A System for Specification and Verification of Interactive, Data-driven Web Applications

Alin Deutsch*, Liying Sui, Victor Vianu†, Dayou Zhou‡
Department of Computer Science and Engineering
University of California, San Diego
{deutsch, lsui, vianu, dzhou}@cs.ucsd.edu

ABSTRACT

In recent research, we have proposed a framework for high-level specification of interactive, data-driven Web applications and established theoretical foundations for their verification [4], as well as implemented a verifier called WAVE [3]. We propose to demonstrate a system which centers on WAVE and consists of various modules dealing with aspects ranging from specification of Web applications to explanation of verification results. Our demonstration will focus on features of the specification language and the verification input, output and performance.

1. INTRODUCTION

Web applications interacting with users or programs while accessing an underlying database are increasingly common. They include e-commerce sites, scientific and other domain-specific portals, e-government, and data-driven Web services. The spread of such applications has been accompanied by the emergence of tools for their high-level specification. A representative, commercially successful example is WebML [2, 1], which allows to specify a Web application using an interactive variant of the E-R model augmented with a workflow formalism. High-level specification of Web applications also provides new opportunities for automatic verification. Such verification leads to increased confidence in the correctness of database-driven Web applications. We next describe a framework for high-level specification and verification of interactive, data-driven Web applications which we have recently proposed [4].

Web application specification We focus on interactive Web applications generating Web pages dynamically by queries on an underlying database. The Web application accepts input from external users or programs. It responds by taking some action, updating its internal state database, and moving to a new Web page determined by yet another query. All inputs, actions, states and the database are modeled as relations. For example, below is the specification of the "laptop search page" of a computer shopping Web application which will be re-visited in Section 3.

Notation For better readability of our examples, we use the following notation: relation R is displayed as R if it is a state relation, as R if it is an input relation, as $\underline{\mathsf{R}}$ if it is a database relation, and as $\overline{\mathsf{R}}$ if it is an action relation.

`Page` LSP
`Inputs:` $\mathsf{laptopsearch}(ram, hdisk, display), \mathsf{button}(x)$
`Input Rules:`
$\text{Options}_{\mathsf{button}}(x) \leftarrow \quad x = \text{“}search\text{”} \vee x = \text{“}view\ cart\text{”} \vee x = \text{“}logout\text{”}$
$\text{Options}_{\mathsf{laptopsearch}}(r, h, d) \leftarrow \underline{\mathsf{criteria}}(\text{“}laptop\text{”}, \text{“}ram\text{”}, r) \wedge \underline{\mathsf{criteria}}(\text{“}laptop\text{”}, \text{“}hdd\text{”}, h) \wedge \underline{\mathsf{criteria}}(\text{“}laptop\text{”}, \text{“}display\text{”}, d)$
`State Rules:`
$\mathsf{userchoice}(\mathsf{r},\mathsf{h},\mathsf{d}) \leftarrow \mathsf{laptopsearch}(r, h, d) \wedge \mathsf{button}(\text{“}search\text{”})$
`Target Web Page Rules:`
$\text{HP} \leftarrow \mathsf{button}(\text{“}logout\text{”})$
$\text{PIP} \leftarrow \exists r \exists h \exists d\ \mathsf{laptopsearch}(r, h, d) \wedge \mathsf{button}(\text{“}search\text{”})$
$\text{CC} \leftarrow \mathsf{button}(\text{“}view\ cart\text{”})$
`End Page` LSP

Notice how the three buttons *search*, *view-cart*, and *logout* are modeled by a single input relation button, whose argument specifies the clicked button. The corresponding input rule restricts it to a search, view-cart, or logout button only. Since the user chooses at most one tuple among the displayed options, no two buttons may be clicked simultaneously.

Observe how the second input rule looks up in the database the valid parameter values for the search criteria pertinent to laptops. This enables users to pick from a menu of legal values instead of providing arbitrary ones.

If the search button is clicked, the state rule records the user's pick of search criteria in the userchoice table. If this pick is non-empty, the second target rule fires and the Web application transitions to the PIP page.

Property specification The properties we wish to verify pertain to the behavior of the Web application which we capture using the notion of *run*. A *run* is a sequence of configurations through which the Web application evolves in response to user inputs. Properties range from basic soundness of the specification (e.g. the next Web page to be displayed is always uniquely defined) to semantic properties (e.g. no order is shipped before a payment in the right

*Supported in part by NSF/IIS-0415257 and NSF/IIS-0347968(CAREER).
†Supported in part by NSF/IIS-0415257.
‡Supported in part by NSF/IIS-0427196.

SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA.

amount is received). Such properties are expressed using LTL-FO, an extension of *linear-time temporal logic* (LTL). For example, the LTL-FO formula

$$\forall x \forall y \forall id[(\mathsf{pay}(id, x, y) \wedge \underline{\mathsf{price}}(x, y)) \ \mathbf{B} \ \overline{\mathsf{ship}}(id, x)]$$

states that whenever item x is shipped to customer id, a payment for x in the correct amount must have been previously received from customer id.

Verification problem The verification problem we are interested in is whether a Web application specification satisfies a property formula, i.e. whether the property is true for all runs of the Web application.

Theoretical results established in [4] show that the verification problem is decidable in PSPACE given that both the Web application and the property satisfy reasonable restrictions called *input-boundedness* which essentially requires the quantified variables in Web page rules to range over values provided in the page's input. The demonstration shows the expressive power of input-bounded specifications by presenting the input-bounded modeling of significant parts of four widely known Web applications, namely a Dell-like computer shopping site, a GrandPrix sports news site, an Expedia-like travel reservation site, and a Barnes& Noble-like book shopping site. The theoretical results have led to an implementation of the WAVE verifier described in [3], in which we have also shown that, through careful engineering, the PSPACE worst-case bound can be prevented from materializing and that efficient verification of the four Web applications is feasible within seconds, using the WAVE verifier. WAVE takes as inputs the specifications of a Web application and a property, outputs whether the property is true or false for that Web application, and, in the latter case, displays a counter-example which is a run of the Web application violating the property. The successful implementation of WAVE is due to a novel combination of classical model checking approaches and database optimization techniques. For input-bounded specifications and properties, WAVE is a *sound* and *complete* verifier in the sense that it only finds valid counter-examples and is guaranteed to find a counter-example if one exists. Moreover, WAVE can still be used as a sound (incomplete) verifier if the restrictions have to be relaxed in exchange for additional specification power. This means that WAVE might raise false alarms, but if it declares the property to hold, then this is indeed the case.

We propose to demonstrate a system for specification and verification of interactive, data-driven Web applications which contains:

- A specification module, which inputs a text file conforming to the grammar of the specification language and processes it. The text file can either be manually produced or be imported from a WebML specification via the WebML import sub-module;
- A verification module, which is essentially the core of WAVE. It inputs the specification and a property, from which it computes the information needed for simulating runs of the Web application. A true or false result is produced at the end of the simulation along with useful verification information;
- An explanation module, which helps the user better understand the output from the verification module by reproducing counter-examples and/or Web application configurations in a more informative way; and

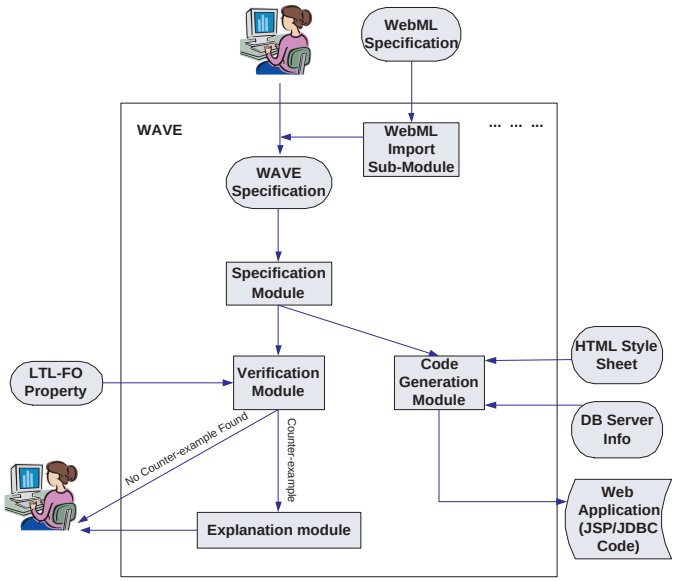

Figure 1: System architecture

- A code generation module, which is orthogonal to the verification module and can be regarded as an interpreter for the specification language. It reads in the specification file and generates JSP pages which correspond to the specified Web application.

Outline Section 2 presents the architecture of our system with detailed descriptions for individual modules. Section 3 outlines our plan for the demonstration. We conclude in Section 4.

2. SYSTEM ARCHITECTURE

The architecture of our system is illustrated in Figure 1.

The system covers many aspects ranging from easy-to-understand specification of Web applications to informative explanation of verification results, which requires no background formal verification. We next describe each module in some detail.

Specification module The specification of a Web application is a text file consisting of two sections. The first section is known as the global declaration, which contains signatures of all relations, categorized into database, state, input and action relations, as well as identifiers for all Web pages. The second section contains the schema for each Web page, an example of which is shown in Section 1. A page schema specifies what user inputs are accepted at that page, as well as the logic implemented by the page using a collection of rules governing how input options are generated for the user (as required by menus, buttons, links etc.). Once an input is provided by the user, the rules specify how (a) the state is updated, (b) actions are performed (such as the sending of a confirmation e-mail) and (c) a transition to the next page is determined. Once the file conforms to the grammar, the parser generates an internal representation of the specification, to be consumed by the verification module and the code generation module.

It is also possible to create the specification file using a graphical interface. Indeed, we are currently collaborating with the WebML group to implement a *WebML import sub-*

module which reads in WebML specifications (in XML) and translates them into our specifications. Consequently, the completion of this sub-module will enable our system to be applicable for any Web application specifiable by WebML, which is significant. The dots to the right of the WebML import sub-module indicate that similar sub-modules can be added for other specification languages as well.

Verification module In a nutshell, we perform verification in search for a counter-example run, i.e. a run violating the property. Since the verification is done at design time, we are required to examine all possible databases. Therefore the major challenge of our verification problem is the fact that we are dealing with infinitely many databases and runs. In [4] we have shown that it suffices to consider a fixed set of constants for possible values in the database, yielding a finite number of databases. WAVE enumerates databases and then the runs based on each database. For each run, the property is checked and violation detected. When the algorithm terminates, a true or false result is produced along with useful verification information, such as a counter-example run in the case of a false property. Since WAVE executes simple queries frequently over small-size tables, a Java-based in-memory database HSQLDB is used instead of a sophisticated DBMS such as Oracle, with significant speedup observed.

Explanation module The explanation module produces runs which violate the property, if any. The runs show the precise sequence of user inputs, the contents of the database, and the sequence of configurations through which the Web application evolves, as well as the exact point where the property is violated. The developer can fast-forward and fast-backward through the run, inspecting the contents of the state and the database at will.

Code generation module Once a specification passes the verification stage, it is used to automatically generate the code implementing the Web application. The code generation module reads in the (successfully parsed) specification file and generates JSP pages which correspond to the Web application specified. Assuming the availability of a configured Tomcat server, the set of generated JSP pages are ready to be viewed in a browser. The code implementing the connection to the underlying database is generated in JDBC. We are currently enhancing this module to allow the Web developer to supply HTML style sheet information to specify the layouts on the generated pages.

3. DEMONSTRATION PLAN

The demonstration will focus on the specification module and the verification module. For the specification module, we will demonstrate a Dell-like computer shopping Web application (available at [5]) which is modeled using our specification language. The demo window shown to the audience is divided into three frames. The upper-left frame contains the current page of the Web application itself. The lower-left frame shows the page specification for the particular Web page above it. Finally the right frame displays the contents of all state relations for the current Web page on the left, as well as the static database relations and global schema declaration (via the buttons "Show database" and "Show global declaration", respectively).

We will demonstrate how various HTML components, internal state updates and page transitions are modeled in the specification language. We will perform actual user interactions in the Web application such as filling in text fields, picking from drop-down menus and clicking buttons, and will demonstrate how contents in each frame changes in response to the user interactions. Finally we will use examples to informally describe the input-boundedness restrictions which are satisfied by this Web application specification.

For the verification module, we will demonstrate how temporal properties of the computer shopping Web application described above are specified and verified using the Web interface for WAVE (available at [5]). When the Web application to be verified is selected (via the radio button), the drop-down menu below is populated with a list of prepared properties described in natural language. When one of the properties is selected, its formal specification as a LTL-FO formula is displayed in the text area below.

We will use a few properties to (informally) demonstrate to the audience the ingredients of LTL-FO, after which we will select a few of them as input to WAVE (together with the Web application specification) for verification (via clicking the button below). The output from WAVE will be displayed.

We will explain the results and statistics displayed. In case of a false property, we will demonstrate the counter-example run displayed, walking the audience through the individual steps of the property-violating interaction. We will refer back to some property specifications to informally describe the input-boundedness restrictions. The demo audience will be encouraged to edit the properties and initiate a verification run. Finally, we will explain the output of the code generation module, pinpointing the correlation between high-level rules and the generated JSP and JDBC code.

4. CONCLUSIONS

In this paper we have proposed to demonstrate a system for specification and verification of interactive, data-driven Web applications. We have outlined a demonstration plan which focuses on the specification mechanism and the verifier WAVE. The demonstration requires no background in formal verification or in Web application development. The preliminary version of our demonstration can currently be accessed online at [5].

5. REFERENCES

[1] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, and I. Manolescu. Specification and design of workflow-driven hypertexts. *J. Web Eng.*, 1(2):163–182, 2003.

[2] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, 2002.

[3] A. Deutsch, M. Marcus, L. Sui, V. Vianu, and D. Zhou. A verifier for interactive, data-driven web applications. In *SIGMOD Conference*, 2005.

[4] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web services. In *PODS*, pages 71–82, 2004.

[5] WAVE. Project website. http://www.db.ucsd.edu/wave/.