Distributed Programming in Java
Distribution (5)

RPC-style middleware

- Disadvantages:
  - Strongly coupled
  - Synchronous
  - Limited extensibility

- Advantages:
  - Transparency
  - Type safety

Space-based middleware

- Based upon tuple spaces
- A tuple space is an implementation of the associative memory paradigm for parallel/distributed computing
- It provides a repository of tuples that can be accessed concurrently.
- Producers post their data as tuples in the space, and the consumers then retrieve data from the space that match a certain pattern. This is also known as the Blackboard metaphor.
- Tuple spaces were the theoretical underpinning of the Linda language developed by David Gelernter and Nicholas Carriero at Yale University.
Space-based Design

• Requires design of distributed data structures and distributed protocols that operate over them.
• Distributed data structure is made up of multiple objects stored in one or more spaces.
  • E.g., ordered list of items represented by a set of objects, each of which holds the value and position of a single list item.
• Using collection of objects in shared space allows multiple processes to concurrently access and modify the data structure.

JavaSpaces I

• JavaSpaces is a service specification providing a distributed object exchange and coordination mechanism (which may or may not be persistent) for Java objects.
• It can be used to store the system state and implement distributed algorithms.
• In a JavaSpace all communication partners (peers) communicate by sharing state.
• Using JavaSpaces, distributed applications are modeled as a flow of objects between participants, which is different from classic distributed models such as RMI.

JavaSpaces II

• Achieves scalability through parallel processing.
• Provides for reliable storage of objects while reducing the complexity of traditional distributed systems.
• Processes perform simple operations:
  • Write new objects into a JavaSpace,
  • Take objects from a JavaSpace, or
  • Make copies of objects the JavaSpace.
• JavaSpaces is part of Jini technology.
JavaSpaces Architecture

Application components (or processes) use the persistent storage of a space to store objects and to communicate.

The components coordinate actions by exchanging objects through spaces; the objects do not communicate directly.

Processes interact with a space through a simple set of operations…
JavaSpace Primary Operations

- write(): Writes new objects into a space
- take(): Retrieves objects from a space
- read(): Makes a copy of objects in a space
- notify(): Notifies a specified object when entries that match the given template are written into a space

JavaSpaces Technology Application Model

- JavaSpaces service holds entries, each of which is a typed group of objects expressed in a class that implements the interface `net.jini.core.entry.Entry`.
- Once an entry is written into a JavaSpaces service, it can be used in future look-up operations.
- Looking up entries is performed using templates, which are entry objects that have some or all of their fields set to specified values that must be matched exactly. All remaining fields, which are not used in the lookup, are left as wildcards.

JavaSpace Entry Details

- Entries in a JavaSpace are simple Java Objects that follow a few simple rules:
  - All data persisted in the space must be exposed in public fields.
  - The Entry interface must be implemented.
  - This is a marker interface, requiring no methods to conform to the interface contract.
  - Objects must be used for the properties (i.e., no primitive fields.)
Simple Server Example

```java
public class Server {
    public static void main(String[] args) {
        // Create the server object
        Server server = new Server();
        server.start();
    }
}
```

Fields must be public

Simple Client Example

```java
public class Client {
    public static void main(String[] args) {
        // Create the client object
        Client client = new Client();
        client.connect();
    }
}
```

JavaSpaces Technology Application

[Diagram of JavaSpaces Technology Application]
JavaSpaces: Multi-user Chat System

- All the messages that make up the discussion are written to a space that acts as a chat area.
- Participants write message objects into the space, while other members wait for new message objects to appear, then read them out and display their contents.
- The list of participants can be kept in the space and updated whenever someone joins or leaves the discussion.
- Because the space is persistent, a new member can read and view the entire discussion.

Discussion: Advantages

- You can implement such a multi-user chat system in RMI by creating remote interfaces for the interactions discussed.
- Using JavaSpaces technology, you need only one interface.

Multi-user Chat: The Message

```java
import net.jini.core.entry.*;
public class MessageEntry implements Entry {
    public String content;
    public MessageEntry() {
    }
    public MessageEntry(String content) {
        this.content = content;
    }
    public String toString() {
        return "MessageContent:" + content;
    }
}
```
Multi-user Chat: Writing the Message

```java
JavaSpace space = getspace();
MessageEntry msg = new MessageEntry();
msg.content = "Hello there!";
space.write(msg, null, Lease.FOREVER);
```

Leases can be shorter: e.g. 60*60*1000 ms

```java
space.write(msg, null, 60 * 60 * 1000);
```

---

Multi-user Chat: Reading the Message

```java
MessageEntry template = new MessageEntry();
MessageEntry output = (MessageEntry) space.read(template, null, Long.MAX_VALUE);
```

-- Template has null fields -- act as wildcards
-- Read will match any MessageEntry
-- Will block if no MessageEntry
-- Can use readIfExists(...) to avoid blocking

---

The “Full” Client

```java
import java.util.*;

public class Client {  
private JavaSpace space;
}
```

Code to find JavaSpace service (see course page)
Using JavaSpaces

- Download jini distribution (www.jini.org)
- Create subclasses of Entry for your application tuple(s)
- Compile including jini-ext.jar in classpath.
- Run Launch-All from installverify directory
- Select Register and choose IP address
- Run your clients including:
  - jini-ext.jar, jini-core.jar,
  - reggie.jar and outrigger.jar

JavaSpace Advantages

- Simple: very straightforward API
- Expressive: small set of operations but complex distributed applications possible.
- Supports loosely-coupled protocols:
  - Uncouples senders and receivers
  - Dynamic: servers can come and go
- Eases burden of writing client/server systems
  - Concurrency issues dealt with by space
  - Transactions supported (not described here).

Further Reading

- Books:
- Articles
  - Angerer, Bernhard (2003). Space-Based Programming. onjava.com