The Continue Server (or, How I Administered PADL 2002 and 2003)

Shriram Krishnamurthi[⋆]

Computer Science Department
Brown University
Providence, RI, USA
sk@cs.brown.edu

Abstract. Conference paper submission and reviewing is an increasingly electronic activity. Paper authors and program committee members expect to be able to use software, especially with Web interfaces, to simplify and even automate many activities. Building interactive Web sites is a prime target of opportunity for sophisticated declarative programming languages. This paper describes the PLT Scheme application Continue, which automates many conference paper management tasks.

1 Introduction

The submission and review phases of computer science conferences have become increasingly electronic. Indeed, many conferences operate without using any paper until the copyright forms come due. The many phases of this process—paper submission, download by reviewers, review submission, paper discussion, review dissemination and final paper submission—all employ electronic media and formats.

The increasing use of automation benefits authors and reviewers alike. Authors no longer need to mail bulky cartons containing several copies of their submissions. Program chairs no longer need to receive and store these cartons. And having the papers in electronic form helps reviewers search and bookmark fragments of a paper.

All this automation places immense pressure on one group of individuals: the program committee (PC) chairs. They are forced to install servers, which is often an onerous task, and worse, to maintain them. They must fret about privacy and security, which they may ill-understand. They need to support a variety of platforms and formats, all while providing a reasonable user interface. Finally, they must support the electronic management of the review process.

In the early days of (partial) conference automation, interfaces tended to demand that authors FTP to a particular site, save their paper in a hopefully unique filename, and then notify the maintainer by email of their upload. Server installation and maintenance was therefore easy, because host departments usually provided FTP facilities. Security was usually accomplished by making the

[⋆] This work is partially supported by NSF grants ESI-0010064 and ITR-0218973.

directory unreadable, but this made it difficult for authors to ensure their file had been uploaded successfully. Invariably, authors would forget to, say, set the transmission type to binary, causing an entirely avoidable exchange to ensure. Those were not halcyon days.

The development of the Web as a medium for interaction has greatly helped conference submission. The Web provides many of the benefits of the FTP approach, while tying it to a better interface. As a result, several servers now perform conference paper management. Unfortunately, some of these servers are difficult to install, and some are even commercial enterprises (in return for which they ostensibly provide professional, around-the-clock service).

The computer science community could benefit from a simple yet powerful server that provides most of the features currently available from existing servers. By exploiting the powerful features of declarative languages, such a server can demonstrate the benefits of declarative programming to a wider audience. This paper presents CONTINUE, a Web package that meets both these requirements.

2 CONTINUE as an Application

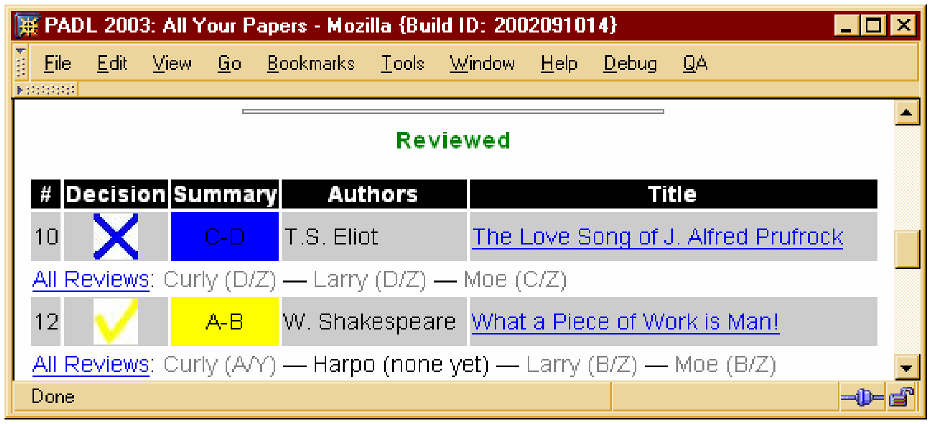
CONTINUE has two heads. It presents one face to the submitter, and entirely another to PC members. We will describe each of these in turn, focusing on the most useful or interesting features.

In general, CONTINUE looks much like any other server program or conference manager: maybe a little less slick than others, but also less garish. This commonness of interface is important, because it means users are less likely to be confounded by the interface, and consequently less likely to blame a language for the program's author's faults.

2.1 Submitting Papers

One of the more vexing problems that PC chairs face is incorrect contact information for an author. The first line of defense against this is to ensure that authors provide valid email addresses. And the simplest way of ensuring an address is valid is to actually use the address to establish contact. This is a common pattern now used by many registration sites, and we view the act of submitting a paper as a form of registration. (Erecting a barrier is also a valuable safeguard against junk submissions, spam, etc.)

To wit, the primary submission page asks authors to enter an email address and other contact information (particularly their Web page URL). It sends a message containing a special URL to the address input. The author needs to use the transmitted URL to continue the submission process and, in particular, to obtain a form that permits file upload. CONTINUE implements the standard HTTP upload protocol, so users will find nothing unfamiliar about using this server. (The actual protocol implementation is part of the PLT Web server API.)


PADL 2003: All Your Papers - Mozilla {Build ID: 2002091014}
File Edit View Go Bookmarks Tools Window Help Debug QA
Done

Reviewed

#	Decision	Summary	Authors	Title
10	✗	C-D	T.S. Eliot	The Love Song of J. Alfred Prufrock
All Reviews: Curly (D/Z) — Larry (D/Z) — Moe (C/Z)				
12	✓	A-B	W. Shakespeare	What a Piece of Work is Man!
All Reviews: Curly (A/Y) — Harpo (none yet) — Larry (B/Z) — Moe (B/Z)				

Fig. 1. Reviewer Interface

2.2 Reviewing Papers

Papers are assigned to PC members by the PC chairs. A chair's interface provides access to this feature, but it is not made available to individual PC members. Only a PC member assigned to a paper can submit a review for it (though I plan to add a feature whereby *any* PC member can submit a "note" to the authors: see section 4.1). To make assigning PC members more flexible, Continue offers PC chairs the option of downloading a list of all the papers as an XML document. The chairs can use XSLT or other transformation tools to endow each paper with reviewers. Continue accepts the resulting XML document and converts the embellishments into reviewer preferences in its internal format.

Each PC member logs in using a unique username and password to obtain their assigned papers. The default page a PC member sees lists the following:

– a progress bar showing how many papers they've reviewed, and how many remain;
– a link (anchored by the text "How are the others doing?") that lets them compare their progress against that of the other PC members;
– a link to all submitted papers;
– papers for which their review is pending;
– papers they have finished reviewing (see figure 1).

PC members have two incentives for completing reviews. One, of course, is the progress bar comparator. The other is that they cannot see reviews by other PC members until they have filed their own review. While this feature is obviously easy to circumvent (by submitting a blank review), we hope social forces will prevent most PC members from doing so. At any rate, it does ensure that a PC member cannot accidentally view another member's review before completing their own, thereby reducing bias in their own evaluation.

One vexing question that faces authors of conference managers is how to cleanly summarize reviews. This in turn depends on the reviewing system the conference adopts. Because of this dependency, in principle, the paper management software should be parameterized over the review regime, and should perhaps offer a domain-specific language for specifying how to summarize data.[1]

Continue does not currently adopt this complex strategy. Instead, it implements Oscar Nierstrasz's "Identify the Champion" (henceforth, ItC) pattern for program committees [14]. Like most review regimes, ItC includes two summary data on each paper: the overall evaluation, and the reviewer's expertise. What distinguishes ItC is, rather than using passive terms such as "good" and "bad", it uses active phrases such as "I will champion [this paper] at the PC meeting" and "I will argue to reject this paper". Nierstrasz argues that this language, and the notion of making champions explicit, will help a PC more rapidly converge on papers that it is actually going to accept.

While ItC prescribes codes for reviews (A (will champion)–D (argue against) for overall quality, X (expert)–Z (non-expert) for expertise), it does not dictate a presentation mechanism. We therefore experimented with a number of different schemes that kept the two units of data separate. These were somehow unsatisfying, because they cluttered the screen without providing appreciably more information. The clutter was increased by the presence of another, unavoidable, field: the final decision.

The present scheme, implemented by Continue, is due to Phil Wadler. Wadler's scheme is to ignore the expertise (since this is often reflected in the overall rating) and deal with *ranges* of overall ratings. The lexicographically ordered pairs of ratings form a triangular matrix. Wadler's scheme assigns a color to each corner, and lets them (roughly) fade toward one another. It puts the most visually striking colors at the corner where there is most need for discussion due to the greatest variance of opinion (A–D). To wit:

A	B	C	D	
yellow	yellow	orange	red	A
	yellow	green	purple	B
		green	blue	C
			blue	D

Continue designates the final decision using the same color scheme. Thus, accepted papers get a yellow tick, and rejected papers a blue cross. (The blue cross on a white background looks remarkably like an inverted Saltaire.)

[1] An alternative is to use a swarm of colors and codes to cover almost every possible situation. One popular conference manager uses so many colors, it pops up a window of color codes every time a PC member logs in. Since pop-ups usually contain advertisements, many Web users have come to regard pop-ups as nuisances and close them immediately. The unfortunate PC member must then click on a link to see the color codes explained again. Convenience notwithstanding, I believe having a pop-up to explain color codes is a symptom, not a solution.

3 Continue as a Program

The slogan for the implementation of Continue is

Continue = functional programming + interactivity + databases

The API design means the interactive portions also appear functional, resulting in a very elegant and maintainable program.

3.1 Caching and Performance

At the outset, Continue had to decide how much to generate dynamically and how much should be cached statically. For instance, should Web pages of review summaries always be regenerated, or should they be stored and updated only when a review changes? The tradeoffs are fairly obvious, but worth recounting. Static caches increase performance, but also introduce cache effects: delayed propagation, concurrency, and so forth. So the decision hinges on the frequency of updates and the need for up-to-date information.

For a conference paper review system, performance is governed by the following parameters:

– The load distribution is not uniform. Most days few or no reviews are logged. There is a flurry of submissions as the deadline approaches, and a few may come in only after the deadline.[2]
– Reviewers are often spread around the globe, so there is no "safe" time for re-generating caches.
– The cost of stale data can be high. When discussing papers, for instance, it sometimes really matters whether a paper was assigned a slightly favorable or slightly unfavorable review. PC members need to be able to instantaneously see recent additions as well as changes.

An early version of Continue (used for PADL 2002) performed extensive caching. Unfortunately, this led to greater manual overhead for the server maintainer to ensure that information propagated quickly enough. For the 2003 edition, I therefore decided to experiment with generating all information dynamically. Would the server cope?

With Continue, there is an additional performance parameter that needs tuning. Every interaction point captures a new continuation; as we have explained elsewhere [9], these continuations are difficult to garbage collect, because a server cannot know when a client might still maintain a reference to them.[3] The PLT Scheme Web server provides a parameter that allows the user to specify a timeout period following which continuations are reaped. Because this timeout

[2] We won't name names.

[3] Though there are space-efficient implementations of continuations [10], PLT Scheme cannot exploit them because of the constraints of interoperability with languages hostile to garbage collection.

can be inconvenient to users, and also as an experiment, we ran CONTINUE with no timeouts.

I'm happy to report that, for the reviewing phase of PADL 2003 (which received 58 submissions), CONTINUE held fast. There were no appreciable lags, nor did space usage skyrocket. The decision phase consumed the following resources (these figures represent an instantaneous snapshot taken just after decisions were made):

total memory:	159 Mb
resident set size:	159 Mb
CPU time:	49 minutes

These are modest numbers for a modern server application!

3.2 Functional Programming

A good portion of CONTINUE code is straight-up functional programming: maps, filters and folds. This is hardly surprising, because these looping paradigms accurately capture many presentation tasks. Want to turn database records into HTML rows? Map the row constructor over them. Want to select a reviewer's papers out of all submitted ones? Filter by whether they have been assigned to the paper. And so on.

CONTINUE also benefits from the ability to conveniently construct HTML using Scheme s-expressions (a benefit noted by several others). In general, programmers are more likely to construct abstractions if the language makes this easy. The ease with which s-expressions support HTML construction makes it convenient to define style sheets, resulting in benefits both external (interface consistency) and internal (code reuse). For instance, this template generates all conference pages:

```
(define (generate-web-page title bodies)
  `(html
    (head
     (title ,conference-short-name ": " ,title))
    (body [(bgcolor "white")]
          (p (h3 (a [(href ,conference-home-page-url)]
                    ,conference-short-name)))
          (p (h4 ,title))
          (hr)
          ,@bodies
          (hr)
          (p ((align "RIGHT"))
             (small
              "Web site implemented using the "
              (a ((href "http://www.plt-scheme.org/"))
                 "PLT Scheme Web server"))))))
```

Here is one more example, which combines higher-order function use with s-expression based HTML generation. This function generates the progress bar of PC member reviews (see section 2.2):

```
(define (make-pc-progress-list)
  `((p [(align "center")]
      (table
       [(width "90%")]
       ,@(map (lambda (pc-member)
                `(tr
                  (td  [(width "30%")
                        (align "right")]
                   ,pc-member)
                  (td  [(width "70%")]
                   (table [(width "100%")]
                    (tr
                     ,@(pc-member-progress-bar pc-member))))))
              pc-keys)))))
```

3.3 Interactivity

Continue's name pays tribute in two directions: to the START conference manager by Gerber, Hollingsworth and Porter at Maryland, an early utility of this kind, and to continuations, which are at the heart of Continue's interactive capabilities. Graham [4], Hughes [11], Queinnec [15] and others have all noticed the profound utility of continuations to Web programming; Continue puts this observation to work.

The PLT Scheme Web server provides a primitive called **send/suspend** [9]. This primitive captures the current continuation and stores it in a hash table, associated with a unique name. It then generates a URL which includes that name such that visiting the URL results in the Web server extracting the continuation from the table and invoking it. **send/suspend** takes as an argument a function of one argument, which it invokes on this generated URL. (As we will see below, this function usually—but not always!—generates a Web form with the provided URL as the form's handler.)

The **send/suspend** primitive makes writing interactive Web programs considerably easier. Instead of breaking down a single program into multiple sub-programs, which must then carefully orchestrate their communication, storing data in hidden fields, cookies, databases and so on, **send/suspend** permits the programmer to code in direct style, so all the benefits of lexical scope are conveniently at hand.

For instance, here is a typical use of **send/suspend**. This is the top-level routine for PC members:[4]

[4] Some code fragments have been cleansed slightly for presentation purposes.

```
(define (go pc-member)
  (choose-action
    (request-bindings
      (send/suspend
        (lambda (k-url)
          (generate-web-page "All Your Papers" ···))))
    pc-member))
```

Note how the use of **send/suspend** makes the structure of this program as simple as if it had been reading input from the keyboard. In particular, the program can refer to identifiers such as *pc-member*, which are lexically bound, without the contortions that the CGI protocol normally engenders [8].

Not all uses look quite so functional. Here is a sequence of interactions with the user, extracted from the code where CONTINUE requests and then saves a paper's review:

```
(begin
  (send/suspend
   (lambda (k-url)
     (show-submission-page-w-review-link key k-url)))
  (write-review pc-member
                key
                (request-bindings
                 (send/suspend
                  (lambda (k-url)
                    (show-submission-and-review-page
                     key k-url
                     (read-review pc-member key))))))
  ···)
```

In this fragment, the first **send/suspend** returns no useful value. It simply generates a page that the user must click through to generate the review form. The second page actually receives a value from the form, extracts the essential data, and saves the PC member's review. (Note that *pc-member* is a reference to a lexically scoped identifier.)

There is one other interesting use of **send/suspend**, for handling email confirmation (see section 2.1). It's worth stepping through most of this function (figure 2). First, the header is in $\boxed{0}$. The first thing the function body does is ask the user for their contact information ($\boxed{1}$). If the user enters invalid information, CONTINUE signals an error and asks again ($\boxed{2}$). Otherwise, it installs an exception handler (elided) and sends the URL in a mail message ($\boxed{3}$).

In particular, after sending the message, this code generates a page so the user can verify the address to which the server sent the message. If the user detect an error, he can click on a button, which resumes computation with *get-contact-info*—thereby generating the contact information form all over again. In contrast, if he does actually receive the URL in his mail and pastes it, computation

```
\boxed{0} = (define (get-contact-info email name url) \boxed{1})
\boxed{1} = (let* ([contact-bindings
                    (request-bindings
                      (send/suspend
                        (page/email-info email name url)))]
                  [email (get-binding 'cont-auth-email contact-bindings)]
                  [name (get-binding 'cont-auth-name contact-bindings)]
                  [url (get-binding 'cont-auth-web-page contact-bindings)])
              \boxed{2})
\boxed{2} = (if (or (string=? email "") (string=? name ""))
               (begin (send/suspend (page/empty-inputs email name))
                      (get-contact-info email name url))
               \boxed{3})
\boxed{3} = (begin
              (send/suspend
                (lambda (k-url)
                  (send-email email name url k-url)
                  (send/suspend (page/verifying-email-address email))
                  (get-contact-info email name url)))
              (get-submission email name url))
```

Fig. 2. Email Confirmation Code

resumes at the **send/suspend**, whose continuation is the invocation of *get-submission*—which enables the user to actually upload the paper.

These examples illustrate how continuations greatly simplify the construction of powerful interactive Web programs. The last use was an unexpected benefit of having the full power of continuations available to the programmer, because this was not an application we foresaw when first designing the server.

3.4 The Database

Continue uses a database that

– is relatively fast,
– supports concurrent access,
– has been tested extensively,
– is easy to install, and
– on most systems, is supported by backup:

the *filesystem*!

There are natural reasons for using a traditional database manager for storing records. But there are several equally good reasons for Continue's choice:

Installation Simplicity No installation is necessary. The filesystem is already "running" when the user installs Continue. Database managers are often not trivial to install, and CONTINUE's design goal is to make it easy for people to run conference sites. Even just the installation instructions become vastly more complex, and become highly dependent on the individual platform.

Maintenance Simplicity Once installed, someone needs to keep an eye on the database manager, and introducing an additional component results in an extra point of failure. It's pretty easy to diagnose that the filesystem is down and, if it is, it doesn't matter whether the database is up.

Programming Simplicity There is admittedly an allure to keeping the program simple also. Though most languages offer database primitives (PLT Scheme's database manager, SrPersist [18], supports the standard ODBC protocol), sometimes quite nicely (SchemeQL [19] provides an elegant Scheme interface to SQL), the installation and maintenance overhead do not appear to justify the effort.

Ease of Access In an emergency, the filesystem is easy to access without knowing to use query languages or sophisticated clients.

It would appear that the main reasons for *not* using the filesystem are efficiency and concurrent semantics. CONTINUE uses files thoughtfully to minimize the possibility of concurrent writes, so that the only circumstances under which one document might overwrite another are the same that would occur with a database.[5] We have not found efficiency to be a problem in practice (especially given the latency introduced by the network). Most reviews and other data files tend to be small. CONTINUE stores them as Scheme s-expressions, and Scheme implementations tune the *read* primitive to operate as efficiently as possible.

3.5 Security and Privacy

Security is vital in an application such as CONTINUE. The need for security, however, creates a constant conflict against the desire for convenience. CONTINUE cannot protect against the malicious or careless PC member handing out their password, thereby compromising the entire process. But it can create mechanisms that make it less necessary for PC members to be tempted to disperse this information.

The tradeoff classically arises when a PC member P wants to request a subreviewer S to examine a paper. P does not want to have to download the paper and a form, save them to files, mail the files, then manually copy S's feedback back into a form. In turn, S is likely to perform their service better if they were given a pleasant interface. This may tempt P to send S a password. CONTINUE makes this unnecessary by creating one-shot passwords *implicitly.*

The URL that CONTINUE (really, the PLT server) generates has this form:

```
http://host/servlets/pc-member.ss;id281*k2-95799725
```

This URL is essentially unforgeable. Therefore, P can send the URL generated for the review form to S. The review form contains all the necessary data (the

[5] CONTINUE could easily keep backups of all overridden files, though this would require designing an interface for recovering data from the backups.

abstract, paper and review form), so P does not need to send other files as attachments. In turn, once S submits a review, the URL *ceases to be active*! CONTINUE enforces this by using the **send/finish** primitive in place of **send/suspend** after receiving the review. The **send/finish** primitive reaps the continuation after its use, so subsequent attempts to access it will not succeed. Though **send/finish** was defined primarily to control resource usage, it has proven extremely valuable for defining policies such as this. Once S has submitted his review, P can edit it, secure in the knowledge that her changes cannot even be seen, much less changed, by S. P must explicitly generate a fresh URL for S so if she wants S to make changes.

CONTINUE also uses the traditional HTTP password mechanism to protect against inadvertent access. Therefore, simply stripping off part of the generated URL is insufficient: the intruder must know the username and password of a PC member also. This appears to be about the same level of security that other Web conference managers employ.

CONTINUE does assume that the PC chair(s) can see all decisions. This appears to inevitable for an effective decision-making process. Since many conferences disallow submissions by PC chairs, this seems to be a reasonable decision.

Finally, CONTINUE does ensure a paper co-authored by a PC member is treated with care. It prevents the PC chair from (accidentally) adding the PC member as a reviewer of their own paper, and anyway ensures that the paper does not show up on the PC member's view of assigned or all papers. Thus, a PC member should not be able to determine who wrote their reviews.

4 Future Directions

First we'll examine some observations about users and uses, then discuss some interesting research questions that CONTINUE engenders, and conclude with intended features.

4.1 Usability Issues

Observing the use of a running program raised several usability concerns to contemplate and address.

First, PC members sometimes sent the same review URL to more than one sub-reviewer. The second sub-reviewer who attempted to submit a review found that their URL had expired (see section 3.5). This was clearly a failing on my part, because I hadn't adequately explained this to users. (It's easy to say this should be in the documentation, but as Joel Spolsky notes, "Users don't have the manual, and if they did, they wouldn't read it" [17].) It is anyway unclear what to do in this case, since one PC member can't count themselves twice!

CONTINUE currently does not allow a PC member to file *comments*—or even the equivalent of an amicus brief—as distinct from *reviews*. Yet PC members often have a brief note to make to the author, but haven't read the paper in enough detail (or lack the authority) to file a full review. This would be a valuable feature, along with room for comments from the PC meeting.

There may be a bug in CONTINUE. A PC member complained that the server had lost one prior review. Investigation suggests that the fault may partially have been the user's (a sub-reviewer): it appears that they may have made the browser stop part-way through the review submission process. Since we have seen this behavior only once, it's unclear where the problem lies.

Finally, some mail agents take undue liberties with email messages. In particular, they feel free to reformat mail text by inserting line breaks at apparently appropriate locations. Sometimes, however, the text they break may be a URL, such as that for continuing the submission process (this process is described in sections 2.1 and 3.3). For PADL 2002, we received some complaints of non-functional URLs that we traced to this problem. (One paper submitter, in particular, initiated the process five times!) We could have used MIME-formatted messages, but instead made some changes to the URL the PLT server generates. As a result, we received no complaints for PADL 2003 (or other conferences that have used CONTINUE), nor saw instances of repeated initiation.

4.2 Research Problems

Since I also do research in computer-aided verification, I have thought about modeling and verifying CONTINUE's behavior. CONTINUE has several behavioral properties worth verifying. For instance: No PC member should have access to reviews for their own paper (because this would compromise the identity of the reviewers). PC members should not be given access to their paper when they ask to see all submitted papers. A paper may have *multiple* PC member authors, and the software should be sensitive to this fact.[6] No sub-reviewer can override the review of a PC member without permission (in the form of a fresh URL). The reviewers of a paper should only be in the set of those assigned to review it by the PC chairs. A PC member should (presumably) not have both a note and a review for the same paper. The list continues.

It should be clear from surveying this list that most properties involve a strong interplay between data and control. While automated tools such as model checkers [2] are very good at verifying control properties, it is unclear that they are appropriate here for two reasons: (1) they are generally weak at modeling rich data, and (2) it is unclear how to effectively model continuations (beyond unfettered nondeterministic jumps, which would greatly increase the expense of verification). Theorem provers are generally effective for reasoning about data, but may not have sufficient automation for discharging some of these properties, especially the temporal ones. Paul Graunke and I attempted to construct a model using Alloy [12], but had difficulty effectively representing continuations. At any rate, I believe this remains an open problem for verification research.

[6] One manual mailing in 2002 failed to account for this, but the other PC member was kind enough to point out the mistake before any information was compromised.

4.3 Features

CONTINUE currently provides no useful sorting features. A PC member—and PC chair, in particular—might want to sort papers by status, quality and so on, to better order an upcoming discussion. It would be relatively easy to implement each feature, but I need to experiment with different interfaces for presenting the numerous options: Clicking on table headers? Buttons? Pull-down menus?

I believe the most significant missing feature is discussion tracking. Experience suggests that PC members are loath to discuss over almost any medium other than email. In particular, Web forms are often unwieldy, especially due to the poor editing interfaces in most Web browsers. Editing tools aside, the problem remains: how to associate messages with papers? Invariably, a discussion will span more than one paper (especially if the conference has related submissions), and it must then be associated with each of those papers.[7] It is especially frustrating when a message is stuck in the archive corresponding to a different paper—some messages really should be in multiple archives.

The easiest proposal is to ask authors to annotate each message with paper identifiers, perhaps by decorating the subject line. Needless to say, this is hopelessly error prone *and* unlikely to be effective in practice. Another possibility is to use a rule-based approach, but this requires generating a suitable set of rules for classifying messages. Anyone who has wrestled with rules for spam filtering is probably aware of the difficulty of doing this effectively. Worse, the design of rules usually comes about through experience, but there is relatively little time to gain experience with a set of messages in this domain.

The problem reduces to one of artificial intelligence. The machine learning community uses Bayesian [13] and other approaches in place or in addition to rule-based learning for classification. Some authors have, for instance, recently begun to use this corpus to classify messages as spam [16]; Androutsopoulos, et al. analyze this approach [1] and Graham reports success (with some interesting observations) with it [7]. In turn, we can ask of each message and each paper, "Relative to this paper, is this message spam?" The success of this approach depends on having a corpus for identification, and in this domain, the corpus should be effective from the very beginning. I think the title, author list, abstract, keywords and paper number would be a sufficient positive corpus for this purpose. There are some subtleties: for instance, if multiple papers have authors of the same name, we'd need a way to make that name insignificant for the purpose of classification. Nevertheless, this seems like a promising direction to explore: It will permit PC members to adhere to a comfortable interface, it requires relatively little manual setup or intervention, and it should make discussion and debate more effective by providing useful archives at no human cost.

[7] This raises an interesting privacy-related verification question about ensuring PC members do not inadvertently see discussion about their submission amidst messages about related ones. The odds of this increase since, given their expertise, they are likely to be judging related submissions. The Web, and passwords, offer a privacy mechanism; this section suggests a way of implementing the corresponding policy.

5 Conclusion

CONTINUE appears to be successful. It has twice supported a conference of non-trivial size with virtually no errors. It is in its "second system" phase, which has made its design more extensible. It appears to present a credible alternative to commercial and other conference paper management systems.

CONTINUE depends heavily on the PLT Scheme [3] implementation. The (live) experiments indicate that PLT Scheme can handle these workloads. The language and implementation offer very high cross-platform portability, so PC chairs are not bound to specific platforms (and can even more the sever across platforms amidst the process). In addition, the PLT Scheme Web server offers a convenient interface for writing interactive programs, and its primitives make it easier to construct interesting security and privacy protocols.

This experience contains a broader message for declarative programmers: Paul Graham is right [6]. If you can build your software as a Web application, you have the flexibility of choosing a better programming language. The Web largely frees programmers from the constraints of platform independence, ease of interface generation, ease of upgrading[8], absence of complex installation problems, absence of library dependencies .... It is possible to oversell the case, but there's no doubt that the Web does give programmers new powers. (Indeed, Graham and his partners illustrated this through Viaweb/Yahoo! Store [4], and ITA Software is doing so through Orbitz [5].)

Program declaratively, and seek the comforts that the Web affords. If you write a useful enough application, make it stable and evolve it rapidly, eventually others may want to know how you produced it.

Acknowledgments

I thank C.R. Ramakrishnan and Gopal Gupta for supporting my proposal to run PADL 2002 on a server that I hadn't yet written. Thanks to Gopal, Phil Wadler and Veronica Dahl for permitting the experiment to continue in PADL 2003. All of them, Phil in particular, suggested numerous improvements. The authors and reviewers of PADL 2002 and 2003 bore CONTINUE with remarkably good cheer. Thanks also to the other workshops that have permitted its use.

Paul Graunke was of immense help by maintaining the PLT Web server and assisting with PADL 2002. David Tucker similarly helped me with PADL 2003. Oscar Nierstrasz provided helpful encouragement as we devised a visual scheme for identifying champions. Finally, special thanks to the PLT Scheme team, especially Matthew Flatt, Robby Findler and Matthias Felleisen, whose credo of testing languages through applications pervades this work.

[8] Indeed, the server was constantly in development during the review period. I did some of the development while in Edinburgh for a conference. Thanks to the Web, changes were easy to prototype, test and install from a (considerable) distance.

References

1. Androutsopoulos, I., J. Koutsias, K. Chandrinos, G. Paliouras and C. Spyropoulos. An evaluation of naive Bayesian anti-spam filtering, 2000.
2. Clarke, E., E. Emerson and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
3. Findler, R. B., J. Clements, C. Flanagan, M. Flatt, S. Krishnamurthi, P. Steckler and M. Felleisen. DrScheme: A programming environment for Scheme. *Journal of Functional Programming*, 12(2):159–182, 2002.
4. Graham, P. Beating the averages, April 2001.
`http://www.paulgraham.com/avg.html`.
5. Graham, P. Carl de Marcken: Inside Orbitz, January 2001.
`http://www.paulgraham.com/carl.html`.
6. Graham, P. The other road ahead, September 2001.
`http://www.paulgraham.com/carl.html`.
7. Graham, P. A plan for spam, August 2002.
`http://www.paulgraham.com/spam.html`.
8. Graunke, P. T., R. B. Findler, S. Krishnamurthi and M. Felleisen. Automatically restructuring programs for the Web. In *IEEE International Symposium on Automated Software Engineering*, pages 211–222, November 2001.
9. Graunke, P. T., S. Krishnamurthi, S. van der Hoeven and M. Felleisen. Programming the Web with high-level programming languages. In *European Symposium on Programming*, pages 122–136, April 2001.
10. Hieb, R., R. K. Dybvig and C. Bruggeman. Representing control in the presence of first-class continuations. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 1990.
11. Hughes, J. Generalising monads to arrows. *Science of Computer Programming*, 37(1–3):67–111, May 2000.
12. Jackson, D. Alloy: a lightweight object modelling notation. Technical Report 797, MIT Laboratory for Computer Science, Feburary 2000.
13. Lewis, D. D. Naive (Bayes) at forty: The independence assumption in information retrieval. In Nédellec, C. and C. Rouveirol, editors, *Proceedings of the European Conference on Machine Learning*, pages 4–15. Springer-Verlag, 1998.
14. Nierstrasz, O. Identify the champion. In Harrison, N., B. Foote and H. Rohnert, editors, *Pattern Languages of Program Design*, volume 4, pages 539–556. Addison-Wesley, 2000.
15. Queinnec, C. The influence of browsers on evaluators or, continuations to program web servers. In *ACM SIGPLAN International Conference on Functional Programming*, 2000.
16. Sahami, M., S. Dumais, D. Heckerman and E. Horvitz. A Bayesian approach to filtering junk email. In *AAAI Workshop on Learning for Text Categorization*, July 1998.
17. Spolsky, J. User interface design for programmers.
`http://www.joelonsoftware.com/uibook/fog0000000249.html`.
18. Steckler, P. SrPersist. `http://www.plt-scheme.org/software/srpersist/`.
19. Welsh, N., F. Solsona and I. Glover. SchemeUnit and SchemeQL: Two little languages. In *Scheme and Functional Programming*, 2002.