Chapter 2

Next Server Pages Preliminaries

The Next Server Pages technology (NSP) [57][62][127] is presented by the concrete amalgamation of NSP concepts with the programming language Java [79]. In the NSP approach a server page is considered to be code of a programming language with defined syntax and defined type system. In contrast, Java Server Pages (JSP) [143][20] code is just a convenient notation for Java Servlets [49]. It is essentially a mix of HTML and Java code, which may occur inside special opening respective closing scriptlet signs[1]. The semantics of a JSP server page is given by the effect of a preprocessor that yields a Java Servlet by placing the HTML parts of the server page into output statements and adding these statements to the Java code that occurs inside the scriptlet signs.

2.1 A Motivating Example

The improvements of the Servlet API over raw CGI [38] programming, support for retrieving values and a session mechanism, are available to JSP developers as a matter of course, anyway JSP technology has the same disadvantages as other CGI based technologies, as described in the introduction: JSP based systems may lack client page description and type safety. JSP does not offer a natural server-side call to server scripting code[2].

As an instructive example consider the system that comprises the two server pages given in listing 2.1 and listing 2.2. The first server page generates a registration client page that contains a form for gathering a customer's name

[1] Actually beyond being a convenient notation JSP technology offers some sophisticated mechanisms like tag libraries and JSP actions , especially such for supporting the integration of Java Beans into a server page. Furthermore JSP is intended to foster techniques for separating content from layout. But all these issues are orthogonal to the current discussion.

[2] The combination of the JSP actions `jsp:include` and `jsp:param` does not provide a usual type-safe parameter passing mechanism, it just provides a convenient notation for the low level Java Servlet include mechanism.

Listing 2.1

```
01 <html>
02   <head>
03     <title>Registration</title>
04   </head>
05   <body> <%
06     int j;
07     boolean c;
08     // computation of the variables j and c
09     for (int i=0; i<j; i++) { %>
10     <form action="http://localhost:8080/NewCustomer.jsp"> <%}%>
11       Name: <%
12       if (c) { %>
13            <input type="text" name="customer"><br> <%
14       } %>
15       Age: <input type="text" name="age"><br>
16       <input type="submit">
17     </form>
18   </body>
19 </html>
```

and age. This form targets a server page, that stores the received data to a database and forwards the request to another server page. Several dynamic errors may occur. First of all, the opening form tag in line 10 of listing 2.1 occurs inside a loop body. Only if the variable `j` has value `1` before the loop is entered, a valid HTML document is sent to the browser. The input capability for the customer name occurs underneath a control structure in line 13. Only if the boolean variable `c` evaluates to `true` the input capability will actually occur on the registration page. Listing 2.3 shows the result page for the case that variable `j` has value `3` and variable `c` evaluates to `false`. The fact that the form tag occurs three times in the client page is actually tolerated by some of the ubiquitous web browsers, nevertheless it must be considered a dynamic client page description error. For example, in the case that variable `j` has value `0` even a tolerant browser cannot recover from the error, because together with the form tag the vitally information about the targeted page is missing.

As an example for a dynamic client page type error, assume that the customer parameter of the business logic method in line 16 of listing 2.2 must not be a null object, i.e. the targeted server page relies on the reception of a customer name. However, it is not guaranteed that an input capability for the customer name is offered to the user by the registration page. If no such input capability occurs as in listing 2.3, no respective value is sent on submitting the form and consequently the effort to retrieve such a value in line 10 of listing 2.2 will result in yielding a null object.

Even more obviously, the developer would like to have a tool that gives a warning with respect to line 09 in listing 2.2, because a value bound to name

Listing 2.2

```
01 <html>
02   <head>
03     <title>NewCustomer</title>
04   </head>
05   <body> <%!
06     import myBusinessModel.CustomerBase; %> <%
07     String customer;
08     int age;
09     String foobar = request.getParameter("foobar");
10     customer = request.getParameter("customer");
11     try {
12       age = Integer.parseInt(request.getParameter("age"));
13     } catch(NumberFormatException e){
14       // error handling
15     }
16     CustomerBase.createCustomer(customer,age);
17     // further business logic %>
18     <jsp:forward page="Somewhere.jsp"/>
19   </body>
20 </html>
```

foobar will never be provided by a form of the registration page[3].

Another kind of dynamic type error may occur with respect to the customer age parameter. A user may enter a value that is not a number, which is an event that must be caught by the developer. Appropriate error handling code, either through client-side or server-side scripting, must be provided, which is a tedious and error prone task.

2.2 The NSP Document Structure

NSP modifies and augments XHTML [172] due to the special needs of developing parameterized strongly typed server pages. The modifications and augmentations are succinctly defined by an XML DTD of Core NSP in appendix C.1. Most importantly, an implementation of NSP, be it a full fledged container or a reference implementation that compiles NSP code to JSP code, will always only send XHTML to the browser, and consequently NSP technology can be used immediately with existing standard browsers, i.e. no plug-ins are needed[4].

[3] The JSPick tool presented in chapter 6 is able to detect such sources of error.

[4] Though NSP is an XML document that is interlaced with imperative code, its objectives must not be confused with that of Extensible Server Pages (XSP) [117], a dynamic XML technology. XSP is an integral part of the XML publishing framework Cocoon [39], which targets separation of concerns between content, logic, and style of web publishing applications. In contrast, the NSP approach is not oriented too much towards building information architectures [155] for the time being, but towards improving the stability of enterprise information

Listing 2.3

```
01 <html>
02   <head>
03     <title>Registration</title>
04   </head>
05   <body>
06     <form action="http://localhost:8080/NewCustomer.jsp">
07     <form action="http://localhost:8080/NewCustomer.jsp">
08     <form action="http://localhost:8080/NewCustomer.jsp">
09       Name:
10       Age: <input type="text" name="age"><br>
11       <input type="submit">
12     </form>
13   </body>
14 </html>
```

The name NSP/Java is used for the concrete language that results from merging NSP concepts with the programming language Java. If it is obvious from the context that not only the entirety of NSP technology concepts is meant the term NSP is used for NSP/Java, too. An NSP server page consists of a signature definition, a java definition block and a core document. There are NSP server pages that may be called across the net by a form or a link, others may be called by another NSP server page on the server side in order to be included. For NSP server pages that may be called across the net the core document consists of a head and a body, for the other kind of server page the core document content is enclosed in include tags. A first example is given in listing 2.4 and listing 2.5,which shows the NSP counterpart of the customer registration system, which has been discussed in section 2.1.

The signature of a server page is defined with appropriate parameter tags. Attributes are used for specifying the name and the type of a formal parameter.

Definition 2.2.1 (Web signature) *A web signature is a record type which consists of the several specified formal parameters of an NSP server page as labeled components.*

We coin another term for the concept of web signature, namely formal superparameter. Analogously, an actual superparameter is the entirety of actual parameters provided by a form or a link targeting an NSP server page with respect to a given formal superparameter.

Definition 2.2.2 (Web server page) *A web server page is an NSP server page that may be called across the net by a form or a link.*

Definition 2.2.3 (Include server page) *An include server page is an NSP server page that may be called for inclusion into another NSP server page.*

systems [102] with a tiered ultra-thin client architecture.

Listing 2.4

```
01 <nsp name="Registration">
02   <html>
03     <head>
04       <title>Registration</title>
05     </head>
06     <body><java>
07       boolean c;
08       // computation of the variable c </java>
09       <form callee="NewCustomer"><java>
10         if (c) {</java>
11           <input type="String" param="customer"></input><java>
12         } else {</java>
13           <hidden param="customer">"DefaultName"</hidden><java>
14         } </java>
15         Age: <input type="int" param="age"></input><br>
16         <submit></submit>
17       </form>
18     </body>
19   </html>
20 </nsp>
```

The web server pages are the front components of a tiered system's presentation layer, therefore we use the term dialogue method synonymously for such server pages throughout the dissertation, especially to distinguish them from business methods. Similarly we use the term dialogue submethod for NSP include server pages.

In NSP server pages Java code is be placed inside java tags. Some Java code may be placed between the signature definition and the server page core. It is called Java declaration in accordance with JSP terminology and hosts code that is intentionally independent from the single server page invocation, like e.g. import declarations or definition of state that is shared by all the several sessions. Java code that occurs in the server page core is executed upon invocation of the server page. The server page parameters are accessible in the inline Java code. In addition to the tags it is possible to use java expression tags as a controlled variant of direct, i.e. Java coded, writing to the output stream[5].

In NSP no special non-XML syntax for expression scripting elements like the JSP <%= and %> signs is available. Because of that it is not possible

[5]Note that a static type system like NSP's cannot prevent such dynamic errors that result from using the output stream to send computed description languages tags to the browser directly. Generally, using the output stream for sending description language is considered bad style and may lead to dynamic errors. It is an NSP rule that the output stream must not be used in a way that corrupts the otherwise type and description safe NSP system. For example, if a corrupted form that is caused by a prohibited use of the output stream contains invalid input capabilities or a wrong number of input capabilities, this is considered just as a dynamic error, like division by zero.

Listing 2.5

```
01 <nsp name="NewCustomer">
03   <param name="customer" type="String"></param>
04   <param name="age" type="int"></param>
05   <java>import myBusinessModel.CustomerBase;</java>
06   <html>
07     <head>
08       <title>NewCustomer</title>
09     </head>
10     <body>
11       <java>
12         CustomerBase.createCustomer(customer,age);
13       </java>
14       <forward callee="Somewhere"></forward>
15     </body>
16   </html>
17 </nsp>
```

to generate NSP tag parts, especially attribute values may not be generated. Element properties that may have to be provided dynamically are supported by elements instead of attributes in NSP. As a result an NSP server page is a completely block structured document and therefore it is possible to give a precise convenient definition for the syntax of NSP, which is the essential basis for detection and exploration of further concepts of syntax analysis and syntax manipulation. Moreover an NSP server page is a valid XML [19] document. Therefore NSP will benefit immediately from all new techniques developed in the context of XML. In particular, NSP can be used in combination with XML style sheet technologies [34][126].

NSP Forms and Hyperlinks

An NSP form specifies the targeted dialogue method via a callee attribute. Somewhat similar to the parameter passing mechanism in ADA [171], a targeted formal parameter of a called method is explicitly referenced by its name. For this purpose the tags in line 13 and 15 of listing 2.4 have parameter attributes.

In the present simple example the form provides actual parameters exactly for the formal method parameters, either by user input or hidden parameters. In general the overall notion of type-safe calls of NSP methods demands for sophisticated guidelines and rules for writing forms presented in chapter 3 and a new innovative widget set presented in section 2.3.

The syntax of NSP hyperlinks follows the NSP form syntax, listing 2.6 provides an example. Therefore the NSP hyperlink syntax is very natural, i.e. no low-level, tedious handling with special signs is needed in order to use hyperlinks with parameters. An NSP link element must not contain any other code than hidden controls and one linkbody element that gives the link text, in fact it must

contain exactly one hidden control of correct type for every formal parameter of the targeted dialogue method.

Listing 2.6

```
01 <link callee="NewCustomer">
02   <hidden param="customer">
03     "John Q. Public"
04   </hidden>
05   <hidden param="age"> 32 </hidden>
06   <linkbody> underlined link name </linkbody>
07 </link>
```

A Motivating Example

All the code of the lines 7 to 15 of listing 2.2, which is needed for the reception of parameters, becomes a two-line web signature declaration in listing 2.5.

In listing 2.1 the opening form tag is a loop body, which may lead to dynamic errors. In NSP this is prevented from the outset by NSP syntax. An element may only occur as a whole, i.e. with both opening and closing tag underneath a control structure.

It is not ensured that the registration page generated by listing 2.1 offers an input control for the customer name. In contrast, in the NSP code in listing 2.4 it is for example not allowed to omit line 13 without violating the NSP coding guidelines, i.e. without provoking an error message at compile time.

Obviously faulty requests to non-existent parameters like in line 9 of listing 2.2 are not possible in NSP code, because NSP is strongly typed and all formal server page parameters have to be declared.

Furthermore in an NSP system it is statically ensured, that data provided by the user is dynamically checked. That is, a server side dynamic error like the one described for the age parameter in listing 2.2, would not be possible in an NSP system because of the concept of active controls, which is introduced in section 2.3.1 and 2.3.3.

2.3 NSP Interaction Controls

NSP supports the usual HMTL/XHTML controls, but they are confined and elaborated further due to the special needs of a type-safe server pages technology. Each control is supported by its own element. Each control has a param attribute[6], which is used to directly specify the name of the targeted formal parameter. Another usage of the param attribute is specifying a field of a complex actual parameter of form message type as explained in section 3.5.

[6]The param-attribute replaces the name-attribute of HTML/XHTML controls.

2.3.1 NSP Active Direct Input Controls

In NSP the input tag is used for a direct input field only[7]. NSP provides different direct input fields for different programming language types. That is NSP is typed already on the level of controls[8]. The type attribute is used to specify the type of a direct input field[9]. The direct input controls of NSP are active input controls in the following sense. A form that contains a direct input field cannot be submitted if the user has entered data that is not type-correct with respect to the field's type. Instead of that an error message is presented to the user. Furthermore in NSP a formal parameter can be specified to be required by the developer. Then a form that contains a direct input field that targets this formal parameter cannot be submitted if the user has entered no data in this field. Again an appropriate error message is presented to the user. Dynamic type checking of data provided by the user and dynamically ensuring data entry are ubiquitous problems in web interface development. The developer must provide solutions by either client-side scripting with one of the ECMAScript [70][96] derivatives or server-side scripting, a tedious and error-prone task. With the NSP notion of active direct input controls it is possible to statically ensure the desired dynamic checks.

In NSP/Java the Java types int, Integer, String and the type Date of the JDBC API [186] are supported by direct input fields. If data entry for a formal parameter is optional and no data is entered by the user, a null object is passed as actual parameter on submit. The single exception to this rule is the type String: if no data is entered in a field for an optional parameter the empty string is sent instead of a null object. For formal int parameters data entry is implicitly required[10]. There are two further direct input capabilities for the type String, i.e. a text area and a password field.

Related Work

The XForms standard [69] allows for specifying constraints on data gathered by a form. Type correctness and entry requirements are important instances of possible constraints. But the XForms approach goes beyond these. It is possible to specify arbitrary calculations and validations on the gathered data. An event model allows for the appropriate reaction on the violation of the given constraints. Similarly the PowerForms technology [16] defines a declarative lan-

[7] In HTML/XHMTL the input tag is used for a couple of conceptually unrelated controls, like direct text input, radio buttons, check boxes etc. Every of these controls has its specifics, especially the behavior of non-direct input controls differs fundamentally from the behavior of direct input controls. Therefore in NSP every control type becomes a markup language element.

[8] In HMTL/XHTML controls always only yield text, i.e. pure string data.

[9] In HMTL/XHTML the type attribute is used to specify a widget kind.

[10] The type int is a Java primitive type and no null object is available for signalizing that no data has been entered. Alternatively it would be possible to select a certain int value to take over the role of the null object, most probably zero. But then the receiving dialogue method cannot distinguish between an actually entered data and the event, that no data has been entered.

guage for expressing input formats and interdependencies between input fields. Then client side code is generated for ensuring the declared constraints. Both technologies allow for sophisticated constraints, however the notion of NSP active controls is different. In XForms and PowerForms constraints can be given with respect to forms. But it is not ensured, that all forms targeting a critical server side action actually prevent the same errors from occurring. In contrast, in NSP, strictly based on the typing of server pages, the constraints are given for formal parameters of dialogue methods. This way it is possible to statically ensure the desired dynamic checks, that is NSP active direct input controls are dynamically type safe in the sense that they make NSP based systems type safe with respect to dynamically entered data[11].

2.3.2 Actual Object Parameters

In the NSP approach arbitrary objects can be passed virtually across the web, i.e. passed to the user agent and passed to a dialogue method on submit in the sequel. An object can be passed this way as hidden parameter, item of a select menu, or value of a radio button or check box[12]. A value of primitive type is just copied to the user agent and to the dialogue method on submit, thereby preserving its present type. But NSP supports passing of objects of arbitrary non-primitive types. Thereby, somewhat similar to the RMI parameter passing mechanism [164], two different parameter passing semantics are supported. First, as a default, a reference to the object in quest is passed to the receiving dialogue method. As an alternative, if the passed object is serializable [165], the object is copied and a reference to this copy is passed to the receiving dialogue method. Based on the semantics of serialization arbitrary deep copies of object nets can be passed as parameters. At the extreme, choosing to pass an object net as a completely deep copy fully coincides with the notion of message in the sense of [178], i.e. data dictionary objects.

A hidden parameter, select menu item, radio button or check box value must be given as a Java expression. This expression must have the type of the encompassing control. The encompassing tags implicitly switch to a Java expression modus, i.e. no additional expression tags are needed in order to explicitly signalize a Java expression. Furthermore this rule applies wherever appropriate in order to give a Java object as a certain desired property of a control. For example a default value may be given for direct input fields. For this purpose input field tags may contain a type-correct Java expression[13] [14]. Again the input field tags implicitly switch to a Java expression modus.

[11] A programming language is dynamically type safe, if untrapped errors are prevented [27].

[12] In HTML/XHTML only text, i.e. string data, can be passed as hidden parameter, item of a select menu, or value of a radio button or check box. The JSP expression scripting element supports implicit coercion for data that is convertible to String in order to send it to the user agent. However the data is received as string value by the targeted server pages.

[13] The default value of a direct input field must match the input field's type or it must be the empty String.

[14] In HTML/XHTML the tag's value attribute is used for specifying the default value.

2.3.3 NSP Active Single Select Controls

NSP supports single select menus and multiple select menus, which are distinguished as usual by a multiple attribute. A select element has option content elements. The NSP option element has no value attribute, instead of this it has a value content element and a label content element. The value tags contain a Java expression for a selectable data item. The label tags contain a Java String expression that is displayed in the select menu control and denotes the respective data item. In NSP the radio button is supported by an element.

Single select menus and radio buttons are conceptually equal. If a single select control targets a formal parameter that is not an array parameter, it must be ensured that the user actually chooses one of the items. Therefore NSP select menus and radio buttons are active again: a form encompassing such a control cannot be submitted unless the user has finally chosen one of the items[15].

2.3.4 Auxiliary NSP Interaction Controls

NSP supports hidden parameters. Hidden parameters offer a way to overcome the stateless nature of the hypertext transfer protocol [72]. More importantly[16] hidden parameters enable arbitrary reuse of a dialogue method in contexts where only a part of its web signature data should be determined by user interaction.

NSP distinguishes between two check box concepts, that reflect the two different ways in which the HTML/XHTML check boxes are used. First, the check box element yields an actual boolean parameter, true for a checked check box, false for an unchecked check box. Second, a set of value check box elements can be used to offer the user a choice of items. In the value check box tags a selectable data item is given as a Java expression. Assume a set of value check boxes that target the same formal parameter. Such a value check box set is conceptually equal to a multiple select menu, because the user may check a couple of the offered alternatives. A value check box set yields a value array on submit.

[15] Both NSP radio buttons and NSP select menu options have checked attributes. Instead of employing activity for radio buttons and select menus the NSP approach could rely on the request for comment document on HTML [12] that specifies: if no radio button resp. select option is checked, the user agent chooses the first one as pre-selected. Unfortunately a lot of ubiquitous browsers do not implement this behavior. For this reason even the HTML specification [150] differs from [12] in this point. As a result a set of radio buttons or a single select menu may yield no actual parameter as a result. But exactly this must be prevented, if a formal parameter is targeted that requires data entry. There are two other solutions to this problem than active controls. First, an NSP container could dynamically ensure that a checked attribute is added when needed, that is it could simulate the behavior demanded by [12]. Second, the NSP type system could be extended by checks that forces the author to ensure that checked entities always are provided. Anyway single select capabilities with no pre-selected items are desired, simply in all situations where all of the items are equal candidates.

[16] Hidden parameters allow for maintaining state between client/server exchanges. Other opportunities for this are techniques based on client state persistence [105] and URL-rewriting. Anyway today's APIs offer high-level session-tracking mechanisms.

2.3.5 NSP Submit Button

In HTML/XHMTL the submit button can also specify a name/value pair[17]. This is an important facility that is used in web applications to offer the user a choice between different functionalities for the same form[18]. For this purpose the values that represent different functionalities must be defined and the targeted script must switch to the correct alternative on submit. The respective code quickly becomes fault-prone, it's design suffers an "ask what kind"[19] antipattern. In contrast, in the NSP approach it is possible to specify a targeted dialogue method for each submit button. The NSP submit element has an optional callee attribute for this purpose. If no callee-attribute is given for a submit-element, the callee-attribute of the encompassing form is inherited. Furthermore a submit-element can contain hidden-parameters, again NSP gains from its solid theoretical basis: a form can target a variety of different dialogue methods providing different functionality and possessing different web signatures. This allows for flexible and at the same time robust design.

[17] In HTML/XHMTL the submit button is subsumed under the input element.
[18] A form is allowed to have several submit buttons.
[19] We take a generalized view of "don't ask what kind" as explained in 8.1.2.