# Construction of a Flexible Data Structures Laboratory

**Jorge Villalobos, Danilo Pérez, Juan Castro, Camilo Jiménez**
Universidad de los Andes, Departamento de Ingeniería de Sistemas y Computación
Bogotá, Colombia
{jvillalo, danil-pe, davi-cas, camil-ji}@uniandes.edu.co

**Abstract**

In a computer science curriculum, the data structures course is considered fundamental. In that course, students must generate the ability to design the more suitable data structures for a problem solution. They must also write an efficient algorithm in order to solve the problem. Students must understand that there are different types of data structures, each of them with associated algorithms of different complexity. A data structures laboratory is a set of computational tools that helps students in the experimentation with the concepts introduced in the course. The main objective of this experimentation is to generate the student's needed abilities for manipulating complex data structures. This paper presents the main characteristics of the laboratory built as a support of the course. We illustrate the huge possibilities of the tool with an example.

**Keywords:** Data structures, educational tools, algorithm visualization, algorithm animation, active learning.

## 1. INTRODUCTION

Over the past ten years, the computer science curriculum has evolved quickly as a result of the amount of new concepts, requirements and emerging technologies. At this point, the courses are getting more crammed due to the teacher's fear of leaving out important topics. The companies are demanding more from our graduated students in terms of their competences and their abilities with new technologies, methodologies and tools that the market requires. Furthermore, there is less time to teach what was taught before. Moreover, it has to be pointed out that, as Hogason mentions in [1], it's not only an increase in the number of concepts that a student must handle in a course; there is an increase in the complexity as well. In some cases, this statement has led us to remove complete topics from the courses (or even complete courses) that were considered critical. In other cases, all of the subjects are included on a superficial way in order to give some room for the new ones that have been appearing. This is really concerning in the courses where, apart from generating knowledge, the intention is to generate skills on the students. As a consequence, the challenge we face is to try to produce the necessary abilities on the students in less time, taking advantage of the new information technologies, and the great capability they have in terms of understanding and manipulating visual elements.

In a classic computer science course like Data Structures, there is usually an emphasis in three aspects: first, the presentation of the many existing data structures with their intrinsic properties, classified by type (linear, recursive, etc.). Second, the generation of the abilities required to build the algorithms that manipulate these structures efficiently. And finally, the course looks forward to give the student the capabilities to follow a methodology to design (define and justify) the suitable data structures to solve a given problem. In consequence, this course has been considered strategic in the computer science curriculum for more than 25 years, since it allows a connection between the programming courses and the courses from the software engineering branch.

Being aware of the important role this course plays in the formation of our students, our group at Los Andes University wrote and published the books we considered appropriate to support this learning process [9] [10]. These books have been used over the past ten years by more than 30 Latin American universities. The challenge now, is to make that learning process more efficient, guaranteeing the proper generation of abilities by the student. This is the main goal of the project in which the work presented in this article is developed. Besides a new version of the book, we want to build a set of computational tools to support the learning process. We name this set of tools a "laboratory of data structures", hence making the parallel with the laboratories used by the students from other engineering programs to practice the concepts seen in class. This way, the book and the laboratory would complement each other, forming a solid support for the teachers and students of the course.

In a data structures laboratory, a student should be able to do the following:

- Animate an algorithm. For example, the student should be able to graphically follow the insertion process of an element in an AVL tree or an ordered list.
- Make an experiment about the efficiency of different algorithms used to solve the same problem on a data structure. The student should be able to set up the way the data for the tests is generated (e.g. by increasing the number of elements of the structure, or by changing its distribution). Then, he should be able to execute each of the different algorithms. And finally, he should be able to visualize the results from different points of view (e.g. making graphics of the time each algorithm spent over each data size).
- Visualize an algorithm written by the student. For example, if the student writes a sorting algorithm for a vector, the laboratory should allow him to visualize it in runtime, showing the way the structure is changing and the relation of these changes with the written code.

In addition, this laboratory should be completely configurable, extensible and dynamic so that it could be adapted to the approaches the different universities can give to the course. A teacher should be able to add a new data structure into the laboratory, to set up an experiment for the students, or to add a new way of visualizing a result.

The objective of this paper is to present the data structures laboratory (developed in the last year by the Software Construction Research Group), which satisfies the previously established requirements. This laboratory was built as a set of Eclipse plug-ins, and it will be used the next semester as a support to the course. This article is structured in the following order: in section 2, the data structures laboratory is presented from different points of view (functional architecture, logical architecture, physical architecture). In section 3, the functionality of the laboratory is shown with an example. In section 4 are mentioned some related works. Finally, section 5 includes the main conclusions.

## 2. LABORATORY ARCHITECTURE

LED is the name of the developed data structures laboratory. Its purpose is to define a completely flexible platform that allows the student to train himself in the design and implementation of data structures. The flexible expression refers to the capability of the platform to support the adding or removing of some of its functional components during execution in a simple way, without the need of rebuilding it. By functional component we refer to a new data structure, a new data generator, a new result interpreter, or even a new sensor type for a structure. We look forward to obtaining a flexible, adaptable, extensible, and dynamic platform, to be used in a simply way by students, teachers, and the academic community.

From a structural point of view, LED is composed by four elements: (1) an engine that handles a set of abstract concepts like structure, experiment, sensor, test, trace, interpreter, etc., which allows these to appear and disappear in runtime and guarantees their proper execution. This engine is installed in Eclipse as a plug-in. (2) A framework that allows building any functional component in a short time. This framework contains a set of 60 classes where the developer can find the elements to extend the laboratory. (3) A set of structures, with the elements to support a typical Data Structures course. (4) A set of tools that allow managing all of the above and, in particular, managing a server where all of the available functional components are published and where the engine connects to when it is launched.

Currently, the laboratory has the following data structures available:

- Vector
- Ordered List
- Syntax Tree
- List
- Binary Tree
- 2-3 Tree
- Stack
- Binary Ordered Tree
- Directed Graph
- Queue
- AVL Tree
- Hash table

In the following sections, the main characteristics of the laboratory are presented in detail, using the functional, physical and logical views of its architecture.

## 2.1. Functional Architecture

LED is a data structure learning support tool for two types of actors. Firstly, the students who can have a direct approach to the concepts learned on a subject. Through experimentation, they can have a better idea of the way the data structure works, and they can have a better understanding of the algorithms operation. In the same way, it is possible to comprehend better some notions like the complexity of an algorithm and the importance of the efficiency, among others. On a second instance, LED looks forward to providing the teacher with a simplified development environment for the creation of new data structures that can be animated and analyzed within the platform. It tries to reduce not only in time, but in complexity, the necessary process to build new data structures with which the students can interact.

The following table summarizes the main functional aspects associated with each one of the actors:

| Actor | Functionality | Description |
|---|---|---|
| Student | Animating an algorithm | The student must select one of the data structures installed on the LED engine. Then, he chooses among all the possible interpreters for the structure the one that gives him a better understanding of the algorithm he wants to work with. An interpreter can be, for example, a graphic viewer of the states of the structure during the execution of the algorithm. The third step consists in choosing a data generator for the structure. The laboratory has different generators, all of them configurable. Finally, the student selects the algorithm he wants to animate.<br>With this information, LED generates the data structure and executes the algorithm, making sure that the sensors and the trace handlers associated with the selected interpreter are correctly installed on the structure. Finally, it gives the control to the interpreter, which allows the student to browse the results. |
| Student | Testing an algorithm | It is a variant of the previous functionality, in which the student develops the algorithm whose operation he wants to visualize. In this case, the laboratory helps the student for developing the algorithm. |
| Student | Making an experiment | An experiment is a structured set of tests for which there is a global definition and a visualization of the set of results as a whole. To do this, the student starts by selecting the structure he is going to work with. Then, he defines the number of tests that compose the experiment. The following step consists of defining the way the data for each one of the tests is going to be generated. This can be done individually or by describing a sequence of growth or decrease. Later on, the student selects an interpreter that allows him to consolidate the test results. Finally, he selects or develops the algorithm he wants to work with. |
| Student | Updating the components installed in the workspace | Through this functional aspect, the student is able to load new components on his workspace. For this purpose, LED connects to the server and verifies the version of the structures it finds. Finally, it asks the student if he wants to install them in the workspace. This allows the updating of the student work elements in a dynamic way. |

| | | |
|---|---|---|
| Teacher | Building a new structure | The teacher can create a new structure, using the elements available in the framework. The creation of a structure implies the construction of all the components necessary to allow its animation. That means the development or adaptation of the following chain of components: sensor, trace handler, test interpreter, experiment interpreter. |
| Teacher | Building a new component to extend an existing structure | The teacher can also define a new component for an existing structure. This can include the creation of a new sensor, or even the development of a new experiment. The laboratory provides all the necessary tools to make this development in the less complexity level possible. |
| Teacher | Publishing a new component | The teacher can publish the new components that he has developed or has adapted, to the server that he is connected to. |

## 2.2. Physical Architecture

From a physical point of view, LED is made up of two main parts, as shown in figure 1. The first one is the laboratory engine, which constitutes the base of the experimentation tool. The second is a server where the new structures and tools are published and stored.

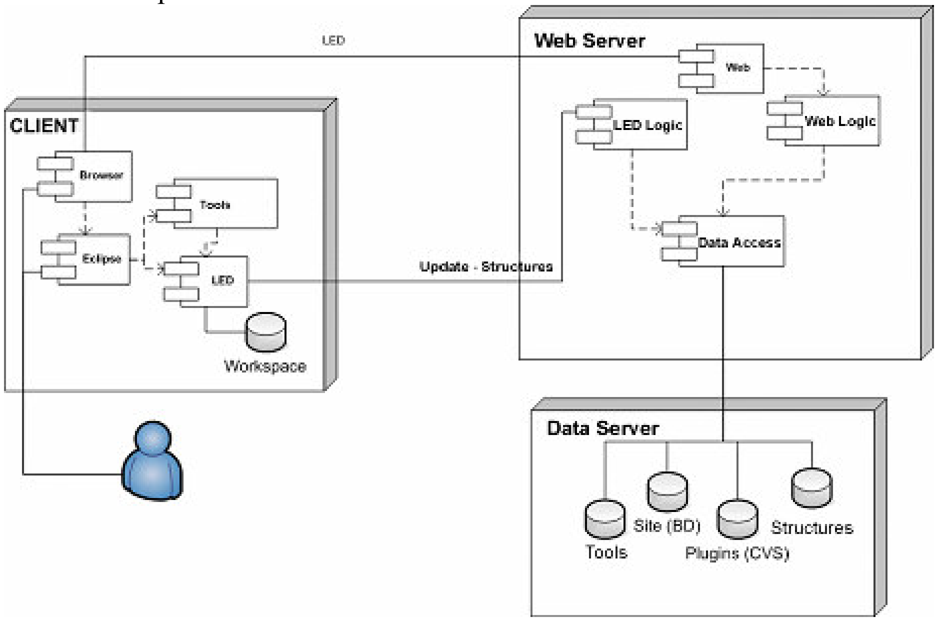


Figure 1 - Physical architecture of LED.

Eclipse [11] was chosen as the base platform for the development of the main application. It is a tool created by IBM and OTI for the integrated development of applications. What makes it attractive and strong is its architecture rather than its functionality. This architecture is based on a plug-in model that allows “plugging” and “unplugging” of functional software components in almost a transparent way. In addition, Eclipse was built under microkernel architecture. So, it only has a small core and the rest of the services are installed around it. This core consists of a dynamic environment of discovery, loading and execution of plug-ins, which are assembled forming modules or subsystems that define and use extension points between them. Indeed, those subsystems are the ones that give the true behavior to the platform since the core only knows about the base concepts. In this way, the desired functionality is incorporated into the platform.

LED takes advantage of the flexibility and adaptation mechanisms proposed by Eclipse. The main plug-in implements the functionality required by the base model and the graphic interface. This plug-in defines some extension points so that the tools and data structures can connect to it. These extension points also allow extending the functionality of the tool in order to add new services. Similarly, the data structures that are part of the platform are built as Eclipse plug-ins using the defined extension points. The data structures can be seen as plug-ins of the LED engine, as shown in figure 2.

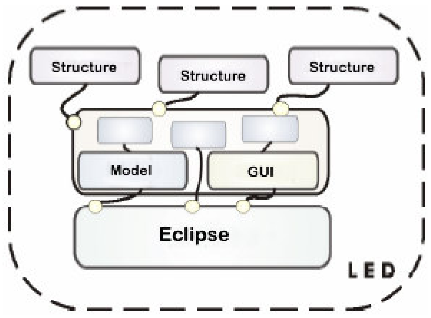


Figure 2 - Framework architecture and tools in terms of plug-ins.

On the other hand, there is the central server of the laboratory. It is a conventional Web server where all the functional components available in LED are published. It is important to say that the process of downloading new structures or updating the existing ones is carried out by adapting the mechanisms that Eclipse provides for this through the same web browsing protocol.

## 2.3. Logical Architecture

In this section we present the logical architecture of the laboratory. More specifically, we present the design of the framework it provides. To do this, it is necessary to distinguish the functionality that the framework offers in two independent and complementary layers. Those layers are the base model and the support for the graphical interface.

### *2.3.1. Model*

Firstly, we present the layer that represents the platform base model. This layer includes the classes that represent the main concepts of the laboratory. At the same time, it registers and handles the behavior of the data structures, as event based systems. This is done using the following elements:

- The definition of the notion of structure in the framework. The model must handle and control the intrinsic characteristics of all data structures, and the properties that allow the integration of those data structures with the rest of the elements of the laboratory.
- The definition of monitoring patterns for the structures. For a data structure analysis, it is necessary to establish patterns that allow gathering significant information about its manipulation in a standard way.
- The definition of translating patterns, with the purpose of interpreting the behavior of a data structure. It is essential to define a convergence point in which the data gathered by the monitoring patterns is interpreted. This ensures the existence of a relation between the interaction with a data structure through an algorithm and its later visualization and analysis.
- The definition of the concept of test and experiment. The model must consider the execution of one or more algorithms and must evaluate different interpretations according to the case.

Under this conception, a logical architecture of four levels is established, as shown in figure 3. In a first level, there is the notion of data structure and the interaction mechanisms between it and the core of the platform. On a second level, there is what we call monitoring schemes. They are based on the concepts of events, sensors and traces, and they are responsible for monitoring the behavior of a data structure when an algorithm is executed. Each event represents a modification of the structure (e.g. addition, elimination or modification of a component). Each sensor is an observer of some of the events generated by the structure. Their responsibility is to store those events in traces. Thus, in the proposed model, it is possible to have several sensors associated to one or more structures, as well as traces connected to one or more sensors. On the third level we find the interpreters. These ones define the ways to achieve the synthesis, analysis and navigation of the information registered in the immediately lower level, and they are associated with the traces or directly with the sensors. Finally, on the highest level, we have defined the notions of test and experiment. A test is the execution of an algorithm with a pair “structure – interpreter”. On the other hand, an experiment corresponds to the execution of a set of tests to which a unique interpreter is associated; the mentioned interpreter is capable of combining all the results obtained in the tests.

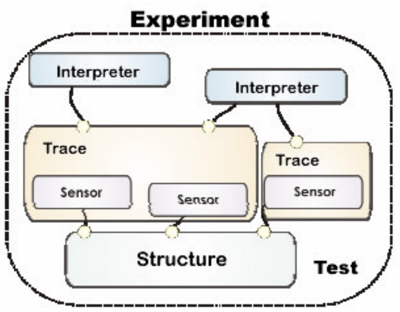


Figure 3 - Conceptual diagram of the four levels model.

### *2.3.2. Graphical interface*

This part is in charge of supporting all the graphical and interaction aspects. This element is composed by a set of graphical artifacts that allows the implementation of the different interpreters. In this way, it assists the typical interaction and visualization tasks in this sort of problems. This part of the logical architecture is composed at the same time by different modules that complement each other. Its purpose is to free the interpreters from the use of any graphical library, particularly from SWT [12] [13]. It is focused in simplifying the creation of graphical interfaces for functional components, reducing its development time.

Firstly, we have the graphical interface base concepts. These include the artifacts that provide the lowest functionality level, hiding the graphical technology used. Here we have the panel notion, whose graphical representation can be seen in figure 4. A panel represents a tab within the view that corresponds to the presentation of the interpreters used in the experiment. A panel offers the methods necessary to handle the graphical elements, among which we find the addition of buttons and other interaction elements, the drawing of lines and geometric figures, etc. Additionally on this level, there are classes in charge of handling the listeners.

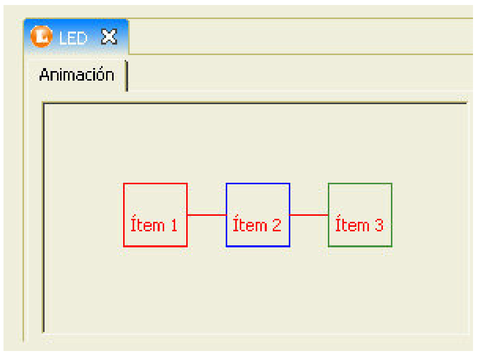


Figure 4 - Graphical representation of a panel in a LED interface.

On the other hand, we have the classes responsible for handling the layouts of graphical elements. This includes the mechanisms and tools necessary to make the creation of a layout a simple task. In this way, LED has a set of default layouts where we can find graphical distributions for linear structures, hierarchic structures, and non hierarchic structures among others. These layouts are configurable and adaptable according to the requirements.

Additionally, LED includes a set of classes that represent different types of graphical elements that are going to enlighten the construction of a structure viewer. Among these we have boxes, circles and connectors, which can be passive or active. This allows certain graphical elements to answer to certain interaction events. As a complement, there are classes for handling colors, fonts and images within the graphical interface model. Similarly, there are functions offered to improve the visualization on the screen, such as zooming and scrolling.

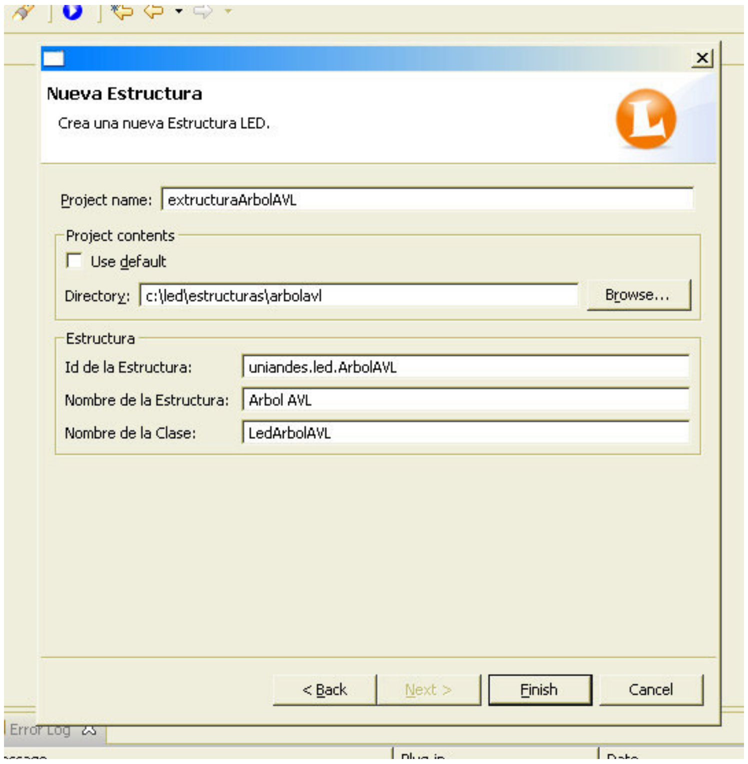
## 3. An example of use of the laboratory

This section illustrates through a simplified example the process used to create a data structure. This is done in order to be able to experiment with that structure later. For this to be possible, the following example is divided in two sections according to the actors involved. In first place, the creation of an AVL tree by the teacher is presented. From this process, a package containing the data structure and a set of associated interpreters is obtained, parts which constitute an installation unit in LED. Then, it is shown the way a student uses that package in the laboratory in order to make a test.

### 3.1. Creation of a Structure

To create a new data structure within the laboratory, the teacher has support at the framework level as well as at the tools level. This process is described in detail step by step:

- Definition of the project. As a first step, the user asks for the construction of a new data structure, creating a new Eclipse project of a particular type (LED Data Structure Creation). The following data is requested: the project name, the new structure identification, the new structure name, and the name of the class that will implement the structure's functionality (see figure 5).



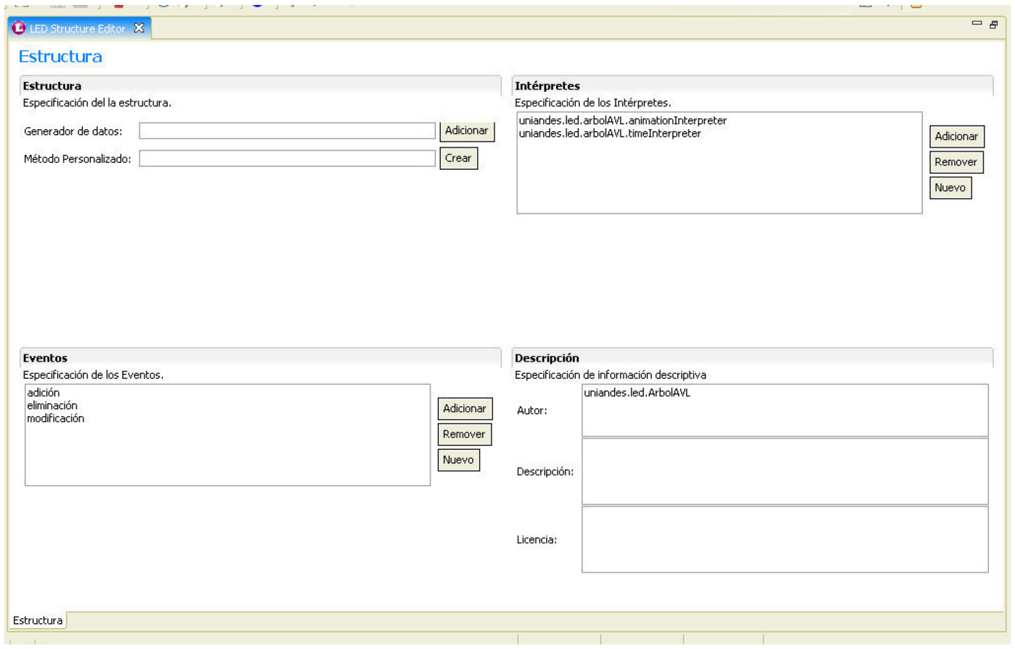
Nueva Estructura
Crea una nueva Estructura LED.

Project name: extructuraArbolAVL
Project contents
Use default [ ]
Directory: c:\led\estructuras\arbolavl
Estructura
Id de la Estructura: uniandes.led.ArbolAVL
Nombre de la Estructura: Arbol AVL
Nombre de la Clase: LedArbolAVL

Figure 5 - Wizard for the creation of a new structure project.

- Then, the user can use an editor that allows him to define the package that will contain the structure. This editor also allows the user to specify the project functional pieces (structure, events and interpreters) and the description of the project (author, description, and license type, among others). A screenshot of this editor is shown in figure 6.

- After the project definition, the user begins the implementation of the structure functionality. LED generates a java file with the basic elements of the new specified structure.



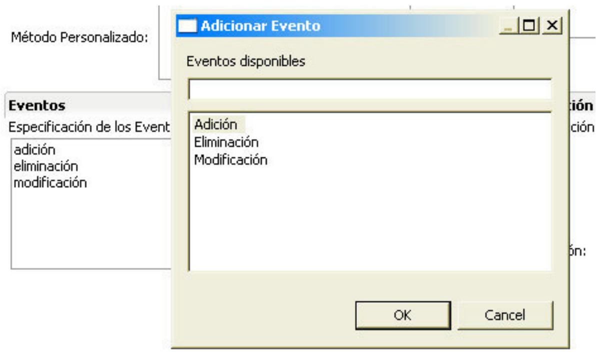
uniandes.led.arbolAVL.animationInterpreter
uniandes.led.arbolAVL.timeInterpreter

adición
eliminación
modificación

Autor: uniandes.led.ArbolAVL

Figure 6 - Project editor.

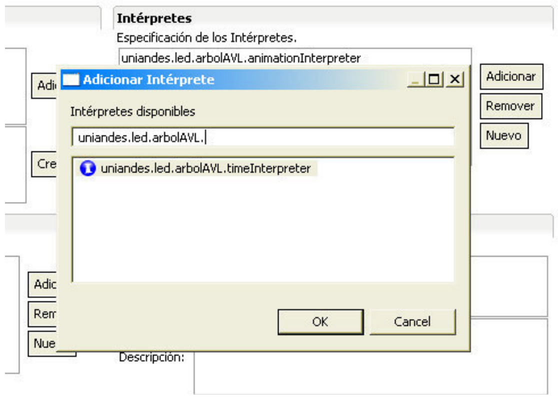
- Once the new data structure is developed, we proceed with the definition of the events that will be triggered when the data structure is manipulated by an algorithm. The project editor allows doing this procedure in a simple way by displaying a list of existing events (see figure 7). New events can be created to adjust to particular requirements of the structure. To make this happen, the user must specify the names of the classes that will implement the functionality of the new events. The templates that provide the base for the creation of the events are generated in the same way as with the data structure java file.



Adición
Eliminación
Modificación

adición
eliminación
modificación

Figure 7 - Declaration of events.



uniandes.led.arbolAVL.animationInterpreter

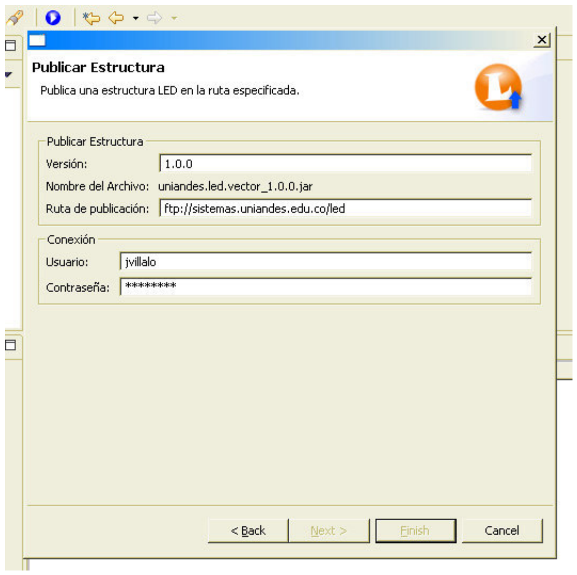
uniandes.led.arbolAVL.

uniandes.led.arbolAVL.timeInterpreter

Figure 8 - Declaration of interpreters.

- The next step is the definition of the interpreters. They are defined using the LED editor, either by adding them from the existing ones or by defining new ones (see figure 8). To define new interpreters, it is possible to use the elements provided by the framework. In this example, we use the interpreter defined by the platform for the animation of algorithms for AVL trees.

- Finally, once the declaration and definition of the different parts of the project are finished, the developed components can be published in the laboratory server. This is done with a tool that allows the specification of the path where the new structure will be published, as it can be seen in figure 9. In doing so, the tool packages the project into its final form (as a special Eclipse plug-in in a JAR archive) and then, it uploads it in the specified server.



Publicar Estructura

Publica una estructura LED en la ruta especificada.

Publicar Estructura

Versión: 1.0.0

Nombre del Archivo: uniandes.led.vector_1.0.0.jar

Ruta de publicación: ftp://sistemas.uniandes.edu.co/led

Conexión

Usuario: jvillalo

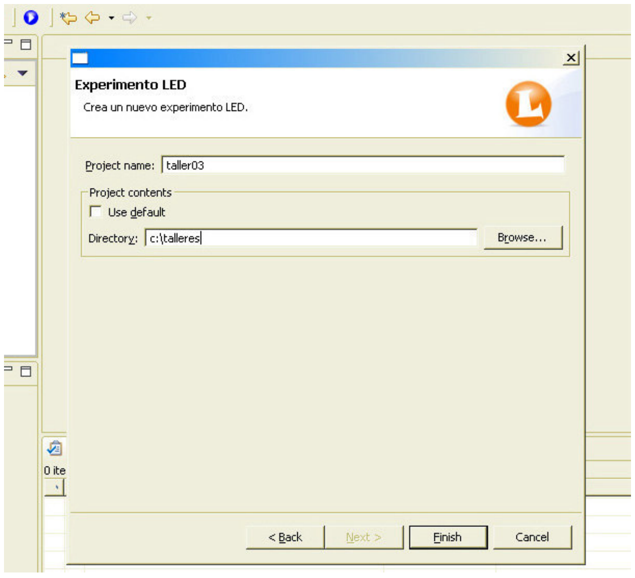
Contraseña: ********

Figure 9 - Publishing of a structure on the laboratory server.

## 3.2. Experimentation on a Structure

The procedure carried out by a student for interacting with an existing structure is simple. First, he must choose the structure. Then, he must add the interpreters needed to perform the test or the experiment. Immediately after, he must parameterize the test, indicating the data filling type and the number of iterations. Finally, he must execute the experiment and analyze the results using the chosen interpreters for this matter. This procedure is shown in detail step by step:

- The first step is to create a new experimentation project in LED, as it is shown in figure 10. We have to give a name to the project in order to identify it. After this, the user can use an editor for choosing the structure.



Experimento LED

Crea un nuevo experimento LED.

Project name: taller03

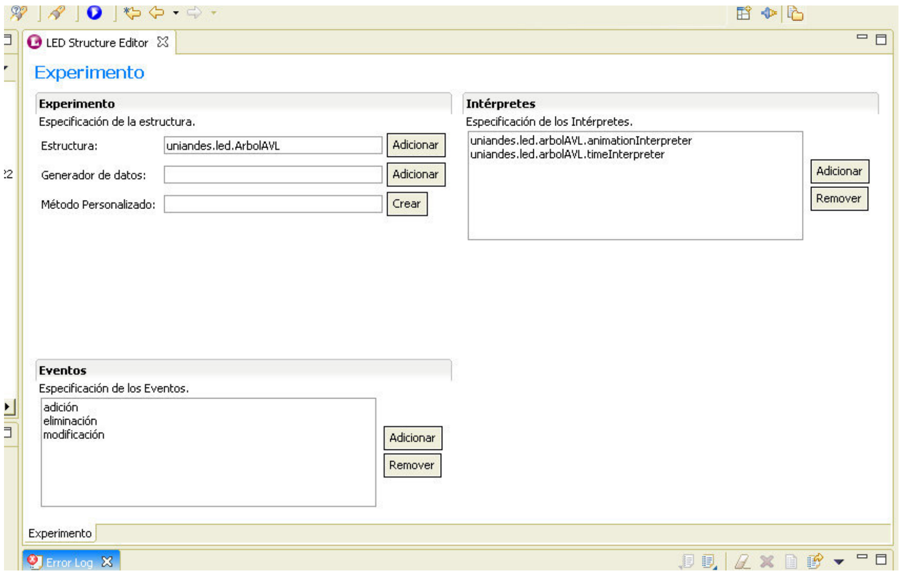
Project contents

Use default

Directory: c:\talleres

Figure 10 - Creation of a new experiment.

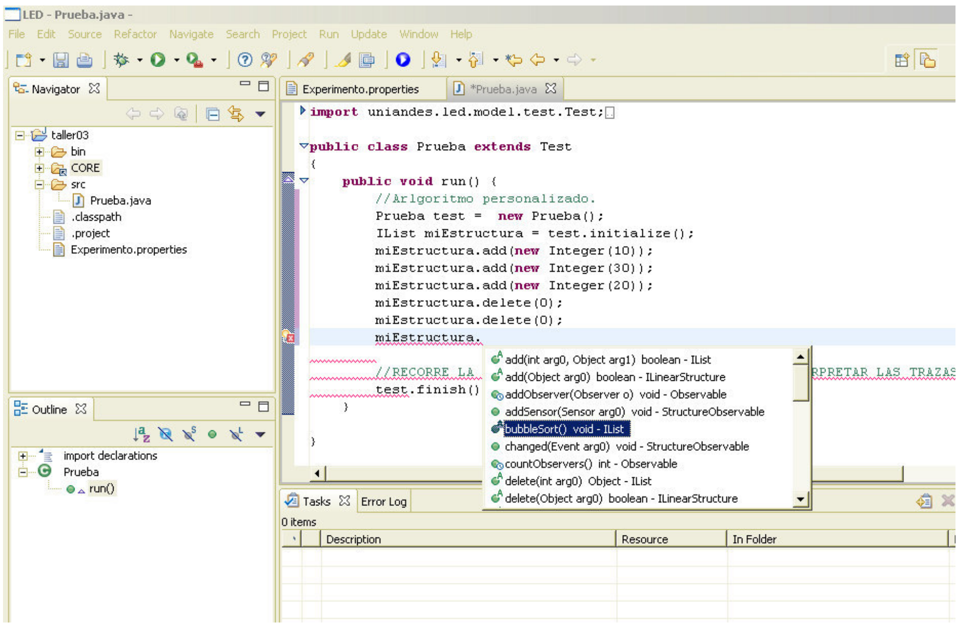
- The second step is to parameterize the test. The editor allows the definition of the test on the selected structure, and the parameters for the corresponding experiment. As shown in figure 11, it is necessary to specify some parameters like how the data structure is going to be filled, the interpreters that will be used, the number of iterations of the test, and the possibility to define a custom method within it.



LED Structure Editor
Experimento
Experimento
Especificación de la estructura.
Estructura: uniandes.led.ArbolAVL
Adicionar
Generador de datos:
Adicionar
Método Personalizado:
Crear
Intérpretes
Especificación de los Intérpretes.
uniandes.led.arbolAVL.animationInterpreter
uniandes.led.arbolAVL.timeInterpreter
Adicionar
Remover
Eventos
Especificación de los Eventos.
adición
eliminación
modificación
Adicionar
Remover
Experimento
Error Log

Figure 11 - Experiment editor in LED.

- Figure 12 shows the process for creating a custom method that is going to be animated. This method or procedure belongs to a special class in the framework that allows its integration with the rest of the experiment. In this way, the basic functionality of the experiment is extended. This, along with the definition and selection of new interpreters, offers a great flexibility in the definition of experiments that can be made.

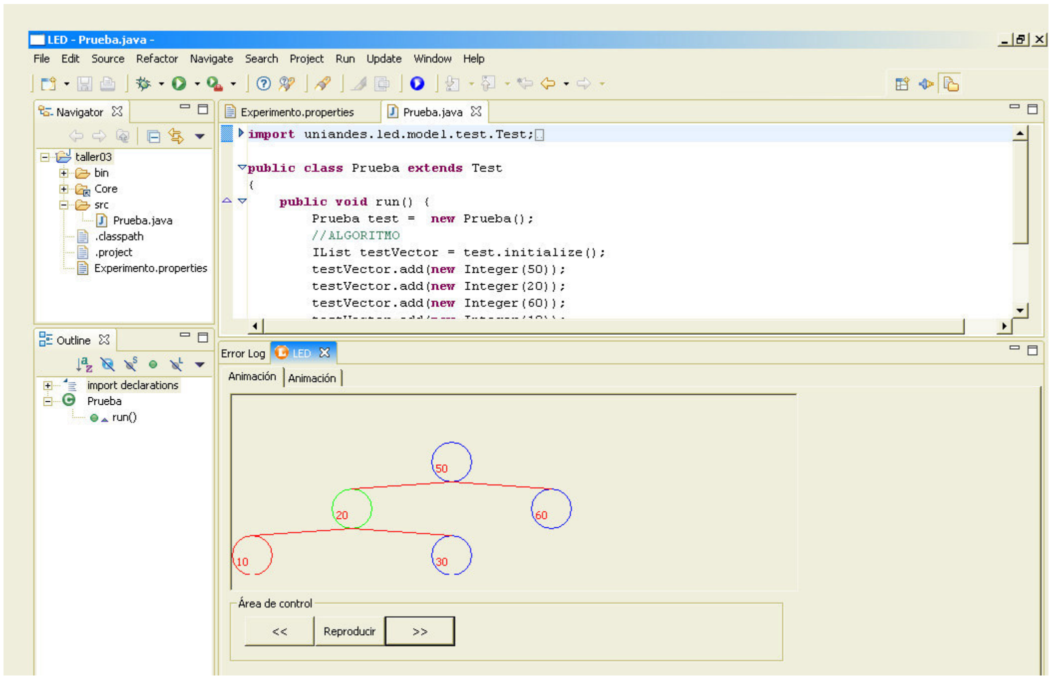


```
import uniandes.led.model.test.Test;

public class Prueba extends Test
{
    public void run() {
        //Arlgoritmo personalizado.
        Prueba test =  new Prueba();
        IList miEstructura = test.initialize();
        miEstructura.add(new Integer(10));
        miEstructura.add(new Integer(30));
        miEstructura.add(new Integer(20));
        miEstructura.delete(0);
        miEstructura.delete(0);
        miEstructura.

        //RECORRE LA                                        RPRETAR LAS TRAZAS
        test.finish()
    }

}
```

```
add(int arg0, Object arg1)  boolean - IList
add(Object arg0)  boolean - ILinearStructure
addObserver(Observer o)  void - Observable
addSensor(Sensor arg0)  void - StructureObservable
bubbleSort()  void - IList
changed(Event arg0)  void - StructureObservable
countObservers()  int - Observable
delete(int arg0)  Object - IList
delete(Object arg0)  boolean - ILinearStructure
```

Figure 12 - Custom method edition.

- Once the parameters of the desired experiment are defined, we can execute that experiment. To do so, we simply press the “Execute Experiment” button of the tool. This executes the steps defined for the test according to the specified parameters. After the execution, the selected interpreters are shown, each one of them with their corresponding gathered and translated data, and according to each criterion (see figure 13). After the experiment execution, the student can analyze the information gathered. He also has the option to save the results for its later study.



```
import uniandes.led.model.test.Test;

public class Prueba extends Test
{
    public void run() {
        Prueba test =  new Prueba();
        //ALGORITMO
        IList testVector = test.initialize();
        testVector.add(new Integer(50));
        testVector.add(new Integer(20));
        testVector.add(new Integer(60));
```

Figure 13 - Execution of the experiment.

## 4. RELATED WORK

There are several projects that propose some tools and trainers for supporting programming education. One of the most mentioned was developed by the *Alphaworks* research laboratories of IBM. It is called *Robocode* [1] and consists of a sort of battle arenas where war tanks created by the participants combat each other. Its academic interest relies on the creation of this war machines. This process is performed by the programming in Java of the different parts of the tank (body, radar and gun). This programming is achieved through a framework provided by the same tool. Through it, the complex world of Java that can be exhausting at the beginning is hidden from the student, introducing him gradually in its concepts in a different and fun way.

Following the same line, there are two projects, *CodeRuler* [2] and *CodeRally* [3]. Both of them are also property of IBM. The difference and special attractive for the development of this project relies in that they are constructed as Eclipse plug-ins.

On the other hand, there are the projects built for the visualization and animation of algorithms. It is important to note the effort the Universidad del Sur Bahía Blanca / Argentina shows with its applications. Here we find SVED [5], a system for the visualization of algorithms for searching and manipulating different data structures. Its attractiveness comes from its vision of being a didactic resource for the programming education through graphical interfaces which are friendly and simple to visualize and implement. Last but not least, there is ZEUS [6], an application developed in 1991 by the *Systems Research Center*. It provides an environment for the animation of algorithms that is more general than SVED.

It is also important to notice about the projects being developed in Georgia Tech, where they offer a framework for animating algorithms called POLKA [7]. Its general purpose is the construction of algorithms and animation of programs. POLKA is a descendant of the XTANGO system, whose purpose is very similar. It uses an interface for UNIX called SAMBA [8], which produces interactive animations for interpretations in ASCII codes. In this group, the work developed by John Stasko stands out [14][15][16], who has focused efforts in the subject of data structures visualization.

## 5. CONCLUSIONS

In this article we presented LED, a data structures laboratory whose main objective is to support the students of the course in the generation of the abilities required to design and handle the data structures needed to solve a problem. In this article, a review of the different aspects of the laboratory was made, making particular emphasis on the functional, logical and physical architecture. An example was used to illustrate the different possibilities that LED gives to a student.

There are several points that are worth mentioning from the laboratory. The first one is that it is constructed to be extended. It has a very flexible architecture, with the purpose that each teacher extends the laboratory with his own experiments, algorithms and structures. The second point to emphasize is that the laboratory has a dynamic architecture which can be modified and updated in execution. This means that if there is a new version of any component of the laboratory, or a new data structure, LED is able to connect itself with the respective server and adapt the work environment of the student. Finally, it is worth mentioning that the laboratory is based on graphical interfaces which are very close to the mental models the student has of the different structures. Somehow, what the student sees in the laboratory is similar to what he sees in the blackboard during the teacher's explanations, diminishing in this way the gap between the theory and what the student sees when he develops a computer program.

## REFERENCES

[1] Hogason K.E. High Performance Computer Simulation and Algorithm Simulator. ACM Journals of Educational Resources in Computing. Vol. 2, No 1, (March 2002), pp. 131-148.
[2] "Robocode", http://aplhaworks.ibm.com/tech/robocode
[3] "CodeRuler", http://aphaworks.ibm.com/tech/coderuler
[4] "CodeRally", http://aphaworks.ibm.com/tech/coderally
[5] "SVED, Sistema de Visualización de Algoritmos", http://cs.uns.edu.ar/lidine/publicaciones/
[6] "Algorithm Animation at SRC", http://research.compaq.com/SRC/zeus/home.html
[7] "Polka Animation System", http://www.cc.gatech.edu/gvu/softviz/parviz/polka.html
[8] "Samba Algorithm Animation System", http://www.cc.gatech.edu/gvu/softviz/algoanim/samba.html
[9] Villalobos, J. Diseño y Manejo de Estructuras de Datos en C, McGraw-Hill, 1996.
[10] Villalobos, J. and Quintero, A. Estructuras de Datos: Un Enfoque desde Tipos Abstractos, McGraw-Hill, 1990.
[11] Gamma, E. and Beck, K. Contributing to Eclipse: Principles, Patterns, and Plugins, Addison Wesley Professional, 2003.
[12] Harris, R. and Warner, R. The Definitive Guide to SWT and JFACE, Apress, 2004.
[13] Northover, S. and Wilson, M. SWT: The Standard Widget Toolkit, Volume 1 (The Eclipse Series), Addison-Wesley Professional, 2004.
[14] Stasko, J. Data Structure Visualization, in Handbook of Data Structures and Applications, D. Mehta and S. Sahni, Chapman & Hall/CRC, 2004, pp. 44-1 - 44-13.
[15] Stasko, J. and Hundhausen, C. Algorithm Visualization, in Computer Science Education Research, Sally Fincher and Marian Petre, RoutledgeFalmer, London, 2004, pp. 199-228.
[16] Hundhausen, C., Douglas, S., and Stasko, J. A Meta-Study of Algorithm Visualization Effectiveness, Journal of Visual Languages and Computing, Vol. 13, No. 3, June 2002, pp. 259-290.