Advanced Control Flow (2.1 legacy document)

Table of contents

1 Comments.....4

Table of contents

1 A different approach.....3
1.1 What are continuations?.....3

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. A different approach

Web applications are essentially event-driven applications. Such applications have to react to events generated from the client browser, and they respond to these perhaps by changing their internal state and generating a response.

The result is that even a simple application that needs to collect some information from the user using more than one page, has to somehow maintain the input accumulated so far from the user. This input is a characteristic of the application state. Another characteristic of the application state is where the program processing is.

Let's look at an example. Suppose we want to write a very simple calculator, which collects the numbers to be added, as well as the operator, each in a separate page. It would be very nice if we could write something like this:

```
function calculator() { var a, b, operator; cocoon.sendPageAndWait("getA.html"); a =
cocoon.request.get("a"); cocoon.sendPageAndWait("getB.html"); b =
cocoon.request.get("b"); cocoon.sendPageAndWait("getOperator.html"); operator =
cocoon.request.get("op"); try { if (operator == "plus") cocoon.sendPage("result.html", {result:
a + b}); else if (operator == "minus") cocoon.sendPage("result.html", {result: a - b}); else if
(operator == "multiply") cocoon.sendPage("result.html", {result: a * b}); else if (operator ==
"divide") cocoon.sendPage("result.html", {result: a / b}); else
cocoon.sendPage("invalidOperator.html", {operator: operator}); } catch (exception) {
cocoon.sendPage("error.html", {message: "Operation failed: " + exception.toString()}); } }
```

In this example, the calculator function is called to start the calculator application. We'd like the sendPageAndWait function to be a special function, that takes as arguments an HTML file to be sent as response, and some optional data that needs to be placed dynamically in it. We would like sendPageAndWait to send the response page and then block the executing thread, until the user clicks on a link in the response page, which sends a request back to the server. This request resumes the processing at the point it was left, right after the call to sendPageAndWait.

This approach looks very powerful, since the flow of pages within the application can be described as a normal program. Using this approach you no longer have to think of your Web application as a finite state machine, which transitions from one state to another, and in the process generates response pages.

A big disadvantage of the approach above is that we need to maintain a thread alive until the user hits the link on the response page. This is clearly very expensive!

It would be very nice if we could capture the state of the application, its stack of function calls, which includes local variables, the global variables and the program counter, and save them into an object. If this object would give us the ability to restart the processing from the point stored in it, this would be what we need!

1.1. What are continuations?

A continuation is exactly the type of object that we need. Think of a continuation as an object that, for a given point in your program, contains a snapshot of the stack trace, including all the local variables, and the program counter. You can not only store these things in the continuation object, but also restore the execution of the program from a continuation object. This means that the stack trace and the

program counter of the running program become the ones stored in a continuation.

Continuations are powerful concepts from the world of functional languages, like Scheme, but they are becoming popular in other languages as well.

1. Comments

add your comments