

Compiler's development using UML and Code Generation

IGNACIO GONZÁLEZ ALONSO

Language and System's Area. Computer Science Department. Universidad de Oviedo. Campus de Mieres (s/n) Mieres. Asturias. Spain.

gonzalezaloinacio@uniovi.es

ABSTRACT. In this article we describe the process and architecture theory to develop a Compiler using UML and Code generation techniques. This approach consists of using UML Case tools that generates fully functional code, and use a framework for compiler's construction. It allows to draw compiler rules from lexical to semantic actions. The UML approach allows the reuse of lexical rules, syntactical rules and semantic actions.

Keywords : Compilers' construction, UML, Reuse, CASE, Code Generation, Framework, Grammars rules.

INTRODUCTION

Nowadays compiler's construction is based on grammar rules [1]. Tools are developed to process this rules, and hardly any part of a language compiler can be reuse to construct a new one.

This process is usually based on text-rules, which are hard to process in an academic environment, and that limits the possibilities of reusing parts of a grammar.

All of the development cycle is long and expensive, and not good covered with well test Software construction technology as UML [2] or Code Generation [3]. We should develop a new process, architecture and framework that consists in improve reuse, extensibility, reduce compilers production times, and achieve an easy way to learn how to create compilers.

At this point we should limit the scope of this article: this is a theoretical approach, no compiler was generated with this process yet. But and skeleton of UML classes and Secuence diagrams were created to prove the real possibilities of the model.

A key pattern, allowed by the creation of the framework development was inversion of control. The primary benefits of OO application frameworks stem from the modularity, reusability, extensibility, and inversion of control they provide to developers [4]. Inversion of Control make possible to us to draw everything, and to forget about grammar files that initialize something using other language different from UML. Every part of the compiler is developed with UML. There are some framework works done already [5] and [6]

The transformation from UML to Code (in this time: Java Code) could be made with any CASE tool that can generate the Java source. But no limits to target language are established by the model, or by any Code Generation approach. Simple code generation approach is supposed, but MDA[7] could perfectly fit on the process.

MATERIALS AND METHODS

The proposed process consists on the next steps:

- 1.-The creation of a framework for developing compilers. All the constructors are written with IoC pattern. (Fig. 1)
2. - Creation of classes for our compiler. This class inherits from the frameworks ones. (Fig. 2, Fig. 3, Fig. 4, Fig. 5)
3. - Drawing the sequence diagrams with the instances from classes and to load the IoC constructors. (Fig.6, Fig. 7, Fig.8)
4. - Run the code generation tool of the CASE tool to transform the UML model to code. The code generated is not limited by definition, only by the CASE tool.
5. - As any other software, here you should test it.

RESULTS

On the next pages you will see the results of applying this process: the framework (in UML notation), the concrete classes, and the sequence diagrams.

We no annex code because extension of the article. But you can generate it with any CASE tool that supports code generation for Class diagrams and for Sequence diagrams (this last ones creates the calls and the instance objects).

Figure 2

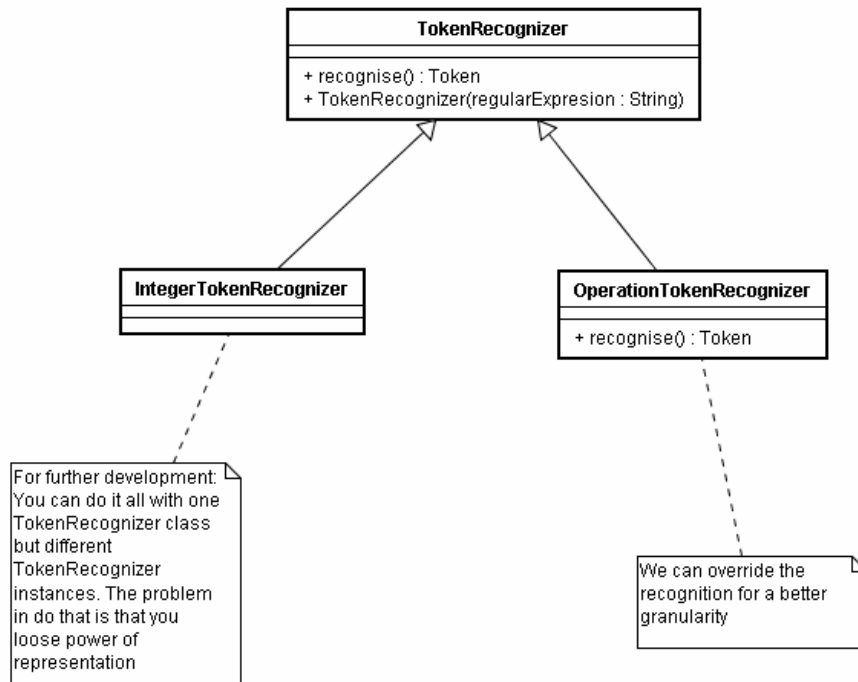


Figure 3



Figure 4

SemanticRule
- syntacticRule : SyntacticRule - action : Action
+ SemanticRule(syntacticRule : SyntacticRule, action : Action) + run() : void

Figure 5

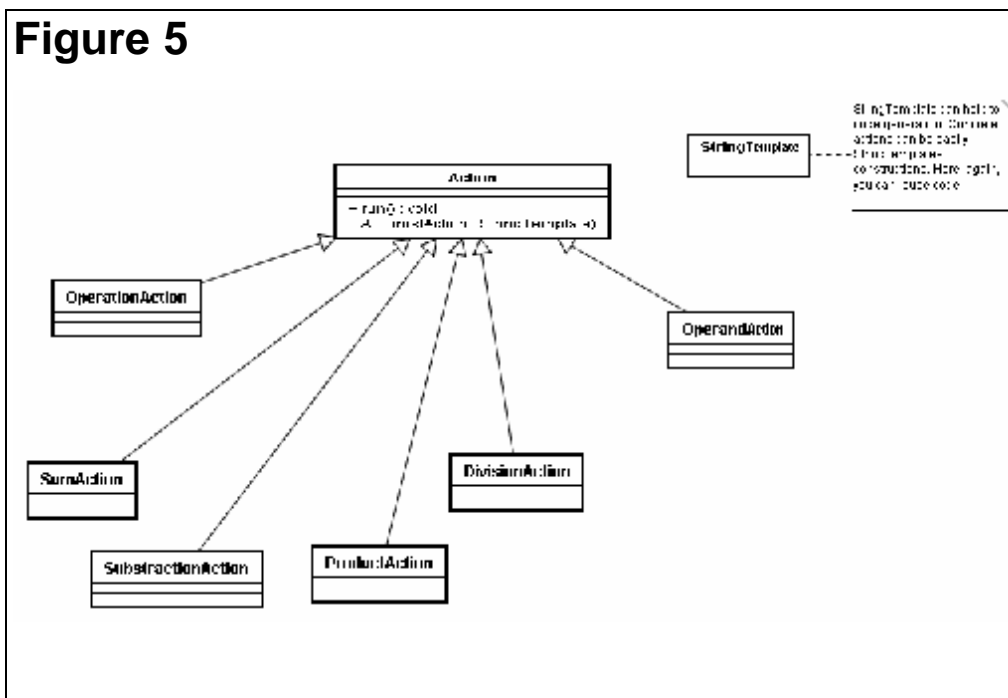


Figure 6

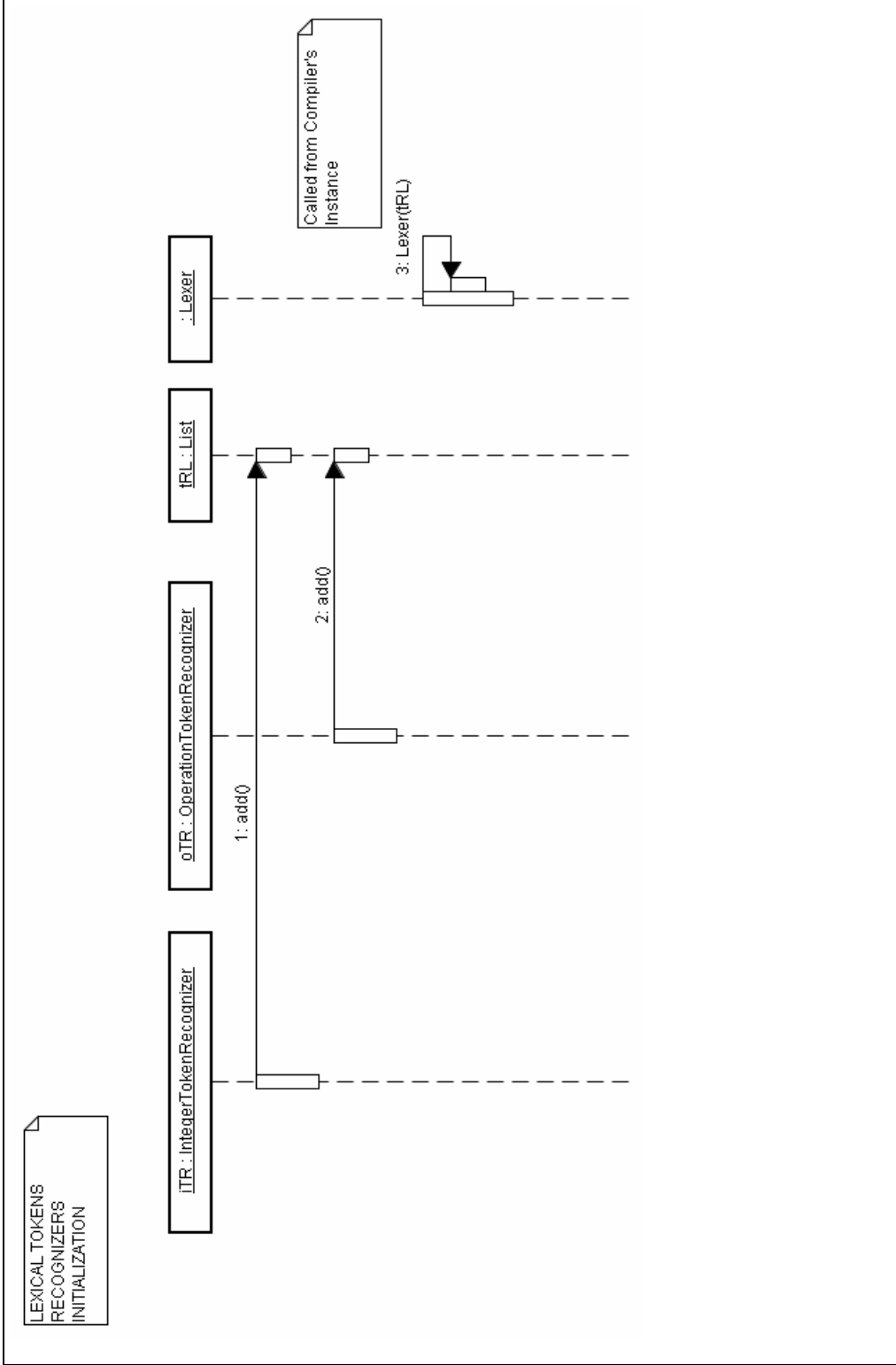
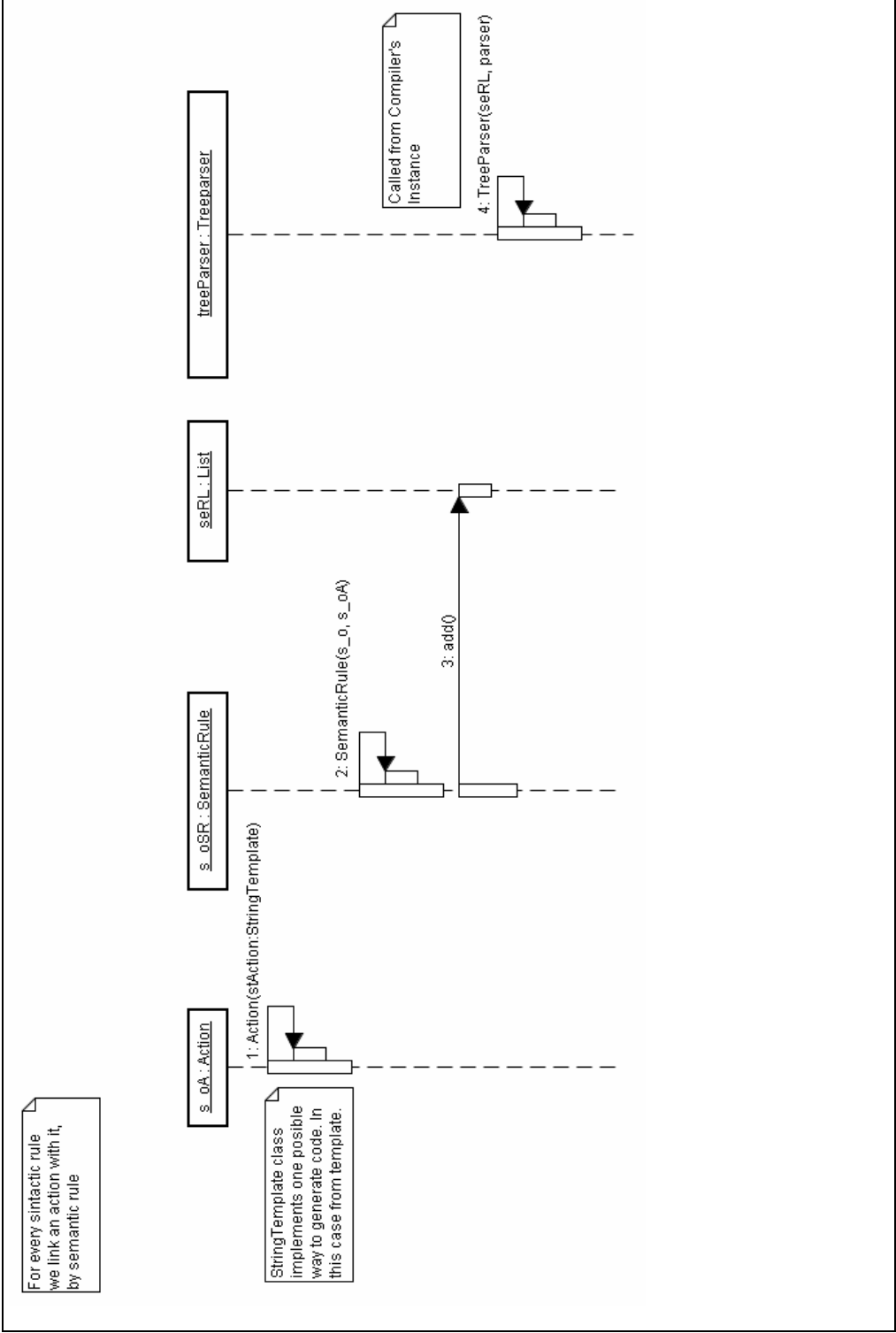


Figure 8



Developing with UML is a well-known process so compilers construction with this process is feasible, and now this process simplify it.

Developing a compiler skeleton with this process was clear, straightforward and inexpensive in time comparison from other long cycle compiler developing process. But a more in deep analysis should be done here.

From learning and academic point of view, this is a visual way to create compilers. UML CASE tools are visual tools, so with this process you substitute grammar rule writing by grammar rule painting.

We also found some disadvantages of this approach that will mention in the next lines.

The first one is produced with the new way of express grammar rules. UML notation can be tedious if your CASE tool is not a very usable one. So you have to repeat some tasks first times. After your first compiler or with the help of others you can avoid a lot of this, just reusing previous UML developed rules.

The second disadvantage is the need of a good UML design, as much complete as you can. When you are trying to create source code from a model you should be as explicit as you can. In some ways, UML is not complete enough to programming so perhaps if actions can not be describe with a template, perhaps you have to write them outside UML. Nowadays, MDA is trying to solve such question [8]. Further work should ensure that any compilers known till present can be developed with a template mechanism. This is a further investigation fork.

A natural consequence of this process will be the accumulation of rule classes inherited from framework. This accumulation will facilitate the development by reusing UML rules. It's very important to notice that reusing a lot of UML painted classes is easier than the reuse of grammar rules, because, UML is much more restrictive about the way we can write rules, than a formal grammar based on regular expressions.

DISCUSSION

After the development of our first theoretical compiler with this new process we consider proved the feasibility of a compiler visual perspective developing process.

It allows reuse of compilers grammars and actions. It allows, also, to improve the Domain Specific Languages construction and to expose compilers construction knowledge to a wider public.

Also this process simplifies teaching and transmitting information about concrete compilers because is written in a well-known and standard language as UML is. The code generation techniques are placed here to reduce costs. This techniques has the common limits of Code Generation(high level Code Generation). You should carefully choose your CASE Code Generation tool, and don't be afraid in a real development case to extend the code generated with your own code. More interesting implications will came if we see the process as an XML transformation, but this is enough material for a new article, that is now under development.

LITERATURE CITED

- [1] Compiler's construction based on grammar classic reference.
- [2] Booch, G. Object-Oriented Analysis and Design, with Applications (2nd Edn.) Benajmin/Cummings, 1994
- [3] CODE GENERATION classic reference
- [4] M. Fayad and D. Schmidt. Object-oriented application frameworks. Communications of the ACM, 40(10), October 1997.
- [5] David Basanta, M. Cándida Luengo, Raul Izquierdo, Jose E. Labra, J. Manuel Cueva: „Improving the Quality of Compiler Construction with Object-Oriented Techniques” ACM SIGPLAN, Vol. 35, No. 12, pp. 41-51, December 2000.
- [6] Carroll, Steven. A Framework for Incremental Extensible Compiler Construction. International Journal of Parallel Programming, Vol 32, No. 4, August 2004
- [7] Stephen J. Mellor, Marc J.Balcer „Execute UML” Addison-Wesley Professional; 1st edition May 15, 2002
- [8] Object Management Group, *Model Driven Architecture (MDA)*, OMG Document ormsc/2001-07-01 edition, July 2001.