GeReDiF: Using XML as a Structured Data Format in Grid Applications

Bita Gorji-Ara, Mohammad Hossein Reshadi, Mehdi Fakhraii
University of Tehran
Electrical and Computer Engineering Department
14399 Tehran, Iran
Phone: 98-21-800-9215; Fax 98-21-877-8690
Bita@cad.ece.ut.ac.ir, Reshadi@cad.ece.ut.ac.ir, Fakhraii@chamran.ut.ac.ir

Extended Abstract

Although distributed processing, in principle, provides speed up and access to high performance resources, it will be little used if people could not easily exploit it for their applications. In this way, many developers of grid applications have attempted to facilitate development of specific distributed applications using different mechanisms[1][2] such as designing generic libraries and frameworks. These frameworks use various formats for input, output and communication between nodes of the grid. Choosing appropriate format has major influence in the performance of the whole system as well as maintenance, revision and easy-to-use.

Every distributed framework imposes some overhead for initialization of a task and communication between nodes, especially in processes such as dynamic load balancing, which requires large amounts of data to be transferred and re-loaded. Using a structured format can reduce this overhead through decreasing the delay of loading data. In this project, we introduce a Generic Re-configurable Distributed Framework (GeReDiF), which utilizes benefits of XML[3] to define a structured data format for input, output and communications. We show that how XML can represent complex data structures such as graphs which can be loaded much quicker than any other formats.

The first standard version of XML was introduced in 1998 by World Wide Web Consortium (W3C). Like HTML, it is based on SGML (Standard Generic Markup Language), a strong language designed for storing very large structured documents. Tags and attributes in XML describe the structure and meaning of the data and in fact, XML is both data and document. XML grammar relies on regular expressions and hence its grammar is simple and its processing is fast. Using XML has other benefits:

- Since XML is a text format, it can be used in heterogeneous networks and it is portable to any other parallel platform.
- XML can represent the exact form of used data structure[4]. As it will be described, this feature can facilitate the maintenance of a distributed application.
- Designers can use their own set of tags, which are meaningful for themselves, and they are not limited to predefined keywords.
- Learning XML is not burdensome on the programmers, because XML is very simple and similar to HTML
- As a standard, XML is fully supported by the World Wide Web Consortium (W3C) and this support has led to many available free and documented tools with which developers can view, convert, load, save and check the validity of XML files.

Although XML has many good properties, it needs some modifications before it becomes useable in grid applications. For example: XML is originally designed to represent a text, which is tagged. While in our new application, we usually use tags rather than text. Therefore, one enhancement can be restriction of text with special begin and end delimiters. This enhancement makes XML compiler much simpler and quicker. Besides, new format is still a standard XML and is loadable in available free tools.

When a data structure is saved in a file, loading each node of data structure is an easy task, but setting the relations of the nodes (i.e. pointers between nodes) is a time consuming task and may require several searches in the node list. Although XML can represent a tree very well and it can be loaded without any extra search, most of the time, data structures are not as simple as a tree. To address this issue, first, the *spanning* tree[1][5] of the data structure should be represented in XML format and then the remaining links should be created by referencing existing nodes. To reference the nodes we have used a revision of XPath[6] standard, which is optimized for both saving and loading.

To customize the GeReDiF framework for a specified application the following steps are required:

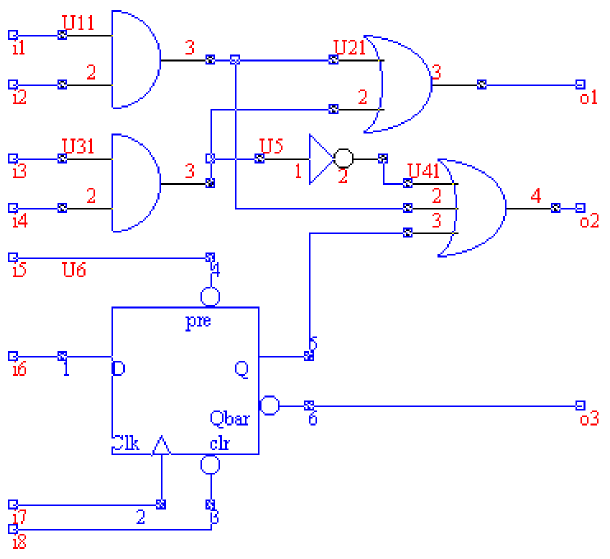
[1] *Spanning* tree of graph G is a tree, which covers all nodes of the G.

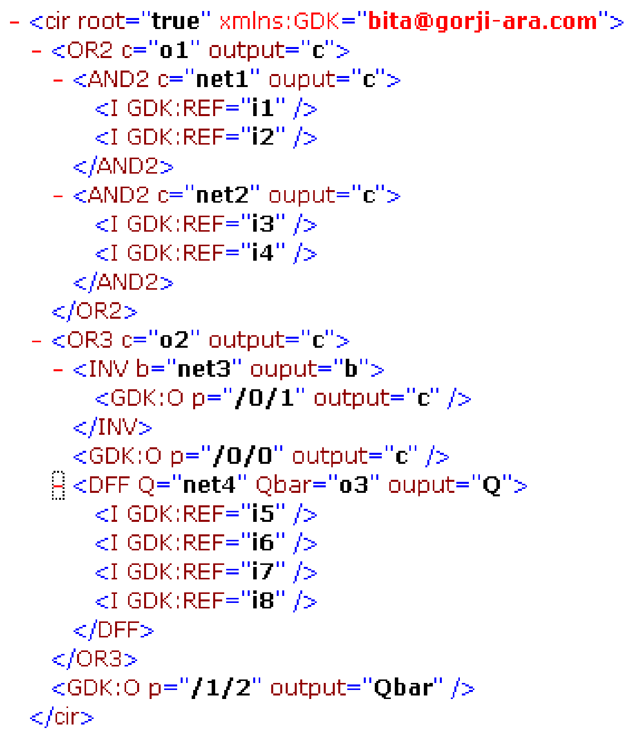
- The data structure of the sequential algorithm should provide means of representing itself in XML format.
- All classes in the sequential algorithm should inherit directly or indirectly from one of the basic classes, e.g. TGeneralNode, in the distributed library. There are some virtual methods in the basic classes that the inherited ones should implement or overload them to customize some features of the framework. Examples of the virtual methods are: XML loading, node addressing and task partitioning.
- There are some classes in the distributed framework that do not have a correspondence in the sequential algorithm but are necessary for customizing the framework. For example, the TCustomXMLDataBase provides a mapping between the classes and the XML tags used in the sequential algorithm.
- The framework should be re-compiled with new library.

One feature that makes a distributed application maintainable is providing a facility for applying the improvements and changes in the sequential algorithm to the corresponding distributed one. In GeReDiF, the classes of the sequential algorithm are directly used in the process and hence changes in the properties are automatically applied to the distributed version of the algorithm. On the other hand, changes in the relations of sequential nodes are transferred to the distributed algorithm through XML files.

GeReDiF framework has been designed and implemented and is going to be customized for two CAD algorithm (fault simulation and, match and cover algorithm for synthesis).

Following figures shows a circuit and corresponding XML file designed for fault simulation algorithm.




```
- <cir root="true" xmlns:GDK="bita@gorji-ara.com">
  - <OR2 c="o1" output="c">
    - <AND2 c="net1" ouput="c">
        <I GDK:REF="i1" />
        <I GDK:REF="i2" />
      </AND2>
    - <AND2 c="net2" ouput="c">
        <I GDK:REF="i3" />
        <I GDK:REF="i4" />
      </AND2>
    </OR2>
  - <OR3 c="o2" output="c">
    - <INV b="net3" ouput="b">
        <GDK:O p="/0/1" output="c" />
      </INV>
      <GDK:O p="/0/0" output="c" />
      <DFF Q="net4" Qbar="o3" ouput="Q">
        <I GDK:REF="i5" />
        <I GDK:REF="i6" />
        <I GDK:REF="i7" />
        <I GDK:REF="i8" />
      </DFF>
    </OR3>
    <GDK:O p="/1/2" output="Qbar" />
  </cir>
```

1. References

[1] J. A. Chandy, S. Parkes and P. Banerjee, “Distributed Object Oriented Data Structure and Algorithms for VLSI CAD”, Proceedings of International Workshop on Parallel Algorithms for Irregularly Structured Problems, Santa Barbara, CA, August 1996.

[2] Lan Foster, Carl Kesselman, *The Grid: Blueprint for a new computing Infrastructure*, ch. 8, Morgan Haufman publishers, 1999.

[3] C.F. Goldfarb, P. Prescod, *The XML Handbook*, ch. 53, second edition, Prentice Hall, 2000.

[4] M. H. Reshadi, B. Gorji-Ara, and Z. Navabi, “HDML: Compiled VHDL in XML”, IEEE VIUF, 2000

[5] J.A.Bondy and U.Murty, “Graph Theory with Applications”, American Elsevier Publishing Co., 1976.

[6] C.F. Goldfarb, P. Prescod, *The XML Handbook*, ch. 59, second edition, Prentice Hall, 2000.