Universität Basel

Software Architecture

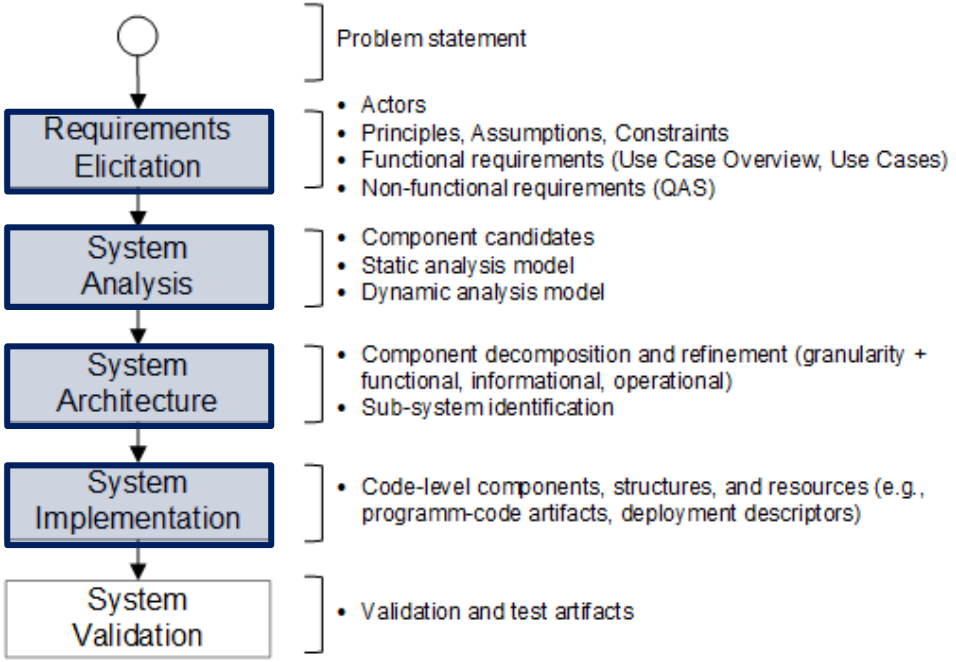
Exercise – Gaming Platform (iteration #5)

BSc

Ingo Arnold

Exercise Opening
Motivation

You **create an initial design** for a **gaming platform** by incrementally following the process outlined, below. Note that your design should focus less on the algorithmic and more on the structural solution aspects.



Exercise Opening
Motivation

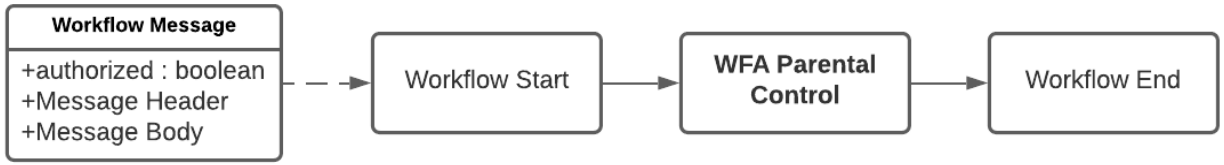
A new requirement has recently been introduced by a stakeholder group. You are expected to incorporate the requirement and respective responses into the architecture.

- The use case **add game to platform** was re-discussed in depth and the *quality assurance* stakeholder group requested to go through a comprehensive authorization process whenever new games are added to the platform.
- Quality Assurance wants to ensure that different quality concerns (e.g., parental controls, accessibility, non-maliciousness of the code base) are mandatorily reviewed by the appropriate departments.
- The requirements for the authorization process were so extensive that it was already decided to establish the capabilities of a *workflow engine*. This allows authorization processes to be flexibly changed (i.e., reconfigured) – that means: without going through recompilation cycles.
- The workflow engine requirements are outlined in more detail on the subsequent slides

Exercise Opening
Motivation

A new requirement has recently been introduced by a stakeholder group. You are expected to incorporate the requirement and respective responses into the architecture.

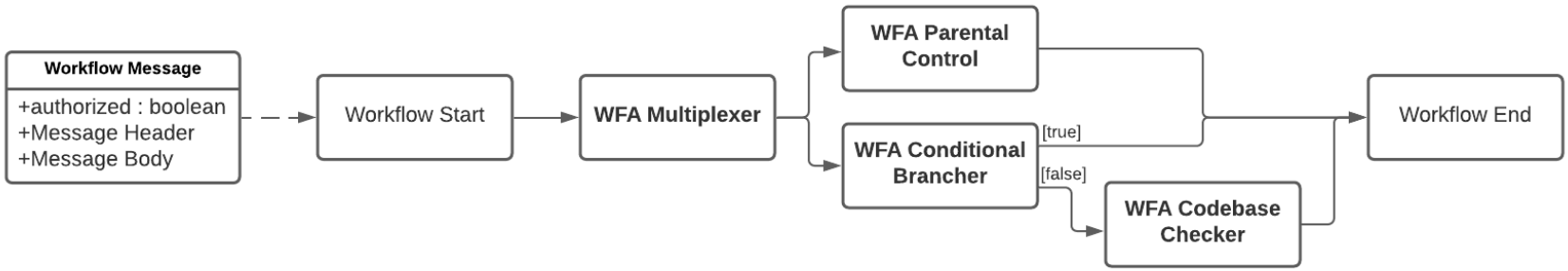
- A workflow comprises …
  - A set of workflow activities (WFA)
  - Connections between workflow activities (chaining activities unidirectionally)
  - An initial message or request (Workflow Message) that is populated into the first workflow activity and then propagated through the entire workflow (i.e. activities following the first one in the chain) through to its end
- In the simple workflow I illustrated below, a *workflow message* is populated into the component *workflow start*. Next the message is reviewed by a single workflow activity (*WFA Parental Control*), before it is finally processed by the *workflow end* component.



Exercise Opening
Motivation

A new requirement has recently been introduced by a stakeholder group. You are expected to incorporate the requirement and respective responses into the architecture.

- In a more complex workflow configuration a message (*workflow message*) is populated into *workflow start* from which it is further handed into a *WFA multiplexer* component. A multiplexer broadcasts the message into all connected components. In this case these are *WFA parental control* and *WFA conditional brancher*. Conditional brancher makes a decision (based on defined logical reasoning) and forwards the message to either *WFA codebase checker* (if the result of the logical reasoning was *false*) or right into the *workflow end* component. *WFA parental control* hands its processed message over to *workflow end*.



Exercise Opening
Motivation

A new requirement has recently been introduced by a stakeholder group. You are expected to incorporate the requirement and respective responses into the architecture.

- The workflow framework shall provide a set of WFA types which enable the composition of workflows in a very flexible and reusable manner.
- All WFA types have the following characteristics in common:
  - A WFA component maintains connections to at least one subsequent component (e.g., *WFA parental control* maintains exactly one connection, *WFA conditional brancher* maintains two connections, while *WFA multiplexer* maintains many connections).
  - A WFA component receives *workflow messages*, processes them, and propagates the processed message to its connected WFA components for further massaging.
  - Any WFA component receives its input from either a *workflow start* or from a respective WFA component (i.e., its predecessor in the workflow chain).
  - Irrespective of the concrete work a WFA component performs on its message input, the output is propagated to either the final receiver (*workflow end*) or to the next WFA component in the chain.
  - Workflow messages are comprised of a *header* and a *body* portion. They further maintain an *authorized* flag, which is toggled to *false* if a WFA component decides to veto its workflow approval.

Exercise Opening
Motivation

A new requirement has recently been introduced by a stakeholder group. You are expected to incorporate the requirement and respective responses into the architecture.

- In an initial version of the workflow framework the following WFA component types establish the framework:
  - **WFA multiplexer** components broadcast received messages (e.g., based on a particular rule) to all registered successor WFA components.
  - **WFA parental control** components validate received messages regarding G-rated content and propagate the transformed message into their registered (singular) successor.
  - **WFA code base checker** components validate received messages regarding a non-malicious code base and propagate the transformed message into their registered (singular) successor.
  - **WFA conditional brancher** components have 2 outgoing WFA component branches (i.e., a *true* and a *false* branch). A defined branching algorithm is applied to data in the message header in order to make a branching decision and to determine which of both successors receives the message for further processing.
  - **Workflow messages** contain a set of name-value pairs as header attributes while the respective game is included in their body portions.

Exercise Opening
Motivation

A new requirement has recently been introduced by a stakeholder group. You are expected to incorporate the requirement and respective responses into the architecture.

- The *workflow start* component acts as a workflow configurator and passes a received message into the first component in its particular workflow.
- The *workflow end* component may receive messages from several predecessor components. It evaluates all messages and calculates an overall authorization result. Once a workflow is completed, *workflow end* components can be queried to check whether the workflow was authorized in its entirety.

Exercise Opening
Motivation

Incorporate new requirements and specify them based on quality attribute scenarios. Distinguish between use case specific and system-wide requirements, constraints and assumptions.

Analyze the new requirements and **create an analysis model** of appropriate component candidates that highlights their static and dynamic relationships.

Create the complementary facets of the system architecture in the form of an architecture approach. Clarify component interfaces and specifically consider patterns that might come in handy in implementing the new requirements.

Finally, **implement your design in code** and assemble the individual parts into an overarching solution.

Exercise Agenda

- Requirements Elicitation
- System Analysis
- System Architecture
- System Implementation

Requirements Elicitation
Incorporate new Requirements

Specify a scenario that reflects the more complex workflow, above.

Requirements Elicitation
Incorporate new Requirements

Specify a scenario that reflects the more complex workflow, above.

1. A *chess* game is created and included in a workflow message body.
2. The message header is set to the name of the game.
3. A WFA parental control component is connected to WFA multiplexer component.
4. The workflow end component is connected to the WFA parental control component.
5. The workflow end component is also connected to a WFA conditional brancher (true-branch).
6. A WFA codebase checker component is connected to the same WFA conditional brancher (false-branch) and the workflow end component is connected to the WFA codebase checker.
7. The WFA conditional brancher is finally connected to WFA multiplexer.
8. WFA multiplexer (as the first WFA component) is registered with workflow start.
9. The message is populated to workflow start and propagated further through the entire workflow. Each individual WFA component performs its duties on the received message and sets the authorized flag, accordingly.
10. Workflow end receives multiple messages (one from each of its predecessors) and calculates the overall authorized flag.
11. Workflow end is queried for whether the message was accepted or rejected by the entire workflow.

Exercise Agenda

- Requirements Elicitation
- System Analysis
- System Architecture
- System Implementation

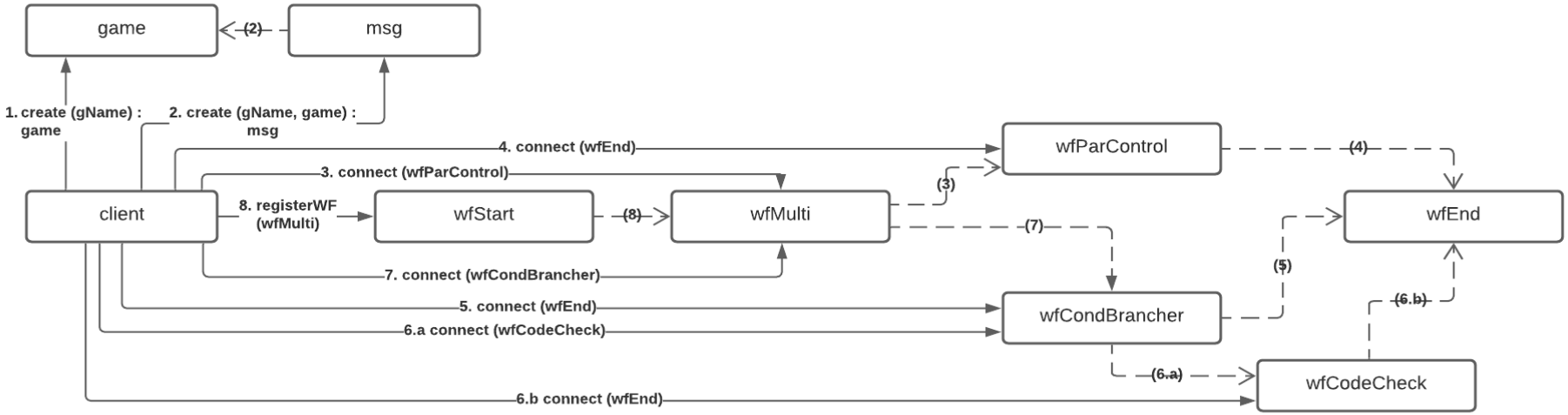
System Analysis
Scenario

Create the scenario in the form of interacting objects (activity diagram).

System Analysis
Scenario

Create the scenario in the form of interacting objects (activity diagram).

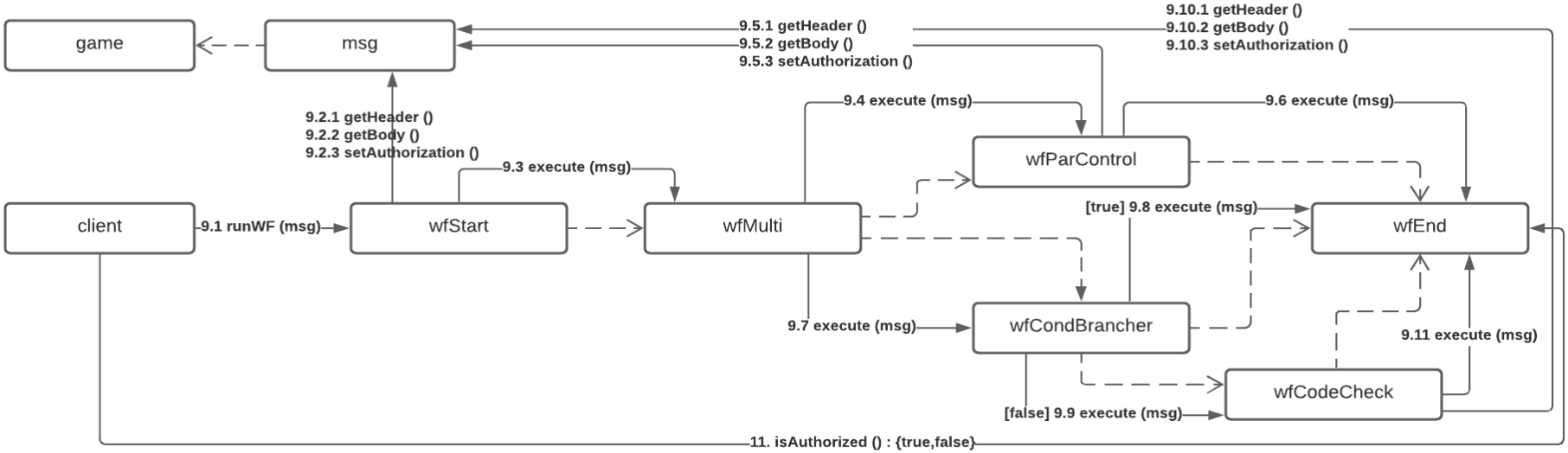
- Building and registering the workflow with the workflow start component.



System Analysis
Scenario

Create the scenario in the form of interacting objects (activity diagram).

- Performing the workflow and querying authorization status.



Exercise Agenda

- Requirements Elicitation
- System Analysis
- System Architecture
- System Implementation

System Architecture
Systematic Development

Systematically develop an object-oriented design in the development view for which you perform the steps, below.

1. Create a list of all classes to be programmed, so that ...
   - each object can be instantiated by exactly one class
   - identically behaving objects are instantiated by a common class
   - each class instantiates at least one of the required objects
2. Each arrow arriving at an object (e.g., in a sequence diagram) represents the activation of a method. For these you provide a callable (public) method in the corresponding class. The result of this step are complete method signatures including *return value* and *parameters* as well as additional *information about method behaviour*. This means at this stage you also consider which information is passed between objects.

Universität Basel

System Architecture
Determine Classes

Create a list of all classes to be programmed.

Class	Object(s)

System Architecture
Determine Classes

Create a list of all classes to be programmed.

Class	Object(s)
Game	game
WFMessage	msg
WFStart	wfStart
WFMultiplexer	wfMulti
WFParControl	wfParControl
WFEnd	wfEnd
WFCondBrancher	wfCondBrancher
WFCodeCheck	wfCodeCheck
WFBrancher	super type for WFCondBrancher
WFBroadcaster	super type for WFMultiplexer
WFTask	super type for WFCodeCheck, WFParControl
WFBuildingBlock	super type for WFBroadcaster, WFBrancher, WFTask

System Architecture

Determine Class Methods and Signatures

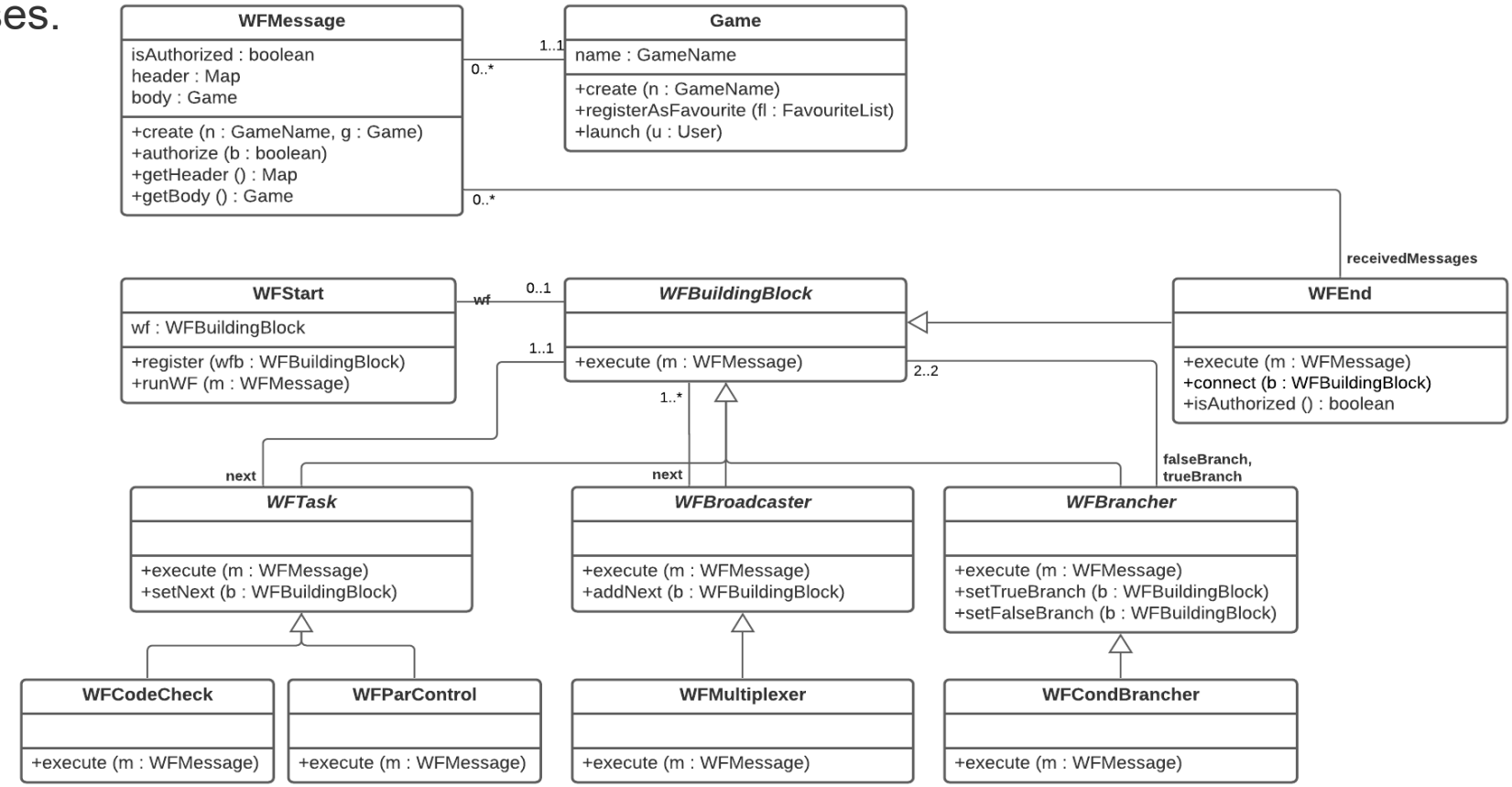
Determine methods, method signatures, and relationships between identified classes.

Universität Basel

System Architecture

Determine Class Methods and Signatures

Determine methods, method signatures, and relationships between identified classes.



Lecture Agenda

- Requirements Elicitation
- System Analysis
- System Architecture
- System Implementation

System Implementation
Implementation in Java


```
* package com.unibas.exercise.gameplatform.increment5.workflow;
```

Questions