Science of Computer Programming 94 (2014) 3–17

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Process-aware continuation management in web applications

Matthias Book [a,*], Marco Buss [b,1], Volker Gruhn [a]

[a] paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Gerlingstr. 16, 45127 Essen, Germany
[b] OPITZ CONSULTING Deutschland GmbH, Tempelhofer Weg 64, 12437 Berlin, Germany

HIGHLIGHTS

- Continuations are used to cope with unforeseen user navigation in web applications.
- However, their management can incur significant memory overhead under heavy load.
- We present mechanisms for pruning continuation trees based on the dialog structure.
- This also prevents users from backtracking into completed transactions through the GUI.
- Depending on the structure of the dialogs, this enables significant memory savings.

ARTICLE INFO

Article history:
Received 18 October 2012
Received in revised form 14 June 2013
Accepted 30 July 2013
Available online 16 August 2013

Keywords:
Web engineering
Navigation
Continuations

ABSTRACT

Web applications are subject to an interaction challenge not found in other user interfaces: In addition to the widgets that web pages are built of, browsers provide further navigation features such as the Back and Forward buttons that are beyond the developer's control. Continuations have been suggested as a means to cope with the arbitrary navigation patterns that users may perform using these features. While an elegant solution in theory, continuations can incur a significant memory load in practice, and may offer more navigation options than business requirements mandate. We therefore propose a dialog control logic that augments the continuation approach with strategies for automatic elimination of continuations that will likely not be needed anymore, or whose invocation shall be prevented due to business requirements. This way, we aim to realize the benefits that continuations can provide to web applications, while ameliorating the drawbacks that they exhibit in practice.

1. Introduction

Web-based user interfaces have become highly popular front-ends for information systems that shall be available anywhere, anytime and on any device, since they allow the construction of ideal thin clients: Web browsers merely render hypertext pages according to markup generated by the server-side presentation logic. Due to the stateless nature of the Hypertext Transfer Protocol (HTTP) [1], the separation of server-side presentation logic and client-side rendering engines goes so deep that all client requests would seem like unrelated events to the application server, if it did not implement additional measures to establish the concept of user sessions spanning multiple request–response cycles.

The decoupling of hypertext markup generation and rendering would not be a problem if all possible ways of interacting with the rendered user interface could be specified at the time it is generated, i.e. if users could only interact with a web application through the widgets it provides on its pages. However, web browsers typically provide additional widgets

* Corresponding author.
E-mail addresses: matthias.book@paluno.uni-due.de (M. Book), marco.buss@opitz-consulting.com (M. Buss), volker.gruhn@paluno.uni-due.de (V. Gruhn).
[1] Work performed at the University of Leipzig, Germany.

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.07.015

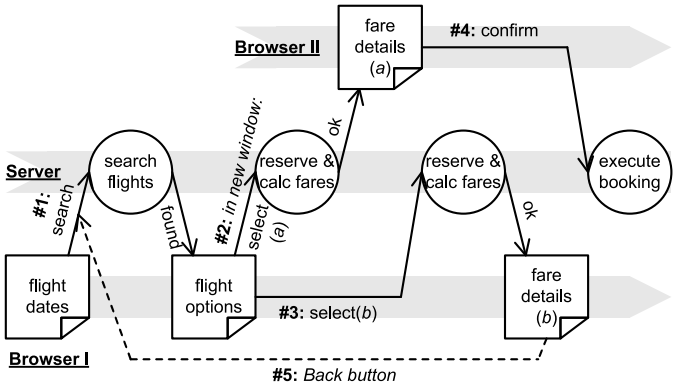

Fig. 1. Multiple window scenario.

outside the page that enable users to interact with the application on the level of their individual navigation sequence (i.e. the series of requests that users issue while navigating an application). These extra-page widgets include a Back button for re-issuing a previous request from the navigation history; a Forward button for re-issuing a later request from the navigation history (if the user had backtracked in the history before); and a Reload button for re-issuing the request for the currently displayed page. Most browsers have also adopted a feature for displaying the response to a particular request in a new browser window (or tab), enabling users to follow two parallel navigation "threads" from that point on. While these windows share the same session, cookies etc., they typically keep separate navigation histories, enabling users to move back and forth in branched navigation sequences.

1.1. Dialog synchrony breaks

Since the extra-page navigation features were originally devised to support browsing of static hypertext documents, they assume that a concept of state only exists on the client (in terms of the page it is currently displaying), while the server is a stateless entity that can produce a meaningful response to any request at any time (by simply delivering a document from its file system). In web applications, however, a stateful server is essential for realizing all but the most trivial business logic. In this case, client and server constitute two separate state automata: The server's automaton model is a fixed implementation of business requirements governing which transitions are permissible from any given state. The client's automaton model, in contrast, is continually adapted to reflect the user's navigation history, and allows (through the browser's extra-page navigation features) arbitrary transitions between all its states.

As long as the user triggers only state transitions that exist in the server's automaton model (by using only intra-page navigation features provided by the server), the client and server's state will remain in synchrony. However, if the user triggers a state transition that only exists in the client's model (by using one of the extra-page navigation features provided by the browser), the synchrony between the models can be broken. Re-establishing the synchrony of the client and server's dialog models can already be tricky, but much more serious is the undefined application behavior and corrupted data model that may result from such unexpected and uncontrollable synchrony breaks [2].

As an example (Fig. 1), a travel portal may offer a user two flight options *a* and *b* for a given date (#1). To compare both, the user may request the fare details for option *a* in a separate browser window (#2), and then request the fare details for option *b* in the original browser window (#3). If the user then decides on the earlier-selected option *a* (#4), the server may not realize on which page the confirmation was made, and erroneously book the later-selected option *b* (an actual bug in a popular travel portal [3]). Furthermore, if the user would click the Back button to return from the *fare details* page to the *flight options* page (#5), the browser would re-issue a *search* request that the server may not expect in its current state, potentially leading to undefined reactions.

Since extra-page navigation operations are not uncommon (clicking the Back button, for example, has been consistently found to be the most frequent mechanism used to revisit a previously displayed page [4,5]), they cannot be disregarded as exotic cases that may be dealt with through some crude solution (e.g. an error message) that protects state integrity, but ignores usability. Rather, we believe application developers should handle them as carefully as regular clicks on intra-page links or buttons.

One might argue that the bugs described above could easily be fixed by coding the application logic in a way that does not make assumptions about preceding pages, e.g. by adding an additional request parameter specifying the flight to be confirmed on the *fare details* page. While this is true, it would require additional logic in several layers of the application that is not motivated by business, but purely technical requirements: Instrumenting the page links with the flight ID, restoring the associated flight data from the back-end, combining it with flight-unspecific data present in the session, etc. The need to provide such individual logic in the application and presentation layer for every expected – and (due to backtracking etc.) also unexpected – page transition may significantly complicate the design and implementation of web applications, and provoke elusive errors.