Designing and Using Software Tools for Educational Purposes: FLAT, a Case Study

Jose Jesus Castro-Schez, Ester del Castillo, Julian Hortolano, and Alfredo Rodriguez

***Abstract*—Educational software tools are considered to enrich teaching strategies, providing a more compelling means of exploration and feedback than traditional blackboard methods. Moreover, software simulators provide a more motivating link between theory and practice than pencil-paper methods, encouraging active and discovery learning in the students. The use and development of educational software is a field that has grasped the attention of teachers and researchers from different disciplines and theoretical frameworks in the last few years. In this paper, the authors present SoftwarE for Learning Formal languages and Automata theory (SELFA), an educational software simulator tool, designed to improve the quality of teaching in Theory of Computation courses. The aim of this tool is to make it easier to teach and to learn the main concepts of this subject, whose level of abstraction makes both activities difficult. The main advantage of this tool over other software tools designed with the same purpose, is that it has been developed using web technologies. This allows the user to collect and analyze data on how and when a student or group has used the tool. These numerical data can then be used to evaluate the student's work.**

***Index Terms*—Computer-supported learning, formal languages and automata theory (FLAT) learning, formal learning environments, theoretical computer simulators, WWW-based course-support systems.**

I. Introduction

USING educational software can be useful in the teaching and learning of any given subject and it may be a great aid in the quest for excellence and improvement in the quality of teaching in any university institution. The aim of this paper is to present SoftwarE for Learning Formal languages and Automata theory (SELFA), an educational software simulator tool (http://apps.oreto.inf-cr.uclm.es/selfa/) and to show how it has been designed and developed to improve teaching quality in the subject of formal languages and automata theory (FLAT) in the School of Computer Science at the University of Castilla-La Mancha (UCLM), Spain. The goals of this subject are to introduce the theory of computation through a set of abstract machines that serve as models for computation—finite automata, pushdown automata, linear-bounded automata, and Turing machines—and to examine the relationship between these automata and formal languages. Additional topics beyond the automata classes themselves include deterministic and nondeterministic machines, regular expressions, regular grammars, and context-free grammars.

SELFA may be projected onto a whiteboard at the front of the class to help teachers improve instruction, as a classroom aid. This capability improves the lecture by making it possible to include interactive animations of the various automata or algorithms. Moreover, the tool may be included as an interactive, hands-on component on which students have a chance to work, so that they can study the subject, and check whether they are properly grasping and absorbing the associated theoretical knowledge. Both of these applications of SELFA can significantly increase student motivation and productivity.

In most cases, the use of examples to teach concepts and algorithms is vital in making the students understand them better. In these examples, a perfectly well-defined series of steps is followed. Doing this by hand is too time consuming and so the performance of these steps should be automatized and implemented in an application. The interested reader can find a variety of software tools that have been designed, implemented and used to that end. Examples include FLUTE [9], FOLA [23], JCT [24], [25] JFLAP [26], [27] SEFALAS [18], SEPa! Project [30], or THOTH [11], [12]. These tools are being used in the teaching and learning of FLAT concepts and are freely available via the Internet.

In the design of the tool proposed in this paper, the authors have tried to include the most interesting aspects of the tools aforementioned, namely:

- a graphical user interface that is appealing to the user;
- a text mode input that allows the user to interact easily with the tool;
- an interactive presentation mode to allow experimentation with concepts and algorithms;
- a visual, textual and tabular object representation that provides several levels of abstraction;
- a display of the intermediate steps of the different algorithms rather than just the final solution.

In addition, certain extra characteristics have been added, such as the ability to run via a web browser without any need for local installation or automatically installed plugins (SELFA uses a Client/Server architecture) and the use of a database to collect tool use data. The recorded information will be used as relevant to generate reports and statistics to allow the teacher analyze work done by students. Web-based design is the preferred option in educational software design, since it simplifies installation and platform compatibility issues, and provides immediacy of access [4], [8], [24].

Manuscript received October 15, 2007; revised November 30, 2007. Current version published February 4, 2009. This work was supported by Research Projects e-PACTOS (ref. PAC-06-141) and SARASVATI (ref. PBC06-0064) funded by Junta de Comunidades de Castilla-La Mancha (JCCM).

The authors are with the Department of Technology and Information Systems, Escuela Superior de Informática, University of Castilla-La Mancha, Ciudad Real 13071, Spain (e-mail: JoseJesus.Castro@uclm.es; Ester.Castillo@uclm.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TE.2008.917197

Other aspects of the tool that are worth highlighting are that it allows the following:

- a set of objects (automata and grammars) to be defined, manipulated and used by means of algorithms or operations in one single step;
- the use of variables for storing intermediate results which can then be worked on subsequently.

These features allow the student to experiment with automata and grammars in order to fully develop a solid understanding of these objects.

The SELFA design is based on a formal specification language, which is used to define automata (finite-state and pushdown) or grammars (regular and context-free) and the algorithms or operations that may apply to them. Grammars and the corresponding parsing machines are based on automata theory. The language also provides variables to save intermediate results for future use. Once this formal language that defines what an input should be is established, it is necessary to generate a program for processing that language, as well as an architecture that will facilitate its use and a framework for the integral management of the tool functionality.

Using a formal specification language and its processor to solve problems allows the teacher to make use of the tool to demonstrate the application of some of the subject knowledge to solve a real-life problem. This ability can significantly improve the pedagogy of the course.

The remaining part of this paper is laid out as follows: in the next section the problem to be solved is set out and analyzed. Section III tackles how the SELFA [16] application has been designed and developed. After that, an example will be given of how the tool is used, as far as its main operation is concerned. Finally, some conclusions are put forward, along with data on the use of the tool in the academic years 2005–2006 and 2006–2007.

II. FLAT: An Overview

The subject of FLAT is of great importance in the overall training of those who seek a degree in computer engineering, since it provides the fundamental theories of the discipline. The basic tenets of FLAT enable a better grasp of computer engineering, together with its origins (the historical evolution of the theory of computation and mathematical theory) and equip one to explore its problems and possibilities [15], [19]–[21]. Both formal languages and automata theory are central topics in the curriculum of Computer Science.

At the present time, there seems to be a general concern that education should be practical, above all. As a result, FLAT is no longer a core subject in most computer science curricula. Significantly, the topics of which FLAT consists do not appear in the primary proposals for foundation subjects in degree courses for computer engineering in Spanish universities [1]. The authors believe, or at least firmly hope, that universities will at least give students the chance of receiving teaching in this discipline, within the optional subjects they offer. When students leave university, they should be familiar with automata, grammars, and formal languages—knowledge which will allow them to analyze, understand, and solve problems.

The FLAT subject topics can be structured in four thematic units, each one of which groups together various themes. The topics are as follows:

- *Introduction to formal languages and grammars*;
- *Regular languages and grammars*;
- *Context-free languages and grammars*;
- *Turing machines*.

To learn a subject and get a true grasp of the knowledge and concepts within it inevitably means that the student must experiment with concepts and algorithms. In most cases, doing this is not at all feasible on a large scale in class, due to the quantity of topics to be studied and to the limited time available.

Taking steps to reduce this problem has, thus, become increasingly important. One possible solution consists in giving out material to the students for them to study before it is covered in class, so that class time can be fully given over to doing experiments and running the algorithms step-by-step. The drawback with this approach is simply that the concepts that the students have to learn are often too complicated and abstract for them to be understood without their having been first explained by the teacher.

An alternative solution involves designing and handing out a whole set of exercises, along with their solutions, so that each individual can solve them on his or her own and check the result him/herself. The difficulty here is that this repository of exercises needs to be continually refreshed.

Instead, this paper proposes a third solution, the design and development of an application which has been given the name SELFA. This application allows students to choose a set of exercises on the subject material, and then to generate solutions to those exercises, as well as to access all the information needed to understand them. This approach could motivate students in their study of the subject, as they are the ones choosing their own exercises, and would also foster creativity and their capacity for analysis.

Furthermore, the tool is also meant to be used in lectures given by the teacher as an aid in his or her explanations, thereby improving the teaching itself. The design and development of SELFA sets out to improve the whole teaching-learning process of the subject of FLAT.

The present situation of European universities is that they are moving toward the European Higher Education Area (EHEA) and toward the European Credit Transfer System (ECTS) [2], whose aims are to reflect the real effort required of, and spent by, the student in reaching a number of goals. So in addition to the aforementioned, features a system that is able to monitor each student's use of SELFA has had to be incorporated, to allow that individual's work on FLAT to be assessed. This monitoring system is one of the main advantages of SELFA over other tools designed with the same purpose [3], [6], [9], [17], [18], [22]–[27], [30].

III. Design of the SELFA Tool

In the design of the SELFA [16] application, other existing tools have been taken into account. These include JFLAP [26], [27], JCT [24], [25] SEFALAS [18], as well as the SEPa! Project [30] and ProleTool [28], [29]. The needs of potential users of the

tool have been also kept in mind. SELFA is intended to be used by two kinds of user—computer science students and university teachers. The needs, wishes and priorities of these two groups may be the same, but they may well differ, and this must be borne in mind to achieve a design that will satisfy both types of user.

The main functionality that the tool must offer, from the point of view of the teacher and student, is that of accepting exercises and of working out the solutions to these. This process should be performed clearly and simply, so that understanding the way it reached those solutions is straightforward.

That simplicity is a common goal for both sorts of user, but there are other desirable characteristics that would be wished to add to that objective if the application is to become both practical, likely to be used. The tool should be as follows:

- time unlimited—be able to be used at any time of the day;
- computer-independent and simple to install, so that it can be used in the student's personal computer or in those of the labs, the theory classes or on the teacher's own computer;
- easily upgradable. Implicitly, this tool is design with continual improvement in mind. Any change in the application ought to be easily transferable to computer upon which it has been installed. An upgrade system needs to be built in, to that end.
- intuitive, with a user-friendly interface, both when exercises are being introduced, when explanations are being explored, and in all the operations that the user can carry out with the tool;
- flexible in operation—making it easy to put in as many exercises as are wished and even use intermediate results to perform operations on these;
- accurate—providing information that is correct and error-free;
- comprehensive—providing all the information that is needed to understand the answers that have been obtained;
- platform-independent—able to be executed on any platform, i.e., Unix, Linux, Windows... providing consistent appearance and functionality across platforms.

From the teachers' point of view, it would be desirable for the application to have the following general characteristics, being as follows:

- archetypal—serving as an example for future students of how to use theoretical and practical knowledge in the subject to solve a real problem. In the lectures it should be an example of how to apply concepts studied;
- accessible, having easy and fast access so that it may be used as a teaching aid tool in any given location;
- functional, having access to the most important with options, appropriate presentation of the information generated, depending on the particular setting and purpose;
- transparent—allowing work done with the tool to be viewed, together with the user, if the user allows this. This information can later be used to assess the work done by the student when he or she is learning the subject concepts;
- pedagogically useful—being usable as a teaching tool to design automata, illustrate how automata work and the operation of several useful algorithms;
- widely available as free software, developed with a GPL licence, so that people who are interested in doing so can improve the tool.

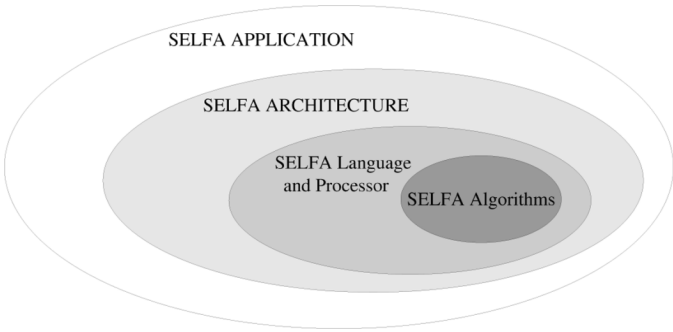


Fig. 1. Incremental design and development of SELFA tool.

The tool was designed incrementally (see Fig. 1). The first step in the process was to decide what exercises were to be done, implementing the algorithms to carry that out. Second, a mechanism for invoking the algorithms was set up. Then this was integrated into an architecture that fulfilled the initial requirements, and finally the capability was added for the system to meet the demands that the user had for the tool. Each step in the design process yielded a product that students and teachers were able to try out and test. Into each of these was built a mechanism to follow up its use. The tool also incorporated a system for gathering comments and suggestions from users, to contribute to the improvement of the tool.

During this process, new requirements emerged which were considered in the final design and development of the SELFA tool. These additional characteristics were that the tool should

- have a language that is easy to learn and use;
- be efficient, to make it possible to work out answers to problem-exercises quickly;
- provide helpful support, in the form of a kind of user's manual system, where users can refer their queries about anything to do with the tool;
- employ uniform notation, using the same notation for automata as for grammars; moreover, this notation should resemble as closely as possible that used in the most important books in the field [15], [19]–[21];
- provide diagram support, showing answers in diagram form to make the solutions easier to understand.

The students were involved in the design of the tool, and their comments having been drawn upon for its subsequent improvement has encouraged them to use the tool.

Following section details the design decisions taken to meet user requirements.

A. *Definition of the SELFA Input*

The first decision to be taken was that of how to input the exercises to the tool. This required the definition of what exactly the input was going to be. An input to the SELFA system consists in the definition of one or more automata (finite-state or pushdown) and one or more grammars (regular or context-free), along specifying a series of operations on those elements. These

operations may or may not give intermediate results which may be used for further operations.

The definition of an *exercise* will imply the use of a reserved word associated with each operation, together with some arguments, amongst which a previously-defined grammar or automaton on which to apply the operation will be found.

To determine how the input to SELFA was going to be provided, the various alternatives used in the existing tools were studied: by means of forms [9], [10], in diagrams [25], [27], [30] or using an input language [13], [18], [23], [29]. The diagram option is a good one for users who are not specialists in this area, but the drawbacks are that interaction with the tool is slow and work has to be done on-line. Input by means of forms has as its main advantage that of being a guided process, which makes it ideal for users who are new to the process, but this has the same negative aspects as does diagram input.

Therefore, the authors chose to use an input file written in a given language in which automata, grammars and the exercises to be done are specified. This alternative, though having the disadvantage of forcing the user to learn the language, has the major advantage of allowing users to work "off-line" and of providing a fast way of inputting exercises. To write the exercises, all that is needed is a text editor to input the exercise or exercises for which an answer is required. The drawbacks to this method can be reduced by designing a language that is simple and easy to learn.

A further decision is that of what a solution consists of and how that will be presented to the user. As solutions, SELFA provides the following:

- when the input is correct: new automata or grammars, with additional information that is useful in understanding the solution and how it has been reached;
- when the input is not correct: a set of useful messages for detecting errors in the input.

The tool offers a mechanism which to allow the user to navigate both through the input, and through the information given as output.

The exercises that SELFA solves can be put into three large groups [16], shown in the following:

- exercises on finite-state automata;
- exercises on pushdown automata;
- exercises on grammars (regular and context-free).

Each exercise solved requires the design and implementation of several algorithms.

B. Designing the Input Language for the SELFA Tool

A little language (SELFA language) was designed to define the SELFA input (see Section II-A) and its processor implemented.

Designing the input SELFA language was a rather delicate process, since this is one of the most important parts of the tool. This language could in future be expanded to include additional features.

The characteristics of the language are as follows:

- its similarity to CUP language, which makes it easier to learn;
- its allowing all the exercises the user wants to perform to be included in a single file input, thus letting all the problems be solved in one operation. This feature is useful when there is limited time on the Internet or when the user is working off-line.
- its allowing intermediate results of the operations performed to be stored in variables which will be operated on again later;
- its having a single print operation, which will behave in one of two ways according to what it is given as its argument (an automaton or a grammar).

The definition of a grammar normally implies the establishing of three aspects [14] as follows:

- Lexical structure; the building blocks of the language called lexical symbols, lexical units or tokens. They will be specified by regular expressions.
- Syntactic structure; rules which establish how to construct valid sentences in the language using the tokens as basic elements. The syntactic structure of the language will be described precisely by a context-free grammar.
- Context and semantic; rules to check context conditions imposed by the language specification and to collect information for semantic processing.

The first noteworthy aspect is that the language designed is case sensitive, meaning that it differentiates between upper and lower case letters. Thus, the interpretations of the words `automaton` and `Automaton` will be different.

The tokens of the SELFA language are as follows:

- Identifiers—names chosen by the user to identify objects of interest.
- Elements. The symbol IT corresponds to the token **element**. The elements are those words which the user employs to name language terms such as the members of the input alphabet of an automaton or its states, as well as the terminal and nonterminal symbols of a grammar. Other examples include the elements that make up the productions of a grammar and the transitions of an automaton.
- Separators. There is a set of characters that support the syntactic of the language. This set consists of punctuation marks:
  ;, :=(implication) |(alternative) {} ( ) =(assignation) $(empty-string)
- Keywords. They are names chosen by the authors to name operations and to help determine, as separators do, the syntactic structure.

The designed language allows users to include comments with explanatory text in its code, thus improving clarity and legibility. Comments in the language are used in a similar way to that of the C or Java programming languages, that is to say, beginning with "*" and will end with "*\." The processor will ignore all the characters that are found within both marks.

The syntax of the proposed language is not too complicated; its goals are to help the user to become familiar with it quickly and to make it as simple and convenient as possible to use. The syntax expressed in Extended Backus Naur Form (EBNF) is shown in http://apps.oreto.inf-cr.uclm.es/selfa/. In this language there are semantic restrictions that can not be expressed by means of syntax [16].

Examples of files written in SELFA language can be found in [16].

C. Processing the Input of the SELFA Tool

The SELFA tool needs a program that accepts as input a text in the language presented in Section III-B, then checks whether the input text is valid and correct or not, and produces as output the solutions of those exercises given, as well as all the information needed to understand them. In other words, the program analyzes the input, constructs a semantic representation, and synthesizes their output from it.

This program, a language processor, has been designed and developed in the traditional modular way, i.e., the front-end has been broken down into three modules and the back-end as a single module [14]. The lexical analysis module isolates SELFA language tokens in the input text. The syntax analysis module converts the stream of tokens into an abstract syntax tree (AST) in accordance with the SELFA language syntax. The context handling module collects context information from various places in the input text, annotates nodes with the results, checks context conditions imposed by the SELFA language specification and translates language-specific constructs in the AST into more general constructs (intermediate code). The solution generation module processes the intermediate code, performing all operations in automata and grammars.

In the language processor construction process, attention was first of all paid to the analysis or front-end phase. The lexical analyzer was designed and generated automatically from regular descriptions of the tokens. The syntax analyzer was generated automatically. The syntax structure to be recognized was specified using a context-free grammar. The automated context-handling methods implemented are based on attribute grammars.

Once the analysis phase had been developed, the synthesis or back-end phase was tackled, where a series of algorithms was implemented. These carry out a set of defined operations on automata and grammars. During the synthesis phase, the analysis and gathering of information are integrated, by invoking the algorithms which solve the problem-exercises. From the valid and correct inputs, the analysis phase of the processor obtains the relevant information from each exercise and communicates this to the synthesis phase, which in turn takes on the task of solving these problems by invoking the algorithms that perform the appropriate operations.

The language processor has the following characteristics:

- *Portability.* Having been implemented in Java programming language the processor is thus multiplatform, allowing it to be run on any computer, regardless of its operating system (reducing machine dependency). Note also that the output of the processor will be given in XML language, as will be discussed in more detail in Section III-D.
- *Extendibility.* The processor was designed in such a way as to be able to add new characteristics if desired; for example new algorithms could be added for solving more problem-exercises.
- *Integrity.* The processor works properly, obtaining the right answers when the input is correct and giving a report on as many errors as possible when the input is not correct. It has a mechanism for error recovery, any syntactical error this could be repaired so that the processor can continue processing the input, analyzing as much of it as possible.

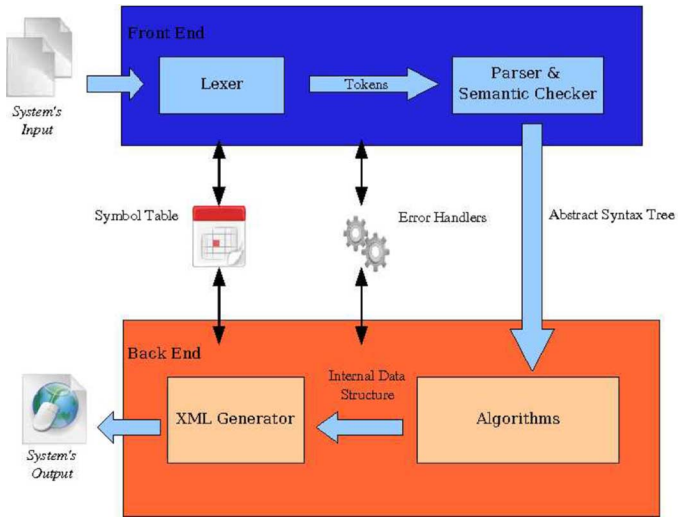


Fig. 2. Simplified structure of the language processor that has been developed.

Fig. 2 shows how the data flow is produced, from the input to the processor to its output.

D. Defining the SELFA Tool Output

Section III-B above shows how the exercises are to be input to the tool by means of a language, as well as how the inputs, or better still the exercises, are to be analyzed and processed, using a processor of that language. Section III-C indicated how the answers to these are worked out by the invoking of the corresponding algorithms. The task that remains is to determine how the solutions are going to be presented to the user in such a way that s/he is able to navigate through them, as well as how to give them in a form that can be displayed on any computer.

The internal format in which the solutions will be generated is the eXtended Markup Language (XML), because this is a language that is easy for external applications to generate and use. What is more, these solutions in XML can be formatted to return the answer in HTML, describing how to display the data in a Web page, thanks to XSL technology. That feature would be really useful when integrating this part into the external interface of the tool to format and transfer data in an easy and consistent way on the World Wide Web.

XML allows the definition of the markup elements, so as to tailor a document to the user's needs, storing and structuring the data in that document as desired. An `<Automa>` element that holds other elements, such us `<Alfabeto>`, `<Estados>` and `<Transiciones>`, and so forth can be created.

E. Description of SELFA Architecture

One of the most important requirements for this tool is that it be as easy to install and update as possible. The ideal would be for it to need no installation at all, which would make its use on any computer more feasible, regardless of where that application is running. Something else that is just as essential is for the tool to register and monitor user activity, with special attention to student use. Given this, the decision was taken to create a centralized application using a Client/Server Web architecture.

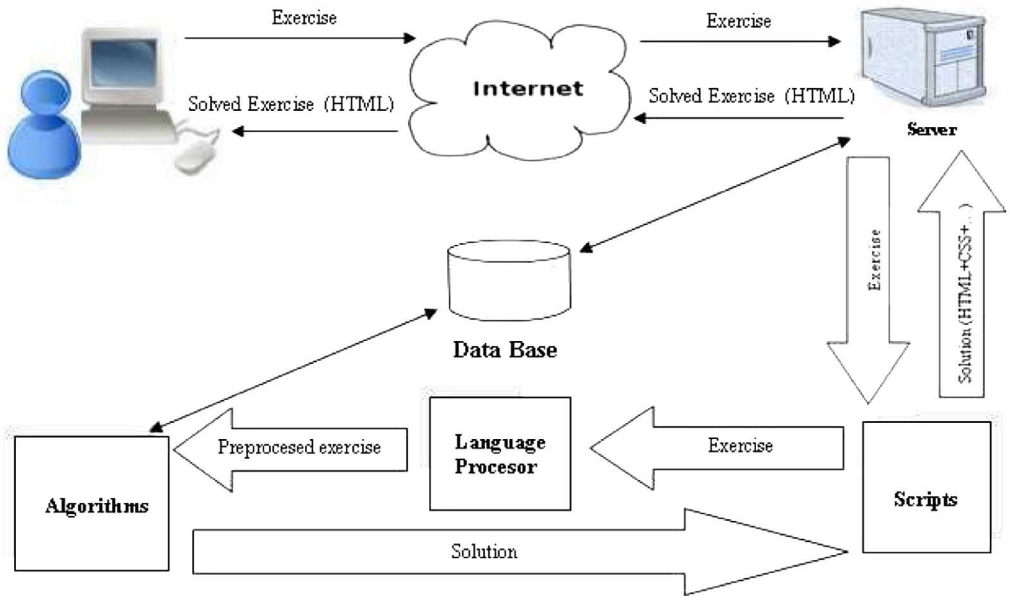


Fig. 3. The architecture of the SELFA tool.

The architecture chosen for the tool was a multilayer Web architecture, in which each layer offers its services to the layer immediately above it and receives the services of the layer below. By doing all this, the level of coupling and of complexity in the tool has been reduced.

The Web architecture has the advantages outlined as follows:

- all the workload falls on the Server, where SELFA is running; this makes the Client (user side) very light;
- there is no local installation in the Client, just a browser connected to the Internet;
- the architecture allows collection and storage of user data in a database held in the server.

The updatings of SELFA, therefore, are immediate, since the centralized application means that, for the changes to be visible wherever the application is used, it is only necessary to update the Server side.

The Web part of the tool was developed using the PHP scripting language, which has the characteristics of being executed on the Server, of being multiplatform and of supporting a large number of users. The database management system on which the users' data, and activities they carry out with the tool, will be registered and stored will be MySQL. MySQL is extremely powerful; it is also free and has a high level of productivity, a low level of consumption and a short response time. Answers are returned in HTML.

Fig. 3 shows the architecture of the tool, together with its information flow. The process starts when the user (student or teacher) sends the exercise over the Web to the Server, where the application is running [16] by means of a form that is accessible to any browser (e.g., Firefox, Microsoft Explorer, etc.).

The Client establishes a connection with the Server, the user authenticates himself and sends exercises. Once the Server has received the exercise or exercises provided by the user, it extracts the relevant information about the person who has invoked the tool, by means of a set of scripts. The Server then processes the file using a language processor constructed for that purpose. This processor analyzes the file and extracts from it the information necessary to the problems posed, so it can then invoke the algorithms for generating a solution to them.

The scripts then take charge of "Encapsulating" and formatting this information on a web-page so this can be published on the Server for the user to see the end result which contains the answers to his or her submitted exercises.

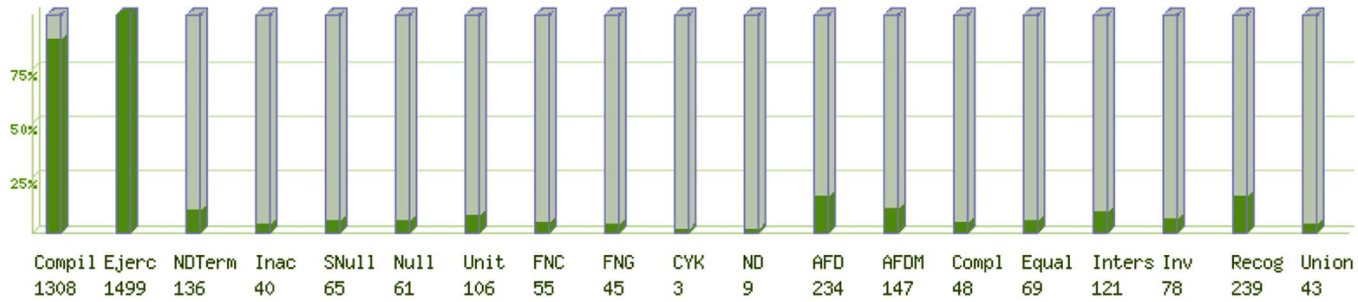
F. SELFA's Additional Capacities

The SELFA tool, apart from solving the exercises proposed by the user, enables:

- management of SELFA users (creating, modifying or deleting user accounts);
- administration of SELFA functionalities and permissions (setting what can be done, when and by whom);
- publication of SELFA news (publishing information about recent events related to the subject or tool and sending these to users by e-mail);
- storage of SELFA users' use data in each academic course (generating reports by a student or group of students in the form of tables and diagrams);
- management of a repository of exercises written in the SELFA language. This repository could be used to store the examples used in normal classrooms, to explain the algorithms or to give solution to exercises proposed.

To do all this, SELFA uses a database made up of the following tables:

- *Config*: This table stores all the data related to the configuration of the tool, such as the Server passwords, the jsp page address, and so on.
- *Exercises*: Here all the details about the exercises in the repository are contained—the route of each proposed exercise, a description of it and the date on which it was included or modified.




```
Número de Compilaciones: 1308
Número de Ejercicios:  1499

NDTerm: 136    (9.07%) Eliminación de Producciones que no deriven en cadena de terminales
Inac: 40       (2.67%) Eliminación de Simbolos Inaccesibles
SNull: 65      (4.34%) Producciones que derivan en la cadena vacia
Null: 61       (4.07%) Eliminación de Producciones que derivan en la cadena vacia
Unit: 106      (7.07%) Eliminación de Producciones Unitarias
FNC: 55        (3.67%) Forma Normal de Chomsky
FNG: 45        (3%) Forma Normal de Greibach
CYK: 3         (0.2%) Algoritmo CYK
ND: 9          (0.6%) Pasar de AFND con lambda transiciones a AFND
AFD: 234       (15.61%) Pasar a AFD
AFDM: 147      (9.81%) Pasar a AFD Mínimo
Compl: 48      (3.2%) Realizar el complemento de un autómata
Equal: 69      (4.6%) Comprobar la igualdad de dos autómatas
Inters: 121    (8.07%) Realizar la intersección de dos autómatas
Inv: 78        (5.2%) Realizar el inverso de un autómata
Recog: 239     (15.94%) Reconocimiento de una cadena por un autómata
Union: 43      (2.87%) Realizar la Unión de dos autómatas
```

Fig. 4. Statistical bar-graph diagram.

- *News*: This table stores the relevant information about the news published on the tool, such as the route of the text files containing the news and the date on which it was published.
- *Users*: On this table all the data referring to the users that have subscribed to the tool are registered, such as their first name and surname(s), identity card number, their nickname and e-mail address.
- *Operations*: This is one of the most important tables in the application, since it is here that the operations performed by the users are saved. The most relevant fields in the table are: user, exercises done and the date on which they were submitted to the tool. This information grouped by the student or by group of students in tabular form, can be shown by means of a bar graph or a pie chart (see Fig. 4).
- *Punctuation*: The remarks and comments made about the tool by users are saved onto this table.

The information needed to fill in the table "Operations" in the database is extracted directly by the language processor during the analysis of an input. The person who invokes the tool and the date when that was done is known thanks to the execution of an authentification process. This process is voluntary, and is done when the person calls up the tool. In the semantic check, the problems to be solved are analyzed and their particular type classified—this is information which will be stored on the table.

IV. Experimental Results

In this section, we report on the authors' experience using SELFA to enhance the teaching of formal languages and automata theory and to make this more attractive to the students. The tool was evaluated with real users, the students of the BSc in Computer Science at University of Castilla-La Mancha. SELFA was running in the second term of the academic years 2005–2006 and 2006–2007, which was just when the classes in the subject of FLAT began in the undergraduate course of Computer Engineering. The number of students enrolled in the course was 90 and 66, respectively. The tool was offered to the students as a possibly useful aid and the usage was not always mandatory. Nearly half of these students created an user account in SELFA (45 and 30 students) and thus had a user name and a password. However, only 25 and 21 students regularly used the tool in their studies (in the academic years 2005–2006 and 2006–2007, respectively), that is, they used the

TABLE I
SUMMARY OF THE RESULTS ON USE OF THE TOOL

Variable	05/06	06/07
SELFA registered users	45	30
Students who used the tool in their studies	25	21
Files submitted	1196	1299
Exercises done	1449	1195

TABLE II
EXERCISES CARRIED OUT ON GRAMMARS IN SUMMARY FORM

Exercises	05/06	06/07
Eliminating nongenerating variables	134 (9,24%)	43 (3,31%)
Removing unreachable symbols	40 (2,76%)	4 (0,31%)
Obtaining nullable symbols	63 (4,35%)	43 (3,39%)
Removing lambda productions	60 (4,14%)	44 (3,39%)
Removing unit-productions	104 (7,18%)	19 (1,46%)
Chomsky Normal Form	55 (3,8%)	24 (1,85%)
Greibach Normal Form	45 (3,11%)	3 (0,23%)
CYK Algorithm	3 (0,21%)	40 (3,08%)
Exercises carried out on Grammars	504 (34,78%)	220 (18,41%)

TABLE III
EXERCISES CARRIED OUT ON AUTOMATA IN SUMMARY FORM

Exercises	05/06	06/07
To convert λ-NDFA into NDFA	6 (0,41%)	54 (4,16%)
To convert λ-NDFA and NDFA into DFA	228 (15,73%)	217 (16,71%)
To obtain MDFA from FA	144 (9,94%)	150 (11,55%)
To perform the FA complement	44 (3,04%)	61 (4,7%)
To perform the FA inverse	63 (4,35%)	56 (4,31%)
To carry out the intersection of two FA	121 (8,35%)	93 (7,16%)
To carry out the union of two FA	38 (2,62%)	82 (6,31%)
To check if two FA are equivalents	69 (4,76%)	120 (9,24%)
To check if a string is recognized by FA	232 (16,01%)	142 (10,93%)
Exercises carried out on Automata	945 (65,22%)	975 (81,59%)

tool more than 15 times per month during the instruction period (see Table I).

The overall results of SELFA's use, measured by means of the amount of exercises done by the students (1449 and 1195 in the academic years 2005–2006 and 2006–2007, respectively), show that students did use this tool (see Table I). Tables II and III show the statistics on the type and quantity of exercises done on grammars and automata.

The authors created a questionnaire and passed it out to the SELFA users to ascertain whether they considered the tool to be useful by them in studying the concepts and algorithms of FLAT. The ease of use of the tool was also included. Two questions were formulated to measure these aspects: *How useful do you find the tool?* and *How easy is to use?*. The students who used SELFA were asked to answer anonymously. Respondents could choose among five possibilities to the first question: "Not Useful" (1), "Not very Useful" (2), "Useful" (3), "Very Useful" (4), and "Extremely Useful" (5); and other five to the second one: "Very Difficult" (1), "Not Easy" (2), "Easy" (3), "Quite Easy" (4), and "Very Easy" (5). Only one answer could be selected. The assessment of SELFA by those students who employed the tool in their studies, as regards the ease of its use and its usefulness is clear: they find it very useful and quite or very easy to use. On average, students rate the tool usefulness as a 4.4 and 4.3 and the ease-of-use feature as a 4.3 and 3.85 on a five-point scale in the academic years 2005–2006 and 2006–2007 (see Table IV).

TABLE IV
MEAN VALUE GIVEN TO THE TOOL BY THE STUDENTS (VALUED OVER 5)

Variable	05/06	06/07
Ease of use	4,389	3,85
Assessment of usefulness value	4,417	4,3

TABLE V
ACADEMIC RESULTS OBSERVED

	05/06	06/07
Students enrolled	90	66
Students Passed in FLAT	32	21
Students Passed in FLAT and who are SELFA users	24 (75%)	10 (47%)

With respect to the usefulness of the tool in teaching the concepts of the FLAT subject, after an analysis of the marks obtained in the exams of the academic years 2005–2006 and 2006–2007 by students who used the tool (25 and 21, respectively), 24 and 10 of them passed the exam in the subject, which is 75% and 47% of all those who passed, 32 and 21, respectively. Moreover, these students had better results than students who did not use the tool. The results obtained were highly satisfactory, with students who used the tool have gained more knowledge than those did not. SELFA has been a useful aid in the learning of the subject of FLAT (see Table V) and so the aim is for it to carry on in operation over upcoming academic courses.

V. CONCLUSION AND FUTURE WORK

In this paper, the authors have given a view of work done at the School of Computer Engineering, University of Castilla-La Mancha, Spain, to improve the teaching quality, the monitoring of students' work, and the assessment mechanisms in the subject of Formal Languages and Automata Theory.

A tool called SELFA has been presented here, whose main objective is to allow exercises on different areas of the subject to be formulated, as well as to provide the answers to these. SELFA can be used both by the teacher in his or her lectures in order to make the subject topics more interesting and attractive to students, and students in their attempts to grasp the subject matter. Moreover, the tool allows the storage and monitoring of each student's work throughout the duration of a course. This information is used to evaluate the student's work.

SELFA design is based on a language processor (see Sections III-B and III-C), which is executed in a client/server architecture, allowing it to be used via the World Wide Web (see Section III-E). This feature constitutes SELFA's main advantage over other tools which share the same overall goals [18], [24]–[27].

Another advantage of SELFA is that it serves as example of how to use theoretical and practical knowledge in the subject to solve a real problem—with a resulting strong pedagogical impact. In lectures has been used as an example of how to apply in practice concepts that have been studied in theory, which has been valued highly by the students.

Future work includes continuing experimentation with the proposed tool and extending it, adding new operations. For

example, obtaining the regular expression from an automaton using the characteristic equation method [5] or the construction of an automaton from a regular expression using Thomson's Algorithm [31] will be added to SELFA. Several operations have been recently implemented, they have yet to be tested with student, so there is no usage data available. This data will be obtained and analyzed. The final EBNF specification of the SELFA language and operations supported by the tool can be found in [16].

ACKNOWLEDGMENT

The authors would like to thank the School of Computer Engineering, University of Castilla-La Mancha, and to the students of 2005–2006 for their help and support in this project, as well as for their suggestions, which contributed towards its development and improvement.

REFERENCES

[1] "Agencia nacional de evaluación de la calidad y la acreditación (ANECA)," Libro Blanco del Título de Grado en Ingeniería Informática 2007 [Online]. Available: http://www.aneca.es/activin/docs/libroblanco_jun05_informatica.pdf

[2] "European commission, education and training," The Bologna Process, Towards the European Higher Eduation Area 2007 [Online]. Available: http://ec.europa.eu/education/policies/educ/bologna/bologna_en.html

[3] A. O. Bilska, K. H. Leider, M. Procopiuc, O. Procopiuc, S. H. Rodger, J. R. Salemme, and E. Tsang, "A collection of tools for making automata theory and formal languages come alive," in *Proc. 28th ACM SIGCSE Tech. Symp. Comput. Sci. Educ.*, 1997, pp. 15–19.

[4] C. M. Boroni, F. W. Goosey, M. T. Grinder, and R. J. Ross, "A paradigm shift! The internet, the web, browsers, Java, and the future of computer science education," in *Proc. 29th ACM SIGCSE Tech. Symp. Comput. Sci. Educ.*, 1998, pp. 145–152.

[5] J. A. Brzozowski, "Derivatives of regular expressions," *J. Assoc. Comput. Machin.*, vol. 11, no. 4, pp. 481–494, 1964.

[6] C. I. Chesñevar, M. L. Cobo, and W. Yurcik, "Using theoretical computer simulators for formal languages and automata theory," *ACM SIGCSE Bull.*, vol. 35, no. 2, pp. 33–37, 2003.

[7] N. Chomsky, "Three models for the description of language," *IEEE Trans. Inf. Theory*, vol. 2, no. 3, pp. 113–124, 1956.

[8] D. Cole, R. Wainwright, and D. Schoenefeld, "Using java to develop web based tutorials," in *Proc. 29th ACM SIGCSE Tech. Symp. Comput. Sci. Educ.*, 1998, pp. 92–96.

[9] V. Devedzic, J. Debenham, and D. Popovic, "Teaching formal languages by an intelligent tutoring system," *J. Educ. Technol. Soc.*, vol. 3, no. 2, pp. 36–49, 2000.

[10] J. L. Diez and J. Diaz, Java Parser Project Home Page, 2007 [Online]. Available: http://paginaspersonales.deusto.es/josuka/jparser/parser.html

[11] C. Garcia-Osorio, A. Arnaiz-Moreno, and A. Arnaiz-Gonzalez, "Enseñanza asistida de teoría de autómatas y lenguajes formales mediante el uso de THOTH," in *XIII Jornadas de Enseñanza Univ. Inf.*, 2007, pp. 425–432.

[12] C. Garcia-Osorio, A. Arnaiz-Moreno, and A. Arnaiz-Gonzalez, THOTH Project Home Page, 2007 [Online]. Available: http://pisuerga.inf.ubu.es/cgosorio/THOTH/

[13] M. T. Grinder, "Animating automata: A cross-platform program for teaching finite automata," *ACM SIGCSE Bull.*, vol. 34, no. 1, pp. 371–375, 2002.

[14] D. Grune, H. E. Bal, C. J. H. Jacobs, and K. G. Langendoen, *Modern Compiler Design*. New York: Wiley, 2001.

[15] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Reading, MA: Addison-Wesley, 2006.

[16] J. Hortolano, A. Rodriguez, and J. J. Castro-Schez, SELFA Project Home Page, 2007 [Online]. Available: http://apps.oreto.inf-cr.uclm.es/selfa/

[17] T. Hung and S. H. Roger, "Increasing visualization and interaction in the automata theory course," *ACM SIGCSE Bull.*, vol. 32, no. 1, pp. 6–10, 2000.

[18] J. F. Jodar and J. Revelles, SEFALAS Project Home Page, 2007 [Online]. Available: http://lsi.ugr.es/~pl/software.php

[19] D. Kelley, *Automata and Formal Languages: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1998.

[20] H. R. Lewis and C. H. Papadimitriou, *Elements of Theory of Computation*. Upper Saddle River, NJ: Prentice-Hall, 1998.

[21] J. C. Martin, *Introduction to Languages and the Theory of Computation*. New York: McGraw-Hill, 2003.

[22] J. McDonald, "Interactive pushdown automata animation," *ACM SIGCSE Bull.*, vol. 34, no. 1, pp. 376–380, 2002.

[23] Q.-N. Tran, "Interactive symbolic software for teaching formal languages, automata and beyond," *J. Comput. Sci. Colleges*, vol. 22, no. 4, pp. 129–136, 2007.

[24] M. B. Robinson, J. A. Hamshar, J. E. Novillo, and A. T. Duchowski, "A java-based tool for reasoning about models of computation through simulating finite automata and turing machines," in *Proc. 30th Ann. ACM SIGCSE Symp. (Special Interest Group on Computer Science Education)*, New Orleans, LA, 1999, pp. 105–109.

[25] M. B. Robinson, J. A. Hamshar, J. E. Novillo, and A. T. Duchowski, The Java Computability Toolkit Project Home Page, 2007 [Online]. Available: http://humboldt.sunyit.edu/JCT/

[26] S. H. Rodger and T. W. Finley, *JFLAP: An Interactive Formal Languages and Automata Package*. Sudbury, MA: Jones & Bartlett, 2006.

[27] S. H. Rodger and T. W. Finley, JFLAP Project Home Page, 2007 [Online]. Available: http://www.jflap.org

[28] P. A. Santos and J. J. Castro-Schez, "Una herramienta para la enseñanza y aprendizaje de la asignatura Procesadores de Lenguajes," *XII Jornadas de la Enseñanza Univ. Inf.*, pp. 499–506, 2006.

[29] P. A. Santos, J. Santos, and J. J. Castro-Schez, PROLETOOL Project Home Page, 2007 [Online]. Available: http://oreto.inf-cr.uclm.es/proletool/

[30] J. J. Tamagnini, S. V. Cavadini, P. L. Berdaguer, D. A. Cheda, F. M. Pachado, and M. Petersen, SEPa! Project Home Page, 2007 [Online]. Available: http://www.ucse.edu.ar/fma/sepa

[31] K. Thompson, "Regular expression search algorithm," *Commun. ACM*, vol. 11, no. 6, pp. 419–422, 1968.

Jose Jesus Castro-Schez received the M.S. and Ph.D. degrees in 1995 and 2001, respectively, both from the Computer Science Department, University of Granada, Spain.

He is an Associate Professor of Computer Science, University of Castilla-La Mancha, Ciudad Real, Spain, teaching formal languages, automata theory, and compiler technology. His research interests include: knowledge acquisition, machine learning, decision support, electronic commerce, and issues of representation in AI. He is the author of numerous papers on AI-related subjects. He is currently interested in the educational implications of computer-supported collaborative learning.

Ester del Castillo received the B.S. and M.S. degrees from the Computer Science Department, University of Granada, Spain.

She is an Assistant Professor with the Department of Technology and Information Systems, Escuela Superior de Informática, University of Castilla-La Mancha, Ciudad Real, Spain. She teaches formal languages, automata theory, and theory of computation.

Julian Hortolano received the B.S. and M.S. degrees from the Department of Technology and Information Systems, Escuela Superior de Informática, University of Castilla-La Mancha, Ciudad Real, Spain.

Alfredo Rodriguez received the B.S. and M.S. degrees from the Department of Technology and Information Systems, Escuela Superior de Informática, University of Castilla-La Mancha, Ciudad Real, Spain.