1. Many-to-Many Invocation:
   A New Object Oriented Paradigm to build Software Infrastructures for Serverless Ad Hoc Collaborative Systems

Hans-Peter Bischof  Alan Kaminsky
hpb@cs.rit.edu  ark@cs.rit.edu

°RIT

A copy of this talk can be found at
°http://www.cs.rit.edu/~hpb/Talks/M2mic_OS/index.html
2. Current Infrastructure

2.1. Current Hardware Infrastructure
2.2. Administration

- ip addresses
- router in place
- routing tables
- ... and more
2.3. Current Software Infrastructure
2.4. Typical Communication Pattern

- one to one: ip addresses and ports
- one to many: broadcast/multicast
- many to a many: multicast and channels
2.5. Typical Ways to Communicate

- Sending data
- Invoking methods
2.6. Remote Method Invocation: The Idea
2.7. SenderProxySource Code

....
    private static final long INTERFACE_ID = 32;
    private final long String__METHOD_1 = 0;
....

public void knock( String _0) {
    try {
        ByteArrayOutputStream aS = new ByteArrayOutputStream(1024);
        ObjectOutputStream p = new ObjectOutputStream(aS);
        p.writeObject(_0);
        p.close();
        RmiPacket aRmiPacket = new RmiPacket( INTERFACE_ID,
                                               String__METHOD_1,
                                               aS.toByteArray() );
        netWorkCommEndPoint.sendDataToMany(aRmiPacket);
        e.printStackTrace();
        System.exit(1);
    }
}
2.8. ReceiverProxySource Code

....

private static final long INTERFACE_ID = 32;
private final long String__METHOD_1 = 0;

aRmiPacket = netWorkCommEndPoint.getData();
ByteArrayInputStream iS = null;
ObjectInputStream ip = null;
iS = new ByteArrayInputStream(aRmiPacket);
ip = new ObjectInputStream(iS);

if (aRmiPacket.getMethodId() == String__METHOD_1) {
    
    String __0 = (String)ip.readObject();
    aThisClass.knock(__0);
}

...

• rmic generates the Sender and ReceiverProxys
2.9. A typical Client-Server Architecture
2.10. A typical Client-Server Architecture Problem

- If the server goes away, the whole application dies, even though the clients can communicate directly.
- Helpful: No central servers!
3. Different Kind of Infrastructure

3.1. Different Kind of Hardware Infrastructure
3.2. Different Kind of Environment: Ad Hoc Network

- Play at the theater is a little boring
- Meetings: play backgammon/poker/chatting
- NFL Superbowl XXXVIII
3.3. A typical Serverless Architecture
3.4. Different Kind of Software Infrastructure

- To simplify programming
  - Object oriented abstraction of many-to-many communication
  - Broadcast method invocations: M2MI

- To simplify deployment
  - No proxy compilers, codebase servers, activation daemons, ...
  - Automatic proxy synthesis

- To simplify operation and administration
  - No network addresses, ad hoc routing protocols, ...
4. M2MI: A New Paradigm for Ad Hoc Collaborative Systems
   • M2MI provides an object-oriented method call abstraction based on broadcasting.
   • M2MI-based systems do not require central server
   • M2MI-based systems do not require network administration
   • M2MI simplifies system deployment by eliminating the need for always-on application servers
   • M2MI is well-suited for an ad hoc networking environment where central servers may not be available
4.1. References

- Omnihandle: Refers to all objects that implement an interface

- Multihandle: Refers to a group of objects that implement an interface

- Unihandle: Refers to one object that implements an interface
4.2. Using Omnihandles

- Export remote objects
  
  ```java
  M2MI.export(a, Foo.class);
  ```

- Get an omnihandle
  
  ```java
  Foo allFoos = (Foo) M2MI.getOmnihandle(Foo.class);
  ```

- Invoke a method on the omnihandle
  
  ```java
  allFoos.y();
  ```
4.3. Using Multihandles

- Get a multihandle
  
  Foo someFoos = (Foo)M2MI.getMultihandle(Foo.class);

- Attach objects
  
  someFoos.attach(a);

- Invoke a method on the multihandle
  
  someFoos.y();
4.4. Using Unihandles

- Export remote object and get unihandle
  ```java
  Foo b_Foo = (Foo)M2MI.getUnihandle(b, Foo.class);
  ```

- Invoke a method on the unihandle
  ```java
  b_Foo.y();
  ```
4.5. Characteristics of M2MI Invocations

- M2MI is an object oriented abstraction of many-to-many communication.
- Semantics of M2MI:
  - Methods may have arguments
  - Objects passed by copy (object serialization)
  - Handles give pass-by-reference
  - M2mi methods can not return a value.
  - M2mi methods can not throw an exception
  - Parameters are passed as pass-by-value.
  - Method calls are non-blocking
5. M2MI Based Application

5.1. The Idea for a Chat Application

![Chat Application Diagram]

```java
allChats.putMessage("Hello there");
allChats.putMessage("Hi y'all");
allChats.putMessage("Greetings folks");
```
5.2. The Chat Interface

```java
public interface Chat {
    public void putMessage(String line);
}
```
5.3. Chat Object Source Code

```java
public class ChatObject implements Chat {
    private String myUserName;
    private Chat allChats;

    public ChatObject (String theUserName) {
        myUserName = theUserName;
        M2MI.export(this, Chat.class);
        allChats = (Chat)M2MI.getOmnihandle(Chat.class);
    }
    public void send(String line) {
        allChats.putMessage(myUserName + " > " + line);
    }
    public void putMessage(String line) {
        myChatFrame.addLineToLog (line);
    }

    public static void main (String args []) {
        String input;
        ChatObject aChatObject = new ChatObject( args[0] );
        while ( ( input = readFromTerminal() ) != null )
            aChatObject.send(input);
    }
}
```
6. Service Discovery
   • Service discovery today can done for example with JINI
   • Requires a server
   • How can a service discovery work in an ad hoc network?
   • There are no servers in an ad hoc network.
6.1. The Idea

```text
printDiscovery.request (theClient);
theClient.report (c_Printer, "C");
theClient.report (a_Printer, "A");
theClient.report (b_Printer, "B");
c_Printer.print (theDocument);
```
6.2. The Interfaces

public interface PrintDiscovery {
    public void request(PrintClient client);
}

public interface PrintClient {
    public void report(PrintService printer, String name);
}

public interface PrintService {
    public void print(Document doc);
}
6.3. Other M2MI Applications

- Conversations
  Conversations in quiet spaces, conversations in noisy spaces, . . .

- Groupware
  Presentations, whiteboard, note taking, file sharing, document authoring, calendar scheduling, . . .

- Sensor networks
  Video surveillance, medical monitoring, battlefield intelligence, . . .

- Middleware frameworks
  - Shared tuple spaces, . . .

- Multiplayer Games!
7. M2MI Architecture

7.1. Layers

- Application Layer
- Invocation Layer
- Messaging Layer
- Data Link Layer
- Physical Layer

\{ M2MI, M2MP \}

\{ Ethernet, 802.11, Bluetooth, ... \}
7.2. Software Architecture
8. Status

- Initial version of M2MI written in Java
- Tested on desktop hosts
- Some performance and throughput measurements done
- Uses UDP/IP for transport
- Another version uses Ethernet raw sockets for transport
- Several M2MI-based collaborative applications developed Chat, IM, whiteboard, calendar, file sharing, tuple space
8.1. In Progress

- M2MI monitoring API
  - Observe and debug M2MI invocations flowing through the network

- M2MI security
  - Confidentiality, participant authentication, service authentication
  - Serverless techniques: Zero knowledge proofs, . . .
  - Elliptic curve based techniques

8.2. Security in Ad-Hoc Networks

- How do you achieve security in an ad-hoc network?
- Major Challenge: no servers.

- Problems to solve:
  * Confidentiality
  * Data Integrity
  * Participant Authentication
8.3. Interactive Zero Knowledge Proofs in General

- Interactive Zero Knowledge Proofs allow one party to prove its knowledge of a secret to another party without ever revealing the secret itself.

- It is useful for interactive proofs to have the following properties:
  * Completeness. The verifier always accepts the proof if the fact is true and both the prover and the verifier follow the protocol.
  * Soundness. The verifier always rejects the proof if the fact is false, as long as the verifier follows the protocol.
  * The verifier learns nothing about the fact being proved (except that it is correct) from the prover.
  * The verifier cannot even later prove the fact to anyone else.
8.4. Ali Baba’s Cave

- Alice wants to prove to Bob that she knows the secret words that will open the portal at R-S in the cave,
- but she does not wish to reveal the secret to Bob.
8.5. Zero Knowledge Proofs in Detail

The general protocol works as follows:

- There is a prover who knows the secret and a verifier who wishes to know if the prover knows the secret.
- The prover proposes a "hard" problem to the verifier.
- The verifier asks one or more questions about a hard problem until she or he is convinced that the prover really knows the answer to the problem.
- Classic examples of the "hard" problem include factoring the product of large primes, graph isomorphism, and discrete logarithms.
- Problems that are well-suited for ZKPs are typically NP-complete, yet verifiable in polynomial time.
- However, a close analysis of specific ZKP algorithms suggests that many ZKPs are not well-suited for an ad hoc environment, because of CPU/memory limitations.
- RSA
8.6. Algorithms

- ZKP’s for Quadratic Residues

Given:

\[ p, q \in \mathbb{P}. \]
\[ n = p \times q \]

\( x \) is a quadratic residue of \( n \) if and only if there exists an \( y \) such that:
\[ x = y^2 \mod n. \]

\[ Z_n^* = \{ i \mid i \in \mathbb{N} \text{ and } i < n \text{ and } i \text{ and } \gcd(n, i) = 1 \} \]

1. Repeat:
   - Repeat the following steps \( \log_2 n \) time
     a) Peggy chooses \( v \in Z_n^* \) at random and then she computes: \( y = v^2 \mod n. \)
     Peggy sends \( y \) to Victor.
     b) Victor chooses \( i \in \{ 0, 1 \} \) at random and sends \( i \) to Peggy.
     c) Peggy computes: \( y = u^i \times v \mod n \), where \( u \in Z_n^* \) and is a solution to \( u^2 \equiv x \mod n \).
     Peggy sends \( z \) to Victor.
     d) Vic checks that:
     \[ x^2 \equiv x^i \times y \mod n \]

2. Accept:
   - Victor accepts Peggy’s proof, that \( x \) is a quadratic residue of \( n \), if step (d) is verified \( \log_2 n \) rounds.

8.7. Other Algorithms

- ZKPs for Graph Isomorphism
- ZKP’s for Graph 3-coloring
8.8. Future Plans

- Go small and wireless
  - Port M2MI and M2MP to small mobile devices
  - Test with wireless networking

- Push M2MP into the kernel

- Develop lots of M2MI-based applications in a variety of domains

- Devise reusable design patterns and class libraries for M2MI-based collaborative applications
9. Thanks
Jim Waldo inspired the idea for M2MI when he said: Everyone that’s out there, call this method" during a discussion about M2MP.
10. Questions

From:  *http://www.myfonts.com*
11. Publications

2002 Alan Kaminsky and Hans-Peter Bischof  
"Many-to-Many Invocation: A New Object Oriented Paradigm for Ad Hoc Collaborative Systems"  
OOPSLA 2002, Seattle, Washington, USA, November 2002

2002 Hans-Peter Bischof and Alan Kaminsky.  
"Many-to-Many Invocation: A new framework for building collaborative applications in ad hoc networks."  
CSCW 2002 Workshop on Ad Hoc Communication and Collaboration in Ubiquitous Computing Environments,  
New Orleans, Louisiana, USA, November 2002.
12. Java Reliable Multicast System
This research was supported by a grant from Sun Microsystems.
The JRMS project is completed.
12.1. Overview of the Problem


- Create a network service which enables building multicast applications that distribute information over IP networks.
12.2. Aspects of Reliable

- What are the limits for reliability?
  - What happens if a receiver joins a channel late?
  - Is it possible to recover under all circumstances (pruning a receiver)?
  - What happens if a receiver can not keep up with the load?
12.3. JRMS Architecture
12.4. Repair Heads

- What is happening, if a receiver misses a packet?
- Does the TCP/IP idea work?
- A repair tree helps.
12.5. Throughput Problem

Graphical Representation of the Throughput

| Data sent | Bytes   | 140000000  |
|          | KBytes  | 136718.750 |
|          | MBytes  | 133.514    |
|          | GBytes  | .130385     |
12.6. Congestion Control Algorithm: Window Size

Every receiver has received this packet

last packet which can be send
12.7. Congestion Control Algorithm: Sending Speed

Every receiver has received this packet

last packet which can be send
12.8. The Result

12.9. What was Done at RIT

- Stress test environment
- I was 3 times in Boston for a longer period of time to work with the team on JRMS.
- All tests have been done at RIT
- Two different lectures have been given in this area.
- Two researcher from SunLabs gave a talk in my lecture.
12.10. Publications

Authors: Dah Ming Chiu, Miriam Kadansky, Joe Provino, Joseph Wesley, Hans-Peter Bischof and Haifeng Zhu
The paper has been accepted.

2001  "A Congestion Control Algorithm for Tree-based Reliable Multicast Protocols",
Technical Report, Sun Microsystems, Report Number: TR-2001-97,
http://research.sun.com/technical-reports/2001/,
Authors: Dah Ming Chiu, Miriam Kadansky, Joe Provino, Joseph Wesley, Hans-Peter Bischof and Haifeng Zhu

2001  "JRMS Stress Test Environment ", only published on the web, download from
http://www.experimentalstuff.com/Technologies/JRMS/
Author: Hans-Peter Bischof

2001  "JRMS Tutorial", only published on the web, download from
http://www.experimentalstuff.com/Technologies/JRMS/
Authors: Joe Binder, Hans-Peter Bischof, Jonathan Coles, Damian Eads, Collin Fagan, Jeffrey Myers
13. Questions

From: "http://www.myfonts.com"
14. Contact Information
The Anhinga Project
http://www.cs.rit.edu/~anhinga/

Hans-Peter Bischof
http://www.cs.rit.edu/~hpb/
hpb@cs.rit.edu

Alan Kaminsky
http://www.cs.rit.edu/~ark/
ark@cs.rit.edu