# Toward Dynamic Application Protocols in Heterogeneous Distributed Computing Systems

Dennis Patrone
Bina Ramamurthy
Department of Computer Science and Engineering

University at Buffalo
The State University of New York

**GMU-AFCEA Symposium 2009– Critical Issues in C4I**
**National Conference Center, Lansdowne, VA 19-20 May 2009**

## Outline

- Motivation and Background
- Overview
- Code Examples
- Summary & Future Work

2

# Motivation: Efficient & Flexible SOA

- Service Oriented Architectures
  - Goal: Agility (from a "business" perspective)
  - Approach: Loose coupling of services and clients
  - State-of-the-art: Web Services
    - Achieve environment independence (language, platform, operating system) by standardizing network protocols and using auto-generated skeletons (server) and proxies (client) to facilitate communication

3

# Camera Example

**Simplified Distributed System Technology Standardized Network Protocol**

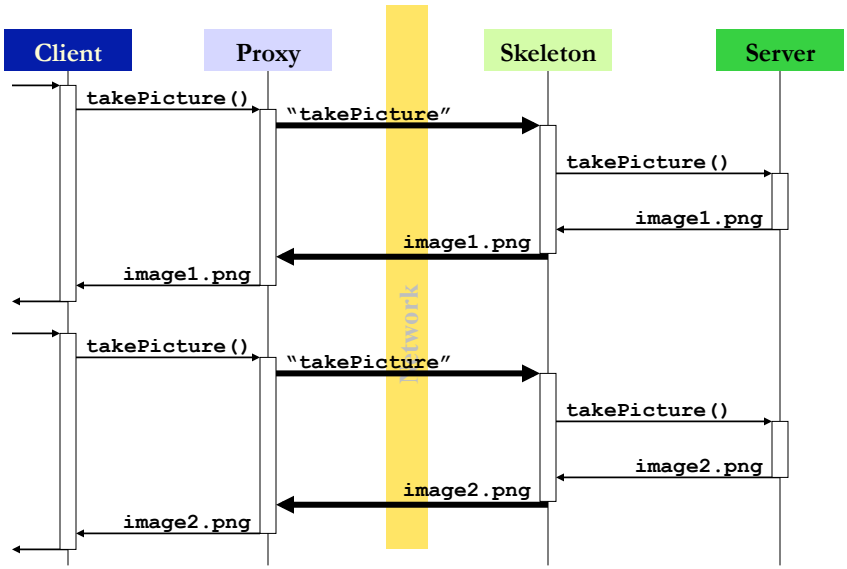
• Client will send method name as a string, followed by any parameter values in order of definition in the interface.

• Server will respond with any return value, or an acknowledgement upon completion of service.

```
<service name="Camera">
   <operation name="takePicture">
     <input element="void" />
     <output element="PNG" />
   </operation>
</service>
```

4

# Camera Service 1.0




5

# Motivation: So what's wrong with that?

- All protocols have biases introduced at definition time
  - Based on as
  - The original
  - The assump
  - The environ
    original assu
  - Standards n

ns

**Camera Example:**

**What if the camera's field of view doesn't change often yet clients poll for images frequently??**

**If the network protocol is standardized, we must upgrade the interface (and therefore all existing services & clients) to exploit that knowledge…**

*How can we i … y" situations?*

*How can we tune protocols to specific situations yet maintain loose coupling?*

6

## Solution: Make the Proxy Dynamic!

- Dynamically upload the service proxy to the client at runtime
  - Allows the service to change the logic on both sides, meaning service implementations can tune network protocols to their own requirements and expectations– or even "teach" clients how to perform the service locally!
- To share code dynamically, we need to agree on a standard environment
  - *Jini anyone?*

7

## Isn't that environment just…

- The Java Virtual Machine (JVM)? Or the .NET platform?
- Both the JVM and .NET utilize virtual *machines*. So while they abstract *real* machines, they do not play nicely with each other (or other virtual machines)
  - One can not execute compiled Java bytecode in a .NET environment, nor .NET's Common Intermediate Language (CIL) instructions on a JVM!
- In large, cross-enterprise distributed systems we want (need!) "environment independence"…

8

## Application Logic Markup Language

- Share the proxies' *application logic* – not their "*compiled logic*" (class files!)– in XML!
  - Source code is just ***metadata***
    - That's *exactly* what XML captures!
    - Source just describes how to manipulate application data
  - Capture ***actionable knowledge*** in XML instead of inferred semantics from XML tag names and (one-off) XSD's
  - Can be understood by ***any environment*** (hardware, OS, programming language) capable of handling XML
  - A general purpose "application logic" language: *Application Logic Markup Language (ALML)*

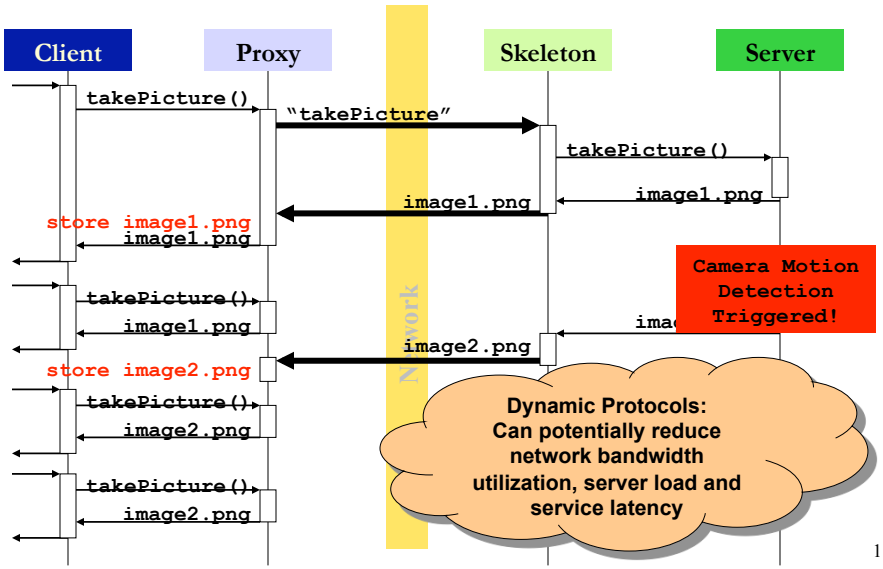
9

## Wait one minute…

- Did you just say we need ***language independence*** and that we can achieve it by agreeing on ALML as a ***common language***?!?
- Ultimately we have to agree on ***something*** or there can be no useful communication
- Using XML opens the possibilities to the ***widest set*** of potential participants
- Any environment that can parse a string can ultimately understand an XML document
  - And most (all?) contemporary, network-aware programming languages already have some level of support for parsing XML documents

10

# Camera Service 2.0: More Efficient…




11

# … and Adaptable!

- We have turned a "***pull***" into a "***push***" protocol without changing the service interface or client code
  - We could even supply both protocols and select the most appropriate one based on current usage patterns
- Previously-deployed Camera services ***do not break***
  - They just work a little less efficiently
- Previously-deployed Camera clients ***do not break***
  - They just download the new proxy definition when they encounter a new Camera
- For algorithmic-based services (i.e., those that are strictly computational) we could provide the ***entire service*** as the proxy and client could execute it locally

12

# ALML Overview

- XML schema and related specification
- Object oriented
- Provides standard OO capabilities
  - Packages / modules
  - Classes / interfaces
  - Members / methods
  - Inheritance
  - Visibility control
  - Flow control (if, for, while, do)
  - Expressions
- Java-based reference implementation

13

# ALML Method Definition

```
<xs:complexType name="methodDef">
    <xs:sequence>
        <xs:element name="type" type="typeDef" />
        <xs:element name="parameters" type="parametersDef" />
        <xs:element name="throws" type="typeDef"
                minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="block" type="blockDef" />
    </xs:sequence>

    <xs:attribute name="name" type="identifierDef" use="required" />
    <xs:attribute name="visibility" type="visibilityDef" use="required" />
    <xs:attribute name="static" type="xs:boolean" />
    <xs:attribute name="final" type="xs:boolean" />
    <xs:attribute name="abstract" type="xs:boolean" />
</xs:complexType>
```

14

# ALML Statements

```
<xs:complexType name="statementDef">
    <xs:choice>
        <xs:element name="variable" type="variableDef" />
        <xs:element name="expression" type="expressionDef" />
        <xs:element name="if" type="ifDef" />
        <xs:element name="for" type="forDef" />
        <xs:element name="while" type="whileDef" />
        <xs:element name="try" type="tryDef" />
        <xs:element name="return" type="expressionDef" />
        <xs:element name="throw" type="expressionDef" />
        <xs:element name="noop" type="empty" />
        <xs:element name="assign" type="assignDef" />
    </xs:choice>
</xs:complexType>

<xs:complexType name="ifDef">
    <xs:sequence>
        <xs:element name="condition" type="expressionDef" />
        <xs:element name="then" type="blockDef" />
        <xs:element name="elseIf" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType><xs:sequence>
                <xs:element name="condition" type="expressionDef"/>
                <xs:element name="then" type="blockDef"/>
            </xs:sequence></xs:complexType>
        </xs:element>
        <xs:element name="else" type="blockDef" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
```

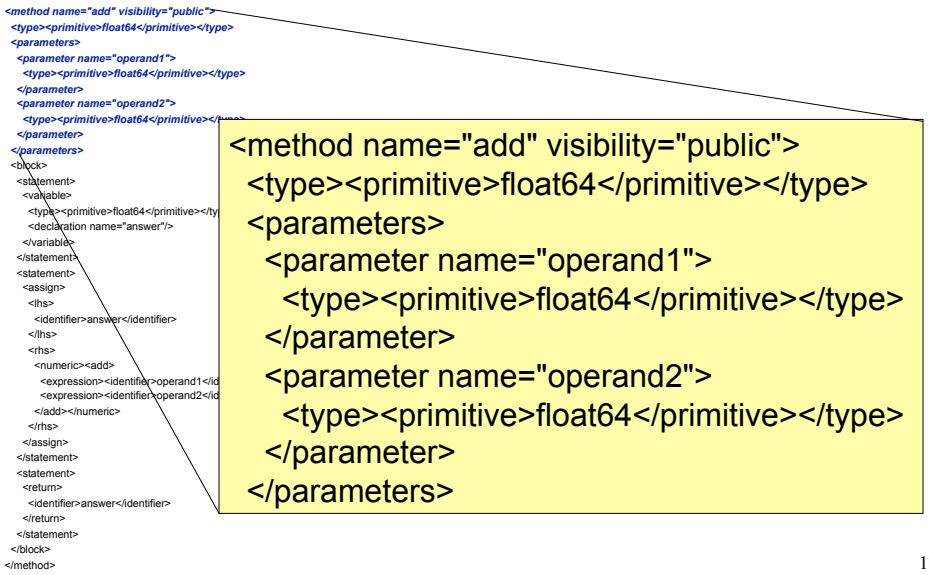
15

# double add(double v1, double v2)

```
<method name="add" visibility="public">
 <type><primitive>float64</primitive></type>
 <parameters>
  <parameter name="operand1">
   <type><primitive>float64</primitive></type>
  </parameter>
  <parameter name="operand2">
   <type><primitive>float64</primitive></type>
  </parameter>
 </parameters>
 <block>
  <statement>
   <variable>
    <type><primitive>float64</primitive></type>
    <declaration name="answer"/>
   </variable>
  </statement>
  <statement>
   <assign>
    <lhs>
     <identifier>answer</identifier>
    </lhs>
    <rhs>
     <numeric><add>
      <expression><identifier>operand1</identifier></expression>
      <expression><identifier>operand2</identifier></expression>
     </add></numeric>
    </rhs>
   </assign>
  </statement>
  <statement>
   <return>
    <identifier>answer</identifier>
   </return>
  </statement>
 </block>
</method>
```

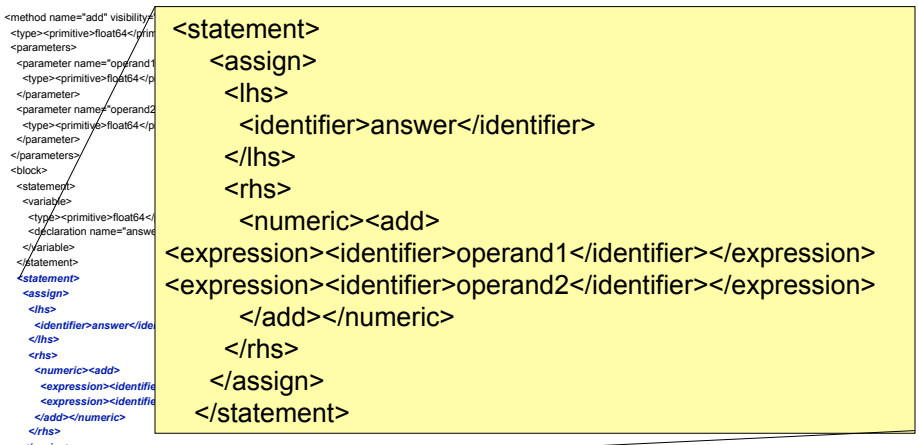
16

# double add(double v1, double v2)




17

# double add(double v1, double v2)




18

## Summary

- We have shown that XML can be used to capture standard object oriented programming concepts ***in an actionable way***
- Describing application logic in XML should allow for sharing that logic among ***heterogeneous*** systems
- ALML is a first step toward an efficient and flexible heterogeneous SOA that can
  - Adjust network protocols at runtime based on the current situation to "optimize" key system parameters such as network bandwidth consumption, sever load, or service latency
  - Enable services to define multiple protocols and dynamically choose the appropriate one– at runtime– for the current situation
  - Teach remote systems to perform certain services for themselves while maintaining proper "definition authority"

***This is a work-in-progress!***

19

## Future Work

- Extend reference implementation to other programming languages (C#, Ruby, …)
- Handling  system "built-in" libraries
- Handling 3rd party libraries
- Native system interactions
- Find and bind SOA infrastructure
- Security, security, security!

20

Thanks!

Questions? Comments?

dpatrone@cse.buffalo.edu
bina@cse.buffalo.edu

University at Buffalo
The State University of New York

21

22