Beyond the Demo
Designing and Building a "Real" Ur/Web Application

Benjamin Barenblat

May 16, 2013

Today, learning management systems are widespread. However, they are invariably difficult to maintain, slow, and proprietary. My 6.UAP project focused on designing a forum module for a different kind of learning management system – a speedy, free,[1] secure LMS written in the Ur/Web research language. While I didn't accomplish all my initial goals, I did produce a reasonably-sized body of Ur/Web code suitable for others to build upon in future development of both the proposed LMS and other, unrelated projects. I also identified a number of Ur/Web design patterns (and anti-patterns); I believe that promoting this "soft" knowledge is the royal road to widespread Ur/Web deployment.

1 Background

1.1 Learning management systems

The Web has always been associated with academia; it's thus natural that professors use it to disseminate course materials to students. With the Web's present ubiquity, entire systems have been designed to help professors generate dynamic, featureful course Web sites. These *learning management systems* typically integrate gradebooks, assignment distribution and submission systems, calendars, and occasionally even wikis or question-and-answer fora.

At MIT, rather than using an off-the-shelf LMS such as Blackboard [1] or Moodle [2], IS&T has developed and deployed a custom solution known as Stellar [3]. Stellar, while a generally functional product, routinely garners the ire of students,

[1] Throughout this report, I use the term "free" as defined by the Free Software Foundation; consider it roughly synonymous with "open source."

professors, and administrators alike: It is difficult to maintain, difficult to customize, and extraordinarily slow. Nonetheless, the MIT Committee on Educational Technology reported in 2011 that no commercial LMS provided the feature set and customizability of Stellar, recommending that MIT retain the Stellar system for the foreseeable future.

1.2 Ur and Ur/Web

Ur [4], designed and implemented by Professor Adam Chlipala, is an advanced, strict, purely functional, statically typed programming language intellectually descended from Haskell [5] and OCaml [6]. However, its type system is substantially richer than that provided by either language, being more in the tradition of dependently typed languages such as Coq [7]. In particular, while Haskell and OCaml functions may only map values to values, Ur functions may also map types to types. This enables metaprogramming à la Template Haskell [8] and Camlp4; unlike metaprograms written in those languages, however, Ur metaprogramming is typesafe – checked for well-typed behavior *before* expansion.

As its name implies, Ur is designed as an *ur-language*: a programming language which serves as a base to implement specialized, domain-specific languages. To date, only one DSL, Ur/Web, has been implemented; Ur/Web melds Ur's advanced metaprogramming features with a hefty standard library, enabling rapid development of RESTful, SQL-backed Web applications. The Ur/Web compiler (there is no separate compiler for Ur yet) compiles Ur/Web to C and JavaScript, allowing client and server code to be trivially combined in a single application, and the generated code is highly efficient. To sweeten the pot, Ur/Web promises to statically guarantee generated programs immune to code injection, cross-site scripting vulnerabilities, and certain types of cross-site request forgeries.

2 Motivation

Ur/Web has not gained much traction since its release in 2010; to date, only one large application exists written in Ur/Web, and it is nonfree. Discussion of the situation within Ur/Web's patron research group, the Programming Languages and Verification Group at CSAIL, suggests that Ur/Web faces the chicken-and-egg problem inherent in releasing any new programming language, and that the solution is to bootstrap Ur/Web by demonstrating its applicability to a large software system. The group, aware of the MITCET report, thus began an informal project to

construct a modular learning management system in Ur/Web. The overall vision describes a system in which professors would receive autogenerated Ur/Web skeletons for their course Web sites, which they would modify by adding or removing modules – e.g., fora, wikis, or calendars. I chose to focus on one specific module: a class forum which would combine the best features of Stack Overflow and Piazza to offer students a safe, well-designed place to get the answers they needed. At the same time, I would investigate Ur/Web best practices for applications "beyond the demo" – that is, large, complicated applications, which need to sustain continual development for a lengthy deployment period.

2.1 Prior work

2.1.1 Stack Overflow

In the world of top-tier online forums, Stack Overflow [9], catering to "professional and enthusiast programmers," is the undisputed leader – one would be hard-pressed to find a Course Sixer who has not consulted Stack Overflow at least once. Stack Overflow presents a clean, typographically sound interface; it runs fast on virtually any hardware selection, from twelve-core i7s to ULV Atoms; and for one reason or another, it has attracted a massive body of over 1½ million registered users. However, Stack Overflow is designed for professionals and enthusiasts, not for students. In fact, the Stack Overflow community has been at times openly hostile toward students – "Smells like homework" is a common response to introductory programming questions on the site. And while anyone can create a Stack Overflow clone specialized to another topic area, all such clones must be built through Stack Overflow's parent, Stack Exchange, and they all must be public. (The Stack Overflow software itself is nonfree, so running a private or independent Stack Overflow instance is impossible.)

2.1.2 Piazza

Piazza [10] attempts to fill this gap by providing private fora professors can create for use in their classes. Each forum provides a safe environment for (possibly anonymous) students to ask questions and for classmates and staff to answer. However, Piazza fails. In contrast to Stack Overflow, the Piazza user interface is ugly – its designer obviously believes that jQuery, not tasteful design, is the hallmark of a beautiful site. The site is sluggish, bringing my netbook to a crawl; the underlying software is nonfree; and there is no public API, making writing a better client a Herculean (and Sisyphean) task. Piazza's underlying model suffers serious issues

as well: unlike Stack Overflow, where a proliferation of answers is common (and indeed encouraged), Piazza supports only one real answer per question. Furthermore, while Stack Overflow's voting and reputation systems allow users to easily assess the quality of any answer, Piazza lacks any such mechanism other than direct staff endorsement.

On campus, reaction to Piazza is mixed. General consensus finds it better than nothing, but it obviously has a long way to go before it achieves wide satisfaction.

2.2 Designing a better system

Fundamentally, the major problem for both Stack Overflow and Piazza is that of audience. Stack Overflow caters to professionals and enthusiasts, and while Piazza claims to support students, the site's slogan is "The (Free) Efficient Way to *Manage* Class Q&A" (emphasis mine), making it clear exactly who the site is intended for – professors. In contrast, I sought to build a site intended for students, a site to provide students the speed and design of Stack Overflow with the privacy and support of Piazza. In short, I wanted to build a site optimized around a single primary activity: asking a question.

To guide my work, I put forth the following concrete design principles, roughly in priority order:

- A clean, intuitive interface will make the site a pleasure to use.
- A voting scheme will allow students to rate answers. Voting power will be weighted according to institutional authority – staff votes will carry more weight than student votes.
- The site code will be free so that institutions can customize it to their needs and students can easily code against it.
- The site will run fast on all hardware.
- A RESTful API will allow the site to interact well with other projects.

3 My contribution

For my 6.UAP project, I constructed a basic forum module for the hypothetical PLV LMS, a deployment of which is available at https://bbaren.scripts.mit.edu/urweb/6.947. (Note that MIT personal certificates are required for authentication; while

you can browse the forum as an unauthenticated user, you cannot participate by asking new questions, answering existing ones, or voting on questions or answers.)

3.1 Application architecture

Since I was the first person to write a module for the new LMS, I got to design the overall application architecture. To keep the system as modular as possible, each subapp – the forum, the wiki, the gradebook, etc. – lives in its own Ur/Web package, with its own Ur project file, its own static files, and potentially its own build system. (Ur/Web's compiler, as a whole-program compiler, makes a separate build system for each subapp currently redundant; however, if future subapps make calls to C code via Ur's foreign function interface, they will need integrated build systems, probably using autotools.) Each subapp presents a templating functor, which allows the main application to enforce a consistent look and feel over the entire site. As an example, the forum application presents the following functor.

```
functor Make(Template : sig
                    val generic : option string (* title *) → xbody → page
                end) : sig
    val main : unit → transaction page
end
```

The larger application parametrizes the forum and exposes the generated main function.

```
structure Forum = Forum.Make(struct
        fun generic (pageName : option string) (content : xbody) : page =
            <xml>
              <head><title>{[pageName]}</title> (* etc. *) </head>
              <body>
                (* Code to generate the page menu, etc. *)
                {content}
              </body>
            </xml>
    end)
val forum = Forum.main
```

3.2 The forum itself

The forum subapp is a fairly straightforward SQL-backed application. At present, it fulfills most of the design principles I describe in section 2.2. Its interface is

clean and the site code is free, licensed under the GNU AGPL [11]. The app is not as featureful as I desired – one can only vote on questions at present, not answers, and there is no support for weighting votes according to institutional role – but the app in its current state is functional and performant.

The main page of the forum lists the five most recently-asked questions, along with a friendly invitation to ask a new question. Users may enter a detail page for any question by clicking its title; the detail page allows voting on and answering the question.

Currently, the app relies on only two SQL tables – one for questions and answers (collectively, "entries") and one for votes.

3.2.1 Authentication

The forum leverages Ur's type system to enforce policies regarding user access and mutation. In particular, the Author module defines two types:

```
type usernameOrAnonymous = option string
type username = string
```

(These types are defined as synonyms, rather than wrappers, to ease SQL serialization; see section 4.3.) The types are exported abstractly; thus, functions which require users to be authenticated may accept a username as a parameter, while those which allow anonymous access should accept usernameOrAnonymous parameters.

Currently, there is no SQL table for users; instead, users are identified entirely through MIT personal certificates (SSL client authentication). Whenever a client connects to a page, the page may use the function

```
val Author.current : transaction Author.usernameOrAnonymous
```

to request the MIT username of the connected user. Author defines the function

```
val name : usernameOrAnonymous → username
```

which pages may use to take differing actions based on whether or not a user is authenticated. For instance, the detail page uses Author.name to determine whether or not to display voting buttons.

3.2.2 Voting

Voting appears a deceptively simple scheme. However, it is actually quite complicated to build a voting system incorporating well-defined invariants – e.g., that each user may vote only a finite number of times. To simplify the problem, I chose

to allow each user up to one vote per question, and I temporarily abandoned my goal to incorporate users classes (professor, student, etc.) into vote weight. The resultant system stores each vote in a vote table, and it uses SQL constraints to ensure that all votes are valid (and that each user only votes once).

4 Ur/Web in practice

In general, I was quite pleased with Ur/Web's performance as a programming language, and I'd certainly give it serious consideration when determining which language to implement my next Web-based project in. The Ur/Web edit-compile-test cycle was fairly rapid, and once I had a few support scripts set up, redeployment was trivial. The language is generally well-documented – if only within the voluminous reference manual – and minimal examples, in the form of Ur/Web demos, cover a surprisingly large number of potential use cases for the language. That the Ur/Web compiler is free software was also a great boon; several times during development, I encountered strange compiler bugs, and I was virtually always able to patch the compiler and continue working. (Some of the bugs are specific to the manner in which I deployed Ur/Web; others, which have wider ramifications, I plan to enter into the Ur/Web bug tracker shortly after this project's conclusion.)

4.1 Deployment on Scripts

I chose to deploy my project on SIPB's Scripts service [12], a shared hosting platform available to the MIT community. I was generally happy with deployment on Scripts, though I encountered a few minor issues. First was the simple issue of getting a working Ur/Web binary to exist on Scripts; since MLton produces dynamically linked executables, I needed to compile Ur/Web on Scripts itself. Fortunately, Scripts allows SSH access, and thus, with a bit of help from the service maintainers, I was able to produce a working `urweb` executable.

However, the compiler did not work on first invocation – it was developed on Debian, while Scripts runs on Fedora. Thus, when performing its final link, the vanilla Ur/Web compiler generates a command line for GCC which is incompatible with the Scripts architecture. Happily, a trivial compiler patch fixed this.

So that others might have an easier time working with Ur/Web on Scripts, I have made available my installation of the Ur/Web compiler (version 20120925) via AFS. If you are deploying an Ur/Web application on Scripts, you can run `. /mit/bbaren/ur/setup.sh` (note the initial period) to set your environment variables

appropriately; then, when you run urweb, you'll run it out of my locker. You will need to compile your application on Scripts itself, and since the Ur/Web runtime is not installed globally, you'll need to pass the −static flag to the compiler.

4.2 Cross-site request forgery

My friends, upon hearing me brag about Ur/Web's security model, enjoyed banging on the application to try to break it. (I remember in particular one friend, who immediately stuffed script tags into a question and applauded when Ur/Web refused to treat them as HTML.) I was eventually forced to eat my words by David Benjamin, however, who discovered cross-site request forgery vulnerabilities in my application. Ur/Web promises to protect against certain types of cross-site request forgery; however, the Ur/Web compiler currently only defends against cross-site request forgery when cookies are used for authentication. Since my application did not use cookies, I inadvertently deactivated the CSRF protection code paths, and my application became vulnerable.

The workaround in this particular case (see commit b446185d1ec6) was to trick Ur/Web into believing I am using cookies when I am not. For instance, the Author.current function described in section 3.2.1 now begins

```
cookie bogus : unit
val current =
    addressOpt ← getenv (blessEnvVar "SSL_CLIENT_S_DN_Email");
    _ignored ← getCookie bogus;
    (* ... *)
```

getCookie bogus will always return None, but it doesn't matter – Ur/Web is (fortunately) insufficiently intelligent to determine that the cookie is ignored, and so this will properly trigger Ur/Web's CSRF protection at the appropriate time. An unfortunate consequence of this approach is that I had to sacrifice another of my secondary goals, a RESTful API for the forum; I simply lack the time to build a real authentication system that enables secure API use without admitting CSRF.

It's unclear to me precisely what impact this issue has upon the wider Ur/Web ecosystem. Very few developers use SSL client authentication, so it's unlikely that this issue will manifest itself outside MIT. It should be possible to add certificate authentication primitives which would trigger the Ur/Web CSRF protection code paths to the compiler itself; however, that project is certainly beyond the scope of this one.

4.3 The pattern matching anti-pattern

One clear anti-pattern I identified through work in Ur/Web (as well as work on my Decaf compiler in 6.035) is pattern matching on data constructors outside of the module in which they were defined. While it may be tempting to define `data Foo = Bar | Baz` in one module and pattern-match on `Bar` and `Baz` in another, this is (to use Professor Chlipala's favorite insult) anti-modular, coupling `Foo`'s interface to its implementation in such a way as to make later implementation changes extremely time-consuming. I strongly encourage developers to avoid pattern matching across module boundaries, and indeed, to export all types opaquely.

In Ur/Web, there is an even larger incentive to export types opaquely. The `sql_injectable` type class (representing those types which are trivially serializable into SQL tables) is a closed class, meaning developers cannot add developer-defined types to it. Even if an isomorphism exists between a developer-defined type and (a subset of) a built-in type, the developer is out of luck: he or she must manually serialize and unserialize values of the type when dealing with database operations. While the Ur/Web type system prevents the developer from forgetting to serialize correctly, an even better solution is to declare the isomorphism in the form of a type synonym. The `Author` module follows this pattern (see section 3.2.1). However, this pattern only works if the type is exported opaquely: if the developer exposes the isomorphism to the rest of the program, usage of the type becomes unsafe, precisely because the developer can now pattern-match on the isomorphic, but unrelated, underlying type.

5 Future work

While I am pleased with the work I did this term, much future work remains. The forum module is certainly not ready for prime time, and I'm hopeful that others (and, potentially, myself) will continue work on it after I leave the Institute. More generally, however, it's clear that Ur/Web visibility needs to improve substantially for it to acquire real market share.

I believe much progress can be made by improving the human factors:

- Ur/Web needs a better build system. The project abstraction is generally pleasant to use, but certain small details – such as its lack of comment support – threaten to render it unsuitable for extremely large projects.
- The Ur/Web compiler needs better error messages. At this point, I am extremely good at deciphering the messages it presents; I have even arrived

at the point where the messages are occasionally helpful, rather than universally confusing. However, for those who do not know what "unification" is, or that the HTML tags embedded inside Ur files are secretly desugared to XML-building combinators (with fairly complicated record types, I might add), or that type variables in Ur/Web type signatures are not automatically universally qualified, urweb's error messages are less than ideal.

- Parts of the Ur/Web standard library need prose documentation. Type signatures in the standard library interface files provide *almost* enough documentation to use the standard library to its fullest potential. However, hypertext (or even just PDF) documentation would allow better searchability and easier perusal, particularly as Ur/Web stops being distributed in source form.

Beyond this, however, I believe the best future for Ur/Web is simply one in which developers use it. As more developers use Ur/Web and recommend it to their coworkers, more code examples will appear online, more documentation and tutorials will become available, and more feedback on improvement will return to the authors. I look forward to seeing the forthcoming Ur/Web learning management system as a large, complicated Ur/Web system – should it succeed, it will not only be a major boon for the Ur/Web community, but one for MIT and the collegiate environment as well.

References

[1] *Blackboard: Technology and Solutions Built for Education*. http://blackboard.com/.

[2] *Moodle: Powerful Free Tools to Educate the World*. http://moodle.com/.

[3] *Stellar: The* MIT *Course Management System*. https://stellar.mit.edu/.

[4] Adam Chlipala. *The Ur Programming Language Family*. http://impredicative.com/ur/.

[5] *The Haskell Programming Language*. http://haskell.org/.

[6] Institut national de recherche en informatique et en automatique [INRIA]. *The Caml Language: OCaml*. http://caml.inria.fr/ocaml/.

[7] ———. *The Coq Proof Assistant*. http://coq.inria.fr/.

[8] Tim Sheard and Simon Peyton Jones. "Template Meta-programming for Haskell". In: *Proc. Haskell Workshop* (2002). https://research.microsoft.com/en-us/um/people/simonpj/papers/meta-haskell/. See http://haskell.org/haskellwiki/Template_Haskell for information on the current state of Template Haskell.

[9] *Stack Overflow*. http://stackoverflow.com/.

[10] *Piazza: The (Free) Efficient Way to Manage Class* Q&A. http://piazza.com/.

[11] Free Software Foundation. "GNU Affero General Public License." 2007. https://www.gnu.org/licenses/agpl.html.

[12] Student Information Processing Board. *scripts.mit.edu*. http://scripts.mit.edu/.