Hydro: Language Support for Loosely Coupled Components

Keunwoo Lee
University of Washington

Joint work with:
Craig Chambers (Univ. of Washington)
Anthony LaMarca (Intel Research Seattle)
Motivation

Ubiquitous computing

\sim \text{ "large, heterogeneous, highly dynamic distributed systems"}

- System constituents frequently change
  - Example: home sensor network

- System must remain correct as it evolves

- Can programming systems technology help the programmer with this task?
Context: Rain

- Messaging and service discovery infrastructure from Intel Research
- Semistructured message format
  - Asynchronous XML messaging over HTTP
- Current state of system:
  - Programmers use library calls to assemble, receive, and dispatch messages
  - Awkward; error-prone; no static checking
Proposal: Hydro (phase 1)

• Can we design a language that
  • looks nicer than library calls (easy)
  • supports message dispatching
  • makes some type safety guarantees

and

• retains flexibility of semistructured messages

  flexibility of interface types aids evolution
Messaging in Rain

• Typical message:

```
<message>
  <sender><serviceid id='...'/></sender>
  <recipient><serviceid id='...'/></recipient>
  <body>
    <database>plants</database>
    <commandType>execute</commandType>
    <command>CREATE TABLE ...</command>
  </body>
</message>
```
Messaging in Rain (1)

```
<message>
  <sender><serviceid id='...'/></sender>
  <recipient><serviceid id='...'/></recipient>
  <body>
    <database>plants</database>
    <commandType>execute</commandType>
    <command>CREATE TABLE ...</command>
  </body>
</message>

• Handler code:

```java
void process(Message m) {
    if (m.getBody().containsChildNamed("database")
        && m.getBody().containsChildNamed("commandType")
        && m.getBody().containsChildNamed("command")) {
        String cmd =
            m.getChildElement("command").getTextChild();
    ... 
    }
}
```
Messaging in Rain (2)

• What if we have a Java/XML serialization library?

```java
void process(Message m) {
    MessageObject o = deserializeXML(m.getBody());
    if (o instanceof ExecuteMessage) {
        ExecuteMessage d = (ExecuteMessage)o;
    } else ...
}
```

• Looks nicer... but brittle: depends on Java type system

• What happens if later component adds another tag?

```xml
<body>
    <database>plants</database>
    <commandType>execute</commandType>
    <command>CREATE TABLE ...</command>
    <auditKey>0x27328732</auditKey>
</body>
```

• Still no support for dispatch or static checking
Hydro: dispatch via patterns

• Programmers want to dispatch **events** (messages) to **behavior** (methods)

• Solution: **a pattern language** that compactly specifies
  • What messages can a method handle?
  • What parts of the message is programmer interested in?

• Make patterns extensible using **structural subtyping**
  • ...should get flexibility for free.
Handler example

public service class PlantStore {
    handler #database[d@String],
        #commandType["execute"],
        #command[cmd@String] {
            this.getDb(d).execute(cmd);
        }
    handler #database[d@String],
        #commandType["query"],
        #command[cmd@String] {
            QueryResult q = this.getDB(d).query(cmd);
        }
}

On receipt, messages dispatch to most-specific matching handler
Research problems

• Short term (phase 1):

  • How flexible can we make the pattern language while retaining tractable checking?

  • How can we smoothly integrate the pattern language into a practical language such as Java?

• Long term: How can we provide advanced support for dynamic evolution of component interfaces and behavior?