CptS 464/564
Lecture 6
September 9, 2004

9th September 2004

Where we are

- Starting Chapter 2, DS Paradigms
  - Paradigm: A set of assumptions, concepts, values, and practices that constitutes a way of viewing reality for the community that shares them, especially in an intellectual discipline. (American Heritage Dictionary on-line)
  - In Chapt. 2 we look at 11 concepts that commonly arise in DS. We flesh each one out with a little bit of context, but don't try to put them together as *systems*. A bottom-up approach.
  - In Chapt. 3 then, we look more from the top down and describe models of whole systems.

Naming

Names: logicians and philosophers have thought deeply for centuries about the meaning and use of names. Even Shakespeare got into the act. "What's in a name? That which we call a rose by any other name would smell as sweet" (Romeo and Juliet). The kinds of philosophical dilemmas that arise are illustrated in the title: "What is the name of this book?" (Smullyan)

Fortunately, naming in (distributed) systems is not so problematic.

Naming is fundamental in computer science. We see it in programming languages, operating systems (files), mail systems, and distributed systems of all kinds. The fundamental concept is that of a binding between a name and a value. Important issues:

- what does a name look like (*syntax*),
- in what *context* is the name bound to the value

- can a value, itself, be a naming context (hierarchical naming: internet names (DNS))

Naming contexts frequently provide binding of names to *addresses*. An address is a name that is useful for locating an entity. Frequently, use of an address requires looking it up in another context (more later on addresses).

Uses for names:

- providing a natural-language-like name for something that would otherwise be unpronounceable or unmemorable. In programming languages, names stand for memory addresses; in the internet machines are given names because IP addresses are immemorable.
- when values are addresses, provide a level of indirection; indirection allows changing the path from a name to its referent at different times. Example: postal holding/forwarding. My street address is normally bound to my house; but if I go away and request that my mail be held, it becomes temporarily bound to a box in the back room of the postoffice. If I move it becomes bound to a process that readdresses the mail and resubmits it for processing.
- reuse of names and organize resources: *hierarchical naming*. There are too many things to name to let us use a unique word or two for each one. Solution: name a context in which a name is to be interpreted. Ex: poipu.eecs.wsu.edu; chauser@pullman.com; 730.Carolstar Dr.SE.Pullman.WA.USA; /users/hauser/.login. Note how as we move down the hierarchy (poipu.eecs.wsu.edu) the type of value remains "naming context" until we get to the end when 199.237.73.189 of type "address" is the final value. Note how the address is itself a hierarchical name.

What kinds of things are named in DS? computers (poipu.eecs.wsu.edu), services (ssh, telnet, ftp, http), objects (in the program+data sense), files, people.

Addresses are not the only things that can be bound to names: other *attributes* can be included in a binding. (e.g. person's title, business affiliation; computer's CPU type, OS, etc.). N.b. addresses are just one form of attribute.

The process of looking up a name to determine its attributes is called *name resolution* or *resolving the name.* Some directory services allow looking up names from attributes: *reverse resolution.*

Non-standardized terminology:

- some names are intended only for interpretation by programs, not people. Some people call these *identifiers.* Identifier formats are chosen for efficiency of lookup and storage. Examples: NSF file handles, CORBA IORs, ethernet addresses

- some names have no internal structure to reveal anything about the value they refer to. Such names are called *pure names* or *opaque names.* Sometimes the purity of names is just a fiction. Example: CORBA IOR: for CORBA clients and servants, IORs are to be treated as opaque, but for the ORB itself, the IOR contains useful information. Ethernet addresses are 6 bytes and opaque for most purposes. But in fact they have structure that reveals the manufacturer. Other names a truly pure. Example - name formed by applying a one-way hash function to a value. What can you do with a pure name:
  - Look it up in the appropriate context
  - Compare it to another name
- *impure names or transparent names*: have internal structure reflecting the structure of the space in which they name something chauser@eecs.wsu.edu.

Aliases - names that resolve to other names. Example: Unix symbolic link. mailserver.eecs.wsu.edu resolves to (right now) viggen.eecs.wsu.edu. (18 mos. ago it was jaguar.eecs.wsu.edu.) Indirection again. mailserver.eecs.wsu.edu is a published, "well-known" name that is configured into many mail tools. Using an alias lets the administrators move the actual service from machine to machine as needed.

Addressing

Addresses: special kind of name that has meaning in a particular communication protocol

Point-to-point address (aka unicast address): allows addressing an object as an individual

Logical group name/address: name for a set of objects. If a *group communication protocol* is available a group name may not have to be translated to a set of point-to-point addresses.

Important point: an address is a name; addresses may go through many layers of name resolution before a message can be delivered.

Name resolution

- includes name-to-address translation
- Approach 1: broadcast
  - Taking roll on the first day of class
  - Problem: consumes resources on all nodes that receive the broadcast
  - Problem: not all communication protocols support broadcast

- Approach 2: name server
  - How do you know how to talk to the name server?
  - It has a *well-known* address (everybody does it the same way)
  - It has a preconfigured address (everybody does it differently but you can't play in a particular environment until you've been told the configuration information.
  - Bootstrapping: DHCP – "what is my IP address, what is my hostname, what is my router, and what is my name server" broadcast. Note that this is a reverse name lookup – ethernet address to IP address and hostname

Name server implementation

- idea of a name server is simple enough
- implementation is challenging
  - *availability* is important (availability=(timeAvailable/timeNeeded)). Different nodes may measure different availability. It is important that availability be high at all nodes.
  - *low-latency response* is important – in hierarchical name lookup latency means time to go from qualified name to name at next protocol level. E.g. from DNS name to IP address
  - scaleability is important: provide name resolution for hundreds of millions of names, administered by millions of different organizations, worldwide.
- distributed name service for hierarchical names
  - Each server stores bindings for only a portion of the name space; crossing between portions is defined as part of the protocol
  - Each portion is replicated at several, but not all, servers; replication is defined as part of the protocol
  - Division into portions can be made based on geography, administrative boundaries, ... How to draw partition boundaries is not part of the protocol
  - Name service *agents* implement client part of the protocol. Two possibilities for agent behavior
    - first name server contacted completely resolves the hierarchical name (recursive resolution)
    - each name server contacted by the agent performs one step of the resolution and passes back identity of next name server to contact and the question to be asked of it
  - Caching is key to good performance

    - Cache: a copy of information kept close to where it is used, in order to improve latency and availability, reduce communication, and reduce name server load
    - Caching has different importance and different characteristics at different levels of the hierarchy: at the top level, where com, edu, biz, net, us, ... are looked up, things change very infrequently, but lookups are very frequent. Name servers at the next level (for looking up up wsu in edu for example) may be the most difficult level: high lookup frequency, relatively high change frequency, very large number of names so caches aren't as effective.
    - Caching may mean that wrong answers are returned sometimes (when bindings are changed); protocol allows client to force a lookup to go to an "authoritative" server so answer does not come from a cache
- Replication vs. Consistency
    - We'll look at these issues in later paradigms. For now, notice that
        - if we allow updates without synchronously updating all the replicas then clients may read different results from different servers
        - if we insist on consist results then we have to make updaters wait until many replicas are available –> unavailability for update
    - Usual decision for name servers: availability is more important than consistency
        - assumptions: name data changes infrequently so inconsistency is rare
        - if you get an obsolete address and try to use it, it won't work
        - if an obsolete address does work, it's harmless
        - how good are these assumptions?