World With Web

A compiler from world applications to JavaScript

R. Emre Başar, Caner Derici, Çağdaş Şenol

İstanbul Bilgi University, Department of Computer Science

{reb,cderici,csenol}@cs.bilgi.edu.tr

Abstract

Our methods for interacting with computers have changed drastically over the last 10 years. As web based technologies improve, online applications are starting to replace their offline counterparts. In the world of online interaction, our educational tools also need to be adapted for this environment. This paper presents WorldWithWeb, a compiler and run-time libraries for mapping programs written in Beginning Student Language of PLT Scheme with World teachpack to JavaScript. This tool is intended to exploit the sharing-enabled nature of the web to support the learning process of students. Although it is designed as an extension to DrScheme, it is also possible to use it in various settings to enable different methods of user interaction and collaboration.

Keywords JavaScript, web, compiler, Scheme

1. Introduction

The role of computers and World Wide Web (WWW) in our life has gone through a series of changes through the last decade. Computers used to be just a static, desktop only tool. The design of WWW also reflected that nature by being a static source of information. Although the WWW is designed to be a source for sharing information [2], it constituted a producer-consumer relationship between the author and the visitor of a web site. In that scenario users had a passive role of accessing the web, using the web browser on their computers.

Mobile networks, wireless access and availability of powerful mobile devices changed the way we interact with computers. This change in the environment also triggered a change in our approach to WWW. By the rapid growth of Web 2.0 and technologies related to it, users dropped their role as passive consumers of information, and became collaborators and creators of the content [11]. This change of role is due to the fact that Web 2.0 enabled users to create and share their content easily, without going through all the hassle of creating and maintaining a personal web site.

While the world has changed, our teaching and development tools mostly stayed the same. Although our tools for teaching programming is much better than the ones we had 10 years ago, they have not adapted to the change. One of the main drawbacks of our current tools is that they are "offline". We believe that in the age of blogs, social networks and other types of sharing media, writing a program that guesses the number you had in mind is useless unless you share it with other people.

To address this need, we developed a compiler, WorldWithWeb[1], that enables users to share their creations easily. Using WorldWithWeb it is possible to create an interactive animation using the "Beginning Student" language of DrScheme with the world.ss teachpack and publish it as a standalone application that runs in the web browser. Being able to produce a self contained, browser-based application makes it possible for the user to share it in all types of online media.

2. WorldWithWeb

The aim of WorldWithWeb is to create a bridge between DrScheme programming environment and the web. This way, users will be able to share their applications easily, enabling them to be a part of the connected world, instead of just a consumer. Using this approach it is much more easier for users to share their creations and get feedback from different people around the world (not just their friends, teachers or families) and more importantly gain online reputation.

To accomplish this goal, we need to enable the user to speak the "lingua franca" of the web: JavaScript [1]. Although JavaScript is a language with a fine set of features and tons of libraries for creating online applications, it is not a perfect fit for pedagogic purposes. A pedagogic language, as defined by Felleisen et al [5] needs to be simple and light as far as possible. DrScheme's "Beginning Student" language (BSL) and other languages at the HtDP category are designed with this purpose in mind, and they are a perfect fit for pedagogic methodology behind HtDP.

For this purpose we created a compiler that compiles programs written in the BSL to JavaScript. This way, a student can create a program in the DrScheme environment, using a language that is designed to help his/her learning process. Then he/she can automatically create a web page containing the application, sharing it over the web with the rest of the world.

2.1 A note on Beginning Student Language and World.ss

DrScheme development environment is able to restrict, or extend a user's access to the underlying language. This feature is called "Languages". One of these languages, is the "Beginning Student" language, which is used by HtDP [4]. This language is trimmed down to restrict the user to a nearly-pure functional subset of Scheme without higher-order functions. Although the BSL provides only some basic utilities for writing programs, it is possible to extend the language with libraries called teachpacks [15].

[1] For source code, screenshots, demos and other information please visit: http://vc.cs.bilgi.edu.tr/trac/worldwithweb

Proceedings of the 2009 Scheme and Functional Programming Workshop
California Polytechnic State University Technical Report CPSLO-CSC-09-03

World.ss [3] is a teachpack designed for creating interactive animations in a purely functional programming style. The imperative parts of creating an animation is handled by internals of world.ss package. This way, the student is left with a purely functional programming interface where he/she can design and code the animation focusing on the design methodology as imposed by HtDP.

3. Related Work

The Moby Scheme Compiler [8] is an effort to make programs written in BSL and world.ss available on mobile devices. It also contains some extensions to world.ss package. Using those extensions, it is possible for the user to create applications that exploits the extra functionality (like GPS or tilting detection) provided by those devices, while staying within the BSL. While Moby opens up new possibilities in front of students for creating applications that runs on devices other than classical PC's, it does not solve the problem of sharing those applications with each other.

scheme2js [9] is a compiler from Scheme to JavaScript, intended to provide complete interoperability between JavaScript and Scheme. While it has similar goals with WorldWithWeb, it should be considered as a foreign function interface between Scheme and JavaScript. Although it is possible to use scheme2js as a tool for implementing low level interfaces of WorldWithWeb, the complex relations between PLT Scheme GUI libraries and underlying OS makes it impossible to use that kind of low-level implementation directly.

Processing.js[2] is an implementation of the Processing [14] for JavaScript. Although it presents the same API to the users, it does not provide any tools for converting from Java to JavaScript automatically. To be able to run a Processing application with Processing.js the user needs to re-write the entire application using JavaScript.

O'browser, which is developed within the Ocsigen project[3] is a virtual machine for OCaml, written in JavaScript. It supports loading OCaml bytecode directly into the VM without any need for recompiling. Although it provides the advantages of a statically typed functional programming language, users need to have previous knowledge about HTML and DOM to create applications for the web. This disadvantage creates a barrier for the beginning student to create applications using that framework.

4. Implementation

The design of WorldWithWeb is based on two distinct parts. One part is the core compiler, which translates the Scheme code to JavaScript. The second part is the runtime libraries, implemented in pure JavaScript. This strict separation allows us to freely experiment on the implementation of runtime, while keeping the compiler as small as possible.

4.1 Compiler

Since the web provides the user with many different methods (i.e. blogs, forums, social networking sites) for publishing the content, our implementation also needs to be adaptable to various media. Therefore, WorldWithWeb is designed as a library that can be used by various frontends to create final output. This way, the user can create new frontends that can read code from any kind of resource and to create output that is appropriate for the destination medium. Currently the only output format is HTML, linked with required JavaScript libraries and user code.

The output of the compiler is a `pinfo` structure which is defined as:

[2] http://www.processingjs.org

[3] http://www.ocsigen.org

```
(define-struct pinfo (code
                      function-mappings
                      tests
                      images))
```

Code field of the structure is the generated JavaScript code for the program. The source code is contained as a string, so it can be used directly with `display` or similar functions to create the main JavaScript file.

Function mappings are provided as a bridge between BSL functions and the JavaScript libraries that implements them. They simply rename the functions available from the libraries to their Scheme counterparts. Since the inclusion of this map might be accomplished using different methods in different environments, it's left to the backend to decide to include them or not.

The tests in DrScheme testing framework need a different evaluation order. They need to be evaluated after all of the other top-level expressions. Otherwise it becomes hard to test functions, especially for mutually recursive cases. To satisfy this need, tests are stored separate from the user's code and presented as extra information to the frontend application. This also allows the frontend to remove the tests if the destination platform is not appropriate for running tests.

DrScheme allows a user to embed images directly into source code. Since there is no direct method for accomplishing this in JavaScript, through the compilation process, embedded images are extracted from source code, and saved as image files to the disk. Images in source code are then replaced with a function call that loads those images on demand. The images field of the `pinfo` structure contains these names, in the order of appearance in the source code.

4.2 Data type correspondence

One of the most important challenges when translating one language to another is to define the data structures of the source language in the terms of destination language. Scheme, and more generally Lisp family of languages, are especially famous in that area because of their unorthodox nature of implementing various language features, such as numbers, Object Oriented Programming techniques etc... While JavaScript has some properties of functional programming languages, like higher order functions, it also lacks some features, like exact numbers or symbols of Scheme. Therefore, while it is possible to create a one to one correspondence in some data types like functions or booleans, many other data types requires special handling.

4.2.1 Numbers

Numbers in WorldWithWeb follows the Scheme number model closely, the numeric tower is implemented fully, including support for big numbers. The support for exact numbers is implemented using Matthew Crumley's BigInteger library[4].

The numeric tower is implemented using a class hierarchy that matches the hierarchy of numbers in the numeric tower. Inexact numbers of Scheme are double precision floating point numbers as defined by IEEE754 [7]. Since this model exactly fits to the JavaScript number model, JavaScript numbers are directly used for implementing inexact numbers.

4.2.2 Symbols

Symbols are one of the most interesting data structures that differentiate Lisp family of languages from others. Although a symbol is similar to a string in other languages, unlike a string it is guaranteed to be unique. While in many other scenarios symbols might

[4] http://silentmatt.com/biginteger/

be simply represented as strings, keeping the uniqueness invariant is important in this case, since the identity equality depends on that uniqueness feature. To accomplish this, symbol constructors are wrapped within a function that maintains a hash table of symbols produced so far. That way, if a requested name is already in the symbol table, it is returned. If the symbol is not in that table, a new symbol object is created and added to the table.

4.2.3 Characters

In many programming languages strings are just modeled as an array of characters. JavaScript, however, follows a different approach. In JavaScript there is no concept of a character. Instead, characters are modeled as strings with length 1. While a pragmatic approach might recommend to implement the same method, it is impossible to follow this method for implementing Scheme characters.

In Scheme, strings and characters are two distinct data types. There are also many functions that operate between these domains. For this reason, we decided to follow the Scheme approach and put a clear distinction between characters and strings. A character in WorldWithWeb is represented as a simple structure, holding an exact number, the Unicode code point of that character.

4.2.4 Structures

Structures in Scheme are simple data types to hold compound data. In BSL, a structure definition consists of a name and a list of fields. That definition introduces not only the structure itself but also a constructor, field accessors and equality tester. Since BSL is a purely functional subset of Scheme, structure definitions in BSL do not introduce field mutators.

Structures in WorldWithWeb are modeled as objects in the JavaScript's prototype based object system. A structure definition introduces an object which is built by the constructor function. Fields of the structure correspond to the the object's fields. Also field accessors and equality tester are defined as ordinary functions that work on object's fields.

4.2.5 Images

Being able to embed images directly in source code of a program is one of the most interesting features of DrScheme. This way, images can be used and modified like any other value in the language. Unfortunately JavaScript has no support for directly embedding images in the source code[5]. To handle this problem in a compatible way, images in the source code are saved as resources and they are replaced with a call to a function which loads the image on demand.

4.3 JavaScript Libraries

The second part of WorldWithWeb is the supporting libraries, written in pure JavaScript. These libraries imitate the primitive functions found in BSL and the world.ss teachpack. The libraries are designed to be exact imitations of their Scheme counterparts. For this reason, they consume the same number of parameters, produce same types of values as their Scheme counterparts and raise the same kinds of error messages.

4.3.1 Beginning Student Language

BSL contains the most basic functions for users to create applications. While its contents are limited to a subset of Scheme, it is big enough to let the users create useful programs. It mainly consists of number and string operators.

The BSL functions are implemented as ordinary JavaScript functions, using the datatypes mentioned above. The naming of the functions follows the naming scheme used by Moby Scheme Compiler.

4.3.2 Testing

There are two motivations behind the testing implementation of WorldWithWeb. First motivation is that testing is a crucial part of programming education. Getting used to writing proper test cases is important and the user should get feedback for his/her tests. While the user can test his/her program in DrScheme environment, it is also an extra safety measure to see that his/her tests pass on the web interface too.

Secondly, all projects need testing. WorldWithWeb is no exception. All test cases for WorldWithWeb libraries are written in Scheme and then compiled to JavaScript. This way we can be sure that as long as the tests pass in both environments our libraries are fully compatible with their Scheme counterparts.

The Scheme testing library provides three testing primitives. Two of them, `check-expect` and `check-within` are no different than any other function call. They are directly implemented as functions. On the other hand `check-error`, which checks if a given expression produces a certain error message cannot be implemented directly.

The error mechanism in JavaScript works using exceptions and in the evaluation order of JavaScript it is not possible to catch exceptions that happen while the arguments of a function are being evaluated. To handle this case, WorldWithWeb compiler wraps the expression that is expected to raise the error in an anonymous function, effectively delaying the evaluation. Later, when the test is evaluated the function is called and the expected exception is catched by ordinary exception handlers, and tested against the expected value.

4.3.3 Images

The world.ss teachpack consists of two parts. One part is the image.ss library which concentrates on creation and manipulation of images and shapes in various ways. The other part is the world.ss which manages the events from the outside world and controls the flow of world state between handlers for those events.

The images library is an imitation of image.ss teachpack found in DrScheme and used by the world.ss package. The library provides all of the functions that enables user to create static images and compose them in various ways. The JavaScript implementation also provides same primitive shapes and covers most of the image manipulation functions.

Images in WorldWithWeb are modeled as objects. All image objects provide a set of methods that makes it possible to use them in a generic manner. These methods include width & height calculations, pinhole alignment and a method called `__draw`.

The Scheme implementation of images relies on the underlying drawing primitives, provided by the GUI [6] framework. The images in that implementation are created as bitmap instances. This method allows the images library to use the methods of bitmap objects for all kinds of width/height calculations. Unfortunately, this approach is not possible in WorldWithWeb since it will require the implementation of a GUI style drawing system in JavaScript. Width and height calculations in WorldWithWeb is done directly by images themselves. For most of the primitive shapes, that approach works directly by applying appropriate formula for the shape. It is also possible to calculate this data for composite images. Although most shapes are implemented in a straightforward way, some of them (like triangles) do not produce the same result with their Scheme counterparts. This approach also fails on calculating the width and height of text objects.

The main challenge about images is the drawing of images on an HTML canvas element. The canvas element is a simple container,

[5] Actually, images can be embedded in data urls with base64 encoding, but this feature is not available in all browsers.

for a drawing context object. The drawing context provides simple drawing primitives like lines, bezier curves etc... The drawing of a world scene is accomplished by passing this drawing context through all the image objects in the current scene.

All image objects are designed to have a method called `__draw` which has three parameters. The first parameter is a drawing context, provided by a canvas element. The second and third parameters are the coordinates that the image object should draw itself. While ordinary objects just draw themselves in the drawing context, overlays, scenes and other composite objects manage the drawing of their sub-objects on the drawing context, passing the context from one element to another in the correct order.

4.3.4 World

As we mentioned in the previous topic, the world.ss library provides the abstraction mechanisms for the events coming from outside world. Although this is the main role of world.ss, it also provides some extra drawing primitives for creating "scenes".

In world.ss terminology, a scene is an image with a pinhole at 0,0 coordinates. For this reason, we implemented scenes as an extension of ordinary drawing primitives, provided by images library.

The main role of the world.ss is handling events. This is implemented in terms of timers and DOM events [13]. Mainly all handlers are defined as wrappers around the user-defined handler functions. For each event, the world is updated by the results of the handler function, and redrawn.

There are mainly three kinds of events in world.ss model. The first kind of event is the tick event, which is independent of the user interaction. It is the simplest event, which calls the handler function in each time tick. This handler is implemented as a JavaScript timer. JavaScript provides a `setTimeout` function which calls the provided handler in given periods.

The second kind of event is the user input. In plain world.ss, the only input user can provide is by using keyboard and mouse. The input from those devices are handled by key and mouse event handlers. In JavaScript, it is possible to access these events using event listeners. Each DOM object provides an interface, `addEventListener`, which allows the user to hook into the events occurring on that element. Using this interface, handler functions can access the details of the event (character code, mouse coordinates, modifier keys etc...). The handler wrappers for mouse and keyboard events get the raw JavaScript events and provide that information into user's handler functions in a format that imitates the Scheme interface.

The last event type is the redraw events. Redraw of the scene is actually not a real event but it is triggered by all kinds of handler events for the scene to represent the current state of the world in the window. Redraw events are currently invoked by handler functions, in a manual fashion. Each handler function invokes the redraw handler after changing the world.

Another handler, which is not tied to an event is `stop-when`. The `stop-when` handler decides when the animation should stop. When the condition checked by the handler is satisfied, all other event handling stops, effectively stopping the animation.

5. Possible Applications

While WorldWithWeb is mainly designed to be used within DrScheme programming environment, it is possible to use it as a library to provide different kinds of functionality from all types of applications. This way user might be provided with a richer environment which is adapted to his/her development environment. While it is possible to use the library to integrate user's application to various web services as discussed before, it might also be used in many different setups providing different services.

5.1 Interactive Environment

As Papert [12] noted in 1980's, programming plays an important role in a child's learning process. Unfortunately, many of the students around the world have no access to a personal computer of their own and need to use public computers instead. Because of the locked down nature of those public access computers, most of the time, the student will not have access to DrScheme environment. For many types of software, this problem is solved by rich Internet applications. This way, users have the opportunity to access their spreadsheets, instant messaging systems and all kinds of documents from anywhere in the world.

Following this idea of online applications, it is possible to create an online programming editor that will provide a simple editing environment for Scheme code. The user can write his/her code in that environment, and then send the code to the server by clicking the "Run" button on the web page. The server will compile the application to JavaScript and send it as a response to the user's web browser, to be evaluated. That way, the user can see the result of his/her code directly in the web browser without the need for any other tool.

5.2 Gadget-like applications

As we mentioned earlier, sharing lies in the heart of web. While there are many different methods for sharing different kinds of content, social networking sites became hubs where the sharing gets centralized. One of the most important feature of social networking sites are the "gadgets" they present to their users. A gadget is a small application, created using JavaScript and HTML. With initiatives like OpenSocial [10], it is possible to create a gadget and share it with other people across different social networking sites.

Since gadgets are just composed of HTML and JavaScript embedded inside a meta data container, it is possible to use WorldWithWeb to create these gadgets automatically. All that's required is to create a frontend that generates appropriate XML structure from the generated code.

Enabling this kind of sharing might be a real boost for the user motivation, since the user's application directly becomes a part of a network that is built especially for sharing purposes.

6. Conclusion & Future Work

World.ss is a library that provides abstractions that enables users to create complicated animations and simulations without going into the imperative roots of creating an animation. WorldWithWeb takes this one step forward, providing the user to share his/her creation on the web without worrying about the underlying protocols or languages. Although it does provide the user with everything he/she needs to create an interactive animation, there is still room for improvement.

6.1 Universe.ss

Universe.ss is a teachpack extending world.ss by enabling the user to create multiuser client/server applications like multiplayer games while staying in a purely functional programming environment. The core concept in universe.ss is the "message". A message is a simple packet of information contained in an S-Expression [15]. The client environment communicates with the server, and other clients using messages.

The application model proposed by universe.ss fits perfectly into the development model of Web 2.0 applications. A Web 2.0 application communicates with the server and other clients using simple messages encoded using various methods. One of these encoding methods is the JavaScript Object Notation (JSON) which makes it possible to encode any kind of JavaScript data in a simple text only format. Most Web 2.0 applications work by passing Java-

Script objects encoded in JSON format between client and server. The server application distributes this data to other clients using polling techniques.

This correspondence in methodology makes it possible to extend WorldWithWeb to cover universe.ss. The implementation of single user applications in universe.ss is trivial, since that part of the universe.ss is nearly the same with world.ss. The main difference of universe.ss shows itself in multi-user applications. To implement multi-user applications, the messages need to be encapsulated and sent/received using XMLHttpRequest's [16].

The implementation of the server side is a much more interesting problem. That application can be modeled as a proxy in front of the universe.ss server. The proxy can capture the JSON encoded objects, create the corresponding Scheme object, and pass it to the actual universe.ss server. Then the inverse of this method can be used to encode objects from universe.ss server going to the clients. The most important advantage of this method for implementing the server side is that the server application can be used as-is without any modifications.

6.2 Better Scheme Semantics

As noted by Loitsch and Serrano, compiling Scheme to JavaScript while keeping the Scheme semantics intact is not a simple process. The lack of first class continuations and similar techniques makes it impossible to make a direct translation between two languages.

Since WorldWithWeb just focuses on a subset of Scheme language, some of these problems (like first class continuations) disappear by themselves. However, some other things like exact numbers and proper tail calls remains. As we mentioned earlier, WorldWithWeb already implements exact numbers using a fractional numbers library.

Proper tail calls are currently not implemented. While this is not a serious problem for small applications, as the applications get more complicated, it is possible to hit a memory barrier. We are currently working on various methods for implementing tail calls in JavaScript.

6.3 Other Language Levels

While compiling BSL programs might motivate the student to a certain level, the tools should be available to him/her through the learning process. This requires adding support for other language levels to the compiler.

Although it is rather simple process to add support for language levels like "BSL with List Abbreviations" or "Intermediate Student" by just extending the library functions, the addition of "Intermediate Student with Lambda" language will require the addition of higher order functions and anonymous functions. This addition, probably will not be so hard, considering that JavaScript already has support for higher order functions and anonymous functions.

Adding the "Advanced Student" language will probably be the hardest part, since that language level introduces mutation. Introduction of mutation into the language creates two important requirements for the compiler: Sequencing and function call semantics.

In purely functional languages the evaluation order of the expressions do not effect the result of the computation. Breaking purity by introducing mutation requires that expressions are evaluated in correct order. When values can be mutated, the programmer should be able to know beforehand when the mutation will happen to be able to reason about the program. For this reason compiling a language with mutation should not effect the evaluation order. Since JavaScript is not designed to be a purely functional language, it already contains proper sequencing constructs. The only thing to be done is to make sure that JavaScript sequencing semantics are in correspondence with the Scheme sequencing semantics.

The second problem is preserving the function call semantics. In the presence of mutation, the results from functions might depend on the function call strategy used. To make sure that the JavaScript version of the program produces the same results with Scheme version, function calls might require special treatment.

Acknowledgments

We would like to acknowledge Chris Stephenson, M. Fatih Köksal and E. Pınar Hacıbeyoğlu for their support through the development process of WorldWithWeb and the preparation of this paper.

References

[1] *ECMAScript Language Specification.* 1999.

[2] Tim Berners-Lee. Information management: A proposal. *CERN, March*, 1989.

[3] Matthias Felleisen, Robert B. Findler, Kathi Fisler, Matthew Flatt, and Shriram Krishnamurthi. How to design worlds, 2008.

[4] Matthias Felleisen, Robert B. Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to Design Programs*. MIT Press Cambridge, Mass, 2001.

[5] Matthias Felleisen, Robert B. Findler, Matthew Flatt, and Shriram Krishnamurthi. Structure and interpretation of the computer science curriculum. *Journal of Functional Programming*, 2004.

[6] Matthew Flatt, Robert B. Findler, and John Clements. GUI: PLT graphics toolkit. Reference Manual PLT-TR2009-gui-v4.1.5, PLT Scheme Inc., March 2009.

[7] Matthew Flatt and PLT Scheme. Reference: PLT scheme. Reference Manual PLT-TR2009-reference-v4.1.5, PLT Scheme Inc., March 2009.

[8] Shriram Krishnamurthi. The moby scheme compiler for smartphones. In *Proceedings of the International Lisp Conference*, 2009.

[9] Florian Loitsch and Manuel Serrano. Compiling Scheme to JavaScript.

[10] J. Mitchell-Wong, R. Kowalczyk, A. Roshelova, B. Joy, and H. Tsai. Opensocial: From social networks to social ecosystem. pages 361–366, Feb. 2007.

[11] Tim OReilly. What is web 2.0: Design patterns and business models for the next generation of software.

[12] Seymour Papert. Redefining childhood: The computer presence as an experiment in developmental psychology. In *Proceedings of the 8th World Computer Congress: IFIP Congress*, 1980.

[13] Tom Pixley. Document object model (DOM) level 2 events specification. *W3C Recommendation, November*, 2000.

[14] Casey Reas, Ben Fry, and John Maeda. *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press Cambridge, Mass, 2007.

[15] PLT Scheme. Teachpacks. Reference Manual PLT-TR2009-teachpack-v4.1.5, PLT Scheme Inc., March 2009.

[16] Anne van Kesteren and Dean Jackson. The XMLHttpRequest object. *World Wide Web Consortium, Working Draft WD-XMLHttpRequest-20070618*, 2007.