Transforming XML Schemas into Java Swing GUIs

Patrick Lay, Stefan Lüttringhaus-Kappel
Institut für Informatik III, Universität Bonn
{lay,stefan}@iai.uni-bonn.de

Abstract: When designing an XML-based Web Content Management System (WCMS), one usually has to define and maintain two separate entities just for the editorial part: the database schemes (in form of XML Schema instances) on the one hand and the graphical user interfaces (GUIs) for data maintenance on the other hand.

In this paper we present a method for generating the GUIs automatically from the XML Schema instances, thus the GUI can be specified in a declarative, implicit way. The target platform is Java Swing. Our transformation is based on compile-time generated templates and the XML persistence feature of Java 2.

1 Introduction

The developer of a database application like an XML-based WCMS has to carry out two separate tasks, i. e. define the data schemes for the data first and then create a suitable GUI for inserting, editing and deleting the data. This comprises a large overhead, as Myers et al showed in [MR92] where they found out that, statistically, an average of 48% of the written code and nearly half of the design, implementation and maintenance time during development is associated with GUI creation and maintenance.

Thinking of a book database, for example, one first has to define a XML Schema which describes the fields (like author or title) and then implement a GUI which enables users to work with that data. Furthermore, it should be possible to access a WCMS with various clients like standalone applications, web browsers or PDAs. Therefore, in the straightforward approach, one has to implement many different GUIs.

In this paper, we show how to automatically generate a GUI from schema data, thus saving the effort of creating the GUI manually. Another benefit is that if the schema changes (e. g. when new fields are added to the schema), the GUI can be re-generated very easily.

XML-based languages are gaining more and more influence and XML-based GUI description languages like XHTML, XForms or WML are also understood by most clients. While generating (X)HTML and WML GUIs out of the corresponding XML Schema can be accomplished in a relatively easy way using XSLT stylesheets, we show that more sophisticated Java Swing GUIs can also be generated and that the XML serialization features of Java 2 are useful towards that aim.

Simple XHTML GUIs offer easy access since no extra software is needed, while a Java Swing GUI offers more flexible and sophisticated input capabilities that browser-based forms can not provide. Furthermore, standalone GUIs offer immediate response (e. g. in case of erroneous data entry) while browser-based GUIs often require the interaction with the server, for example to display error messages. example. Obviously, both approaches have their advantages and disadvantages.

We focus on GUIs for data maintenance, i. e. inserting, editing and deleting data, since the generation of arbitrary complex GUIs requires the specification of various declarative models (e. g. domain, presentation and task model, see [SE96]), thus it cannot be covered in whole by automated processes based on just schema data.

The rest of this paper is organized as follows: Section 2 presents the basic idea behind our approach, i. e. the generation of a Swing GUI from XML Schema data. In section 3 we extend this approach by introducing a template mechanism. The paper closes with an overview of related work and final conclusions.

2 From XML Schema to a GUI

Since XML Schema itself is written in XML, it is easy to use XSLT stylesheets to transform schema documents into other XML languages. By using different stylesheets, we can generate XML-based GUIs for XHTML and WML in a straightforward way[1].

But on some occasions the functionality of the above mentioned mechanisms (like XForms) may not be sufficient for the input task at hand. These forms, for example, use a static layout of input elements[2]. Many of these restrictions can be overcome by using a Java Swing GUI, which can provide more functionality and more flexible input capabilities. In the following we present a way to automatically generate Java Swing GUIs from XML schemas using XSLT stylesheets.

Sun's Java JDK 1.4 introduced a new persistence scheme which allows the serialization of arbitrary JavaBeans into an XML file. Such a file can, in turn, be deserialized into Java objects in a Java application running in a different address space [Mi02].

The basic idea is that we generate the serialized XML representation directly from the XML Schema using an XSLT stylesheet, i. e. we only use the deserialization functionality of the JDK. This way, the serialized beans are sent to the client which, in turn, deserializes them to display the GUI. The advantage is that the actual Java classes have to be transferred to the client only once while the much smaller and most up to date GUI description can be transferred, for example, once for every session.

When designing a Java Swing GUI, it is reasonable to split up the GUI into separate communicating components. There can be, for example, several swing elements for one XML element, accompanied by one or more instances of controlling classes. In particular, we build three trees of different components:

- the content, i. e. the actual XML data, is managed in an XML DOM tree,
- the Swing GUI elements are stored in a swing tree, and

[1] with HTML forms, for example, or the more sophisticated XForms

[2] A commonly used workaround is to use JavaScript, but this solution is inadequate because it involves unexpected behavioural inconsistencies between different browsers.

- the controller objects, also organized in a tree, set up the various structures and keep them synchronized.

So this is nothing but a straightforward application of the successful Model-View-Control (MVC) design pattern [KP88]. Here the model corresponds to the DOM tree, the view to the swing tree and the controller to the control structures, respectively.

Consider the following Schema, which is shown here in EBNF-like notation (for the sake of simplicity and readability, since XML Schemas tend to get lengthy):

$$\begin{array}{lcl} \textit{Books} & \rightarrow & Book^* \\ \textit{Book} & \rightarrow & Title\ Authors\ Year\ Type \\ \textit{Authors} & \rightarrow & Author^* \\ \textit{Type} & \rightarrow & Online \mid Printed \end{array}$$

Year is of type `xsd:integer` while *Title*, *Author*, *Online* and *Printed* are of type `xsd:stri`

When generating the GUI, the system can use specialized input fields depending on the data type. In our example, the input field for “Year” permits only numerical input. Since the view is independent of the model, many different GUI elements could be generated for every XML element, like progress bars, spinners, or sliders. The only requirements for components are that they are JavaBeans, and that they implement the relevant interfaces.

In our example, every complex type has one the following three basic structures:

1. **Sequence,** e. g. $P \rightarrow abc$, i. e. consecutive elements, denoted by the `xsd:sequence` element
2. **Repetition,** e. g. $B \rightarrow b^*$, i. e. elements that occur n times (with $0 \leq min \leq n \leq max \leq \infty$), specified by the `minOccurs` resp. `maxOccurs` attributes of the schema.
3. **Alternative,** e. g. $D \rightarrow (e|f)$, denoted by `xsd:choice` elements

For simplicity, we restrict the following discussion to these basic cases, which cover many real-world examples. The extension to arbitrary XML Schema particles, from which any complex type can be assembled, is straightforward.

Sequences can be handled by simply concatenating the input fields while the other two constructs require the insertion of additional GUI elements like “add” and “delete” buttons to add and remove elements in a repetition, or selection buttons for alternatives.

All simple XML Schema types like `xsd:integer` or `xsd:string` are mapped directly to (possibly specialized) input fields like `JTextField`.

All XML Schema types, both complex and simple, are transformed into Swing components by a regular XSLT stylesheet. In order to modify the generation process, e. g. replace the default Swing elements (a `JSpinner` instead of a numeric text field) or add additional elements (a `JLabel` with additional descriptions), we could modify the stylesheet directly. This is, however, not desirable since we wish to keep the stylesheet generic to handle all cases. The solution is to provide an annotation system, i. e. the generation process can be altered by adding annotations to the schema. For instance, consider the schema entry for the Year element:

```
<xsd:element name="Year" type="xsd:integer"/>
```

Suppose we wanted a class named `Widget` instead of the default `JTextField` for this element, all we would have to do is to add an annotation to the schema (`gen:input`):

```
<xsd:element name="Year" type="xsd:integer" gen:input="Widget
```

3 Templates

As the structure of the GUI can change at runtime, some components cannot be generated statically at generation time. In our book example, it is unknown a priori how many author fields will be needed. It is, however, not necessary to compute all components and their interconnections at runtime. Instead we use templates computed at generation time, which are instantiated at runtime, possibly many times. A template represents a single element in the XML document and consists of:

model: a reference to a DOM node of the document being edited,

swing (sub)tree: represents the view, e. g. a `JTextField`, or a more complex assembly of several Swing components,

controller: manages the business logic of the element, i. e. it knows about its content model and has references to the relevant portions of both the DOM model and the Swing tree. The controller also handles events like the user pressing add/delete buttons.

At generation time, a template is generated for every element in the corresponding XML Schema file and stored in a *template library*.

During runtime, the appropiate templates are instantiated from the template library. For elements of type 2 (repetitions), `minOccurs` entries are generated, while the other types are generated only once (since both `minOccurs` and `maxOccurs` default to 1)

Templates are also instantiated and inserted, for example, if the user presses the “Authors: add” button. In this case a whole new book record is instantiated and appended. When pressing a “delete” button, the corresponding components are deleted.

By default, templates are chosen by name and type of the XML Schema element, but a different template can be specified via an annotation. In a similar fashion, help texts can be encapsulated in templates and displayed at appropiate places in the GUI.

4 Related Work

Most GUI generation processes (see the Seeheim-Model in [Gr84] or other model-based approaches in [SE96], [SSC^{+}95] or [Ba93]) have in common that they expect a GUI model *explicitly*, i. e. that a designer does this manually. Here lies the major difference to our approach, where the GUI is only described implicitly through the XML Schema, so one manual step in the implementation process becomes superfluous. The “fully” model-based approaches mentioned have been proven to be too inflexible[Pu96]. An overview over declarative models can be found in [dS00].

In [LC01], Luyten and Koninx presented an XML-based approach for generating user interfaces, which also lacks the fully automatic generation from schema but relies on writing the GUI specification in an XML language explicitly.

An alternative to the XML serialization mechanism that we use in our approach is SwingML[C
Here, the Swing GUIs are described in a separate XML-based language. Like in our approach, such description files are usually generated by the server and then sent to the client which renders it. An advantage of XML serialization is the generality, i. e. every Swing element can be utilized while in SwingML any Swing element that is not covered in the specification resp. the renderer cannot be displayed, for example user-defined GUI elements. On the other hand, SwingML documents are easier to read for humans. This is not important for our use case, though, since the GUI descriptions are usually processed automatically.

Several diploma theses written in our workgroup investigated related fields, e. g. the generation of GUIs elements for the web browser Mozilla ([Ko03])

5 Conclusions and future work

We presented an approach for automatic generation of GUIs for data maintenance in WCMSs that makes the explicit specification and implementation obsolete. The main idea behind this approach is that the GUI is generated from the XML Schema data, with the optional addition of annotations. In a first step, this is pretty straightforward for XML-based GUI languages like XHTML or WML, while the generation of more sophisticated GUIs like Java Swing requires some more work. We achieved this goal by utilizing the XML persistence for JavaBeans and implemented a template-based system to handle repetitions and choices. These templates put the MVC design pattern into practice by separating the model, view and controller in suitable Java classes.

Our system is capable of generating sophisticated editing masks for XML data just from schema data automatically, thus saving the extra effort for explicitly designing them. Our approach is very flexible because arbitrary XML Schemas can be transformed into a Swing GUI for data maintenance, i. e. there is no restriction on the input. On the client side, Swing GUIs provide a sufficient amount of input elements which could be easily extended if needed.

There are some aspects that we will concentrate on in the near future: to begin with, we will extend our collection of useful GUI component for various data types in different contexts with some emphasis on ergonomic aspects (see, among others, [Ni93]). Secondly, we will further refine the annotation system.

Finally, one of the topics we are currently investigating is the generation of XML Schema from sophisticated object-oriented models, thus eliminating the need to write relatively low-level XML Schema definitions manually.

References

[Ba93] Balzert, H.: Der JANUS-Dialogexperte: vom Fachkonzept zur Dialogstruktur (in German). In: Doberkat, E.-E. (Hrsg.), *Proceedings der GI-Fachtagung Softwaretechnik 93*. S. 62–72. Dortmund, FRG. November 1993.

[Cu04] Cuellar, E. SwingML. 2004. `http://swingml.sourceforge.net/`.

[dS00] da Silva, P. P.: User Interface Declarative Models and Development Environments: A Survey. In: Palanque, P. und Paternò, F. (Hrsg.), *Proceedings of DSV-IS2000*. volume 1946 of *LNCS*. S. 207–226. Limerick, Ireland. June 2000. Springer-Verlag.

[Gr84] Green, M.: Report on dialogue specification tools. *j-CGF*. 3(4):305–313. December 1984.

[Ko03] Kohlhaas, C.: Pflegeoberflächen für Internet-Informationssysteme (in German). Diploma thesis. Department of computer science III, Rheinische Friedrich-Wilhelms University Bonn. 2003.

[KP88] Krasner, G. und Pope, S.: A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming*. 1(3):26–49. 1988.

[LC01] Luyten, K. und Coninx, K.: An XML-based runtime user interface description language for mobile computing devices. *Lecture Notes in Computer Science: Interactive Systems: Design, Specification, and Verification: 8th International Workshop, DSV-IS 2001. Glasgow, Scotland, UK*. 2220:1–15. 2001.

[Mi02] Milne, P. Long term persistence of javabeans components: XML Schema. Sun Microsystems Inc. 2002.
`http://java.sun.com/products/jfc/tsc/articles/persistence3/`.

[MR92] Myers, B. A. und Rosson, M. B.: Survey on user interface programming. In: *Proceedings of SIGCHI'92: Human Factors in Computing Systems*. May 1992.

[Ni93] Nielsen, J.: *Usability Engineering*. Academic Press. 1993.

[Pu96] Puerta, A.: Issues in automatic generation of user interfaces in model-based systems. In: Vanderdonckt, J. (Hrsg.), *Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces (CADUI'96) Namur, 5-7 June 1996*. Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces (CADUI'96) Namur, 5-7 June 1996. 1996.

[SE96] Schlungbaum, E. und Elwert, T.: Automatic user interface generation from declarative models. *Workshop of Computer-Aided Design of User Interfaces*. 1996.

[SSC$^+$95] Szekely, P. A., Sukaviriya, P. N., Castells, P., Muthukumarasamy, J., und Salcher, E.: Declarative interface models for user interface construction tools: the MASTERMIND approach. In: *EHCI*. S. 120–150. 1995.