Configuring Web Wizards in XML

Marcos Aurélio Domingues and José Paulo Leal

DCC-FC & LIACC, University of Porto
R. Campo Alegre, 823 – 4150-180 Porto, Portugal
{marcos,zp}@ncc.up.pt

Abstract. A "wizard" is a common pattern used in graphical interfaces when an application needs to collect a large number of parameters. Wizards use progressive disclosure to present windows with small sets of parameters, and parameters selected in the first windows control those presented in subsequent windows. This concept of wizard can also be used to develop web interfaces to existing monolithic systems that lack a GUI of some sort and would benefit from being accessible on the Internet. In this paper we propose a framework for developing web wizards whose extension points are XSLT transformations based on a XML description of the systems parameters. With this approach the system and its web interface are loosely coupled and thus parameters can be changed or mapped differently into the system just by reconfiguring XML files.
This framework has been tested in the development of web wizard for a system that generates mathematics exercises using constrained grammars. The system, developed within project AGILMAT, has a large number of parameters that difficult its use by novice users. This paper also describes a number of features that were developed for the AGILMAT's web wizard.

1 Introduction

Most systems are designed with an users interface in mind. Nevertheless, there are cases where complex systems are developed with a poor users interface and thus need to be re-engineered in order to made them available to a larger user base. This is often the case with systems that steam out of research projects since they were developed around concepts that were not completely defined in the design phase. Understandably, these systems have a large set of parameters that must be exposed in the users interface.

In order to develop web interfaces to existing systems that lack a GUI of some sort, we propose a framework for developing web wizards interfaces whose extension points are XSLT transformations based on a XML description of the systems parameters. With this approach the system and its interface are loosely coupled and thus parameters can be changed or mapped differently into the system just by reconfiguring XML files.

Decoupling the users interface from the systems core is consensually recognised as an important design principle. In our approach with take this principle a step further. Since parameters are defined in configuration files, they can be easily added, changed or removed from the web interface. This feature is specially important in systems that are still being used in research. In these systems, changes in configuration parameters are frequent and user interfaces cannot be designed around rigid concepts.

Communication between the web server that manages the interface and the system is made trough input/output (I/O) streams. The parameters collected in the interface are converted into system commands using XSLT transformations and injected into its standard input stream. Data returned from the system's output stream is converted to XML and processed with XSLT transformation back into the web interface.

We claim that the approach we purpose is effective for developing a web interface to systems when certain condition are present:

– it was developed without an users interface in mind;
– it has a large number of configuration parameters;
– it has a small number of (ideally a single) actions;
– it can be fully controlled trough standard input/output.

All these conditions are met in AGILMAT. The goal of this research project is to produce a system to generate high-school and pre-university level exercises of mathematics. Exercises are generated using constrained grammars that are configurable by an extensive set of parameters. This system is implemented in Prolog and can be controlled by executing goals in the interpreters shell.

The ultimate goal of AGILMAT is to provide a tool for teachers available on the web. A first web interface was developed to AGILMAT and is currently being used for testing purposes in the development of the system [1]. This first web interface was tested with a focus group of high-school teachers. The results gathered from this experiment convinced us that the web interface would have to be profoundly changed, in order to reduce the actual number of parameters that a typical user would have to set.

This paper is organized as follows. We start by presenting the framework proposed for developing web wizards using XML (sect. 2). In Section 3 we present a case study where our framework has been tested in the development of web wizard for the AGILMAT. We also describe a number of features that were developed for the AGILMAT's web wizard. Then we conclude suggesting a few of the possible lines of future work (sect. 4).

2 A Framework for Developing Web Wizards

In this section, we present the framework proposed to make easier the development of web wizards. The key points of this framework consist in storing a XML description of the parameters of a system into a file and using XSLT transformations on this file to get the web wizards that will facilitate the configuration of the system parameters. The XSLT transformations act as extension points between the web wizard and the the system. The framework is illustrated in Fig. 1: browser, web wizard generator and system are represented as strong squares, points of transformation are represented as strong T labeled circles, physical files with XML documents are represented as strong file icons and XML documents in memory are represented as dotted file icons.

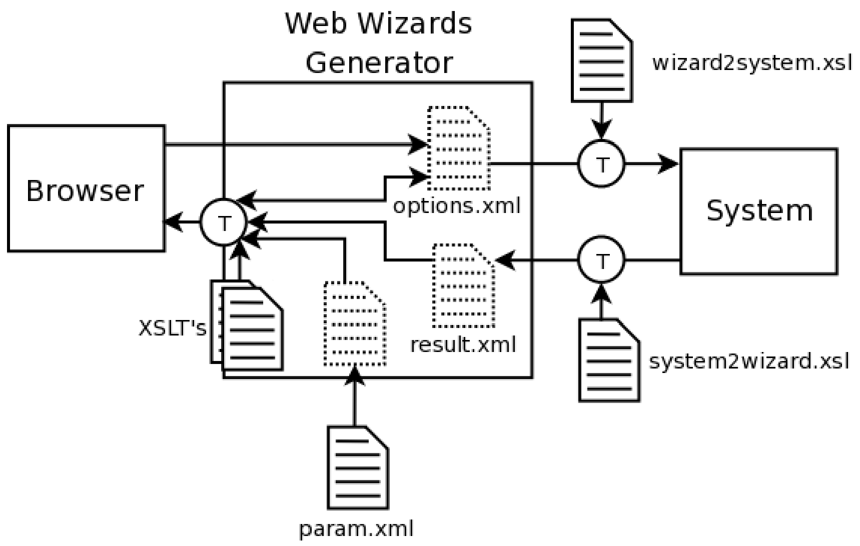


Fig. 1. Framework for Developing Web Wizards.

In Fig. 1, the file `param.xml` stores a XML description of the system parameters. This file is loaded to a DOM object and used with others document objects to make the XSLT transformations that generate the web wizards. The wizards generator has two other DOM objects in memory: `options.xml` and `result.xml`.

The DOM object `options.xml` manages and collects the parameters that are sent to the monolithic system. When the web wizards generator is initialized, the document `options.xml` is also initialized with the parameters from the file `param.xml`. This initializing process consists in applying XSLT stylesheets on the file `param.xml`. These generate the web wizards HTML and load the system parameters to the document `options.xml`. This document is never serialize into the file system. During interaction, the document `options.xml` is updated with the parameters received from the user browser.

XSLT transformations are also responsible for the appearance of the web wizards. In this way, to change the appearance of the web wizards, we only need to edit the appropriated XSLT stylesheets.

When the interaction requires executing an action in the system, the parameters in the document `options.xml` are convert to commands of the system. The conversion of formats is made using the document `wizard2system.xsl` to perform a XSLT transformation injected in the systems's standard input.

The DOM object `result.xml` stores the results from the system, in XML format. System output in XML format is integrated in the web wizards generator using also a stylesheet. This requires a previous conversion of the system output into XML format. The conversion can be performed either in the system or in the generator.

In this framework, when the need to add or change a parameter to the system arises, it is only required to edit the files `param.xml`, `wizard2system.xsl` and/or `system2wizard.xsl`. The programmer does not need re-write and re-compile the web wizards generator.

In the next section, we present a case study where our framework has been tested in the development of web wizard for a system that generates mathematics exercises using constrained grammars.

3 Case Study

In this section we describe the application of our framework to the project AGILMAT - Automatic Generation of Interactive Drills for Mathematics Learning [1, 2].

The project AGILMAT [1, 2] aims at developing a system of Constraint Logic Programming [5] to automatic generation and explanation of mathematics exercises that is flexible enough to be easily customizable to different curricula and users. Its major guiding principles are: the abstraction and formal representation of the problems that may be actually solved by algebraic algorithms covered by the curricula, the customization of these models by adding further constraints, and designing flexible solvers that emulate the steps students usually take to solve the generated exercises.

3.1 Architecture of the AGILMAT System

To make the AGILMAT system available for students and teachers, we use our framework to develop a web wizard. In Fig. 2 we present the contextualization of the AGILMAT system inside the framework proposed to develop web wizards: web wizard generator, cache and AGILMAT system are represented as strong squares and rectangles, internal modules of the AGILMAT system are represented as dotted rectangles, points of transformation are represented as strong T labeled circles, switcher is represented by strong rhombus, physical files with XML documents are represented as strong file icons and XML documents in memory are represented as dotted file icons. We also have a small screenshot of the web wizard developed for the AGILMAT system.

In the AGILMAT system (Fig. 2) there are two main modules that are written in Prolog and act as filters: the **expression generator** processes the user constraints and produces an expressions and types file, the **exercise generator and solver** processes this file and produces exercises and theirs solutions. This last module is the control of the system and makes use of several libraries that

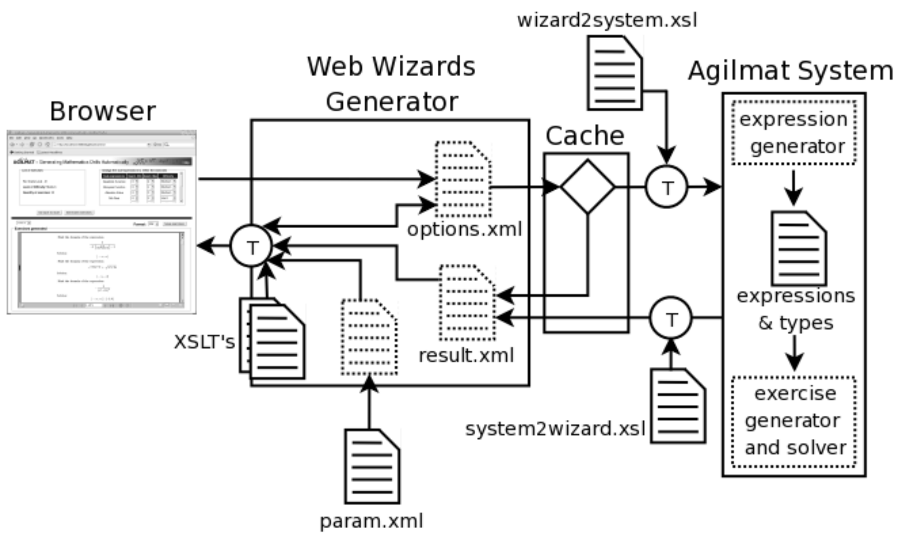


Fig. 2. Contextualization of the AGILMAT system in the framework for developing web wizards.

handle arithmetics, set operations and symbolic constraints (to solve inequations, disequations and equations).

The AGILMAT's web wizards uses the document `system2wizard.xsl` to convert the exercises and theirs solutions, represented by prolog predicates and clauses, to a XML representation. With a XML representation, we can further transform the exercises to the format XML - Question & Test Interoperability (XML - QTI) [3] with mathematical expressions represented in MathML.

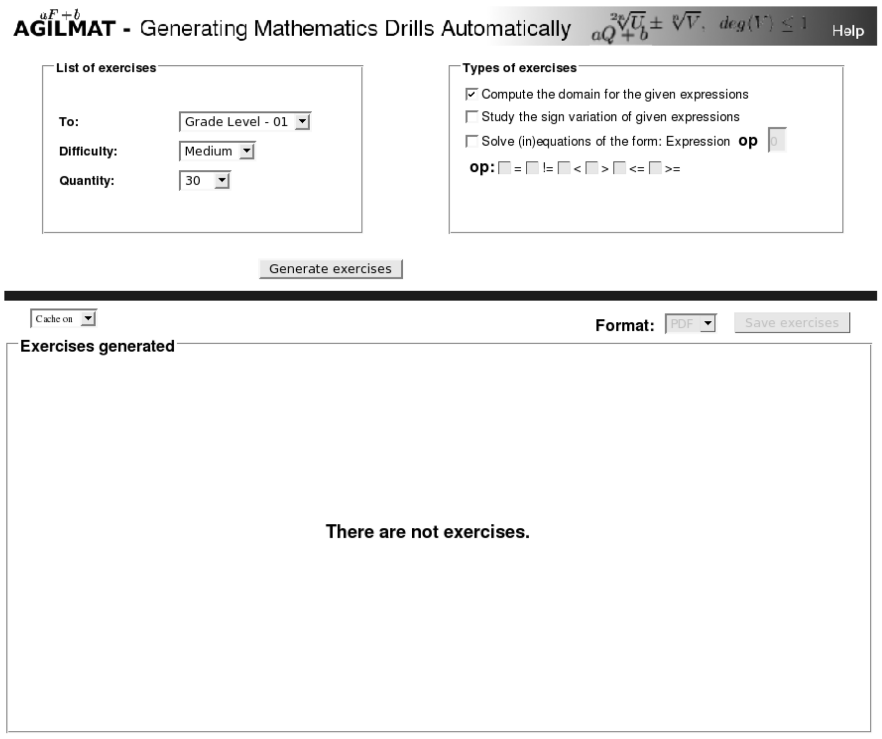
In the current version, exercises and theirs solutions are converted to a $\LaTeX$ representation that is converted to different formats, such as: HyperText Markup Language (HTML), Portable Document Format (PDF) and PostScript (PS). The PDF file is embedded in the web interface. We are not yet using the document `system2wizard.xsl` to convert the exercises and theirs solutions to a XML representation. We hope to use this document in the next version of AGILMAT.

The web wizard of AGILMAT collects and manages a set of options to control the generation of mathematics exercises. As explained in Section 2, the web wizards generator is responsible for managing the user-system interaction. It receives data from the file `param.xml` and HTML forms, makes several XSLT transformations, produces the user constraints and runs the AGILMAT system to generate the exercises. The web wizard generator has been written in Java.

The current release of AGILMAT takes a considerable time to generate exercises. To improve the response time of the system we developed a cache system. When the cache is activated, the web wizard looks up a set of exercises previously generated with the same configuration values. We expect that most teachers will initialize using the default values, or changing only the parameters presented in the first screen. The cache system will provide them with almost imediate feedback in these first atempts that will encourage them to explore AGILMAT and set parameters that will generate exercises more adapted to their goals.

3.2 Using AGILMAT Web Wizard

In Fig. 3 we present a screenshot of the first screen of the web wizard of AGILMAT. The most recent release of the system is available at `http://www.ncc.up.pt:8080/Agilmat`.



List of exercises

To: Grade Level - 01
Difficulty: Medium
Quantity: 30

Types of exercises

- [✓] Compute the domain for the given expressions
- [ ] Study the sign variation of given expressions
- [ ] Solve (in)equations of the form: Expression **op** 0

op: [ ] = [ ] != [ ] < [ ] > [ ] <= [ ] >=

Cache on

Format: PDF

Exercises generated

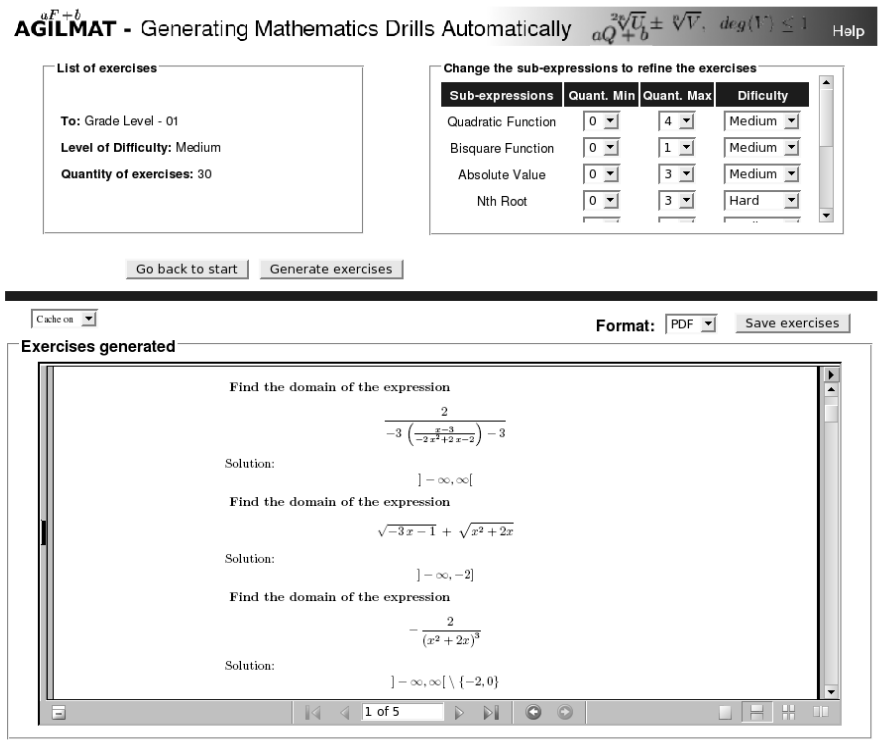
There are not exercises.

Fig. 3. Screenshot of the first screen of the AGILMAT wizard.

A novice user of AGILMAT starts generating exercises defining only a minimal set of options. In particular the user can define a **profile** of exercises that will be generated. This makes the system initialize the range of each parameter of AGILMAT system with suitable default values. These values may correspond not only to grade levels (Tenth grade, Eleventh grade, etc) but also to specific topics of the curricula. The advantage of this option is to allow novice users to generate exercises for their particular needs without having to set a large number of parameters.

The user can also define the quantity and types of exercises that will be generated. The available types are "Compute the domain", "Study the sign variation" and "Solve (in)equations". Exercise sheets may contain several different types of exercises. The user can generate exercises immediately after setting these parameters and expect them to be adequate to his or her needs.

Afterwards, the user can change the quantity and difficulty of each sub-expression that can appear in exercises (second screen of the AGILMAT wizard, ilustrated in Fig. 4). These parameters give extra control over the generated exercises. Each parameter has a selection with certain number of options. The available parameters, their options and default values are dependent from the initial selection of exercise profile.



List of exercises

To: Grade Level - 01
Level of Difficulty: Medium
Quantity of exercises: 30

Change the sub-expressions to refine the exercises

Sub-expressions	Quant. Min	Quant. Max	Dificulty
Quadratic Function	0	4	Medium
Bisquare Function	0	1	Medium
Absolute Value	0	3	Medium
Nth Root	0	3	Hard

Format: PDF

Exercises generated

Find the domain of the expression

$$\frac{2}{-3\left(\frac{x-3}{-2\,x^2+2\,x-2}\right)-3}$$

Solution:

$$]-\infty,\infty[$$

Find the domain of the expression

$$\sqrt{-3\,x-1}\;+\;\sqrt{x^2+2x}$$

Solution:

$$]-\infty,-2]$$

Find the domain of the expression

$$-\frac{2}{(x^2+2x)^3}$$

Solution:

$$]-\infty,\infty[\setminus\{-2,0\}$$

Fig. 4. Screenshot of the second screen of the AGILMAT wizard.

3.3 Configuring AGILMAT's Web Wizard

The first AGILMAT web interface was tested with a focus group of high-school teachers. The results gathered from this experiment convinced us that the web interface would have to be profoundly changed, in order to reduce the number of parameters that a typical user would have to set.

Taking this fact in consideration, we decided to group the system parameter into profiles in such way that the choice of a profile would set the parameters default values and hide irrelevant parameters. In Fig. 6 we present fragments of the file `param.xml` that focus on a profile. The node `configs` is the root node of the file `param.xml`. The elements `profile` contain a set of parameters that represents a grade level or a specific topic of the curricula, as for example, the type of the exercises that will be generated. We can have simple parameters or composition of parameters, that can have one or more values. The element `option` represents a simple value and the element `options` represents an interval of values. Parameters can be hidden from the user by changing the attribute `isVisible`.

```
<!-- ... others parameters ... -->

<parameter isVisible="true" id="rdivisao" name="divisao">
  <option label="Hard" value="8"/>
  <option label="Medium" value="5" default="true"/>
  <option label="Easy" value="3"/>
</parameter>

<!-- ... others parameters ... -->

<parameter isVisible="false" id="rdivstype" name="divstype">
  <option value="" exprs="$rdivisao + 2" default="true"/>
</parameter>

<!-- ... others parameters ... -->
```

Fig. 5. Interdependency between two parameters.

In a profile, parameters value can be computed from the values of others parameters. In Fig 5 we show how the expression `$rdivisao + 2` sets the value of parameter `divstype` where `$rdivisao` references the value of another parameter.

Managing a large number of profiles with common parameters in a single `param.xml` is too canbersome. To deal with this problem, we implemented profile inheritance to reuse profile definitions, and made use of the XML inclusion mechanism to split the configuration document in several files. In Fig. 6 the element `profile` presents the attribute `extends` whose value is the name of the extended profile. The extension can be done between profiles in the same or in different files using `xinclude` [4]. An example of the inclusion mechanism `xinclude` is presented in Fig. 6.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ... others elements ... -->

<configs>

  <profile name="p11" label="Grade Level - 11" extends="p01">

   <screen_wizard name="first">
      <parameter isVisible="true" id="qtyexe" name="quantity_exercise" >
        <options start="5" end="30" step="5" default="5"/>
      </parameter>

      <!-- ... others parameters ... -->

      <composite label="Types of exercises" isVisible="true">
        <parameter hasDependents="false" isVisible="true" id="edom" name="domain">
          <option label="Compute the domain" value="false" default="true"/>
        </parameter>
        <parameter hasDependents="true" isVisible="true" id="econ" name="constraint">
          <option label="Solve (in)equations: Expression" value="false" default="true"/>
        </parameter>
        <parameter depends="econ" isVisible="true" id="econccond" name="cond">
          <option value="0" default="true"/>
        </parameter>
        <parameter depends="econ" isVisible="true" id="econoeq" name="eq">
          <option label="=" value="false" default="true"/>
        </parameter>
      </composite>
   </screen_wizard>

   <screen_wizard name="second">
      <parameter isVisible="false" id="rabsquotbasic" name="absquotbasic">
        <option value="100" default="true"/>
      </parameter>

      <!-- ... others parameters ... -->

      <composite label="Quotient of Functions" isVisible="false">
        <parameter isVisible="false" id="fmidivbasic" limit="min" name="divbasic">
          <options start="0" end="2" step="1" default="1"/>
        </parameter>
        <parameter isVisible="false" id="fmadivbasic" limit="max" name="divbasic">
          <options start="0" end="2" step="1" default="1"/>
        </parameter>
        <parameter isVisible="false" id="rdivstype" name="divbasic">
          <options start="0" end="2" step="1" default="2"/>
        </parameter>
      </composite>

      <!-- ... others composites ... -->

   </screen_wizard>

</profile>

<!-- ... others profiles ... -->

<xi:include href="param01.xml" xpointer="p01" xmlns:xi="http://www.w3.org/2001/XInclude"/>

</configs>
```

Fig. 6. Fragments of the file `param.xml`.

4 Conclusions and Future Work

In this paper we propose a framework for developing web wizards whose extension points are XSLT transformations based on a XML description of the systems parameters. With this approach the system and its web interface are loosely coupled and thus parameters can be changed or mapped differently into the system just by reconfiguring XML files. We have successfully tested our framework in the development of web wizard for a system that generates mathematics exercises using constrained grammars. The system, developed within project AGILMAT, has a large number of parameters that difficult its use by novice users.

As future work, we plan to developt a document `system2wizard.xsl` to convert the exercises and theirs solutions from the AGILMAT system (represented by prolog predicates and clauses) to a XML representation, completing the proposed framework inside the project AGILMAT. With a XML representation, we will can make many others transformations, as for example, to transform the exercises to the format XML - Question & Test Interoperability (XML - QTI) [3]. We also plan to test our framework by developing a web wizards generator for an application with characteristics differents of the AGILMAT system. This will help us to better identify the features that are common to the framework and separate them from those specific to AGILMAT.

Acknowledgements

This work is partially supported by Fundação para a Ciência e Tecnologia (FCT), and Program POSI, co-financed by EC fund FEDER, under project AGILMAT (contract POSI/CHS/48565/2002).

References

1. A. P. Tomás, J. P. Leal. A CLP-Based Tool for Computer Aided Generation and Solving of Maths Exercises. In V. Dahl, P. Wadler (Eds), *Practical Aspects of Declarative Languages, 5th Int. Symposium PADL 2003*, Lecture Notes in Computer Science 2562, Springer-Verlag (2003) 223–240. ©Springer-Verlag.
2. A. P. Tomás, N. Moreira, N. Pereira. Designing a Solver for Arithmetic Constraints to Support Education in Mathematics. DCC-FC & LIACC, University of Porto, August 2005. *(working paper)* `www.ncc.up.pt/~apt/AGILMAT/PUBS/solver-Aug05.pdf`
3. IMS QTI Specifications. IMS Global Learning Consortium, Inc. `www.imsglobal.org/question/index.html`.
4. XML Inclusions (XInclude) Version 1.0. W3C Recommendation 20 December 2004. `www.w3.org/TR/xinclude/`.
5. Marriott, K., and Stuckey, P.: Programming with Constraints – An Introduction. The MIT Press (1998)