Appendix B

The Gently Tool

Specification and Generation of JSP Dialogues

In this appendix we describe the language and the tool Gently for the specification of submit/response style systems and the subsequent generation of an executable interface prototype. A textual description of a form storyboard can be directly expressed in Gently, and is then automatically mapped onto a system structure in compliance with currently discussed web design patterns like the Model 2 architecture. However, Gently integrates specifically well with advanced server pages technologies like NSP [44]. The Gently tool leads over from specification to implementation, therefore in this appendix we give also an outline of the targeted technology, namely the JSP technology for implementing Web interfaces. We discuss the properties and problems of this technology for submit/response style systems.

The language Gently directly supports the system metaphor of Form-Oriented Analysis, namely the system model as a bipartite state machine.

B.1 The Advantages of Gently

In this section and the following section we give arguments for a language and tool like Gently directly from the standpoint of the design phase, thus adding a new motivation to our conceptual views on submit/response style systems. We describe, why the working developer of servlet-based web interfaces can gain advantages for the everyday problems by using Gently.

Servlets give access to CGI style parameter passing through an object-oriented mechanism. Commonly accepted documentation and testing concepts, when used naïvely in the context of developing a servlet based ultra-thin client tier, may consider only the purely technical parameters (HTTPRequest, HTTPResponse) instead of the CGI parameters significant for the business logic.

Moreover, the servlet concept is open for polymorphic use of servlets. One servlet can be designed to respond to different sets of CGI parameters. However,

the servlet mechanism does not perform any checking on the parameter set.

The formal language called Gently defines a discipline restricting the use of this mechanism to a strongly typed concept which is in compliance with best practices. This language serves as input to a JSP generator, producing templates for a type safe dialogue. Gently allows at the same time to document a JSP based dialogue.

More formally, our language enables the specification of the system as the signature of a static object system. The template generator maps each object to one JSP and each dialogue method offered by this object to a protected region within this JSP.

In Gently, form storyboards can be directly expressed, and the bipartite structure of the specification is checked by the Gently tool. The bipartite structure of the form storyboard is matched with the so called Model 2 Design pattern, and therefore supports current state-of-the-art web interface development.

B.2 Introduction to the Viewpoint of Gently

We consider systems with HTTP interface in which the main dialogue functionality is based on basic HTML form techniques or in which the dialogue can be viewed as such. We will use an online bookshop as a running example. We consider the implementation of such systems with Java Server Pages (JSP), which is a proposed standard technique to implement HTTPServlets [2][11][5]. The interface of such a system is used in a request/response style by requesting single pages. For simplicity, we suppose that the system is based completely on pages generated by JSPs, and we will call all pages of the system together the pageset of the interface. Technical differences between JSP and servlets are of minor interest in the context of this paper, so we feel free to use these terms mostly synonymously.

The HTML form standard offers an untyped, textbased remote call mechanism, which is used as the standard parameter passing mechanism for dynamic websites. This mechanism is commonly, but not quite correctly, referred to as CGI parameter mechanism. We prefer *HTML form parameters* in the following as precise term for those parameters. The mechanism can be used in HTTP links as well. Servlets offer HTML form parameters to custom code via the HTTPRequest parameter.

The user of a HTTP dialogue can be viewed as invoking methods on the system via the browser. However, it is not helpful to view the user as invoking the `doGet` (or e.g. `doPost`) methods of the servlet: Applying standard documentation to the customized servlet classes leads only to the documentation of the formal `doGet` methods. The interesting parameters however are the HTML form parameters provided by the forms and links calling the page, which have to be recognized by the custom code.

Each customized `doGet` implementation may be able to understand different parameter sets. In other words, the servlet mechanism is open for overloading.

We define programming guidelines limiting the degrees of freedom in accordance with common best practices, and we define a precise specification methodology, namely a formal language for pagesets.

The programming guidelines demand that each servlet must respond only to a fixed set of parameter lists and that the decision must be based on a hidden parameter. Hence, for the specification language, we view each JSP as a static object offering a set of methods with fixed parameter set.

A whole HTTP dialogue can therefore be seen as the signature of a static object system, each page being an object. Our language will allow the specification of such a signature and the generation of templates for the JSPs of this system.

The described static object system signature is called the dialogue signature of the system. Each method within that signature is called a dialogue method. In well-designed systems, dialogue methods and business methods live on different tiers. Dialogue methods invoke business methods. So the dialogue can be changed without changing the business logic.

Dialogue methods produce result pages. Note that therefore one JSP will potentially generate as many different result pages as it has methods.

Each result page offers to the user the choice between different calls to dialogue methods as the next step in the dialogue. Dialogue methods can be called by links or forms. Forms can be seen as editable method calls offered to the user.

For each dialogue method, our language will permit to specify the links and forms contained in the result page. The syntax of these declarations is derived from the Java method syntax.

From each specified method declaration and call the generator will produce appropriate code templates as described below.

B.3 The Language Gently

The language Gently is a specification language for HTML dialogues. Its type system and syntax are oriented towards Java. Gently allows the specification of a HTML dialogue as a static object system. The signature of this object system is expressed in Gently in a single source file. This source file is conceived as the place of documentation of the functionality of the HTML interface. The documentation for the JSPs shall be provided in the Gently file, and shall be ruled by the design by contract paradigm [9].

The Gently generator performs static type checking on this specification and generates JSP templates, containing protected regions. The JSP templates can be completed by inserting custom code into the protected regions.

Gently considers the different pages of a JSP interface as the static objects which constitute the interface. Each group is specified in the Gently file in a syntax element `group` similar to the `class` construct in Java. Each JSP can have a set of different methods with different parameter signatures. We have introduced this option in order to permit the invocation of one JSP with

different parameter sets. As a result we obtain a two level hierarchical structure of the services offered by a HTML interface. It is not our intention to prescribe a special usage of this feature. The programmer is free to decide about the distribution of functionality over different JSPs.

Each method within a `group` construct has a fixed parameter set. The method construct in Gently is again similar to the method syntax in Java.

In the body of Gently's method construct, two types of declarations have to be provided. In one type of declaration the programmer has to specify all links and forms which shall be offered by the result page of this method. Since links and forms enable calls of dialogue methods, their declarations resemble method calls. They start with the keywords `link` or `form`, they get a label, which enables references to them, and finally they have a method call syntax, e.g.

```
link myLabel calls myGroup.myMethod(myActPar1,..,myActParN).
```

For links, the actual parameters are names of variables. Gently generates a protected region in which variables with these names are accesible (the declaration is placed in front of the protected region). The custom code should assign values to these variables. In the code subsequent to the proteced region, these variables are used to construct an appropriate HTML-link code.

In `form` declarations each actual parameter is a pair of the widget type that shall be generated and a variable name. The values of these variables are used to provide the default values within the widgets.

These `link` or `form` declarations serve as specifications of the possible next actions the user can take.

Our intention with Gently is to maximize local specification value, therefore we demand the other type of declaration in each method, the `called by` declarations. By such declarations the programmer has to specify all places in the dialogue, where links to this method are offered to the user, i.e. all methods in which this method is called in a `link` or `form` declaration.

Naturally, one has to place `called by` declarations in front of `link` or `form` declarations. Both declaration types are separated by `and`, declarations of one type are separated by `or`.

Alternatively, in other versions of the tool, `called by` declarations or even an appropriate state chart visualization could be generated, too.

Further important Gently constructs will be explained directly in the upcoming example.

B.3.1 Two Staged Request Processing

Gently directly supports the system paradigm established in Form-Oriented Analysis, in which the system is modeled as a bipartite state machine. The action and page declarations in Gently enable a two-staged request processing, wich fits to the bipartite structure of Form-Oriented Analysis. The current Gently tool output is consistent with SUNs Model2 [11] as a JSP design pattern. Due to the generative approach of the Gently tool it is however easily possible to support advanced and more flexible system designs like the NSP approach

presented in [44]. Gently offers two method modifiers, `action` and `page`, which qualifies the respective method as corresponding to a server action or client page. A `page` method must call only `action` methods, and only via `link` or `form` declarations. A `action` method must call only `page` methods, and only via `redirect` declarations. The Gently tool checks, whether the specification complies with the given rules, and whether all methods have a modifier.

B.3.2 Generation of Code

The language Gently serves as a formal specification language, but also as input to the Gently tool, a type checker and generator. The Gently tool performs the following static checks:

- It checks whether all calls to methods (in the result pages of other methods) have correct parameter sets. This is important especially for forms. Here the tool checks whether the widgets chosen in each actual parameter matches the type of the corresponding formal parameter.
- It checks whether every `link` or `form` declaration is matched by a corresponding `called by` declaration in the method called.
- It checks, whether every `action` calls only `page` targets via `redirect` declarations, and whether every `page` calls only `action` targets via `link` or `form` declarations.

For correct input, the Gently generator produces the following output code:

- For each `group` construct it produces one Java Server Page with the same name.
- For each method within a `group` construct it produces one block in the Java Server Page. This block is executed whenever the appropriate Gently method is called. Within that block it produces code which performs runtime type checking whether the method is called with correct parameters, and converts the parameters to Java local variables with the same name as the formal parameters in the Gently file. Subsequently it creates a protected region which can be used as method body for the Java code.
- For every `link` or `form` declaration in the Gently method body it creates HTML code, giving a form or link which offers a correct call to the corresponding method. Within that HTML code, the Gently tool again creates protected regions for each HTML parameter, defining where to insert Java code which produces actual parameters. Of course, the programmer could chose to generate some links with Gently and some other links manually. However, since Gently is not only a generation tool but also a specification language, proper documentation requires that every call to the dialogue methods must be specified within Gently.
- The Gently tool produces comments, structuring the generated code.

The Gently tool is developed in Java with the compiler generator JavaCC [8]. It is build using standard compiler construction techniques [15] combined with proposed object-oriented patterns [6].

B.3.3 The Seminar Registration System in Gently

We now give the Gently code for the running example, the seminar registration system. The example makes use of the partitioning concept of Gently, by binding logically connected groups of states together. The example shows also the usage of the `menu` construct. This construct may contain `link` and `form` declarations, which will be included on every result page. It is used to realize the back link in the example. In the specification for the seminar registration here, the back link is contained on every page, including the home page.

```
menu{
   link home   calls Home.home()
}

group Home{
  action home()
       called by home from menu
    and
       redirect Home.list()
  }
  page list()
       called by home from Home.newLink
    or called by redirect from New.newForm
    or called by redirect from Change.changeForm
    or called by redirect from Delete.deleteForm
    and
       link delete calls Delete.deleteLink(person)
    or link change calls Change.changeLink(person)
    or link change calls New.newLink()
  }
}

group Delete{
  action deleteLink(Person selected)
       called by delete from Home.list
    or called by delete from Change.changePage
    and
       redirect Delete.deletePage(selected)
  }
  page deletePage(Person selected)
       called by redirect from Delete.deleteLink
    or called by redirect from Delete.deleteForm
    and
       form confirm calls Delete.deleteForm(PASSWORD passwd)
```

```
  }
  action deleteForm(Password passwd)
       called by confirm  from Delete.deleteLink
    and
       redirect Delete.deletePage(selected)
    or redirect Home.list()
  }
}

group Change{
  action changeLink(Person selected)
       called by change from Home.list
    and
       redirect Change.changePage(selected)
  }
  page changePage(Person person)
       called by redirect from Change.changeLink
    or called by redirect from Change.changeForm
    and
       link delete calls Delete.deleteLink(person)
    or form submit calls Change.changeForm(HIDDEN person,
                                           TEXTFIELD name,
                                           TEXTFIELD phone,
                                           TEXTFIELD studentID,
                                           PASSWORD passwd)
  }
  action changeForm(Person person, String name, String phone,
                    int studentID, Password passwd)
       called by submit  from Change.changeLink
    and
       redirect Change.changePage(person)
    or redirect Home.list()
  }
}

group New{
  action newLink()
       called by new from Home.list
    and
       redirect New.newPage()
  }
  page newPage()
        called by redirect from New.newLink
     or called by redirect from New.newForm
    and
       link home   calls Home.home()
    or form submit calls New.newForm(      TEXTFIELD name,
                                           TEXTFIELD phone,
                                           TEXTFIELD studentID,
                                           PASSWORD passwd,
```

```
                                          PASSWORD passwd2)
  }
  action newForm(String name, String phone, int studentID,
                 Password passwd, Password passwd2)
       called by submit  from New.newLink
    and
       redirect New.newPage()
    or redirect Home.list()
  }
}
```

B.4 The use of the menu concept

The menu concept has become widespread in the domain of web technology. In this section we distinguish the menu concept from the mere technical concept of the browser history.

User interaction with the web browser can be divided into two classes: Interaction on one page, like filling out a form. We have called this *page interaction*. For itself it is a purely client-side activity, it is not of interest for our considerations, which apply to the server side. On the other hand we consider interaction changing the currently shown page, we call this *page change*. In case of submitting forms, user input is transferred with this request.

As a consequence of this, HTTP interfaces can be specified as a set of pages modeled as states of a state machine and a set of page changes modeled as state changes on this machine. Hence one may think of using the state machine as documentation for the dialogue. But a closer look on many HTTP interfaces shows that they offer almost every state transition, so that state machine modeling becomes almost useless.

To circumvent this we propose a classification of page changes into three classes:

- browser-caused. These are state changes via the browser's history mechanism, bookmarks etc. These changes must be considered by the system designer because they must not make the system unstable. However, it is a quietly accepted pattern of HTML dialogue design that a user should not be forced to use such mechanisms in order to get a meaningful dialogue with the system.

- menu-supported. HTML interfaces typically have alongside the changing page a menu, which allows the user to branch fairly unrestricted in the system dialogue. The menu is a substitute for the browser history that is provided by the site itself and built with insight into the semantics. But again, the user should at most in exceptional cases be forced to use the menu. The menu should be always at hand for the user's convenience, but the system should propose the most usual state changes within the page itself.

- page-guided. These page changes are caused by links or buttons on the page and constitute the main alternatives. Navigation along these state changes is what constitutes the use cases [12]. Only these are amenable to meaningful state machine modeling.

Gently supports this classification with the `menu` concept.

B.4.1 Modeling the Use Cases

We consider the distinction between page-guided state changes and the other classes and the reduction of use cases to dialogues along page-guided state changes as essential for a mature model for HTML dialogues.

This result is of course not restricted to HTML/HTTP as special technology. In fact it is valid for an abstract class of ultra-thin clients. They can be summarized as

- page-based clients, in contrast to GUIs based on recursive container structures for GUI elements, where almost every GUI element can have a state of its own.
- pull-based, i.e. especially the page changes are user driven. Nevertheless elements of the page may be fed with pushed information (a continuously updated curve for example).
- Non-modal: the user can suspend or leave the use case and must not stick to page-guided changes. However, restricting the interaction to the page-guided changes must give a meaningful modal dialogue.

This more abstract model of ultra-thin clients could be called the abstract browser.

B.4.2 Working with Gently

In order to develop an Interface with Gently, perform the following steps:

- Use Form-Oriented Analysis to achieve at least a form storyboard of the system, preferably a form chart.
- Encode the resulting form storyboard in Gently.
- Invoke the Gently generator and obtain a set of JSP pages. Within these pages, you find protected regions, in which you are allowed to write code. There are two kinds of protected regions offered by Gently: Regions for HTML coding and regions for Java programming. Use OCL tools for transforming the OCL specification.
- Gently facilitates complete separation of HTML coding and Java programming. The generator offers a switch, which triggers generation of separate Java Interfaces. Instead of editing the second kind of protected regions as described above, all Java custom code then has to be provided following the Hollywood Principle [14].

B.5 Related Work

In contrast to other approaches [13] we do not adress the domain expert but the professional software engineer in a code intensive working environment [1] instead.

The functionality of Gently is different from tools like [10], which provide an interconnection between different language paradigms like HTML and SQL. Gently takes one signature specification and generates code in the different paradigms, Java and HTML, according to this signature specification.

Gently does not prescribe architecture [7] or design. For example, we have discussed how Gently directly supports SUNs Model2 [11] as JSP architecture. As an example for JSP design, one could imagine the commonly accepted application of the GoF-State-Pattern [6] to encapsulate the workflow of a user session in a single servlet. As one may check against these two examples, our proposal neither encourages nor discourages concrete architectures or designs. Instead, it starts as a best practice for documenting JSP based systems and ends up as a development discipline for such systems.

This is in contrast to, for example, the Struts framework [4][2], which leads to very particular Model 2 architectures only. Struts suggests dynamic type checking. Gently on the other hand consequently performs static type checking and ensures that the resulting system is statically well typed [3].

With respect to performance, the use of Gently leads to almost no overhead. The main function of Gently is not to generate Java statements, but to generate matching HTML and Java signatures. Additionally Gently generates conversion code and dynamic type checking, which serves as an additional firewall against invalid calls to the system. This code must be provided in any stable system implementation, even if Gently is not used. The code has minimal execution time.

It is noteworthy that the resulting approach is amenable to generalization, as pointed out in the chapter on further work.

B.6 Gently Contributions

As starting point for our contribution we have observed:

- HTTP offers a page based dialogue paradigm.
- HTML forms introduce a remote call mechanism with key-value lists as parameter sets.
- Servlets do not enforce strong typing, they do not control overloading.
- The servlet mechanism has no inbuilt rejection mechanism for invalid requests (no runtime type checking).
- Naive Java documentation misses the point for servlets.

We proposed specification guidelines based on a formal language Gently and a template generator taking Gently as input. Our main paradigm is that we view a system with HTML interface as a static object system that is described by its signature. The use of Gently offers the following advantages:

- A type system for HTTP requests connected to the Java type system, and support for the enforcement of this type system.
- Generation of runtime type checks according to the type system.
- Generation of HTTP requests in compliance with the type system.
- Generation of a variety of input forms for each signature as well as generation of links.

B.7 Gently Grammar

B.7.1 Gently BNF

```
syntax
 ::=
 <LOCATION> <EQUALS> <LOCATOR> groups ( menu )?

groups
 ::=
 ( group )+

group
 ::=
 <GROUP>  <IDENTIFIER> <CURLYOPEN> methods <CURLYCLOSE>

methods
 ::=
 ( method )*

method
 ::=
 ( <ACTION> | <PAGE>  )? <IDENTIFIER> <OPEN> parameters <CLOSE>
 <CURLYOPEN> ( ( callers ( <AND> calls )? ) | ( calls ) )? <CURLYCLOSE>

parameters
 ::=
 ( parameter ( <COMMA> parameter )* )?

parameter
 ::=
 (  ( <BOOLEAN> <IDENTIFIER> )
   |(  ( <INT> | <STRING> )
       (  ( <ARRAYOPEN> <ARRAYCLOSE> <IDENTIFIER> )
         |( <IDENTIFIER> ( <ARRAYOPEN> <ARRAYCLOSE> )? ) ) ) )

callers
 ::=
 ( caller ( <OR> caller )* )

caller
 ::=
 <CALLED> <BY>
 (   ( <REDIRECT> <FROM> <IDENTIFIER> <DOT> <IDENTIFIER> )
   | ( <IDENTIFIER> <FROM> ( ( <IDENTIFIER> <DOT> <IDENTIFIER> )
   |  <MENU> ) ) )

calls
 ::=
```

```
 ( call ( <OR> call )* )

call
 ::=
 ( link | form | redirect )

redirect
 ::=
 <REDIRECT> <IDENTIFIER> <DOT> <IDENTIFIER>
 <OPEN> actualLinkParameters <CLOSE>

link
 ::=
 <LINK> <IDENTIFIER> <CALLS> <IDENTIFIER> <DOT> <IDENTIFIER>
 <OPEN> actualLinkParameters <CLOSE>

actualLinkParameters
 ::=
 ( <IDENTIFIER> ( <COMMA> <IDENTIFIER> )* )?

form
 ::=
 <FORM> <IDENTIFIER> <CALLS> <IDENTIFIER> <DOT> <IDENTIFIER>
 <OPEN> actualFormParameters <CLOSE>

actualFormParameters
 ::=
 ( actualFormParameter ( <COMMA> actualFormParameter )* )?

actualFormParameter
 ::=
 ( <TEXT> | <TEXTAREA> | <CHECKBOX> | <RADIO> | <COMBOBOX> |
   <MULTIPLELIST> | <HIDDEN> ) <IDENTIFIER>

menu
 ::=
 <MENU> <CURLYOPEN> calls <CURLYCLOSE>
```

B.7.2 Gently Keywords

The Genty keywords encompass group specifiers, types, call specifiers, input field specifiers, signs and auxiliaries. Identifier are alphanumeric values with a leading letter. Comments in Gently are oriented towards Java. It is possible to comment areas with /* comment */, and the rest of a line with // rest of line.

- Group Specifiers

GROUP
MENU

- Types

  INT
  BOOLEAN
  STRING

- Call Specifiers

  CALLED
  BY
  FROM
  REDIRECT
  LINK
  FORM
  CALLS
  OR
  AND

- Input Field Specifiers

  TEXT
  TEXTAREA
  CHECKBOX
  RADIO
  COMBOBOX
  MULTIPLELIST
  HIDDEN

- Signs

  OPEN : "("
  CLOSE : ")"
  CURLYOPEN : "{"
  CURLYCLOSE : "}"
  ARRAYOPEN : "["
  ARRAYCLOSE : "]"
  DOT : "."
  COMMA : ","
  EQUALS : "="

- Auxiliary

  LOCATION