AARHUS UNIVERSITET

Software Engineering and Architecture

Broker II Mandatory

Distributed HotStone

AARHUS UNIVERSITET

Broker II

- Learning Goal
  - Get the *handle object reference* methods implemented
    - **c** = getCardInHand(w,i), attackCard(w,**a,d**), and cousins...
    - Involves objectId generation and usage
  - Get the MiniDraw GUI integrated in a full client
  - Get the Invoker code segregated
    - Refactor 'blob invoker' to a subinvoker/multitype dispatch approach
- Product Goal
  - JUnit test suite that cover **all** broker related code
  - System testing of a *full HotStone GUI based remote play! Wow!*

AARHUS UNIVERSITET

Broker 2.1

- TDD the Game methods that handles object references.
- That is, write JUnit tests that implement the FRDS method

Consider a remote method `ClassB getB()` in `ClassA`, that is, a method that return references to instances of ClassB.

To transfer a reference to an object created on the server side, you must follow this template

- In the Invoker implementation of `ClassA.getB()`, retrieve the objectId of the ClassB instance, and use a String as return type marshalling format, and just transfer the unique object id back to the client.

- On the client side, in the ClassAProxy, create a instance of the ClassB-ClientProxy, and store the transferred unique id in the proxy object, and return that to the caller.

AARHUS UNIVERSITET

Small Steps

- Keep doing it ‘depth-first’
  - But the first ‘deep dive’ have some issues
- I recommend to introduce AlphaStone as your servant
  - To avoid writing too much stub code ☹
- Example:
  - On my test list: “make getCardInHand(Findus, 0) work”

```
@Test
public void shouldGetCardInHand() {
  // Given a game client proxy for a AlphaStone game servant
  // When I ask for card 0 in findus' hand
  Card card = game.getCardInHand(Player.FINDUS, indexInHand: 0);
  // Then I get a valid card
  assertThat(card, is(notNullValue()));
  // Then that card's name is 'Tres'
  assertThat(card.getName(), is( value: "Tres"));
  assertThat(card.getAttack(), is( value: 3));
```

AARHUS UNIVERSITET

Small Steps

- *Take small steps – again a proxy step + an invoker step*
  - (1) Involves making the GameClientProxy method
    - Request String/**objectId** from the server
    - Wrap that **objectId** in a CardClientProxy

```
/usr/lib/jvm/java-11-openjdk-amd64/bin/java ...
--> {"operationName":"game_get-card-in-hand","payload":"[\"FINDUS\",0]","objectId":"one-game","versionIdentity":1}
--< null
```

  - (2) Involves making the Invoker handling
    - If OpName.equals(GAME_GET_CARD_IN_HAND)
      - » Ask game for the card - servant upcall
      - » Return the card's **objectId** as serialized JSON

```
/usr/lib/jvm/java-11-openjdk-amd64/bin/java ...
--> {"operationName":"game_get-card-in-hand","payload":"[\"FINDUS\",0]","objectId":"one-game","versionIdentity":1}
--< {"payload":"\"2847f524-0edf-4c54-8cfc-53d473ebc077\"","statusCode":200,"versionIdentity":1}
```

AARHUS UNIVERSITET

Small Steps - FakeIt

- … But the test case likely fails (it did for me) – why?

```
java.lang.AssertionError:
Expected: is "Tres"
     but: was "Siete"
```

- Because we still have FakeIt lookup code for the card!

```
private Card fakeItCard = new StubCard();
1 usage   Henrik Baerbak Christensen (m1.racimo)
private Card lookupCard(String objectId) { return fakeItCard; }
```

- Now test drive the invoker side's
  - Insert (card id, card) mapping in a **name service**
  - Refactor the fake 'lookupCard()' method into a proper impl.

One method done; repeat until all covered ☺

AARHUS UNIVERSITET

ID Generation

- There are several ways to generate ID's
  - Invoker can do it
  - Domain object can do it
- IDs are rather pervasive (Read: all objects need it, even the client proxy) in a distributed setting so my recommendation is a *role interface on the domain obj:*
  - RoleInterface: *interface Identifiable { String getId(); }*
  - Let Card and Hero extend that interface
  - Let constructor create a random id ala

```
id = UUID.randomUUID().toString();
```

AARHUS UNIVERSITET

Pass-By-Ref Parameters

- Game have many methods whose parameters are object references

```
@Override
public Status playCard(Player who, Card card) {
```

- *But – they are server created objects, so no problem*

If you have a method in which a parameter is a server side object, ala this one:

```
Game game = futureGame.getGame();
lobbyProxy.tellIWantToLeave(game);
```

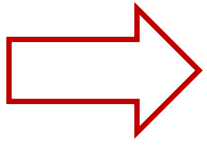
Then your proxy code of course shall just send the objectId to the server. This will allow the server side invoker to lookup the proper server object, and pass that to the equivalent `tellIWantToLeave()` method of the servant object.

Removing FakeIt / Scaffolding

- Once you start removing the scaffolding/fake it code

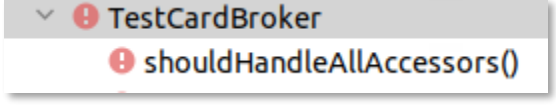
```
    // CARD Methods
} else if (operationName.startsWith(OperationNames.CARD_PREFIX)) {
    // Lookup the right card to invoke the method on
    Card servant = lookupCard(objectId);
```

```
private Card fakeItCard = new StubCard();
private Card lookupCard(String objectId) {
  return fakeItCard;
}
```



```
private Card lookupCard(String objectId) {
  return nameService.getCard(objectId);
}
```

- … other tests may fail…
  - **Huh – what is going on?**




```
/usr/lib/jvm/java-11-openjdk-amd64/bin/java ...
 --> {"operationName":"card_get-name","payload":"[]","objectId":"one-card","versionIdentity":1}

java.lang.NullPointerException Create breakpoint
    at hotstone.broker.server.GameInvokerBrokerI.lambda$populateCardFunctionMap$5(GameInvokerBrokerI.java:78)
    at hotstone.broker.server.GameInvokerBrokerI.handleRequest(GameInvokerBrokerI.java:138)
```

AARHUS UNIVERSITET

Removing Scaffolding

- My original test case created CardClientProxy *directly*

```
card = new CardClientProxy(requestor,  cardId: "one-card");
```

- Now, we cannot do that as
  - *Clients do not create cards – servers do!*

- So the answer is of course…
  - Servers create cards, so our test case must follow that rule

```
Game proxy = new GameClientProxy(requestor);
card = proxy.getCardInHand(Player.FINDUS,  indexInHand: 0);
```

- Ups – and test cases/doubles refactored…
  - Probably other return values, so asserts must be updated…

AARHUS UNIVERSITET

Removing Scaffolding

- Alternatively, you can also insert your ‘test stub’ card directly into the Invoker’s name service…

```
@BeforeEach  Henrik Bærbak Christensen +3
public void setup() {
  // Create and populate the name service with our stub card
  HotStoneNameService nameService = new HotStoneNameService();
  String theOneAndOnlyCardID = "ID42";
  nameService.addCard(theOneAndOnlyCardID,
          // Introduce a single stub card
          new StandardCard(Player.PEDDERSEN,  name: "StubCard",
                  manaCost: 7,  attack: 11,  health: 42,
                  (internalGame, dropIndex) -> {
                  },  effectDescription: "Save the Whales",
                  Categorizable.TAUNT));

  // Create the broker chain
  Invoker invoker = new CardInvoker(nameService);

  crh = new LocalMethodCallClientRequestHandler(invoker);
  Requestor requestor = new StandardJSONRequestor(crh);

  // Tie the proxy to this particular card id
  card = new CardClientProxy(requestor, theOneAndOnlyCardID);
}
```

Note: I have solved the ‘dispatching’ exercise…

Iterables

- And what about Iterable<>?

```
Iterable<? extends Card> getHand(Player who);
```

- The return type is ‘something that can iterate Cards’
  - List<Card> is an Iterable<? extends Card>
  - And what is Card in our context?

- Solution:
  - List of ObjectId’s of the associated cards from server
  - Client must then convert that into a List<CardClientProxy>

AARHUS UNIVERSITET

List<Something>

- Gson can easily marshall and demarshall Lists but you need a bit of 'magic' to define that type. *List<String>.class* is not valid Java.
  - Actually shown an example of it in TeleMed code

```
public List<TeleObservation> getObservationsFor(String patientId,
    TimeInterval interval) {
  Type collectionType =
    new TypeToken<List<TeleObservation>>(){}.getType();

  List<TeleObservation> returnedList;
  returnedList = requestor.sendRequestAndAwaitReply(TELEMED_OBJECTID,
            OperationNames.GET_OBSERVATIONS_FOR_OPERATION,
            collectionType, patientId, interval);

  return returnedList;
}
```

AARHUS UNIVERSITET

List<String>

- So this is the way to go

```
// Define the type of a list of String
Type collectionType =
        new TypeToken<List<String>>(){}.getType();
// Do the remote call to retrieve the list of IDs for
// all cards in the hand.
List<String> theIDList =
        requestor.sendRequestAndAwaitReply(objectId,
                theOperation,
                collectionType,
                who);
// Now convert the ID list into list of CardClientProxies
```

- Fill in the details…

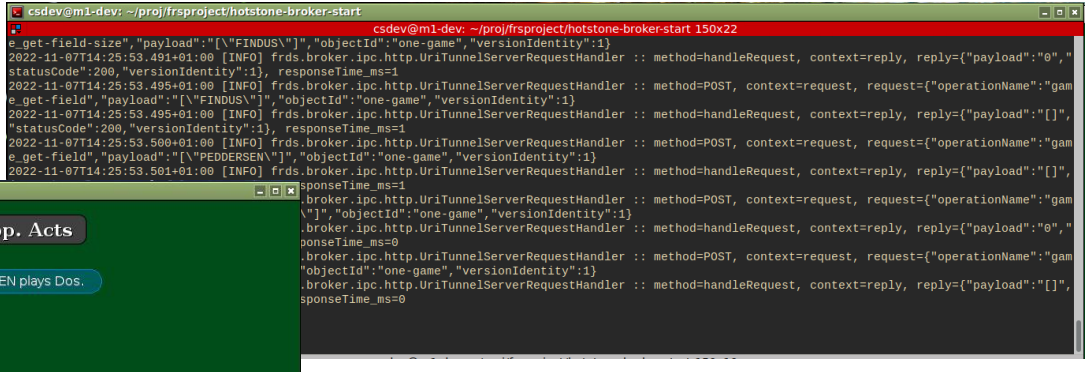
AARHUS UNIVERSITET

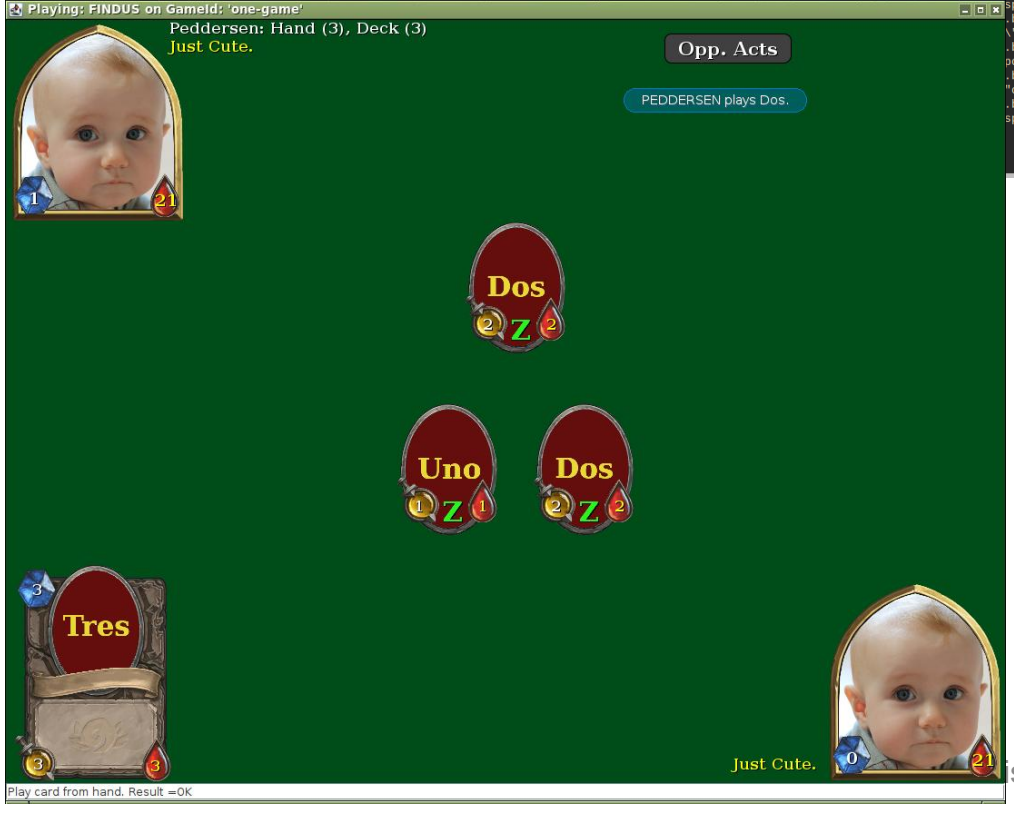
2.2 System Testing

A full distributed playable system…

Product Goal: Full system

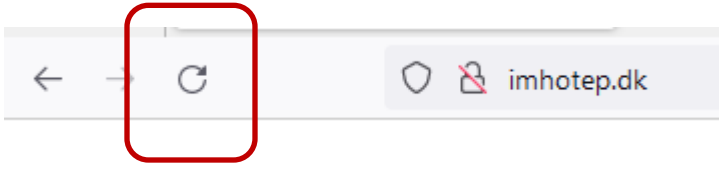
- A server and
  - Two clients

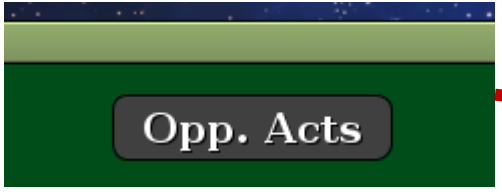




Limitation: No Observer

- FRDS Broker does not support *server-to-client* calls
  - Thus GameServant *cannot* invoke GameObserver's on the client side
- Solution provided: **The refresh button**



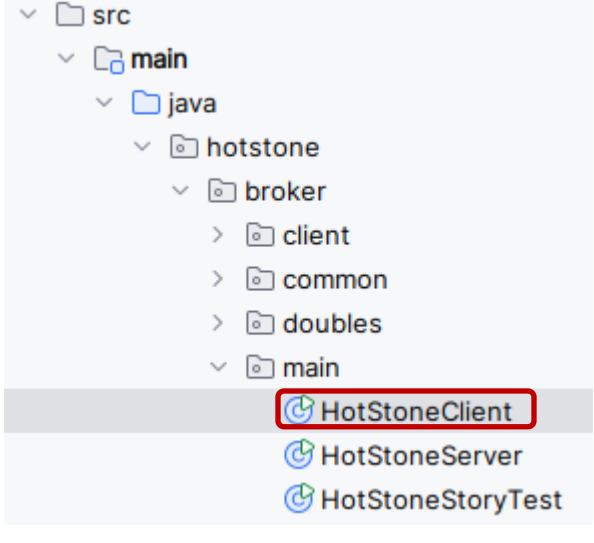


- That is:
  - The player which is *not active player / not in turn*
  - … have to press the 'next opponent action' button every 2-5-10 seconds…

AARHUS UNIVERSITET

Guide to Tackling Integration

- System testing = manual testing
  - But remember the *small steps*
- What do need?
  - A Game server — Done! Made in Broker I mandatory ☺
  - A Game Client — *Starting point* provided…



AARHUS UNIVERSITET

Small Step 1

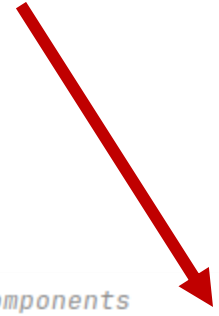
- I often call this ‘First Light’
  - The first time I see “something working”
  - (In Astronomy ‘first light’ is the first time you see the night sky through a new telescope ☺)

- HotStoneClient’s main() method
  - Setup broker chain to server
  - Initialize Iteration 8’s GUI code with our game proxy
  - See a UI pop up ☺
    - Or hit a zillion null ptr exception ☹

AARHUS UNIVERSITET

Hints: HotStoneClient

- The Factory is prepared for Remote Play
  - The 'OPPONENT_MODE'



```
/**
 * Construct factory for minidraw coupled with a HotStone game.
 * @param game The game to be associated with
 * @param operatingPlayer The player that this UI represents
 * @param uiType The type of UI to visualize - either a
 *               HotSeat type (both players use the same UI) or
 *               an Opponent type (each player has own UI).
 */
public HotStoneFactory(Game game, Player operatingPlayer,
                       HotStoneDrawingType uiType) {
  this.game = game;
  this.operatingPlayer = operatingPlayer;
  this.uiType = uiType;
}
```

```
// Create the UI components
Factory factory =
        new HotStoneFactorySolution(game, whoToPlay,
                HotStoneDrawingType.OPPONENT_MODE);
DrawingEditor editor =
        new MiniDrawApplication( title: "Playing: " + whoToPlay
                + " on GameId: '" + gameid +"'",
                factory);
```

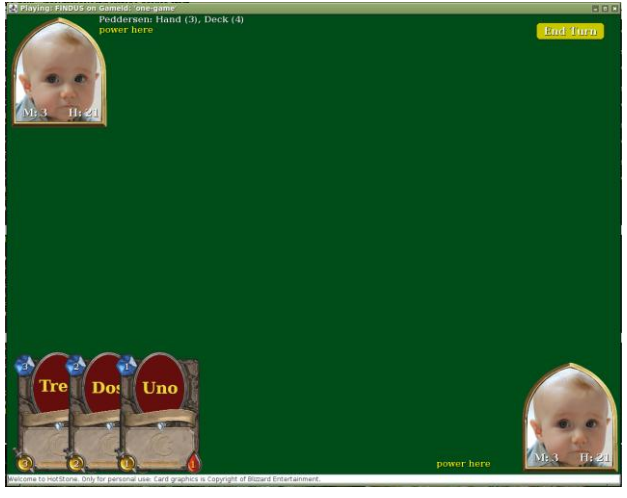
AARHUS UNIVERSITET

First Light

- Just use a Null tool –
  - But – We cannot do anything???

```
editor.open();
editor.setTool(new NullTool());
```

- Yes, but *small steps!!!*
  - *Verify that the UI pop up correctly*



- *I see the correct UI*
- *I see correct server communication*

- I cannot do anything
  - But that is “next small step”, right?!

AARHUS UNIVERSITET

NullPointer Issue

- Most of you probably create CardClientProxies like crazy
  - Every 'getCardInHand()' creates a new CardClientProxy

- But HotStoneDrawing keeps a mapping (card, actor):
  - Given a card, it can fetch the associated Figure

```
// Add the figure to the drawing's collection (for rendering)
add(actor);
actorMap.put(card, actor);
```

```
// Then iterate all fielded cards
for (Card card: game.getField(who)) {
  HotStoneActorFigure actor = actorMap.get(card);
```

- Exercise
  - *Why will this no longer work???*
  - *What is the solution to make the Drawing work again???*
  - *And will this solution be backwards compatible?* Work in 'HotSeat' mode?

AARHUS UNIVERSITET

We are actually almost done!

- What Tool to use?
  - Of course not our ‘HotSeatStateTool’
    - It operates both Findus and Peddersen
- We need a new state tool which is *almost identical but not quite*

```
editor.open();
editor.setTool(new DualUserInterfaceTool(editor, game, whoToPlay));
```

- *That is, a tool which determine what tool to delegate to, based upon the figure underneath…*
- But first – we have to discuss *GUI updates…*

AARHUS UNIVERSITET

Domain -> GUI Updates

The missing Observer pattern ☹

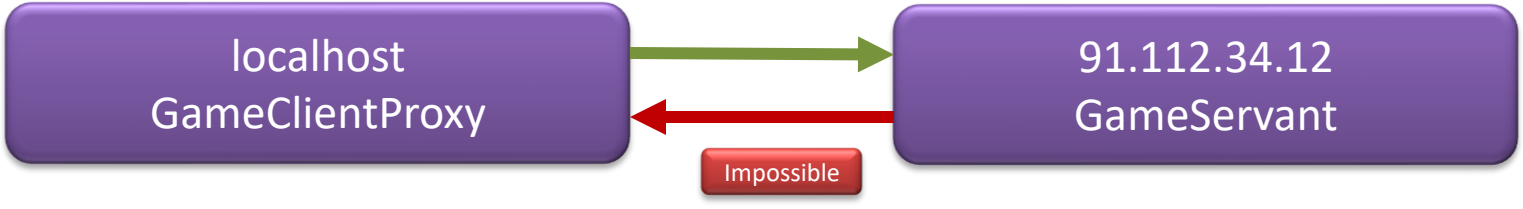
AARHUS UNIVERSITET

GUI Updates: Analysis

- How did it work in the HotSeat variant?
  - Example: *User clicks the Hero figure which is Thai Chef:*
    - *Call game.usePower(…)*
    - *Game calls proper observer methods*

```
onUsePower(FINDUS)          - due to the mutator being called
  onHeroUpdate(FINDUS)      - due to hero spending mana
  onHeroUpdate(PEDDERSEN)   - due to opponent health reduced
```

    - *HotStoneDrawing implements these, and redraws Gfx*
      - In the 'onHeroUpdate()' methods
- Our issue here: *Server cannot call 'onHeroUpdate()'*



AARHUS UNIVERSITET

Missing Observer

- Where does that lead us?
  - FRDS.Broker is a lousy framework! ☺
    - No, but it respects the client-server paradigm, as REST does…
    - *It is not allowed, because it is an architecturally bad idea…*
  - Build observer handling into the GameClientProxy?
    - Hm…
    - **Exercise**: What is difficult here???

```
@Override
public Status usePower(Player who) {
    observerHandler.notifyUsePower(who);
    // ??? WHAT ELSE
    return requestor.sendRequestAndAwaitReply(singletonId,
            OperationNames.GAME_USE_POWER, Status.class,
            who);
}
```

```
onUsePower(FINDUS)         - due to the mutator being called
  onHeroUpdate(FINDUS)     - due to hero spending mana
  onHeroUpdate(PEDDERSEN)  - due to opponent health reduced
```

AARHUS UNIVERSITET

Missing Observer

- HotStone is a really tricky game, as the open-ended world of ‘card effects’ and ‘hero powers’ allows new variants to be made *but only the server knows what happened!*
  - The idea, nevertheless, may make sense in other games
- What ever we do client side, it will always be a *qualified guess*
  - And we *will guess wrong!!!*
    - *Inconsistent GUI* *Findus sees one thing, Peddersen another*
- ***Proposal rejected!***

AARHUS UNIVERSITET

Second Proposal

- The path you will take in the Broker II is a *performance wise catastrophe* – but it works!
  - SWEA is foremost a learning experience, not product development…
- **The Brute Force Redraw Approach**
  - ☹ because it is very expensive performance wise…
  - (and energy-wise ☹☹)

- Proposal: *As we do not know what happened, we simply redraw everything upon every mutation call…*

AARHUS UNIVERSITET

Brute Force Redraw

- Fortunately MiniDraw has the method required

```
drawing.requestUpdate();
```

- In the delivered HotStoneDrawing implementation

```
/** Request update means rebuild Gfx from scratch. */
@Override  no usages  hbc@small22.racimo <hbc@cs.au.dk>
public void requestUpdate() {
  removeAllFigures();
  createAndAddFiguresForGameState();
}
```

- Downside
  - About 40-50 remote calls ☹

```
---> BruteForce Redraw
--> request count: 120
--> request count: 130
--> request count: 140
--> request count: 150
--> request count: 160
<==========----> 75% EXECUTING [20s]
```

When?

- When do I need a *brute force redraw?*
  - Upon every mutator call…
- Who makes the mutator calls?
  - The tools
- Who delegates to the tools?
  - The State tool ☺
  - One small optimization
    - No redraw for null tool ☺

```
@Override
public void mouseUp(MouseEvent e, int x, int y) {
  // All exceptions will probably occur upon mutation so
  // wrap it here!
  try {
    state.mouseUp(e, x, y);
    // A REALLY BRUTE FORCE APPROACH
    if (state != theNullTool) {
      System.out.println("---> BruteForce Redraw");
      drawing.requestUpdate();
    }
  } catch (IPCException exc) {
```

AARHUS UNIVERSITET

(Counting Decorator)

- To get an idea of amount of client requestor calls

```
// Decorator with a counting decorator
requestor = new CountingDecoratorRequestor(requestor);
```

```
@Override
public <T> T sendRequestAndAwaitReply(String objectId, String operationName, Type typeOfReturnValue, Object... arguments) {
  counter++;
  if (counter % 10 == 0)
    System.out.println("--> request count: " + counter++);
  return requestor.sendRequestAndAwaitReply(objectId, operationName, typeOfReturnValue, arguments);
}
```

```
---> BruteForce Redraw
--> request count: 120
--> request count: 130
--> request count: 140
--> request count: 150
--> request count: 160
<==========----> 75% EXECUTING [20s]
```

AARHUS UNIVERSITET

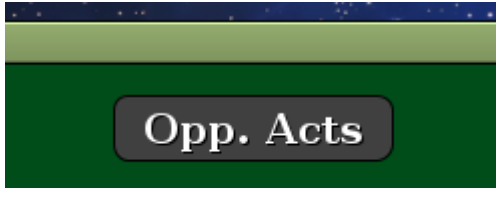
… Back to the Tool

We are actually almost done!

- We need a new state tool which is *almost identical but not quite*

```
editor.open();
editor.setTool(new DualUserInterfaceTool(editor, game, whoToPlay));
```

  - *Which, qua the previous analysis, does brute force redrawing after each mutation call…*
- In addition, there is one new button to cope with:
  - The ‘refresh’ button
    - You tool must handle clicking this…



AARHUS UNIVERSITET

OpponentButtonTool

- What should an OpponentButtonTool do?
  - Well – redraw the GUI, right ☺
- One missing thing
  - The Blue Message boxes are gone ☹
  - *Leave it at that is quite ok…*
    - *Same argument – we do not know what to write in them*
  - *[How come Henrik's remote game does include them???]*

AARHUS UNIVERSITET

A Third/Forth Path

- The performance penalty is problematic
  - Testing on localhost is probably OK
  - Having a server in Amsterdam is not OK

- I will discuss another path taken in the HotStone game server…
  - … in my energy-efficiency talk
    - Zillions of network calls wastes quite a lot of energy!

AARHUS UNIVERSITET

Broker 2.3

- Split the ‘blob invoker’ from 2.2 into subinvokers, following the principles in multi type dispatch.
- Ala
  - RootInvoker
    - Delegating to
      - GameInvoker
      - CardInvoker
      - HeroInvoker
- FRDS §5.4 issue 2

Multi Type Dispatching

Consider an **Invoker** that must handle method dispatching for a large set of roles. To avoid a *blob* or *god class* **Invoker** implementation, you can follow this template:

- Ensure your *operationId* follows a mangling scheme that allows extracting the role name. A typical way is to construct a String type *operationId* that concatenates the type name and the method name, with a unique seperator in between. Example: “FutureGame_getToken”.
- Construct **SubInvoker**s for each servant role. A **SubInvoker** is role specific and only handles dispatching of methods for that particular role. The **SubInvoker** implements the **Invoker** interface.
- Develop a **RootInvoker** which constructs a (key, value) map that maps from role names (key) to sub invoker reference (value). Example: if you look up key “FutureGame” you will get the sub invoker specific to the **FutureGameServant**’s methods
- Associate the **RootInvoker** with the **ServerRequestHandler**. In it’s *handleRequest()* calls, it demangles the incoming *operationId* to get the role name, and uses it to look up the associated **SubInvoker**, and finally delegates to its *handleRequest()* method.

AARHUS UNIVERSITET

Broker 2.3

- This exercise *does* give points in your score, but…
- It is valid to skip it entirely
  - Permanently on the backlog
- At the Exam, I have yet to see a student who solves the exam exercise so fast, that we get to discuss multitype dispatch

AARHUS UNIVERSITET

Side Note

How I have solved the
‘missing Observer’ issue.

In a Compositional Way

Idea

- The observer events are there!
  - They happen on the server side
- We just can't get them over the network
  - Broker does not allow *server to call an Observer on client*
  - Only Pass-by-Value

- One Solution:
  - *Record* observer events as 'pass-by-value' objects on server
  - *Transfer* this List<ObserverEvent> to client
  - *Replay* this list of events on the client side

AARHUS UNIVERSITET

GameEvent Recorder

- How to Record all observer events?

```
public class GameEventRecorder implements GameObserver {  18 usages  He
  private List<GameEvent> gameEventList;  13 usages

  public GameEventRecorder() { gameEventList = new ArrayList<>(); }
```

```
@Override  Henrik Bærbak Christensen *
public void onAttackCard(Player playerAttacking, Card attackingCard, Card defendingCard) {
  gameEventList.add(new GameEvent(GameEvent.Type.ON_ATTACK_CARD, playerAttacking,
          getIDOf(attackingCard), getIDOf(defendingCard)));
}
```

AARHUS UNIVERSITET

Add that observer to Game

- Actually need one recorder for each player!
  - Why?

```
// Register a recorder on all game event notifications, one for each
// player (so findus does not replay/clear events that peddersen
// still needs to replay)
for (Player player : Player.values()) {
  GameEventRecorder eventRecorder = new GameEventRecorder();
  game.addObserver(eventRecorder);
  gameContext.assignEventRecorderToPlayer(player, eventRecorder);
}
```

Client Side

- The GameClientProxy's call of any mutator must now
  - Ask server for that list of events (and clear it server side)
  - Replay the events

- This is *not the responsibility of a ClientProxy!*

- We know the pattern to solve this, right?

AARHUS UNIVERSITET

Decorator

- Decorator: Add behavior to existing class

```
public interface MyGame extends Game {  3 imp
  void requestRecordedEventListAndReplay();
}
```

```
public class MyGameEventReplayDecorator implements MyGame, Identifiable {
  private final ObserverHandler observerHandler;  3 usages
  private final GameEventPlayer replayer;  2 usages
```

```
private boolean requestEventListAndReplay() {  6 usages  Henrik Bærbak Christensen +3
  // Next, do one more call to servant, requesting all recorded
  // events emitted through the observer of game state changes
  Type collectionType =
          new TypeToken<List<GameEvent>>() {  Henrik Bærbak Christensen
          }.getType();

  List<GameEvent> theEventList =
          requestor.sendRequestAndAwaitReply(decoratee.getID(),
                  OperationNames.GAME_GET_RECORDED_EVENTS_AND_CLEAR, collectionType,
                  whoAmIPlaying);
```

```
  // and then replay it on the local observer
  boolean replayedAWonOrEoTEvent = replayer.replay(theEventList);
```

AARHUS UNIVERSITET

Hidden Method

- Relies on a Hidden Method in the Invoker

```
requestor.sendRequestAndAwaitReply(decoratee.getID(),
        OperationNames.GAME_GET_RECORDED_EVENTS_AND_CLEAR, collectionType,
        whoAmIPlaying);
```

```
    // The Hidden method: Retrieving the chain of last observer events and Clear it
    // upon having returned it to client (so no events are duplicated.)
} else if (operationName.equals(OperationNames.GAME_GET_RECORDED_EVENTS_AND_CLEAR)) {
    // Get all recorded observer notified events
    List<GameEvent> eventList = lobby.getEventListFor(objectId, who);
    // Form the reply
    reply = new ReplyObject(HttpServletResponse.SC_OK, gson.toJson(eventList));
    // and clear the event recorder for the requesting player

    lobby.clearEventList(objectId, who);
```

AARHUS UNIVERSITET

Latest Addition

- Add ‘auto fetch every 2,5 seconds and reply’
- *Not the responsibility of the ReplayDecorator*
- We know the pattern to solve that ☺

```
public class MyGameAutoEventReplayDecorator implements MyGame {
```

```
    timer.scheduleAtFixedRate(new RetrieveEventListAndReplayTimerTask(),
            delay: 0, DELAY_BETWEEN_SERVER_PULLS_SECOND);
}

class RetrieveEventListAndReplayTimerTask extends TimerTask {  1 usage
  @Override  hbc@small22.racimo <hbc@cs.au.dk> +1
  public void run() {
    // Contrary to intuition, you need to pull the event list
    // from the server even when you yourself is in turn. Otherwise
    // I ran into a occassional bug in which the server state
    // changed playerInTurn before adding the event to the event list
    // which caused the client UI to never refresh the EndTurn button.
    var drawing = objectManager.getDrawing();
    if (drawing != null) {
      try {
        drawing.writeLock().lock();
        requestRecordedEventListAndReplay();
      } finally {
        drawing.writeLock().unlock();
      }
    }
  }
}
```

AARHUS UNIVERSITET

So a double decorated Game

- Creating the resulting ‘Game’ is a three step process

```
// Make the normal game proxy for 'simple' interaction
GameClientProxyMarker gameClientProxy =
        proxyFactory.createGameClientProxy(requestor, gameid, clientNameService);

// And decorate it as a MyGame, i.e. a game instance which can
// request all recorded observer events to be replayed client side.
// IMPORTANT We must keep the same id and name service !
MyGame myGame = new MyGameEventReplayDecorator(requestor,
        gameClientProxy, proxyFactory, whoToPlay);

// MyGame and DrawingEditor have a mutual dependency in the case
// of using the AutoReplay of observer effects, so we create an
// object manager to handle that mutual dependency
ObjectManager objectManager = new ObjectManager();

// And decorate THAT with the ability to pull the event list from the server
// every N seconds; iff autoload==true
if (autoload) {
  myGame =
          new MyGameAutoEventReplayDecorator(myGame, objectManager,
                  whoToPlay, DELAY_BETWEEN_SERVER_PULLS_MILLISECOND);
```

AARHUS UNIVERSITET

Compositional Design

- Morale:
  - I needed to add a lot of complex additional behavior
  - But I used ***compositional design*** to **change by addition!**
- Only *adding new classes, not changing any!*
  - Event recording by **adding an observer** server side
    - And separate classes to record events and replay them
  - Fetching event list from server by **adding decorators** on the GameClientProxy

- **All additions are under automated test control!**

```
public class TestTransferGameEventsOverNetwork {
  private Game servantGame,  2 usages
          clientGame;  26 usages
```

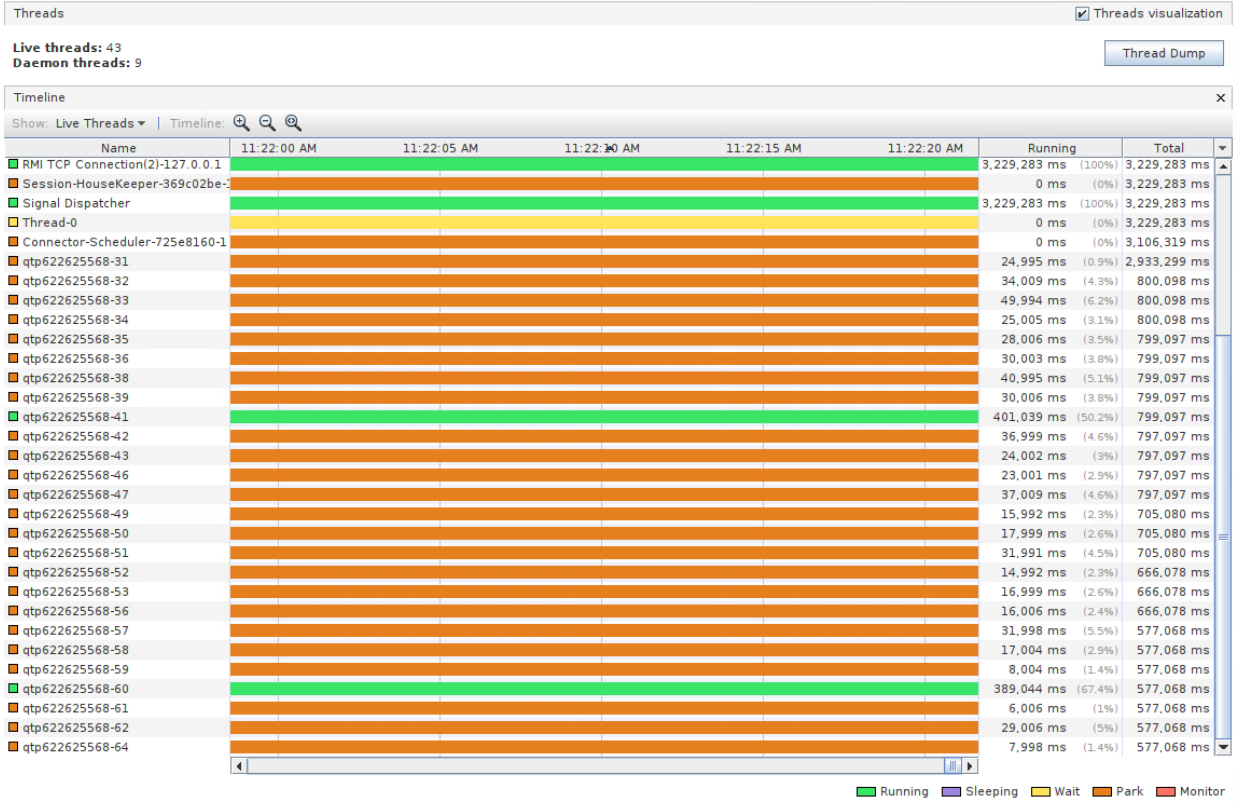
AARHUS UNIVERSITET

HotStone Server Loads

Modern CPUs are bored…

hotstone.littleworld.dk

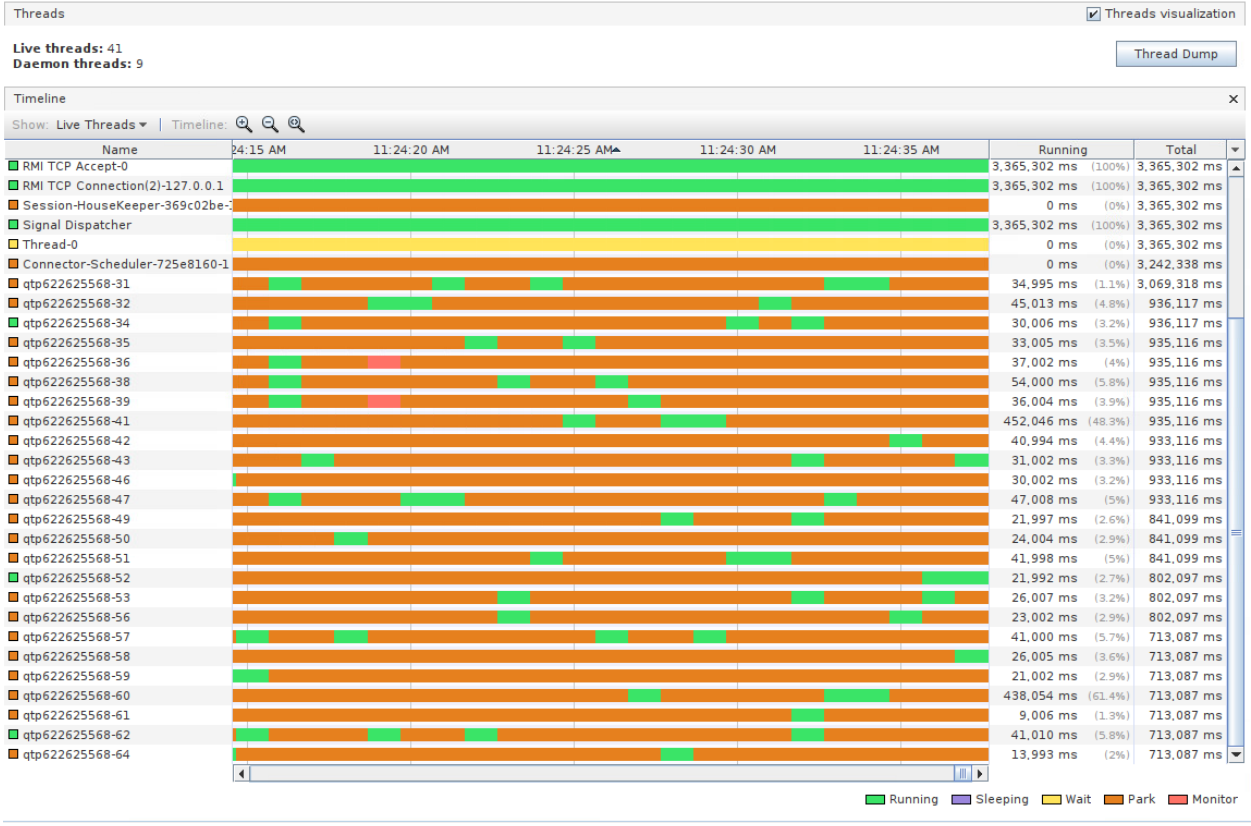
- HotStone game server – with no games running
  - Worker threads are “parked”
  - HTTP broker uses Spark-Java uses Jetty
    - Which may run up to 200 threads



AARHUS UNIVERSITET

hotstone.littleworld.dk

- With 300 concurrent games running



Still way from begin loaded

AARHUS UNIVERSITET

Conclusions

Happy Coding...