Francisco J. Almeida Martínez
Jaime Urquiza Fuentes

Teaching LL(1) parsers with VAST - A usability evaluation

Número 2009-01

Serie de Informes Técnicos DLSI1-URJC
ISSN 1988-8074
Departamento de Lenguajes y Sistemas Informáticos I
Universidad Rey Juan Carlos

Índice

1. Introduction…………………………………………………….. 1

2. Description of the evaluation.…………………………………….. 2
2.1. Subjects ………………………………………………………………… 3
2.2. Experimental Design………………………………………………………… 4
2.3. Tasks …………………………………….……………………………… 4
2.4. Protocol ………………………………………………………………… 5

3. Results of the Evaluation…..…………………………………….. 6
3.1. Instructors' observations…………………………………………………… 6
3.2. Answers to the questionnaire………………………………………………… 7

4. Conclusions.………………………………………………….. 8

5. Acknowledgments…………………..………………………….. 9

References ………………………………………………………... 9

Apendix: ……………..…………………..…………….……….... 10
A. Knowledge Test ………..…………………………………………..………… 10
B. Student's Opinion Questionnaire about VAST ……….………..………… 11
C. Student's Opinion Questionnaire about ANTLRWorks ……..……………… 12

Teaching LL(1) parsers with VAST
A usability evaluation

Francisco J. Almeida-Martínez and Jaime Urquiza-Fuentes

LITE – Laboratory of Information Technologies in Education
Technical Superior School of Computer Science Enginnering
Rey Juan Carlos University
{fancisco.almeida,jaime.urquiza}@urjc.es

Abstract. VAST is an educational tool to be used in compiler and language processing courses. It is focused on syntax trees visualization. Its main aim is to be as much independent from the parser generator as possible. Allowing the visualization and animation of syntax trees produced by self developed parser. We have evaluated the usability of VAST in a comparative evaluation with the ANTLRWorks tool. In this report we detail the subjects, method, immediate results and conclusions of this evaluation.

1 Introduction

VAST[1] is an educational tool to be used in compiler and language processing courses. The current version allows visualizing syntax trees and their construction process. The main advantages of VAST follow: it is designed to be as independent from the parser generator as possible, it allows students to visualize the behaviour of parsers they develop, and it has an interface designed to easily handle huge syntax trees.

VAST offers an API –VASTapi– designed to be used when the parser is building the ST, and a graphical interface –VASTview– to visualize the ST generated. First, the user annotates the syntax specification, then generates the parser with any parser generator, and visualizes the syntax tree when the parser is executed, Figure 1 summarizes this process.

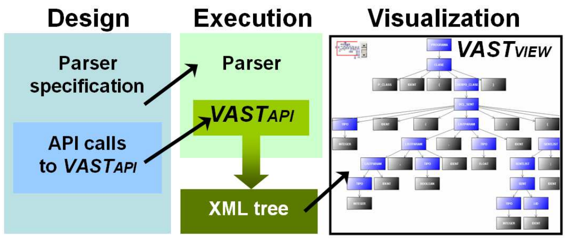
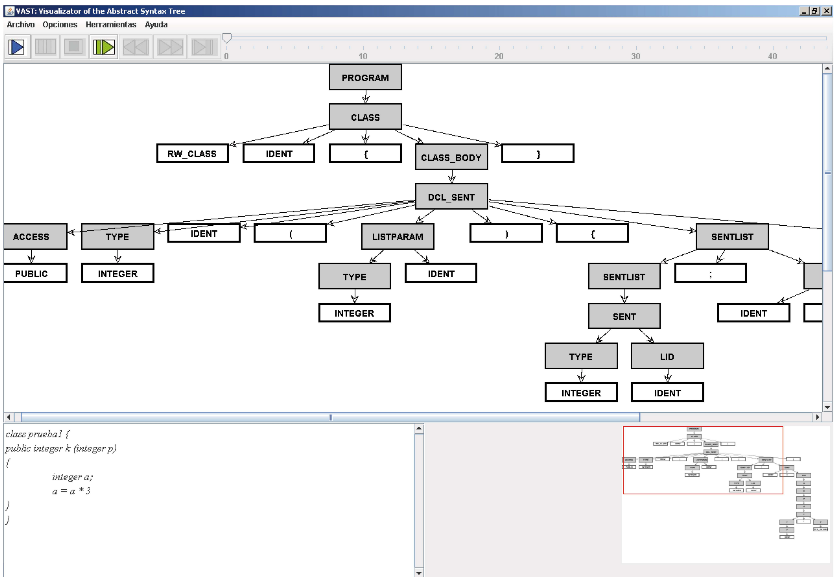


Fig. 1. Use and structure of VAST

We have designed VASTApi as simple as possible close to the concepts that are being visualized. VASTview is a visualization interface specially designed to show syntax trees, see Figure 2. It also shows synchronized views of the input stream and the stack, the later is still under development. As these trees can be huge we have added special visualization features as global/detailed views and a zoom facility. Finally we allow the users to change the configuration of the graphical representations.



```
class prueba1 {
public integer k (integer p)
{
        integer a;
        a = a * 3
}
}
```

Fig. 2. A snapshot of the VASTview interface

This report describes a usability evaluation of VAST. It has been carried out within a language processing course, while teaching the LL(1) parsers.

The rest of the report is organized as follows. In section 2 we describe the evaluation, the subject, the method and the protocol. In section 3 we describe the results of this evaluation in terms of instructors' observations and students' answers to a questionnaire. Finally, we draw our conclusions in section 2.

2 Description of the Evaluation

In this section we give a detailed description of the evaluation. We describe the students who participated in the evaluation, the experimental design of the evaluation, the protocol and the tasks performed during the treatment.

2.1 Subjects

59 students participated in the evaluation. They were enrolled in a language processors course of the fourth year of a (five years) BSc in Computer Science at the Universidad Rey Juan Carlos (http://www.urjc.es). The participation was almost voluntary. We say *almost* because it was incentive based with an increment of 2% in the final grade, only if they pass the exam. Gender distribution is unbalanced, 5.4% (3/59) of the students were female.

They completed a learning style test [2], see the distribution of the learning styles in Figure 3.

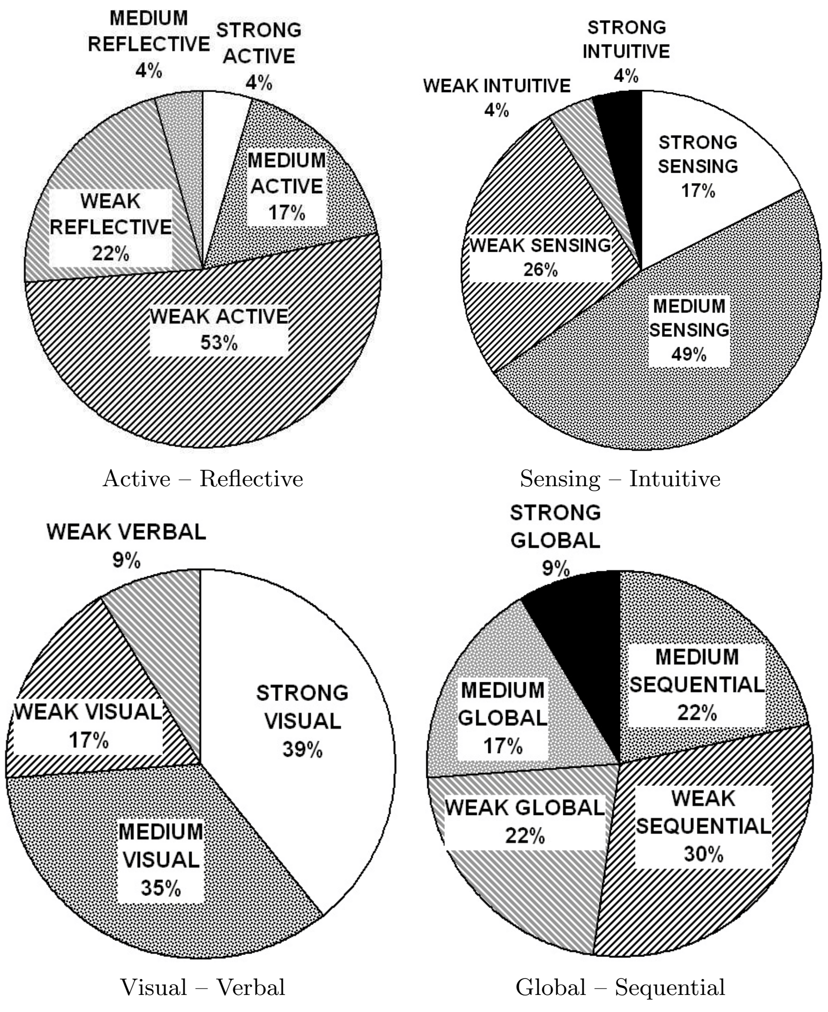


Fig. 3. Distribution of the learning styles within the subjects

2.2 Experimental Design

This evaluation has been designed as a controlled experiment plus an observational study [3]. As we are evaluating an educational tool we have used some guidelines of true experimental studies [4]. It is a controlled evaluation because we divided the subjects in two groups the treatment group –which worked with the tool VAST– and the control group –which worked with the tool ANTLRWorks[1]–. Group design was partially randomized. Both groups were balanced as much as possible using the pretest (see appendix A) grades. Students were classified by their pretest grades but the selection of the group (treatment/control) for each student within these classes was randomized.

The independent variable is the tool used: VAST(treatment) or ANTLRWorks(control). The dependent variables collect students' opinion about four different aspects. The data has been collected using two different questionnaires depending on the group: treatment (appendix B) and control (appendix C). Table 1 maps aspects and questions in the questionnaires.

Table 1. Questionnaire of students' opinion

Students' opinion about	Questions in treatment questionnaire	Questions in control questionnaire
Ease of use	1,2,5	1,4
Learning improvement	3	2
Quality of the tool	4,5	3,4
Student's satisfaction	6,7	5
Other aspects	8-11	6-9

2.3 Tasks

All the tasks completed by the students have to be documented using visualizations and textual explanations. The tasks were three exercises of LL(1) parsing concepts. First we explain how students have to generate the visualizations depending on the group they belong to.

Control Students have to code the grammar with the grammar editor of ANTLRWorks, if they want to see static visualizations of the syntax trees generated by their parser, they only need to use the interpreter. But if they want to see the animation of the process then they have to use the debugger facility. Both the interpreter and the debugger ask the user to specify the input stream, the starting rule and the line ending platform.

[1] http://www.antlr.org/works/index.html, 2009

Treatment Students have to code the syntax specification using a general purpose editor. Then they annotate the specification using simple VASTApi calls. Next they generate the parser using ANTLR with the annotated specification and compile the generated parser. Each execution of the parser –using the console– produces the visualization that can be viewed with VASTview.

In the first exercise students were asked to change a grammar of arithmetic expressions so existing precedences of binary operators were changed. The original grammar was:

```
S := F N
N ::= + F N | - F N | * F N | / F N | λ
F ::= id | cte | ( S )
```

The new grammar must give the highest precedence to `*` and `/` operators –both the same–, then a medium precedence to the `-` operator and the lowest precedence to the `+` operator. And all binary operators must be right-associative.

The solution to this exercise is a grammar together with: a screenshot (or a group of them) showing the existing precedences, the input stream(s) used to produce the visualization(s) and a textual explanation of the solution.

In the second exercise, students were asked to generate two inputs with syntax errors produced by problems with: the *starters* symbols and the expected terminals inside right parts of the grammar rules. Again, the solution to both cases is made up of the input stream, the visualization and the textual explanation.

Finally, in the third exercise students were asked about syntax error recovery situations with the panic mode strategy. For each non terminal symbol `X` in the grammar, students have to generate an erroneous input stream were the group of terminals in `FOLLOW(X)` were used as the synch terminals. Again, the solution is made up of the input streams, the visualizations and the textual explanations.

2.4 Protocol

This section describes the protocol of the evaluation, the order of the different activities performed by students. Table 2 summarizes this protocol.

Two weeks before the experiment, students completed the pretest (see appendix A). Then we formed the control and treatment groups.

Note that VAST needs a parser generation tool. Therefore we ensured a balanced knowledge of the parser generation phase using ANTLR[2]. One week before the experiment, students had a lab session with exercises about the ANTLR parser generator so they were familiar with ANTLR syntax for grammar specifications and their generation process.

The experiment was two hours long. First, instructors gave a tutorial about the corresponding visualization tool –ANTLRWorks and VAST–. Then, the students worked with the exercises. At the end of the session, students completed the opinion questionnaire. Students had one day to solve the problems, thus they sent the documents with their solutions to the teachers.

[2] http://www.antlr.org/, 2009

Treatment group	**Control group**
Knowledge pretest	
ANTLR Lab session	
ANTLR/VAST tutorial	ANTLRWorks tutorial
Execises	
Questionnaire	

Table 2. Protocol of the evaluation

3 Results of the Evaluation

In this section we detail the results obtained from the experiment. During the experiment we observed how students worked with the tools. Therefore results can be divided in instructors' observations and answers to the questionnaire.

3.1 Instructors' observations

The instructors observed how students used the tools during the experiment, detected bugs, found problems and, valuable and useless features.

Control group The control group used the ANTLRWorks environment. This environment encapsulates under the same interface the syntax specification editor and the generation of visualizations.

Students liked the syntax specification editor and how it graphically differentiates between lexical and syntax rules. They detected some possible bugs as: file paths with blank spaces and the use of multi-platform end of lines, in this last case students using a Mac had to select Windows version for the end of lines.

The debugger of ANTLRWorks need a specific free port in the computer. Students found problems because some of the computers in the lab didn't have this port free. Neither students nor instructors could change computers port configuration, and there was not enough time to find other ports using a trial-error process.

Finally, students got confused with some aspects: choosing the end of line platform, choosing the starting symbol and why there were two ways of generating visualizations, the interpreter and the debugger.

Treatment group The treatment group used the ANTLR tool to generate the parser and VAST to generate and view the visualizations. Students used standard editors to code the syntax specifications and annotate it with calls to VASTApi. Finally, they used two tools to compile/execute the parser, some of them used `javac/java` in the console while others used the Eclipse IDE[3].

[3] http://www.eclipse.org/, 2009

Students liked the animations produced with VAST –they payed special attention to the construction order of the syntax tree– and experimented with different graphical configurations. They detected one small bug in the tree drawing algorithm.

Students found an important problem during compilation. Unfortunately, `jdk` configuration was obsolete, thus ANTLR specifications were compiled correctly but VASTApi calls were not. Fixing this problem was hard for both students and instructors because the last `jdk` was installed but not correctly specificated in the `PATH` environment variable. Some students decided to use the Eclipse IDE, others didn't found this problem. Consequently, a number of students felt frustrated with the tool and gave up while others continued working after the lab session.

Other problem observed by the instructor was the number of simultaneously open windows: the editor, the command console, the compilation window –Eclipse IDE or another console– and the VASTview window. Note that students didn't identify this situation as a problem but the instructor observed that this issue should be improved. Finally, we detected two anecdotal problems with VASTApi calls: the `setRoot` and `GenerateXML` methods.

3.2 Answers to the questionnaire

The problems mentioned before prevented some students from completing exercises and the corresponding questionnaire. Thus, the total number of students was 48, 22 in the treatment and 26 in the control group.

All the questions about opinion or quality have to be answered using a Likert scale with five values being: 1 the worst opinion or quality, 3 no opinion or medium quality, and 5 the best opinion or quality. Table 3 shows average results for each dependent variable and a significant differences analysis.

We put some open questions in the questionnaire, thus students had more freedom to explain their opinion about the tools. We asked students to identify which parts of the tools were specially difficult to use, questions 1,2 in the treatment group and question 1 in the control group. While the treatment group identified the compiling process –obviously caused by problems with `jdk`–, the control group identified the debugger together with the error messages of the tool.

Questions 4(treatment) and 3(control) asked students about the quality of the tool. Both groups highlighted the visualization of the tree as a high quality feature. In addition, the treatment group identified the animation of the tree construction process while the control group identified the grammar editor as high quality features. Low quality features were found as well. The treatment group identified, thought not significantly, classpath problems –again related with the `jdk` problem–, while the control group identified the debugger, obviously because some of them could not use it.

Questions 8(treatment) and 6(control) asked students about useful features that should be added to the tools. The treatment group identified the integration of the different tools used –editor, console, parser generator, parser execution,

Table 3. Questionnaire of students' opinion

Students' opinion about	Average grade treatment group	Average grade control group	Significant differences
Ease of use			
General	4.18 (VASTview)	4.36	p > .05
	3.15 (VASTApi)		**p < .05**
Average of parts	4.12	3.80	p > .05
Learning improvement			
Construction process	4.14	4.26	p > .05
Input stream	3.62	3.72	p > .05
Stack	3.46	2.38	**p < .05**
Quality of the tool			
General	3.69	4.18	p > .05
Average of parts	4,11	3,79	p > .05
Student's satisfaction	4.18 (VASTview)	4.09	p > .05
	3.45 (VASTApi)		**p < .05**

visualization– and the possibility of resize windows of VASTview. The control group identified the synchronization between the grammar editor and the error messages together with the animations of the construction of the syntax tree. On the contrary, none of the groups identified significantly any useless features to remove from the tools.

Students were asked for positive aspects in questions 10(treatment) and 8(control). The treatment group identified the visualization of the syntax tree and the animation of its construction process together with the simplicity of the VASTApi methods. The control group identified the grammar editor and the platform independence.

Finally, students were asked for negative aspects in questions 11(treatment) and 9(control). The treatment group identified the fact that windows of VASTview were not resizable and the compilation process. The control group clearly identified the debugger as the most negative aspect. Other aspect identified, but less important than the debugger, was the language of the tool.

4 Conclusions

The results for VASTview are positive regarding the dependent variables: ease of use, learning improvement, quality of the tool and student's satisfaction. Also the instructors observed that students liked the visualization and animation capabilities of VASTview.

ANTLRWorks obtained similar results. Although students got confused due to some professional features as choosing the end of line platform and the starting grammar rule, or having two different visualization tools.

Students' opinion about VASTApi is two-fold. On the one hand, students think that the API is quite simple. On the other hand, the grades for ease of use and students' satisfaction are worse than those for VASTview and ANTLRWorks.

We feel that we should redesign the way that students build the visualizations. We have detected that the generation process of visualizations is made up of many separate steps: grammar edition, grammar annotation, parser generation, parser compilation, input stream edition, parser execution and visualization. This situation should not be a problem for the teachers, but many separate tasks could be a source of students' errors and mistakes.

We plan a global integration, thus parser visualizations will adapt to the typical parser development process of specification, generation and execution. It will be based on two functional integrations:

annotation-generation-compilation it will automatically annotate grammar specifications, generate the parser source code and compile it. The requirement of parser generator independence is affected by the automatic annotation as it will be reached with specific developments for each parser generator. From the students' point of view, making the annotation step transparent to students is much more important than loosing parser generator independence. From the teachers' point of view, we keep independent from the parser generator, because they can still annotate manually parser specifications.

execution-visualization it will allow students to edit the input stream, execute the parser and visualize the ST using the same interface.

5 Acknowledgments

This work was supported by project TIN2008-04103/TSI of the Spanish Ministry of Science and Innovation.

References

1. Almeida-Martínez, F.J., Urquiza-Fuentes, J., Ángel Velázquez-Iturbide, J.: Vast: visualization of abstract syntax trees within language processors courses. In: SoftVis '08: Proceedings of the 4th ACM symposium on Software visuallization, New York, NY, USA, ACM (2008) 209–210
2. Felder, R., Silverman, L.: Learning and teaching styles in engineering education. Engr. Education **78**(7) (1988) 674–681
3. Kulyk, O., Kosara, R., Urquiza-Fuentes, J., Wassink, I.: Human-Centered Aspects. In: Human-Centered Visualization Environments. Springer-Verlag (2007) 13–75
4. Cohen, L., Manion, L., Morrison, K.: Research Methods in Education. Fith edition edn. Routledge Falmer, NY, New York, USA (2001)

A Knowledge Test

1. What are the main features of LL(1) parsers?
2. What is a derivation?
3. Define the *panic mode* error recovery strategy.
4. Given the following grammar:
   ```
   S ::= F S'
   S' ::= + F S' | - F S' | λ
   F ::= id | num | ( S )
   ```
   Give cases of an LL(1) non recursive parser detailing the state of the input stream and the stack before and after the following operations:
   (a) A derivation
   (b) Token recognition in the input stream
   (c) An error recovery using FOLLOW(current non-terminal being derived) as the synchronization tokens.
5. Given the previous grammar, its corresponding non recursive LL(1) parser built with a *panic mode* error recovery strategy using FOLLOW(current non-terminal being derived) as the synchronization tokens and the following parser state: `($ S' ) S , id num num ) - num $)`.
   Detail the next steps of the parser until the stack has just the symbol `S'`.
6. Design an LL(1) grammar for logical expressions with the following features:
   (a) The operators are: the parenthesis, the unary operator `not`, and the right-associative binary operators `xor`, `and`, `or`, `nand` y `nor`. Precedence of the operators is given in the following table:

( )	Highest precedence
not	
and nor	Both the same precedence
or nand	Both the same precedence
xor	Lowest precedence

   (b) Both, precedence and associativity, must be integrated in the grammar. Thus, if an operator O1 has greater precedence than an operator O2 then O1 and its arguments are processed by the parser before O2 and its arguments.
   (c) The arguments of the operators are the locical constants `true` and `false`.
   (d) Some examples of correct streams:
   ```
   not true nor false and false
   false or true nand not ( true xor not false )
   ```
7. The following grammar is a proposed solution to the previous exercise:
   ```
   S ::= A S' | ( S )
   S' ::= xor A S' | λ
   A ::= B A'
   A' ::= or B A' | nor B A' | λ
   B ::= C B'
   B' ::= and C B' | nand C B' | λ
   C ::= not D | D
   D ::= true | false
   ```
   But it could be incorrect, therefore:

(a) Define criteria to evaluate the goodness of the solution proposed.
(b) Evaluate your solution and the proposed one using the previous criteria.

B Student's Opinion Questionnaire about VAST

1. In my opinion, VASTapi is easy to use: [ ]
   Identify which parts, if any, are more difficult to use:

2. In my opinion, VASTview is easy to use: [ ]
   Identify which parts, if any, are more difficult to use:

3. In my opinion, VAST has helped me to understand how LL(1) parsers work regarding:
   The construction process of the syntax tree: [ ]
   How the input stream is processed: [ ]
   How the stack of pareser works: [ ]

4. In my opinion, the general quality of VAST to show how LL(1) parsers work is high: [ ]
   Identify which parts are the best:
   Identify which parts are the worst:

5. Please, grade the ease of use and the quality of the different parts of VAST:

	Ease of use	Quality
Structure of the main menu		
Icons		
Animation controls		
Global view/Navigation		
Expand/Collapse syntax tree		
Zoom		
Input stream view		
Stack view		
Configuration aspects (colors, lines, ...)		
Configuration management		

6. In my opinion, I like VASTapi: [ ]

7. In my opinion, I like VASTview: [ ]

8. Identify which useful features are not present in VAST(VASTapi and/or VASTview):

9. Identify which features present in VAST(VASTapi and/or VASTview) that are useless enough to be removed :

10. Identify **positive** aspects of VAST(VASTApi and/or VASTview):

11. Identify **negative** aspects of VAST(VASTApi and/or VASTview):

C Student's Opinion Questionnaire about ANTLRWorks

1. In my opinion, ANTLRWorks is easy to use: [ ]
   Identify which parts, if any, are more difficult to use:

2. In my opinion, ANTLRWorks has helped me to understand how LL(1) parsers work regarding:
   The construction process of the syntax tree: [ ]
   How the input stream is processed: [ ]
   How the stack of pareser works: [ ]

3. In my opinion, the general quality of ANTLRWorks to show how LL(1) parsers work is high: [ ]
   Identify which parts are the best:
   Identify which parts are the worst:

4. Please, grade the ease of use and the quality of the different parts of ANTLRWorks:

	Ease of use	Quality
Grammar editor		
Interpreter		
Input stream view		
Syntax tree view		
Debugger		
Input stream view		
Syntax tree view		
Stack view		
Events view		
Debugging controls		

5. In my opinion, I like ANTLRWorks: [ ]

6. Identify which useful features are not present in ANTLRWorks:

7. Identify which features present in ANTLRWorks that are useless enough to be removed :

8. Identify **positive** aspects of ANTLRWorks:

9. Identify **negative** aspects of ANTLRWorks: