

Developing and Verifying RESOLVE Components via a Web Interface

Chuck Cook and Hampton Smith

Abstract—Web 2.0 technologies allow users to interact with sophisticated web applications on a universal platform: the Internet browser. By harnessing this technology, RESOLVE can reach a wider audience who, in the face of a rising flood of information, increasingly make their decisions in seconds rather than hours. By simplifying access to both the source code and advanced RESOLVE features such as its integrated prover, we can improve development by rapidly exploring the effects of key changes—such as the addition or removal of a requires clause, or the modification of a parameter passing mode. In addition, such a deployment is ideal for educational applications, allowing for universal access to RESOLVE in an environment familiar to students.

Index Terms—Verification, Software Engineering, Development Tools

I. INTRODUCTION

Wide availability of mature RESOLVE tools is crucial for both broader understanding of the RESOLVE community’s goals and acceptance of a fully functional verifier as a valid part of the programmer’s toolkit. However, most of the tools currently available remain confined to the command line, rather than the Integrated Development Environments (IDEs) that many modern programmers utilize. To date, several impressive tools have been developed to make use of graphical user interfaces. Web Demos developed at Clemson and The Ohio State University use web technologies to provide online access to demonstrations of end-to-end verification with code generation. They do not, however, allow users to enter and experiment with their own code. The Eclipse IDE plugin allows IDE-savvy users to create their own code and syntax check and translate it, but does not take advantage of any other RESOLVE features. While these tools take advantage of the strengths of their respective platforms, they are not enough to bring RESOLVE into the mainstream. This paper introduces a new web-based application designed to address this issue. The application ultimately attempts to combine the strengths of the previously mentioned tools by using modern web technologies to provide global access to

and take full advantage of RESOLVE. The paper is organized as follows: Section II presents a detailed description of the current tools. Section III discusses the technology used by the web interface. Section IV provides a walkthrough with screen shots of the new system, Section V discusses educational uses, and Section VI presents future plans and conclusions.

II. PREVIOUS TOOLS

A. Previous Web Demo [1, 2]

The original web demo designed for the Clemson RESOLVE Verifying Compiler demonstrated the full range of the compiler. In addition to making a large library of components available as examples, it permitted the user to walk through the steps for generating VCs, verifying them with either Isabelle or RESOLVE’s integrated prover, and generating Java code.

However, the demo only allowed a static collection of hard-coded source code examples and provided no samples that contained syntax errors or code that was incorrect and thus unverifiable. In addition, despite being web-based, certain aspects of the demo (specifically, Isabelle verification), could not handle concurrent requests, and so the demo could not be released “into the wild.”

B. Eclipse Integration

Another tool developed for the Clemson RESOLVE Verifying Compiler was a RESOLVE plugin for the Eclipse IDE. With the plugin, users could create arbitrary RESOLVE files, then select a file from the explorer, and compile it with a click.

However, the plugin required that the compiler options be set in the preferences menu. Thus, taking advantage of the diverse features of the compiler largely suffered from the same complications as invoking the compiler from the command line. In addition, installing the plugin was certainly less straightforward than using the demo website from the previous section.

III. WEB INTERFACE DESIGN

The system is made up of two primary components, a web interface and a server backend to communicate with the RESOLVE Compiler. The web-based frontend presents the user with an easy to use and straightforward platform for

Concept	Enhancement	Enhancement Realization	Concept Realization
Unbounded_Stack ▲	UST_Transfer_Capability ▲ UST_Flip_Capability ▲	▲	▲

Input text here

Generate Code

Generate VCs

Verify with Isabelle

Verify with RESOLVE

Input

Results

Clear Textbox

view

Fig 1. Web-based user interface.

developing RESOLVE components. The backend has two major tasks: serving as a communication intermediary between the user and the RESOLVE Compiler and providing content for the frontend.

The web technologies HTML and AJAX (asynchronous JavaScript and XML) make up the backbone of the system frontend. These technologies provide the tools needed to create the right balance of static and dynamic content that is presented to the user (see fig. 1). The basic format of the web page is hard-coded in HTML; AJAX is used to dynamically load information on RESOLVE modules and to remotely invoke the compiler on the server side. In this case, AJAX is implemented using JavaScript to send and receive XML messages over HTTP to and from the server backend.

The backend server uses Java Servlets to provide the necessary functionality. Java Servlets were chosen for two important reasons: 1) the fact that each servlet instance is created in a new thread and 2) their compatibility with the existing RESOLVE Compiler, which is also written in Java. Instantiating each servlet in a new thread allows the system to accommodate multiple simultaneous users, a feature missing from the previous demo. Because both the servlet and the RESOLVE Compiler are coded in Java the servlet can instantiate and communicate directly with the compiler, reducing the security threat often associated with launching external programs from within web applications.

These two core technologies allow the user to select an

available enhancement from a menu and directly input a custom realization, while the system provides the boilerplate code required by the compiler. For example, when a user selects a concept and an enhancement, this system will autonomously generate an appropriate header and footer based on the concept and enhancement selected. This is discussed further in section 4. This technique reduces complexity for the user while ensuring that conventions enforced by the compiler (such as each realization being located in a file of the same name) are followed, thus increasing the flexibility of the server-side implementation. Because the headers and footers are controlled by the system, file names can be randomly generated and multiple realizations can be compiled concurrently without the threat of name collision. An additional advantage AJAX provides is the ability to poll the server, allowing long-running processes, such as verifications performed by Isabelle or the RESOLVE integrated prover, to run to completion without fear of HTTP timeouts breaking the connection before the process has ended.

IV. EXAMPLE

This section presents an example illustrating the capabilities of the new RESOLVE web interface. We begin by selecting a concept from the Concept box at the top of the page. This action will query the server and populate the Enhancement box with any enhancements available for the selected concept. Next, we select the desired enhancement from the Enhancement box—here we've selected to realize a flipping capability. At this point, the system generates an un-editable

Concept	Enhancement	Enhancement Realization	Concept Realization
Unbounded_Stack	UST_Transfer_Capability UST_Flip_Capability		

Realization Demo_Realization for UST_Flip_Capability of Unbounded_Stack;

```

uses Std_Boolean_Fac;

Procedure Flip(updates S: Unbounded_Stack);
Var S_Reversed: Unbounded_Stack;
Var Next_Entry: Entry;

While (not Is_Empty(S))
  changing S, S_Reversed, Next_Entry;
  maintaining #S = Reverse(S_Reversed) o S;
do
  Pop(Next_Entry, S);
  Push(Next_Entry, S_Reversed);
end;
S_Reversed := S;
end Flip;

```

end Demo_Realization;

Buttons: Generate Code, Generate VCs, Verify with Isabelle, Verify with RESOLVE, Input (selected), Results, Clear Textbox, view

Fig. 2. Concept and enhancement selected and RESOLVE code entered into the textbox.

header and footer for the code based on the selected concept and enhancement, printing them just above and below the input textbox. First we enter a valid realization to demonstrate the behavior of the system (see Fig. 2). Once satisfied that the code has been entered properly, we click the “Verify with RESOLVE” button. The selected concept and enhancement information, as well as the manually entered code from the textbox, is transferred to the server backend and the RESOLVE Compiler is invoked. After the compiler completes the job, the results are returned via AJAX and displayed in the textbox, as seen in Fig. 3. The user sees the full output from the compiler with their generic input, without needing to have the compiler installed locally.

```

Realization Demo_Realization for UST_Flip_Capability of Unbounded_Stack;
#####Start Compiler#####
7/24/09 1:42 PM
RESOLVE Compiler/Verifier Version of March 2009.
Options: -translate to compile;
         -VCs [-isabelle] [-verbose] to generate Verification Conditions (optionally for
         isabelle);
         -typecheck to check mathematical typing; -proofcheck to check mathematical
         proofs;
         -sanitycheck to perform various common sense checks; -prove to attempt to prove
         VCs.
0_1 Proved in 222 milliseconds. Overall, 7129 proofs were directly considered
and 361 useful backtracks were performed.
0_2 Proved in 906 milliseconds. Overall, 20545 proofs were directly considered
and 768 useful backtracks were performed.
1_1 Proved in 39 milliseconds. Overall, 1111 proofs were directly considered and
55 useful backtracks were performed.
#####End Compiler#####

```

Fig. 3. RESOLVE Compiler results on screen.

The system is also capable of displaying the results of code that contains bugs or syntax errors. For instance, the example code could contain a bug such that the compiler will not be able to prove all the VC’s. Because the system stores both the

entered source code and compiler results locally, the user can easily flip between them. We begin by selecting the Input radio button and clicking “view.” This will reload the original source code into the input box, allowing us to make changes. To simulate a bug in the code, we comment out the Push operation of the do while loop and again click “Verify with RESOLVE.” As seen in Fig 4, the compiler reports that it was unable to prove and thus skipped VC 0_2.

The system returns a similar message when a syntax bug is introduced into the code. Like before, we select Input and then click “view” to modify the input code. However, this time, after we uncomment the Push operation, we insert a forward slash character before the do while loop end tag to simulate a typographical error. We see in Fig. 5 that the system has identified the error.

```

Realization Demo_Realization for UST_Flip_Capability of Unbounded_Stack;
#####Start Compiler#####
7/24/09 1:46 PM
RESOLVE Compiler/Verifier Version of March 2009.
Options: -translate to compile;
         -VCs [-isabelle] [-verbose] to generate Verification Conditions (optionally for
         isabelle);
         -typecheck to check mathematical typing; -proofcheck to check mathematical
         proofs;
         -sanitycheck to perform various common sense checks; -prove to attempt to prove
         VCs.
0_1 Proved in 223 milliseconds. Overall, 7129 proofs were directly considered
and 361 useful backtracks were performed.
0_2 Skipped after 23892 milliseconds. Overall, 568267 proofs were directly
considered and 21766 useful backtracks were performed.
1_1 Proved in 41 milliseconds. Overall, 1111 proofs were directly considered and
55 useful backtracks were performed.
#####End Compiler#####

```

Fig. 4. Unable to prove all the VC’s.

```

Realization Demo_Realization for UST_Flip_Capability of Unbounded_Stack;
#####Start Compiler#####
7/27/09 2:03 PM
RESOLVE Compiler/Verifier Version of March 2009.
Options: -translate to compile;
-Vcs [-isabelle] [-verbose] to generate Verification Conditions (optionally for
isabelle);
-typecheck to check mathematical typing; -proofcheck to check mathematical
proofs;
-sanitycheck to perform various common sense checks; -prove to attempt to prove
VCs.

Error: aa891e841.rb(15): ne 15:3: expecting "end", found '/'
      ^

Add unparsable: aa891e841.rb
#####End Compiler#####
end Demo_Realization;

```

Fig. 5. Syntax error detected.

V. EDUCATIONAL APPLICATIONS

The RESOLVE Compiler has been and is used as a tool in software development classes. Because the compiler is a stand-alone application, students cannot install it on lab machines. The only recourse available is to install it on their personal computers. However, valuable time is lost while trying to get it working properly across many different operating systems and computer platforms. The new interface will allow students to begin learning more quickly, while being provided with immediate feedback.

In addition, the use of a web interface provides unique opportunities to present features that go beyond what we expect the compiler to provide and allow RESOLVE to be used as an application. For example, after writing some code, students could be asked to comment on the state of certain variables at different key points. The RESOLVE integrated prover could then be used to check their work.

VI. FUTURE WORK AND CONCLUSIONS

This new interface is still in an early state of development. Currently it is only able to accept new RESOLVE enhancement realizations for existing specifications and verify them using the RESOLVE integrated prover. It is not yet able to avail itself of the Isabelle theorem prover. Ideally, users will eventually be able to compile and generate code for any RESOLVE component. Verification is currently limited by the compiler to enhancement realizations. In the future we would like web users to be able create new concept and enhancement specifications that the compiler can utilize on a per-session basis for proving customized realizations. These new specifications, however, would be stored temporarily and only be available until the user closes the web browser window. In addition, a number of user interface and security improvements must be made before the system can be made generally available.

We are already pleased with the capabilities of this interface, and the success it has had in blending the strengths

of the full-featured but static older web demo and the dynamic but narrow-featured Eclipse plugin. We are hopeful that when completed, it will represent an excellent, instant source for delivering the RESOLVE experience to students and researchers alike.

REFERENCES

- [1] Clemson Resolve Web Demo. resolve.cs.clemson.edu. Accessed July 2009.
- [2] Sitaraman M., Adcock B., Avigad J., Bronish D., Bucci P., Frazier D., Friedman H.M., Harton H., Heym W., Kirschenbaum J., Krone J., Smith H., Weide B.W. Building a Push-Button RESOLVE Verifier: Progress and Challenges. Technical Report RSRG-09-01, School of Computing, Clemson, SC (2009).