Department Mathematik/Informatik, Abteilung Informatik
Software & System Engineering
Lecture Software Engineering, WS 2024/25

Lukas Scheib, Alexander Korn,
Adrian Bajraktari, Prof. Dr. Andreas Vogelsang

UNIVERSITÄT ZU KÖLN

Homework 2. System Modeling and Architecture

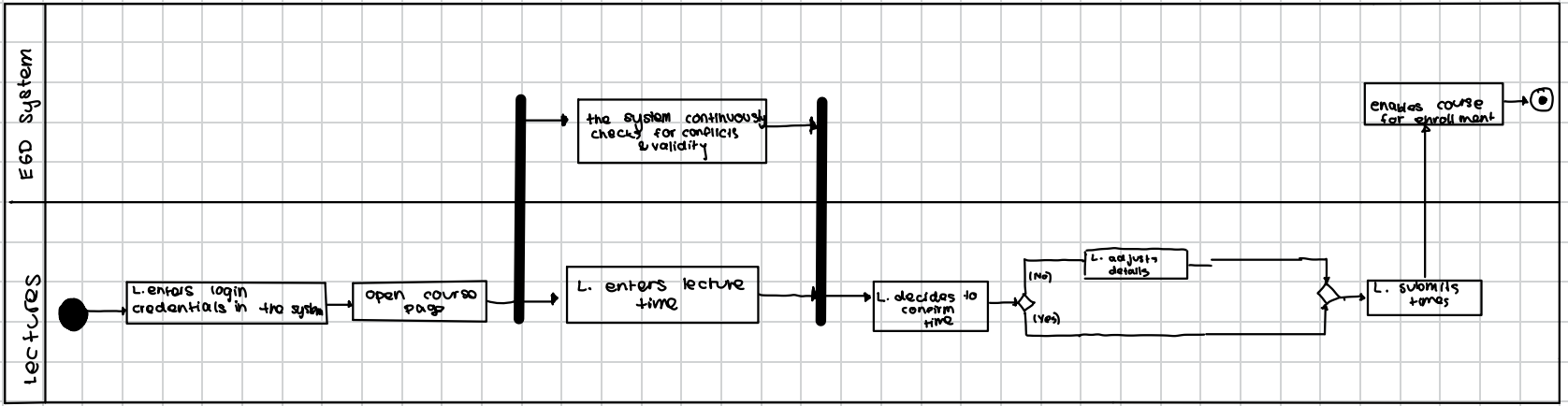
Hand out: 23.10.2024, 18:00
Hand in: 06.11.2024, 16:00

General Remark: Start early so you can ask questions to your tutor in the exercise groups.

Exercise 1. Process Modeling with Activity Diagrams

One of your team members modeled the following use case:

Title	Lecturer entering exercise times in EGD
Description	This use case describes the interaction of lecturers with the Exercise Group Distribution (EGD) system to enter exercise/lab groups for their courses.
Actors	Primary Actor: Lecturer, Secondary Actor: EGD System
Preconditions	• All courses that are offered in the semester have been automatically created by copying them from Klips. • The lecturer has access to the system.
Basic Flow	1) The lecturer enters their credentials to log in to the system. 2) They open up their course's page. 3) While they enter their lecture times and exercise times, the system continuously quick checks them for validity and whether there are initial conflicts. 4) When the lecturer is done, they confirm their changes. 5) The system checks all times extensively and generates a report that shows all open conflicts with other courses. 6) The lecturer may choose to adjust times or to just leave it as is (see alternative flow). 7) At last, the lecturer submits and the system enables the course in the course overview for students.
Alternative Flow (Ajustment)	• The lecturer adjusts the details of their course and resubmits the changes.
Postconditions	• The course is available for students to enroll to.

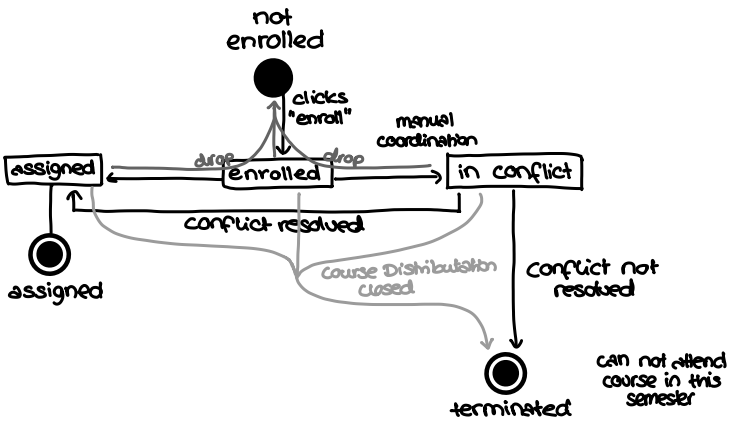

EGD System
Lectures
L. enters login credentials in the system
open course page
the system continuously checks for conflicts & validity
L. enters lecture time
L. decides to confirm time
(No)
(Yes)
L. adjusts details
L. submits times
enables course for enrollment

Exercise 2. State-Machine Modeling

In an interview with a group of lecturers, you identified the following possible states a student can have within a specific course:

- Initially, they are not enrolled in the course.
- When they click "enroll", they enroll in the course. At this point, they are not assigned, yet.
- When the assignment process starts, there are to possibilities: Either, the student can be assigned with no problems, thus having the state "assigned", or there are conflicts that need to be resolved manually in coordination between lecturer and student, thus idling in the "in conflict" state.
- If the student and lecturer can not resolve the conflict, the student's status is terminated as they can not attend the course this semester.
- In any case, when a student is enrolled and drops out of the course, they return to be "not enrolled".
- In any case, when the lecturer closes the course distribution, all student states are terminated.
- Students are still regarded "enrolled" when they are in the states "not assigned", "assigned", and "in conflict".

You decide to model this in a UML state-machine diagram so you can clarify with the lecturers that your team understood everything correctly.



3)

a)
We face a **problem** similar to one described in the lecture: *clients need the same functionality but with different presentations*. This calls for (**idea** of) separation of data presentation (user interface, frontend webpage), application logic (components for managing courses, distributing courses, and handling communication), and data management (KLIPS).

Reasons:
- *Modularity* (Role Separation): The three-tier architecture clearly separates the EGD system into layers, making it easier to manage each part rather independently, not because they are independent from each other, the tiers are interdependent but *functionally separated*. Each tier has specific role: the Presentation Tier handles user interactions, the Logic Tier processes application tasks, and the Data Tier manages storage.
- *Simplicity*: The three-tier architecture is well-suited for medium-sized applications due to its relatively simple pattern, simplifying design, development, testing, and deployment. For a larger system, a microservices architecture might be preferable. However, compared to microservices, this architecture is likely more cost-effective for the EGD system, as it's less complex and avoids additional costs (e.g., the EGD system doesn't require each service to be entirely self-contained, with its own data, logic, and presentation).
- *Maintainability*: This architecture makes it easier to update specific layers without affecting others (functional separation). For example, if a change is needed in one tier, it should have minimal or no impact on other tiers, which simplifies maintenance and reduces the risk of unintended side effects.
- *Scalability*: While not as scalable as a microservice-architecture, I'd say a three-tier architecture still allows some flexibility for scaling specific tiers (without a rework of entire application).

b)