Middleware for synchronous requests

Revision: 528

Sophie Chabridon and Chantal Taconet

September 2021
1. Introduction

1.1 Middleware for distribution
1.2 Goal: interoperability
1.3 Distribution models
1.4 Client-server models
1.5 Middleware for distributed objects history
1.6 Synchronous vs asynchronous mode
1.7 Asynchronous call, synchronous call, buffered message
1.8 Call-back and Inversion of control

2. Synchronous middleware and the big picture

3. Conclusions
1.1 Middleware for distribution

Middleware is a software layer which provides:

- Programming interfaces (common API)
- Protocol for interoperability
  - With data exchange format

...to support distribution and heterogeneity.
1.2 Goal: interoperability

- Existing “legacy code”,

- Numerous languages,

- Several operating systems,

- Various hardware (e.g., little endian, big endian),

- Several network protocols

⇒ need for interoperability!
1.3 Distribution models

- Point to point message
- Point to multipoint message
- Event/action
- Publish/subscribe
- **Client/server**
- Mobile code
- Virtual shared memory
1.4 Client-server models

- **Procedural**
  - Remote Procedure Call - RPC

- **Object-oriented**
  - Remote Method Invocation (Java RMI, Common Object Request Broker Architecture CORBA)

- **Data-oriented**
  - SQL requests
  - REST (Representational State Transfer)- create, read, update, delete over HTTP

- **Traditionnal Web** (HTTP requests)

- **Web Services** (SOAP over HTTP)
1.5 Middleware for distributed objects history

- Comes from two technologies:
  - Objects (inheritance, encapsulation and polymorphism)
  - RPC or Remote Procedure Call (distribution, heterogeneity, data marshalling and unmarshalling)
1.6 Synchronous vs asynchronous mode

- Two entities (e.g., processus) communicate

  - In **synchronous** mode: the two entities (client and server) are active at the same time, after a request, client is waiting for server response.

  - In **asynchronous** mode: entities send messages, they don’t wait for responses, they don’t know when the message will be delivered.
1.7 Asynchronous call, synchronous call, buffered message

**Asynchronous event (push)**

- Event/message
- Event/message handler

**Synchronous call**

- Process request
- Process request handler
- Process reply
- Wait
- Send msg 1
- Send msg 2
- Send msg 3
- Deliver msg 1
- Deliver msg 2
- Deliver msg 3

**Buffered messages (pull)**

- Ask for a msg
- Deliver msg 1
- Send msg 1
- Ask for a msg
- Deliver msg 2
- Send msg 2
- Ask for a msg
- Deliver msg 3
- Send msg 3
1.8 Call-back and Inversion of control

**Synchronous call with callback**
A callback is first registered and later called asynchronously.

**Inversion of control**
The control flow is no more under the responsibility of the application but controlled by the framework.

The service request for A is triggered from the outside through B, which controls A.

Service request for A controlled by B.

callback
callback

result
2 Synchronous middleware and the big picture

- Life cycle (instantiate)
- Persistency

Middleware for synchronous requests
2.1 Introduction of the distributed example

- Which distribution?
- Which abstractions (service, object)?
- Which middleware?
2.2 Principle of distributed objects

Interface (contract)

```java
interface Printer {
    JobInfo submitPrint();
}
```

Automatic generation stub and skeleton implementation

- client
- implementation
- stub
- skeleton

Communications messages

Middleware for synchronous requests
2.3 The stub and the skeleton

t = RPI.submitPrint();

$I().submitPrint()$

client

stub

skeleton

implementation

marshalling

unmarshalling

marshalling

unmarshalling

proxy

PI

message

message

arguments

results

arguments

results

stub skeleton implementation

message

arguments

results

PI

I.
2.4 Proxy Object and inheritance tree

- Proxy: Representative for remote access

![Diagram of a proxy object and inheritance tree]

```
interface
Printer Interface
submitPrint()

<<implements>>
Client
submitPrint()
Printer Implementation
submitPrint()
Printer Proxy
delegates
```

Sophie Chabridon and Chantal Taconet
Middleware for synchronous requests
2.5 Proxy design pattern

Context: A client needs access to a remote service provided by some entity (called the “servant”)

Problem

- Define an access mechanism that does not involve
  - Hard-coding the location of the servant into the client code
  - Deep knowledge of the communication protocols by the client

Desirable properties

- Access should be efficient at run-time and secure
- Programming should be simple: No difference between local and remote access

Constraints: Distributed environment (no single address space)

Solutions

- Use a local representative of the server on the client side that isolates the client from the communication system and the servant
- Keep the same interface for the representative as for the servant
- Define a uniform proxy structure to facilitate automatic generation
2.5.1 Sequence diagram of Proxy

- **c:** Client
- **p:** Proxy
- **s:** Servant

**Pre-processing:**
- e.g., marshalling

**Post-processing:**
- e.g., unmarshalling

**Interface I**

- Service request
- Result

Sophie Chabridon and Chantal Taconet

Middleware for synchronous requests
2.6 Distribution Implementation Process

1. Description of the interface in **IDL**

2. IDL compiler creates the stub and the skeleton

3. Write both **client and server implementations**
2.7 Multi-languages (or multi-ORBs, or multi-OSs)

Interface IDL

```
interface Printer {
    Tache submitPrint();
};
```

Compile IDL → C++

```
class PrinterImpl {
    client Java implementation
};
```

Compile IDL → Java

```
skeleton stub Java
stub C++
```

Client Java

```
printer.submitPrint();
```

Skeleton Java

```
skeleton stub Java
```

Skeleton C++

```
skeleton stub C++
```

Implementation C++

```
class PrinterImpl {
};
```
2.8 Distribution implications

- Objects/service implementation are in different spaces (not the same process, not the same computer ...):
  - Assign a **unique identifier** to each object/service in different spaces
  - Localize objects/service implementations
  - Transports requests and replys
  - Use of a neutral network format for the data
2.9 Invocation sequence diagram

- **c:** Client
- **m:** Printer
- **i:** Printer
- **m:** Printer
- **i:** Printer
- **Broker**
- **ProxyServer**
- **Implementation**

Sequence steps:
1. `submitPrint()` in `c:Client`
2. `forwardRequest()` in `m:Printer`
3. `receive()` in `Broker`
4. `locate()` in `Broker`
5. `receive()` in `m:Printer`
6. `invoke()` in `m:Printer`
7. `submitPrint()` in `i:Printer`
2.10 Middleware for synchronous requests: main concepts

t = rPI.submitPrint();
submitPrint()

middleware protocol
ORB 1
ORB 2

client stub

message

middleware skeleton

message

IDL
interface definition language

Server

implementation

i.submitPrint()
2.11 Inherent complexity of distribution

- No global state

- Poor debugging tools

- Partial failures, network partition

- Requests in parallel (concurrency management)

- Trusting the caller (authentication)
3 Conclusions

1. Introduction

2. Synchronous middleware and the big picture

3. Conclusions
   3.1 Main distributed object middleware
   3.2 Middleware history
   3.3 Comparison of historical synchronous middleware
   3.4 Conclusions
3.1 Main distributed object middleware

- CORBA (OMG) 1991
- Java RMI (Sun) 1997
- SOAP WebService (w3C) 2001
- REST WebService (w3C) 2001
- GoogleRPC (Google) 2015
- GraphQL (Facebook) 2015
3.2 Middleware history

- BEA Tuxedo (IBM Encina)
- OMG OTS−1994
- Arjuna−R−1989
- Recoverability−R−1988
- Nested transaction−R−1980
- BEA MQ
- DEC MQ
- DEC FUSE
- JMS (1998)
- TIBCO TIB
- InformationBus−R−1992
- Field−R−1987
- Orca−R−1989
- Emerald−R−1987
- Network Objects−R−1993
- RPC Systems
- CORBA
- RMI
- RMI−R−1998
- SOAP & WSDL
- XML
- SGML−R−1986
- GML−R−1981
- Scribe−R−1981
- MQTT
- RabbitMQ (2007)
- REST Web Services (2000)
- EJB & JTA (1999)
- JMS (1998)
- Radar
- Recoverability−R−1988
- Nested transaction−R−1980
- RPC Systems
- CORBA
- RMI
- RMI−R−1998
- XML
- SGML−R−1986
- GML−R−1981
- Scribe−R−1981
- MQTT
- RabbitMQ (2007)
- REST Web Services (2000)
- EJB & JTA (1999)
- JMS (1998)
- Radar
- Recoverability−R−1988
- Nested transaction−R−1980
- RPC Systems
- CORBA
- RMI
- RMI−R−1998
- XML
- SGML−R−1986
- GML−R−1981
- Scribe−R−1981
- MQTT
- RabbitMQ (2007)
- REST Web Services (2000)
### 3.3 Comparison of historical synchronous middleware

<table>
<thead>
<tr>
<th></th>
<th>CORBA</th>
<th>RMI</th>
<th>SOAP</th>
<th>REST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>OMG</td>
<td>SUN</td>
<td>W3C</td>
<td>W3C</td>
</tr>
<tr>
<td><strong>Prog. language</strong></td>
<td>multi</td>
<td>Java</td>
<td>multi</td>
<td>multi</td>
</tr>
<tr>
<td><strong>IDL</strong></td>
<td>IDL CORBA</td>
<td>interface Java</td>
<td>WSDL</td>
<td>URIs</td>
</tr>
<tr>
<td><strong>data presentation</strong></td>
<td>CDR/binary</td>
<td>serialisation/binary</td>
<td>SOAP Envelope/XML</td>
<td>JSON/XML/text</td>
</tr>
<tr>
<td><strong>protocole</strong></td>
<td>IIOP</td>
<td>IIOP</td>
<td>SOAP Protocol</td>
<td>HTTP1</td>
</tr>
<tr>
<td><strong>connexions</strong></td>
<td>connected</td>
<td>connected</td>
<td>short connexions</td>
<td>short connexions</td>
</tr>
<tr>
<td><strong>object references</strong></td>
<td>IOR (location independant)</td>
<td>symbolic names/IP+port</td>
<td>URL</td>
<td>URI</td>
</tr>
<tr>
<td><strong>naming service</strong></td>
<td>NS, trading</td>
<td>RMI registry, JNDI</td>
<td>UDDI, WSIL</td>
<td>/</td>
</tr>
<tr>
<td><strong>main advantages</strong></td>
<td>services/efficiency</td>
<td>easy to use in java</td>
<td>SOA</td>
<td>simple</td>
</tr>
<tr>
<td><strong>main difficulties</strong></td>
<td>complex to learn</td>
<td>Java/Java</td>
<td>complex</td>
<td>low level/need a third party</td>
</tr>
</tbody>
</table>

---

Sophie Chabridon and Chantal Taconet

Middleware for synchronous requests
3.4 Conclusions

- Granularity of distribution variable (object, service),
- Complexity of distribution
- Synchronous request middleware is the necessary foundation to build higher level middleware
  - Message Oriented Middleware (MOM) (asynchronous middleware)
  - Application servers
  - Component middleware
  - Compositions and orchestrations
*Middleware Architecture with Patterns and Frameworks.*