1. Many-to-Many Invocation:
A New Object Oriented Paradigm to build Software Infrastructures

1.1. Current Hardware Infrastructure
1.2. Administration
• ip addresses
• router in place
• routing tables
• ... and more
1.3. Current Software Infrastructure

129.21.30.26
nfs server

129.21.30.99
web server

Router

mount

print server

web/nfs
1.4. Typical Communication Pattern

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

- Sending data
- Invoking methods
1.6. A typical Client-Server Architecture
1.7. 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!
1.8. Different Kind of Infrastructure

1.9. Different Kind of Hardware Infrastructure
1.10. Different Kind of Environment: Ad Hoc Network

- Play at the theater is a little boring
- Meetings: play backgammon/poker/chatting
- NFL Superbowl XXXVIII
1.11. A typical Serverless Architecture
1.12. 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, . . .
1.13. 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
1.14. 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
1.15. 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();
  ```
1.16. Using Multihandles

- Get a multihandle
  
  ```java
  Foo someFoos = (Foo)M2M.I.getMultihandle(Foo.class);
  ```

- Attach objects
  
  ```java
  someFoos.attach(a);
  ```

- Invoke a method on the multihandle
  
  ```java
  someFoos.y();
  ```
1.17. 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();
  ```
1.18. 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
1.19. M2MI Based Application

1.20. The Idea for a Chat Application
1.21. The Chat Interface

```java
public interface Chat {
    public void putMessage(String line);
}
```
1.22. 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);
    }
}
```
1.23. 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 sever in an ad hoc network.
1.24. The Idea

printDiscovery.request (theClient);

theClient.report (c_Printer, "C");

theClient.report (a_Printer, "A");

theClient.report (b_Printer, "B");

c_Printer.print (theDocument);
1.25. The Interfaces

```java
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);
}
```
1.26. 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!
1.27. M2MI Architecture

1.28. Layers

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

M2MI

M2MP

Ethernet, 802.11, Bluetooth, ...
1.29. Software Architecture
1.30. 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
1.31. 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
1.32. 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