Dynamic Job Ticket Mechanism

C.J. Sonnenberg and T.K.J. Willems
Océ Technologies B.V., Research & Development
The Netherlands

Abstract

In a digital document production workflow customers have to complete a digital job ticket, usually without proper guidance. Errors can be introduced easily which leads to increased turn around time, damaged customer relation, wasted material or time.

To prevent such errors we propose a dynamic job ticket approach. This approach includes a description of all ticket constraints in a separate XML file. Based on the specified constraints, the user interface adapts to each user selection. In fact the user interface guides the user through the parameter space, preventing contradictions.

Introduction

The Internet enables customers to deliver their jobs digitally to their document service providers. Typically a customer is presented with a Web user interface in which numerous features for print, scan or archive services are offered. Each feature may include multiple selectable options, such as media type and duplex, but the selection of one feature may preclude options of another feature. For example 'transparent' precludes 'duplex'.

To handle this kind of ambiguities, the constraints of a ticket are described in an XML file. XML is a simple, open, non-proprietary, widely accepted data exchange format. Important standards like JDF (Job Definition Format)[3] are based on this format. Available techniques (e.g. schema's, DTD) validate XML documents, but omit the validation of semantic dependencies between XML elements. Promising initiatives like XForms[5] and XincaML[4] might be helpful to tackle this problem.

In our approach we model in XML the capabilities of the document service provider more closely by introducing constraints. A combination of a behavioral design pattern and XML technology lets the user interface adapt to each user selection. More specific, the user interface guides the user through the parameter space, preventing contradictions.

Design Pattern

The architecture is based on the Model-View-Controller (MVC) design pattern.[2] The pattern consist of three modules (see figure 1).

Model: The model is a hierarchical decomposition of all services, features and options offered by the service provider. In addition the dependencies and constraints between features and options on different branches of the option tree are modeled. The modeled constraints include context help; an explanation of the constraint to the user in natural language.

Controller: The controller manipulates the model according to the selections of the user. The controller preserves the integrity of the model by removing conflicting features and options.

View: This component creates the presentation markup of the model.

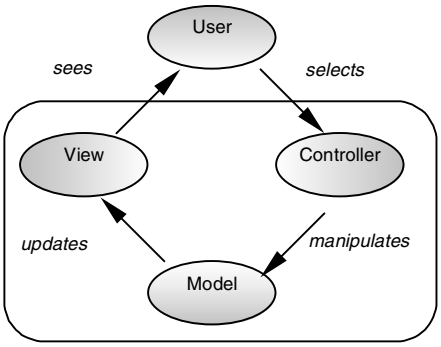

Figure 1. Components of the architecture

XML Technology

This section links the components of the architecture with the technology (see figure 2).

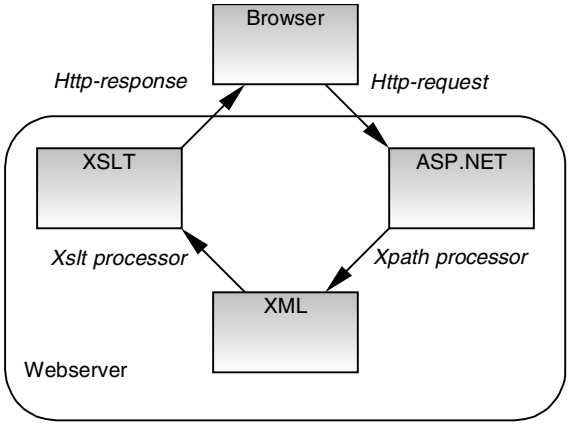

Figure 2. XML technology used in the components

Model: The specification of features and their options are located on different branches of an XML tree. Dependencies and constraints between features and options are added to this XML document. The next section elaborates on this issue.

Controller: The user interacts with the system via a Web page displayed in the browser on the user's computer. Selecting an option of a feature triggers a http-request to the Web server. The Web page reduces the model with all options from features that are conflicting with the user selection. The dynamic server page uses XPath[5] expressions to query and DOM[5] to manipulate the XML document.

View: An XSLT style sheet transforms the model into HTML format for presentation in the browser.

Modeling Constraints in XML

The XML document contains a hierarchical structure of the features and a list of constraints.

```
<option id="mediaID" name="Media">
  <option id="mediatypeID" name="Type">
    <option id="transparentID" name="Transparent" />
    <option id="paperID" name="Paper" />
  </option>
</option>
…
<option id="finishingID" name="Finishing">
  <option id="duplexID" name="Duplex">
    <option id="yesID" name="Yes" />
    <option id="noID" name="No" />
  </option>
</option>
```

Listing 1. XML fragment of features and options

The hierarchical structure represents the superset of possible features. The example in listing 1 shows an XML representation of the features 'duplex' and 'transparent'. All tags in the structure are of the type option. The id attribute uniquely identifies each option. The name attribute is used to represent the option in the user interface.

```
<error id="transparent_duplexID">
  <desc>Select either duplex or transparent.</desc>
  <exist>
    <choice option="transparentID" />
    <choice option ="yesID" />
  </exist>
</error>
```

Listing 2. XML fragement of a constraint

Listing 2 contains an XML fragment, which is an example of a constraint. Two types of constraints are identified: error and warning. Each constraint consists of a user friendly description and a predicate. The user-friendly description guides the user. The predicate defines when the constraint will be applied. In this example the predicate evaluates to false when both options, identified by their id attribute, would be selected. Mathematically the predicate can be expressed by:

$$A \Rightarrow \neg B \wedge B \Rightarrow \neg A \qquad (1)$$

where A and B are both options.

Intuitively these constraints can be applied in situations where features have discrete options as in the example. Features with contiguous values, such a number of copies, can be dealt with by attaching values to ranges e.g. [0..10].

Conclusion

The proposed job ticket approach allows for a better modeling of the capabilities of the document service provider. The behavior of the application eases the specification of job tickets by showing the user only the meaningful options. The dynamic ticket explains the potential consequences of the selection of options to the user. The document service provider receives tickets with less or no errors.

The architecture of the application enables run-time configuration of features and constraints and decouples the presentation and the application logic.

References

1. Andre Tost, Creating presentation markup using XML, XSLT and the MCV design pattern, http://www.ibm.com.
2. Erich Gamma et al., Design Patterns, Addison-Wesley, 1995.
3. JDF, Job Definition Format, http://www.cip4.org
4. XincaML, http://www.alphaworks.ibm.com/tech/xincaml.
5. W3C, http://www.w3.org.

Biographies

Kees Jan Sonnenberg was born in 1974 in Zwolle, the Netherlands. He received his B.S. degree in Chemistry and Computer Science. Currently he is a graduate student in Computer Science at the Eindhoven University of Technology, the Netherlands. Since 1997 he is employed at Océ's Research & Development facility in Venlo, the Netherlands. His research interests include workflow management, color management, XML and driver technology. Kees Jan can be contacted by e-mail at cso@oce.nl.

Teun Willems was born in 1975 in Heeze, the Netherlands. He studied Computer Science and received his bachelor degree in September 1998. Since then he also works at Océ Research & Development in Venlo. His interests include XML, workflow, databases, data modeling, software architectures, Web technology and user interface/interaction. He can be contacted by e-mail at twil@oce.nl.