SERBIAN JOURNAL OF ELECTRICAL ENGINEERING
Vol. 9, No. 1, February 2012, 29-32

UDK: 004.45:519.6 **DOI: 10.2298/SJEE1201029D**

Implementation of Visitor Pattern in Processing a Syntax Tree in Qlab Project*

Aleksandar Đenić[1], Miroslav Marić[1], Marko Mladenović[1], Srđan Božović[2], Miloš Netković[1]

Abstract: Qlab is an open-source project that supports various mathematical calculations, specialized for academic use. It has been developed at the Faculty of Mathematics, University of Belgrade, and is supported by Microsoft Serbia. In this paper we present some of Qlab's successfully implemented core solutions. More precisely, in our approach we use a specialized Visitor pattern to optimize the management of syntax tree commands that our parser sends to our engine. This allows the processing of a larger scale of tree implementation using the Visitor interface.

Keywords: Qlab, Visitor pattern, Syntax tree.

1 Introduction

The main idea behind the Qlab project [1] is to develop open source software which will support complex mathematical calculations. It's being developed by Faculty of Mathematics, University of Belgrade and it's supported by Microsoft Serbia. Developers are students under the leadership of faculty professors. They are divided into teams and in order to use full potential of the faculty, students from all sections and modules are included in the development.

Initial set of Qlab functionalities is based on those of MATLAB, but one of the main goals is to expand that set and to develop more comprehensive mathematical software which will be released under Open source license. In addition, Qlab adheres all MATLAB standards, so every program written in MATLAB will have no problems executing in Qlab, and vice versa.

2 Implementation

Qlab is developed on Microsoft .NET platform, language used is C#. Development environment is Microsoft Visual Studio and .NET Framework

[1]Faculty of Mathematics, University of Belgrade, Studentski Trg 16, 11000 Belgrade, Serbia;
E-mails: djenic@matf.bg.ac.rs; maricm@matf.bg.ac.rs; mladja87@gmail.com; milos.netkovic@gmail.com
[2]MFC-Mikrokomerc, Belgrade, Serbia; E-mail: srdjan.bozovic@gmail.com
*Award for the best paper presented in Section *Artificial Intelligence*, at Conference ETRAN 2011, June 6 – 9, Banja Vrućica – Teslić, Bosnia and Herzegovina.

version used is 4.0. Project is divided into 4 sub-projects: parser, basic mathematical operations, engine and graphical user interface (GUI). Each of them is being developed independently, with respect to standards of communication which were defined earlier. This kind of project organization secured modularity of the project and simplified it's organization. Also, it allows us to add new features and improvements in the future without modifying the entire project. More precisely, if new features or improvements are added to one of the sub-projects, only that one will be modified and, with respect to communication standards, the rest of the application will continue to work. This is a very important feature, especially if we take into consideration the fact that Qlab is an open source project and that anyone can add new features.

The process of execution goes as follows: a user defines functions and scripts he wants to use in the GUI. When a command is called or a function is executed, GUI passes the command to the engine. Engine parses the command and functions and scripts needed for the execution, and based on the syntax tree and basic mathematical operations executes the command. Result is then sent back to the GUI which displays it to the user.

For parsing we used Gardens Point Parser Generator, developed on Queensland University in Brisbane, Australia, which is also open source. It translates standard lex/yacc grammars to C# library which parses text. Parsed text is returned in the shape of a syntax tree.

3 Visitor Pattern

Visitor pattern [2] in object oriented programming is a way to separate algorithms from objects on which it operates. Result of this kind of separation is ability to add new operations on existing structures without changing the structures themselves.

Visitor pattern allows us to add new virtual function to an entire family of classes, without modifying the classes. Instead of adding code to every class, visitor class is created. That class implements all the virtual functions we need. Visitor takes an instance of a class as an argument and by using double dispatch method gets the result. Double dispatch method is a mechanism which passes a function call to concrete functions, based on types of objects involved. Object types are resolved in runtime.

4 Visitor Pattern in Qlab

Syntax tree which is the result of parsing is one of the most important structures in Qlab. Functions, scripts and commands are all being parsed. Syntax trees keep data on all parsed elements and they are the basis for all calculations in Qlab. They are also very important for analysis of parser testing, binary serialization of functions, analysis of programs. That said, it is necessary to implement more syntax tree traversals, different ones will be used for different actions.

```
ABSTRACT CLAS Statement : IVisited
METHODS:
            Accept(Visitor: IVisitor)

CLASS AssignmentStatement : Statement
ATTRIBUTES:
            LeftHandSide: Expression[]
            RightHandSide: Expression
METHODS:
            Accept(Visitor: IVisitor)
               Visitor.Visit(this)

CLASS WhileStatement : Statement
ATTRIBUTES:
            Condition: Expression
            LoopStatement: Statement
METHODS:
            Accept(Visitor: IVisitor)
               Visitor.Visit(this)

INTERFACE IVisitor
METHODS:
            Visit(as: AssignmentStatement)
            Visit(ws: WhileStatement)

CLASS EngineVisitor: IVisitor
METHODS:
            Visit(as: AssignmentStatement)
               Call Accept Method on RightHandSide
               FOREACH exp: Expression IN LeftHandSide
                  Assign Appropriate Value to exp
            Visit(ws: WhileStatement)
               WHILE Condition
                  Call Accept Method on LoopStatement
```

Fig. 1 – *Visitor pattrern in* QLab *engine*.

One of the ways to get more traversals of one tree is to implement a set of methods in every node, every method would be responsible for a different processing of the node. That solution is very bad, since with rise in number of methods, node loses it's basic functionality, being a node of a tree. Also, it leads to loss in project modularity, since engine adds functionalities in structures which belong to parser.

Best solution for these problems is to implement Visitor pattern on the entire syntax tree. Idea is to use node structure, and to make sure that every node has an Accept method which will, as an argument, have an instance of Visitor interface. Visitor needs to have a Visit method for every type of node in the syntax tree. Accept method of a node can now call Visit method of it's class. In that way, Visitor implementations are separated into different structures.

In Qlab, basic node of a syntax tree is an abstract class Statement, all concrete nodes are inheriting that class. Parts of implementation of concrete nodes, AssignmentStatement and WhileStatement, are shown in Fig. 1. Both nodes contain their own specific data and an Accept method. Accept method calls concrete implementation of Visit method of it's class. Ivisitor interface contains Visit methods for every node in the tree, in this case AssignmentStatement and WhileStatement. In the same example there is also a simplified version of EngineVisitor for program execution, which implements concrete Visit methods on tree nodes.

Adding any number of tree processings is fairly easy. All one needs to do is implement visitor interface for each new tree processing. That way, we can easily add a new engine or a new serialization.

4 Conclusion

Architecture of Qlab is modular, split in parts which are independent in both terms of logic and implementation. This sort of project organizations secures high quality code and easier improvements in the future. The development of these modules has been equally challenging as organizing students to use their knowledge and skills to contribute to the project and their future carriers. The visitor pattern was one very useful improvement that made Qlab more complete and represents the future development of Qlab – using advanced technologies making a modern all round mathematical software.

5 References

[1] M. Marić, A. Đenić, M. Mladenović: Razvoj „open sours“ aplikacija na univerzitetima, Qlab projekat Matematičkog fakulteta u Beogradu, Informatika 2010 „Novi trendovi u razvoju informacionih sistema“, Beograd, Srbija, 12. maj 2010, pp. 36 – 39. (In Serbian).

[2] T.A. Davis: MATLAB Primer, CRC Press, Boca Raton, Florida, USA, 2011.

[3] A.W. Troelsen: Pro C# 2010 and the .NET 4 Platform, NY, USA, Apress, 2010.

[4] J.R. Levine, T. Mason, D. Brown: Lex & Yacc, O'Reilly Media, Sebastopol, California, USA, 1992.

[5] B. Judith: C# 3.0 Design Patterns, O’Reilly Media, Sebastopol, California, USA, 2008.