Consistency Verification for GML Data Based on DOM

Xiaoli Gao[1(✉)], Haixia Li[1], Tingguang Yan[1], Zhencai Cui[1], Jiyu Yu[1], and Yehua Sheng[2]

[1] Shandong Water Polytechnic, Rizhao, Shandong, China
wishuluck@126.com
[2] Key Laboratory of Virtual Geographic Environment, Nanjing Normal University, Nanjing, China

Abstract. GML schemas are metadata files, which define the structure, content and restriction of GML instances. As a kernel of the GML parser, consistency verification decides whether GML documents are consistent with the relevant application schemas. In order to parse GML data more effectively and accurately, an algorithm based on DOM was developed as to how GML consistency can be verified. Furthermore, some primary user-defined methods and the homologous regular expression technology, involving in this algorithm, were discussed in detail. Experimental results show that the consistency verification algorithm is efficient.

Keywords: GML · Schema-based parser · DOM · Consistency verification · RegExp · HRegExp

1 Introduction

As the encoding standard for geographic information, GML documents have to meet certain grammar specification to guarantee the correctness of physical structure and logical structure, thus, which can be interpreted, process and sharing [4]. GML parsing is the basis of geographic information storage, compress, transformation, conversion, index, query and share, etc., which is related to studying and application of GML tightly [5]. Almost all the studies related to GML, cannot be separated from GML parsing. GML schema-based parser should be versatile. It is able to preserve the properties of adaptive and self-expansion [6].

The integrative grammatical and semantic database, addresses the issue of how to describe simple data type and complex data type in schemas, and lays the foundation for GML grammar validation and GML grammatical and semantic parsing [1].

The GML grammar validation is the most basic and important foundation of GML schema-based parser, and its purpose are to verify the correctness and the validity of

This work is sponsored by the Special Water Resources Projects Aided by Special Fun of Water Resources Department of Shandong Province (No. sdw200709027) and the Provincial Water Conservancy Science Research and Technology Promotion Project of Shandong Province (No. SDSLKY201304).

F. Bian and Y. Xie (Eds.): GRMSE 2015, CCIS 569, pp. 149–158, 2016.
DOI: 10.1007/978-3-662-49155-3_14

the GML data that based on the two aspects: validity and consistency, to ensure the GML data is available.

Consistency verification is also known as the validity verification. It is an inference that whether the GML instances and the GML patterns can match in the data structure, which mainly for the effective judgment that whether elements, attributes, and data structure of the GML instances conform with the corresponding defined requirements of GML application schemas. When an inconsistent error occurs, it can prompt the wrong position and the wrong type information, etc.

Normative GML documents should comply with the requirements of two aspects: To be well-formed and to be valid.

The well-formed GML document is known as the legal GML document, whose content and structure must comply fully with the GML basic grammar rules. Such as the start tag and the end tag must appear in pairs, and the structure of the document must be a hierarchical tree structure, etc.

The GML instance document that has completely passed the consistency test is referred to as consistent (or effective) GML document. Such kind of document must be well-formed, firstly, which also calls for its content and structure shall abide by the provisions of the relevant application schemas or DTDs [7].

The consistent GML document has more restrictions than a well-formed GML document, and has to follow a stricter specification. A well-formed GML document cannot achieve the standard of consistency. However, consistent GML document will meet the requirements of well-formed. From this point, consistency is the proper subset of the legality [3]. In functional implementation of GML consistency verification, the DOM technology mainly used as the GML parsing scheme.

2 The Specification of DOM API

DOM is the abbreviation of Document Object Model, proposed by the World Wide Web Consortium (W3C), was an application programming interface (API) for XML and HTML documents [8].

When parsing GML data using DOM, GML document is organized into a hierarchy tree structure (called a DOM tree), and the DOM tree is loaded into the memory. In DOM tree, all elements are organized as the tree nodes. DOM parser supplies a series of API methods, provides abundant supports for manipulating tree nodes, such as dynamic traversal, retrieving, adding nodes, removing nodes, modifying nodes, etc. Moreover, DOM parser can establish, update, or access the GML document structure and GML data style [7].

DOM offers the following four basic API classes.

- Document class represents the entire GML document. The document object is actually the root of the DOM tree, which is the entrance leads to access data, or other elements of the DOM tree.
- Node class, on behalf of an abstract node of the DOM tree, is the parent class of many other classes.

- NodeList class, providing the abstract definition of an ordered list of nodes, each node appears as an item with its index value in the list.
- NamedNodeMap class, defines the operational methods set. The methods in the set are used to process elements in an unordered collection of nodes. The instance object is primarily used to access an attribute node, or to describe the corresponding relations between a set of nodes and their names.

The working mechanism of DOM tree is shown in Fig. 1.

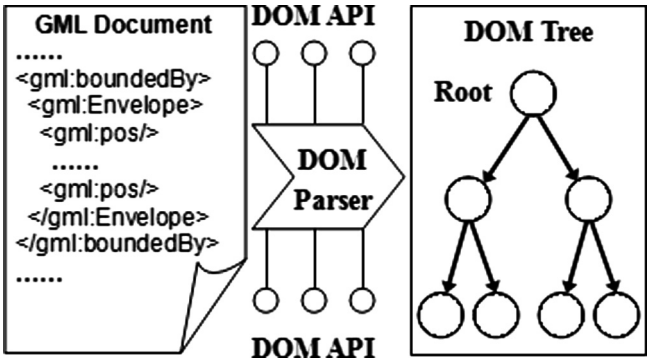

Fig. 1. The working mechanism of GML parsing using DOM.

3 The Realization Principle of Consistency Verification

When the data entries in a GML document are manipulated using DOM, first organize the collections of independent elements, attributes, text, data entity, and so on, into a DOM tree structure in the memory, and make each node represent an embedded object in the GML document. Secondly, a series of API functions are provided; they support the application to access the contents of the DOM tree through the interface functions. So many kinds of operations are allowed to apply to the nodes on a DOM tree, and then process the results of operations, maps the results indirectly to the corresponding documentation [2]. Finally, by complying with the requirements of the application, API functions allow programmers to modify the DOM, and decide whether to save changes back to the GML document.

Consistency validation process depends largely on the basis of the DOM tree and the integrative GML grammatical and semantic database, and the recursive method is utilized in the implementation. It is important in order to inspect whether the contents of a data instance match the corresponding records in the integrative GML grammatical and semantic database. The contents of a data instance are made up of many aspects in each data entity, including the organizational structure, data type, and range of domain [7].

Once found that elements do not agree with the schema definition, the wrong type and the error position will be recorded, so that the verified conclusions will be reported to users at the end of the consistency validation process.

In view of the different element types, the consistency validation behavior will change.

- For the built-in standard simple type element. It is mainly to check whether the value scope of each element complies with the provision of the standard type.
- For the restricted simple type element. It is mainly to check whether the value of each element complies with the facets of the simple type.
- For the complex type element. It is mainly to check whether the hierarchical tree structure which is developed from elements and sub-elements complies with the complex type structure defined by the model.
- For the reference element. It may not directly find the matching record in the integrative GML grammatical and semantic database. Then it is necessary to check the substitution group field in the relevant element definition table. Once the record that has the same name as the reference element has been found, it is time to perform the three conditions, one of the three cases described above will be processed respectively according to the data type of substitution group field is simple or complex, and the corresponding processing module will be performed. If the substitution group field is not present, it indicates that element verification has failed, so the error handling module should invoke.

4 Implementation Algorithm of Consistency Verification

GML parsing relies on DOM trees mapping in memory. The tree data structure belongs to the nonlinear hierarchical structure, and its definition is recursive. Because the recursion has characteristics of concise, delicate, function perfectly, short code, efficient performance, so recursive algorithm was designed in the implementation for consistency verification.

The function of the method named consistency verification is to implement dynamic analysis and check on the specified GML elements. A detailed algorithm description is depicted below.

Input Description: Input objects consist of the specified GML instance document to be verified, the integrative grammatical and semantic database of GML core schemas, and the integrative grammatical and semantic database of GML application schemas.

Function Description: The core function of the algorithm is to check consistency of the specified element. This function returns TRUE on success in consistency verification, or FALSE on failure in consistency verification. When the specified element takes its failure in consistency verification, the method named registerError will be invoked immediately. The function of this method is to record the error location, the error type, the wrong code, etc.

Output Description: Output result is the conclusion through verification test, or error messages caused by verification failures.

4.1 Description of Consistency Verification Method

The pseudo code of the key process in the method called consistency verification is described as follows [1].

```
boolean consistencyVerification(Element checkedElement)
{ //Define a variable to store value returned by method
  boolean verifiedResult=true;
  if ( dataTypeOf(checkedElement)∈standard simple type)
    verifiedResult=
            VerifyStandardSimpleType(checkedElement);
  //Error occurs when verifies standard simple type
  if (verifiedResult==false)
    registerError(); //Invokes the registerError method
  if (dataTypeOf(checkedElement)∈restricted simple type)
    verifiedResult=
          VerifyRestrictedSimpleType(checkedElement);
  //Error occurs when verifies restricted simple type
  if (verifiedResult==false)
  registerError(); //Invokes the registerError method
if (dataTypeOf(checkedElement)∈complex type)
{//To concatenate checkedElement and its child elements
destinationConcatenate=
      generateConcatenateStr(checkedElement);
//To gain the value of  rule field from database
sourceRule=getRuleFromDB(DBFile,checkedElementType);
//To convert the parameter into a regular expression
sourceRegExp=regularization(sourceRule);
//Using regular expression as a template
boolean isMatch= Pattern.matches(
    sourcePattern,destinationConcatenate);
if (isMatch==false) {//Pattern match suffers a failure
  registerError(); //Invokes the registerError method
  verifiedResult=false;  //Assign false to variable
} else
  verifiedResult=true;  //Assign true to variable
//Realize the verification function using recursive
for (eachSubElement∈checkedElement.getChildNodes())
if ((consistencyVerification(eachSubElement))==false) {
  registerError(); //Invokes the registerError method
  verifiedResult=false; //Assign false to variable
} }//To complete the verification of complex type
return verifiedResult; //Returns value of the variable
}  //To stop consistencyVerification method process
```

4.2 Illustrations of Custom Methods

Some primary user-defined methods involving in the method of consistency verification are given in Table 1.

Table 1. Important functions referenced in consistency verification module.

NO.	Method Prototype	Function description
1	String dataTypeOf(Element checkedElement)	To determine the data type of parameter checkedElement
2	void registerError( )	To record and store the error location, the error type, the wrong code, etc.
3	boolean verifyStandardSimpleType(Element checkedElement)	To check whether the parameter of checkedElement complies with the definition of standard simple type according to the records in the integrative grammatical and semantic database
4	boolean verifyRestrictedSimpleType(Element checkedElement)	To check whether the parameter of checkedElement complies with the definition of restricted simple type according to the records in the integrative grammatical and semantic database
5	String generateConcatenateStr(Element checkedElement)	To construct a string expression that is comprised of the element specified by the parameter of checkedElement and its child elements
6	String getRuleFromDB(String DBFileName, Element checkedElement)	To gain the homologous regular expression of a complex type element specified by the parameter of checkedElement from the database called DBFileName
7	String regularization(String sourceRule)	To convert the homologous regular expression into the corresponding regular expression, To prepare for the operation of pattern matching

Specially as is pointed, in generate ConcatenateStr() method, after producing a string expression, pattern matching will be carried out on the expression and the corresponding regular expression of a complex type. The next step is to decide whether the structure of the element specified by the parameter of checkedElement consistents with the structure of data type defined in the pattern files.

4.3 Regular Expression and Homologous Regular Expression

Regular expressions (abbreviated as RegExp) provide a concise and flexible means to identify strings of text, such as particular characters, words, or patterns of characters. RegExp can express the syntactic structure and nesting relationship perfectly, so we select it to describe GML model information defining in complex types. There are numerous advantages in describing and modeling regular language with RegExp. For example, it is easy for people to understand and to use RegExp; it is easy for computers

to parse and to process RegExp, and RegExp can describe complicated things in a simple form [5], etc.

A regular expression is composed of some branches; and a branch is composed of some pieces. The Piece can be seen as the basic independent element of RegExp, which consists of atoms and quantifier.

Primary quantifiers and operators used in RegExp is given in Tables 2 and 3 respectively.

Table 2. List of the quantifiers in RegExp.

No.	Quantifier	Description of usage
1	?	Matches the preceding element zero or one time
2	*	Matches the preceding element zero or more times
3	+	Matches the preceding element one or more times
4	{n}	Preceding element can only occur n times
5	{n,}	Preceding element occurs at least n times
6	{n, m}	Preceding element occurs at least n times and not more than m times
7	No quantifier	Element can only occur one time

Table 3. List of the operators in RegExp.

No.	Operator	Description of usage
1	αβ	Sequence operator, matches token α firstly, then token β
2	α|β	Choice operator, matches either token α or token β
3	(α)	Token α is treated as grouping

The above quantifiers and operators can be brought together to form arbitrarily complex expressions. In order to enhance describing capacity of RegExp for parsing the syntactic and semantic information, some grammar rules are extended. The improved regular expression is called homologous regular expression(abbreviated as HRegExp) in this paper [1].

The following syntax rules were added in HRegExp:

1. For an element in the complex type model, HRegExp decorates the element name with a pair of characters (“<”, “>”) as the delimiters.
2. For an element reference to the complex type model, HRegExp decorates the referenced name with a pair of square brackets (“[”, “]”) as the delimiters.
3. For a named model group in the complex type model, HRegExp decorates the referenced name of the group with a pair of tokens (“[#”, “#]”) as the delimiters.
4. All sub elements in a “sequence group” must be divided by the blank character.
5. All sub elements in a “choice group” must be separated by vertical bar(“|”).
6. All sub elements in a “all group” must be separated by slash(“/”).

7. Uses multi-pairs of nested parentheses("(", ")") to reflect the syntax structure of the nested complex type model.
8. Named model group will conform to the above definition.

Therefore, it can be seen that HRegExp adds some special delimiters for four different types of complex type models, to depict additional semantic information in GML models.

5 Example Test for Consistency Verification Algorithm

The consistency validation program for checking GML data was developed with C#. It can help users to judge whether a GML document is coherent or not [7].

A concrete example test has been laid on. The result of verification as showed in Fig. 2, it indicates that the checked data segment does not meet the requirement of consistency.

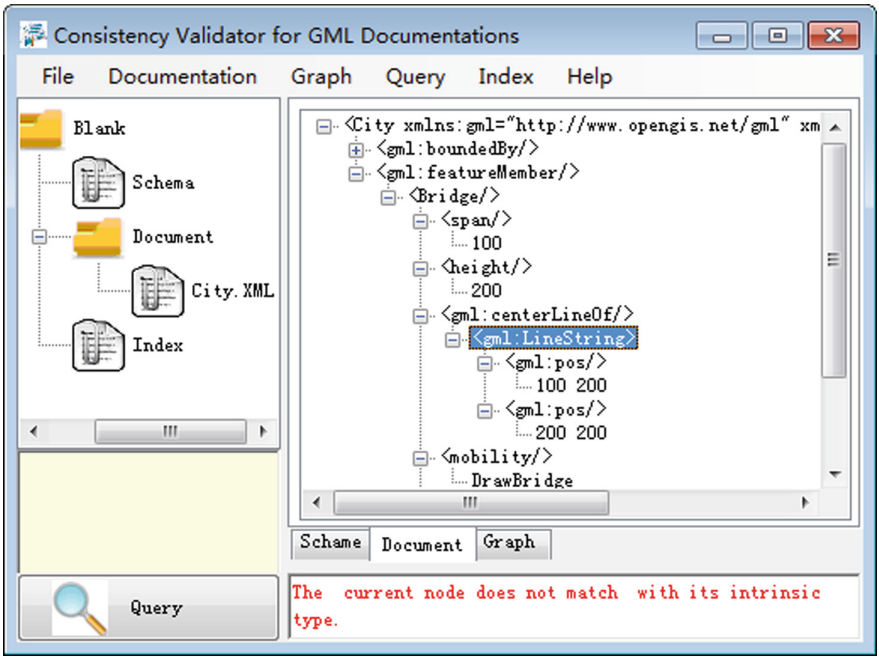

Fig. 2. Example test for a GML data segment file.

The test data comes from a document titled "City.XML", which is a GML data segment file describing the characteristics of the bridge element. Its content is shown as follows.

```
<gml:featureMember>
  <Bridge>
    <span>100</span>
    <height>200</height>
    <gml:centerLineOf>
      <gml:LineString>
        <gml:pos>100 200</gml:pos>
        <gml:pos>200 200</gml:pos>
      </gml:LineString>
    </gml:centerLineOf>
    <mobility>DrawBridge</mobility>
    <spans>  <Gorge/>  </spans>
  </Bridge>
</gml:featureMember>
```

A considerable amount of GML data instances were selected to verify the correctness and reliability of the program. The test results show that: The consistency verification algorithm can verify the consistency of GML data effectively, and the algorithm performance is higher. The detected errors are accurate and reliable [1].

6 Conclusion

DOM parsing technologies based on objects are adapted in the implementation of GML grammar validation, which is contained within GML schema parsing. An algorithm for GML data's consistency verification using recursion technique is built and implemented.

Experiments show that the algorithm relies on the integrative GML grammatical and semantic database, which can make a correct judgment about the GML data consistency, to resolve errors and eliminate hidden dangers effectively for GML application data. The algorithm has the relatively high accuracy and fast execution efficiency.

References

1. Gao, X.: Research on universal GML schema parsing based on grammatical and semantic database. Dissertation for The Degree in M.A.Sc (in Chinese), pp. 11–29 (2006)
2. Lake, R., Burggraf, D.S., Trninic, M., Rae, L.: Geography Mark-UP Language (GML). John Wiley & Sons Ltd, USA (2004)
3. Gao, X., Cui, Z., Jia, N., Xiao, H., Zhang, S.: Design and implementation of essential algorithms for parsing GML schemas. In: 6th International Conference on Intelligent Human-Machine Systems and Cybernetics, vol. 2, pp. 284–287. IEEE Computer Society CPS Press, USA (2014)
4. Lake, R.: The application of geography markup language (GML) to the geological sciences. J. Comput. Geosci. **31**, 1081–1094 (2005)

5. Gao, X., Li, H., Zhang, S., Sheng, Y.: Research on HRegExp applied to GML parsing. In: 5th International Symposium on Computational Intelligence and Design, vol.2, pp. 214–217. IEEE Computer Society CPS Press, USA (2012)
6. Walmsley, P.: Definitive XML Schema. Prentice Hall PTR, Upper Saddle River (2002)
7. Open Geospatial Consortium Inc.: Geographic information-Geography Markup Language Version3.3". http://www.opengeospatial.org/standards/gml
8. The World Wide Web Consortium: XML Technology. http://www.w3.org/standards/xml