AARHUS UNIVERSITET

Microservices and DevOps

DevOps and Container Technology

Distributed Computing

Henrik Bærbak Christensen

AARHUS UNIVERSITET

Distributed System

Definition: Distributed System

A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages. (Coulouris, Dollimore, Kindberg, and Blair 2012)

- Why?
  - To speed up computation
    - Google search, and a few other cases
  - ***To share information***
    - Everything else! (Slight exaggeration!)

AARHUS UNIVERSITET

A MultiStage Approach

- This whole fagpakke is a basically about distributed systems
  - (Part of the definition of Microservices)
- We will approach it in stages
  - The Basics — send and receive network package
  - Broker Pattern — Object Oriented Style
  - REST — Data/Resource centric Style
  - Microservices — Well…
  - Arch. Qualities — Availability, Testability, Scalability…

“Limitation”

- We will stay in an important ”niche”

Client-server Architectures using Remote Procedure Call

… this niche covers quite a few systems in practice ☺

AARHUS UNIVERSITET

Security

- Distributed systems means **Security…**
  - There may be actors whose intentions are not good ☹
- Architecturally, security is quite a pain in our context…
  - Because security techniques are one big set of *hard bindings and strong coupling*
    - You need certificates that tie you to a specific DNS name
      - Certificate stores, key pair generation, trust chains, yaga yaga
- We will cover standard techniques but…
  - Likely disable it quite a lot
  - Not cover more specialized techniques

The History

- Birrell and Nelson, 1984:
  - “allow calling procedure on remote machines”
  - A calls procedure f on B means
    - A suspends, information on f is transmitted to B
    - B executes the f procedure
    - B sends the result back to A
    - A resumes

AARHUS UNIVERSITET

Initial Glimpse

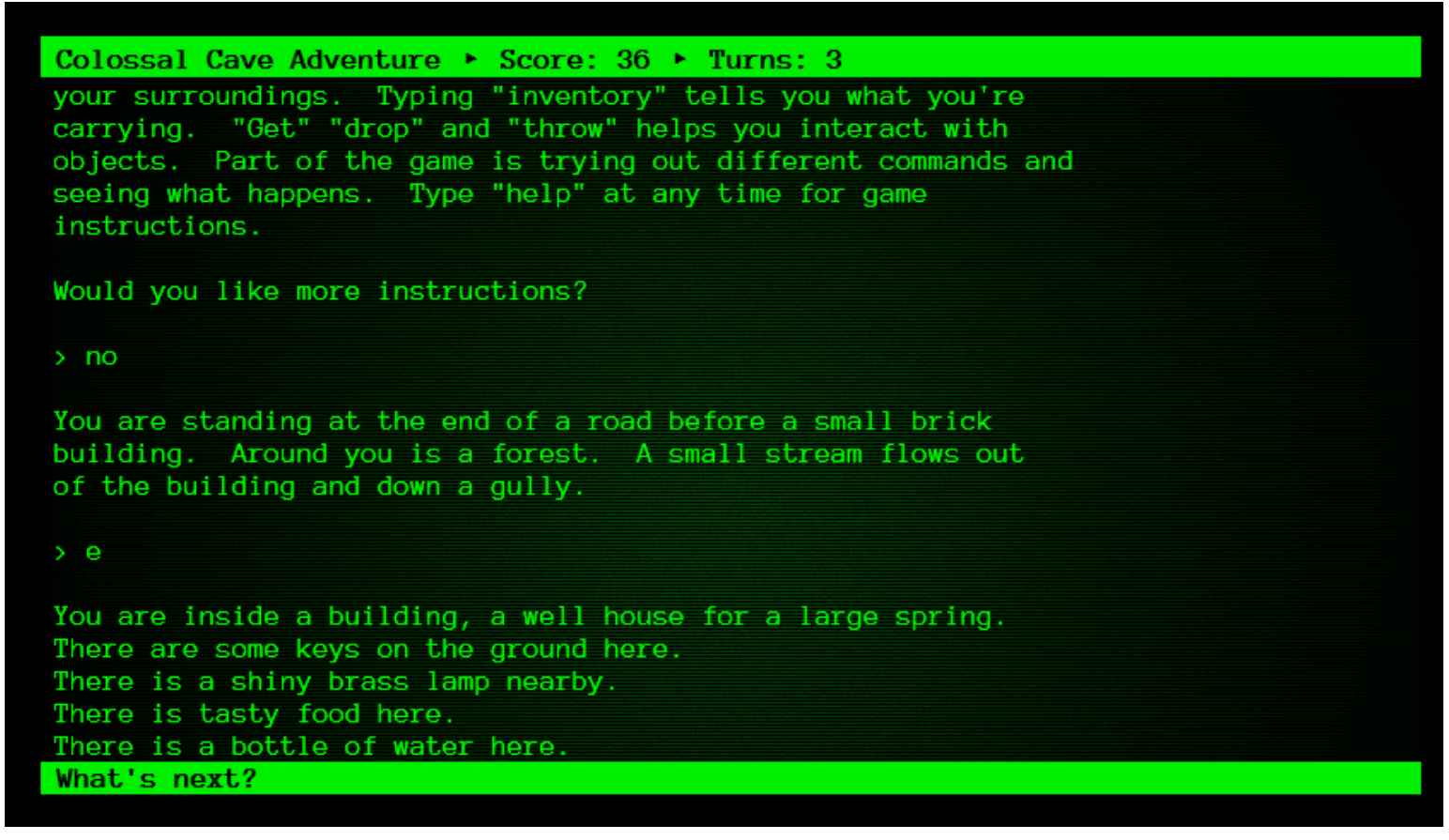
SkyCave

AARHUS UNIVERSITET

Case Study: SkyCave

- SkyCave is a
  - Massive multi-user online, exploration and creation experience with a bit of social networking
  - … with a *horrible* user interface
  - … and high disregards for security
    - Learn that somewhere else…
- Inspired by the very first ‘interactive fiction’ game for a computer: Colossal Cave Adventure
  - Will Crowder, 1972
    - I played my first game in 1986

The History


Colossal Cave Adventure ▸ Score: 36 ▸ Turns: 3
your surroundings. Typing "inventory" tells you what you're
carrying. "Get" "drop" and "throw" helps you interact with
objects. Part of the game is trying out different commands and
seeing what happens. Type "help" at any time for game
instructions.
Would you like more instructions?
› no
You are standing at the end of a road before a small brick
building. Around you is a forest. A small stream flows out
of the building and down a gully.
› e
You are inside a building, a well house for a large spring.
There are some keys on the ground here.
There is a shiny brass lamp nearby.
There is tasty food here.
There is a bottle of water here.
What's next?

So: Let the *player move* east…

```
== Welcome to SkyCave, player Mikkel ==
Entering command loop, type "q" to quit, "h" for help.
> e
You moved EAST
You are inside a building, a well house for a large spring.

>
```

```
private void handleSingleCharCommand(char primaryCommand) {
  switch (primaryCommand) {
  // look
  case 'l': {
    systemOut.println(player.getLongRoomDescription());
    break;
  }
  // The movement commands
  case 'n': {
    tryToMove(getDirectionFromChar(primaryCommand));
    break;
  }
  case 's': {
    tryToMove(getDirectionFromChar(primaryCommand));
    break;
  }
  case 'e': {
    tryToMove(getDirectionFromChar(primaryCommand));
    break;
  }
```

```
private void tryToMove(Direction direction) {
  if ( player.move(direction) ) {
    systemOut.println("You moved "+direction);
    systemOut.println(player.getShortRoomDescription())
  } else {
    systemOut.println("There is no exit going " + direction);
  }
}
```

Broker: Let a ‘player.move(d)’ method on the *client side* be ”transparently” translated into a ‘player.move(d)’ on the *server side*…

AARHUS UNIVERSITET

On the Server Side

```
2019-06-13 14:21:02,180 INFO  [qtp183731339-14:UriTunnelServerRequestHandler] - --> Received request: RequestObject{operationName='player-move', payload='["EAST"]', objectId='user-001##643964d7-b6fa-44cd-9968-0b644421a493', versionIdentity=4}
2019-06-13 14:21:02,182 INFO  [qtp183731339-14:UriTunnelServerRequestHandler] - --< Reply: ReplyObject [payload=true, errorDescription=null, responseCode=200]
2019-06-13 14:21:02,185 INFO  [qtp183731339-16:UriTunnelServerRequestHandler] - --> Received request: RequestObject{operationName='player-get-short-room-description', payload='[]', objectId='user-001##643964d7-b6fa-44cd-9968-0b644421a493', versionIdentity=4}
2019-06-13 14:21:02,185 INFO  [qtp183731339-16:UriTunnelServerRequestHandler] - --< Reply: ReplyObject [payload="You are inside a building, a well no use for a large spring.", errorDescription=null, responseCode=200]
```

```java
@Override
public boolean move(Direction direction) {
  // Convert present room position into Point3 which
  // allows computations
  Point3 presentPosition = Point3.parseString(position);

  // Clone it; we need the values of both present and
  // new position
  Point3 newPosition = (Point3) presentPosition.clone();

  // Calculate a new position given the movement direction
  newPosition.translate(direction);
  // convert to the new position in string format
  String newPositionAsString = newPosition.getPositionString();
  // get the room in that direction
  RoomRecord newRoom = storage.getRoom(newPositionAsString);

  // if it is null, then there is no room in that direction
  // and we return without any state modifications
  if ( newRoom == null ) { return false; }

  updateStateAndStorageToNewPosition(newPositionAsString, newRoom);

  return true;
}
```

```java
@Override
public String getShortRoomDescription() {
  return currentRoom.getDescription();
}
```

AARHUS UNIVERSITET

Issues in Distribution

Why is it hard?

AARHUS UNIVERSITET

Challenge

- How guys like me like to code:

> **Definition: Object-orientation (Responsibility)**
>
> An object-oriented program is structured as a community of interacting agents called objects. Each object has a role to play. Each object provides a service or performs an action that is used by other members of the community.

- Which is then something like:

```
private void tryToMove(Direction direction) {
  if ( player.move(direction) ) {
    systemOut.println("You moved "+direction);
    systemOut.println(player.getShortRoomDescription());
  } else {
    systemOut.println("There is no exit going " + direction);
  }
}
```

Challenge

- However - networks *only support two asynch functions!*

```
void send(Object serverAddress, byte[] message);
byte[] receive();
```

- Which is *not* exactly the same as

```
private void tryToMove(Direction direction) {
  if ( player.move(direction) ) {
    systemOut.println("You moved "+direction);
    systemOut.println(player.getShortRoomDescription());
  } else {
    systemOut.println("There is no exit going " + direction);
  }
}
```

AARHUS UNIVERSITET

Issues (at least!)

- Send/receive is a too low level a programming model
- Send() does not wait for a reply from server (Asynch)
- Reference to object on *my* machine does not make sense on *remote* computer (memory address)
- Networks does not transfer objects, just bits
- Networks are slow
- Networks and Remote computers may fail
- Networks are insecure, others may pick up

Performance

- Just how much slower is a network call compared to a local in-JVM memory call?

Configuration	Average time (ms)	Max time (ms)	Factor
Local call	1,796	3,366	1.0
Localhost	9,731	12,806	5.4
Docker	17,091	35,873	9.5
On switch	22,817	29,427	12.7
Frankurt	494,966	513,411	275.6

- Imagine that your next trip to the supermarket for a soda was 275 times slower???
  - 10 minutes walk versus 46,8 hours walking ☺

AARHUS UNIVERSITET

Elements of a Solution

- On the ‘happy path’, we need to
  - Make the client invoke a synchronous method call on a remote player object using only network send/receive
    - *That is, the client blocks until server has returned reply*
  - Keep our OO programming model: *player.move(‘east’);*
    - *That is invoke specific method on specific remote object*
  - Convert method call and parameters into bits to send it, and convert it back again
    - *That is, convert Enum/Object/Array into bits and back again*
  - Locate the remote right player object
    - *That is, invoke method on object Mikkel, not object Magnus !*

AARHUS UNIVERSITET

Elements Overview

- Solutions are
  - Request/Reply Protocol
    - Simulate synchronous call (solves (partly) concurrency issue)
  - Marshalling
    - Packing objects into bits and back (solves data issue)
  - Proxy Pattern (and Broker pattern)
    - Simulate method call on client (solves programming model issue)
  - Naming Services
    - Use a registry/name service (solves remote location issue)

AARHUS UNIVERSITET

Request/Reply

The Protocol

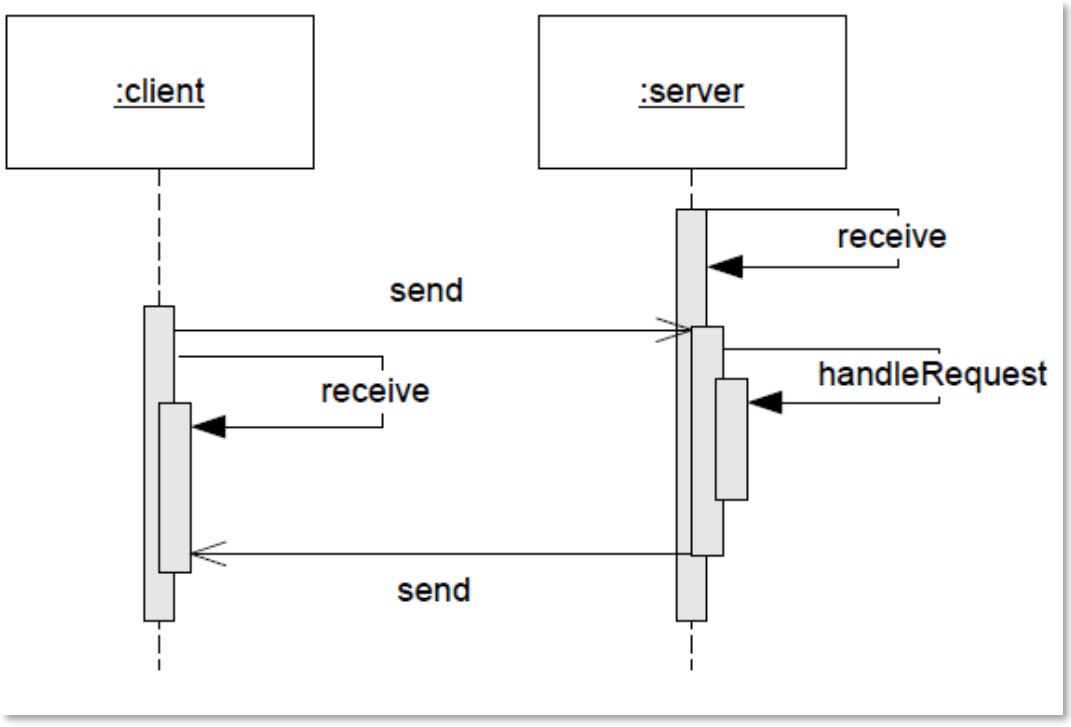
Definition: Request-Reply Protocol

The request-reply protocol simulate a synchroneous call between client and a server objects by a pairwise exchange of messages, one forming the request message from client to server, and the second forming the reply message from the server back to the client. The client sends the request message, and waits/blocks until the reply message has been received.

- Known from every WWW access you have ever made…

Pairing Send/Receives

- Client does
  - Send() and receive
- Server does
  - Receive() and send()
- Roles
  - Client is *active* – initiate action
  - Server is *reactive* – awaits actions and then reacts



Marshalling

Or Serialization

Definitions

Marshalling is the process of taking a collection of structured data items and assembling them into a byte array suitable for transmission in a network message.

Unmarshalling is the process of disassembling a byte array received in a network message to produce the equivalent collection of structured data items.

AARHUS UNIVERSITET

Two Basic Approaches

- There are two approaches
  - Binary formats
    - Google ProtoBuf, propriatary
  - Textual formats
    - XML, JSON, propriatary
- Exercise: Costs? Benefits?

JSON blood pressure

```
{
  patientId:   ''251248-1234'',
  systolic:    128.0,
  diastolic:   76.0
}
```

And we need more

- As we can send only bits, we also need to marshal information about the method and object id!

```
{
  methodName : "processAndStore_method",
  parameters : [
    {
      patientId:  ''251248-1234'',
      systolic:   128.0,
      diastolic:  76.0
    }
  ]
}
```

Note

- Marshalling is fine for atomic datatypes (int, String, double, array, …) but…
- What about *object references?*
  - *inventory.addCustomer(c)* *where c is Customer object?*

- Java RMI can do such things, but it is complex
  - Naming schemes, registries of implementing classes, dynamic class loading, …
- FRDS.Broker *only* support server hold references
  - I.e. *a server never calls an object on a client! ClientServer!!!*

The Details

- **Pass by reference**
  - The Java style for all objects
  - *You do not get the ”Hello” string, you get a reference to it!*
  - public void say(String s);

- **Pass by value**
  - Java does this for primitive types, like int and double
  - You do get the value itself
  - public void deposit(double amount);

In C and C++

- In C and C++ you can actually do both

```
void fooByValue(int value) { ... }
void fooByRef(int* value) { ... }
```

  - If the *first* call adds 10 to value, what happens to value at the *call site?*
    - *int v = 7; fooByValue(v); print(v);*
  - If the *second* call adds 10 to value, what happens to value at the *call site?*
    - *int v = 7; fooByRef(&v); print(v);*

In Our Broker

- The semantics *change* in our Broker Pattern
  - localObject.say(”Hello”) localObject *pass by reference*
  - remoteObject.say(”Hello”) remoteObject *pass by value*
- Exercise: Why?

Our Broker *only* supports *pass by value!*
Server objects are *pass by ID.*

AARHUS UNIVERSITET

JSON Libraries

AARHUS UNIVERSITET

Libraries

- Every distributed system in the world needs to marshall!
- Thus – lots of marshalling libraries around ☺
  - Do NOT do it yourself!!!
    - ~~String json = ”{ name: ”+ object.name+ ”}…~~

- JSON I have used many libraries
  - Json-simple
  - Jackson JSON
  - Gson

Gson

- Gson is the most compact I have used
  - (But have had trouble with ‘date’ objects that marshalls incorrectly!)
- It allows easy marshalling of **record types**
  - Also known as
    - PODO: Plain Old Data Objects,
    - DTO: Data Transfer Object
- Record type (Pascal) / ‘struct’ (C) / ”bean” (java)
  - No complex methods, only set/get methods with no side effects
  - **Must** have a default constructor
- That is: A pure data object, just storing information
  - Akin a ‘resource’ in REST terminology, by the way

Example:

```java
@Test public void shouldMarshallTeleObservation() {
  // This is a learning test, showing Gson marshalling
  Gson gson = new Gson();
  String json = gson.toJson(to);

  assertThat(json, containsString( substring: "\"patientId\":\"251248-0000\""));

  TeleObservation copy = gson.fromJson(json, TeleObservation.class);
  assertThat( copy.getPatientId(), is(HelperMethods.NANCY_ID));
  assertThat( copy.getSystolic().getValue(), is( value: 120.0));
  assertThat( copy.getDiastolic().getValue(), is( value: 70.0));
  assertThat( copy.getSystolic().getUnit(), is( value: "mm(Hg)") );
}
```

- toJson(obj)
  - Marshall
- fromJson(str, type.class)
  - Demarshall, using given type

```json
{
  "patientId": "251248-0000",
  "systolic": {
    "value": 120,
    "unit": "mm(Hg)",
    "code": "MSC88019",
    "displayName": "Systolic BP"
  },
  "diastolic": {
    "value": 70,
    "unit": "mm(Hg)",
    "code": "MSC88020",
    "displayName": "Diastolic BP"
  },
  "time": {
    "date": {
      "year": 2017,
      "month": 6,
      "day": 30
    },
    "time": {
      "hour": 11,
      "minute": 7,
      "second": 26,
      "nano": 0
    }
  }
}
```

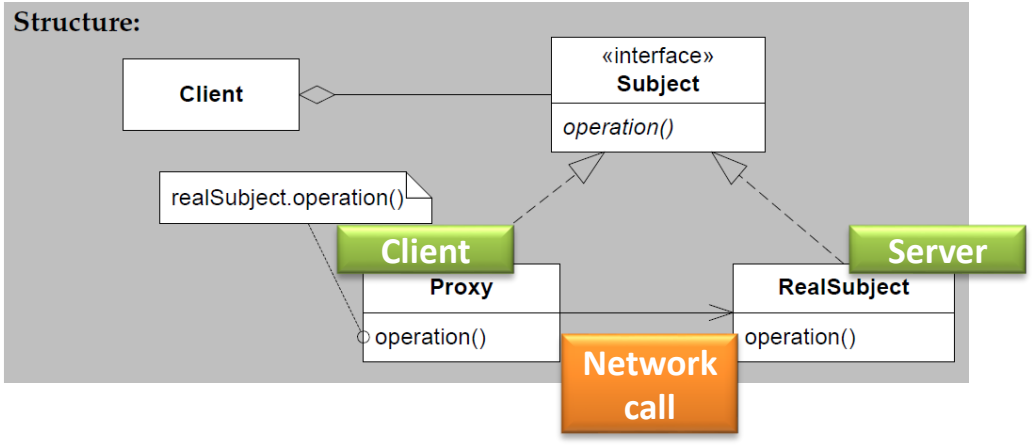
AARHUS UNIVERSITET

Proxy

Example

- TeleMed**Proxy**

```
public String processAndStore(TeleObservation teleObs) {
  byte[] requestMessage = marshall(teleObs);
  send(server, requestMessage);
  byte[] replyMessage = receive();
  String id = demarshall(replyMessage);
  return id;
}
```



Note

- The algorithm of *all methods in the proxy* will be the same
  - Marshall parameters, send, await reply, demarshall, return
- Can be auto generated – this is what RMI does
- We will *hand-code* it, because
  - It actually makes sense if you want very strict control of architectural attributes like availability, testability, performance
    - *Which are major learning goals in this course*

Why QAs

- Why? *One Example:*
- You have a lot of accessor methods, and a single mutator
  - i.e. *state only changes when mutator is invoked!*
- RMI will autogenerate proxy (send/receive) for every method
- That is, every accessor method call will generate network traffic!
- **Performance Antipattern: Chatty interface**
- Pattern: Chunky interface
  - All accessors just return cached state in the proxy instance itself!

AARHUS UNIVERSITET

Name Services

Name Services

- If I do not know, I know someone who does…
  - If I do not know the telephone number of X, I know someone who does
    - Old days: Telephone book
    - More modern days: krak.dk, linked in, facebook, whatever

- DNS: Domain Name Servers

Results for checks on www.imhotep.dk

Host	TTL	Class	Type	Details
imhotep.dk	21599	IN	A	87.238.248.116

Name Services

- Name Services are actually just a fancy name for a Map<Key, Value> data structure which allows me to associate a name/key with an object
  - DNS: I have ‘www.imhotep.dk’ as key, please give me the IP address of the associated computer
  - RMI Registry: I have this name ‘/sensor/temperature/47’, please give me a remote reference so I can talk with the remote object

- These are *server based solutions*
  - You contact a server that keeps the (key,value) database

AARHUS UNIVERSITET

Local Name Services

- For a *single server* system, you do not need an external name service, you can keep the (key,value) in the (memory of) the server

- E.g SkyCave *does* just keep a (playerId, player) mapping at hand…
  - Any upcall of a player method, will lookup in the name service, retrieve the player object, and do the method call on that particular object.
  - Exercise: Liabilities
    - Hint: Horizontal scaling?

Summary

- The Broker Pattern combines
  - Request/Reply protocol
  - Marshalling
  - Proxy pattern
  - Naming Systems
- … to produce something that (on happy days)

- Allows an Object Oriented Programming model to apply to distributed computing