Constraint-Based Invocation of Stateful Web Services: The Beep Store (Case Study)

Sylvain Hallé
Université du Québec à Chicoutimi

Roger Villemaire
Université du Québec à Montréal

***Abstract*—Shopping cart manipulations are a prime example of web services exhibiting stateful behaviour. The Beep Store is a web service synthesizing past work on the study and formalization of stateful interface constraints, and presenting many of the characteristics found in real-world web services, such as Amazon's and PayPal shopping carts.**

***Keywords*-temporal constraints; stateful web services; shopping cart; Amazon web services**

I. Introduction and Motivation

Asynchronous JavaScript and XML (Ajax) refers to a collection of technologies used to develop rich and interactive web applications. A typical Ajax client runs locally in the user's web browser and refreshes its interface using JavaScript according to user input. Popular Ajax applications communicate in the background with a remote server; in many cases, the server's functionality is made publicly available as an instance of a *web service*, which can be freely accessed by any third-party Ajax application.

While web services were originally expected to be stateless, web application backends are generally more complex in nature. To be properly understood by their respective recipients, each request and each response is expected to follow a specific structure, where the possible operations, parameters and values are precisely defined. In many cases, the browser-server exchange also moves forward according to a protocol, where the validity of a request depends on past events.

For such applications to work, the communication between the browser and the server must exactly follow an agreed-upon "contract" encompassing all these aspects. We present a demo application called the Beep Store, which synthesizes in a single environment many characteristics of real-world web application contracts that we studied in previous work.

II. The Beep Store Bundle

Our case study provides a stand-alone, client-server web application, implemented in PHP and JavaScript, that allows registered users to browse a fictional collection of books and music, and to manage a virtual shopping cart made of these elements.[1] It runs out-of-the-box in any modern web browser pointed at the store's URL, which itself can be installed into any standard Apache web server with default settings.

We acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

[1] http://beepbeep.sourceforge.net/examples/beepstore/bundle.zip

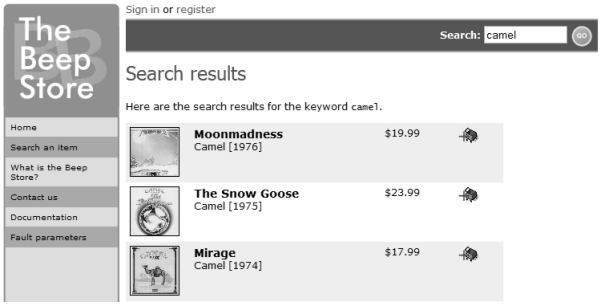

Figure 1. The Beep Store's web interface.

Figure 1 shows a typical application screen. At any time, users can use the search box at the top right of the screen to type any keyword. Similarly, they can click on the "Search an item" menu element at the left to summon a more complete search pane, where they can restrict the search to a specific artist, a specific title, and split the result into pages of a fixed number of entries.

A typical use case scenario for the Beep Store involves a user logging in, searching for items, adding and removing cart items, and eventually logging out. Such a scenario is a purposefully condensed version of popular commercial web sites, such as Amazon or eBay. Indeed, although the Beep Store is a demo application, all its functionalities —and constraints on its use, as we shall see— have been found in at least one of the real-world web services we studied in the past [1]–[4].

III. Constraints on Service Invocation

However, without any clear and mutual understanding of the acceptable requests and responses, an Ajax client might try to send a message that the server does not recognize, and vice versa. Indeed, typical of many web applications, interactions with the Beep Store are subject to numerous constraints involving data parameters, ordering of requests, and even combinations of both. The following constraints are actively enforced by the web service, which will reply with an error upon reception of any message that violates the contract.

1) Data Constraints: The first class of properties expresses constraints over the structure and values inside a single message at a time.

P1. Every message must carry a `SessionKey`.

P2. In any message, each `ItemID` must appear at most once.
P3. In the `ItemSearch` message, the element `Page` must be an integer between 1 and 20.
P4. In the `ItemSearch` message, the element `Page` is mandatory only if `Results` is present; otherwise it is forbidden.

One can see that further constraints of this kind could be added to that list, requiring or forbidding the presence of all possible elements in their respective messages. We stress that many of these constraints cannot be verified by a simple comparison with some XML Document Type Definition (DTD), as they also provide ranges for possible values, and even state that the presence of some element be dependent on the presence of another.

2) Control-Flow Constraints: Other restrictions are related to the sequence in which operations are invoked. Any application introducing the concept of session, or manipulating persistent objects such as a shopping cart, includes control-flow constraints of that kind. For example:

P5. The `Login` request cannot be resent if its response is successful.
P6. All cart operations, such as `CartCreate`, must follow a successful `LoginResponse`.
P7. The `Logout` request cannot be sent before a successful `LoginResponse`.
P8. All requests must be replied with their appropriate response operation (i.e. a `CartAdd` must be followed by a `CartAddResponse`, etc.).

These constraints introduce the notion of *state* into the application: the possible future messages allowed depend on what has happened in the past. Indeed, it does not make sense for a user to try to login again after a successful login. Similarly, since shopping carts must be associated to a logged user, it is impossible to create such a cart without first logging in. An attempt at such operations hints at some programming flaw on the client side, and should be replied by an error message from the server.

3) Data-Aware Constraints: Furthermore, the Beep Store includes properties referencing data elements inside exchanged messages, such that these data elements are taken at two different moments in the execution and need to be compared. Properties having this characteristic have been dubbed "data-aware" temporal properties [5]. For example:

P9. There can be at most one active cart ID per session key.
P10. All cart operations must provide a cart ID returned by some successful `CartCreateResponse`.
P11. An item to delete must first have been added in a previous `CartAdd` or `CartCreate` message.
P12. You cannot add the same item twice to the shopping cart.

It has been shown how these constraints can be formalized using a first-order extension of Linear Temporal Logic called LTL-FO$^+$ [6].

IV. Possible Uses of the Beep Store Bundle

The Beep Store is a simple and well-understood web application; however, since the client-server interaction is subject to numerous, real-world constraints, the code bundle can be put to numerous uses.

- Runtime monitoring: contract compliance can be checked at runtime; the Beep Store hence provides a suitable test environment for evaluating various design choices regarding code instrumentation, monitor location, etc.
- Model checking: the service's code is a single, standalone PHP file where each contract constraint is explicitly verified and each error emitted by the service is clearly marked. This makes it a fertile test environment for the automated model checking of behavioural constraints on service invocation.
- Trace validation: in addition to the web application, the bundle also includes a trace generator, which can produce any number of randomly-built, parameterizable request-response sequences. A set of such sequences can be used as a benchmark for offline trace validation algorithms of stateful properties.

References

[1] Amazon e-commerce service. http://solutions.amazonwebservices.com.

[2] Paypal web service API documentation. http://www.paypal.com.

[3] S. Hallé, T. Bultan, G. Hughes, M. Alkhalaf, and R. Villemaire. Runtime verification of web service interface contracts. *IEEE Computer*, 43(3):59–66, 2010.

[4] S. Hallé, G. Hughes, T. Bultan, and M. Alkhalaf. Generating interface grammars from WSDL for automated verification of web services. In L. Baresi, C.-H. Chi, and J. Suzuki, editors, *ICSOC-ServiceWave*, volume 5900 of *Lecture Notes in Computer Science*, pages 516–530, 2009.

[5] S. Hallé and R. Villemaire. Runtime monitoring of message-based workflows with data. In *EDOC*, pages 63–72. IEEE Computer Society, 2008.

[6] S. Hallé and R. Villemaire. Runtime verification for the web - a tutorial introduction to interface contracts in web applications. In H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. J. Pace, G. Rosu, O. Sokolsky, and N. Tillmann, editors, *RV*, volume 6418 of *Lecture Notes in Computer Science*, pages 106–121. Springer, 2010.