WIMS
A SERVER FOR INTERACTIVE MATHEMATICS ON THE INTERNET

XIAO, GANG

ABSTRACT. We describe in this paper the project "WWW Interactive Mathematics Server", designed for supporting intensive mathematics works via the internet with server-side interactivity, accessible at the address "http://wims.unice.fr". The following main features of the system are described:

1. A modular design allowing applications and software interfaces to be created and maintained independently from each other.
2. Existing interfaces for softwares including MuPAD, PARI/GP, Gnuplot, Povray, Coq (proof assistant).
3. Dynamic rendering of mathematical formulas and animated graphics.
4. A structure of virtual classes, including mechanisms for automatic score gathering and processing.

Design principles of the system are discussed, including session control, security measures and anti-cheating mechanisms.

INTRODUCTION

We describe in this paper our project WIMS (WWW Interactive Mathematics Server), a project aimed at a systematic approach of providing internet-accessible mathematical computations.

Internet-accessed mathematical computations present several advantages over locally installed mathematical softwares. With interfaces based on the html protocol, many common computation needs can be carried out by the user without prior learning of the syntax of a particular software. And software installation and maintenance costs are considerably reduced.

User-friendly access to mathematical computations also means the design of computer-generated mathematical exercises with sophisticated analyses of answers. When it is an Internet server which generates the exercises and processes the answers, analyses of students' progress can be easily achieved, with data and results well-protected against cheating, a feature otherwise hard to realize by locally installed softwares.

In order to obtain a high degree of mathematical sophistication, the basic design concept of the Wims project is server-side interactivity via http protocol, with a modular structure at two levels, which make the system easily enhanceable.

Date: 2 June 1999.

On the one hand, each application (exercise or tool) under the system can be independently created, modified or removed, allowing the system to host a large number of such applications. On the other hand, the system calls various mathematical softwares (numerical or symbolic computation, visualization, proof assistant) as background engines, with an independent interface designed for each such software. This provides an easy means for Wims applications to make use of the power of these softwares in a very versatile way.

As a result, we are able to create applications with unique features for web-based mathematics (interactive theorem proving, animated visualization for user-supplied expressions, exercises with multiple good answers or multi-step answers, etc). And in a few domains such as first year linear algebra where enough applications have been developed, the system is ready for serving mainstream educational use.

Design choices as well as capabilities, shortcomings and unsolved problems of the system are discussed in the paper. Most technical details are omitted in order to keep the paper within a reasonable length and readability. Interested readers can refer to [Wimsdoc] or the source code for these details.

A few other topics are also omitted, such as facilities for user contributions.

1. Basic design principles

WIMS is designed for server-side interactivity on internet applications through http protocol. Client-side interactivity via javascript or java applets can be incorporated into Wims applications, but they usually play a secondary role.

This server-side interactivity is specially designed to support mathematical applications. It allow the use of different mathematical softwares in Wims applications.

The main objective of the project is to build a powerful system for mathematics education on the Internet. Because of the importance of practice in mathematics learning, emphasis has been put on online exercises. The system allows mathematical softwares be used either for the generation of exercises (for example the generation of random unimodular matrices of different sizes in [Visual Gauss]), or for the analysis of user answers (for example PARI/GP is used to compute function composition in [Decomp] where the solution is not unique). Softwares originally designed for local interactive use can also be adapted to power step by step exercises, for example the use of Coq (see Section 3 for detail) in [Logicoq].

Mathematical softwares are also used to power online computational or visualizational tools, which are designed to provide online computational helps for the exercises. But these tools can also be used as stand alone ones.

These online tools on Wims have no vocation to replace traditional use of mathematical softwares. The latter is designed for heavy users to carry out complicated computations via programmed scripts, while the former is intended to 'casual' users who only want to carry

out 'simple' computations, so that it is often not cost-effective for them to learn a whole syntax set of the software for just a few such computations. In fact, for security reasons (see Section 6), we avoid letting the user to directly type in source lines of a computational software via Wims. The security measure of the system also disallows computations requiring a long execution time.

On the other hand, web user interfaces can be made much more user friendly for the computations they support, and it is possible to create applications on Wims combining the computing power of several different softwares.

Of course, this design goal of Wims does not stop people from using it otherwise. For example, we have regularly seen people spending hours or even days on [Factoris], to factor series of integers such as $n^m + n - 1$, for n going from 100 to 1000 and m from 2 to 10. It is clearly not the intention of Factoris to support such activities, as a direct use of [PARI/GP] with a small loop would give them the result within minutes. However, the fact that these people prefer spending so much time on Factoris indicates that they would have to spend more time and effort finding, installing and learning to use PARI/GP directly. Therefore we are not discouraging people from using Factoris (and other online tools) in such an intensive way.

1.1. **Basic structure of the system.** The center of the system is a cgi program, wims.cgi, written in C. It serves as the unique entry point for all user requests, and directs these requests to the appropriate applications.

User requests arrive at the web address of the program wims.cgi, via usual http protocol. These request may contain parameters which will be captured and interpreted by wims.cgi.

Each application under the system is an independent unit, called MODULE. Modules are written in a proprietary scripting language, to be interpreted by wims.cgi. Inter-module communication is achieved via usual http links.

Many capabilities of the system are available to modules via special commands in this scripting language, so that these capabilities can be enhanced or upgraded without modification at the modules' level.

Each module under Wims occupies a separate directory, and contains a number of files for different purposes, written in the scripting language. A usual user request to Wims should contain information for the name of the module requested. wims.cgi will then read the files in that module's directory, and interpret them. The result is an html page generated by wims.cgi, and sent back to the user.

Our principle is to let all sophisticated mathematical computations be carried out by exterior programs, in order to gain maximal power and versatility with minimal cost of development. With this consideration in mind, wims.cgi can only carry out very basic computations by itself. When needed, the module's writer can use a special command to execute an exterior program for mathematical computations (or visualizations). wims.cgi will then call an interface to this

program, which prepares the parameters, executes the appropriate program, and filters the output of the latter to make it directly exploitable by subsequent commands in the module.

The number of programs which can be executed to process one user request is limited only by the server resources. For example, in the module [Factoris], a user-submitted expression is first sent to PARI/GP for a check of syntax consistency. Then the number of variables in the expression is computed. For integers or one-variable polynomials, PARI/GP is executed again to factor it. For multivariate polynomials, MuPAD is called for factorization.

Unlike mathematical computations on locally installed softwares where a program is launched for an interactive session and stays alive until the user ends the session and quits the program, programs launched by Internet accesses must terminate at the end of every process of user request. The main reason for this is that Internet users never explicitly quit a site.

To some extent, the system behaves like a special operating system, with a user interface through the World Wide Web, a programming interface through the modules' scripting language, and interfaced softwares as the system's resources.

1.2. **Session control.** As is necessary for server-side interactivity, session control is implemented in Wims. At each request from a new user, a session is created, with a unique session number (a 8-digit hexadecimal code) assigned to it. A directory is created at the same time, which contains all the temporary files related to the session.

One of the problems for session control is that the server has no means to know whether the user has ended the session. The only solution we have found is to keep the session alive for a certain amount of time (2 idle hours; the delay is configurable), then remove it.

There is also no means for the server to determine whether a new user is really 'new', that is, whether or not he is simply forgetting to tell the number of the session he is working under, for various technical reasons. This is not a serious problem, as the only inconvenience is a few more idle sessions which will soon be removed anyway.

We have decided not to use cookies for session control, due to compatibility considerations.

Web browsers are not designed to support intensive server-side interactivity. In particular, earlier pages stored in the browser's cache (or by proxy servers) may create considerable confusion for exercises with random parameters. There are two possible solutions to this problem.

1. Add a unique tag to the address of each page generated by the server, in order to make the browser see different addresses at each time.

   There is a (slight) ethical inconvenience of this method, due to the fact that it 'forces' the browser to uselessly store all the pages in its cache. Also, confusion will be created when the user uses 'Back' button of the browser to go to an earlier page stored in the

cache, then issues a (new) request to the server by following a link contained in that page.

This method must not be applied when the server is contacted by an Internet robot, for otherwise the robot would never be able to stop by itself.

2. Include a line

<META HTTP-EQUIV="expires" CONTENT="1 Jan 1990">

in the header of the html page, to tell the browser not to use the page stored in the cache.

There are two inconveniences. Each time the user resizes the browser's window, a new page is requested, which may be very annoying if the user is working on an exercise with random parameters; and if the user clicks repeatedly on the 'Back' button of the browser to try to quit Wims, he generates a series of useless queries to Wims, wasting the server resource.

Wims currently adopts the second solution. We have a partial solution for the 'Back' button problem (simultaneous requests generate an error message), but no solution is found for the window resizing inconvenience. And we are weighing the pros and cons to see whether it is worthwhile to switch to the first solution.

1.3. **Error management.** Wims deals with error situations at 4 levels. At the lowest level, an unrecoverable panic generates internal error, with wims.cgi sending nothing through the Internet but registering the error on its log file.

If a bug is found in the module's script, a module-bug error is generated. An error message is sent to the http requester (user) in an unformatted manner, indicating the name of the file and the number of the line where the error is found. This is used to let the developer of the module to debug it through the Internet, without having to log on the server.

More commonly, there may be many module-independent errors in the user's request, ranging from unmatched parentheses, to invalid query string, to requests for non-existing modules, etc. These are user errors which generate a well-formatted error message indicating the nature of the error and including links for the recovery.

Finally, each module can freely define its proper error situations, and incorporate a mechanism to deal with these situations. For example, it occurs frequently that a user submits an equation where an expression is required, or vice versa. The module can detect the error by detecting the presence of the symbol '=' in the input field. In case where no guess is possible, the module sends an error message to the user, telling him to change the type of his input.

Wims is designed to support multi-language. All the error messages sent to the web interface is language-dependent, with the message's language corresponding to that chosen by the user.

1.4. **Availability.** The home site of Wims [Wims home] is freely accessible to all. An online documentation [Wimsdoc] can also be found there, where technical details such as the scripting language for writing modules are documented.

The source code of the system (server and modules) is downloadable from the home site, distributed under the GNU General Public License.

Although it is possible for anyone to work on a remote Wims server through the Internet, it is recommended that local installation be created if intensive work is planned, due to possibly important delay cause by network saturation. The installation of Wims itself is relatively easy (on a Linux operating system), but that of the support softwares is sometimes more complicated.

2. Capabilities of the system

2.1. **Input capabilities.** The types of user input Wims can accept is limited by the http protocol. They include multiple choices (menus), numerical data (real, complex or others), text data (equations, expressions, matrices), plane coordinates (clicking on a picture). Multiple input data can be contained in one user request, via the html form protocol.

The input of geometric objects other than points is not supported by ordinary html standard. Therefore this can only be achieved by text data or by using java applets.

The capability of Wims to process user input is only limited by that of the various softwares interfaced by it. In particular, unlike many other Internet exercise systems where user answers are merely compared with stored standard answers, answers to exercises under Wims are mathematically analyzed, providing a much higher freedom for exercise design. For example we are able to create exercises whose good answers are not unique, or those accepting multi-step answers. Sophisticated error analysis mechanism can also be built into exercise modules, helping users to understand the reason of their failures.

Most exercises under Wims incorporates random parameters (numerical, functional or configurational), so that they are highly non-repetitive. They are usually also configurable by several configuration parameters, intended to teachers for setting up the level of difficulty of the exercises for their students (see Section 5).

Users through the Internet are often not very strict on syntax rules, when they input mathematical expressions. For this reason, a special command (*rawmath*) is implemented in the Wims scripting language, which allows automatic correction of common syntax "errors" whenever there is no ambiguity about the intention of the user. For example, `2x` is corrected to `2*x`, `sint` is corrected to `sin(t)`, `(x-1)(x+1)` is corrected to `(x-1)*(x+1)`, etc. In ambiguous situations such as `x2+1` or `x^1/2`, a warning message prompts the user to enter `x^2+1` or `x^(1/2)`, if he wants to say x^2+1 or $x^{1/2}$.

The algorithm of *rawmath* is constantly improved according to the real behavior of the users shown by log files of the server. Without the use of this command, as much as 30% of expression inputs were rejected due to syntax errors. Under the current version of *rawmath*, syntax errors are reduced to less than 5% of all expression inputs, of which only a small fraction is attributed to 'failures' of *rawmath*, such as rarely used function names which are not recognized (for example csc).

Wims systematically checks all user input fields for consistency of parentheses, and prompts the user for correction when unmatched parentheses are found.

2.2. **Rendering of mathematical expressions.** Currently Wims has several approaches for this purpose.

Common mathematical symbols (such as π, β, $\sqrt{}$, $\leq$, $\int$) can be directly inserted into the text, using inline directives with a syntax similar to TEX.

The command *htmlmath* takes an expression and renders it as best as possible, using available html tags. It is fast, but rendering is not very good due to limitations of html language. $\frac{2}{3}$ can only be rendered as 2/3. It is intended to rendering simple expressions.

Much more powerful is the command *instex*, which takes a TEX source and compiles it to a graphics file which is inserted in the place where the command is. This command has mechanism to detect whether the source is constant or dynamic. For the former, the generated graphics file is stored in a permanent and session-independent place, and reused for subsequent calls, in order to gain speed. At each subsequent call, the file time of the graphics file is compared to that of the file containing the command, and the graphics file is renewed if the latter is newer than the former.

If the source is dynamic (that is, if it contains a variable to be substituted), *instex* generates a temporary graphics file stored in the session's directory, which is never reused for subsequent calls. This guarantees that the formatted expression will follow the changes of the context. In order that the browser does not reuse the graphics file stored in its cache, a unique time stamp is appended to the address of the graphics file.

For the time being, the performance of dynamic *instex* is not satisfactory. Perceptible delay occurs even for one dynamic *instex*. In fact, the mechanism of *instex* is to transform the TEX source into a dvi file, then to a postscript file via *dvips*, and finally to a gif file using *convert*. The last step calls *Ghostscript* to interpret the postscript file, and it is here where the bulk of the delay takes place. Performance can be greatly improved if we have a driver transforming the dvi file directly into gif.

Wims has also a mechanism allowing the user to change the font size of TEX related graphics (*instex* and inline mathematical symbols), in order to make it correspond to the font size of his browser. This mechanism is still rudimentary, in that for registered users, the choice of font size is not permanently stored (so that he should re-choose the font size at each login).

The MathML standard offers an extremely interesting possibility for mathematical rendering. Wims is prepared to support this standard, but this is not yet implemented because most currently used browsers do not recognize it.

Before we reach a stage where almost all installed browsers become MathML capable (which will not be the case before several years), the server must have a mechanism to detect MathML capable browsers, and send MathML codes only when this is the case. This can eventually be done by analyzing the name and version of the browser contained in the HTTP_USER_AGENT environment variable, but the ideal solution is that in the http protocol, there be a standard variable indicating whether the browser accepts MathML.

Once a MathML capable browser is detected, both *htmlmath* and *instex* can be made to output MathML code, allowing MathML support without modification at modules' level. What is needed here is to create routines generating MathML codes from raw mathematical expressions and from TEX sources (planned in the development project of Wims).

One may also create a single output command, which accepts a mathematical expression in its raw form, and automatically prepares an appropriate output (html, TEX, MathML) according to the circumstances.

We have yet to find time to implement a routine translating raw mathematical expressions into TEX source.

2.3. **Dynamic animated graphics.** Another particularity of the system is the capability to output animated graphics in a very convenient way. The command *insplot* accepts a raw mathematical expression, and insert a plot of the expression at the place where the command is. The expressions to be plotted can also include an animation parameter, in order to make *insplot* render animated sequences.

In a similar way, line graphics can be rendered via the command *insdraw*.

Wims has also an interface to Povray, allowing it to render ray-traced 3D graphics. See Section 3 for more details.

All these graphics insertions are dynamic. Static graphics can be inserted using ordinary html links.

2.4. **Simultaneous accesses.** The number of simultaneous accesses is limited only by the server resources. This limit varies according to the speed of the server and the network connections, as well as the nature of the modules accessed. It ranges from 2 to 3 for very intensive requests (such as animated graphics), to more than 50 for usual exercises with no intensive computation nor TEX formatted output, on a server equipped with a Pentium II-266 CPU. This indicates that with normally distributed usage, one server site powered by a reasonably fast computer can satisfy the teaching requirements for several hundred students.

3. Interfaces with background programs

In principle, any program allowing batch mode execution can be interfaced by Wims. The main restriction is that softwares with restrictive licenses should not be made freely accessible by all. Modules which must call restrictive softwares should put access restrictions on them. Partly due to the distribution policy of Wims, we do not have interfaces to such restrictive softwares yet, although many of them are easily interfaceable.

Wims calls interfaces to other programs by a command *exec*. This command accepts also parameters which will be sent to the interface as standard input to the program. The standard output of the interface is put into the return value of *exec*. Apart from this standard communication channel, variables can also be used to communicate with background programs in both ways. On the one hand, all Wims variables are visible by child programs as environment variables. On the other hand, child programs can define variables for Wims by writing their definition in a special file named *exec.var* in the session directory.

For security reasons, only interfaced programs can be called by Wims. The following interfaces are currently available.

1. [PARI/GP]. This is an excellent arbitrary-precision package for computations not involving transcendental functions.

   The header of PARI/GP output is automatically stripped off by the interface. For the rest, each line of the output corresponds to a line in the calling parameter.

   Often it is desired to put preliminary definitions or computations to PARI/GP before output is required. If the preliminary material is so long as to create difficulty or reliability to the computation of the output line numbers, this preliminary material can be put into a variable *pari_header*. The content of this variable will be executed by PARI/GP when it is called, but its output is stripped off.
2. [MuPAD]. This is a symbolic algebra package whose copyright holder allows this kind of use on the Internet (interfaces not giving direct access to the full power of the software).

   The functioning of the interface is similar to that of PARI/GP.

   This program is relatively slow to load. Whenever possible, we prefer PARI/GP which is much faster.
3. [Octave]. An preliminary interface for this numerical package exists, but no module uses it yet.
4. [Gnuplot]. This popular plotting program is accessible by Wims modules only through the command *insplot*.

   Several incompatibilities concerning mathematical expressions are fixed by the interface: for exponentials, '^' is translated to '**'. For divisions, integer fractions like '1/2'

are translated to '1/2.0' (otherwise Gnuplot applies integer division on 1/2 and gives out 0).

The interface includes adaptation making it possible to plot animated sequences, by adding a few extra parameters. This is done by asking Gnuplot to plot images of the animation one by one, then using convert to put these images together into an animated gif file. Performance is reasonably good, as animations with acceptable quality can be generated within a few seconds.

This program is included in standard Linux distributions. But the Wims interface assumes that it is recompiled with gif support enabled.

5. [Povray], a popular ray tracing package.

   Wims does not have a command like *insray*. So Povray should be called via the command *exec*. On successful execution, the interface returns the complete html syntax for addressing the resulting image file.

   Povray has native support for animation, but rendering is usually too slow to permit efficient animation within the execution time limit allowed to the program, unless the system is installed on a super-fast hardware.

   The application [Polyray] makes use of this package. As it seems that there is no danger allowing direct user-supplied source for Povray, Polyray has included a text field which allows the user to do so.

6. TEX. The interface to TEX is exclusively accessible via the command *instex*, whose principle of functioning is described in subsection 2.2.

7. [PostgreSQL]. This is the standard SQL engine used by Wims to manage its databases: applications on the site, virtual classes, user registrations. It is accessed by a special command *sql* which is only available for privileged modules.

8. [COQ]. This is a proof assistant package built upon CAML.

   The program is designed for interactive use. At each proof step, a status is printed including new goals to prove as well as hypotheses for the principal goal.

   A Wims module using this package should support step by step operations. It should store the whole command history in a variable, and at each step, the last command is appended to the command history, then the latter is fed to the interface for Coq.

   The interface captures the last status printed by Coq, and splits this status into two return data, one for the goals remaining to be proven, and the other for the hypotheses for the principal goal.

9. [Imps]. Interface for this mathematical proof system is still under development.

One of the problems which occurs when several programs are called in a single process is that each program has its particular syntax for mathematical expressions. For example, Gnuplot uses `x**3` to denote x^3, which is not recognized neither by PARI/GP nor by MuPAD. In order

to make the mathematical expressions directly chainable from program to program, in the respective filters, routines have been inserted which translates syntaxes used unambiguously by other programs into the syntax proper to the interfaced program. Similarly, function names or constant names which are commonly used but not recognized by the interfaced programs are defined in the interfaces (such as sh, ch, sec, tg).

4. Modules

Wims offers great freedom in the design of modules. In the simplest (but not very useful) case, if an ordinary html file, under the name main.phtml, is put into a directory, this directory will behave as a Wims module, and the html file will be sent to the user when this module is requested.

On the other hand, there is no limit to how a module can be complicated. main.phtml can call other files in the module; inline mathematical symbols and Wims commands can be inserted into these files; in order that the module be correctly indexed on the site, a file named INDEX should be created; auxiliary (static) graphics files can be included; variables which the user is allowed to define when requesting the module should be declared in var.def; etc.

The output of the module is defined by phtml files (phtml stands for 'programmable html'). The format of phtml is a proprietary extension to the html language, by allowing Wims commands and variable substitutions in the file.

In order to achieve maximal independence between Wims commands and html tags, the Wims command language is line-oriented. Wims only interprets lines whose first non-space character is the exclamation mark '!', considering the word following this '!' as a Wims command. All other lines are simply sent to the user, after eventual variable substitutions. This allows Wims commands to take place anywhere in the html result: within html tags, html header fields, http references, javascript scripts, etc. As a result, the interaction between Wims commands and html tags is very flexible.

Of course, the result sent to the user is a pure html file, with all Wims-proprietary codes replaced by their outputs.

Although it is possible to put all the processing of user requests into a phtml file, it is recommended that computational codes be put into variable processing files which have a slightly different syntax. This syntax is more convenient for variable processing. Such a distinction between computation and output also helps multi-language support: when a module is translated to another language, only phtml files need to be translated.

There can be two variable processing files: var.init which is interpreted at the module's initialization, and var.proc which is interpreted at every user request. An arbitrary number of other variable processing files can be present, to be called by the above two.

The quality of the modules is of crucial importance to the system, as it is by this that ordinary users judge the quality of the system. User interface (the html page output) should be designed to achieve good compromise between simplicity, clarity and versatility. Due to the particularity of users through the Internet, simplicity is very important, and ideally the first page of the module should directly lead the user to the main functionality of the module, whether it is an exercise or a tool.

In order to reconcile beginners who just need the basic functionality and advanced users who want to make more complicated computations or more detailed setups, many Wims modules implement a dual-menu mechanism, allowing the user to switch between a simple menu and an expert menu.

Special pages such as introduction, about, help and hint can be included in the module.

Wims has a scoring system for exercise modules. The module can send a score to the scoring system by simply assigning a value to the variable *module_score*. This value should be a number between 0 and 10 (10 for full success). This score can be used by the structure of virtual classes (see Section 5) to monitor the progress of each student.

Inter-module communication is achieved by usual html links: any module can include a link pointing to another. This is particularly useful when solving an online exercise requires computations which can be done by an online computational tool. In order that the exercise remains open when the tool is called, another browser window is opened to host the tool. A new session is also created for the latter.

This mechanism does not work perfectly, especially for unexperienced users. This is because the new browser window is often sized to exactly cover the original one, especially when the browser is in full-screen mode. And the user is often confused, thinking that the exercise is lost. An ideal solution seems not easy to find even with the use of javascript, because there is no way for the server to know what is the screen size of the browser.

5. Virtual classes

Wims is designed to support intensive classroom applications. To this end, it incorporates a structure of virtual classes. A virtual class under Wims can be created and managed entirely online by the teacher (supervisor) of the class. Students can then register into the virtual class, either by the teacher or by themselves (with the register mode definable by the teacher).

There are three modes with which users can connect to Wims: as anonymous visitor, as the teacher or as a student of a virtual class. The first is open access, while the other two are protected by passwords.

The content of a virtual class is a number of work sheets. Work sheets are created and maintained online by the teacher. Each work sheet contains a number of items. Each item is

the address of a Wims module, which may be an exercise (for which the teacher has determined the difficulty level), an online tool, an online lesson text, or other.

The design of the system requires that the teacher of the virtual class takes the responsibility of choosing the exercises for his students, as well as determining the difficulty level of the chosen exercises. For this purpose, pages sent to virtual class teachers contain special links. The teacher has only to click on these links, in order to add a chosen exercise into a work sheet. The difficulty level and configuration of the exercise added to the work sheet will automatically be set to that of the exercise containing the link.

When adding an exercise to a work sheet, the teacher can also set up the number of points each student is required to make on this exercise, as well as the weight of these points in the computation of averages. For example, by assigning more than 10 points to an exercise, the teacher can ask his students to work more than once on this exercise.

Example work sheet will also be created, which can be directly inserted to virtual classes, helping teachers to make their choices.

When a student logs in to the virtual class, he is presented the list of active work sheets. By working on the exercises contained in a work sheet, the score he obtains is automatically gathered by the server. However, points exceeding the assigned number are discarded by the system, preventing the student from gathering extra points by repeatedly working on one (easier) exercise.

Each time a student answers an exercise, the score he gets as well as resulting change to his averages are shown on the page. Experiments show that this instantaneous result is very efficient in inciting the student to work hard.

In the current version of Wims, complete list of scores of a virtual class can only be consulted by the teacher. For each student and each work sheet, the system computes two numbers, the percentage of assigned work which is done and the average of individual scores (between 0 and 10).

In principle, the system allows students to log in to the virtual class any time and from anywhere, and work on the assignments. In our experience, we have seen people issuing objections to this possibility, due to the fact that some students may ask others to login, work and get points for them, especially when the points are used in student evaluation.

For this reason, teacher's pages include menus allowing him to change the score registration status of a given work sheet: open, close, or open for a selected number of sites. With this, we have been able to let students to score only during monitored classroom sessions.

The structure of the system allows much more sophisticated analysis of student performance and progress. Usable data includes time delay between presentation of exercise and answer, number of failures, number of consultations of hints, duration of connections, etc. However, how to efficiently exploit these data is still (serious) subject of discussion.

Ultimately, it is possible to generate intelligent personalized guides to students according to their performance.

It is also possible for the structure to host electronic discussions between teacher and students, or among students. This is not implemented for the time being, only because there is no demand (yet).

6. Security considerations

Security is a very important aspect in the design of Wims. As the system allows remote visitors to execute various programs on the server, care must be taken to avoid the following risks.

1. Resource exhaustion. A user may ask for a computation (for example factoring a very big integer) which takes hours. Not seeing the result coming, he then clicks on the 'reload' button of the browser for several times, and the server would end up with several copies of the background computing software running endless loops, completely exhausting its CPU resource.

   Our solution is to set up a limit to the CPU time allowed to all programs called by Wims. A limit of 20 to 30 seconds, which corresponds roughly to the limit of an ordinary user on the Internet, is sufficient for most computational needs. This limit is configurable at the installation level.

   Other resource limits are defined: memory size, file size, etc. But these are less restrictive, as they are not as crucial as CPU time.

   It also occurs frequently that an impatient user repeatedly clicks on a link if he doesn't see immediate response. This may overload the server if the link requires a lot of computation on it. For this reason, Wims prohibits further requests by the same user until the processing of the first request finishes.

   Finally, a two-level load limit can be defined. Requests to the server are refused when the server load exceeds the limit, with an error message sent to the user, suggesting him to try gain later.
2. Infiltration risk. Several softwares interfaced by Wims contain commands allowing the user to escape to the operating system. Any possibility for a remote user to gain access to such a command is a security hole. In particular, such a possibility should not be present when the user answers an exercise or requests a computation to tool.

   The ultimate solution is to let interfaces to these programs systematically deny access to such commands. For many softwares, it is very hard to achieve an absolute security in this way. So when writing modules, we are taking care not to let raw text supplied by users to be directly fed into background computing programs. Usually user-supplied text appears only as content of a particular function for the background program, and Wims

systematically checks all user input for the consistency of parentheses, partly in order to prevent users to escape to the exterior of the function content.

The server has been functioning under these security measures for more than one year, and up to now no successful attempt of security breaking in this respect has been registered.

3. Bugs or Trojan Horses in user-contributed modules. The system is designed to accept contributed modules. Authors of a contributed module have obviously a much greater freedom to access server resources, than users through the Internet. It seems impossible to eliminate all the possibilities for a contributed module to attack the server. Therefore measures in this direction are principally aimed at limiting eventual damages caused by bugs in modules. These include:
   (a) A strict directory checking. File names cannot start from the file system root, and cannot backtrack to parent directory (the string '..' is prohibited in file names).
   (b) Privilege hierarchy for modules. Non-privileged modules can only execute standard interfaced programs. Only privileged modules can execute any program contained in the directory of the module, or issue SQL queries.
   (c) Operating system privilege restriction. Wims is installed on a user space, with setuid to an ordinary user. Resource limit to this user can be defined from the operating system.
   (d) Resource limits mentioned above.
   (e) Contributed modules are tested and scrutinized before put into public access.
4. Internet robot accesses. Besides robots used by Internet search engines, many people use robot programs to gather entire contents of a site. The latter raises particular concern, as pages generated by Wims are intended to interactive activities, therefore their addresses contain tags designed to make each of them unique. Thus such a robot program, once launched, is unable to stop by itself. It wastes a lot of the network bandwidth, and the pages gathered in this way are useless.

   Our solution to this is to send pages to robots which are different to what is usually generated. Starting pages for each application are presented to robots, because they are necessary for search engine databases. But in these pages, links to answers are suppressed, and if a robot attempts to supply an answer, a special page is sent to indicate an error condition.

   For this purpose, the main program wims.cgi includes a small database of known user agents, both ordinary and robot. This solution is only partial, as some particularly ill-behaved robot programs give false agent information to the server.

Depending on the seriousness of the problem, robot control measures may be further strengthened, for example, by allowing only a limited number of robot accesses during a given period, for any given site.

5. Inter-session infiltration or interference. Nobody should be given access to files of a session not belonging to him. For this reason, temporary files in session directories are not directly accessible by http protocol. This creates a certain inconvenience, as graphics files generated by *instex, insplot*, *insdraw* are stored in session directories. These graphics files can therefore be accessed only via wims.cgi, and when the user saves such a file in his local machine, the default filename given by the browser is wims.cgi which does not have a good extension to make it exploitable by other softwares. For the time being, no solution to this problem has been found.

   In order to prevent session usurp by a third person, Wims has a mechanism to check IP number of the user, and to compare this IP number with that of the user who created the session. But this mechanism is only partially implemented for registered users, as many connections through Internet providers have dynamic IP numbers which change from request to request.

Mainly due to the above security considerations, Wims is currently only implemented on Linux environment which provides a high level of security protections. There is currently no project of porting it to Windows environment.

Anti-cheating is another aspect of the security consideration. This is important because Wims can be used for on-line examinations.

Globally at the server's level, the prevention of IP number change for registered users mentioned above is one of such measures. Various rules are also implemented in the score registration and accounting mechanism for virtual classes. Besides measures already mentioned in Section 5, we have taken into account the fact that as the exercises are randomly generated, a student may start by repeatedly change the exercise until he finds one which looks easier. When the score is counted, the number of new exercises requested and the number of answers are compared. If the difference exceeds a predetermined tolerance, extra requests for new exercises are counted as failures, which has the effect of lowering the score average. Measures are also taken so that no score is registered when the difficulty level of the exercise is different from that defined by the teacher.

The main anti-cheating measures should be implemented in different exercise modules. Scores should only be given once for each exercise generated, therefore the module must ensure that the user cannot repost an answer to an already scored exercise and get the score again, for example by using the 'back' or 'reload' buttons of the browser.

Another anti-cheating measure at the modules' level is governed by the principle of the system: efforts are taken to make exercises as random as possible (including random types).

In this way, when a student copies from his neighbor, he can only copy the solution to the neighbor's exercise. He must then extract the general method out of the neighbor's solution, and adapt it into his own exercise. The ability to do that is already a good achievement of the teaching goal.

References

[COQ] The Coq proof assistant, home address `http://pauillac.inria.fr/coq/`

[Decomp] Exercise on decomposition of a rational function,
`http://wims.unice.fr/~wims/wims.cgi?module=U1/analysis/decomp`

[Factoris] Online factorizer,
`http://wims.unice.fr/~wims/wims.cgi?module=tool/algebra/factor`

[Gnuplot] Plotting software, home address `http://www.cs.dartmouth.edu/gnuplot_info.html`

[Imps] Interactive Mathematical Proof System, home address
`ftp://math.harvard.edu/imps/imps_html/imps.html`

[Logicoq] Interactive proof exercise on first order logic,
`http://wims.unice.fr/~wims/wims.cgi?module=U3/logic/logicoq`

[MuPAD] Multi Processing Algebra Data Tool, home address `http://www.mupad.de`

[Octave] Software for numerical analysis, home address `http://bevo.che.wisc.edu/octave`

[PARI/GP] Software for number theoretic computations, home address `http://pari.home.ml.org`

[Polyray] Online surface plotter by ray tracing,
`http://wims.unice.fr/~wims/wims.cgi?module=tool/geometry/polyray`

[PostgreSQL] SQL database manager, home address `http://www.postgreSQL.org`

[Povray] Ray tracing software, `http://www.povray.org`

[Visual Gauss] Exercise on Gauss elimination,
`http://wims.unice.fr/~wims/wims.cgi?module=U1/algebra/visgauss`

[Wims home] Home address of Wims, `http://wims.unice.fr`

[Wimsdoc] Online documentation of Wims,
`http://wims.unice.fr/~wims/wims.cgi?module=help/wimsdoc`

Département de mathématiques, Université de Nice - Sophia Antipolis, 06108 Nice, France

E-mail address: `xiao@unice.fr`

URL: `http://pcmath126.unice.fr/xiao.html`