Rapid development of web interfaces to heterogeneous systems

José Paulo Leal and Marcos Aurélio Domingues

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

Abstract. The general problem addressed in this paper is the rapid development of web interfaces to software systems using only their command line interface. This kind of system is frequently developed in environments that greatly differ from those where web interface will be implemented. In this setting it is also important to maintain a loose coupling between the web interface and the system it controls since the latter must be able to continue its normal development independently of the former.
We propose a framework to develop web interfaces targeted to these systems whose main feature is the fact that it can be extended without requiring code programming. The hot spots of our framework are XML configuration files to define the interface data, how this data is mapped into the system's commands, and how commands output and the interaction state is mapped into web formatting languages. With this approach the web interface is kept separated from the system it controls, it is easy to define and modify, and is able to capture enough domain knowledge to be a real advantage for the novice or sporadic user.
In this paper we present the proposed framework architecture, loosely inspired in the MVC pattern, its implementation on Java servlet containers, and its application to the AGILMAT system, a high-school mathematical problem generator developed using constrained grammars.

1 Introduction

Software steaming from scientific research frequently lacks a graphical user interface (GUI), at least during the early stages of its development. In this first stages configuration files and command interpreters are used to test it and eventually a GUI will be needed to make it available to other persons.

This type of software frequently depends on specific hardware equipment or other software components that are not easy to install in most computers for a number of reasons. Thus, web interfaces are an obvious choice to implement a GUI for this type of software since they enable the deployment of the interface without the need of deploying the system's core functionalities. In this paper we will refer the software component to which we want to develop a web interface simply as **the system**.

Developing a GUI for a system still in research has its own challenges. On one hand, the GUI should be designed from the users standpoint, creating a conceptual model of the interface adjusted to them and hiding unnecessary implementation details. On the other hand, the system is still under tunning and their developers will want to control all the software parameters from its GUI. Moreover, as the system is still under development, the set of parameters that controls the system may have not stabilized yet.

Decoupling GUI from other software components is generally regarded as a sound design principle. In our scenario we actually need to maintain a very loose coupling between the system and its web interface. We assume that they both run on different machines, probably on different platforms, and that the developers of the web interface have a very limited capability of changing the system implementation. However, we assume that the system was some sort of textual command interpreter and that it can be accessed using I/O character streams.

In this article we present a framework for loosely coupling a web interface to a system, whose main feature is the use of XML [10] documents as extension points. In the following sections we start by presenting the proposed framework architecture, loosely inspired on the MVC pattern [2], where Model, View and Controller are defined by XML documents. We proceed with a general description of the implementation of this framework on a J2EE servlet container, and then with the application of this framework to a mathematical exercises generator named AGILMAT. Finally, we draw some conclusions and point to future work.

2 Architecture

The Model-View-Controller (MVC) architectural pattern [2] is recurrent in the design of applications with a graphical users interface. More than just decoupling the graphical interface from the application logic (Model), it clearly separates the visualization of the application state (View) and the binding of commands to the application logic (Controller). This pattern was originally proposed by Trygve Reenskaug for Smalltalk [6], later adopted by the object-oriented community [2] and more recently "rediscovered" for multi-tiered applications with web interfaces. This variant of MVC was named "model 2" by SUN [7] and is sometimes referred as MVC2.

Consider the diagram in Fig. 1 representing the relationships between the three types of participants in the MVC patterm when applied to web applications. We start by noting that in our scenario the system to which we want to develop the web interface is undoubtedly the model. Our framework to support the web interface must implement the equivalent to the controller and the view and will require some knowledge of the model in order to communicate with it. Having in mind that the system (model) and the web framework (controller and view) will be running on different processes, an important issue is the communication between the two.

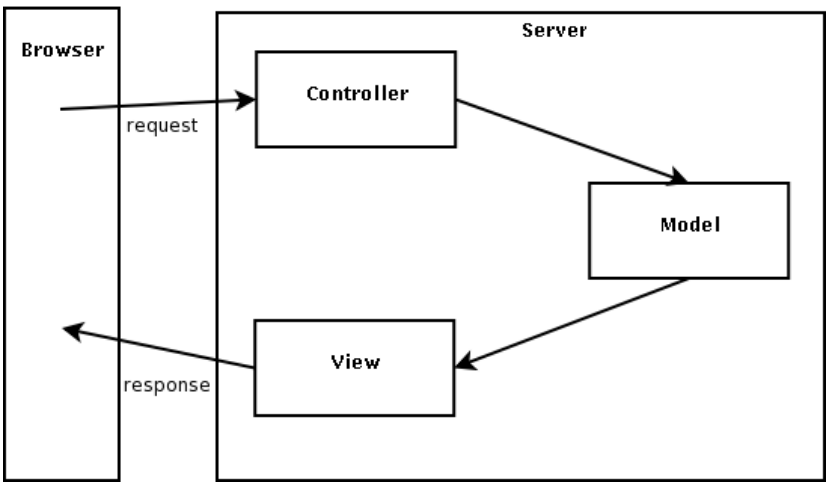

Fig. 1. Model-View-Controller for web applications

In "model 2" web applications running on Java application servers, different types of components are used for implementing each of the three participants of the MVC design: Java beans are used to connect to the application model, Java Server Pages (JSP) implement views, and a single servlet acts as a front controller. Frameworks such as Struts [4] and Spring [3] make use of this design to automate the creation of front controllers that are automatically generated from XML configuration files. These configuration files map client (HTTP) requests to actions and views. Actions are class instances responsible for initializing and activating model beans that will be later used by JSPs to produce views.

The advantage of using beans as model components is the fact that objects of these classes can be conveniently accessed through **properties**. Property values can be queried or modified using conventional methods whose name reflects the name of the property and the intended operation. For instance, a bean with a property named "date" of type `Date` will have the methods with the signatures `void setDate(Date date)` and `Date getDate()`. Although very simple, this assumption is very important to enable the binding of a bean component to a controller action, that will initialize beans and set properties, or to a JSP implementing a view that just queries beans values.

The communication between model beans and other components is based on the fact that they are all objects of the same execution, which is not the case in our scenario. Clearly, there are several ways for invoking methods on a remote process, such as RMI (Remote Method Invocation) between Java processes, or RPC (Remote Procedure Call) between any two processes using web services, just to name a few. Nevertheless, all these approaches require an extra level of complexity on the system's side that we want to avoid, and may have a significant cost in terms of efficiency. Thus, in our setting we assume that the

model is a separate process capable only of simple and limited communication through input and output streams, using a command line shell.

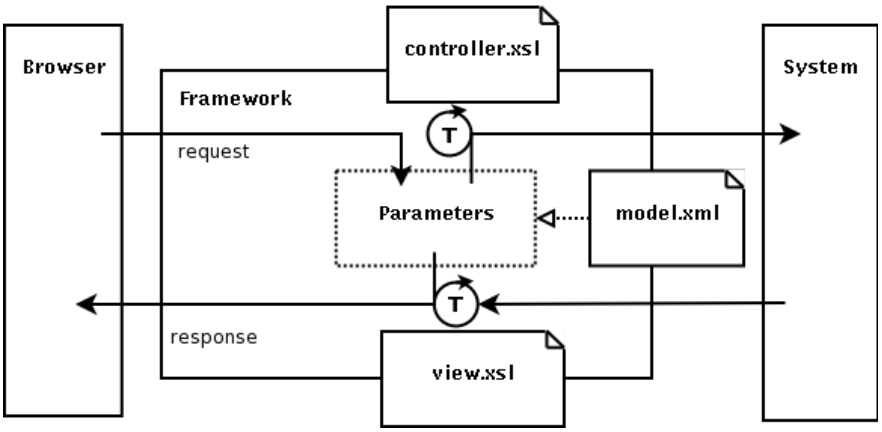

Fig. 2. Framework architecture

In the proposed framework the role of bean properties in steering the model execution and showing its state is attributed to **parameters**. The set of parameters that govern user interaction are defined in an XML document. From the perspective of the MVC pattern, this document can be seen as the definition of the model, not in the sense that it implements all the "business" logic, but in the sense that it defines communication points with the controller and the views. In this perspective parameters have a similar role to that of properties in Java beans. The parameter set is initialized from the XML configuration when a new user session is started and is updated each time a request is received from the browser. This parameter set for each user session holds the state of the interaction. It is represented as a dotted box in the diagram of Fig. 2 connected to the configuration file `model.xml`.

At each request, parameters are converted into commands that are injected in the input stream to be interpreted and executed by the system. The conversion between parameters and system commands must be another extension point in our framework since it depends both on the set of parameters and the commands actually supported by the system. Capitalizing on the use of XML, this hot spot is an XSLT [13] file that converts the document object holding the parameters. From a MVC perspective, the role of this XSLT file is analog to the role of the controller since it relates user interaction to the model. In Fig. 2 the XSLT transformations are represented by arrowed circles with a capital T inside next to the documents with the stylesheets, in this case `controller.xsl`.

In general, system's commands will produce an output. This output combined with the current state of the interaction will update the web interface. Converting

these two types of data into a web formatting language such as HTML must also be an extension point in our framework since each interface will have its own requirements. As the reader would expect by now, this hot spot is also an XSLT file that, from a MVC perspective, has the role of a view since it presents the state of the interaction and the output of commands.

The architecture summarized in Fig. 2 is targeted for Java servlet containers. Its design is clearly inspired on MVC design pattern but is very distant from the "model 2": it does not use beans to connect to the model, it makes no use of JSP to generate views in a web formatting language, it just uses a single servlet as front controller to process requests. In fact, it can be argued that it does not follow the MVC design pattern since none of its participants (model, view and controller) are in fact objects but rather XML documents that configure the framework's extension points.

3 Framework

In this section we present some technical details on the implementation of the framework as a J2EE web application, using a servlet container[1]. The general structure of the framework is described in Fig. 3 by a UML class diagram [1]. This diagram highlights the main classes in the framework and their use of the XML extension points, the files `model.xml`, `view.xsl` and `controller.xsl`.

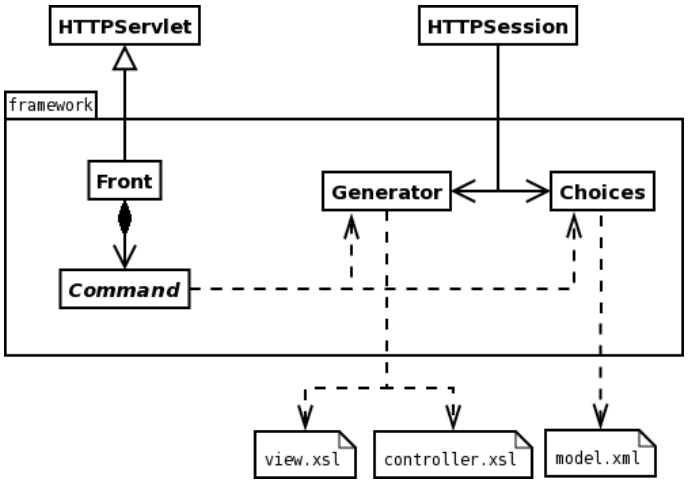

Fig. 3. Conceptual class diagram of the framework

The class `Front` is an HTTP servlet acting as a front controller. It is a single entrance point of all requests from the web interface. The front controller uses

[1] In the development we used the Apache Tomcat Servlet Container, version 5.5

a collection of extensions of the abstract class `Command` to implement the actual processing of each type of requests. These commands depend on the classes configured by the XML and XSL files that are the extension points of the framework.

When an user accesses the web interface via a browser a new session is started on the server side with two new objects assigned to it: an instance to the class `Choices` and an instance of class `Generator`. These two objects are stored in the HTTP session and thus are indirectly accessible to future commands processed within the same session.

The class `Choices` collects and manages choices made by the user during the interaction with the web interface. These choices are values of parameters defined in `model.xml`. The document defining these parameters must be valid according to a specific language defined by a DTD. The parameter definition language includes such features as: composition of parameters, definition of default values as expression involving other parameters, dependencies between parameters, among others. The complete list of features of this language is outside the scope of this paper but a fragment of this document is presented in Fig. 4 as an example.

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

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

      <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>

</profile>

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

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

</configs>
```

Fig. 4. A fragment of the document `model.xml`

The class `Generator` is responsible for applying XSLT transformations, either to generate system commands or to generate a new web interface. When a new instance is started it automatically launches a system's process. This process is associated with the user's session so that it can be reused for subsequent requests within the same session. With this approach, different simultaneous users of the web interface will interact with different processes of the same system. On session termination (when the user logs out or a timeout is reached) this connection with system's process is automatically closed.

There are different moments in the life cycle of a system process: it is initialized, processes commands and is terminated. Each of these moments corresponds to a mode in the XSLT stylesheets that change the way it is processed and commands generated.

To improve the response time of the overall system with the web interface, we developed an optional cache system in the framework. When the cache is activated, the `Generator` looks up for a system output previously generated with the same choice of parameters. Our experience showed that certain sets of choices, specially those selected in an early stage of the interaction, tend to be repeated since they are just default values of the initial screens. In these situations the cache system provides almost immediate feedback in these first attempts which encourages novice users to continue exploring the system's more complex features.

The cache system assumes that the state of the interaction is fully described by the set of parameters used for generating the system's commands. The current implementation lacks an invalidation mechanism for dealing with possible side effects in system's commands. The only control available is in the GUI: the user can switch the cache system off to ensure that commands are actually executed and a fresh output is generated from the system. This issue of cache invalidation will be addressed in a future implementation of the framework.

4 Case Study

In this section we report on the application of our framework to the project AGILMAT - Automatic Generation of Interactive Drills for Mathematics Learning [8, 9]. AGILMAT can be described as a tool for automatic generation and explanation of mathematics exercises, customizable to different curricula and to students with various levels of knowledge. It is a Constraint Logic Programming [5] based system and 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.

To make the AGILMAT system available for students and teachers, we developed a **wizard** using the proposed framework. 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 interface has a rather complex structure, composed of multiple interdependent screens. Fig 5 presents a screenshot of the second screen of the ALGILMAT's wizard: on the top left it shows a summary of the parameters selected so far; on the top right it shows selectors for parameters that are compatible with the current state; on the bottom it shows a set of exercises that were generated with the current selections.

AGILMAT proved to be quite a challenge for the framework since it required a lot expressive power for describing and structuring its parameters in order to support a the complex structure of the web wizard. As could be expected, some of the features of the parameter description languages were in fact "forced" by AGILMAT but we believe they are will be useful for future applications of this framework.

The AGILMAT's web interface uses the document `controller.xsl` to map parameters into Prolog clauses that feed the AGILMAT's system. Although this conversion could be handled entirely on the framework's side, using XSLT transformations, we opted to keep this conversion fairly simple and develop a Prolog module on AGILMAT's system to process the parameters collected by the web interface.

Reciprocally, `view.xsl` is used to produce an HTML interface to display the current interaction state (namely the selected parameter values) and the exercises generated by the AGILMAT's system. For that purpose we had to convert the exercises and their solutions to XML formatting languages, which required the addition of a new Prolog module to serialize terms into an XML format, to the AGILMAT's system. In this case we could not have avoided doing this conversion on the Prolog process side since XLST cannot handle Prolog terms as input.

With a XML representation, we can use `view.xsl` to transform the exercises to the format XML - Question & Test Interoperability (XML - QTI) [11] with mathematical expressions represented in MathML.

In the current version, exercises and their 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 `view.xsl` to convert exercises and their solutions to a XML representation. We hope to use this document in the next version of AGILMAT. The current version of AGILMAT is available in `http://www.ncc.up.pt:8080/Agilmat`.

5 Conclusions and future work

In this paper we propose a framework for developing web interfaces whose extension points are XSLT transformations based on an 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

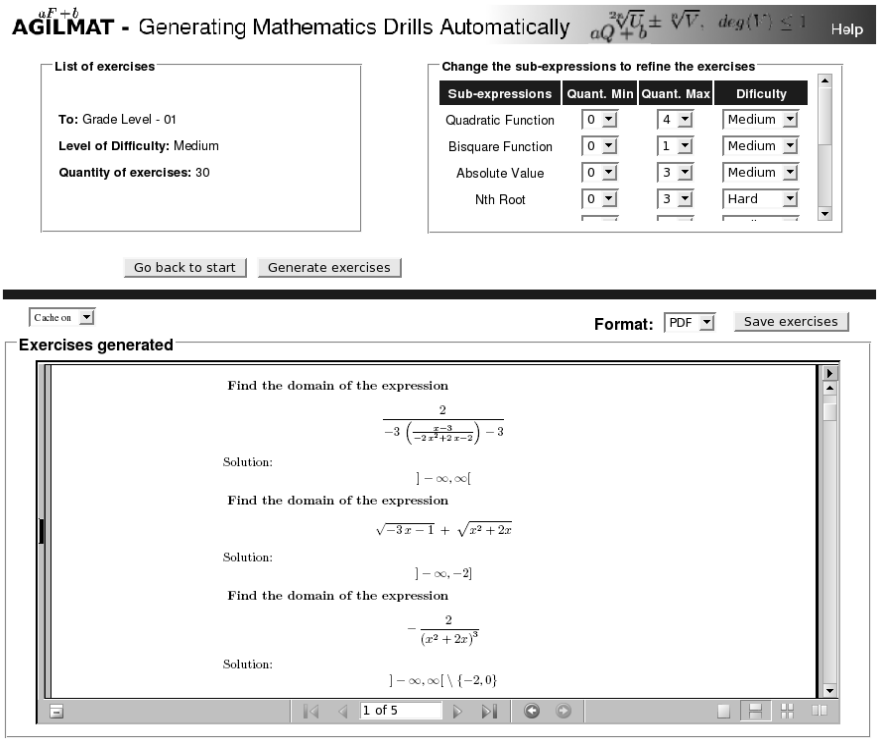

Fig. 5. Screenshot of the second screen of the AGILMAT wizard.

in the development of a 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 test our framework by developing a web wizards generator for other large applications with characteristics different 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.

We plan also to explore the applicability of this framework to graphical applications more complex than web wizards, with system's commands linked to events of a lower granularity, such as mouse clicks or icon dragging. For this kind of application we plan to use Ajax for sending asynchronous XML messages to the framework server, that will process them and feed its data to the system and produce an XML reply to the web client.

Acknowledgements

Work partially funded by *Fundao para a Cincia e Tecnologia* (FCT) and *Programa POSI*, under project AGILMAT (contract POSI/CHS/48565/2002) and by LIACC through *Programa de Financiamento Plurianual*, FCT and *Programa POCTI*, co-financed by EC fund FEDER.

References

1. Grady Booch, James Rumbaugh, Ivar Jacobsn. *UML Reference Manual* Addison Wesley, 1999
2. Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented software*, Addison Wesley Professional, 1994.
3. Rod Johnson, Juergen Hoeller, Alef Arendsen, Thomas Risberg, Colin Sampaleanu. *Professional Java Development with the Spring Framework* Wiley Publishing.
4. Budi Kurniawan. *Struts Design and Programming* Brainysoftware, 2006
5. K. Marriott and P. Stuckey. *Programming with Constraints – An Introduction*. The MIT Press, 1998.
6. Trygve Reenskaug. *MODELS - VIEWS - CONTROLLERS*, Technical note, Xerox PARC, December 1979. Scanned version: `http://heim.ifi.uio.no/ trygver/mvc/index.html`
7. Inderjeet Singh, Beth Stearns, Mark Johnson et al. *Designing Enterprise Applications with the J2EE Platform* Addison-Wesley, 2002.
8. 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.
9. 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`
10. Extensible Markup Language (XML) `http://www.w3.org/XML/`
11. IMS QTI Specifications. IMS Global Learning Consortium, Inc. `www.imsglobal.org/question/index.html`.
12. XML Inclusions (XInclude) Version 1.0. W3C Recommendation 20 December 2004. `www.w3.org/TR/xinclude/`.
13. XSL Transformations (XSLT) W3C Recommendation 16 November 1999 `http://www.w3.org/TR/xslt`