Chapter 1

Introduction

In this dissertation a strongly typed server pages technology is proposed. Server pages technology is a state-of-the-art in the field of web technology.

Web applications are ubiquitous. For every company that wants to stay competitive it is not the question whether to deploy internet technology, but how to deploy it [146]. Highly skilled workers are needed to operate e-commerce technology [183], indeed a research study [77] conducted by the Gartner Group has shown that more than three-quarter of the cost of building an e-commerce site is labor related. Consequently web technology is worth looking at.

The viewpoint of this dissertation is a software architect's and a theoretical computer scientist's viewpoint. Ultra-thin client based tiered enterprise applications benefit from scalability and maintainability. Server pages technologies are widely used in the implementation of ultra-thin client applications. Unfortunately the low-level CGI programming model shines through in these technologies, especially user data is gathered in a completely untyped manner. In this work a strongly typed server pages technology is designed from scratch. The contributions target stability and reusability of server pages based systems. The findings are programming language independent. The results are formalized. The following concepts are combined:

- Parameterized server pages. A server page possesses a specified signature that consists of formal parameters, which are native typed with respect to a type system of a high-level programming language.

- Support for complex types in writing forms. New structured tags are offered for gathering arrays and objects of user defined types.

- Exchanging objects across the web user agent. Server side programmed objects may be actual form parameters and therefore passed to client pages and back, either as messages or virtually as objects.

- Higher order server pages. Server pages may be actual form parameters.

- Statically ensured client page type safety. The type correct interplay of dynamically generated forms and targeted server pages is checked at compile-time.
- Statically ensured client page description safety. It is checked at compile-time if generated page descriptions always are valid with respect to a defined client page description language.
- No unresolved links. It is statically ensured that all generated forms and links point to existing targets within the system dialogue.
- Active controls. NSP direct input controls are dynamically type safe, that is dynamic type checks with respect to data entered by the user are statically ensured. Equally checks with respect to required data are statically ensured.

The proposed server pages technology NSP (Next Server Pages) does not only overcome drawbacks of current CGI based technologies and helps the developer immediately to code cleaner, better reusable, more stable systems. The NSP approach is oriented towards improved web application architecture and improved development techniques from the outset:

- Enabling technology for improved web-based application architecture and design. In NSP a server-side call to a server page is designed as a parameter-passing procedure call, too. This enables functional decomposition of server pages and therefore helps decoupling architectural issues and implementing design patterns.
- Reverse engineering. The NSP concepts are exploited by the fully implemented reverse engineering tool JSPick, which recovers web signatures and form types from Java Server Pages based presentation layers. A formal semantics of the tool is given in pseudo-evaluation style.
- Seamless integration with form-oriented analysis. There exist canonical mappings from form charts, which model systems as constraint language annotated bipartite state transition diagrams, to NSP based systems.
- Formal semantic basis for type safety. The core type system of NSP is given as a convenient Per Martin-Löf style type system. This enables precise reasoning about the NSP concepts.

The aforementioned concepts target to overcome certain drawbacks of CGI (common gateway interface) based technologies, which today provide the single most important means to build presentation layers of web-based systems, as explained in the sequel. A web-based system consists of a set of server side scripts. A script is a code unit that is called across the net by the user by submitting a form or selecting a link. On behalf of this the script triggers business logic and eventually produces a client page that is sent back to the user. The client

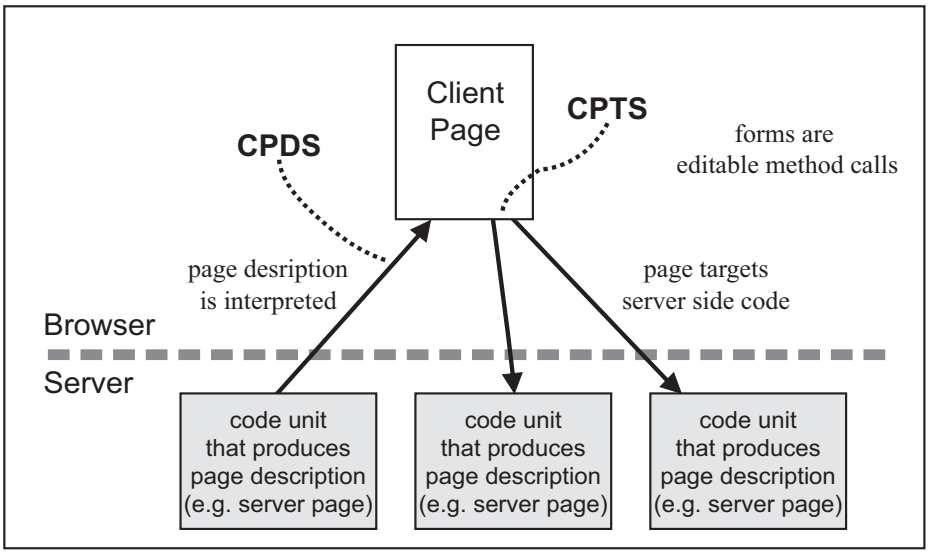

Figure 1.1: CPDS and CPTS. The figure shows the interplay of server side scripts and the browser and visualizes the notions of client page description safety (CPDS) and client page type safety (CPTS).

page is coded in the client page description language HTML or XHTML. The client page description is interpreted by a browser (Fig. 1.1). A page presents the user information and offers her one or more forms and links. Every form and every link targets one of the server-side scripts, that is the user gets explicit control over the dialogue flow. For the NSP approach it is crucial, that a form may be viewed as an editable method call, gathering actual parameters from the user. A form may have pre-selected actual parameters, hidden parameters in more technical terms. It follows that links can be subsumed conceptually under forms.

We coin the two terms *client page description safety* and *client page type safety* for two desired properties of web applications.

Definition 1.0.1 (Client page description safety CPDS) *A web application has the property of client page description safety if its server scripts always only produce valid page descriptions with respect to a defined client page description language.*

Definition 1.0.2 (Client page type safety CPTS) *A web application has the property of client page type safety if the client pages generated by the server scripts always only contain such forms and links, that provide exactly such actual parameters that are expected by the respective targeted server scripts.*

We use terms like client page type error, correctness, or checking according to definitions 1.0.1 and 1.0.2. We have chosen to use some more general, less

technical terms than might be actually needed at first sight like client page description language instead of HTML/XHTML[1]. We did so in order to foster the viewpoint, that web applications are instances of a more general class of ubiquitous applications, which may be characterized as submit/response style applications. Furthermore we wanted to emphasize that the concepts that are investigated in the context of NSP do not stuck to concrete web technology, but could add value to every future ultra-thin client architecture supporting technology.

Current CGI based server-side scripts receive values submitted by a form or link conceptually as stream of named values. In raw CGI programming, the name/value pairs must be retrieved from an input parameter string. The several well known scripting languages and technologies, like PHP [108], Perl [174], Tcl [137], Active Server Pages [110] [147], Python Server Pages [4], Java Servlets [49], and Java Server Pages [143][20] support the retrieval of submitted values with appropriate data structures. But beyond this, these technologies don't support any appropriate notion of a server script signature, i.e. the interesting types of the received string-valued parameters with respect to a business logic or, in more code-oriented terms, with respect to a high-level programming language type system are not supported.

In contrast, NSP offers appropriate tags to define a complex server page signature. Based on this the definition 1.0.2 of client page type safety obtains a precise meaning in the context of NSP, because parameters that are expected by a server page are just the parameters specified by the server page's signature. NSP provides both client page description checking and client page type checking. From the viewpoint of an NSP type system the generated client pages are the actual code, which has to be considered. The generated code is naturally not available at deployment time, therefore NSP defines guidelines and rules for writing the server pages which are

- non-prohibitive: all reasonable applications of scripting are still allowed.
- sufficient: client page description and type safety are ensured.
- convenient: the coding guidelines and rules target the NSP developer. They are natural and easy to understand. The coding guidelines and rules provide the informal definition of the NSP type system.

Conservative Amalgamation

The NSP concepts are programming language independent results. They are reusable. They must be amalgamated with a concrete programming language. For every such amalgamation a concrete non-trivial language mapping must be

[1] Actually NSP guarantees only a weaker client page description safety than valid HTML/XHTML output safety as explained in section 3.6, but this is due to pragmatic reasons and not due to technical reasons: furthermore the NSP notion of client page description safety is justified by existing browser technology.

provided. The NSP concepts are designed in such a way that concrete amalgamations are conservative with respect to the programming language. That is the semantics of the programming language and especially its type system remain unchanged in the resulting technology. In this dissertation the NSP concepts are explained through a concrete amalgamation with the programming language Java. As a result of conservative amalgamation the NSP approach does not restrict the potentials of JSP in any way, for example its state handling facility, the Servlet API session concept, is available as a matter of course. Formal semantics of an NSP core type system is given with respect to an amalgamation with a minimal imperative programming language.

Integration with Form Orientation

NSP based systems benefit from type safety. But beyond this NSP's capability for server pages functional decomposition suggests to reconsider web application architecture and design. In the NSP approach a server pages based presentation layer is characterized as a closed collection of typed dialogue methods, an abstract viewpoint that is shared by form-oriented analysis [178]. Based on functional decomposition it is possible to find canonical mappings from form charts [60] to NSP technology. Consequently NSP becomes an integral part of Form Orientation, a holistic approach for modeling and developing a certain kind of widespread form-based, submit/response style enterprise application, which contributes form storyboarding, form chart modeling, a dialogue constraint language, layered data modeling and the NSP technology.

Related Work

Detailed discussions of related work are provided throughout the paper. Furthermore a comprehensive appraisal of related work is given in section 8.1. To our knowledge the NSP project is the only approach that investigates client page description safety and client page type safety with respect to server pages technology. Thereby the results are programming language independent. The contributions are summarized in section 8.3.

There are two projects on imperative domain specific languages [53] for web programming, i.e. MAWL [8] and Bigwig [18] discussed in section 8.1.2. Several projects address active web publishing in the context of functional programming discussed in section 8.1.6.

It is necessary to mention the debatable important opinion [138] known as Ousterhout's dichotomy. A division is made between system programming languages for programming components from scratch and scripting languages for programming component glue. It is argued that the typeless nature of a scripting language is crucial - a property that enables rapid development. The objectives of the present work contradict this core argument of Ousterhout's dichotomy.

Dissertation Outline

The dissertation is organized as follows. Chapter 2 motivates the NSP approach and explains the overall structure of an NSP document as well as the NSP interaction controls, thereby introducing the concepts of active direct input control and active single select control. Chapter 3 explains the NSP support for gathering array data and data of user defined type. The NSP coding guidelines and rules are given, which together provide the informal description of the NSP type system. The NSP type system is specified in chapter 7 as a Per Martin-Löf style type system for Core NSP, which is the amalgamation of NSP concepts with a minimal imperative programming language and a sufficiently complex type system for modeling all relevant aspects of a modern high-level programming language's type system. Example type derivations for some Core NSP server pages are provided in appendix B. Furthermore this appendix summarizes abstract syntax and typing rules of the NSP definition given in chapter 7, whereas in appendix C an XML DTD for the Core NSP language and an SGML DTD for the Core NSP user interface description language are given. In chapter 4 the architecture that underlies current web application frameworks is analyzed. The concept of server-side call to a server page is introduced as well as the concept of higher order server pages. Motivations of the concepts are given by explanations of their capability to improve web application architecture. In chapter 5 an operational semantics of the NSP/Java language amalgamation is given, i.e. a core reference implementation is defined by the description of a transformation to Java Server Pages and auxiliary technology. In chapter 6 the JSPick reverse engineering tool for design recovery of Java Server Pages based presentation layers is presented. Chapter 8 outlines further directions and provides a detailed discussion of related work. Appendix A is a case study, that exemplifies how to transform a given form chart system specification to NSP code.

Acknowledgements

Special thanks to my *spiritus rector* Prof.Dr.Elfriede Fehr for making possible this dissertation. Special thanks to PD Dr.Martin Große-Rhode for the appraisal of the dissertation. His comments on early versions of the text helped to improve the quality of the dissertation significantly.

Thanks to my *alma mater* "Freie Universität Berlin - Institut für Informatik" for extraordinary good labor conditions and especially thanks to the several institute members who lent a helping hand.

Thanks to Gerald Weber for his ideas and collaboration.