What is the Web's Model of Computation?

Luca Cardelli

Digital Equipment Corporation
Systems Research Center

Global Computation

- The Internet makes *global computation* possible.
- It is becoming particularly appealing for the Web.
- However, "the Web" is not a single uniform structure.
- We should try to understand, in general the properties of Global Computation.

Global Structures

- Many kinds of global information structures.
- Some of them should be programmable: *global computers*.
- Programs for a global computer span multiple locations.
- Main new feature: wide-area distance.
- Drastic consequences for programming.

Many Global Computers

- The Web will be programmable, somehow.
- But a single global structures will not be enough.
  - ~ Breakup of the Web.
  - ~ IntraWebs.
  - ~ Non-HTTP, non-JavaVM global computers.
  - ~ Task-oriented global computers.
  - ~ Filters and firewalls separating global computers.

- What distinguishes one global computer from another?
- How will multiple global computers interact?
- We focus on Web-related global computers because of the Web's huge content.

What is the Web, really?

- To program the Web, and build API's, we need to agree on its model of computation.
- Is it a big file system?
- Is it a big multiprocessor?
- Is it a big distributed object system?
- Is it a big agent system?
- Or something else entirely?
- Or all of the above?
- Or each of the above, separately?

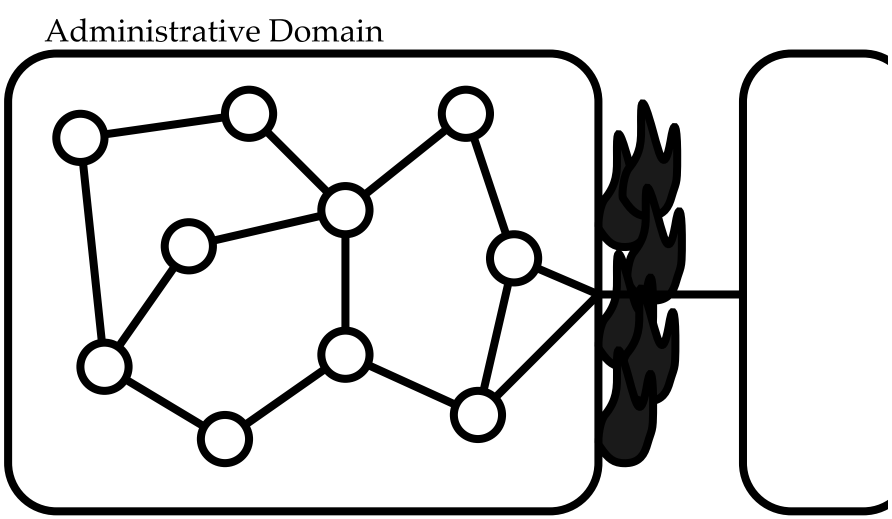
The Web is HTTP (+ CGI + MIME + ...)

- At its essence, the Web is what HTTP lets it be.
- The HTTP model is rather unusual.
  - ~ Stateless servers.
  - ~ Hit-and-run communication.
- Should we patch it, change it, cover it up, or adapt to it?
- What neat abstraction can be built on libwww?
- How can we extract easily the Web content?

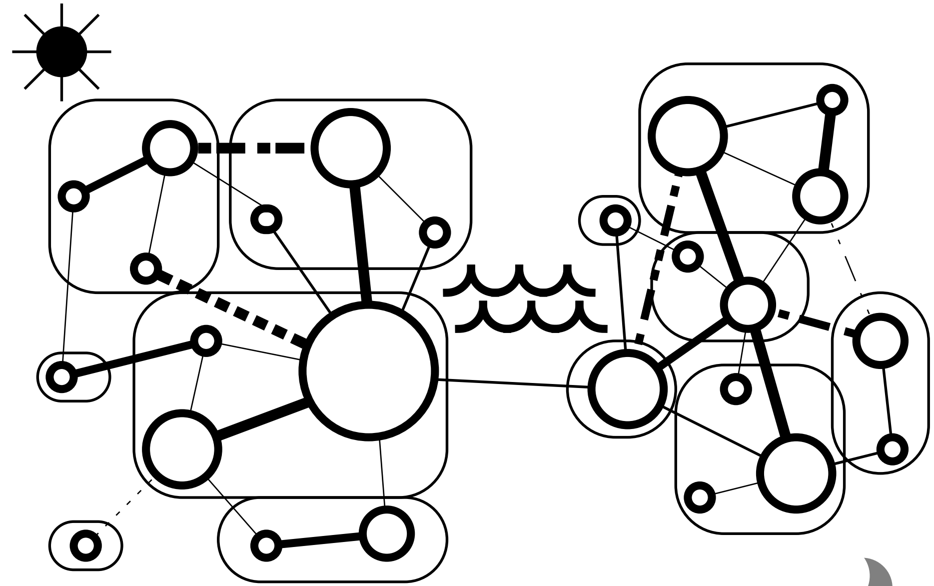
Models of Computation

- To program the Web we need to understand its model of computation.
- "Model of Computation" =
  - ~ Language-independent model for API's
  - ~ Model on which to base language constructs.

Distributed Object Systems



The Web



- What models are adequate for global structures?
  - ~ Procedural (=> RPC/Remote execution)
  - ~ Functional (=> Data flow)
  - ~ Object-Oriented (=> Distributed objects)
  - ~ Relational
  - ~ Concurrent
  - ~ Distributed
  - ~ Mobile

- These models have been well studied and formally characterized.
- But insufficient attention has been given to locality.
- “Who is running what, when, and where?”.
  - ~ What is an appropriate framework for formulating and answering these question?
- Is the Web (HTTP + whatever) already included in one of these models?

- “Model of Computation” = “What can be observed”.
  - ~ Locality
  - ~ Referential Integrity
  - ~ Quality of Service
- These observables are not part of traditional models.
  - ~ Web surfers exhibit new behavior.
- What models and programming constructs can we develop to automate behavior based on such observables? (Exceptions are not enough. No language constructs exists for Quality of Service.)

Programming Languages

- Different global computers may provide different guarantees.
- Therefore, different programming languages and models may be appropriate in each case.
- As an example, consider the following three programming languages that support mobile computation.

Telescript

- Telescript is an agent-based language that explicitly deals with locality, mobility, and finiteness of resources.
- Telescript agents may migrate to new locations while active, but cannot engage in distributed communication.
- Telescript agents run only on a dedicated global computer that guarantees the integrity and security of agents.

Obliq

- Obliq is an object-based language that deals with distribution and mobility.
- Mobile computations maintain distributed connections as they move.
- Obliq can run effectively on any single reliable global computers, but does not deal with firewalls, security, or widespread unreliability.

Java

- Java deals with security and multiple global computers (its programs are allowed to cross firewalls).
- Mobility, however, is restricted to transmitting program sources, in preprocessed form, and not computations.
- As a consequence, Java works satisfactorily in unreliably-connected environments, since passive sources do not maintain connections.
- Java is acquiring support for active distributed computation. It will then have to deal with the same problems as Telescript and Obliq.

Adequacy

- Each language is better suited than the others to a particular global computer.
- None is particularly well suited for general computation on the Web.

Semantics

- There is a need to develop semantic frameworks for these languages.
  - ~ Understand the computational assumptions and requirement of each language.
  - ~ Answer important security questions such as "Who runs what, when, and where?".
  - ~ Reason about programs.

The Obliq Model of Computation

- Both agent-based and object-based.
  Both mobile and distributed.
  Both Telescript-like and RPC-like.
- Assumes a reliable and secure global computer.
- Let us consider some somepeculiar examples.
  - ~ Antipodes Stock Exchange.
  - ~ Remote-ugly-API abstraction.

Uploadable API's

- A database site exports an *engine* (compute server). The engine supplies the database as an argument to incoming client procedures:

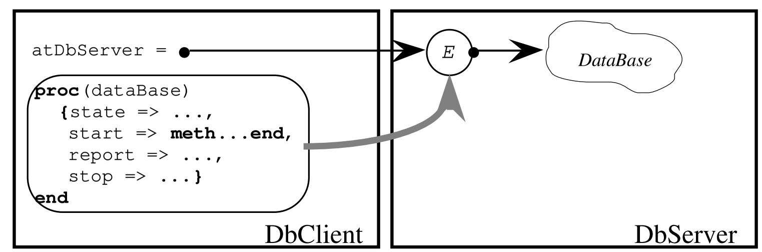
```
(* DataBase Server Site *)
net_exportEngine("DBServer", dataBase, Namer);
```


- A database client could send over procedures performing queries. However, for added flexibility, the client can instead allocate a remote object:

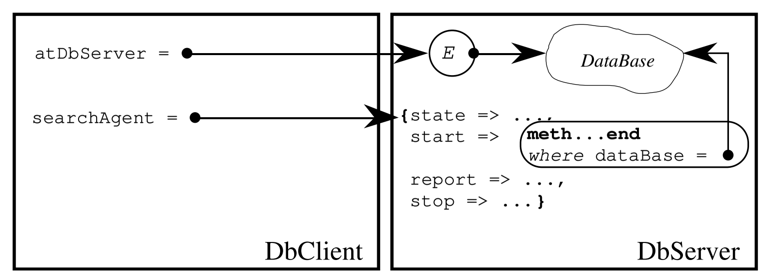
```
(* DataBase Client Site *)
let atDbServer =
  net_importEngine("DBServer", Namer);

let searchAgent =
  atDbServer(
    proc(dataBase)
      {state => ...,
       start => meth ... end,
       report => meth ... end,
       stop => meth ... end}
    end);
```


- The execution of the client procedure causes the allocation of an object at the server site with methods "`start`", "`report`", and "`stop`", and with a "`state`" field.
- The server simply returns a network reference to this object, and is no longer engaged (for the moment).
- The object contains uploaded client code, which can be invoked by the client.

Before the invocation:



After its completion:



Not a Standard Model

- The example of uploaded API does **NOT** follow:
  - ~ The (original) Java model.
    It uploads a live object to a server instead of downloading dead code to a client.
  - ~ The Telescript model.
    The client sends an "agent" to the server, but maintains connections and control.
  - ~ The Distributed Objects model.
    A live computation is transmitted, not just data over RPC.

Programming Issues

- Even standard programming issues acquire new facets in a global context.

Typing

- In order to program the Web, we need some notion of typing for the data that is exchanged.
- The Internet has in fact a rather sophisticated system of data types (MIME).
- To enable Web programming, Web servers will have to start providing typed data (currently, they mostly provide HTML data).
- Moreover, this typed data will have to be mapped to programming language types.

Security

- A model of security is essential for entrusting mobile computations.
- The cryptographic underpinnings of security are well understood.
- But it is not clear how to effectively and flexibly integrate security into programming languages and mobile computations.
- Paranoia rules.
- What should be the syntax, static checking, semantics, and logic of security?

Reliability

- Some global structures will always be unreliable, it seems.
- Unfortunately, like the Web itself, these may also be the most interesting global structures.
- Exceptions are not enough.
- We need to find programming constructs and methodologies (“quality-of-service abstractions”) that can increase the reliability of the substrate to tolerable levels.

Modularity

- An appealing possibility, pursued by Java, is that software components will be fetched dynamically over the network, whenever the need arises.
- This approach requires stronger modularity guarantees than ever before, as well as novel approaches to software production, distribution, and maintenance.
- It also suggests a notion of interfaces and modules as dynamic entities that is rarely found in current languages.

Resource Management

- Handling of finite resources.
  - ~ discovery, coordination, and retrieval
  - ~ time, space, and bandwidth
- Substantial new challenges are offered by the management of money:
  - ~ infinitesimal transactions
  - ~ infinite volume
  - ~ open commerce

Conclusions

- Today: locally-networked personal computers with poor global connections.
- In the future: network terminals that rely heavily and transparently on global resources.
- This transformation will be associated with the development of new computation and programming paradigms.
- These paradigms will be first embedded in libraries, then in API’s, and finally in programming languages.