Towards a logic lab on the cloud: desktop and mobile sessions*

Debora Rodriguez-Aguilar, Juan C. Acosta-Guadarrama[0000−0002−4837−1578], and Víctor M. Morales-Rocha[0000−0001−8213−400X]

National Laboratory for Information Technologies, LANTI, Mexico;
http://www.lanti.org.mx
Autonomous University of Juarez, Institute for Engineering and Technology, JUÁREZ, CHIHUAHUA, 32310, MEXICO
al132081@alumnos.uacj.mx, {juan.acosta,victor.morales}@uacj.mx

Abstract. This case study proposes a novel system of the Internet as a service for Answer Set Programming (ASP) solvers, keeping open sessions for registered users, in such a way they can work ASP knowledge bases anywhere, without having to download, compile, install and configure the solver. The technique can exploit the cloud and mobile potentials, such as processor and memory management, GPS (Global Positioning System), accelerometer, thermometer, heart rate sensor, touchscreen and gestures, voice, images, fingerprints, signatures, phone calls, wristwatch, proximity sensor, and so on. It can also get benefited from cloud computing and data mining potentials, such as massive computing and memory distribution, big-data collection and analysis, centralized work, work sessions, scalability, concurrent teamwork, and more. We started this proposal with the aim to offer an alternative to the everyday use of DLV and CLINGO ASP solvers, readily available on devices that everybody uses, like browsers and mobile devices. This time we focus on setting up a global service, the desktop browser, and mobile interfaces. We also provide a methodology of incremental development, consisting in showing work progress into phases of analysis, modeling, design, and implementation of the solution. In the following, we introduce the problem of computing ASP knowledge bases online and background. Later we add a brief survey on similar works. Then we present our proposal. Finally, we conclude with a discussion and future work.

Keywords: Answer Set Programming solvers · cloud computing · web programming · mobile devices · data mining · usability.

1 Introduction

Answer Set Programming (ASP) is a consolidated declarative programming paradigm first proposed back in 1988 [11]. It provided a foundation to study and experimentation in several investigations, such as the Frequency Assignment Problem, FAP [9], Modeling of Biased Decisions [20], to mention a few. They all make use of ASP software

* This work has been supported by mainly supported by The National Council for Science and Technology, CONACYT, 280712, Consolidación de Laboratorios Nacionales. We are also grateful to the Autonomous University of Juarez, Mexico.

as their core engine. In the same way, there exist development projects to implement generic inference engines under the ASP language, called solvers. A solver is a mechanism that calculates conclusions in ASP semantics, that is to say, that, from specific syntactic rules, it admits well-formed logic sentences, interprets and evaluates them from the knowledge base (KB), and finally gets out the models that satisfy the KB to solve a problem. At present, there are several solvers or inference engines, such as Smodels, PSmodels, and others. However, this case study focuses on DLV [4] and CLINGO [10,21] only, both world-renowned by researchers from around the world. The popularity of both systems is due to their multiple applications in areas such as mathematical logic, knowledge representation, and reasoning, as mentioned before. However, the original use of an inference engine is limited to being used by a command-line interface—CLI.

The problem with traditional command-line applications is that one needs to download them, sometimes to compile them, to install them and to run them on the terminal. For that, most of the times one needs unique skills in programming and operating systems administration, like Linux/UNIX. It is not just old-fashioned and impractical, but also out of the scope of today's Artificial Intelligence (AI) demands. Today's AI applications demand the use of online services, concurrency, mobile devices, lots of data collection, merging of different heterogeneous sources of information, big-data processing and interpretation, which are not an easy task. So, one of the first steps to tackle is an adequate modern graphical user interface (GUI) for the web apps and mobile devices. Not only is it practical and novel but necessary for today's demands. Our proposal consists of a web front-end prototype for those two solvers and more, a portal that admits both registered or unregistered users, concurrency, cloud computing, saved sessions for registered users, and a native Android mobile app to access the cloud service. The first aim of the proposal is to provide a modern ASP solvers environment both for scientists and students, for their experiments and teaching on the formal system. However, its use shall be not limited to that. If we implemented an adequate front-end, it would open up new potentials and branches of research, like scalable cloud computing; and processing of big data collected by mobile devices. The integration of mobile devices with ASP is not luxurious. It permits other nice features desktop ASP applications lack, like portability, and the collection of detailed information to make inferences from, such as frequently visited places, road routes and (path) planning, health monitoring, shopping and sports habits, voice processing and image processing, phone call classifications, text messaging, and scheduling, to mention a few. The collection of such data would be possible due to the different input devices a cellular phone has, like wristwatch apps, proximity, the global-positioning system (GPS), fingerprint, touchscreen, gestures, accelerometer, thermometer, heart-rate sensor, microphone and camera, and so on. It is about exciting new theory and applications of ASP, human-computer interaction, Natural Language Processing, Multi-Agent Systems, and Ambient Intelligence; towards finding patterns in human behaviors and services.

Our proposal presents LogicSite, a multi-platform application prototype both for desktop computers and Android mobile devices. It is intended for users of ASP solvers, who wish to focus on their semantics work rather than dealing with downloading, installing, configuring and typically running text-based interface ASP solvers. What is

more, online users do not even have to worry about a fast parallel processor or memory issues. In this paper, our main areas of study are the implementation of ASP solvers on the cloud, as well as a native mobile app, which can exploit the benefits of such technology. Having such services fully-implemented and available shall keep their users from intricate details of managing local resources to download, compile, install, configure, and run them.

2 Background

As far as we can tell, at present, the existing projects associated with the use of ASP inference engines are intended for web development. Evidence of this assertion is available with the following applications:

- LoIDE [6,12,13] is an integrated development environment (IDE)[1] of type *web-based—software* for the web. Its objective is to offer an advanced and modular web editor for ASP logic languages (DLV and CLINGO). Among its functions are the execution, the processing of logic statements from an external file[2], and an option to download the software to run it locally. Also, it includes functions for the interface such as keyboard shortcuts, selection of filtering options and customization—design and appearance.
- LogicLab [1] is a web app that aims to provide online-running inference engines in a graphic browser. So, users need to focus on their work only, rather than other details like how to download the solvers, how to install them, how to configure them and how to run them on a text terminal. The ASP solvers available in LogicLab are DLV and CLINGO, among others, both with logic program processing parameters.
- Running Clingo [22] is an ASP tool that, as its name says, implements the CLINGO inference engine. Among its functions, we can highlight the reasoning mode, in which one can select an option from the four existing ones—*default, brave, cautious and enumerating all*. Besides, they include a list of ready-to-run examples.
- EmbASP [6,7] is a framework for the integration of logic programming in external systems for the general-audience applications. It offers two implementations (in Java and Python), in addition to providing DLV, CLINGO and DLV 2.0. Mobile apps like DLVfit [6,7,8], GuessAndCheckers [6,7,3], DLVEdu [6,7], and Connect4 [6,7], they have used this framework for logical deductions that they generate and execute internally.

Despite the existence of these applications, the solution they offer individually is not yet satisfactory; this is because one application has functions that another does not and vice versa. This table shows the main features that, in our view, we consider viable or acceptable and desirable for an online laboratory to provide modern service of ASP solvers.

These projects display logic calculation tools in a modern graphical environment. The latter is something new because the everyday use of ASP solvers is through an old-fashioned command line interface, and we had talked about its cons above.

[1] An IDE is a *software system* that allows code editing, compilation and debugging.

[2] The type of files that it allows is JSON—JavaScript Object Notation, a format for the exchange of data over the network.

3 Preliminaries

In this section, we introduce some basic concepts, needed to underpin our proposal. The main ideas here are portability, scalability, and usability, ans security, which state a sort of principles to meet by adequate applications for modern technologies, in our view. Later, we introduce some simple ASP background, where, in this paper, we assume the reader is familiar with its syntax and semantics, easily accessible in the literature.

3.1 Usability

Jakob Nielsen [17] explains, to begin with, that "usability is a fundamental characteristic for customers to visit, take advantage of and return to a website again, increasing the success of their Internet experience." As a result, we define *usability* as the way in which users interact with a website, application or mobile app. The usability of systems

System Usability		
Attribute	**Definition**	**Measurement Method**
Ease of Learning	It is the minimum time required for the user to learn of the functions of the application and to increase their productivity.	Required time for a beginner user to reach the level of expertise.
Efficiency	Ability to perform or fulfill a task productively.	The time required to perform a task.
Mistakes	Incorrect actions committed by the user while using the system, both the number of errors and their type.	The Sum of the number of errors committed by users while using the system or product.
Satisfaction Level	The user's opinion about the system, considering the criteria of each user.	The users' assessment of their experience in the use of the product.

Table 1. Main attributes within Usability Engineering.

or products takes into consideration specific attributes that one can always quantify from time records and testimony of the users who tested the operation of the product—see Table 1. On the other hand, the usability evaluation methods are diverse, some of which we can mention are:

- Survey: It is applicable at the initial stage of development, useful for obtaining the requirements of the product.
- Field observation: Tests analysis of the tasks performed by the end user, in the final stage and during the implementation of the product.
- Focus groups: This method applies to the design and requirements analysis stages.
- Heuristic evaluation. It is carried out by experts in the field, to apply at the beginning of the design, during the development and before the start-up.

The methods here adopted to perform the evaluation depend on the attribute to assess. Besides, each method used in the evaluation, generally, involves the active participation of the client or end user; this because it is vital to consider their opinion about the corrections to implement in the product that they are going to use.

3.2 Security

According to [23], information security is "the protection of information and its critical elements, including the systems and hardware that use, store, and transmit that information." It is also commonly referred to as the protection of confidentiality, integrity, and availability of the components of an information system [14].

An important consideration about developing web and mobile applications is the information security aspect. In general terms, every web application requires, at least, to provide integrity and availability to the information it manages. In some cases, confidentiality is also a desirable requirement. Providing security to applications in such an environment is not an easy task, especially for using a public network, such as the Internet.

3.3 Logic Programming and Answer Sets

One of the main foundations of this proposal is Answer Set Programming, ASP, characterized in several kinds of non-classical logic, with a long background and suitability to represent non-monotonic knowledge. Its main applications in problem solutions range from typical AI toy examples to agent prototypes and planning settings. Another name to identify this semantics from the literature is *Stable Model Semantics* or simply SM for its name in the original paper [11]. In addition to that, [16,18] introduced the use of ASP solvers as a new programming paradigm, though.

The following introduces a general description of ASP, which receives other names as well like *Stable Logic Programming* or *Stable Model Semantics* [11] and A-Prolog. In this paper, we assume the reader is familiar with the rest of its syntax and semantics, easily accessible in the literature. We introduce some basic syntax, though, to get some context of the apps.

Its formal language and some more notation are introduced from the literature as follows.

Definition 1 (ASP Language of logic programs). *In the following $\mathcal{L}_{ASP}$ is a formal language of propositional logic with propositional symbols: $a_0, a_1, \ldots$; connectives: "," (conjunction) and meta-connective ";"; disjunction, denoted as "|"; "$\leftarrow$" (implication, also denoted as "$\rightarrow$"); propositional constants "$\bot$" (falsum); "$\top$" (verum); "*not*" (default negation or weak negation, also denoted with the symbol "$\neg$"); "$\sim$" (strong negation, equally denoted as "$-$"); auxiliary symbols: "(", ")" (parentheses). The propositional symbols are also called atoms or atomic propositions. A* literal *is an atom or a strong-negated atom. A* rule ρ *is an ordered pair* Head $\leftarrow$ Body*, where* Head *is a set of literals and* Body *a set of literals and default-negated literals. Either of the two sets may be empty.*

Strong negation "∼"—also called *explicit negation*, *constructible falsity* and *classical negation*—in logic programs has the following meaning with respect to the default negation "not": a rule $\rho_0 \leftarrow \text{not } \rho_1$ allows to derive ρ_0 when there is no *evidence* of ρ_1; while a rule like $\rho_0 \leftarrow \sim\rho_1$ derives ρ_0 only when there is an *evidence* for $\sim\rho_1$, i.e., when one can prove that ρ_1 is false.

With the notation introduced in Definition 1, one may construct clauses of the following general form, well known in the literature.

Definition 2 (Extended Disjunctive Logic Program, EDLP). *An* extended disjunctive logic program *is a set of rules of form*

$$\ell_1 \vee \ell_2 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m, \text{not } \ell_{m+1}, \ldots, \text{not } \ell_n, \tag{1}$$

where ℓ_i *is a literal and* $0 \leq l \leq m \leq n$.

Naturally, an *extended logic program* (or ELP hereafter) is a finite set of rules of form (1) with $l = 1$; while an *integrity constraint* (also known in the literature as *strong constraint*) is a rule of form (1) with $l = 0$; while a *fact* is a rule of the same form with $l = m = n$. In particular, for a literal ℓ, the *complementary literal* is $\sim\ell$ and vice versa; for a set $\mathcal{M}$ of literals, $\sim\mathcal{M} = \{\sim\ell \mid \ell \in \mathcal{M}\}$, and $Lit_{\mathcal{M}}$ denotes the set $\mathcal{M} \cup \sim\mathcal{M}$; Additionally, given a set of literals $\mathcal{M} \subseteq \mathcal{A}$, the complement set $\overline{\mathcal{M}} = \mathcal{A} \setminus \mathcal{M}$.

The well-known *semantics of an EDLP* consists of reducing general rules to rules without default negation "¬" because the latter get interpreted in *classical logic* by using the well-known *Herbrand models*. In particular, the reduced rules with no default negation Mon of a rule of the form (1) is

$$\ell_1 \vee \ell_2 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m, \tag{2}$$

where ℓ_i are literals and $0 \leq l \leq m$. Note that a literal is either an atom or a strong-negated atom. The latter becomes a new distinguished literal as reduced rules towards the Herbrand models, where complementary literals cannot be elements of the same Herbrand model.

This kind of rules is the *monotonic counterpart* or *positive program*, also known in the literature. Additionally, the monotonic counterpart of a set of rules is the set of the monotonic counterparts of its rules.

Intuitively, the monotonic counterpart is where ASP can coincide with *classical propositional logic*. On the other hand, *default negation* is precisely the main difference between the two systems—alternately with Prolog too. As a result, the corresponding derivation symbols, "←" for ASP and "⊃" for Classical Logic, cannot have the same meaning.

On the other hand, all stable models are the *minimal Herbrand models* of a set of first-order sentences, but not the converse. Additionally, $\mathcal{S}$ is a *consistent answer set* of a given program $\mathcal{P}$ if it does not contain a complementary pair of literals.

Although we have introduced ASP as propositional (*ground*) programs, fixed *non-ground ASP-programs* of arbitrary *arity* are also common in the literature. Accordingly, non-ground ASP-programs with variables or constants as *arguments* are also simpler expressions of more extended ground (propositional) ones without variables, where

each *ground program* $\mathcal{P}$ is a set of its *ground rules* $\rho \in \mathcal{P}$. Besides, a ground rule is the set obtained by all possible substitutions of variables in ρ by constants occurring in $\mathcal{P}$.

Last, one may not conclude this section without mentioning that two significant *advantages of* ASP over other approaches. ASP has been hard work in research both on its *declarative programming framework*, and at least on three efficient, competitive *proving solvers* with broad backgrounds: SMODELS [19], Clingo [21] and DLV [5,15,2], which are available to run online[3]. That means they can run on their server and thus, there is no need to download and locally install the binaries or sources.

This section introduced some basic concepts needed to underpin our proposal. They are a set of principles to satisfy by our end-product applications, and they included methods on how to assess such characteristics, as well as a very general ASP background.

4 Design

Our proposal consists of the design, implementation, and development of a web application and a mobile app for Android devices. Both products should exhibit the ability to function as an online tool for researchers and interested users of logic programming. With their implementation, we put forward a complete suit that meets specific non-functional requirements, such as: Striking and dynamic graphical user interface; intuitive operation; adaptable and scalable content; security mechanisms that do not interfere with usability The web application is a complete regular browser solution, and the mobile app is an exclusive solution for basic tasks, in circumstances where the user needs to calculate a quick deduction on the server side, with the potential to other future helpful features a desktop application lacks. Namely, the mobile app is an open door for new and more robust applications, where not only does it work for small knowledge bases and immediate straightforward deductions, but also as a massive data collection interface suitable for knowledge representation and reasoning. Such data collections would range from temperature, movement, pulse, geographical location, speed, to other more complex data, such as habits, preferences, and so forth. The development of a mobile application and a web application with consistency between the content and the way users interact with it may seem unnecessary. However, its development shall introduce a new, more practical alternative for future users and research and development, as above suggested. Therefore, our development focuses on issues such as portability, usability, exclusivity, and security; current modern features that ease software operation and get to generate comfort to users.

5 Implementation

The primary operation of both applications consists of the use of a command-line text-mode solver. As a result, it is essential to model the way in which the user can interact

[3] Their respective online running versions are available at:
`http://logic-lab.sourceforge.net/` which are (graphical) front-end web interfaces to the originally implemented engines mentioned above.

with each of them. For example, if a user needs to use the DLV solver, they must first communicate with the interface, then with the application, which requests the solver. The processing involves a link with the communication line that links each of the parts that make up the proposed product. In addition to that, *modeling* means that each solver works as a black box, which accepts an input and returns an output. Each procedure that comprises a functionality in the project means requests to the service. That is why, for this flow of requests and responses, we need a controller that acts as a mediator between the view and the model. Then, it is from this modeling that the idea of using the controller view model (CVM) scheme to handle the data arises in detail. This scheme organizes the process of coding or construction of the product. Besides, it facilitates the detection of errors in the flow of requests and responses by the server or model. This modeling, as its name suggests, it is divided into three main parts, which we can describe below:

- Controller: it works as an intermediary between the user (client) and the server. It controls the flow of data and transmitted requests in the communication line. That means that if the user requests to use a particular solver, the controller makes a direct request to the model defined for the handling of such ASP solver. Additionally, it sends the necessary data for processing in the selected solver.
- Model: contains the execution lines of the inference engines and the database. In the case of the latter, we can say that, with the help of the model, the controller can make a CRUD request (*create, update, delete*), with which it can create a new user, update data from an active one or delete a user registered in the system. For the development of the applications that make up this project, we introduce two models, one that controls the sessions and the other that controls the ASP solvers.
- View: It returns the result of the requests made to the model. That is to say; first, the controller sends a request to the model, and the model returns a response, immediately the controller updates the view, and the user can view it.

5.1 The Back-End

The documentation of DLV and CLINGO provides valuable information towards a front-end and reveals that to process logic programs, it is necessary to know how to use a command line terminal at least, for Linux/UNIX in our case study. For example, suppose the following simple ASP knowledge base[4], $\{a.\ b :\!-\ \text{not}\ \sim c.\}$. To get the corresponding inference, one needs to send such program to the required solver through a pipeline. A pipe is a means that accepts an entry as input and sends it out to the subsequent command, in the pipeline, and which in turn is delimited by the use of the character pipe "|." Then, the process of the logic program $\{a.\ b :\!-\ \text{not}\ \sim c.\}$ are expressed as follows:

```
echo "a. b:- not ~c." | ./dlv.bin -- 2>&1 | awk '{print $0}'
```

[4] For the sake of economy in the pipeline command, we introduce an elementary ASP program, which does not mean the method cannot accept thousands of program lines.

where the command `echo` represents the escape of an output, which in this case refers to the logic program in text mode. Such output goes through the pipeline to the DLV solver, represented by the binary file called `dlv.bin`, in this case. The solver takes the output of the previous command `echo` in the pipeline as input and produces a new output, which is the derivation of the logic program. In turn, `awk` takes such output as input and gives it format if required. In this case, `$0` does not affect.

The description of this command line is an integral part of our proposal because it is the central connexion to the backend, and programming languages such as PHP allow the execution of this type of commands through the front end. This type of front end is possible, in PHP, through the use of the *shell_exec* function. We describe its general use as follows:

```
string shell_exec (string $cmd)
```

The function accepts commands in *string* format, that is to say as a string of characters. Therefore, the implementation of an inference engine in a graphical environment only requires the use of this function. It is worth noting that the execution of the logic sentence is in a slightly different syntax, without changing its semantics.

```
shell_exec ("echo 'a. b:- not ~c.' | ./dlv.bin -- 2>&1 '' | awk '{print $0}'")
```

Then, we can conclude that, by employing the previous command, it is easy to implement inference engines such as DLV or CLINGO in a graphical environment—web or mobile—the first phase of an online logic lab. Therefore, the primary operation of the expected product is covered using this type of implementation.

5.2 Connections with the server and the database

A fundamental part of this development is the connection of the web and mobile application, which gets done through the implementation of the client-server architecture. In the first place, it is essential to specify that the client's function gets fulfilled by the application (web or mobile), this is because it is the one who makes the requests and is waiting for the response. On the other hand, the server is responsible for offering accommodation to the PHP script or order file, in order to provide or satisfy the customer's request.

The most relevant requests are, in short, the use of a specific inference engine and the connection to the database in order to execute a CRUD action (Create, Update, Delete) with the information of both registered users and of those about to get enrolled. Previously, this paper introduces how ASP systems are implemented and work in applications. Now, it is necessary to explain how the connection with the database works.

The connection to the database uses the server as the host of the script responsible for the link to the database. Next, one can see this script or snippet of code needed to connect both applications to the database. It is possible to observe that such a connection requires the specification of the server's IP, the type of connection and the credentials of access to the database (user and password).

```
<?php
//Environment variables required in the connection to the database
define("DSN","mysql:host=localhost; dbname=users");
define("USER","root");
define("PASS","**********");
//function to connect the applications with the database
function connectionPDO(){try
    {$base=new PDO(DSN,USER,PASS);
        $base->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        $base->exec("SET CHARACTER SET utf8");
        return $base;      }
    catch (Exception $e){  die("Error: ".$e->getMessage());  }}?>
```

In summary, the implementation of a database allows not only the administration of users but also the synchronization of data between both applications. One should note that the fact that an automatic update simultaneously occurs is an important feature currently in the development of applications.

5.3 LogicSite Laboratory Front-End: web and mobile application

We mainly base the functionality of the applications on the integration of ASP systems. We devote particular interest to the web application, that is why we propose an intuitive interface. We have ordered the set of elements that are available to the user so that they are at their hand. As we can see, the improvements within the interface, in comparison with other existing projects, is notorious, starting with the options bar. On such options bar, we can locate the tools of primary use in the execution of the inference engines.

On top of that, the web application integrated new features that provide greater control and *accessibility* for the user, understanding accessibility as the ease of performing some action, as shown above. Among the most notable features are:

- Syntax auto-completion in reserved words: for the development of this part we have employed an API[5] called bootstrap-suggest[6], a plug-in that allows suggestions to be displayed as the user writes inside a plain-text area. This add-on allows one to enter all those reserved words, typical of the DLV and CLINGO systems, with which such API searches within one's knowledge base a match to what the user is writing. Its operation starts from the detection of the character "#"; once the system locates this element, it displays a list of possible options for the auto-completion of the text in question.
- Detection of the error line: it is common that during the process of writing a logic program, users commit syntax errors, whether a particular character is missing or a misspelled specific reserved word. The error detection works using the output provided by the execution of each solver; so that through the result of the processing, the solver generates the error number and the code line in which it shows up; it is immediately highlighted, making the user's work agiler.

[5] API *(Application Programming Interface)* is a set of rules (code) and specifications that facilitate human interaction-*software*.

[6] The bootstrap-suggest API is available through: `https://github.com/lodev09/bootstrap-suggest`.

– Self-saving: we include a saving function for registered users. This action is carried out automatically as the user writes on the text area, in this way their progress gets stored in a database identified by a unique registration number for each user. Therefore, at each login, registered users can see their latest ASP activity they were working on, making it possible to continue with the writing of logic sentences of their preferred solver. Once the system starts up, it updates the values of the database contents as the user writes, and during the login session, it queries what there is in the database.

Other noteworthy aspects of the web application are navigation within the webpage. Users can always choose to download and use the IDEs of the ASP solvers, download the mobile application, or enter as a registered user and send their ratings and comments about their experience using the system.

Finally, let us introduce the development of the mobile application having the functionalities described below:

– It offers the use of DLV and CLINGO inference engines in a graphical interface.
– It is possible to register and validate users.
– The password reset function we implemented takes into account that the user can forget the password with which they got registered and, therefore, lose access to their account. Therefore, this feature supports the user in the process of recovering their account.
– The import and export functions of files in plain text are a fact. This functionality allows working with external files so that the user can upload ready-made programs, or use a text editor outside the application and load the modifications made for future processing.
– Detection of syntax errors (by pointing out the error line), by writing suggestions on the reserved words registered out of the knowledge base.
– Reset functions, to clean the contents of the work area and start from scratch, as well as an auto-save of the current status of the work of registered users.

Based on this list of functionalities, it is possible to notice that options not considered before are now part of the system and that today they are an innovation. The implementation of error detection, session management, and password recovery offer a beginning for future innovations not only for an online logic lab but other services and applications too.

5.4 Application of Usability Engineering Techniques

The integration of usability metrics in this project is one of the most important factors considered within the objectives. That is why, when developing the applications, we started establishing which usability metrics to implement. Therefore, to decide the distribution of each of its parts, the adequate colors, the symbology, and the content, the following Usability Engineering metrics were taken into consideration:

Location context is applied to control small viewing spaces and prevent the user from getting lost easily. For example, in the applications, we examined the information, and the display extent we counted on, to determine the correct order and the way in which it should get visualized.

Less is more this technique is used to control that the information contained is useful and discard what is unnecessary. Within the applications, we avoided the saturation of information in order not to generate confusion to the user and even to make sure that the intuitive design prevails.

Cognitive load and visualization metric used to control the cognitive load in users, that is, users should not memorize a large number of steps during their interaction with applications. We designed the applications taking into consideration that the execution and selection of the solver and the import or export of files, need to be direct without requiring an extra effort to the user.

Block design structure plays a role in the distribution and presentation of the content to provide an organized interface. For example, in the case of the web application, we developed an options bar that organized the content corresponding to the execution of the selected solver and the functions for itself. In the same way, we examined the rest of the graphical interface for both applications to structure all the content and improve the visualization.

Precise color and symbology implemented for the use of universal symbols associated with the action or information that one wants to convey to the user. In the same way, the colors selected for the interface must be precise and not affect the visualization within the applications. We applied this metric universally, that is, in both applications, we kept the same symbology to avoid confusion among the users, and the same happened with the designation of colors in the interface.

The implementation of these usability techniques is not exhaustive. When including them, there is an innovation concerning the user's experience with both applications. Therefore, these first steps refer to specific milestones of computer-human interaction, especially on ASP terminal-mode solvers.

Throughout the development process of this proposal, it has considered particular features or functions implemented in other web applications. Also, new features were added to improve the user experience within the laboratory. Thus, each of these aspects was analyzed initially, with the intention of identifying its strengths and weaknesses, implementing then those characteristics or functions that would strengthen this project.

5.5 Security Mechanisms

Since individual data used in the proposed applications are indeed sensitive, and because of the in-security of the means of communication used by both web and mobile applications, it is imperative to implement security mechanisms that protect such data from intrusions or alterations. It is also important to note that cloud and web storage platforms depend on third parties, so it becomes nec-essary to provide security to the applications that run on those environments. Some of the data that are considered sensitive in the proposed applications, and prone to protect, are the programming codes generated by users. The access passwords to the applications and the data obtained through mobile devices are to get protected as well. Some examples are biomedical parameters of a patient and any other data that may acquire in the future as input to a module made on the platform. The following describes the security mechanisms proposed for web and mobile applications:

User/Password access control. The username and password used to login on the application are not only to control who can or cannot access the application and its resources. It also allows users to retrieve their work previously developed and saved in the platform. The access control also prevents intruders, or even legitimate users, from accessing the work of another legitimate user. A username and password shall provide a correct balance between security and usability.

Password encryption. Once a new user performs the registration process or login, the password shall get encrypted (ciphered) using a one-way or hash function—in this case, the named SHA-1—and then the system shall store it into the database. Such encryption shall prevent any person, even database administrators, from having access to the password in clear-text mode. In the login process, once the user types in their password, the same function used in the signing process shall calculate the hash value and compares it to the stored value for that user.

Sensitive data encryption. As previously mentioned, there are sensitive data that the platform should protect from intrusions. Because encrypting all the information generated and acquired by the applications reduces the computing performance, only the data that the user considers to be sensible shall get encrypted. This principle also applies to the ASP programs that the user writes on the plat-form. The data shall get encrypted at the database level; that is when they are going to store the ASP programs. This task primarily protects the confidentiality of the data.

The digital signature of ASP source codes. The ASP programs written by users are the most crucial asset on the platform. The integrity of these programs is, therefore, a priority in the proposed applications. In order to protect their integrity, they shall get a digital signature each time the user ends a session, through a public key system. To carry out the digital signature, each user shall hold a public/private key pair. The private key is in a container protected with the user's password. Because the user enters their password to log in, the system shall enable the latter to digitally sign the ASP pro-gram(s) before the logout process completes. As soon as the user's ASP programs get stored into the database, the digital signature for each of the programs shall get also stored. In this way, when the user starts a new session, using their public key, the system shall perform an automatic comparison of the stored program or programs of that user with the digitally signed versions. This task would detect if an intruder modified an ASP program. The digital signature process, as well as the verification of the signature, shall be transparent to the user. The system should request the intervention of the user only if it detected that a program does not match with the last digitally-signed version.

6 Conclusions and Future Work

The contribution of this work consists of the design of a web application and a mobile app to implement the DLV and CLINGO systems, in an online friendly graphical server-side front end. That is, taking into account the fact that, according to our research, there is a limited number of web applications that provide such service and there is no evidence of the existence of any ASP mobile app. In addition to that, existing apps are limited exclusively to offering the use of such unpolished systems, without seriously

considering the user's experience. The conclusions that we get from this research and development work of software are the following. In the first place, we can affirm from the implementation of the ASP logic lab system prototype, that the applications developed throughout this project allow not only to show the inferences from DLV and CLINGO but also the manipulation of such output to become the input for future advances in the growth of the logic laboratory. Such is the case of the extraction of the output for the detection of errors, in the input of logic sentences, which is a fact in this project. Likewise, the session system designed for both applications perform a correct and safe validation of the users in order to offer new functionality that until now had not been considered in the use of ASP solvers and applications. That functionality consists of the self-preservation of the logic clauses written by the users during their visit to the laboratory to get continued. Therefore, the system maintains the users' work as they left it so that they do not have to worry about having abandoned (and lost) their activity.

In the same way, this session system shall allow maintaining synchronization between the applications, that is to say, if, during a short time the user starts writing in a logic program in the mobile app, their work goes reflected in the web application to get continued. The design and implementation of this function give rise to future advances for the user's version history so that not only could they have access to their current work but also to resume or restore logic formulas previously deleted by themselves. Finally, we state the fact that the development of this project opens the possibility of more future research and development. It is worth noticing that our proposal is just a beginning for what, in a particular moment, could be a fully-fledged laboratory for users interested in mathematical logic. That is, with the code lines of the development of this research we introduce a new alternative for the use of ASP solvers and systems, by integrating sessions management to control an auto-save function and synchronization between applications. However, in a future proposal, not only can the user experience be reconsidered but also the manipulation of the resulting inferences for novel applications, such as in sports medicine and health, the administration of services and the management of massive or large-scale data—big data. As we can see, the manipulation of the information collected by these systems offers considerable possibilities in different areas (health, education, and entertainment), which are individually exploited for a long time already, previously mentioned in this research. In short, this project centered on changing the user interface of ASP solvers and systems, acknowledging that the task of the user should focus more on the ASP language rather than on the difficulty of downloading, compiling, installing and running the solver or system itself.

References

1. Acosta Guadarrama, J.: Logic Lab (2016), http://logic-lab.sourceforge.net/
2. Alviano, M., Faber, W., Leone, N., Perri, S., Pfeifer, G., Terracina, G.: The Disjunctive Datalog System DLV. In: First International Workshop, Datalog 2010, Oxford, UK, March 16–19, 2010. Revised Selected Papers. pp. 282–301 (2010). https://doi.org/10.1007/978-3-642-24206-9_17, https://doi.org/10.1007/978-3-642-24206-9_17
3. Arieta, V.: GuessAndCheckers (2016), https://github.com/vincenzoarieta93/GuessAndCheckers
4. Calabria, U.: DLV (1997), http://www.dlvsystem.com/dlv/

5. Calimeri, F., Dell'Armi, T., Eiter, T., Faber, W., Gottlob, G., Ianni, G., Ielpa, G., Koch, C., Leone, N., Perri, S., Pfeifer, G., Polleres, A.: The DLV System. In: Flesca, S., Ianni, G. (eds.) Proceedings of the 8th European Conference on Artificial Intelligence (JELIA). Springer, Cosenza, Italy (September 2002)
6. Calimeri, F., Fuscà, D., Germano, S., Perri, S., Zangari, J.: EMBASP (2015), https://www.mat.unical.it/calimeri/projects/embasp/
7. Calimeri, F., Fuscà, D., Germano, S., Perri, S., Zangari, J.: A Framework for Easing the Development of Applications Embedding Answer Set Programming. CoRR **abs/1707.0** (2017), http://arxiv.org/abs/1707.06959
8. Campisano, D.: DLVfit (2015), https://github.com/brainatwork/DLVfit
9. Garcia-Mata, C., Marquez, P.: Answer Set Programming y el Problema de Asignación de Frecuencia, FAP. ELECTRO (2009)
10. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo = ASP + Control: Preliminary Report. Tech. rep., Aalto University, Finland; University of Potsdam, Germany (2014), http://arxiv.org/abs/1405.3694
11. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Kowalski, R.A., Bowen, K.A. (eds.) 5th Conference on Logic Programming. vol. 88, pp. 1070–1080 (1988)
12. Germano, S., Calimeri, F., Palermiti, E.: LoIDE: a web-based IDE for Logic Programming Preliminary Technical Report. CoRR **abs/1709.0** (2017), http://arxiv.org/abs/1709.05341
13. Germano, S., Palermiti Eliana, C.F.: LoIDE (2016), https://github.com/DeMaCS-UNICAL/LoIDE
14. ISO/IEC: ISCO/IEC 27002: Code of practice for information security management (2005), www.iso.org
15. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic **7**(3), 499–562 (2006). https://doi.org/http://doi.acm.org/10.1145/1149114.1149117
16. Marek, V.W., Truszczynski, M.: Stable models and an alternative logic programming paradigm. In: V.W., M., M., T., D.S., W. (eds.) Artificial Intelligence, chap. The Logic, pp. 375–398. Springer, Berlin, Heidelberg (1998). https://doi.org/10.1007/978-3-642-60085-2_17, http://link.springer.com/10.1007/978-3-642-60085-2_17http://arxiv.org/abs/cs/9809032
17. Nelsen, J.: Usabilidad, diseño de sitios web. Prentice Hall, 1 edn. (2000)
18. Niemela, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25**(3/4), 241–273 (1999). https://doi.org/10.1023/A:1018930122475, http://link.springer.com/10.1023/A:1018930122475
19. Niemela, I., Simons, P.: Smodels—an implementation of the Stable Model and Well-Founded Semantics for normal logic programs. In: Proceedings of the 4th LPNMR ('97). LNCS, vol. 1265, pp. 420–429. Springer, Dagstuhl Castle, Germany (1997)
20. Peinado Portillo, M.A.: Modelado de Decisiones Sesgadas en Answer Set Programming. In: Acosta-Guadarrama, J.C., Rodas Osollo, J., Ramírez-Bouchot, M. (eds.) RCCS. pp. 1–8. CEUR, ISSN 1613-0073 (2016), http://ceur-ws.org/Vol-1784/99990001.pdf
21. Potassco, U.P.: Clingo (2016), https://potassco.org/clingo/
22. Potsdam University, P.: Running Clingo (2017), https://potassco.org/clingo/run/
23. Whitman, M.E., Mattord, H.J.: Principles of Information Security. Course Technology Press, Boston, MA, United States, 3rd edn. (2007)