Naming System
Design Tradeoffs

Noah Mendelsohn
Tufts University
Email: noah@cs.tufts.edu
Web: http://www.cs.tufts.edu/~noah
Goals

- Explore choices and tradeoffs relating to naming – how they impact what systems can do
- We’re focusing on designing naming systems...
- ...not primarily on choosing individual variable or document names
Why is this so important?
Why naming systems are important

- What you can name and how you name it tends to be fundamental to what a system can do
- Names tend to reflect or even determine the structure of a system
Names Reflect or Determine System Structure
Examples

- The Web *is* a graph of URI-based links and resources
- The Unix file name hierarchy integrates most resources in a Unix system (including processes, etc.)
- To be a phone: you need a phone number!
- In the US, a taxpayer *is* someone with a Social Security number
- In some computers, I/O devices are “mapped” into memory...an ordinary “store” into the right location may write to disk or send a network packet
Uniform names / Uniform access
Uniform names and operations

- If you have a uniform naming systems, and uniform operations – then you can operate on things you haven’t seen before
- Random exploration and unplanned connections become possible
Uniform naming and interface: Unix/Linux File Descriptors

- Single hierarchical filename space
- *The same name space is used for:*  
  - Files  
  - Sockets  
  - Devices  
  - Pipes
- For open files, sockets, pipes, etc. Unix uses file descriptors coded as small integers – common namespace for open resources

This uniformity makes possible the following flexibility:

```
sort /u/noah/somefile.txt
sort -r           # sort stdin
ls | sort -r       # from pipe
sort -r < /dev/tty # from console
```

So, Unix/Linux illustrate the uniform naming/uniform interface principle at two levels: hierarchical file names and also open file descriptors
int sock, file;
char c;
sock = socket(AF_INET, SOCK_STREAM, 0);   //open a socket
file = open("myfile", r);                 //open a file
(...missing code to bind socket, etc.)

// same routine reads from file or network
// read from socket
// read from file

// this routine works on any file, socket
// or character device
char
readchar(int fd) {
    char buf;
    int = read(fd, &buf, 1);               // should check errors 😃
    return buf;
}
open system call

open() system call returns file descriptor (int)

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int creat(const char *pathname, mode_t mode);

DESCRIPTION
Given a pathname for a file, open() returns a file descriptor, a small, non-negative integer for use in subsequent system calls (read(2), write(2), lseek(2), fcntl(2), etc.). The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

By default, the new file descriptor is set to remain open across an
socket system call

**socket() system call returns file descriptor (int)**

```
#include <sys/socket.h> /* See NOTES */
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

**DESCRIPTION**

socket() creates an endpoint for communication and returns a descriptor.

The **domain** argument specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in `<sys/socket.h>`. The currently understood formats include:

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
<th>Man page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF_UNIX, AF_LOCAL</td>
<td>Local communication</td>
<td>unix(7)</td>
</tr>
<tr>
<td>AF_INET</td>
<td>IPv4 Internet protocols</td>
<td>ip(7)</td>
</tr>
</tbody>
</table>
Code example

```c
int sock, file;
char c;
sock = socket(AF_INET, SOCK_STREAM, 0);   //open a socket
file = open("myfile", r);                 //open a file
    (...missing code to bind socket, etc.)

// same routine reads from file or network
char c = readchar(sock);                // read from socket
char c = readchar(file);                // read from file

// this routine works on any file, socket
// or character device
char
readchar(int fd) {
    char buf;
    int = read(fd, &buf, 1);              // should check errors 😊
    return buf;
}
```
Code example

```c
int sock, file;
char c;
sock = socket(AF_INET, SOCK_STREAM, 0);  // open a socket
file = open("myfile", r);                 // open a file

// same routine reads from file
char c = readchar(sock);
char c = readchar(file);

// this routine works on any
// or character device
char
readchar(int fd) {
    char buf;
    int = read(fd, &buf, 1);               // should check errors 😃
    return buf;
}
```

Open sockets and files named uniformly with int ids...

...enables uniform operations applied to both
Uniform naming and interfaces are essential to the Web

- **URIs name everything**
- **Same dereference rules for all URIs**

If I give you a URI, the same rules always apply for accessing it.

That’s what makes it possible to explore the Web!

(...imagine if instead the access technique for each page were different...)

References:
Global Names
Global vs. local names

- Global names: resolve the same way everywhere
- Local or relative names: resolution depends on context

```plaintext
sort test.txt
sort ./test.txt
```
Global vs. local names

- Global names: resolve the same way everywhere
- Local or relative names: resolution depends on context

Relative names depend on current directory

```
sort test.txt
sort ./test.txt
```
Global vs. local names

- Global names: resolve the same way everywhere.
- Local or relative names: resolution depends on context.

```
sort test.txt
sort ./test.txt
sort /h/noah/example/test.txt
```
Global vs. local names

- Global names: resolve the same everywhere.
- Local names: resolution depends on context.

sort test.txt
sort ./test.txt
sort /h/noah/example/test.txt

Firefox http://www.cs.tufts.edu/~noah/test.txt

Absolute URI resolves same way everywhere in the world!
Global names: uniquely powerful

- The key to building consistent global systems
- My phone number works anywhere in the world: 1-617-555-1212
- If I e-mail you a URI you can access it from anywhere in the world!
Global names: same name → same object

- …but what is the “same” object?
- `www.nyt.com` is the New York Times home page – it changes every few minutes
- Browsers know who prefers to speak French…is it OK to return a French page for the same URI?
- Can I return a compact page if the access is from a small device?

We’ll explore these issues when we discuss naming for the Web.
Principle: Global naming leads to global network effects

Reference: http://www.w3.org/TR/webarch/#pr-global-id
Absolute and Relative Names
Names can be **absolute** or **relative**

- Every name is resolved relative to some **context**
- If the name is resolved wrt/ a local context, we call it **relative**:
  
  
  ```
  Unix: ../a/b/c
  URI: <a href="../test.txt">...
  ```

- Names resolved relative to global context are **absolute**

  ```
  <a href="http://www.cs.tufts.edu/~noah/test.txt">
  ```

Only absolute names provide location-independent naming
Note: Some URI schemes don’t have global context

URI schemes that resolve differently according to context cause problems

<a href="file:///C:/noah/testfile.html">

Reference: [http://www.w3.org/TR/webarch/#pr-global-id](http://www.w3.org/TR/webarch/#pr-global-id)
Names and Addresses
Names and Addresses

**Name**: an identifier for something

- Not all names are addresses. Examples: GUIDs, URNs, your name!
- For names that are not addresses: some other means is needed of resolving the name to an object (e.g. hash table, registry, associative search, etc.)

**Address**: a name that helps you *locate* something

- Examples: HTTP URIs, phone numbers, postal addresses
- There is (usually) a specified means of *dereferencing* a name to get to the thing it points to
- Examples
  - Most CPUs provide Load/Store to dereference memory pointers
  - RFC 3986 points to rules for resolving URIs that are addresses (e.g. http-scheme URIs)
  - International agreements provide means of resolving a phone number to the phone it names
  - The US post office can route a letter by ZIP code, and then to the local town, street, house, etc.
Too Many Names
Multiple names for the same thing (aliases)

- **Can be handy**
  - Even for simple reasons: http://example.com vs HTTP://EXAMPLE.COM
  - Absolute vs. relative names for the same thing
  - a/b/c vs a/b/../b/c

- **Creating more than one name for an object tends to cause confusion**

- **Reasoning about systems with aliases is much harder**

- **Even allowing for multiple names causes trouble**
  - If I have a cached copy of http://example.com, is it usable for references to HTTP://EXAMPLE.COM? -- cache code must be prepared to check
  - Computer languages that allow pointer aliasing are very hard to optimize…it’s very hard for the compiler to keep track of which pointers might be referencing a given part of memory

- **On the Web: aliases make it hard to tell which pages are popular**

Reference: http://www.w3.org/TR/webarch/#avoid-uri-aliases
Principle: aliases complicate reasoning about a system
Too Few Names
Name *collisions*: using one name for multiple things

- We often don’t want to bother to name *everything*
- **Examples**
  - Multiple versions of a page that changes
  - The mobile vs. full size version of a Web page
  - A resource (a movie) vs information about the resource (blog about the movie)
- But when we do that...we lose the ability to reliably pick out one or the other

Reference: [http://www.w3.org/TR/webarch/#pr-uri-collision](http://www.w3.org/TR/webarch/#pr-uri-collision)
Good practice: give distinct things distinct names
Short names vs. Unique names

- When I joined Tufts, I asked for email noah@cs.tufts.edu...
  - …then I met Noah Daniels

- I could have asked for mendelsohn@cs.tufts.edu, but...
  - …people can’t remember how to spell my last name!

  Short names are good
  Unique names are good
  Short names tend not to be unique

This is a recurring and very, very important system design problem!
Who Gets to Make Up Names?
Distributed extensibility and naming

- We don’t want to have to ask permission to make a Web site
- We don’t want to have to ask W3C to try a new HTML tag
- We don’t want to have to ask permission to start using a new experimental data format on the Web

  Short names are good
  Unique names are good

  Names anyone can generate w/o collision tend to be long & ugly
Anyone can make their own…

- **Java package**
  - Example: `edu.tufts.comp.150ids.MyDemoClass`

- **URI**
  - Example: `http://local.google.com/maps?q=tufts+university&hl=en&sll=42.036922,-71.683501&sspn=3.08011,6.92688&hq=tufts+university&t=m&z=16`

- **GUID**
  - `44EC053A-400F-11D0-9DCD-00A0C90391D3`

*These names are globally unique and can be generated in a distributed way... but they are clumsy and hard to remember*
Short names are convenient

- HTML `<video>` tag
- C++ keywords: `for, while`
- Local e-mail name: noah@cs.tufts.edu
- Media type: `image/jpeg`

Someone has to decide who gets the popular ones...

...tends not to scale
Constraint: distributed extensibility can be very important...but often results in cumbersome names
Opaque Names
Tradeoff: opaque vs. transparent names

- Opaque names: users and code should not determine information from structure or content of names
- Transparent names: information may be determined from inspection of the name
Advantages of opaque names

- No special cases to avoid when allocating names
  - Any name usable for any object
- All names treated uniformly – often more efficient
- Separation of concerns: the name solves one problem only...identifying an object
Transparent Names
Reflect System Structure
Advantages of “transparent” names

- It’s very handy to know something about an object from its name
- Names can reflect the structure of the system:
  - /comp/150IDS/files suggests a hierarchy of files for Comp Sci courses
- If consistent patterns are used, we can often predict one name from another
  - If http://weather.example.com/?city=miami is the Miami weather
  - …then maybe http://weather.example.com/?city=boston will get you Boston?
- Lots of different types of information winds up in names (for better or worse)
  - File type, parent directories, DNS name of host, operations
- Only trust what the specifications say counts
  - In a URI, .html does not necessarily mean HTML, .jpeg need not be image/jpeg, etc.
  - mailto: definitely identifies a mailbox, because the specifications say it does in all cases
Must Addresses Resolve?

We said before that an address is a name for which there’s an agreed resolution strategy ... if you try, are you guaranteed to succeed?
In some systems every legal name resolves

- Some programming languages will create a variable when you access it
- Some hypertext systems won’t let you make a link unless the target exists
- Indeed, Tim Berners-Lee’s early work on the Web was rejected by academics because he didn’t insist on such consistency...
Dangling links and “404” status codes

- Indeed…one of Tim’s key contributions was the realization that enforcing such consistency is impractical on a global scale.
- You can have a global Web, or you can have all links resolve…but not both!
- That’s why the 404 HTTP failure code is one of the most important architectural features of the Web!
In some systems, the name “protects” the resource

In Java, if you have a reference to an object, that object will remain in memory until the reference is cleared:

```
Myclass name_a = new Myclass(); // create object
Myclass name_b = name_a;       // copy the name

name_a = null;                  // the new object still exists at this point
// because name_b is referencing it

name_b = null;                  // all references gone...object will (probably) be deleted
```

For this reason, Java has nothing like C’s “free” or C++’s “delete”…Java is a “garbage collected” language
In some systems, the name “protects” the resource

In Java, if you have a reference to an object, that object will remain in memory until the reference is cleared:

```java
Myclass name_a = new Myclass(); // create object
Myclass name_b = name_a;        // copy the name

name_a = null;                   // the new object still exists at this point
// because name_b is referencing it

name_b = null;                   // all references gone...object will (probably) be deleted
```

For this reason, Java has nothing like C’s “free” or C++’s “delete”…Java is a “garbage collected” language
Miscellaneous
Indirect identification

- Sometimes we use an identifier for one thing

  mailto:noah@cs.tufts.edu

- To indirectly identify something else

  noah@cs.tufts.edu is the teacher of the course

Sometimes this “trick” is very useful, but it can also cause confusion.
Summary
We’ve covered a lot…what’s important?

- Names tend to be critical to the structure & success of a system
- There are a number of characteristics of names that are important to understand
  - Opacity, names vs. addresses, relative vs. absolute, global vs. local, etc.
- Make sure you understand the key messages from each section of this presentation
- In many cases, the tradeoffs are difficult…but there are many excellent examples of existing systems you can study