

CISC 839

Software Engineering of Usable Computing Systems

T.C. Nicholas Graham

<http://www.cs.queensu.ca/~graham/cisc836>

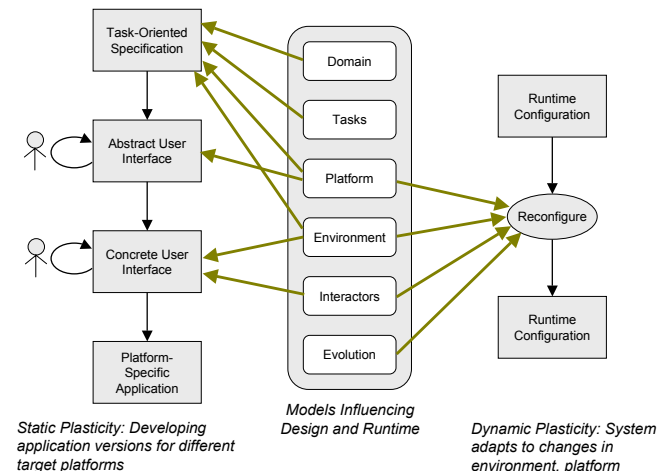
Plasticity in User Interfaces

- ⌘ Proliferation of device types has led to difficulty in producing, maintaining different versions of products
- ⌘ Platform versions differ in
 - ⊞ User interface
 - ⊞ Functionality
 - ⊞ Development techniques

Version Problem

- ⌘ Therefore, each version requires its own
 - ⊞ system task model
 - ⊞ UI design
 - ⊞ Architecture
 - ⊞ Evaluation
 - ⊞ Usability testing, heuristic evaluation, etc.
- ⌘ Maintaining consistency between versions
- ⌘ Product branding

Plasticity in User Interfaces



Solutions

- ⌘ Still a research area
- ⌘ Static plasticity
 - ☒ Generate some portions of UI from higher-level models
 - ☒ E.g. from task model (Adept, Trident, Teallach)
 - ☒ E.g., from abstract UI description (XML/XSL, WML)
- ⌘ Dynamic plasticity
 - ☒ System uses runtime models to adjust its behaviour

Example [Crease et al., 2000]

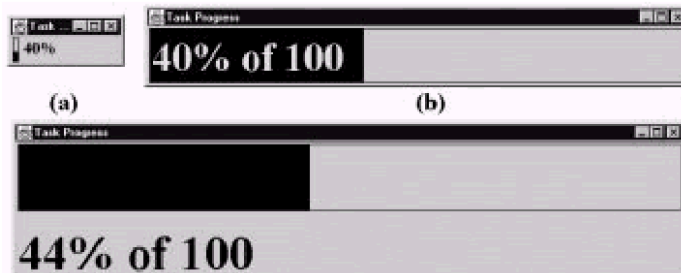
⌘ Motivation

- ☒ Context of use of hand-held devices constantly changes
- ☒ E.g., context of mobile phone includes
 - ☒ Ambient light, sound
 - ☒ Use - when phone is used for voice, user may not be able to read display

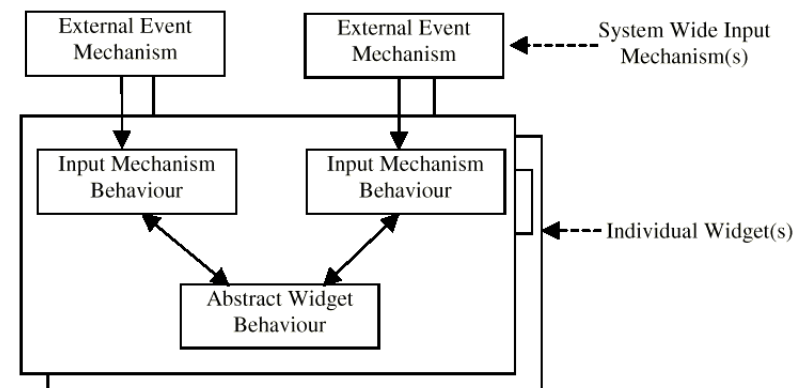
⌘ Architecture

- ☒ Abstract user interface provides interacts with application
 - ☒ Specified as finite state machine
- ☒ Concrete instantiation of user interface runs as coprocess with abstract interface

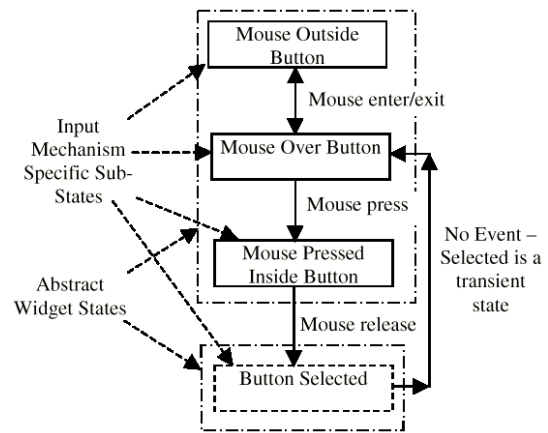
Simple Example: Three Implementations of a Progress Bar



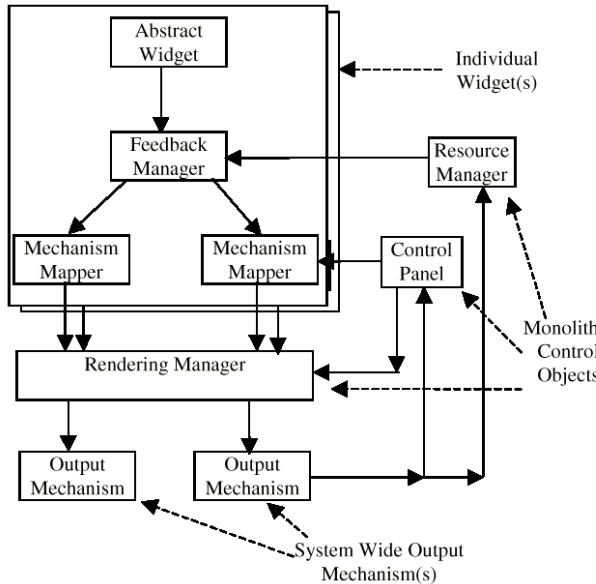
Input Architecture



Example - mouse selection



Output Architecture



Dynamic Discovery Architectures

Example: Printing a Photograph

- ⌘ I take a picture on my digital camera
- ⌘ I want to print it



Example: Printing a Photograph

- ⌘ I go to a shop that has printing services
- ⌘ The shop has several printers at different prices
 - ☑ Colour vs black and white
 - ☑ Photo-grade paper vs normal paper



Black and White
(\$0.50/page)



Colour (\$2/page)

Example: Printing a Photograph

- ⌘ I select a print command on the camera
- ⌘ My camera uses a wireless network connection to find out information:
 - ☑ What printers are available
 - ☑ What their properties are
 - ☑ Colour/B&W, Price, ...



Black and White
(\$0.50/page)



Colour (\$2/page)

Example: Printing a Photograph

- ⌘ My camera has an LCD display that shows which printers are available
- ⌘ I choose one, and start printing on the selected printer



Black and White
(\$0.50/page)

Select Printer

☒ Black and White
⌘\$0.50/page

☐ Colour
⌘\$2.00/page



Colour (\$2/page)

Interesting Features of Example

- ⌘ My camera **dynamically discovers** existence of two printers
 - ☑ Consequence of invoking "print" in the printing shop
- ⌘ My camera dynamically determines how to communicate with the chosen printer

Dynamic Discovery

Architectures

⌘ Features

- ☒ Runtime ability to find architectural components offering **services**
- ☒ Ability to negotiate how to communicate with discovered services
- ☒ Ability to dynamically configure connections to these services
- ☒ Ability to repair faults ("self-healing")

Examples of Services

- ⌘ Printing
- ⌘ Faxing
- ⌘ Statistical analysis
- ⌘ Maps
- ⌘ Phone book
- ⌘ C++ compiler
- ⌘ Disk storage
- ⌘ Downloading
- ⌘ English→French translation
- ⌘ Credit card validation
- ⌘ Airline flight information
- ⌘ ...

Examples of Dynamic Discovery Architectures

⌘ Jini

- ☒ Java technology
 - ☒ W. Keith Edwards, *Core Jini*, Prentice Hall, 1999
 - ☒ <http://www.jini.org>

⌘ Bluetooth

- ☒ Wireless dynamic discovery protocol for small devices
- ☒ Consortium: 3Com, Ericsson, IBM, Motorola, Nokia, Microsoft, ...
 - ☒ <http://www.bluetooth.com>

⌘ UPnP

- ☒ Universal Plug and Play
- ☒ Microsoft initiative, based on XML, focus on home networking
 - ☒ <http://upnp.org>

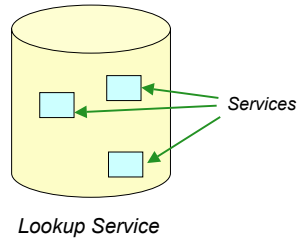
Jini

⌘ Primary features

- ☒ Discovery
- ☒ Lookup
- ☒ Leasing

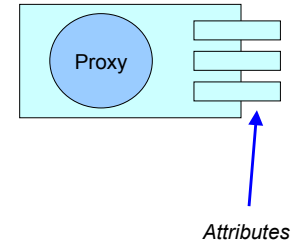
Lookup Service

- ⌘ Services are registered via a **Lookup Service**
- ⌘ Services register with a lookup service
- ⌘ Clients find services via lookup service
- ⌘ Lookup service is a form of “yellow pages” where services are advertised and found



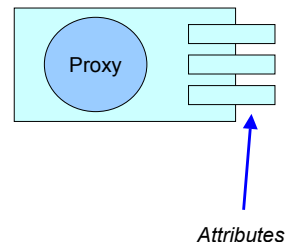
What is a Service?

- ⌘ A service has a **type**
 - ☒ E.g., Printer, FaxMachine, MapDataBase, ...
- ⌘ A service has a **proxy**
 - ☒ Code implementing the service
 - ☒ Typically code that connects to the implementation of the service
- ⌘ A service has an **id**



What is a Service?

- ⌘ A service has a **set of attributes**
- ⌘ E.g., for printer, attributes may be:
 - ☒ Speed
 - ☒ Location
 - ☒ Cost
 - ☒ Colour/B&W
 - ☒ Paper types, sizes available
 - ☒ ...



Service Types

- ⌘ Service types are simply Java interfaces
 - ☒ A set of methods that the service implements
- ⌘ The proxy is Java code implementing the service type
- ⌘ Example (next slide)
 - ☒ Jini **PrintService** interface

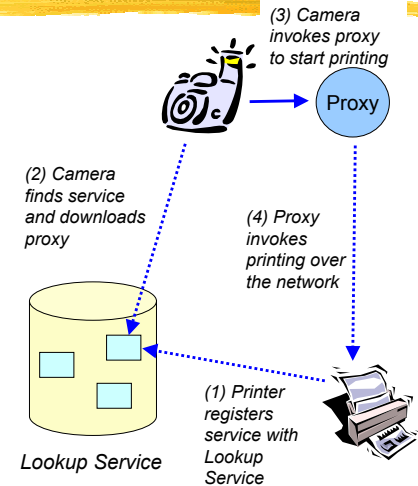
Method Summary

<code>net.jini.core.event.EventRegistration</code>	<code>addEventListener(Class[] theCategories, PrintServiceAttribute[] theValues, net.jini.core.event.RemoteEventListener theListener, MarshallableObject theHandbackObject, long theRequestedLeaseDuration)</code> Add an event listener to this Print Service.
<code>DocPrintRequest</code>	<code>createDocPrintRequest()</code> Create a Print Request object, bound to this Print Service object, that is able to print a single doc.
<code>DocPrintRequest</code>	<code>createDocPrintRequest(Doc theDoc, PrintRequestAttributeSet theAttributes)</code> Create a Print Request object, bound to this Print Service object, to print the given doc with the given set of printing attributes.
<code>PrintServiceAttributeSet</code>	<code>getAttributes()</code> Obtain a snapshot of this Print Service's attribute set.
<code>Attribute</code>	<code>getDefaultAttributeValue(Class theCategory, Settings theSettings)</code> Determine this Print Service's default printing attribute value in the given category.
<code>AttributeSet</code>	<code>getDefaultAttributeValues(Settings theSettings)</code> Determine all of this Print Service's default printing attribute values.
<code>Class[]</code>	<code>getSupportedAttributeCategories(Settings theSettings)</code> Determine the printing attribute categories a client can specify when setting up a job for this Print Service.
<code>Object</code>	<code>getSupportedAttributeValues(Class theCategory, Settings theSettings)</code> Determine the printing attribute values a client can specify in the given category when setting up a job for this Print Service.
<code>DocFlavor[]</code>	<code>getSupportedDocFlavors(Settings theSettings)</code> Determine the print data formats a client can specify when setting up a job for this Print Service.
<code>Settings</code>	<code>getUnsupportedSettings(Settings theSettings)</code> Determine the settings for a supposed print job that this Print Service does not support, if any.
boolean	<code>isAttributeCategorySupported(Class theCategory, Settings theSettings)</code> Determine whether a client can specify the given printing attribute category when setting up a job for this Print Service.
boolean	<code>isAttributeValueSupported(Attribute theValue, Settings theSettings)</code> Determine whether a client can specify the given printing attribute value when setting up a job for this Print Service.
boolean	<code>isDocFlavorSupported(DocFlavor theFlavor, Settings theSettings)</code> Determine whether a client can specify the given print data format when setting up a job for this Print Service.

Source: <http://www.jini.org/standards/>

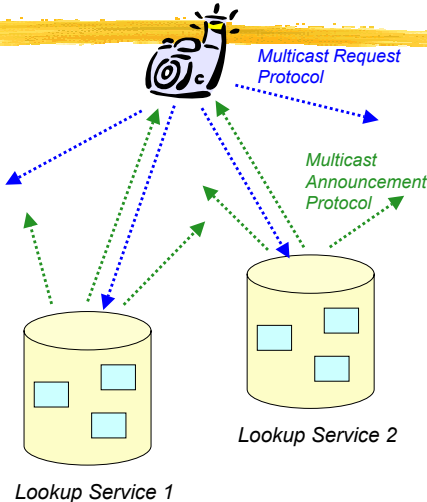
Using a Service: Overview

- Services register themselves with Lookup Service(s)
- Clients search for services. When they find a match, they download the service's proxy
 - Java code
- The client runs the proxy (locally)
- If necessary, the proxy connects back to the service to carry out its task



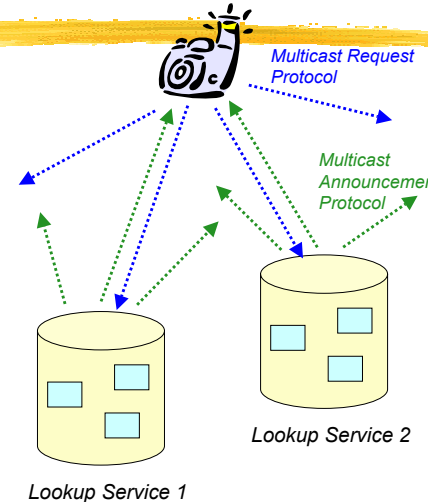
Discovery

- Both services and clients need to be able to find local lookup services
 - To register, find services
- In any location, there may be numerous lookup services available



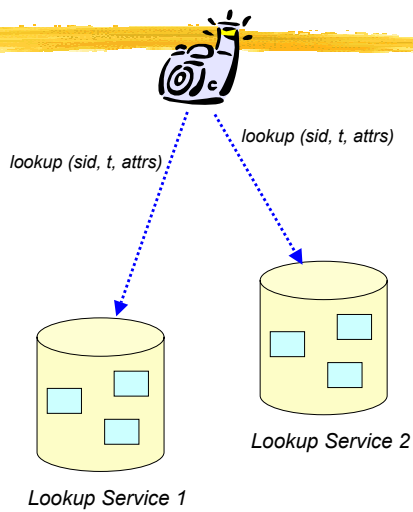
Discovery

- Client (e.g. camera) searches for Lookup Services via Multicast Request Protocol
- Lookup Services broadcast their presence via Multicast Announcement Protocol
- Via these mechanisms, a client can find a set of Lookup Services



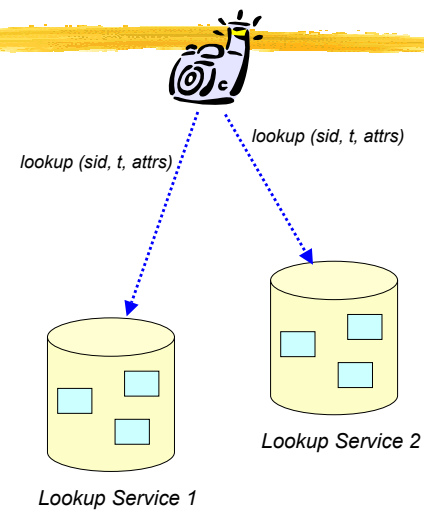
Lookup

- ⌘ When looking for a service, client performs lookup on lookup service(s)
- ⌘ Lookup based on any of
 - ☑ Service id
 - ☑ If service has known id
 - ☑ Type
 - ☑ E.g., Printer, MapDataBase, ...
 - ☑ Attributes
 - ☑ E.g., Price < \$3, Speed < 10 minutes, ...



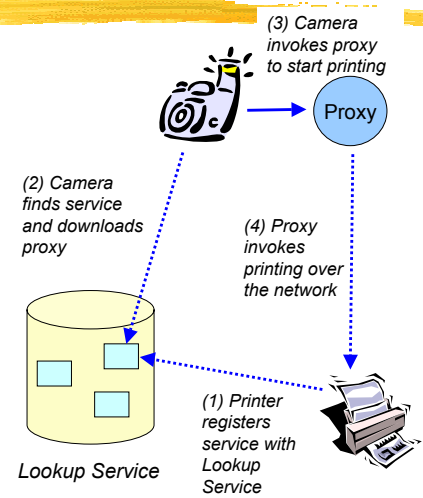
Lookup

- ⌘ Lookup provides all matches to query
- ⌘ Can specify any subset of parameters
 - ☑ E.g., just type, just attributes, ...



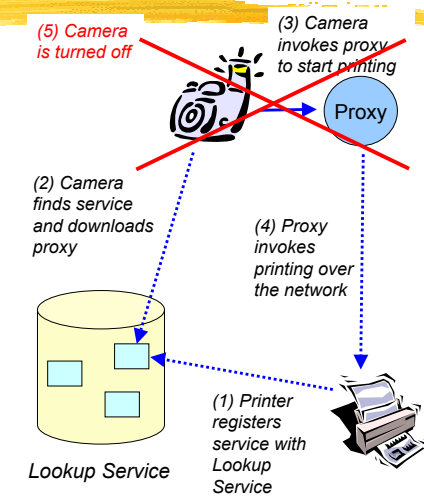
Fault Tolerance

- ⌘ What happens if part of the system fails?
 - ☑ Case 1: the client crashes while holding a service
 - ☑ Case 2: the service crashes while still registered with the Lookup Service



Fault Tolerance (1)

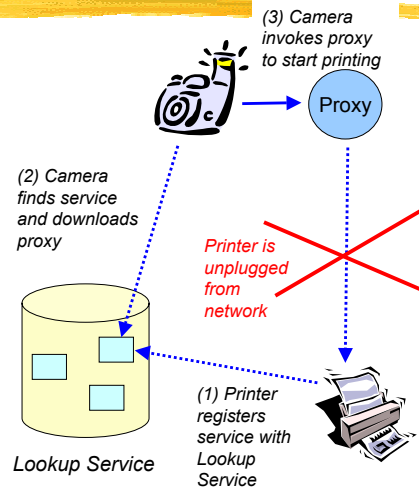
- ⌘ E.g., Camera runs out of batteries while print job being spooled
 - ☑ Camera holds access to the printer
 - ☑ Printer now unavailable forever to other clients



Fault Tolerance (2)

⌘ E.g., someone accidentally unplugs printer from network. Printer no longer responds to proxies.

- ☒ Printer is still registered with Lookup Service
- ☒ Clients will still erroneously attempt to use this printer



Solution: Leasing

⌘ Problem is "hold and wait"

- ☒ Never allow a client to obtain a service and hold it indefinitely
- ☒ Never allow a service to obtain a slot in the service lookup and hold it indefinitely

⌘ Leasing restricts how long a service can be held

Leasing

- ⌘ When a service is obtained, it is actually **leased** for a period of time
- ⌘ Before the lease expires, the client must renew the lease, or the service is no longer available to that client
- ⌘ Typical lease time: 5 seconds
- ⌘ Therefore, if client fails, eventually the lease will expire, and the service will be made available again

Leasing and Service Registration

- ⌘ Service registration also follows the lease approach
 - ☒ A registration with a Lookup Service has a fixed duration
 - ☒ Typically a lease of 5 seconds
 - ☒ If the lease is not renewed, the service will no longer be listed
 - ☒ Therefore, services cease to be listed if they are no longer available
 - ☒ If a service is temporarily unavailable, it can be re-registered when it returns

Self-Healing

- ⌘ Leasing leads to **self-healing**
- ⌘ That is, if the distributed system breaks, eventually leases will expire, and the system will return to a correct state

Jini

- ⌘ Primary features
 - ⌘ Discovery
 - ⌘ Lookup
 - ⌘ Leasing

Bluetooth

- ⌘ Wireless connection of small devices
- ⌘ E.g., PDA's, cell phones, ...
- ⌘ Range 10m – 100m
- ⌘ Bandwidth 57.6 kbps to 721 kbps

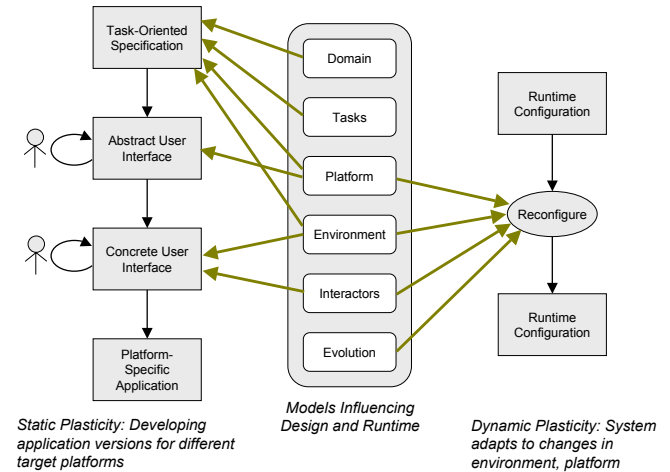
Bluetooth Protocols

- ⌘ Inquiry – find all nearby access points
- ⌘ Service discovery – determine what services available from access point
- ⌘ Establish application-level connection to access point to use service

UPnP

- ⌘ Target is home network
 - ☒ E.g., networking computers, printers, thermostat, lights, ...
- ⌘ Control point(s) maintain knowledge of what devices available
 - ☒ Clients search for control points via multicast
 - ☒ HTTP Multicast UDP (HTTPMU)
 - ☒ Control points communicate via HTTP UDP (HTTPU)
 - ☒ Devices described via XML

Plasticity in User Interfaces



Class Exercise

- ⌘ Architect a travel planner using Jini, Bluetooth
 - ☒ Supports finding best route, purchasing tickets, paying with credit card