# Dagstuhl Seminar 07051
# Programming Paradigms for the Web: Web Programming and Web Services
# 28.01.2007–02.02.2007
# Working Group Outcomes

Rick Hull[1], Peter Thiemann[2], and Philip Wadler[3] (editors)

[1] Bell Labs, Lucent Technologies
600 Mountain Ave., 2D-510, Murray Hill, NJ 07974, USA
hull@alcatel-lucent.com
[2] Albert-Ludwigs-Universität Freiburg, Institut für Informatik
Georges-Köhler-Allee 079, 79110 Freiburg, Germany
thiemann@informatik.uni-freiburg.de
[3] University of Edinburgh, Department of Computer Science
James Clerk Maxwell Building, The King's Buildings, Mayfield Road, Edinburgh
EH9 3JZ, Scotland
wadler@inf.ed.ac.uk

**Abstract.** Participants in the seminar broke into groups on "Patterns and Paradigms" for web programming, "Web Services," "Data on the Web," "Software Engineering" and "Security." Here we give the raw notes recorded during these sessions.

**Keywords.** Web programming, web services, programming paradigms, analysis and verification, implementation techniques and optimizations

## 1 Overview

During the initial plenary session, the attendees as a group created a list of topics, which became working groups. These included "patterns and paradigms," "web services" "data on the web," "software engineering," and "security."

"Patterns and Paradigms" concerned familiar problems of web programming and their solutions (patterns), as well as whole solution styles (paradigms). This group was large enough to break into several sub-sections which each took a different path.

The "Web Services" group looked into the problems that arise when programming web services, that is, programs that talk to other programs over the web (as opposed to users). This included issues like service discovery and composition.

"Data on the Web" asked whether data should be organized in new ways for web programming, and whether databases should take on a new form to better support it.

Dagstuhl Seminar Proceedings 07051
Programming Paradigms for the Web: Web Programming and Web Services
http://drops.dagstuhl.de/opus/volltexte/2007/1127

"Software Engineering" examined the process of creating software, and the special difficulties that arise in programming web software—for example, the novel interaction between programmers and web designers, or the demands placed on software engineering through the always-on nature of websites.

The "Security" group, last but certainly not least, examined new security challenges that have cropped up with the web, which becomes increasingly important as we become more dependent on web browsers in our day-to-day use of computers.

Each attendee participated in more than one working group, and each discussion was captured on the seminar's wiki or on paper. This document contains the raw records of these discussions.

These discussions incorporate the active participation of all the attendees: Serge Abiteboul, Nick Benton, Ezra Cooper, Daniel Deutch, Susan Eisenbach, John H. Field, Christophe Fouqueré, Alain Frisch, Martin Gasbichler, Michael Gruninger, Haruo Hosoya, Richard Hull, Dean Jacobs, Trevor Jim, Shriram Krishnamurthi, Niels Lohmann, Florian Loitsch, Bogdan-Eugen Marinoiu, Florian Matthes, Jay McCarthy, Tova Milo, Yasuhiko Minamide, Tom Murphy, Anders Møller, Matthias Neubauer, Barry Norton, Peter Patel-Schneider, Matthias Radestock, Mukund Raghavachari, Helmut Seidl, Manuel Serrano, Bertrand Souville, Jianwen Su, Peter Thiemann, Philip Wadler, Stefan Wehr and Jeremy Yallop.

## 2 Outcomes

This section collects the outcomes of the working groups. Each subsection is authored by members of the group with some editorial changes.

### 2.1 Patterns and Paradigms

Discussion participants: E. Cooper, J. Field, T. Jim, S. Krishnamurthi, J. McCarthy, M. Neubauer, J. Yallop

**What are the challenges of multi-tier programming and what are approaches to overcome these?**

- challenges of multi-tier/locale programming
  - keeping browser and server in sync
  - keeping schema and program in sync
  - keeping application logic and database in sync
  - exacerbated by scalability, security aspects
- tierless programming languages and paradigms
  - HOP (client/server)
  - Links (client/server, database)
  - ML5 (generalized client/server)
  - Reactors/Collage (databases, transactions, client/server)

- tiered approaches
  - Servlet analysis [Møller et al]
  - introspective Framework [Matthes et al]
  - Flapjax [Krishnamurthi et al]

**Discussion Notes**

*Tiered/tierless discussion points*

- should there be a dichotomy at all?
- web service (broadly defined... lots of autonomous entities on the net) interaction necessarily introduces "tier" distinctions
- explicitly distinguished servers vs. implicit "theServer" in standard client-server architecture
- one distinction: explicitly distributed programming model vs. virtual single function/process/program
- introduce tier distinctions as a separate annotations/advice
- resolved: tiered/tierless distinction is not very important... just a question of style
- dimensions:
  - primitives for communication?
  - is there a distinguished/implicit server?
  - do some nodes have distinguished capabilities (e.g., only broker can talk to external web services)
  - how is reactivity managed?
    - explicit language support
    - libraries
    - patterns (e.g., callbacks)

*Functional-reactive programming a solution?*

*Web games as challenge problem*

- hard to do using standard web components
- synchronizing multiple browsers difficult
- scaling/performance issues in the presence of shared state

*Long-running transactions are difficult*

- has this been solved outside the web?
- even harder to do with standard web protocols
- how do you support this pattern?
- maybe it's always ad-hoc?

*User management / identity / login / access control*

- very common pattern; many languages provide no support
- support for "roles"/policy?
- session management closely related

*Page flow management*

- is this "inside" the web language or outside?
- how to translate designer's page flow design into code?
- special support for MVC pattern?
- should we have special abstractions?
- is the whole idea of "page flow" becoming dated?
- want to make "logical pages" linkable (i.e., they need a URL)
- this may be partly a matter of proper software architecture
- managing browser cloning, back button, etc.

*Dealing with untrustworthy client, network, ...*

*How to program crypto protocols on the web?* (cross-cutting concern)

- can't assume that they're baked into the infrastructure
- not a good idea to bake all protocols into the infrastructure
- HTTPS doesn't do it all, e.g., non-repudiation

*Flickr as challenge problem* ("rich" GUIs as challenge?)

- photos can be part of multiple albums
- coordinating little web widgets, animations, etc.
- how to do this stuff w/o 1000's of lines of JScript?
- what is the right GUI toolkit for the web?
- how to do fancy stuff while maintaining ability to access web in a lightweight way?
- easy stuff is easy, hard stuff practically impossible (on the web)

*Interfacing with legacy*

- JScript
- Web services
- data stores
- third-party libraries (client-side, server-side, whatever)

*Lack of good interface/contract technology for web components*

- a lot of good ideas out there, just need to get them used
- need "ontologies", but current technologies too heavyweight
- is there something lighter... like folksonomies.

*Resource management for stateful services*

- does affect web programming models
- stateless is nice. When do you release stuff that's no longer relevant? [With today's storage capacities, is it necessary to release anything?]

*"Basic" web programming benchmarks/baselines*

- wiki/blog
  - users can add content
  - versioning
- mini eCommerce (catalog order app). Not THE wine store, but something with a clean spec.
- hotel selection with appropriate user interaction pattern
  - user explores graph of options, needs to move back and forth through space without getting losing context
  - want back/clone/refresh to have sensible semantics from user perspective
- client-side mashup (e.g., Yaggle)
- draggable lists and similar UI functionality
- chat

*Performance*

- optimizations in this space are primitive or nonexistent
- want to optimize across tiers
- want implementation to reflect nonfunctional requirements
- take advantage of existing work on data synchronization,

  distributed message opt., data- and control migration
- background syncronization of client-server state

*Want challenge problem(s) to test scalability* Specifically, set of well-defined apps/input sequences

*Browser is an OS* but not a very good one at this point

- want to blur OS/PL boundary... implies PL needs more OS-like features
- need to do it right... learn from previous efforts

**What are some representative programs that highlight challenges in web-based programming (and how do various frameworks compare on these?)**

**What are the primary design patterns/idioms/mechanics of web programming and of web services, and produce a useful taxonomy/ontology?**

**Summary Group #1** *scribe Ezra Cooper*

*Two extremes of web programming:*

- Document-centric, open
  Hypertext-Web ⇒ Content Managemnt Bookmarking, Sending links to a friend, Importance of URI & Document Searchable, Accessible
- Experience-centric, closed
  Browser as Interface to a remote Application, Flash-based games & apps

*Web services and web applications*

- Agreement that apps are an instance of web services, or:
- two possible interfaces to one core.
  - User-facing ⇒ Web Applications Computer - Computer ⇒ Web Services programing
  - Two approaches: Human User first, Serve first
    User-Centered design vs. Service design
- Commonalities: Remote store
  Composabality at the page level (mashups), e.g. Flickr images, Maps,

*Patterns (and Problems)*

- URL. Encodes a view (of data) and/or some actions
- Redirect after POST so that action cannot be repeated.
- Unexpected control flow due to back-button (controlling this)
- Persistence of previous states, mutability of state
- Paging (of lists of results)
- Login screen
- General umbrellas:
- Authorization - Access Control
- (Untrusted Clients)
- Controlling machine consumers: Crawling / Robots as Roles
- Redundant validation (Client and Server)
- Incremental Checking / Form Validation (?)
- Site Navigation and Orientation (Menus, . . . )
- Page-level assembly of components (fragments)
- 
  - Portlets
  - Mashup (components from different sites)
- Notification
  - RSS-Feeds (Streams) & Ping vs. Event-Based architectures
    Processing feeds : combining filtering.
  - Notification of Clients through blocking read ⇒ "Comet" (synchronous mode of interaction)
    problem of routing information between multiple clients connected at once (technical problem: # of sockets, when to close it?)
  - Loose structures
    * Tagging vs. Ontologies
    * Queries vs. Folders

**Summary Group #2** *scribe Jay McCarthy*

*Login*

- Get login and password
- Identity management
- Unsolved problem: Many logins for closely related entities
- Difficulty: HTTP-Auth or GET/POST authentication (different security and relates to below)

*Correlating new login with running sessions/servers*

- Found in web services and web applications
- Jay doesn't like being logged out when you open new sites
  - Not like in PLT
  - More difficult in different authentication regimes
    Maybe change browsers to stop re-logging when open a new window, but this is new semantics
  - Firefox profiles are like this
  - Most businesses are not interested in this
- Concurrent sessions of some kind managed by clients
  - Some limitations on browsers and many servers don't work with it
  - Manuel [Serrano]: Starts Firefox and Mozilla when he wants this.
- Determining shared ontologies based on what both speak, so they can establish common meaning

*Service discovery*

- Like mining jungloids and type inhabitation
  But with more interesting description of what is needed than types
- Do I ask for one service or service composition?
  Dynamic composition for latter

*Conversations*

- One service looking for 3 things and another can deliver 2
- Manufacturing product: Needs plant utilization, orders, and available machines
- Calculator: Addition, Subtraction, and not Multiplication
- Only machines, rather than just clients and servers
- Rather than a programmer looking through a manual and finding the services, he specifies what is needed, rather than how to get it
- How does a conversation end?
  One ultimate initiator, with a goal that is achieved
- Like distributed transactions? and long-running transactions?
- Built on top of service discovery?
  When a service doesn't answer yes or no, but maybe or partially

*Transactions*

- [What is a long-running transaction?]
  - No hope of locking everyone else out, because it is so long
  - Must apply compensations to undo, but external observers saw inconsistencies
  - No, a better notion is: Unfair, maybe, if it is too long it will always be preempted
- Most credit card payment services are not as robust as you'd like

- Ad-hoc protocols may not be enough when you need to integrate multiple components
- BPEL extension to compensate multiple parties and receiving failure notification
- Compensations may not be exact — send a sorry card for the bomb
- Traditional ways just don't work
- One level beneath is failure discovery and propagation
  - Are these broken on the web as well?
  - Lack of connectivity is like hardware errors

*Reusing code without cutting and pasting or reading*

- Moving things from the backend to the frontend
- Interfaces and types
  - Are the types in web services enough? Or are the web applications enough?
  - Patterns of interaction, rather than just input/output types
    Like session types
  - Do we need more?
  - Types/contracts have been useful outside the Web
  - Maybe the Web is different in this way and need more
  - And with more complicated types, how do you verify that implementations meet
- Checking something discovered dynamically, must be checked with contracts
  - Or with proof carrying code—by Phil Wadler or compilers
  - You must have some trusted base
- Problem of saying interesting enough things without requiring NP-hard verification
  - The perfect is the enemy of the good
  - Confusing a product with a sum
  - Confusing an airplane's idea of customer to a train's idea of customer

*Is Web Programming fundamentally different?*

- Idiom that implies convergence: Portals—user oriented web page that integrates others—presentation oriented component integration
- Shown in a browser, but composed in a style like web services

- Web programming is providing a fronted for web services?
  Portals compose the frontends, not the services
- Where it happens — server or client — doesn't matter
  Client-side portal or Server-side mashup
- What to learn?
  - The server has finer composability
  - The client is more sexy and ad-hoc
- Why *should* they be different?

*Buying a house is complicated in the US, with manual transactions*

- So many people are involved
- People hate each other
- Misunderstandings about tiny bits of the contract
- People don't want to share some secrets, must share others, and keep them consistent
- What would stop you?
  - Web services don't handle the long running-ness
  - If something important fails it is difficult to undo
  - Fire lawyer, real-estate, etc in the middle and get another
- Could we hack it together or do we need something principled?
- Tax applications are like this
  They release new versions every year

**Summary Group #3** *scribe Shriram Krishnamurthi*

Because good software engineering practice demands that we focus on the problem space before we dive into the solution space, we organized our discussion around problems and challenges that people had identified. These issues fell into the following loose categories.

*Use Patterns*

- Web users employ a host of browser-endowed operations that confuse and complicate the flow of control in the application, which must now guard itself against these.
- The need for continuous provision of service places new demands on the structure and implementation of applications.
- The Web offers both the ability to continuously monitor what users are doing — without having to ask for their permission — while simultaneously demanding that sites do [the monitoring] to ensure a reasonable quality of service (a concern that software vendors of traditional desktop applications were previously able to push onto the end-users).

*Program Structure*

- Compared to desktop applications, Web applications have a much broader range of scopes (single-client, multi-client, browser, shared, ...) and extents (round-trip, session, eternal, ...) for data.
- The tiering of applications creates difficulties, with different languages [and data encodings] often used in different tiers.
- The fragmented structure of Web applications (broken up by response points) complicates the task of program analysis, because just reconstructing what the "program" is takes effort (and may yield an answer that is not compatible with the expectation of traditional program-analysis tools).
- While any platform may use multiple languages, this seems almost de rigeur on the Web, creating a further challenge for program analyses.

*Data Management*

- The distribution of data across autonomous and potentially uncooperative peers makes it difficult to apply many traditional database techniques.
- Query planning and related tasks must now take into account the access control restrictions of the domain.
- The lack of centralization in Web services, such as loosely-coupled B2B processes, poses a challenge for data administration.

*Security and Privacy*

- Publishing data through Web interfaces demands the securing of sensitive data.
- Centralizing data on the Web, and giving different people access to the same central data, highlights the need for care in data dissemination (access control and information flow)
- Moving data between clients and servers, or between services, demands the proper use of encryption and related techniques.
- Whereas application writers have traditionally not needed to deal with structuring programs around multiple users, [such structuring] becomes critical in a typical shared Web application.
- The various forms of patterns that sites use to identify users and data lead directly to means for violating the privacy of users; how can we have the former without the latter?

*Scalability*

- Scalability must now be performed across a variety of potentially uncooperative and often non-homogenous machines.
- The scalability techniques available to Web applications, such as router hardware support for dispatching to servers on the basis of cookie information, has no analogue in Web services.

### 2.2 Web Services

**Two very different approaches to typing arise from logic: ontologies and description logic on the one hand and types as found in programming languages on the other. How do these compare and contrast?**

- WSDL-S allows to attach rich information concerning the types of input/output arguments of web services. In particular, one can link to OWL-DL and use a description-logic-based specification, and/or one can use a conventional rich typing mechanism.
- The Curry-Howard correspondence or propositions-as-types relates logic to types and proofs to programs.

**How do we describe the effects of web services in a way that supports reasoning and querying? How do the approaches used in the semantic web community compare with the type and effect systems used in the programming languages community?**

- OWL-S, DAML-S
- effect systems in programming languages (Gifford and Lucassen; Wadler and Thiemann)

**What forms of service discovery, composition, and analysis might we expect to realize in the short, medium, and long term?**

- What are useful approaches to expose, reason about, and program with [data, info, content] in the context of web (services) programming?
- Is the problem of "automated composition/synthesis of services" discussed by the Semantic Web Services community handled in some way by techniques from other fields (e.g., type theory)?
- Is there a relationship between the business model and composition? (Interest in automatic service composition might be less because it removes opportunities for selling ads.)
- BP-QL

### 2.3 Data on the Web

*edited by Serge Abiteboul, Rick Hull, Dean Jacobs*

**Why is managing data on the web fundamentally different than in other distributed database settings?** While accessing and managing the data on the web can take advantage of much of the standard database management techniques, the workshop participants identified several areas where new techniques are emerging or are needed. Key areas are reflected in the questions below. We focus here on database reads. Database updates are usually controlled by application-specific web services, and these side-effecting services have been discussed in the preceding section.

At a fundamental level, the challenges of data management on the web stem from loose coupling of services, their autonomous creation and management, the emergence of community-created data sets, and the predominance of streaming rather more static data.

**What are the challenges of managing and working with meta-data in the web context?** In a local database, and in a centrally managed distributed database, you know the schema, you know its semantics and you can easily write applications on it. On the Web, you possibly just discovered the schema and it may not even be what you expected or wanted to see. Indeed, for many large bodies of data on the web today, such as user-generated photos or videos,

there is essentially no database schema, and tags are used to find objects of interest. Furthermore, these tags are often generated by users through various community mechanisms. This contrasts with the conventional approaches to data management, which rely on a database schema or ontology, as in the area of the Semantic Web. Going forward, it will be important to develop linkages between these two approaches to describing and accessing web data.

For cases where database schemas are available, the data that you are interested in may come from (a possibly large number of) heterogeneous data sources typically as Web services. This is branching to all the work on (semantic) integration and to the need for also exchanging information about resources. An immediate challenge is to provide a unifying database schema for the data of interest to a user or application, along with mappings that relate the data offered by the individual services to the unifying schema. Existing techniques include "Global as View", the "Local as View" approach pioneered in [**?**], and hybrids of these. For example, [**?**] develops an approach based on the use of the Entity Relationship model for representing the global schema and description logic based reasoning have been developed. Ontologies, or techniques such as Data Exchange [**?**] additionally provide mechanisms to incorporate relationships between data sources. A key challenge concerns how robust the above frameworks are in the context of loose coupling, data source evolution, and inexact correspondences between the schemas of the data sources. Querying across the union of multiple unifying schemas is also a challenge. There is also a sociological challenge, namely, it is not clear what will motivate communities to invest the resources to develop and maintain unifying schemas and their associated mappings. One approach that may help overcome this problem is provided by the ActiveXML framework [**?**]. An ActiveXML source essentially provides an XML schema, but some nodes of the XML document may be pointers to remote ActiveXML sources. This approach permits a more "organic" or distributed approach to the creation of unifying schemas.

**What makes data on the web different than traditional data?** Streams are at the core of Web data management. They arise naturally in many places, e.g., subscriptions (cf. RSS), sensors, and interaction with browsers. Furthermore, when query processing large quantities of data it is sometimes convenient to stream the answers, because of both the transfer time latency (especially if to a browser) and the possibly cyclic nature of how the data sources refer to each other. Recent work in the database community on streaming data is relevant here, but has not yet been focused on challenges specific to the web, such as the need to support subscriptions against an integrated view of multiple streams.

Change control also takes a very different flavor. Standard transaction techniques developed for distributed database systems are typically not adapted, because of the autonomy of sources and possible transfer time latencies. There is more emphasis on aspects such as reconciliation, asynchronicity, pub/sub mechanisms, and broadcasting.

Finally, scaling comes with a new bias in some applications, not because of the size of data or the rate of queries (that is standard in databases) but in the number of peers that are possibly involved, thousands or more. An important aspect here is distributed optimization in the context of loosely coupled sources. For example, query plans may be distributed, but challenges arise in communicating plans between the sources and enabling cooperation between them (e.g., for tuning, healing, statistics gathering, etc.)

**What can be done about the "impedance mismatches" for data on the Web?** The "impedance mismatch" between programming languages and databases has been studied for many years. The problem is that data types in programming languages differ from data types in databases, thus type conversions must be performed at the boundary between the two. Such type conversions make programming more difficult and more error prone, and make program execution less efficient. Techniques for automatic type conversion, such as Object/Relational mapping, reduce but do not eliminate the programming complexity. More comprehensive solutions include pushing programming language types down into the database, e.g. in persistent programming languages, and pulling database types up into the programming language. Neither of these approaches has achieved wide-spread acceptance.

The introduction of loosely-coupled programming over the web has exacerbated the problem by introducing a second type conversion boundary. Data traversing the web using Web Services protocols is represented as XML. When it enters a program for processing on an application server, it must be converted to programming language types, a process referred to as data binding. Again, automatic type conversion can reduce but not eliminate the programming complexity. Moreover, it can result in a loss of information, as not all aspects of an XML document can be captured. At a fundamental level, XML is becoming the intended form for data rather than just a form for its transport.

Researchers have been pursing the expected comprehensive solution: elevating XML to being a first-class data type in programming languages and databases (e.g., [**?**] was presented at the workshop). This approach is likely to achieve acceptance for applications that are fundamentally transport-oriented, however it may not be universal. Computationally-intensive applications will continue to bind to programming language types and query-intensive operations will continue to bind to database types.

### 2.4 Software Engineering

*edited by Susan Eisenbach*

One key issue in software engineering is the interaction between social and technological problems.

**Are there problems in software engineering and software lifecycle that are somewhat unique to the web?** With regards to trouble shooting, debugging, and maintaining operational web applications (24x7, high volume), are

there aspects that make this different than in more conventional kinds of applications?

**How can we foster separation of concerns between web designers and web programmers?**

- Contracts for Cooperation (Moeller and Schwartzbach)
- Templates (as in Florian Matthes' talk)
- real practical problem (Matthias Radestock)

**Even assuming that the models and tools for producing code, data, processes are sensible, what are the non-functional factors that can make a website fail?**

- scalability & performance
- usability
- evolution and deployment
- availability 24/7
- adding value through customization
- translation

**Open software engineering research questions**

- How do you do testing of websites? (Some tools available, e.g., Selenium.)
- How do you encapsulate the different concerns of a website?
- How do we evolve websites continuously, distributed and consistently?

**Are there Software Engineering concerns that should be taken into consideration when designing programming models and languages?** Having good technology to produce a website is desirable, but not sufficient to guarantee that the website will be successful. There are many problems that arise when people use websites. Some of these occur with any large distributed program, but there are problems that are primarily or completely web based. For example, there are some ways to test websites but it is clear that many are going live without thorough testing.

One of the promises of the web is that it is usable by the untrained user $24 \times 7$. This puts constraints on the user interface that non-web applications do not have. Programmers as a group may have difficulties empathizing with technologically challenged website users. The $24 \times 7$ nature of the web means that a poorly designed website, if used at all, will need serious support. One of the ways to get new users confident in using technology is to customize software so that it is very simple and then add the functionality as the users become familiar with the system. The idea of not giving users the full functionality of the product and then revealing functionality is often surprising to developers.

The web development world uses designers, but this in itself causes difficulties as abstraction skills are very different between programmers and designers. Can we characterize a common basis of understanding for both designers and programmers? Can we find a model that could present both the designer's and the programmer's view? As an example, there seems to be no way in today's tools for a designer to say there should be alternating colors for the items of a list. This kind of problem is not unique to web programming. How does the game programming world solve this problem? Research languages are being designed without thought to usability and how designers can be part of the development process. See the talk by Florian Matthes http://kathrin.dagstuhl.de/files/Materials/07/07051/07051.MatthesFlorian1.Slides.pdf

The massive growth in computer users because of the web makes performance and scalability a first class problem on many sites. If a site does not have a fast response time and there is any competitor people will seek out the competition. We don't yet have cost effective solutions that would enable all websites to have reasonable performance.

One of the pressures on websites is that they must always provide new functionality and today's look. Hence, web programmers must be able to update live code and persistent data, not an easy task. If the site is large and there is also distributed development then there is a serious problem of maintaining a running, consistent site. Serious work needs to be done on evolution and deployment.

### 2.5 Security

**What is wrong with JavaScript and browser security mechanisms?**

- The same-domain restriction prevents a script in a web page from accessing data from pages from other web sites (domains). E.g., a script in a page from skeletor.evil cannot access data from a page from bank.com, when both pages are loaded in a browser.
  - Good: this prevents Skeletor from stealing banking information.
  - Bad: this prevents mashups.
    - People are trying to evade the same-domain restriction by using Flash-JavaScript bridges. Flash has different, and, in some ways, less onerous restrictions.
    - People also get around this with proxies, e.g., Yaggle (the Yahoo/Google Maps mashup of Shriram et al.). A problem is that the proxy might require your credentials, so it must be highly trusted.
- Shriram Krishnamurthi points out that there is insufficent isolation of tabs for different sites in the same browser (one tab takes up all cycles and the others starve). This may be improved in new browsers. Unfortunately this means that concerns that are typical for OSes start to become concerns of the browser.

**What changes to JavaScript and browsers could improve security?**

- We would like hooks to be built into the browser and ?JavaScript interpreter that would allow us to intercept certain actions before they occur.
  - This was used in Trevor Jim's talk to prevent script injection, via whitelist and other policies.
  - We might be able to use this to implement a more fine-grained security model, e.g., something like Java's. Jay mentioned running an instrumented interpreter written in JavaScript; Trevor thinks this has already been done.
- Trevor Jim and Jay McCarthy both suggested having more efficient crypto libraries exposed. This would allow us to program up new security protocols and primitives directly in JavaScript without paying a huge performance penalty.
  - Some of Trevor Jim's policies to prevent script injection would get better performance this way (whitelists).
  - Jay McCarthy had a couple of examples: a bank transfer purchase order often requires authentication beyond that given by https; and multi-party protocols, where https is clearly not sufficient.
- Tainting (dynamic tracking of information flow) could be useful. Note, this used to be in JavaScript, but was removed.

**How can we formalize script injection/cross-site scripting and prove that a web application is not vulnerable to it?** We considered this question but do not have a final answer. It may be possible to formulate a very general and abstract notion, but, in the end, what you want is going to involve a semantics for a web application framework and a proof that no app written in the framework can exhibit a certain behavior (producing an injected script).

**What kinds of languages and language features can help us write more secure web applications?**

- Systems that act as web application "compilers" can "compile in" security. By web application compiler we mean a system in which you write a program and which transforms that program into a client-side and server-side program.
  - Trevor's suggestion is a security monitor that operates at the web API (inserting checks at each http message). The enforced policies could be derived from a static analysis of the program and could include control-flow, possible values of parameters, etc. This can foil some script injection attacks as well as some other attacks.
  - Good: tool support could be a uniform solution that can be applied uniformly.
  - Good: static analysis may be enhanced: rather than trying to analyze the client- and server-side separately, you have a single, uniform analysis.
  - Bad: this may hide details of the compilation target and these could be exploited.

  - Anders Møller gave a talk about JSPs generating XML via printfs, and checking the output. This means that a web applicatoin compiler may not be necessary for automated hardening.
- Information flow systems are clearly useful, e.g., the work of Myers et al. It is notable that none of the tool builders here is doing this so far.
- Denial of service protections. Systems could build in monitoring that attempts to ensure that the web application is only used by humans, e.g., prevent 100 clicks per second in a session. Have compilation flags like `-Ohumans` and `-Orobots`. (Like most denial of service protections this would be far from perfect.)
  - This sort of protection could have made things harder for the Samy/MySpace worm.
- High-level languages are easier to reason about in general, but beware that the actual program running is at the low-level and may have different behavior.
- Static checks give earlier guarantees of errors (resource utilization etc.). But in security, you don't control part of the system, hence guarantees are different than in the typical scenario. If you are worried about security you will need additional dynamic checks that are not needed in a non-security scenario.

## 3 Conclusions

The meeting was very productive, it provoked many new ideas and provided new perspectives on the topic. The participants learned a lot from each other, in particular, there was a lively exchange of knowledge between the programming language and the database world. We hope to further consolidate the results in an article that provides research directions for web programming and related areas.