Using Behavior Templates To Design Remotely Executing Agents For Wireless Clients

Eugene Hung and Joseph Pasquale
Dept. of Computer Science and Engineering
University of California, San Diego
Motivating Scenario
Problem

- Wireless clients are diverse, varying in:
  - Network Bandwidth
  - Network Reliability
  - Graphical Display
  - User requirements

- Web services are not flexibly customizable
  - Scenario: Wireless E-Commerce
Outline

- Previous Approaches
- Design Goals
- Solution – ReAgents
- Architecture
- Behavior Library
- Experiments
- Conclusions
Previous Approaches

- Network-based
  - Active Networks
- Server-based
  - WAP (Wireless Application Protocol)
- Intermediary-based
  - Web Proxies
  - Mobile Agents
Design Goals

- The ideal customization solution will be:
  - **Flexible** enough to handle user needs
  - **Transparent** to servers (deployable)
  - **Easy** to program and understand
  - **Efficient** when used
Solution: ReAgents

- **ReAgents – Remotely Executing Agents**
  - Contain Customizing Logic (CL)
  - “One-shot” mobility to ReAgent host
  - Behavior-based development
Behavior: Filter

- Reduces server data to client specifications
- **Customizing Logic:** Data-reducing algorithm
- **Sample Application:** Low-bandwidth filtering
Behavior: Encoder

- Transforms data for reverse-transform at client
- Customizing Logic: Reversible data transformation
- Sample Application: Encrypted transfer for privacy
Behavior: Monitor

- Polls object on server until desired state is reached, then reacts to state change
- **Customizing Logic**: Object state test and reaction
- **Sample Application**: Custom stock trader
Behavior: Cacher

- Bypasses server communication by storing frequently accessed server data close to client
- **Customizing Logic:** Cache management policy
- **Sample Application:** Resource-poor client caching
Behavior: Collator

- Sends same request to many servers and merges results
- **Customizing Logic:** Results-collation algorithm
- **Sample Application:** Shopping comparison agent
Experiment

- File transfer time reduced 30-75%
Experimental Results

- ReAgent overhead is **low**
- Overhead **scales well** as file size increases
Conclusion

- ReAgents customize for wireless clients
  - Flexibly
    - Customizing logic
  - Transparently
    - Server is bypassed
  - Easily
    - One-shot mobility simplifies security and semantics
    - Behaviors provide structured, patterned development
  - Efficiently
    - Results show good performance and scalable overhead
Questions?
ReAgent Architecture
Usage

- ReAgent created by chaining Behaviors
- Behaviors created by instantiating with CL
- Example: Custom Stock Trader

ReAgent reagent = new ReAgent();
Behavior m = new Behavior ("Monitor", "MyPriceWatch.class");
Behavior t = new Behavior ("Filter", null);
reagent.addBehavior (m, null);  // (no converter for monitor)
reagent.addBehavior (t, "GenerateStockBuyRequest.class");
reagent.launch("middleman.org");
reagent.process("GET http://stock.org/viewprice.cgi/?p=GOGL");