Distributed Systems

Seminar 4 – Java RMI and Alternative Communication Mechanisms

Dr. Thomas Springer
- Name the main advantages of remote method invocations compared with remote procedure calls.

  - RMI is extension of RPC mechanisms available in the object orientated world, can be combined with the advantages of object orientated paradigm
  - objects can provide remote interfaces -> more flexible implementation of the server functionality due to objects of varying granularity.
  - data can now be transferred not only as copies (value parameter semantic) but also references to the corresponding objects (reference parameter semantic) i.e. object references can be transferred as call parameters and return values of methods.
  - With the integration of remote method invocation into Java through Java RMI, the advantages of Java such as portability and garbage collection for distributed objects can also be gained.
Exercise 4.2

- An object for the management of the warehouse provides a method for removing items (removeItem) with a given product ID (long productId) and amount of items (int itemNumber) from the warehouse which is called during the order processing from the ordering system. The method sends a status message (String statusMessage) as output parameter.

  a) Model through Java futures the parallel removal of three items by the ordering system from the warehouse. Sketch the main artifacts to be implemented.

  b) Discuss, how threads can be used in contrast to invoke and handle parallel calls.

  c) Discuss the advantages and disadvantages of futures and threads for the implementation of parallel RPC calls.
Execution of Callable by thread provided by ExecutorService

ExecutorService executor = Executors.newFixedThreadPool(10);
Future<Long> future; //create future
Callable<Long> worker = new MyCallableLong(); //create worker
future=executor.submit(new MyCallable()); //execute Callable in thread

Get result based on future
long result;
result=future.get(); //waits for the computation to complete
result=future.get(timeout, timeunit); //waits for at most the given time
for the computation to complete

future.isCanceled() //true, if this task was cancelled before it
completed normally
future.isDone() //true, if task completed (normal, canceled, exception)
future.cancel(boolean mayInterruptIfRunning) //cancel execution of task
1. Remote Interface: plain java interface

2. Serializable Objects representing complex data structures

3. Futures – using Callable – one per Remote Method

4. InterfaceImpl implementing server functionality

5. Server for instantiating and registering Impl with registry

6. Client for looking up Impl and making calls
Callable for each procedure/method call
Similar to Runnable but with return value

Client
1. creating executor -&gt; manages multiple threads/callables (java.util.concurrent.ExecutorService)
2. Make calls - Future&lt;String&gt; f1 = executor.submit(new CallableRemoveItem(1, 2, server));
3. Get Results - String r1 = f1.get();
What is a thread?

- Lightweight process
  - common address space
  - fast creation and process switching
  - large number of processes possible
    - use in RPC-Server - implementations
    - use in Client, too ⇒ asynchronous

- Process/Thread assignment
  - process creation per call or
  - process – Pool
• **Client-site:** Simultaneous calls on several servers

• **Server-site:** Processing of several calls

![Diagram showing client-server interactions](image-url)
<table>
<thead>
<tr>
<th></th>
<th><strong>Threads</strong></th>
<th><strong>Futures/Promises</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integration Level</strong></td>
<td>Low level (OS APIs)</td>
<td>Integrated into programming language</td>
</tr>
<tr>
<td><strong>Flexibility</strong></td>
<td>High, arbitrary call bundling and handling possible (e.g. push notifications)</td>
<td>Limited to embedded mechanisms</td>
</tr>
<tr>
<td><strong>Typing</strong></td>
<td>No support at call level</td>
<td>Strictly typed, checked at compile time</td>
</tr>
<tr>
<td><strong>Result checks</strong></td>
<td>Embedded in call thread or explicitly in main thread</td>
<td>Cyclic checking in main Thread</td>
</tr>
<tr>
<td><strong>Developer Effort</strong></td>
<td>High, explicit thread handling</td>
<td>Low, embedded, threads encapsulated</td>
</tr>
</tbody>
</table>
Publish/subscribe is an alternative interaction scheme to request/response used for RPC.

a) Explain the roles of the communication partners and the message exchange for publish/subscribe.

b) Explain the principles of a message channel and a message queue.

c) Which advantages and disadvantages result from the use of message based systems in comparison to RPC systems.
2 roles:
- Supplier – creates messages
- Consumer – processes messages

m:n Communication based on messages
- Inherent asynchronous
- binding flexible based on subscriptions
- Type of exchanged messages defined by Channel, Subject or Content
- N:M – communication
  - multiple Supplier and multiple Consumer
- asynchronous
  - supplier and consumer does not have to run at the same time
- loosely coupled
  - no direct message exchange between Supplier and Consumer
- Reliable message transfer based on persistent queues in the message channel
Request/Response (RPC)

- close coupling (rebinding difficult)
- 1:1 communication
- inherently synchronous (problem of disconnections)
- familiar interaction scheme with call and result
- Client and Server are synchronized
- additional effort for reliability necessary (at-most-once-semantics)
- client/server-systems (data processing with results)

Publish/Subscribe (Message Queuing)

- **loosely coupled** (better support for rebinding)
- m:n communication
- **inherent asynchronous** (better handling of disconnections)
- **simple interaction scheme**
  - basis for more complex interactions
  - additional afford required to create complex interaction schemes
- explicit synchronization required
- no result for message
- reliable message exchange based on message queues
- load balancing, parallelization, batch processing, event distribution
Exercise 4.4

- Which phases can be differentiated in stream based communication and what purpose do they serve?
Solution E4.4

![Diagram showing the call flow between Participant 1, SIP-Proxy 1, SIP-Proxy 2, and Participant 2. The process includes INVITE, 100 Trying, 180 Ringing, 200 OK, ACK, Use Phase (e.g., with RTP), BYE, and 200 OK messages.]

- Participant 1
  - INVITE
  - 100 Trying
  - 180 Ringing
  - 200 OK

- SIP-Proxy 1
  - INVITE
  - 100 Trying
  - 180 Ringing
  - 200 OK

- SIP-Proxy 2
  - INVITE
  - 180 Ringing
  - 200 OK

- Participant 2
  - Connection Establishment
  - Connection Closure
  - Use

Use Phase (e.g., with RTP)
The communication mechanisms discussed so far are based on different protocols. Sketch the protocol stacks according to the ISO/OSI model for RPC, Web services, AJAX, message-based and stream-based communication.