An Auto-configuring Service Discovery System

Adam MacBeth
PhD Quals Talk
May 11, 2001
The Vision – Pervasive Computing

- Software configures itself automatically
- Applications run in any space on any device
  - Adapt to available hardware/software
- “Lego-computing”
  - Use functionality from multiple devices
- At the very least: find the nearest printer (Duh!)
Outline

- The Problem
- Background/Prior Work
- Our Architecture
- The Implementation
- Results
- Future Work
What Is Service Discovery?

- A way to find resources in a dynamic environment
- 2 Key Entities:
  - Client (e.g. Palm Pilot)
  - Service (e.g. Printer)
- What is a service?
  - Anything that offers an interface and makes its presence known
Architectures for discovery

- **Server-based (the 3rd party)**
  - Services are registered with a server
  - Clients query the server
  - Manual server configuration required

- **Server-less (peer-to-peer)**
  - Clients broadcast requests
  - Services respond directly
  - No server configuration
Early-binding Discovery (Lookup)

1. Client communicates with the Server.
2. The Server provides the necessary information.
3. The Client uses the information to connect to the Service.

Adam MacBeth - PhD Quals
Late-binding Discovery
Previous Systems

- **Server-based:**
  - Jini: type queries, leasing
  - INS: Late-binding, auto-configuring overlay, expressive language
  - Berkeley SDS: Scalability and security, XML for description

- **Peer-to-Peer:**
  - UPnP: Ease of configuration for home/office

- **Tension between ease of configuration (peer-to-peer) and features (server-based)**
Goals for Discovery in one.world

- A clean, integrated interface (novel)
- Support for late-binding (must use a server)
  - Push some logic to the server, simplify applications
- Automatic establishment of servers
  - Without a server, discovery doesn’t work!
  - Manual configuration is burdensome, especially in ad-hoc settings.
- Scale to local network
one.world Overview

- Support for programming pervasive applications
- Three abstractions:
  - Tuple (data)
  - Component (functionality)
  - Environment (grouping, isolation)
- Events
  - Tuples used for communication between components
  - Explicit import/export of event handlers
- Structured I/O
  - Use tuples for all forms of I/O (storage, network)
- All resources are leased
Remote Event Passing

- Network communication between event handlers
- Event handlers must be exported prior to use
- Result is RemoteReference
  - Host, Port, Unique reference for the handler object
- RemoteEvent includes:
  - Source and destination of type RemoteResource
Discovery Interface

- Integrates late-binding discovery with REP
- EXPORT: Export services with BindingRequest
  - Service = event handler + tuple descriptor
- LATE-BINDING:
  - DiscoveredResource destination sent to discovery;
    LocalizedResource destination sent to REP
  - DiscoveredResource includes a Query for a service
- LOOKUP: Early-binding discovery uses ResolutionEvents
REP code

RemoteReference ref =
    new RemoteReference(128.95.4.91, 5103,
    new Guid());

RemoteEvent remote =
    new RemoteEvent(this,null,ref,event);

request.handle(this,remote);
Discovery Code

DiscoveredResource disco =
    new DiscoveredResource(query);

RemoteEvent remote =
    new RemoteEvent(this, null, disco, event);

request.handle(this, remote);
System Architecture

- All one.world nodes run a DiscoveryClient component in the kernel.
- Server nodes run a DiscoveryServer.
- Client hides protocol details from application, but not errors.
Client Responsibilities

- **Export Services**
  - System-level leases
    - `RemoteReference` lease, server lease
  - Application-level lease
  - All leases are dependent on one another
- **Forward discovery requests to server**
  - If multiple servers exist, search them all
  - Retries not supported, errors propagated to application
DiscoveryServer

- Handle service registrations/renewals
  - Stored in registry with descriptor and lease
- Answer queries (late and early-binding)
- Advertise presence
  - Use known IP multicast address
  - Send RemoteReference to server
DiscoveryServer (cont.)

- Registry is implemented as an `ArrayList`
- Linear-time search
- `TupleFilter` used to match queries to descriptors
- Relaxed semantics for better performance
  - Once search is started, don’t restart
  - Application (early) or server (late) can retry
Server Election

- A variation on Distributed Consensus problem
  - Impossible in general case
- Use a heuristic to rank nodes (aka capacity)
- Exchange capacities, best node wins
- Election properties:
  - Selectivity – the best leader is chosen
  - Uniqueness – all nodes agree on leader
- Neither is very important in this case
  - Bounded election time IS important!
  - Unknown number of hosts
What makes a good heuristic?

- Server should be capable
  - Plenty of cycles, memory, bandwidth
- Server should be available
  - Prevent frequent elections
- Current heuristic: uptime + total memory
- Use IP address to break ties
- We need more experience
Election Algorithm

- If no server announcement is heard after 2 periods, send START message
- Upon hearing a START, set bound timer, and send CAPACITY message
- Receive capacities until timeout
- If local capacity == max received capacity
  Start DiscoveryServer
Election Duration

- All hosts need to exchange messages within time bound
  - Otherwise, multiple servers may be elected
  - At least one server is always elected
- total latency = (N * latency) + contention
- latency = propagation + transmit + queue
Multiple Servers

- Clients route requests to all available servers
- No cooperation between servers
- Attrition through capacity comparison
  - Goal of one server
Example - 10 Mb/s Ethernet

- Assume 256 hosts
- Max. propagation: 51.2 μs
- Transmit (for 700 bytes)
  = 5600 bits / 10 Mb/s = 560 μs
- Contention: Experimental Ethernet data shows linear increase in delay with # hosts [Boggs88]
Example (cont.)

- For 256 hosts/768 byte packets:
  - Average time to send: 200 ms
  - Std. Dev.: 665 ms
  - Assuming 99% within 3 Stdev: 200 ms + (3 * 665 ms)
    = 2195 ms

- Total latency = 256 x (51.2 µs + 560 µs) + 2195 ms
  = 2351 ms for full completion

- Reasonable for human time-scale operation
Election Scalability

- Ethernet is feasible, what other technologies?
- Bluetooth is slower (1 Mb/s), but fewer hosts
- Wide-area probably not feasible
  - Delay increases
  - Number of hosts increases
  - Multicast support?
  - 1 s = 1000 hosts X 1 ms = 10 hosts X 100 ms
Server Performance

- Tested with 1 server, 3 clients, early-binding
- Increased request rate until latency became unreasonably large
- Is registry the bottleneck?
Server Performance - Latency

![Graph showing latency vs request rate]

- 1 registry entry
- 100 registry entries
- 1000 registry entries
Server Performance – Std. Dev.
Registry Performance

Constant time per registry entry -> Exponential decrease per second
Future Work

- Usefulness of local discovery with permanent server (works now)
- Real apps
- Evaluation of heuristics
- Evaluation of performance in different network environments
Conclusion

- Server election resolves tension between:
  - Ease of configuration
  - Server-based features
- Election shows reasonable responsiveness
- Server performance may need improvement
Acknowledgements

- *one.world* team: Robert Grimm, Janet Davis, Eric Lemar, Steve Swanson
- Advisors: Gaetano Borriello, Steve Gribble