure 2-2, the <import> and <include> elements allow relative schemaLocation values, so the DIGGS schema files can be written with relative schemaLocation values, eliminating the need to use XML Catalogs.

One reason for DIGGS embracing the OASIS XML Catalog standard is documented in the e-mail thread http://osdir.com/ml/text.xml.xerces-j.user/2007/msg00507.html.

When this thread was started, the DIGGS Schema files were configured such that the <import> and <include> elements used relative schemaLocation values and did not need to use a Catalog file.

This thread describes an issue validating a DIGGS XML instance document. Note the distinction between an XML instance document and an XML Schema document. An XML Schema document (or set of documents) describes an XML instance document, and an XML instance document is validated against an XML Schema.

The issue described in this thread is related to how the Apache Xerces-J XML implementation can fail to validate DIGGS XML instance documents that use relative schemaLocation values to refer to the DIGGS XML Schema files. Relative schemaLocation values in instance documents are resolved differently in Xerces-J than in other XML applications, such as the XML IDEs targeted by DIGGS.

Example 2-6 shows an example of using an absolute schemaLocation value in a DIGGS XML instance document.

```
<Diggs xmlns="http://schemas.diggsml.com/1.0a"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:gml="http://www.opengis.net/gml"
       xmlns:xlink="http://www.w3.org/1999/xlink"
       xmlns:witsml="http://www.witsml.org/schemas/131"
       xmlns:diggs_geo="http://schemas.diggsml.com/1.0a/geotechnical"
       xmlns:diggs="http://schemas.diggsml.com/1.0a"
       xsi:schemaLocation="
 http://schemas.diggsml.com/1.0a
     http://schemas.diggsml.com/schemas/1.0a/diggs/kernel.xsd
 http://schemas.diggsml.com/1.0a/geotechnical
     http://schemas.diggsml.com/schemas/1.0a/diggs/geotechnical.xsd">
```

**Example 2-6: Absolute schemaLocation value in DIGGS example file**

 Copyright © 2009 Compusult Limited 

The thread describes the desire to distribute example DIGGS XML instance documents (see the 1.0a/examples/ folder in the distribution) that will validate in the targeted XML IDEs without modification. The examples are validated by Xerces-J during development. The problem is the need to produce examples that can be validated by Xerces-J during development and validate, unmodified, in XML IDEs when distributed. The resolution of this issue was the decision to use OASIS XML Catalog.

Finally, note that relationships in the Hole.xsd file define a circular dependency because Hole.xsd depends on Geotechnical.xsd, which depends on Hole.xsd. The W3C XML Schema specification does not forbid circular dependencies in includes and there is nothing fundamentally wrong with doing this. The important point is that when a set of schema files define a circular dependency, every schemaLocation value for any particular file must refer to exactly the same location.

### 2.3.2 Recommendation

As stated in the summary, the issue can be resolved by ensuring that an OASIS XML Catalog for DIGGS is properly configured in the schema parser.

However, we offer a low priority recommendation for DIGGS to not use OASIS XML Catalogs. The use of OASIS XML Catalogs adds an unnecessary external dependency and complication to the DIGGS schemas.

The issues discussed in the e-mail thread referenced above can be addressed and the OASIS XML Catalog requirement can be eliminated by:

- Updating the DIGGS schemas to use relative schemaLocation references (as in Figure 2-2).
- Updating the DIGGS examples to not specify a schemaLocation attribute (see Example 2-6 for an example of how the schemaLocation is used in the examples). The schemaLocation attribute should be removed from the sample documents.
- Including supporting documentation that instructs users how to use their XML IDE to validate the examples. For example, in XMLSpy, the user must open the example file, use the **DTD/Schema** menu option, and choose **Assign Schema...**
- Using an alternate method for validating the examples during development, such as the xjparse utility (http://nwalsh.com/java/xjparse/).

It should be noted that the third-party schemas used by (and distributed with) DIGGS, such as the GML schemas and the WITSML schemas, use relative schemaLocation values.

If the DIGGS community decides to continue requiring the OASIS XML Catalog, we recommend that the requirement to configure the Catalog be more prominent in the installation instructions.

 Copyright © 2009 Compusult Limited 

### 2.3.3 Rating

**Importance:** Low Importance. This issue can be resolved by ensuring the XML Schema validator being used is correctly configured with an appropriate OASIS XML Catalog file. We describe how the schemas can be written to not use OASIS XML Catalog files and suggest this will help simplify the schemas.

**Scope/Difficulty:** Low Impact. The issue as described is due to a user configuration error. However, the schemas can be changed to eliminate the source of the configuration error by using relative schemaLocation values (see Figure 2-2) in the DIGGS schemas.

**Time Estimate:** 1 day

## 2.4 Table Object Implementation / Table Object Data Validation

*Table objects are not consistently implemented throughout the schema.*

*Table object data are not validated.*

### 2.4.1 Summary

These issues are specified separately on the DIGGS forum. Both issues have been consolidated into one section here as the recommendation is intended to resolve both issues.

The DIGGS table object is an XML construct designed to hold a table of values. The values in the table are contained in a single XML element as a delimited string. The column definitions, row, column and decimal separators are defined with the data. Example 2-7 shows an example of a table object.

 Copyright © 2009 Compusult Limited 

```
 <diggs:Table>
    <diggs:columns xmlns="http://schemas.diggsml.com/1.0a">
      <diggs:Column index="1">
        <dataType>xs:double</dataType>
        <meaning>Index</meaning><uom>m</uom>
      </diggs:Column>
    <diggs:Column index="2">
      <dataType>xs:double</dataType>
      <meaning>Measure</meaning>
      <uom>MN/m2</uom>
      <source><Ref xlink:href="#DIGGSINC-CPT-CONE-1-RES"/></source>
    </diggs:Column>
...
    <diggs:Column index="5">
      <dataType>xs:double</dataType>
      <meaning>Measure</meaning><uom>kN/m2</uom>
      <source><Ref xlink:href="#DIGGSINC-CPT-CONE-1-PWP"/></source>
    </diggs:Column>
    </diggs:columns>
    <diggs:blockSeparator>;</diggs:blockSeparator>
    <diggs:decimalSeparator>.</diggs:decimalSeparator>
    <diggs:tokenSeparator>,</diggs:tokenSeparator>
    <diggs:data>
0.010,0.1300,0.40,0.0000,0.0013;
0.020,0.2400,0.40,0.1.0a,0.0078;
0.030,0.5500,0.40,0.0040,0.0126;
0.070,0.9600,0.40,0.0200,0.0191;
...
5.430,40.9100,0.20,9999.0000,9999.0000;
5.440,40.5400,0.40,9999.0000,9999.0000;
    </diggs:data>
  </diggs:Table>
```

**Example 2-7: Sample DIGGS 1.0a Table Object**

This example demonstrates a table with the following characteristics

- five columns (defined by the <diggs:column> elements)
- row delimiter ‘;’ (defined in the <diggs:blockSeparator> element)
- decimal separator ‘.’ (defined in the <diggs:decimalSeparator> element)
- column delimiter ‘,’ (defined in the <diggs:tokenSeparator> element)

The actual data values are within the <diggs:data> element. For readability, the example shows each row of data on a single line; however, this is unnecessary. The first five values (up to the ‘;’) correspond to the first row of values. The first value is assigned to the column with <diggs:column index=”1”>, the second value is assigned to the column<diggs:column index=”2”>.

 Copyright © 2009 Compusult Limited 

The main benefits of this format are compactness and flexibility. The format will not result in large file sizes (i.e., XML bloat) from repeated XML elements for large amounts of data. In contrast, Example 2-8 below presents a hypothetical table structure that does result in XML bloat:

```
 <Table>
    <Row>
      <Value>0.0</Value><Value>0.0</Value> ... <Value>0.0</Value>
    </Row>
    <Row>
      <Value>0.0</Value><Value>0.0</Value> ... <Value>0.0</Value>
    </Row>
...
    <Row>
      <Value>0.0</Value><Value>0.0</Value> ... <Value>0.0</Value>
    </Row>
  </Table>
```

**Example 2-8: Bloated Table Structure**

In Example 2-8, each table value is delimited by a <Value>,</Value> pair. This adds 15 characters of XML overhead per table value. For a table with 5 columns and 100,000 rows, this adds over 7 MB to the file size.

There are several drawbacks to the DIGGS 1.0a approach:

- The table contents cannot be validated by XML schema validation.
- Due to the generic nature of this object, it could be used to represent the same data in different ways. For example, since the column names are defined with the data, different implementers could choose different names for the same value.
- The structure also allows variations in the representation of numbers, as the decimal separator can be customized by the choice of delimiters. The generic nature of the object combined with the lack of XML Schema validation makes it more likely for interoperability issues to arise between implementers.
- The flexibility of the table object allows it to represent much of the information contained in a DIGGS document. This has led to inconsistent usage in that a table object has been used in some places and not in others. For example, as mentioned on the DIGGS forums (http://www.diggsml.com/forum/viewtopic.php?f=17&t=44#p111), the Dilatometer (DMT) object is defined explicitly, but the Static Cone Test (CPT) is represented as a table.

 Copyright © 2009 Compusult Limited 

### 2.4.2 Recommendation

There is no clear cut resolution that solves all validation and consistency issues, while providing the desired flexibility and minimal file size. However, we recommend that a data-specific object be defined for each case where tabular values will be encoded. Example 2-9 shows the recommended pattern for this object.

```
<complexType name="MyTableObjectType">
       <complexContent>
       ....
       <sequence>
           <element name="StandardColumnName1" type="diggs:DataArrayType"
              minOccurs="0"/>
           <element name="StandardColumnName2" type="diggs:DataArrayType"
              minOccurs="0"/>
           <element name="StandardColumnName3" type="diggs:DataArrayType"
              minOccurs="0"/>
           <element name="StandardColumnName4" type="diggs:DataArrayType"
              minOccurs="0"/>
           ....
           <element name="ExtensionValue" type="diggs:DataArrayType"
              minOccurs="0" maxOccurs="unbounded"/>
       </sequence>
   </complexType>
```

**Example 2-9: Recommended Table Object Pattern**

The general structure is to explicitly define each “standard” column expected in the table and, if necessary, allow a repeatable generic column in the structure. This handles the usual case of “standard” columns in a structured and well-defined way, but also allows inclusion of project-specific columns. The Static Cone Test object shown in Example 2-10 is a sample of this pattern.

```
<complexType name="StaticConeTestType">
    <complexContent>
    ....
    <sequence>
        <element name="Index" type="diggs:DataArrayType" minOccurs="0"/>
        <element name="Depth" type="diggs:DataArrayType" minOccurs="0"/>
        <element name="TipResistance" type="diggs:DataArrayType"
           minOccurs="0"/>
        <element name="SleeveFriction" type="diggs:DataArrayType"
           minOccurs="0"/>
        ....
        <element name="ExtensionValue" type="diggs:DataArrayType"
           minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

**Example 2-10: StaticConeTestType Definition**

 Copyright © 2009 Compusult Limited 

The “standard” columns are called out explicitly, while the <ExtensionValue> element allows additional columns to be specified. Each column is of type DataArrayType. This is an abstract object that is a placeholder for a datatype-specific array. The DataArrayType and a sample specialization are defined in Example 2-11.

```
<complexType name="DataArrayType" abstract="true">
       <complexContent>
           <extension base="....">
               <sequence>
                   <element name="uom" type="diggs:DiggsStringType"
                       minOccurs="0"/>
                           <element  name="source"  type="diggs:ReferencePropertyType"
                                       minOccurs="0"/>
               </sequence>
           </extension>
       </complexContent>
   </complexType>

   <complexType name="DoubleDataArrayType">
       <complexContent>
           <extension base="diggs:DataArrayType">
               <sequence>
                   <element name="values">
                       <simpleType>
                           <restriction base="string">
                               <pattern value="(([-+]?[0-9]+[.][0-9]+[ ]) |
                               (null[ ]))*(([-+]?[0-9]+[.][0-9]+)|(null))+"/>
                           </restriction>
                       </simpleType>
                   </element>
               </sequence>
           </extension>
       </complexContent>
   </complexType>
```

**Example 2-11: DataArrayType Definition**

The row values for each column can be validated by a regular expression, as shown in Example 2-11. In this case, the DoubleDataArrayType specialization contains a regular expression that forces the column values to be either a numeric double or a “null” string. Example 2-12 below shows a sample XML instance of this StaticConeTest object.

```
<StaticConeTest xmlns="http://schemas.diggsml.com/1.0a"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <id>uuid-7ce3023c-870c-404d-a466-db021ba65880</id>
   <Index xsi:type="DoubleDataArrayType">
     <id>uuid-1607ff0a-5b7d-47a4-869d-c299411fa589</id>
     <uom>m</uom>
     <values>0.010 0.020 0.030 0.040</values>
   </Index>
   <Depth xsi:type="DoubleDataArrayType">
     <id>uuid-dd2981a6-c03b-47a6-b00a-f7a8db77e6bd</id>
     <uom>MN/m2</uom>
     <values>0.1300 null 0.5500 0.6800</values>
   </Depth>
   <TipResistance xsi:type="DoubleDataArrayType">
     <id>uuid-ef9c60aa-389d-4844-9252-dc90a3c578bc</id>
     <uom>uS/cm</uom>
     <values>0.40 -0.40 0.40 null</values>
   </TipResistance>
   <SleeveFriction xsi:type="DoubleDataArrayType">
     <id>uuid-9046fa5c-1f50-48ef-93da-661e4f5b5098</id>
     <uom>kN/m2</uom>
     <values>0.0000 0.0040 0.0070 0.0120</values>
   </SleeveFriction>
   <ExtensionValue xsi:type="DoubleDataArrayType">
     <id>uuid-0b7366f2-32a9-40f7-b5c8-144cfb2c76ea</id>
     <uom>kN/m2</uom>
     <source>
       <Ref xlink:href="#DIGGSINC-CPT-CONE-1-PWP"/>
     </source>
     <values>0.0013 0.0078 0.0126 -0.0017</values>
   </ExtensionValue>
 </StaticConeTest>
```

**Example 2-12: Sample StaticConeTest**

This recommended approach has several potential drawbacks:

- It does not support free text data cells without using escape sequences for the delimiters. Note that this is not an issue if the datatype for the column is strictly numeric.
- It does not support having a customizable decimal separator. The example requires the use of a decimal point and does not allow using a comma. This is good because it forces lexical representation of a decimal, as defined by the XML Schema specification, and does not burden consumers with having to implement special parsing for numerics.
- XML Schema validation cannot force each column to have the same number of rows, as would be expected in a table. This is the trade-off that allows the table data to be represented in a compact format.

 Copyright © 2009 Compusult Limited 

The object presented above provides a tabular structure that can be almost completely validated without requiring excessive file sizes.

The resolution to the issue of when to use a table object as opposed to a non-tabular representation is highly dependent on the expected amount of data, as well as the flexibility required for a particular construct. This should be achieved by analysis and consensus amongst the DIGGS schema designers and the rationale for the choices should be clearly documented.

#### 2.4.3 Rating

**Importance:** Medium Importance. The recommended updates would make the content of table objects more structured and yield easier interoperability amongst implementers. The rationalization and documentation of when table objects are chosen would improve overall schema consistency and simplify implementation.

**Scope/Difficulty:** Medium Impact. The process of determining when to use the table object structures and the definition of the "standard" columns for any particular table object requires consensus amongst DIGGS stakeholders. Updating the schemas themselves is simply a matter of following the suggested pattern. The amount of schema files affected will depend on where table objects are deemed appropriate.

**Time Estimate:** 2+ days

### 2.5 Code Table Implementation

*A consistent approach to code tables has not been implemented throughout the schema. Codes are not validated against a dictionary.*

#### 2.5.1 Summary

A code table is a named list of values. Given the name of a list and a value from that list, it is expected that *meaning* can be inferred from the value.

For example, the Julian Calendar Months can be seen as a code table. Given the value 4 from this code table, one can conclude that this value is meant to refer to the fourth month. From this we can derive a language specific name for the value (April, in the English language), and that this month has 30 days. Other examples of well known code tables are "States of the United States of America" and "Counties of England".

 Copyright © 2009 Compusult Limited 

An example of a DIGGS code table is agsCodeLists-V1 (see the file 1.0a/source/CodeLists/agsCodeList_V1.xml in the DIGGS schema distribution). Given value ConeResistance from this code table, one can conclude it refers to an AGS 3.1 CPT Cone Parameter.

DIGGS code tables cannot be validated by XML Schema validation. This allows implementers to use code lists inconsistently, is error prone, and increases the likelihood of interoperability issues. Code tables are represented in DIGGS using a combination of the GML CodeType and Dictionary constructs, shown respectively in Example 2-13 and Example 2-14.

```
<complexType name="CodeType">
   <simpleContent>
     <extension base="string">
       <attribute name="codeSpace" type="anyURI" use="optional"/>
     </extension>
   </simpleContent>
 </complexType>
```

**Example 2-13: GML CodeType**

```
<element name="Definition" type="gml:DefinitionType"
   substitutionGroup="gml:_GML"/>

 <complexType name="DefinitionType">
   <complexContent>
     <restriction base="gml:AbstractGMLType">
       <sequence>
         <element ref="gml:metaDataProperty" minOccurs="0" maxOccurs="unbounded"/>
         <element ref="gml:description" minOccurs="0"/>
         <element ref="gml:name" minOccurs="1" maxOccurs="unbounded"/>
       </sequence>
       <attribute ref="gml:id" use="required"/>
     </restriction>
   </complexContent>
 </complexType>

 <element name="Dictionary" type="gml:DictionaryType"
   substitutionGroup="gml:Definition"/>

 <complexType name="DictionaryType">
   <complexContent>
     <extension base="gml:DefinitionType">
       <choice minOccurs="0" maxOccurs="unbounded">
         <element ref="gml:dictionaryEntry"/>
         <element ref="gml:indirectEntry"/>
       </choice>
     </extension>
   </complexContent>
 </complexType>
```

**Example 2-14: GML Dictionary**

 Copyright © 2009 Compusult Limited 

An XML Schema element of type gml:CodeType specifies a value and a codeSpace, where the value is asserted to have meaning in the given codeSpace. For example, the DIGGS Detector object has a measurand element of type gml:CodeType:

```
<complexType name="DetectorType" mixed="false">
   <complexContent mixed="false">
     <extension base="diggs:IdentifiedFeatureType">
       <sequence>
         <element name="hasCRS" type="diggs:AnyCRSPropertyType" minOccurs="0"/>
         <element name="measurand" type="gml:CodeType" minOccurs="0"/>
         <element name="position" type="gml:PointType" minOccurs="0"/>
         <element name="type" type="gml:CodeType" minOccurs="0"/>
         <element name="measurementGroups"
           type="diggs_mon:GenericMeasurementGroupPropertyType" minOccurs="0"/>
       </sequence>
     </extension>
   </complexContent>
 </complexType>
```

**Example 2-15: Detector XML Schema Definition**

The DIGGS CPTCone object (indirectly) contains a list of Detector objects:

```
<complexType name="CPTConeType" mixed="false">
   <complexContent mixed="false">
     <extension base="diggs:EquipmentType">
       <sequence>
         <element name="detectors" type="diggs_mon:DetectorPropertyType"
           minOccurs="0">
         </element>
       </sequence>
     </extension>
   </complexContent>
 </complexType>
```

**Example 2-16: CPTCone XML Schema Definition**

Putting all this together, a sample instance of a DIGGS CPTCone object may be represented as shown in Example 2-17 below.

 Copyright © 2009 Compusult Limited 

```
<dg:CPTCone xmlns="http://schemas.diggsml.com/1.0a/geotechnical">
   <d:id>DIGGSINC-CPT-CONE-1</d:id>
   <detectors>
     <dm:Detector>
       <d:id>DIGGSINC-CPT-CONE-1-RES</d:id>
       <dm:measurand codeSpace="agsCodeList_V1.xml">ConeResistance</dm:measurand>
     </dm:Detector>
     <dm:Detector>
       <d:id>DIGGSINC-CPT-CONE-1-COND</d:id>
       <dm:measurand codeSpace="agsCodeList_V1.xml">Conductivity</dm:measurand>
     </dm:Detector>
     <dm:Detector>
       <d:id>DIGGSINC-CPT-CONE-1-LSFR</d:id>
       <dm:measurand
        codeSpace="agsCodeList_V1.xml">LocalSideFrictionResistance</dm:measurand>
     </dm:Detector>
     <dm:Detector>
       <d:id>DIGGSINC-CPT-CONE-1-PWP</d:id>
       <dm:measurand codeSpace="agsCodeList_V1.xml">PorewaterPressure</dm:measurand>
     </dm:Detector>
   </detectors>
 </dg:CPTCone>
```

**Example 2-17: CPTCone XML Instance**

One can conclude that this CPTCone object is measuring “Cone Resistance”, “Conductivity”, “Local Side Friction Resistance” and “Porewater Pressure”.

Using this approach, the XML Schema cannot force that measurand values in the CPTCone object are valid values for a CPT Co. Also, the XML Schema cannot force that an appropriate codeSpace value is used and cannot validate that the value specified actually exists in the given codeSpace. This allows several classes of issues to occur, including:

- Accidental Errors. Typos in the measurand value or codeSpace name.
- Incorrect usage of a code table and value. For example, a code table and value may be misinterpreted and used somewhere that does not make sense.
- Interoperability Problems. An implementers may define their own version of agsCodeList_V1.xml, and choose their own names for the CPT Cone Parameters.

These issues significantly reduce the power of a general purpose DIGGS software application. For a software application to assign meaning to a value or to take action based on a given value, it must be coded to know about that value. To be recognized, the value must match exactly what the software application is expecting. There are ways to implement software to dynamically look up values and take corresponding actions; however, this places a considerable burden on DIGGS software implementers.

 Copyright © 2009 Compusult Limited 

The current DIGGS code table approach provides extremely important and useful flexibility. Since the code table values are not “hard-wired” into the DIGGS Schemas, project-specific code tables can be used without having to change the DIGGS Schemas. This is an important feature, as it allows cooperating project-specific software applications to use the DIGGS schemas to exchange information using project-specific code tables.

### 2.5.2 Recommendation

We recommend that an extendable XML Schema enumerations pattern be used for code tables. This approach defines the “standard” values for a code table in the schema and provides an optional extension mechanism. This explicitly defines the normal case in the schemas, making it validatable and easy to use.

The recommended pattern for extendable XML Schema enumeration can be defined as a regular XML Schema enumeration combined with an optional gml:codeType that can be used to specify custom entries. The custom entries are expressed using gml:codeType as it is currently used in DIGGS 1.0a. Example 2-18 shows an example of this approach for CPT Cone Detectors.

```
<complexType name="CPTConeDetectorType">
   <sequence>
     <element name="value">
       <simpleType>
         <restriction base="string">
           <enumeration value="Temperature"/>
           <enumeration value="PhotoIonization"/>
           <enumeration value="PH"/>
           <enumeration value="Slope"/>
           <enumeration value="PhotoMultiplier"/>
           <enumeration value="ConeResistance"/>
           <enumeration value="Conductivity"/>
           <enumeration value="FluorescenceIntensity"/>
           <enumeration value="FlameIonization"/>
           <enumeration value="RedoxPotential"/>
           <enumeration value="LocalSideFrictionResistance"/>
           <enumeration value="CvDissipation"/>
           <enumeration value="DryElectricConductivity"/>
           <enumeration value="NaturalGammaCount"/>
           <enumeration value="PorewaterPressure"/>
           <enumeration value="EXTENSION"/>
         </restriction>
       </simpleType>
     </element>
     <element name="extensionValue" type="gml:CodeType" minOccurs="0"/>
   </sequence>
 </complexType>
```

**Example 2-18: CPTCone Detectors Enumeration with Optional Extension**

 Copyright © 2009 Compusult Limited 

The element <value> is an enumeration that contains the "standard" values for the code table with the addition of an EXTENSION entry. When an extension is desired, the EXTENSION value is specified for the <value> element and the <extensionValue> is populated with the custom value.

```
<CPTConeDetector>
   <value>EXTENSION</value>
   <extensionValue codeSpace="MyCPTDectectors.xml">MyDetector</extensionValue>
 </CPTConeDetector>
```

**Example 2-19: Example CPTCone Detector XML Instance with Extension**

### 2.5.3 Rating

**Importance:** High Importance. The adoption of XML Schema enumerations is an important issue. It greatly enhances the ability of software implementers to produce more powerful applications. It makes the schemas easier to apply where standard values are used and greatly reduces interoperability issues due to errors and misinterpretations.

**Scope/Difficulty:** Medium Impact. The process of creating the extensible enumerations according to the pattern specified above is a matter of translating the code tables already used by DIGGS. Updating the schemas such that the lists are specified appropriately requires going through the DIGGS schemas and assigning the appropriate type. This requires knowledge of the DIGGS schemas and the DIGGS domain.

**Time Estimate:** 3 Days

## 2.6 Key Fields / Use of Unique Identifiers

*Key field requirements had been removed from DIGGS objects, resulting in object referencing via a combination of hierarchy and unique identifiers. Absence of key fields is causing database implementation issues.*

*The non-hierarchical XML implementation of DIGGS requires the use of unique identifiers to establish relationships between some objects, impacting database mapping and deployment.*

### 2.6.1 Summary

Entities described by the DIGGS schemas need to be uniquely identified. Historically (prior to an XML representation of DIGGS documents), a combination of key fields have been used to

 Copyright © 2009 Compusult Limited 

uniquely identify the entities described by DIGGS. A key field is an attribute of the entity being described, where the value is unlikely to change. A combination of one or more key fields can be used to uniquely identify an entity. Relationships between DIGGS objects can be established by assigning the key field combination of one object to another related object.With the advent of an XML representation of DIGGS documents, a more XML-centric approach for identifying and relating DIGGS objects has been defined. An XML document is a hierarchical, tree like, structure with exactly one root element. Every other element in the XML document can be related to the root element by following the parent-child relationship between nodes. The XML format provides implied relationship between an element and its descendents.

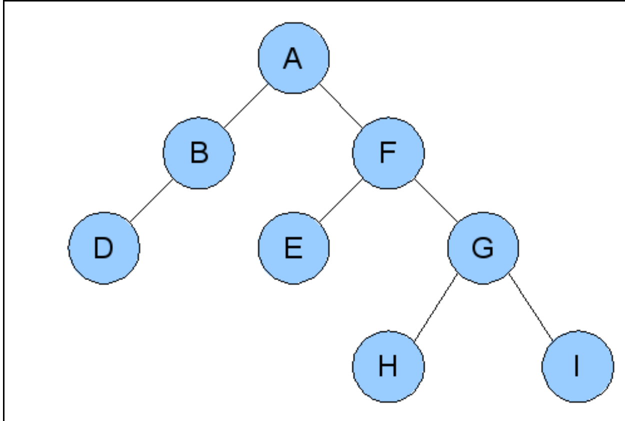


**Figure 2-3: Tree Structure**

As shown in Example 2-3, each node in the diagram is related to the root node labeled A. Also note that the nodes E, G, H, and I are related to F, while D is not related to F.

DIGGS uses the XMLSchema ID type to help define relationships beyond the implied hierarchical ones in the structure of XML. To define this relationship, an object that is to be referred to outside the XML hierarchy must be assigned an ID value.

Example 2-20 shows how this is used in DIGGS. A Detector object is identified by the value DIGGSINC-CPT-CONE-1-RES and this detector is referenced from a Column object via <Ref xlink:href="#DIGGSINC-CPT-CONE-1-RES" />.

 Copyright © 2009 Compusult Limited 

```
<Diggs>
  <equipments>
    <diggs_geo:CPTCone xmlns="http://schemas.diggsml.com/1.0a/geotechnical">
      <diggs:id>DIGGSINC-CPT-CONE-1</diggs:id>
      <detectors>
        <diggs_mon:Detector>
          <diggs:id>DIGGSINC-CPT-CONE-1-RES</diggs:id>
          <diggs_mon:measurand codeSpace="agsCodeList_V1.xml">
            ConeResistance
          </diggs_mon:measurand>
        </diggs_mon:Detector>
        ...
      </detectors>
    </diggs_geo:CPTCone>
  </equipments>
  <projects>
    <Project>
      ...
      <locations>
        <Hole>
      ...
          <diggs:insituTesting>
            <StaticConeTest>
              <diggs_geo:tabularData>
                <diggs:Table>
                  <diggs:columns xmlns="http://schemas.diggsml.com/1.0a">
                    ...
                    <diggs:Column index="2">
                      <dataType>xs:double</dataType>
                      <meaning>Measure</meaning>
                      <uom>MN/m2</uom>
                      <source>
                        <Ref xlink:href="#DIGGSINC-CPT-CONE-1-RES" />
                      </source>
                    </diggs:Column>
                    ...
                  </diggs:columns>
              ...</diggs:Table>
              </diggs_geo:tabularData>
            </StaticConeTest>
          </diggs:insituTesting>
          ...
        </Hole>
      </locations>
    </Project>
  </projects>
</Diggs>
```

**Example 2-20: Sample DIGGS Relationship**

The ID values used by DIGGS are referred to as codespace identifiers. A codespace identifier is a combination of company ID, the codespace, and a unique value within the codespace. This

 Copyright © 2009 Compusult Limited 

combination is guaranteed to be unique across all of DIGGS. The current approach requires any company or organization that intends to develop DIGGS documents to register with DIGGS and receive a company ID. The company is expected to maintain their own internal list of unique IDs. Any ID used in a DIGGS document is required to be of the form {Company ID}-{Internal ID}. This ID is guaranteed to be unique across all DIGGS implementers.

Using key fields to identify objects in DIGGS has the following disadvantages:

- in practice, the value of a key field may change, thus invalidating the purpose of the unique identifiers; and,
- this approach is difficult to maintain when many fields are needed to generate a unique value. For example, the meeting notes from the March 2009 DIGGS meeting in Orlando indicate that some objects require as many as eight fields to generate a unique value.

The concept of key fields is not used in the DIGGS 1.0a schemas. Unique identifiers are expected to be used to identify DIGGS objects in DIGGS XML documents.

The following issues have been raised:

- Key fields continue to be used by DIGGS users and thus must be enforced in the XML schema encoding of DIGGS.
- The use of unique identifiers places an unnecessary burden on implementers, as this is an extra attribute that must be stored and maintained with a DIGGS object.
- The DIGGS 1.0a schemas allow IDs to be specified on too many objects that do not need to be uniquely identified.

### 2.6.2 Recommendation

DIGGS' usage of IDs to establish relationships outside the XML hierarchy and to establish relationships to external documents and resources is reasonable. The requirement that they be persistently maintained with the other traditional attributes is not considered to be an excessive burden to DIGGS implementers.

IDs should only be used where they are needed; that is, in cases where relationships need to be established outside the XML hierarchy, either within the same XML document or in another document referenced via the xlink mechanism.

Key fields can be established by the schema through the use of the XML Schema <key> element.

 Copyright © 2009 Compusult Limited 

```
<xs:element name="MyDiggsObject">
   <xs:complexType>
     <xs:sequence>
       <xs:element name="fieldA" type="xs:string"/>
       <xs:element name="fieldB" type="xs:string"/>
       <xs:element name="fieldC" type="xs:string"/>
       <xs:element name="fieldD" type="xs:ID"/>
     </xs:sequence>
   </xs:complexType>
 </xs:element>
 <xs:element name="MyDiggsObjects">
   <xs:complexType>
     <xs:sequence>
       <xs:element ref="MyDiggsObject" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute name="id" type="xs:ID" use="required"/>
   </xs:complexType>
   <xs:key name="DiggsObjectKeyFields">
     <xs:selector xpath=".//DiggsObject"/>
     <xs:field xpath="fieldA"/>
     <xs:field xpath="fieldB"/>
     <xs:field xpath="fieldC"/>
   </xs:key>
 </xs:element>
```

**Example 2-21: XML Schema <key> element**

The example shown in Example 2-21 uses the XML Schema <key> element to enforce that the combination of fieldA, fieldB and fieldC are unique within the document.

The XML Schema ID type and <key> elements can be used to enforce uniqueness within an XML document, however, they cannot enforce global uniqueness. For example, depending on the key fields used to uniquely identify an object, it is possible for key field combinations to collide.

The codespace identifier approach currently used by DIGGS is viable for generating unique values within DIGGS. However, we offer a low priority recommendation that Universally Unique Identifiers (UUIDs) be used instead of codespace identifiers.

UUIDs are often used for generating unique identifiers in computer systems. The scheme provides a reasonable level of confidence that independently-generated UUIDs will not be duplicated.

The approach has the following advantages over codespace identifiers:

- UUIDs are unique, not just to DIGGS, but also globally, and thus could be used elsewhere without values colliding.
- UUIDs are easily generated by software.

 Copyright © 2009 Compusult Limited 

- It requires no central management. DIGGS producers can independently generate UUIDs without any need for coordination.
- DIGGS implementers do not have to maintain a special DIGGS ID to be used with their DIGGS company ID.

### 2.6.3 Rating

**Importance:** Medium Importance. The overuse of ID values within DIGGS 1.0a make the schemas more difficult to generate and maintain. The schemas will be easier to work with if they contain only the necessary IDs. Enforcement of Key Fields by the schemas is important, as this is the established way objects within the DIGGS domain are identified. It is important for the schemas to recognize and enforce this usage.

**Scope/Difficulty:** Medium Impact / Difficulty. Identifying the DIGGS objects that require IDs will require knowledge of the DIGGS domain. Updating the DIGGS schemas to use the <key> element to enforce Key Fields requires a moderate knowledge of XML Schema, and a knowledge of the DIGGS domain.

**Time Estimate:** 3+ days

## 2.7 Creating Empty XML Files

*An empty XML file cannot be generated from the schema.*

### 2.7.1 Summary

The assertion was made at the March 2009 DIGGS meeting in Orlando that empty XML files cannot be generated from the schema. In particular, it is stated that, using XMLSpy, a corrupt file was produced that was missing many objects (include the Hole object).

However, during our analysis of the schemas, the XML IDEs were able to create empty XML documents (with only mandatory elements or with all elements) from various DIGGS files, such as the high-level Diggs element, as well as objects deep in the DIGGS object hierarchy, such as the Hole object.

Note that XMLSpy creates the placeholder element <auto-generated_for_wildcard/> for the XML Schema <any> element, which is the XML Schema wildcard element. XML Schema allows any element to be used where the <any> element is used. The <any> element is used by a GML object employed by DIGGS.

 Copyright © 2009 Compusult Limited 

### 2.7.2 Recommendation

The targeted XML IDEs all try to generate schema-valid sample files. A schema-valid file, by definition, cannot contain every possible combination of mutually exclusive choices provided by an XML Schema. Therefore, the resulting XML file may or may not contain large portions of a schema, simply based on the combination of selected choices.

Note that a schema-valid file may not be valid in terms of the DIGGS business rules. This will happen when the rules comprising a complete DIGGS Project file cannot be expressed and enforced by XML Schema. For example, XML Schema cannot specify rules such as “a beginning date must come before an end date”. XML Schema language is capable of expressing certain rules regarding the construction of an XML document, such as whether an element or attribute is mandatory or the data type of an element or attribute. If the requirement is for a schema-valid DIGGS file to also be a business-rule-complete DIGGS file, then all business rules defining a complete DIGGS file must be expressible in XML Schema. A very in-depth knowledge of the capabilities of both XML Schema and the DIGGS domain would be required to implement this or to even determine if this is possible. However if this is a requirement, it will likely involve reducing the complexity and variation of information that can be expressed by a DIGGS document.

It is also worth noting that the DIGGS 1.0a schemas provide a large amount of variation in the resulting DIGGS document due to the amount of choice provided by the use of inheritance. Any schema-valid XML document, by definition, cannot contain all available mutually-exclusive choices. One can easily see that it would not be practical to generate a sample for every possible combination of choices or to prompt a user at every possible choice point.

### 2.7.3 Rating

**Importance:** Low Importance. This issue worked as expected in the targeted XML IDEs. The ability to generate empty XML files from an XML Schema is not a good metric for assessing the quality or usefulness of an XML Schema.

**Scope/Difficulty:** Large Impact. Designing an XML schema that enforces the business logic of the DIGGS application domain and can also generate a valid or partially-valid XML instance document is a difficult task requiring significant knowledge of both XML Schema and the DIGGS application domain.

**Time Estimate:** 5+ days

## 2.8 GML Implementation

*Full vs. partial GML implementation - current implementation adds too much complexity.*

### 2.8.1 Summary

The DIGGS Online Forums suggest that implementation of DIGGS as a GML feature application schema adds significant complexity to DIGGS and that there may be simpler ways to use GML, such as using a GML profile.

Prevalent use of inheritance from GML objects is a source of the DIGGS schemas' complexity. For example, all objects in the DIGGS Geotechnical namespace inherit (indirectly) from the gml:AbstractGMLType object. Analysis of examples distributed with the DIGGS schemas shows that only the following three fields inherited from the gml:AbstractGMLType object are used: gml:id attribute, gml:name element, and gml:description element. Figure 2-4 illustrates the fields used from the gml:AbstractGMLType object.

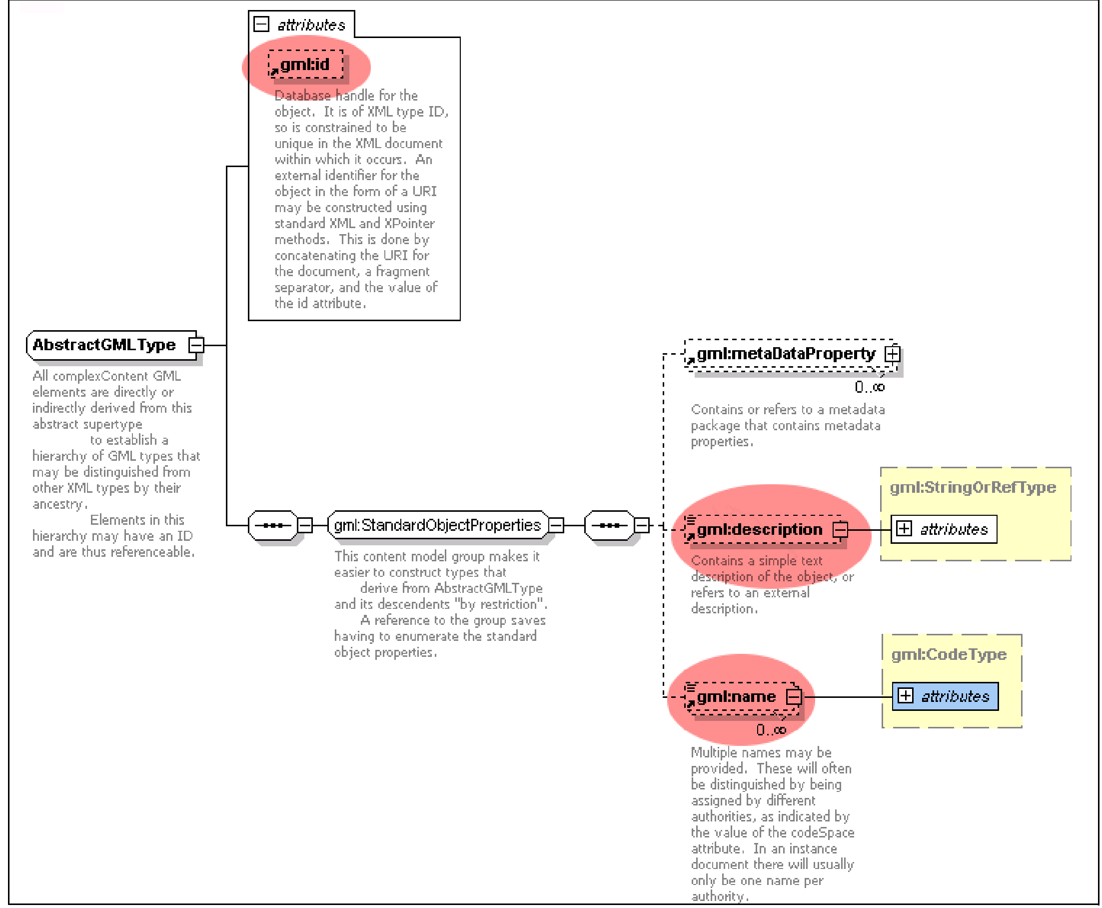


**Figure 2-4: GML AbstractGMLType**

 Copyright © 2009 Compusult Limited 

Another primary source of complexity is the unneccessary inheritance of GML objects in an attempt to follow the GML property pattern. However, the inheritance used in the DIGGS property pattern violates the GML specification.

See Section 4.0 for further discussion of GML usage in DIGGS.

### 2.8.2 Recommendation

We recommend simplifying the DIGGS schemas by reducing the amount of inheritance from GML objects and refactoring the DIGGS schemas to remove inheritance from AbstractGMLType. The DIGGS schemas primarily inherit from GML through:

- DIGGS object hierarchy via the diggs:FeatureType object and the diggs:DiggsBaseType object; and,
- the property pattern used throughout the schemas.

We also recommend that the following steps be taken:

1. Update diggs:FeatureType and the diggs:DiggsBaseType so that these objects no longer extend gml:AbstractFeatureType.
2. Copy the following elements from gml:AbstractGMLType to the diggs:FeatureType and diggs:DiggsBase objects:
    - gml:id attribute
    - gml:name element
    - gml:description element
3. Update the Property pattern used throughout the schemas to not inherit from any GML objects. See Section 4.0 for a more detailed discussion of the DIGGS and the GML property pattern.

The result of first step is that DIGGS objects will no longer be GML Features and DIGGS will no longer be a GML feature application schema. This will have the potentially negative result that generic GML applications may not be able to recognize DIGGS objects. GML is Feature-centric and generic GML applications are likely implemented to process objects that derive from gml:AbstractFeatureType.

The changes specified above do not alter other uses of GML in DIGGS. See Section 4.0 for a discussion of how DIGGS uses GML.

As discussed in Section 4.0, DIGGS uses only a subset of GML. It is possible to distribute the DIGGS schemas with this subset. A GML subset is known as a GML profile. The complete rules that define a GML profile are contained in the GML Specification. Creating a GML profile

 Copyright © 2009 Compusult Limited 

involves editing the GML schemas to include only the desired objects and dependencies. This is a difficult task, so the specification provides (non-normative) scripts to aid this process.

Several GML profiles have been officially defined, such as the GML Point Profile (http://www.ogcnetwork.net/gml-point), a GML subset that restricts geometries to Points expressed in the well-known EPSG:4326 coordinate system. Note that, according to DIGGS examples, this profile appears to be too restrictive for DIGGS because it uses only the EPSG:4326 coordinate system.

We do not recommend creating and maintaining a custom GML profile for DIGGS or adopting a pre-defined profile. Creating and maintaining a formal GML profile for DIGGS is unnecessary work. The existing official GML profiles are probably either too restrictive or not restrictive enough to meet DIGGS needs.

Instead, we recommend a simpler approach to limit the GML types that can be used. The DIGGS schemas currently allow assignment of varied and complex geometries to DIGGS objects. The geometry of a DIGGS object is specified in the diggs:FeatureType object. Example 2-22 shows that the geometry is of type gml:MultiGeometryPropertyType.

```
<complexType name="FeatureType" mixed="false">
   <complexContent mixed="false">
     <extension base="gml:AbstractFeatureType">
       <sequence>
         ...
         <element name="geometry" type="gml:MultiGeometryPropertyType" minOccurs="0">
         ...
       </sequence>
     </extension>
   </complexContent>
 </complexType>
```

**Example 2-22: Current DIGGS FeatureType geometry Definition**

Figure 2-5 shows that the gml:MultiGeometryPropertyObject allows expression of complex geometries such as MultiSurfaces and MultiSolids. This adds significant complexity to DIGGS consumers, as a complete implementation has to parse and recognize these geometries. This may be unnecessary if DIGGS objects are always expressed with a simpler subset of geometry types.

 Copyright © 2009 Compusult Limited 

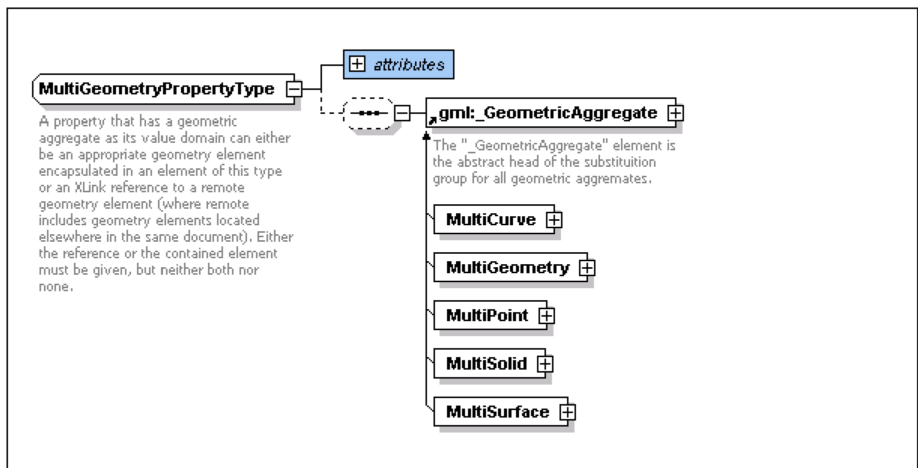


**Figure 2-5: GML MultiGeometryPropertyType**

The XML schema type of the geometry element can be modified to allow a smaller set of possible geometry objects. For example, if DIGGS objects need only geometries of type Point, Polygon, and Line, an XML Schema type can be created to enforce this, as shown in Figure 2-6. The DIGGS FeatureType object can then be modified to use this GeometryPropertyType object, as shown in Example 2-23.

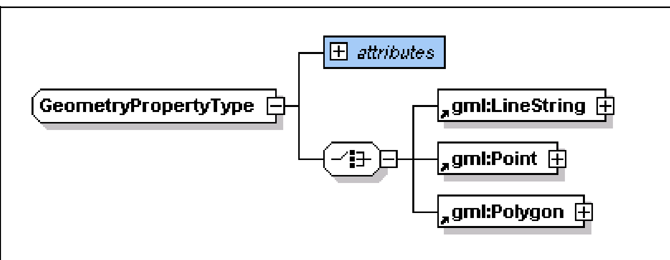


**Figure 2-6: Simplified Geometry Property Type**

 Copyright © 2009 Compusult Limited 

```
<complexType name="GeometryPropertyType">
   <choice>
     <element ref="gml:LineString"/>
     <element ref="gml:Point"/>
     <element ref="gml:Polygon"/>
   </choice>
 </complexType>
 ...
 <complexType name="FeatureType" mixed="false">
   <sequence>
     ...
     <element name="geometry" type="diggs:GeometryPropertyType" minOccurs="0"/>
     ...
   </sequence>
 </complexType>
```

**Example 2-23: FeatureType Object Modified to use Simplified Geometry Property**

The net effect of these changes is that the geometry element of the FeatureType object can now only be a Point, Polygon, or LineString.

### 2.8.3 Rating

**Importance:** Very Important. The use of GML adds considerable complexity and makes the DIGGS schemas more difficult to understand by implementers. The Property pattern currently used by DIGGS violates the GML specification and adds unneccessary complexity to the DIGGS Schemas.

**Scope/Difficulty:** Large Impact. Modifying the schemas to not inherit from GML objects affects just about every DIGGS Schema file. The updates would result in DIGGS objects that are no longer GML features and DIGGS itself not be a GML feature application schema.

**Time Estimate:** 5 days

## 2.9 Flattening DIGGS Files

*DIGGS XML files cannot be flattened (i.e., mapped to an equivalent database with tables and fields).*

### 2.9.1 Summary

This issue is also discussed in Section 3.1.

This issue is not well defined on the DIGGS forum. It has been suggested that this issue may be related to recursion in the schemas; i.e., a recursive relationship cannot be represented in a database. This is incorrect, because nothing prevents two database tables from referencing each other and a single database table can even reference itself.

It has also been noted on the DIGGS forum that criteria for being able to map to a relational data model have not been determined.

The issue may be more related to difficulty in parsing the DIGGS XML Schema and auto-generating a relational data model. The assertion is made in the DIGGS Meeting Presentations and Papers from the March 2009 DIGGS meeting in Orlando that XMLSpy failed when attempting to Create DB Structure from XML Schema. We successfully exercised this function in XMLSpy. The schemas have to validate prior to exercising the function by configuring the XML Catalog correctly. Note that we did experience occasional crashing of XMLSpy when attempting to execute this function.

XMLSpy is the only targeted XML IDE that provides this function. The expected structure of the resulting database is not included in the XMLSpy documentation.

To exercise this function in XMLSpy, a file that directly defines at least one global type must be open and focused. In particular, and as shown in Figure 2-7, the convenience files, such as Geotechnical.xsd, that include only all files defining the Geotechnical namespace types, will not work. However, high-level schema files, such as Diggs.xsd, and files describing objects deep in the Diggs inheritance hierarchy, such as Hole.xsd, can generate a database in XMLSpy.


Copyright © 2009 Compusult Limited

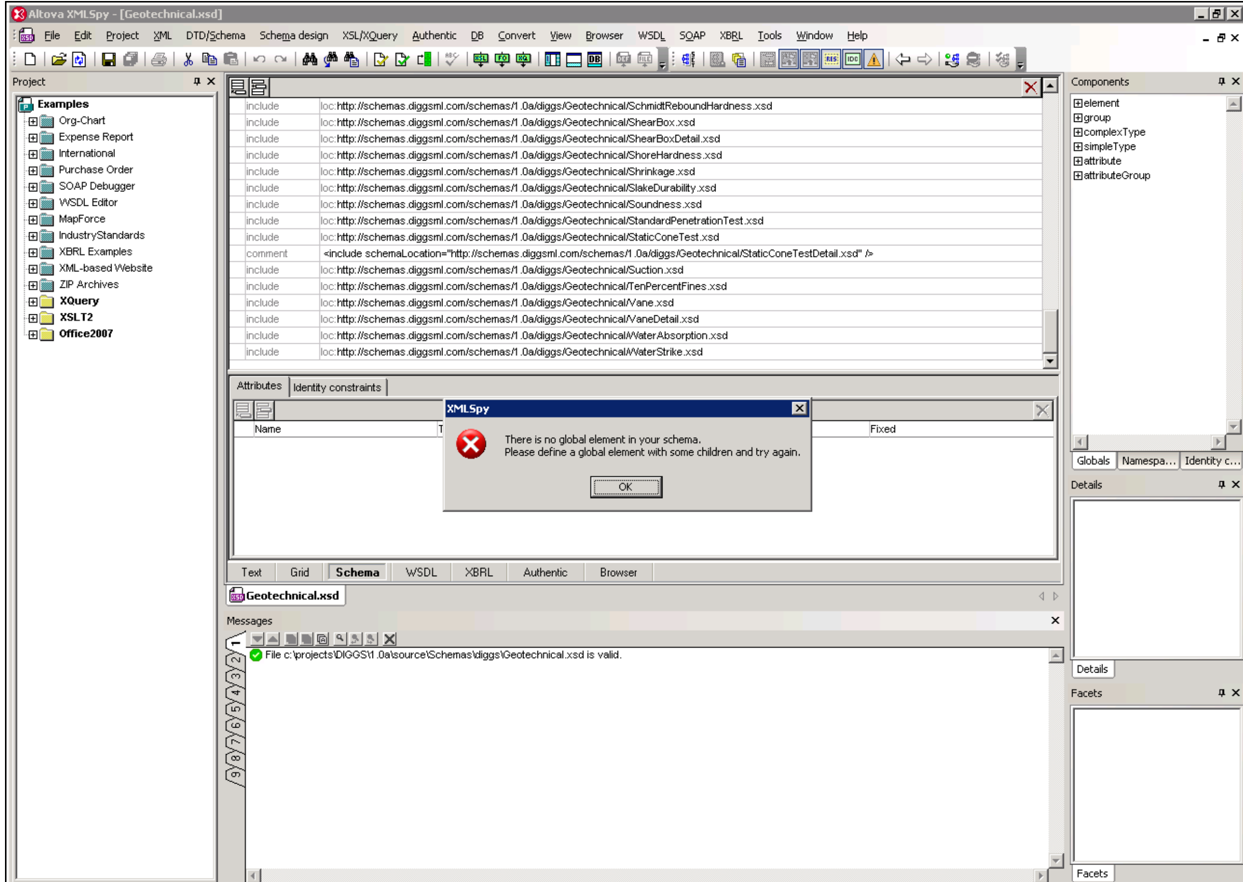


**Figure 2-7: XMLSpy DB Create Warning Message**

### 2.9.2 Recommendation

In our opinion, the ability to map DIGGS XML files to a relational database schema is not a good criterion for measuring the quality or usefulness of an XML Schema. It would be more practical to assess the ability to map an XML Schema to *a specific* database schema.

As noted in Section 3.1, an XML Schema can theoretically be mapped to a relational data model. The issues with this are the complexity of parsing XML Schema and the quality and efficiency of the auto-generated data model. The less complex the schemas, the more likely it is that XML Schema tools will successful parse and auto-generate a relational data model. It should be noted that an auto-generated data model from an XML Schema, in particular the one generated by XMLSpy, may not be of any practical use. In our opinion, the resulting database generated by

 Copyright © 2009 Compusult Limited 

XMLSpy is intended to be a starting point that must then be hand-tailored to meet the objectives of the implementers.

### 2.9.3 Rating

**Importance:** Low Importance. This issue as stated and described does not relate to any known metric for assessing XML Schema quality. The function worked as expected in XMLSpy.

**Scope/Difficulty:** Medium Impact. As noted in the above summary, XMLSpy 2009 generated a relational data model for the current DIGGS 1.0a Schema. The ability of XMLSpy (or any XML Schema tool) depends on how well it can parse the XML Schema and generate a database.

**Time Estimate:** Not Applicable

## 2.10 Naming Convention for Related Elements

*The naming convention for related elements is not consistently implemented throughout DIGGS.*

### 2.10.1 Summary

This is an issue with the names currently used for concepts in DIGGS. The examples given are that the base of object is sometimes called BASE and sometimes called BOT. This inconsistency is attributed to names inherited from AGS. Another example is that a device is named CPTCone whereas the test data is named StaticConeTest. These inconsistencies make the schemas harder to understand.

### 2.10.2 Recommendation

This is not really a technical issue with the DIGGS Schemas. The schema design and review process must ensure that consistency is maintained throughout. Defining conventions will help with overall schema consistency. A thorough review should be undertaken to identify and correct any deviations from the convention or related inconsistencies prior to next DIGGS release. Obviously, inconsistencies related to a requirement to be compatible with an existing standard, such as AGS, cannot be changed without either breaking compatibility or modifying the existing standard.

 Copyright © 2009 Compusult Limited 

### 2.10.3 Rating

**Importance:** Low Importance. This issue is a consistency issue with respect to the names used in the schemas and is not fundamentally related to the structure of the XML Schemas.

**Scope/Difficulty:** Low Impact. Resolving this issue (where possible - note the issue with names inherited from AGS) is a matter of reviewing the schema and renaming objects for consistency.

**Time Estimate:** 1 day

## 2.11 Inheritance of Objects

*Inheritance of objects (e.g., language, units, comments, remarks, roles, etc.) when unwarranted, adds unnecessary complexity.*

### 2.11.1 Summary

The March 2009 DIGGS meeting notes indicate that inheritance is overused in the DIGGS Schema and suggest that too many objects inherit too many inappropriate things. For example, many objects inherit a language property, resulting in the ability for different languages to be used in a single DIGGS project.

As shown in Example 2-24, it is possible to have a DIGGS file indicating English at the top level and containing a Hole object expressed in French with two StandardPenetrationTest objects, one in German, the other in Portuguese.The schema allows this because the Diggs, Hole, and StandardPenetrationTest objects all inherit a language element from their base objects.

 Copyright © 2009 Compusult Limited 

```
<Diggs ... xml:lang="en">
  ...
  <projects>
    <Project>
  ...
        <locations>
        <Hole ... xml:lang="fr">
  ...
          <diggs:insituTesting>
            <StandardPenetrationTest xml:lang="de">
  ...
            </StandardPenetrationTest>
            <StandardPenetrationTest xml:lang="pt">
  ...
            </StandardPenetrationTest>
          </diggs:insituTesting>
          </Hole>
  ...
      </locations>
  ...
    </Project>
  </projects>
  ...
</Diggs>
```

**Example 2-24: Language Inheritance**

The objects in the DIGGS Geotechnical namespace are arranged in two object hierarchies, rooted at DiggsBase and FeatureType. Figure 2-8 and Figure 2-9 respectively illustrate these hierarchies for the Dilatometer and HoleType objects. As shown, both of these DIGGS objects actually extend from the gml:AbstractGMLObject. This section focuses on the object hierarchies as if rooted at the DIGGS DiggsBase and FeatureType objects, while other sections of this document discuss the inheritance from GML.

 Copyright © 2009 Compusult Limited 

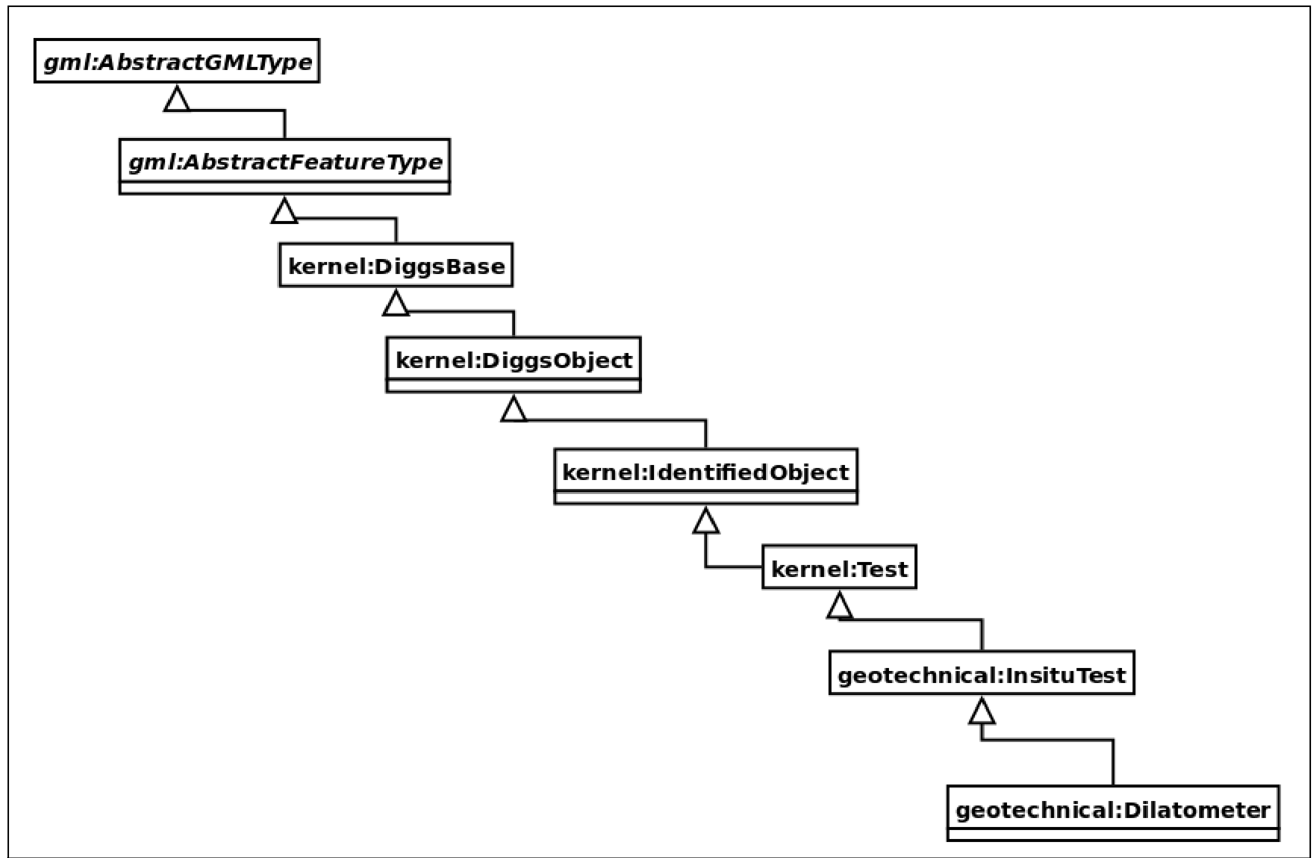


**Figure 2-8: DiggsBaseType Object Hierarchy**

 Copyright © 2009 Compusult Limited 

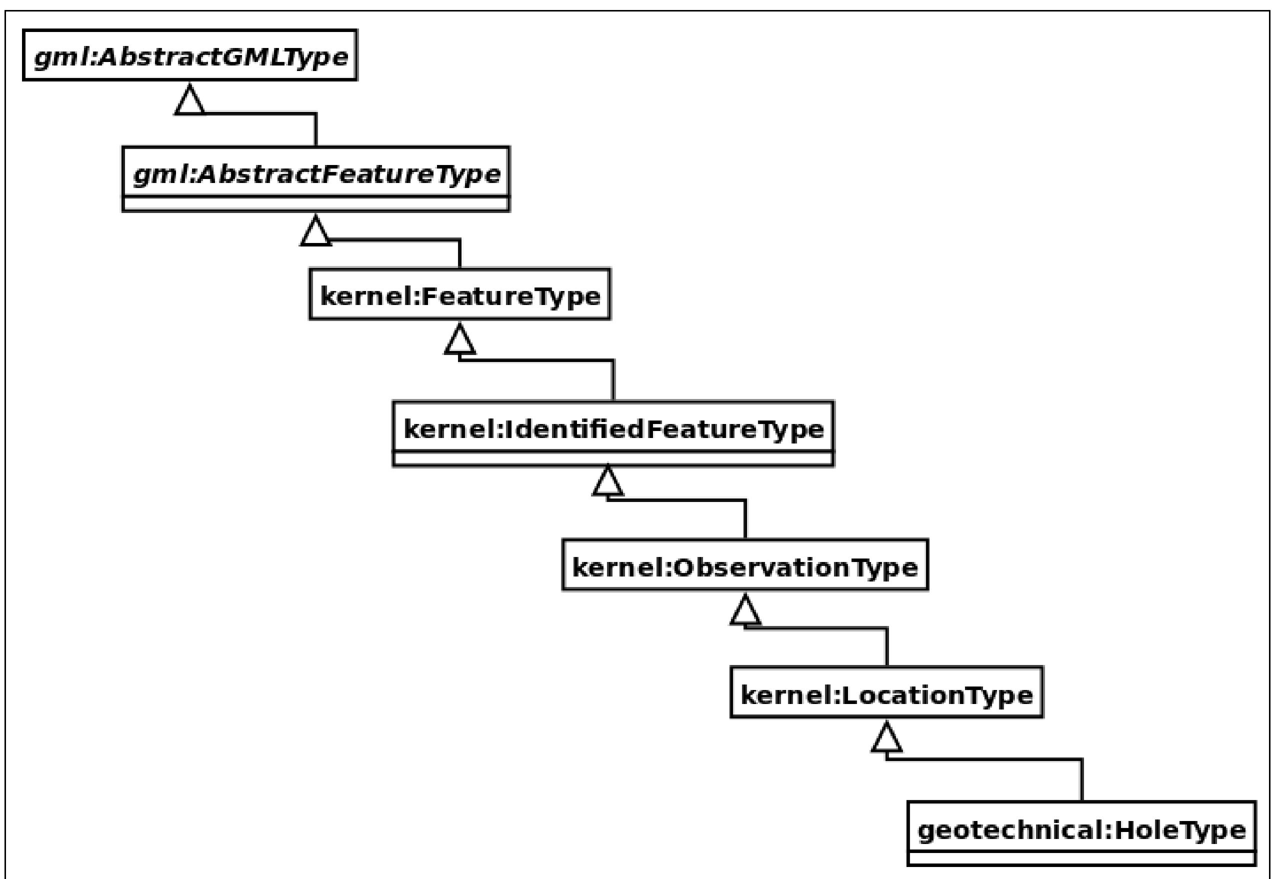


**Figure 2-9: FeatureType Object Hierarchy**

Both of these hierarchies are seven levels deep. Also, based on our analysis of the schemas, over 100 DIGGS objects are in a hierarchy five levels deep. Software development best practice suggests that deep object hierarchies should be avoided and that composition should be favored over inheritance.

In the DIGGS object hierarchy, the inheritance relationship is rarely used polymorphically. A polymorphic inheritance relationship usually occurs when a base type element is used in the schemas. For example, the DiggsBaseType object is never used polymorphically. The DiggsBaseType object is only used where other objects extend it.

The InsituTest object is used polymorphically in the PileConstruction object. The PileConstruction object contains an InsituTestType property, which allows any of the 18 objects that derive from InsituTestType to be used by the PileConstructionType object.

 Copyright © 2009 Compusult Limited 

### 2.11.2 Recommendation

Inheritance should be used only when appropriate, such as defining specializations of a general concept where it would be reasonable to use the general object as a placeholder. Specializations of the general concept can be “plugged-in” where it is specified. For this purpose, it is very appropriate and powerful and should be used as much as possible. However, it should not be used as a shortcut for adding properties of one object to another, especially if not all of the properties of the parent object are appropriate.

The DIGGS schemas should be refactored to employ inheritance only where it is used polymorphically. This is a significant effort, because inheritance is very prevalent throughout the schemas. In particular, the DiggsBase, DiggsObject, and IdentifiedObject objects should probably be removed, and certainly no objects should derive from these.

These objects are examples of inappropriate use of inheritance. Their purpose appears to be as a shortcut for adding a common set of attributes and elements to other objects. For example, the DiggsBase object contains the following child objects:

- xml:lang attribute
- diggs:associatedFiles element
- diggs:remarks element

These items are likely common to many of the objects in the DIGGS schema, so a shortcut for ensuring that all DIGGS objects contain these items is to have all DIGGS objects derive from the DiggsBase object.

However, this is incorrect because it does not model an “is a” relationship. For example, a DIGGS CompactionDetail object is never considered to be a DiggsBase object. Conversely, the relationship between an InsituTest and a StaticConeTest is probably an appropriate “is a” relationship, i.e., a Static Cone Test “is an” Insitu Test.

Also, as noted above, since the language attribute is at the root of the object hierarchy, it is inherited in appropriate ways.

Objects legitimately requiring elements and attributes belonging to DiggsBase should include these directly. This may seem repetitive, but it is the correct approach. To illustrate this, consider the hypothetical DIGGS object in Example 2-25. This object inappropriately extends the DiggsBaseType object. A valid instance of this element is shown in Example 2-26. Note that the fields inherited from DiggsBaseType are populated with values.

 Copyright © 2009 Compusult Limited 

```
<xs:element name="MyDiggsObject"/>
 <xs:complexType name="MyDiggsObjectType">
   <xs:complexContent>
     <xs:extension base="diggs:DiggsBaseType"/>
   </xs:complexContent>
 </xs:complexType>
```

**Example 2-25: MyDiggsObject Inappropriate Inheritance**

```
<diggs:MyDiggsObject lang="en">
  <diggs:associatedFiles>
    <diggs:Ref xlink:href="http://reference.com?id=ZZZ"/>
  </diggs:associatedFiles>
  <diggs:remarks>
    <diggs:Remark>
      <diggs:id>A-1</diggs:id>
      <diggs:content>This is a remark</diggs:content>
    </diggs:Remark>
  </diggs:remarks>
</diggs:MyDiggsObject>
```

**Example 2-26: Sample MyDiggsObject**

An alternative approach that does not use inappropriate inheritance is shown in Example 2-27. In this case, MyDiggsObjectType does not extend DiggsBaseType; instead it duplicates the content.

```
<xs:element name="MyDiggsObject"/>
 <xs:complexType name="MyDiggsObjectType">
   <xs:sequence>
     <xs:element name="associatedFiles"
       type="diggs:AssociatedFilePropertyType"/>
     <xs:element name="remarks"
       type="diggs:RemarkPropertyType"/>
   </xs:sequence>
   <xs:attribute name="lang"/>
 </xs:complexType>
```

**Example 2-27: MyDiggsObject Alternative**

In the software development world, “strong coupling” is an undesirable software attribute and inheritance is considered to be one of the strongest forms of coupling. Strong coupling can lead to issues such as change in one object forcing a change in other objects and causing a ripple effect in other objects. Objects can be difficult to understand in isolation.

 Copyright © 2009 Compusult Limited 

### 2.11.3 Rating

**Importance:** Large Importance. Resolving this issue will make the schemas more usable and understandable in the short term. It will also reduce dependencies among objects, which should make future schema changes easier.

**Scope/Difficulty:** Large Impact. The current schemas extensively use inheritance. Refactoring this requires knowledge of the DIGGS application domain and XML Schema. It will also affect the structure of many of DIGGS objects.

**Time Estimate:** 5+ days

 Copyright © 2009 Compusult Limited 

# 3.0 PERFORMANCE REQUIREMENTS

Five major objectives were identified by the DIGGS Project Team as Performance Requirements required for the next DIGGS release. The following sections discuss the Performance Requirements in terms of the current DIGGS 1.0a Schema and provide a schema modification roadmap towards meeting them.

The Performance Requirements are usually related to one or more of the Core Issues described in Section 2.0. In general, addressing the Core Issues should increase the likelihood of achieving the Performance Requirements.

## 3.1 Ability to be Mapped to a Relational Database

Theoretically, the XML defined by an XML Schema can be mapped into a relational database table structure.

For example, one possible approach is to generate a table for each non-abstract complex type contained in the schema. The table columns should all be simple valued elements or attributes directly contained within the type, including those inherited through substitution groups. For each directly contained complex type, an intersecting table should be generated that maps a row in the containing complex type to rows in the contained complex type table. Mandatory versus optional simple elements can be enforced by the database; however, choices (either via the XML Schema choice element or substitution groups) cannot be enforced. This approach will generate a Relational Database Management System (RDBMS) data model that can store the information in the XML instance documents that are defined by an XML schema document.

While theoretically possible, this approach depends on parsing XML Schema, which is a complex and non-trivial task. Typically, the larger and more complex a schema is, the more likely XML Schema parsers will have issues parsing it. Note that the only targeted XML tool that provides this functionality is XMLSpy. We confirmed during our analysis that this function could be executed by XMLSpy for various DIGGS schema files.

The DIGGS schemas should be as simple as possible and the use of features such as extension and restriction should be minimized to help ensure that they can be mapped to a relational database using automated tools.

Note that mechanical mapping of the DIGGS schema (or any XML Schema) to a relational data model may not result in an efficient or easy to use that model.

In our opinion, this objective and the expected result requires further clarification. That is,

- Must the data model store XML instance documents without loss of information?
- Must the data model enforce all the constraints defined in the XML Schema?

 Copyright © 2009 Compusult Limited 

- Must an existing tool (such as XMLSpy) be able to generate a relational data model from the schema?
- Must the resulting data model meet any specific requirements or can it be any data model, as long as the mapping function succeeds?

As stated above and in Section 2.9, a relational data model can be created so that an XML Schema can be mapped to it and the XMLSpy application can generate a database from various existing DIGGS 1.0a Schema files. However, we believe that these capabilities do not positively correlate with the quality or usability of the schema.

If the DIGGS community continues to consider this a priority objective, the following steps will help ensure the schemas can be mapped to relational data model:

1. Clearly define the requirements of the relational data model.
2. Using XMLSpy, (or whatever tool will be used to generate the data model), start with a small subset of the DIGGS schemas and then tailor the schemas so the tool will generate a data model meeting the requirements defined in Step 1.
3. Iteratively add components, confirm that defined requirements are met and, as required, tailor the schemas accordingly.

## 3.2 Ability to Work with Leading XML Editing Tools

The targeted XML Integrated Development Environments (IDEs) all performed basically as expected with the existing DIGGS schemas. In our assessment, the most difficult issue we encountered when working with the schemas was the use of OASIS XML Catalog. Section 2.3 discusses this issue in detail, concluding that this is not unnecessary and that the schemas can be modified to not require the use of the OASIS XML Catalog. Unless the catalog is configured correctly, the tools will indicate that the schemas are invalid.

We recommend the following steps to ensure the schemas will work with the targeted XML tools:

1. Define which tool functions must work. In each case, define both the function and the expected result. For example, it should be stated whether the schemas must validate in each tool. If certain specialized functions of a given tool must work, this should also be clearly defined. For example, the Create DB from XML Schema function in XMLSpy must generate a relational data model that meets specific requirements.
2. Perform periodic, regular tests with the tools to ensure that the required functions work and generate the expected results.

 Copyright © 2009 Compusult Limited 

## 3.3 Ability to Generate Blank DIGGS XML Files Using Standard XML Editing Tools

The XML IDEs targeted by the DIGGS community provide the option to generate XML from the schemas. This feature can be used to generate a starting point that can then be edited to produce a working sample. When generating sample XML from a schema, note that:

- The generated XML is valid only according to rules enforced by XML Schema. Domain-specific business rules that ca not be expressed in XML Schema language will not be reflected in the generated XML. For example, if the schema defines minimum and maximum concepts for a particular object, where it is understood that the minimum is always less than or equal to the maximum, this rule cannot be enforced by XML Schema and therefore will not be reflected in the generated XML.
- XML Schema can specify the ability to have alternatives in the generated XML. For example, XML Schema defines a choice element indicating that a valid XML instance document contains only one of the specified choices. Choice can also be specified by using substitution groups, where any member of a particular substitution group is a valid alternative for an element.

It must be understood that generated XML will be schema valid (or close to schema valid) only if it contain one of these alternatives. The IDEs typically do not provide an interactive way to select an alternative at each decision point, so the resulting XML may not contain the desired choices.

In general, the IDEs provide the following options:

- **Select Root element** - Any global element can be chosen as the root of a generated sample.
- **Generate all or mandatory elements and attributes** - A minimal sample can be generated that contains only mandatory elements or attributes and a complete sample can be generated that contains all mandatory and optional elements.
- **Fill elements and attributes with data** - Generated samples will have elements and attributes populated with values based on the data type for the element or attribute. XMLSpy will not generate an XML document if the Fill Elements with Data option is selected, apparently due to its inability to generate a DIGGS ID value. Oxygen will generate an XML document; however, the DIGGS ID it creates does not match the required pattern. In general, the resulting XML document cannot be schema valid unless the elements and attributes are populated, because mandatory elements or attributes require data.

As discussed in Section 2.7, the targeted XML IDEs could do this for various DIGGS global elements, including the top level Diggs object, as well as an object such as Hole that is deep in the DIGGS object hierarchy. These functions worked as expected. In our assessment, the automatically-generated samples are intended to be just a convenience mechanism for producing an initial document that can then be hand-edited to meet a particular need.


Copyright © 2009 Compusult Limited

In our opinion, even though these functions worked as expected in the targeted XML tools, this capability does not positively correlate with the quality or usability of an XML Schema.

## 3.4 Demonstrate Import / Export Between Vendors - Ability to Exchange Examples

Successfully importing and exporting DIGGS documents between vendors with no loss of information will be enhanced if the rules defining a valid DIGGS document are clearly and unambiguously stated. This allows vendors to implement processing software without having to interpret what to expect from other vendors.

The schemas should be designed so that XML Schema validation can be used wherever possible to determine the logical validity of a DIGGS document. As discussed in Section 1.1, XML Schema may be unable to completely enforce all rules defining a logically-valid DIGGS document. Rules that cannot be enforced by XML Schema must be clearly and unambiguously documented outside the schema.

Interoperability between vendors can also be enhanced by minimizing the amount of choices and options in the schemas. In practice, when a schema provides many choices and options, vendors may only implement support for the ones that are most important to them. They may provide only limited support (or none at all) for alternatives they feel are not critical for using their software. Note that this consideration may make it more difficult to develop an implementation, but the result will be improved interoperability.

Prior to the next DIGGS release, several interoperability experiments should be defined and executed in consultation with key software vendors to identify and resolve potential interoperability issues with the schemas and the specification.

 Copyright © 2009 Compusult Limited 

# 4.0 ASSESSMENT OF GML USAGE

## 4.1 Were GML Standards, Conventions, and Best Practices Implemented Correctly?

The DIGGS 1.0a schemas are implemented as GML 3.1 feature application schemas. The high level requirements of a GML feature application schema are as follows:

- All feature types declared in an application schema shall derive either directly or indirectly from gml:AbstractFeatureType.
- All geographic features shall be global elements in the schema.
- The name of an element that instantiates a GML shall be its Feature Type.
- The name of feature element is the semantic type of the feature.
- The children of a feature are always properties that describe the features.

These requirements appear to be followed throughout the DIGGS Geotechnical schema files. However, there is an error in the implementation of the last requirement listed above. The error is described below.

In addition to the requirements of a GML feature application schema described above, the GML specification recommends the following lexical conventions:

- objects are instantiated as XML elements with a conceptually meaningful name in UpperCamelCase;
- properties are instantiated as XML elements whose name is in lowerCamelCase;
- abstract elements have an underscore prepended to their _name;
- the names of XML Schema complex types are in UpperCamelCase ending in the word Type; and,
- abstract XML Schema types have the word Abstract prepended.

These conventions are followed consistently throughout the DIGGS Geotechnical schema files.

The DIGGS 1.0a schemas violate Section 7.4.3 in the GML 3.1 specification with respect to the definition of a GML Object and the GML property pattern. In particular, the GML Specification states the following:

- A GML object is an XML element of a type derived directly or indirectly from AbstractGMLType. From this derivation, a GML object may have a gml:id attribute.
- A GML property may not be derived from AbstractGMLType, may not have a gml:id attribute, or any other attribute of XML type ID.

 Copyright © 2009 Compusult Limited 

- An element is a GML property if and only if it is a child element of a GML object.
- No GML object may appear as the immediate child of a GML object.
- Consequently, no element may be both a GML object and a GML property.

The basic pattern followed in the DIGGS Geotechnical namespace is that an object is defined by an XML Schema complex type named {Object}Type that (indirectly) derives from AbstractGMLType (via AbstractFeatureType - see Figure 4-3 for an example of the DIGGS hierarchy). Additionally, an XML Schema complex type named {Object}PropertyType is created that appears to be intended to support the GML property pattern and the requirement that properties of GML Features must be encoded as child elements of the feature. The {Object}PropertyType complex type inherits from GML AbstractFeatureCollectionType, which derives from GML AbstractGMLType. Hence, the DIGGS {Object}PropertyType is, by definition, a GML object. The GML specification is violated anywhere that a DIGGS {Object}PropertyType is a child of any DIGGS {Object}Type object.

The property pattern suggested by GML is demonstrated in Example 4-1, which describes a SatelliteImageType GML Feature that has a property for the date and time the image was captured.

```
<xs:simpleType name="DateAndTimePropertyType">
   <xs:restriction base="xs:dateTime"/>
 </xs:simpleType>

 <xs:element name="SatelliteImage" type="example:SatelliteImageType"
    substitutionGroup="gml:_Feature"/>

 <xs:complexType name="SatelliteImageType">
   <xs:complexContent>
     <xs:extension base="gml:AbstractFeatureType">
       <xs:sequence>
         <xs:element name="imageDateAndTime"
            type="example:DateAndTimePropertyType"/>
       </xs:sequence>
     </xs:extension>
   </xs:complexContent>
 </xs:complexType>
```

**Example 4-1: GML Feature with Date Property**

The DIGGS schemas attempt to follow this pattern; however, the DIGGS Property objects derive from AbstractGMLType, which means they cannot be used as child elements of DIGGS objects.

As shown in Example 4-2, the HoleType contains a child element of type BackfillPropertyType. Both HoleType and BackfillPropertyType indirectly derive from gml:AbstractGMLObject which is a violation of the GML specification.

 Copyright © 2009 Compusult Limited 

```
<complexType name="BackfillPropertyType">
   <complexContent>
     <extension base="gml:AbstractFeatureCollectionType">
       <sequence>
         <element ref="diggs_geo:_Backfill" minOccurs="0" maxOccurs="unbounded"/>
         <element ref="diggs:Ref" minOccurs="0" maxOccurs="unbounded"/>
       </sequence>
     </extension>
   </complexContent>
 </complexType>

 <complexType name="HoleType" mixed="false">
   <complexContent mixed="false">
     <extension base="diggs:LocationType">
       <sequence>
         ...
       <element name="backfills" type="diggs_geo:BackfillPropertyType" minOccurs="0"/
                                           >
         ...
       </sequence>
     </extension>
   </complexContent>
 </complexType>
```

**Example 4-2: Invalid Property Pattern Used by DIGGS**

The DIGGS 1.0a Schema is implemented as a GML 3.1.1 feature application schema. GML 3.1.1 was the latest GML version at the time of development, and thus was a reasonable choice of the GML version to use.

DIGGS needs many concepts defined in GML, such as the ability to represent geospatial locations, coordinate reference systems, and units of measure. Therefore, GML is a good choice to represent these concepts in DIGGS.

We recommend that DIGGS should continue to use GML and that GML 3.2.1 be used. Version 3.2.1 was released as an ISO Standard 19136:2007 (http://www.iso.org/iso/catalogue_detail.htm?csnumber=32554) and many improvements were made through the ISO standardization process. In particular, changes where made to address some issues that XML tools and parsers (such as JAXB) had with consuming GML schemas. GML 3.2.1 was tested to demonstrate that it works in common XML tools and code generation utilities. GML 3.2.1 has also been updated to make it easier to understand. For example, the “_Name” convention has been replaced by “AbstractName”.

 Copyright © 2009 Compusult Limited 

## 4.2 Is DIGGS Best Implemented as a GML Application Schema?

The primary reason for choosing to use GML in DIGGS is that it supports many XML Schema types required by DIGGS, such as geometry objects, coordinate reference system information, and dictionary structures. By using GML, there is no need to develop DIGGS-specific representations of these types.

Another, lower priority motivation for using GML as the basis for DIGGS is so that a general purpose GML application can read DIGGS files. However, as mentioned during the presentations at the March 2009 meeting in Orlando, the DIGGS 1.0a Schema files did not work in three major Geographic Information System (GIS) software packages (i.e., AutoCAD Map, ArcMap, and MapInfo). DIGGS data could be viewed using the GML viewer from Snowflake Software, but the attributes did not display.

As noted above, the ability to read DIGGS files in GML applications is a low priority objective. By implementing DIGGS as a GML feature application schema, general purpose GML applications may be able to read DIGGS files. As noted above, this did not work in the current implementation, but that may be due to errors in GML usage by the DIGGS Schemas.

The DIGGS schemas do not achieve any other functionality or significant benefit as a GML feature application schema. Figure 4-3 shows the usage of GML in the HoleType object hierarchy. The three fields gml:id, gml:description, and gml:name are the only fields inherited from the GML AbstractFeatureType derivation that are used. Other DIGGS objects use GML objects as properties (i.e., as child elements). For example, as shown in Figure 4-3, the diggs:FeatureType object has the following child elements based on GML types

- status: gml:CodeType
- defaultCRS: kernel:AnyCRSPropertyType
- geometry: gml:MultiGeometryPropertyType
- otherCRS: kernel:AnyCRSPropertyType

The GML types gained from being a DIGGS feature application schema (via inheritance from GML AbstractFeatureType) are simple, general purpose types that are not specific to GML. Implementing DIGGS as a GML feature application schema does not help to meet DIGGS objectives.

 Copyright © 2009 Compusult Limited 

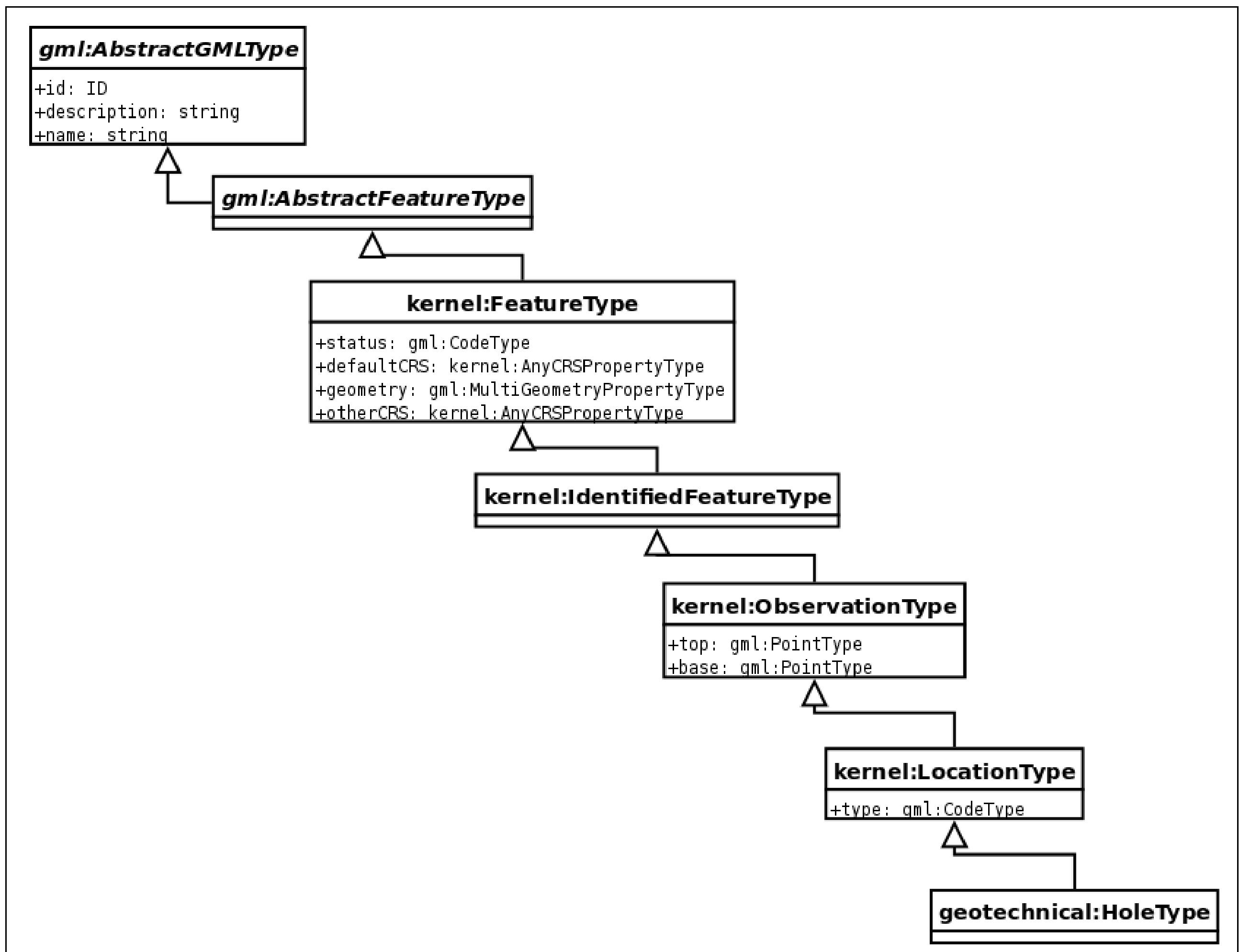


**Figure 4-3: Hole Inheritance Hierarchy**

We recommend that DIGGS continue to use the GML types applicable to DIGGS, as shown in the diggs:FeatureType, diggs:ObservationType and diggs:LocationType objects. However, the derivation of diggs:FeatureType from gml:AbstractFeatureType does not benefit DIGGS and increases DIGGS complexity. We therefore recommend that DIGGS not be implemented as a GML feature application schema, but continue using applicable GML elements.

An implication of our recommendation is that DIGGS would no longer be a GML feature application schema. As stated above, DIGGS would lose no functionality from this modification, but general purpose GML applications would no longer recognize DIGGS objects because they will not be GML Features.

 Copyright © 2009 Compusult Limited 

# 5.0 WEB-BASED COMMUNITY SCHEMA DEVELOPMENT

There is no well-established XML Schema specific approach to Web-based community schema development. However, XML Schema development does not present any special issues that would preclude the use of the well-known Revision Control Systems (RCS) for DIGGS Schema development. XML Schema documents are text documents and are well-suited to typical RCS. An RCS provides a history of revisions, with comments regarding changes, concurrency control, and the ability to diff revisions and merge files in a local repository via distributed updates.

The Subversion RCS is our recommended RCS implementation because it is freely available and actively developed. It is supported on UNIX, Linux, Windows, and Mac OS X. It supports access through many Graphical User Interfaces (GUIs) and Integrated Development Environments (IDEs), such as the Oxygen XML editor. The Subversion RCS also allows multiple users to work on parts of the schemas concurrently by managing conflicts between simultaneous edits.

The DIGGS online forums have been used effectively as part of the development process up to the current version. They should continue to be used to aid in the collaboration process.

 Copyright © 2009 Compusult Limited 

# 6.0 OTHER ISSUES

## 6.1 Validation Issues in Oxygen

### 6.1.1 Summary

For the schemas to validate in Oxygen running on Linux, the following changes had to be made to the distribution:

- The file kernel.xsd was renamed to Kernel.xsd. Filename case is significant when using Linux.
- In the file 1.0a/source/Schemas/diggs/geophsyical.xsd:

```
<import namespace="http://schemas.diggsml.com/1.0a"
   schemaLocation="http://schemas.diggsml.com/schemas/1.0a/diggs/kernel.xsd"/>
```

was changed to:

```
<import namespace="http://schemas.diggsml.com/1.0a"
    schemaLocation="http://schemas.diggsml.com/schemas/1.0a/diggs/Kernel.xsd"/>
```

- In the file 1.0a/source/Schemas/gml/3.1.1/smil/smil20.xsd:

```
<import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="xml-mod.xsd"/>
```

was changed to:

```
<import namespace="http://www.w3.org/XML/1998/namespace"/>
```

 Copyright © 2009 Compusult Limited 

# 7.0 ASSESSMENT OF SCHEMA ORGANIZATION AND COMPOSITION

## 7.1 Assess Schema Organization and Composition Strategy

The DIGGS Schemas are organized into the following five subgroups:

1. **Kernel:** Common objects shared among the other subgroups
2. **Environmental:** Objects related to the Environment subgroup
3. **Geotechnical:** Objects related to the Geotechnical subgroup
4. **Monitoring:** Objects related to the Monitoring subgroup
5. **Piling:** Objects related to the Piling subgroup

The files that define each subgroup are contained in a separate folder and each subgroup is defined within its own XML namespace. For example, all files defining objects in the Geotechnical subgroup are contained in the Geotechnical folder and all objects in this folder are in the DIGGS Geotechnical namespace http://schemas.diggsml.com/1.0a/geotechnical.

There are convenience files at the top level to tie things together. For example, there is a file Geotechnical.xsd that includes all files in the Geotechnical namespace.

In general, each object is defined in a separate file within each subgroup. For example, within the Geotechnical sub-group there are separate files for the Hole object (Hole.xsd), Backfill (Backfill.xsd) object, and Dilatometer (Dilatometer.xsd) object. Each file generally defines one type and the file is named after the global element.

Within each file, the following four global types are defined:

1. A complexType that defines the object.
2. An abstract element that instantiates the complexType defined in the file. This element is in the substitution group of the parent abstract element for this object.
3. A non-abstract element that instantiates the complexType defined in the file. This element is in the substitution group of the abstract type defined in the file.
4. A complexType named {Object}PropertyType that groups a sequence of unbounded elements of the type defined in the file and a sequence of unbounded diggs:Ref elements. This complexType also extends the gml:AbstractFeatureCollectionType object.

For example, the Hole.xsd file defines the following:

1. A complexType named HoleType that defines a Hole object. This object extends the diggs:LocationType object.

 Copyright © 2009 Compusult Limited 

2. An abstract element named _Hole that is of type HoleType in the substitutionGoup of the diggs:_Location element.
3. A non-abstract element named Hole that is of type HoleType in the substitutionGroup of the _Hole element.
4. An complexType named HolePropertyType that defines a sequence of _Hole elements and diggs:Ref elements. This type extends the gml:AbstractFeatureCollectionType.

The physical file and directory structure is clearly organized and easily understood. For the most part, the file naming convention is consistently implemented. Also, the structure of each individual file is consistently implemented. A consistent, regular file naming convention and file structure greatly enhances the ability to understand and become familiar with the schemas.

The logical organization of the DIGGS schemas overuses and inappropriately uses XML Schema inheritance. As discussed in Section 2.11, the schemas have the following inheritance problems:

- The inheritance hierarchies are too deep. Some of the object hierarchies in the DIGGS schemas are up to seven levels deep.
- The inheritance between many DIGGS objects is used inappropriately. The inheritance relationships are rarely used polymorphically in the schemas.
- There is little benefit in the inheritance from gml:AbstractFeatureType. Although this is a requirement of GML feature application schemas, no gain in functionality is achieved and the schemas are made more complicated as a result.
- The property pattern unnecessarily inherits from gml:AbstractFeatureCollectionType.

The layout of the individual files is consistently implemented; however, the convention of creating both an abstract and non-abstract element for each DIGGS type is unnecessary.

```
<element name="_Hole" type="diggs_geo:HoleType" abstract="true"
   substitutionGroup="diggs:_Location"/>

 <element name="Hole" type="diggs_geo:HoleType" abstract="false"
   substitutionGroup="diggs_geo:_Hole"/>
```

**Example 7-1: Abstract and Non-Abstract Hole Elements**

As shown in Example 7-1, the abstract _Hole element is of type HoleType and, since it is a member of the diggs:_Location substitutionGroup, can used anywhere diggs:_Location is specified. However, since _Hole is abstract, only a non-abstract member of the _Hole substitutionGroup can be used wherever _Hole is specified.

 Copyright © 2009 Compusult Limited 

As also shown in Example 7-1, the Hole element is a non-abstract element of type HoleType and can be used wherever _Hole is specified. The result is that a Hole element can be used where a diggs:_Location element is specified.

```
<element name="Hole" type="diggs_geo:HoleType" abstract="false"
  substitutionGroup="diggs_geo:_Location"/>
```

**Example 7-2: Simplified Hole element**

As illustrated in Example 7-2, the same level of functionality can be achieved with a single element of type HoleType that is in the substitutionGroup of diggs:_Location. The head of a substitution group does not need to be abstract and there is no requirement of GML feature application schemas to use this pattern.

Overall, the physical layout of the DIGGS files is very good. The logical arrangement is problematic due to the overuse of inheritance and consistent definition of redundant abstract and non-abstract global elements for each of the global complexTypes created.

 Copyright © 2009 Compusult Limited 

## 8.0 SUMMARY OF RECOMMENDATIONS

We suggest that the DIGGS schemas are not yet ready for official release. As a roadmap forward, we recommend that the following items be addressed for the next DIGGS release:

1. Refactor the objects to limit the use of inheritance. The overuse of inheritance is probably the single biggest issue with the DIGGS schemas. However, this will also be the most difficult issue to address, since XML Schema inheritance is currently used very extensively throughout the schemas. Most objects belong to one of the two main DIGGS inheritance trees.
2. Do not focus on making the DIGGS schemas GML feature application schemas. Being compliant as a GML feature application adds complexity to the DIGGS schemas and does not directly yield or overlap with any required DIGGS functionality.
3. Use the extensible XML Schema enumeration approach for code lists described in Section 2.5.
4. Use the table object construct suggested in Section 2.4.
5. Use the XML Schema <key> element to enforce DIGGS Key Fields within XML instances.
6. Consider using Universally Unique Identifier (UUID) values instead of codespace identifiers for DIGGS objects.
7. Consider using the suggestions from Section 2.3 so that the DIGGS schemas do not require an XML Catalog.

 Copyright © 2009 Compusult Limited 