Chapter 5

Operational Semantics of NSP

In this chapter an operational semantics of NSP is given by describing a transformation of NSP components to JSP [143] pages, JavaScript [73][130] and Java 2 Platform API [166] objects. The operational semantics considers only NSP components that form a correct system with respect to the static semantics of NSP.

Basic Transformations. An NSP server page is translated into a JSP server page of same name up to technical details like renaming the file's ending. The resulting document is pure XHTML up to the exceptions described in section 3.6 plus Java code inside scriptlet signs. For web server pages the result is a complete XHTML page, for include server pages it is a valid XHMTL fragment, that may occur somewhere in a document body. In NSP both programming language declarations and programming language code are placed inside the same kind of tag, they are distinguished by their occurrence. NSP declarations are mapped to JSP declaration scripting elements, NSP Java code is mapped to JSP scriptlets. NSP Java expressions are mapped to JSP expression scripting elements.

The transformation resolves the differences between NSP and XHTML. An element that represents an XHTML attribute is mapped to this attribute in the respective encompassing element.

An NSP form is mapped to a JSP form. In case that the NSP form does not contain a submit button that specifies a callee different from the encompassing form's callee, the callee-attribute is mapped to the XHTML form's action-attribute. Thereby the attribute's value must be added to a given constant URL representing the site location, because the NSP form's action-attribute specifies only the name of a targeted server page. If the NSP form contains submit buttons with own callees, the form is mapped to a JSP form that targets a specific dispatching front component. An NSP submit button is mapped to an HTML input element of submit-type. The callee of a submit button is mapped to a name/value pair of the corresponding HTML input element, whereas the

NSP reserved word "callee" is chosen as name. The name/value pair of an HTML submit button is only transmitted, if the form has been submitted with this button. Therefore the receiving dispatching front component can use the "callee"/callee pair to forward the form's request to the correct web server page.
Transforming NSP Controls and Web Signatures. An NSP direct input control is mapped to a JSP input control with the type-attribute set to text. If a formal parameter of basic type is targeted, the mapping of the param-attribute is straightforward, it just becomes the name-attribute. Handling arrays and complex form data is explained below. The input-element's content, which possibly specifies a default value, becomes the value-attribute. NSP direct input controls and single select controls are active: if data entry must be type checked or if data entry respective selection is required, a JavaScript function is generated that is assigned to the onblur-event handler of the respective JSP input control. If the input control looses its focus, this function checks if the requirements are fulfilled and possibly pushes an appropriate warning to the user. More importantly a JavaScript validation function is assigned to the onSubmit-event handler of each form. JavaScript for checking requirements will be generated as part of that validation function. On submit, if a constraint violation has occurred, all violations will be reported and the form is being prevented from being submitted.

The other NSP controls are mapped to JSP controls accordingly. The transformation is interesting with respect to the values of arbitrary type. In the NSP approach arbitrary objects can be passed virtually to the user agent and back to a dialogue method on submit. Values of primitive type pose no problem, they can just be passed as Strings, whereas they are converted twice. For objects of non-primitive type an object reference must be passed virtually across the web user agent. If the object that should be passed is serializable an appropriately deep copy must be produced first. Then an object reference to this copy must be virtually passed across the web. For this purpose a surrogate string key is generated for the object reference in quest, the reference is stored under this key in a hashtable. The key is passed as a usual textual hidden parameter, select menu item, radio button value or check box value. If the key is received by a dialogue method it is used to retrieve the actual object from the hashtable.

If an NSP control targets a formal parameter of basic type, its param-attribute is mapped to the name-attribute of the JSP control. If an NSP control targets a formal array parameter, its param-attribute is mapped to the name-attribute of the JSP control again. But this time an index is attached to the control's value behind a special sign. An exception to this is the NSP element for a multiple select. The contained option elements take over the role of controls and their values are treated in the way just described.

An NSP control may be used in order to provide data for an attribute of an object net, that is for an item in an actual parameter of form message type. Such a control occurs underneath at least one NSP object-element. All object-elements and controls that together provide an actual parameter determine a labeled finitely branching tree. Leafs are given by controls, labels are given by form message attribute names or indexed attribute names in case of array attributes. Each control in such a tree determines a path expression made

of labels. This path expression is concatenated to the actual control's value behind a special sign. The HTML name attribute is the param-attribute of the outermost NSP object-element that contains the NSP control in quest.

A param tag of a web server page is mapped to a get method on the Servlet request object that retrieves the value stored under the formal parameter's name. In case of a formal parameter of basic type the getParameter-method is executed. If the retrieved value has primitive type or has been provided by a direct input widget it is parsed with respect to the formal parameter's type. If the retrieved value is a key that represents an object that has been passed virtually across the net, it is used to get this object from the respective hashtable. In case of a formal array parameter a getParameterValues-method is executed and the index information is taken from the retrieved values. Based on the obtained values and index information an array is constructed. Similarly in case of a formal parameter of form message type the getParameterValues-method is executed and the path expressions, which specify the locations of the items in a complex object net, are taken from the retrieved values. Based on the obtained values and construction information an object net is instantiated.
Transforming Hyperlinks, Server-Side Calls, and Higher Order Server Pages. An NSP hyperlink is mapped to a JSP hyperlink. The form's callee-attribute is mapped to the XHTML form's href-attribute. Again the the site URL must be added to the callee. Furthermore the actual parameters of the NSP hyperlink, which are given as hidden parameters, must be added to the action URL as name/value pairs in the special XHTML syntax based on question mark, ampersand and equal sign.

An NSP call to an include server page is mapped to the JSP include-action. The callee-attribute of the NSP call-element is combined with the site URL and becomes the page-attribute of the JSP include-action. An actual parameter object of the call is stored into a hashtable under the targeted parameter's name. It is retrieved by the targeted server page.

Formal parameters of page type are used inside NSP form, link, and call elements tags. These usages must be enclosed in JSP expression scripting elements in the images of these tags under the described transformation.