Name, Binding, and Reference

Monday, November 26, 2007
9:16 PM

- Computer system manipulate objects.
- Two ways one object can use another.
  - Use by value: Create a copy of the other object and pass that to the first object.
  - Use by reference: Name the other object and pass the name.
- Why use name?
  - Use by value does not permit **sharing.**
  - Allow designer to defer which object the name refers to until later.

- Naming model
  - Operations
    - value <- RESOLVE(name, context)
    - status <- BIND(name, context)
    - list <- ENUMERATE(context)
    - result <- COMPARE(name1, name2)
  - Vocabs
    - *value* = object or another name
    - *binding* = a map from a particular name to a particular value
    - *name space* = an alphabet of symbols together with syntac rules that specify which names are acceptable.
    - *name-mapping algorithm* = an algorithm that associates some names of the name space with some values
    - *resolve* = to perform the name-mapping algorithm from a name to the corresponding value
    - *context* = one of the inputs to the name-mapping algorithm. The common form is a set of name-value bindings
    - *universal name space* = a name space that has only one context. No matter who utters the name, the name has the same binding.
    - *free name* **=** a name that is not bound in a context
    - *stable binding* = a binding that stays the same for the lifetime of the namespace

- Three types of name mapping algorithms
  - Simple table lookup
  - Path name resolution
  - Search

- Default and explicit context reference
  - Who supplies the context argument in RESOLVE(name, context)?
  - *Default context reference* = the resolve supplies the context reference
    - Constant built into the resolver
    - Variable from the current environment
  - Explicit = specified by the object
    - Per object
    - Per name (*quantified name*)
  - Actually, the context argument is not the object. It is itself is a name, called the *context reference.* Resolver must resolve the context reference to a real context object. So, resolving is recursive. This recursion must end somewhere.
  - Problems
    - Object that utters the name does not provide an explicit context, and the name resolver chooses the wrong context.
    - Different contexts might bind different names for the same object.
      - Hard to pass name from one context to another.
      - Example: Phone numbers

- Path names and naming networks
  - *Path name*

        - Explicitly include a reference to the context in which it should be resolved.
        - Have multiple components.
            - Least significant component = the name
            - All other parts = reference to the context in which the name is to be resolved.
    - Recursive! Need to resolve the reference to the context (which itself is a path name) before we can resolve the name.
    - Default context references for path name (the recursion must end somewhere!)
        - *Root* context built in to the resolver.
        - The resolver can store the path name of the default context. This needs to be resolved again. (Think working directory).
    - *Absolute path name* = resolved using root context.
    - *Relative path name* = resolved using the path name to the default context.
    - *Naming network*
        - Contexts are objects.
        - Each context may bind a name to other contexts.
        - Name resolve chooses one context as root, and traces a path from the root to the first named context in the path name, and then the next, and so on until the path name runs out.
        - Different objects can have different path names in a naming network.
        - Names that refer to the same objects are *synonyms*.
        - Users may express path names relative to different roots. So it's hard for them to share names.
    - *Naming hierarchy*
        - Naming network that is actually a tree.
        - Root context is the root of the tree.
        - Every object has a unique path name.
        - Very constraining. Not found in practice.
    - Can add *indirect names* (names that are resolved to another path name) to naming hierarchy to permit cross-hierarchy linking.

- Search
    - Use an ordered list of contexts instead of a single default context.
    - Name resolver tries to resolve the name in the first context. If the result is not-found, then it tries the next context. This repeats until the list is exhausted or the resolver finds the first context that the name has a binding.
    - Search path is usually implemented as a per user list. This permits *user-dependent binding*. (Think about PATH variable.)

- Context Layers
    - Found in programming languages
    - Context are arranged into layers.
    - When the resolver cannot resolve a name in some layer, it tries resolving the name in the enclosing layer.
    - *Scope* = the range of layers a name is bound to the same object.
    - *Global name* = a name that is bound only in the outermost layer.

- Name discovery
    - How did you know to use this name?
    - *Name discovery protocol* = inform an object's importer the name that the object exports.
        - Exporter *advertise* the existence of the name.
        - Importer *searches* for an appropriate advertisement.
    - Forms
        - *Well-known name* = a name advertised so widely that it can be counted on to be stable.
        - Broadcast
        - Search = ask google
        - Reverse broadcast = ask everyone whether he/she knows the name of something
        - Narrowcast: Send "hello my name is …" down the wire and hope that the other end will listen and reply.
        - Introduction = party and dating services

        - Physical rendezvous

- Names and modular sharing
    - *Modular sharing* = can use an object, which itself is modular, without knowing the names of modules it uses.
    - Name conflicts is a syndrome of the lack of modular sharing.
        - Serious problem because resolving it means changing ways in which objects use name. This means changing the object itself!
    - *Imposed names* = names chosen by someone else
    - Common way to provide modular sharing: give each object its own context, and figure out a way to cross reference between contexts.
    - Programming languages use static scoping and closure to solve this problem. This mechanism is not found in file systems.

- Metadata and name overloading
    - *Metadata* = information about the object that is not a part of the object itself.
        - Name
        - Location
        - Time modified
        - Etc.
    - *Overloaded name* = name that has metadata in it
        - hello.c
        - Leonardo da Vinci
        - Physical address
        - ZIP Code
    - *Pure name* = no metadata inside, so has no relation to the object it refers to.
    - Problems with overloaded name.
        - Unstable name
            - Directory name /disk05. What if you move all the files to Disk 4 instead?
        - Tension between name stability and the need to update the overloaded information.
    - *Address*
        - Name of a physical location or a virtual location that maps to physical location.
        - Addresses are always overloaded. It always has information about the physical location of the object being referred to.
        - Address adjacency and physical adjacency go together. Arithmetic on addresses have corresponding physical meanings.
        - Addresses are extremely unstable.
    - How to cope with unstable names?
        - **Hide unstable name with indirection.**
        - Have user of the object refer to the object by a generic name.
        - The generic name itself is bound to the unstable name.
        - Can change the generic-unstable-name binding later.
        - Every problem in computer science can be solved with another layer of indirection.

- Generating unique names
    - Use consecutive integers as names.
    - Choose names at random from a large name space.
        - It's hard for a real machine to be perfectly random.
    - Use hash functions.
        - But names generated this way is very unstable.
    - Hierarchical naming scheme.
        - Useful for assigning names in a geographically distributed system.
        - Exploit delegation
        - Examples
            - Host names.
            - MAC addresses

- Relative lifetime of names, values, and bindings
    - *Dangling name*

        - A name that outlives its binding.
        - Resolve to irrelevant values or not-found.
        - Can lead to serious errors when names are reused.
        - Can be dealt with by verifying the object resulted from resolution if it meets the name user's expectation.
    - *Orphan*
        - An object that outlives its binding.
        - Cannot be accessed by names again.
        - Can be dealt with by reference counting or garbage collection.