

## THE POSSIBLE-WORLD WIDE WEB

*Bill Wadge*

Computer Science Department  
University of Victoria  
PO Box 3055  
Victoria BC Canada V8W 2Y2  
wwadge@engr.uvic.ca

*Alan Yoder*

Dept. of Computer Science and Engineering  
384 Fitzpatrick  
Notre Dame, IN 46556  
agy@cse.nd.edu

### ABSTRACT

We argue that the World Wide Web is a large scale example of indexical programming; in particular, it is (or should be regarded as) a real-time demand-driven tagged dataflow system. We sketch an indexical semantics for the Web and suggest some approaches to caching based on previous indexical work in real-time computing.

### IP ON YOUR DESKTOP.

The World-Wide Web is the first large-scale experiment in Intensional Programming and Education. We in the IP community can learn a great deal from this experiment, and hopefully make some useful contributions as well. In this article we begin to educe some of these lessons, and sketch some modest and preliminary contributions.

### POSSIBLE WURLDS

The most basic semantic concept of IP is the *intension* - an entity which varies over a space of indices (also called possible worlds). Mathematically, an intension is formalized as a function which assigns to each index its value (its *extension* at that world). In other words, an intension is an *indexed family* of data values.

It should be clear then, that the Web is *exactly* an intension. The Web is nothing more or less than an indexed family of pages, the indices being the URLs. (URL's are required by the protocol to be unique, a necessary condition for indexing).

Education is tagged, demand driven dataflow. In the education model, an intension is computed lazily. Each demand is tagged by an index, and is answered by the extension at that index. In the education model, demands and results flow as packets in an asynchronous network.

Clearly, the Web model is education - a page does not appear until a demand is generated (by clicking a link), and the demand and the response are both tagged with a URL. The Web itself is a giant dataflow net using a tagged packet-switching model.

This network, however, differs from the asynchronous networks assumed by distributed computing theorists in that it makes assumptions about global clock synchronization. Formally, we say that an event which occurs at time  $t$  at processor P1 is guaranteed to have occurred in the interval  $[t-\omega, t+\omega]$ , where  $\omega$  is the maximum error allowed by the system between any two clocks in the system. This assumption is easily justified in today's internetworked world, as is the assumption that URL's (and the internet ID's from which they are derived) are unique. When these assumptions are violated, things break, and people become motivated to fix them.

Finally, the basic feature of intensional programming is the intensional (or context-switching) operator. Clearly, each link to a particular page corresponds to an intensional operator which switches the context to the URL of that page.

## CACHE AS CACHE CAN

Now that we have placed the Web in the IP family, we can attack some of the problems with the Web from an IP point of view, and perhaps adapt existing solutions to analogous problems in other IP systems.

The most pressing problem is probably that of reducing the sheer volume of Web traffic on the internet. Much of it is clearly unnecessary because it involves sending the exact same pages over and over, often to the same destinations. The obvious solution is caching.

File-level caching can be used to greatly cut down on off-campus network accesses for large sites. In current implementations (the CERN http server in particular) this is done through the use of proxy servers, which serve the dual functions of caching and allowing Web access to clients on sites with firewalls.

The introduction of caching of course raises the usual cache consistency concerns. Suppose an interaction between client C and server S through proxy P. If the file changes on S, there is no way of notifying P so it can refresh its cache, because servers on the Web are stateless.

The problem here is only partly technological; it is also semantic. In what sense is a caching scheme *correct*? Or, more to the point, in what sense are the obvious caching schemes *incorrect*?

## THE WEB STREAM

The crucial word in the section above is "change": if the pages never changed, there would be no problem.

This suggests that the semantic approach to the Web defined in the first section must be refined if we want to develop and evaluate caching strategies. The simple approach defines the Web as a family indexed by URL alone; given a URL  $u$ , we have the page  $P(u)$ .



The correctness condition is the following: if we issue a request for the page at URL, we must eventually be provided with  $P(u)$ . Caching is transparent because it is of no importance whether or not our copy of  $P(u)$  originated with the server for  $u$  or with a proxy. We only have to ensure that the cache labels its contents correctly, ie that the page stored in the cache and labelled  $u$  really is  $P(u)$ .

In fact, the Web evolves, and we can formalize this (using a well-known IP technique) by adding a time parameter. Then given a URL  $u$  and a time  $t$ , we have a page  $P(u, t)$ .

It is not so easy, however, to see what the new correctness condition should be. If we issue a request at time  $t_0$  for the page at  $u$ , what should we expect in return?  $P(u, t_0)$ ? This is impractical, because the requests take time to reach the relevant server, and during this time the page may change. It seems that the ad hoc implementation gives you  $P(u, t)$  for some  $t$  between  $t_0$  and the time  $t_1$  at which the response arrives.

This correctness criterion is pretty loose because it doesn't even guarantee that two pages fetched (even at the same time) from the same server are consistent. At time  $t_0$  you can issue requests for the pages at  $u_0$  and  $u_1$ , and at time  $t_1$  receive pages  $p_0$  and  $p_1$ . Yet you cannot assume that  $p_0$  and  $p_1$  are of the form  $P(u_0, t)$  and  $P(u_1, t)$  for a single  $t$ .

Even this loose correctness criterion breaks down with naive caching, because a request can be interrupted by a proxy with a stale cache. All you now then is that a request issued at time  $t_0$  and answered at time  $t_1$  is the value  $P(u, t)$  for some  $t$  less than  $t_1$  - not much of an achievement, since this condition is guaranteed by causality!

## CURRENT SOLUTIONS

Web technicians are aware of these problems (at least informally) and have already developed some initial and pragmatic solutions.

These solutions include the following measures: 1) when a document is originally received from a server, an expiry date may be given in the header, 2) a client can require a proxy to check to see if the data in its cache (corresponding to a given request) is current, and 3) proxies may periodically refresh documents in their caches during periods of inactivity.

Making sure files are not cached when they must not be is handled differently depending on whether the client or server has the need. A client which displays current weather information and updates it every five minutes, for example, needs to be sure that its proxy does not return stale data from the request five minutes ago. Some servers, on the other hand, reuse the same documents over and over (building them on the fly) as responses to query requests. These documents should rarely be cached (identical queries might be an exception); because proxies are stateless we therefore must never cache them. Existing Web protocols, while not yet universally used, allow these behaviors as follows.

If the client  $C$  wants to be sure to get the most current version of a file, it sends its request to the proxy  $P$  using a conditional GET request (with an "if-modified-since" field in the header).  $P$  then forwards this message to the server  $S$ , which returns either



the whole file, if necessary, or just a "not-modified" header to P. (This optimization does away both with the necessity of setting up and tearing down two connections, which would be necessary in a query-and-fetch scheme, and with the unnecessary consumption of bandwidth if the whole file were always returned).

If the server, on the other hand, needs to be sure the client always comes back for the next version of the file, it does so by attaching an expiry field to the file header which is equal to the current date and time.

### BUT WHAT DO THEY SOLVE?

These ad hoc methods accomplish something, but it is hard to say precisely what.

The handling of an unconditional request still *guarantees* nothing not ruled out by causality.

The handling of unconditional requests guarantees only that the requester receives the page as 'published' by the server at some point after request was issued .. and that is assuming the servers respect their expiry dates.

It seems more appropriate to evaluate these schemes as heuristics, ie quantitatively. Roughly speaking, they make it more likely that a request issued at time  $t_0$  and satisfied at time  $t_1$  is satisfied with  $P(u, t)$  for some  $t$  between  $t_0$  and  $t_1$ . In other words they improve performance and increase the likelihood of satisfying the (loose) correctness criterion.

Obviously, there is room for improvement.

### STAMP OUT INCONSISTENCY

Those readers familiar with IP and some recent ISLIPs will recognize that these problems have, to a certain extent, already been dealt with.

If we take into account the changing nature of the Web, it is clear that the Web is an example of *real-time* dataflow - dataflow in which the relative arrival times of datons (and not just their stream indices) are semantically significant [FL89,P91].

The crucial idea, already being adopted in practice (as we have seen), is to use *timestamps* on both requests and responses (this is not the same as adding extra time dimensions). In the context of the Web, it seems that the appropriate idea is to use Faustini's original approach of having *intervals* as time stamps.

The idea is that a page, whether in transit or in storage, is stamped with two times,  $t_a$  and  $t_b$ . The second stamp,  $t_b$ , is basically the expiry date above; the first is its dual, the date at which the information became (becomes) valid. The consistency condition is this: if a page Q is tagged with a URL  $u$  and stamped with  $t_a$  and  $t_b$ , then Q must be equal to  $P(u, t)$  for *all*  $t$  in the interval  $[t_a, t_b]$ . (We will simplify the discussion by assuming the clock skew is negligible.)

In order for this model to fit with existing protocols, we have to assume that servers "tell the truth" when they attach expiry dates to responses. In particular, an expiry date which is later than the time the response is shipped has to be assumed to be a lower

bound on the latest timestamp at which the file given in the response will be consistent. This model therefore does not tolerate byzantine failures.

If the answers to requests are timestamped, what about the requests themselves? Initially, we assumed that a request should carry its time of issue. This convention, however, is questionable. Different requests will, inevitably, be issued at different times, so there is no guarantee the answers will form a consistent picture. Also, we might be interested in a page at a specific earlier time: the time at which the current session began, or 8am this morning, or (eg with newspapers) the page from last month.

The most general solution is to label a request with an *inquiry* time  $t_i$ , the time of interest, usually (but not always) close to the time at which the request originates.

It is also in the spirit of real-time computing that requests carry an expiry time  $t_e$ , after which the request can be considered as cancelled (the expiry time might usually be the issuing time plus a small interval).

## WHAT WE WANT FROM THE WEB

Therefore, given a request for the page with URL  $u$  with inquiry time  $t_i$  and expiry time  $t_e$ , what do we require of the Web?

The first and most basic correctness condition is as follows: a request for the page at  $u$  must produce a response whose tags are valid in the above sense.

Even this simple condition can give us assurance about consistency between pages. Suppose we have pages  $p_0, p_1, p_2$ , etc, tagged by corresponding intervals. If the *intersection* of the intervals is nonempty, then we know that these pages are of the form  $P(u_0, t), P(u_1, t), P(u_2, t) \dots$  for some *single*  $t$  .. in fact, for *any*  $t$  in the intersection of the intervals.

The next condition is that a request with inquiry time  $t_i$  must produce a response for which  $t_a \leq t_i \leq t_b$ . This corresponds to requiring that we receive the page as it was at the time of inquiry.

The third condition is that the response arrives before  $t_e$ . This says that the answer arrives *on time*.

Other conditions are possible, as well as other requests - for example, we could replace  $t_i$  by an interval, and interpret it as a request for a copy of the page as it was at *some* point in the interval. And some tricky implementation issues arise. But we can use indexical semantics and our experience with realtime IP to formulate these conditions and request modes, and to evaluate them from the user's and the implementor's point of view.

## REFERENCES

- [B95] Tim Berners-Lee et al, CERN Server User Guide, available at <http://www.w3.org/hypertext/WWW/Daemon/User/Guide.html>
- [FL89] A. Faustini and E. Lewis, Toward a Real-time Dataflow Language, in J. Stankovic and K. Ramamrithan, editors, Hard Real Time Systems, pp139-145, IEEE,



1989.

[N93] Roger Needham. Cryptography and Secure Channels. In Distributed Systems, Sape Mullender, ed. Addison-Wesley, 1993.

[P91] John Plaice, Some Thoughts on Dataflow Real-time Programming Languages, Proceedings ISLIP 91, pp70-79, SRI International, Menlo Park, 1991.

## APPENDIX: RELAXING THE GLOBAL CLOCK SYNCHRONIZATION REQUIREMENT

It might be asked, are any consistency constraints possible in the *absence* of global clock information? It turns out that only so much can be done, and one must accept a "relativistic" universe. What this ends up meaning is that any given set of queries must always use timestamps which refer to the *local* time at the server. For example, a client which requests a page of stock market prices as of 14:10 GMT must take it on faith that the server's clock is accurate. This is usually not a problem, since servers which deliver this kind of data are generally trusted in other ways as well.

For ordinary synchronization purposes, the protocols work as follows. There are four cases, involving interaction between a client C and a server S. Two of them can be accomplished within the protocol:

- 1) C has a page  $P(u, t)$  from S and wants a page  $P(u, t+dt)$ .
- 2) C has a page  $P(u, t)$  with a timestamp  $(t_a, t_b)$ , and wants a page  $P(u, t_b+dt)$ .

Two others cannot:

- 3) C has a page  $P(u, t)$  from S and wants a page  $P(u, t'), t' < t$ .
- 4) C has a page  $P(u, t)$  with an expiry time of  $t_e$ , and needs to know whether P has expired.

Cases 1 and 2 are easily accomplished, because all times are expressed in terms local to the server, so there is no synchronization required. Case three cannot be accommodated, because  $t'$  is expressed in the local time of the client, which requires clock synchronization (if it is expressed in the local time of the server, the problem is trivial to convert to the first case).

The fourth case similarly cannot be decided without synchronization, because the client doesn't know what time it is on the server, and so can't decide when time  $t_e$  has arrived. However, a conditional GET request *can* be used to do the synchronization; if the page has expired, the server will send a copy of the new page, and if it has not, it will say so. This reduces the expiry time to a kind of advisory, which could be used in caching proxy servers as follows (but currently is not). Let the expiry time represent a fallback capability; if the server cannot be contacted to determine expiry, the expiry

date can be used with reference to local clocks to determine whether to continue, or whether to abort the given session.

In summary, the only semantics which are impossible without the global clock assumption involve a desire to specify times in terms of the local time at the client. Unfortunately, as the World Wide Web evolves, this is likely to impact a large class of problems, because almost all computers could conceivably become both client and server, and interact in arbitrary ways. The assumption of global clock consistency seems therefore to be both necessary and desirable.

In practice, there are well-known ways of achieving close (but not exact) clock synchronization between two processors. This is usually done by referring to a trusted third party (the authentication server in the Kerberos protocol for example [N93]), then advancing or retarding clocks by two seconds every minute until synchronization is achieved (the smallstep approach is necessary so that programs such as *make*, which compare timestamps, don't get confused).