Formal Verification of Web Applications

Christian Ammann
University of Applied Sciences Osnabrück
Post Office Box 1940, 49009 *Osnabrück*
c.ammann@hs-osnabrueck.de

Abstract

Web applications are executed in a browser environment, widely used and can be developed with the Google Web Toolkit (GWT). This paper presents a domain-specific language (DSL) for modeling GWT web application and an automated transformation of the DSL into Promela which is the input language of the model checker Spin. Spin checks the complete state space of the corresponding web application model, informs the developer about possible errors and therefore increases software stability and quality.

1 Introduction

The *Google Web Toolkit* [1] is an open source framework for the development of web applications. Static elements are implemented with HTML while dynamic elements are developed with Java. The client side of an application is automatically transformed into JavaScript. The advantage of this approach is: Almost the complete web application can be developed in Java and therefore the existing tool support for Java like IDEs, debuggers, etc. is available. A GWT application can use *remote procedure calls* (RPCs) for the communication with a server (e.g. Tomcat). RPCs are asynchronous and therefore the GWT provides a special callback object.

Besides classical testing, a model checker can be used to verify whether a software system (e.g. a stock trading web application) meets its requirements. For a successful verification, the web application is transformed into the model checker's input language and enriched with formal requirements. Afterwards, the model checker verifies the complete state space of the model and typically generates an error path if a requirement is not met. This approach leads to the following problem: A GWT application is written in Java and has to be transformed into the model checker input language. Therefore, it is necessary after each modification of the GWT application to validate whether both models (Java and model checker input language) are still equivalent.

We solve this problem in section 2 and present a domain-specific language (DSL) for the description of GWT applications. Afterwards, section 3 describes a transformation algorithm for the DSL to Promela. Promela is the input language of the award winning [2] model checker Spin [3]. The related work is presented in section 4. Ideas about further work are described in section 5.

This work is part of the project *KoverJa* [4] which is supported by the *German Federal Ministry of Education and Research* (BMBF).

2 A DSL for Web Applications

This section introduces a simple GWT application and derives from it a DSL for the description of web applications. The example is a simple login mask which consists of a button and two text boxes. A user enters his username and a password. Both are transmitted to the webserver with a remote procedure call. The server verifies whether the username and password pair is correct and associates the result with the corresponding session. Afterwards, a new html page is opened which displays the username if the login was successful or otherwise an error message. The complete GWT Java project is available at [5].

The DSL for the implementation of the case study consists of the following parts: The client side is a set of (HTML) *pages*. A page contains GUI elements (e.g. a buttons). Furthermore, it can provide an *onLoad()* function which is executed when the page is loaded. A client has a unique *session* id which is bound to a session object. The session object contains a set of variables which can be accessed by the server. The *server* is a set of methods which are called as RPCs by the client. RPCs are asynchronous and therefore a reference to a *callback* object is added as an additional parameter to the parameter list. Each callback object contains an *onSuccess()* method which is executed when RPC was successful. The following listing contains the server side of the login web application and demonstrates the GWT DSL:

```
session{
  bool valid_user = false;
}

server{
  bool verifyLogin(string user,
  string password){
    if(user.equals("christian") &&
    password.equals("ammann")){
      valid_user=true;
    }
    else{
      valid_user=false;
    }
    //return session variable
    return valid_user
  }
}
```

Listing 1: Server Side DSL GWT Example

The session contains a boolean variable which is set to true after a successful login. The server provides the RPC *verifyLogin()* with two parameters: *username* and

password. The function evaluates whether both are correct (the only valid username/password pair in this example is *christian/ammann*) and stores the result in *valid_user*. The complete DSL example is available at [5].

3 Transformation to Promela

We presented in the last section a domain-specific language for the description of GWT web applications. This section provides a transformation algorithm from the DSL into Promela.

We use abstraction for the implementation of strings: Promela supports enumerations (so called *mtypes*). We generate an enumeration which elements represent the different strings of a model. The strings in listing 1 are transformed into the following Promela code:

```
mtype{christian, ammann, unknown}
```

Listing 2: Strings in Promela

Besides *christian* and *ammann*, a user could enter something else in a textbox. We model this class of input strings with the element *unknown*.

The next listing demonstrates the implementation of the *get()* method which is provided by each textbox object:

```
inline get(ret){
  if
  :: ret = christian;
  :: ret = ammann;
  :: ret = unknown;
  fi;
}
```

Listing 3: Get() Method in Promela

The if-block sets non-deterministically *christian*, *ammann* or *unknown* which has one major consequence: The model checker verifies the system behavior for all possible return values of the function.

For a successful transformation of the client side, the question regarding the callback object is: When is the *onSuccess()* method executed? Empirical measurements have shown to the following behavior (implemented with pseudo code):

```
button_a behaviour:
  rpc1(callback1);
  execute statements;
  rpc2(callback2);
  execute more statements;
  onSuccess() of callback1;
  onSuccess() of callback2;
```

Listing 4: Runtime of Callback objects

The *behaviour* block of button *a* consists of some regular statements (e.g. variable assignments) and two RPCs. The onSuccess() methods are executed deterministically at the end of the *behaviour* block. The complete transformation example of the login web application (especially the server side) is available at [5].

4 Related Work

One key feature of web applications is the pending amount of clients which are connected to a server. Bauer et al. [6] describe dynamic communication systems (DCS) and use as an example a communication protocol for cars.

Another interesting approach by Leung et al. [7] describes a modeling and verification approach for web pages. It proposes to model web pages with statecharts.

5 Further Work

Further work will be the development of the missing translation algorithm into a GWT Java project and the corresponding transformation templates. The generated Promela code reflects the behavior of the *Google Chrome* browser. Support for other browsers like *Firefox* and *Internet Explorer* is still missing and has to be implemented.

6 Acknowledgement

I would like to express my gratitude to Stephan Kleuker and Elke Pulvermüller for supporting this work.

References

[1] Google Web Toolkit. `http://code.google.com/webtoolkit`; accessed 18-August-2011.

[2] System Software Award. `http://awards.acm.org/homepage.cfm?awd=149`; accessed 18-August-2011.

[3] Gerard Holzmann. *The Spin Model Checker: Primer and reference Manual*. Addison-Wesley Professional, 2003.

[4] KoverJa Projekt - Korrekte verteilte Java Applikationen. `http://www.edvsz.hs-osnabrueck.de/kleuker/CSI/KoverJa`; accessed 14-September-2010; in german.

[5] GWT Login Case Study, Xtext Grammar and DSL Example. `http://home.edvsz.fh-osnabrueck.de/chammann/gwt`; accessed 18-August-2011.

[6] Jörg Bauer, Ina Schaefer, Tobe Toben, and Bernd Westphal. Specification and verification of dynamic communication systems. In *ACSD '06: Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, pages 189–200, Washington, DC, USA, 2006. IEEE Computer Society.

[7] Karl R. P. H. Leung, Lucas Chi Kwong Hui, S. M. Yiu, and Ricky W. M. Tang. Modeling web navigation by statechart. In *24th International Computer Software and Applications Conference*, COMPSAC '00, pages 41–47, Washington, DC, USA, 2000. IEEE Computer Society.