NTNU
Kunnskap for en bedre verden

Department of Computer Science

TTM4115 - Design of Communicating Systems

The Digital RAT System

Authors:

Andreea Gheorghe

Helene Amlie

Henrik Granlund

Marius Arder

Nina Skresholdt Torvund

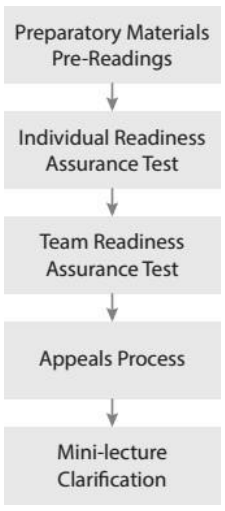
Thomas Grønbakk Nilsen

April 2023

Contents

1 Problem & Background

Having the students coming to class prepared is critical in order to have any possibility of deeper classroom conversations and meaningful problem-solving activities. Students need to be accountable for their initial preparation and ready for the activities. As a part of their team-based learning philosophy, the Department of Information Security and Communication Technology (IIK) at NTNU (Norges Teknisk Naturvitenskapelige Universitet) has implemented the Readiness Assurance Process (RAP) in many of their subjects. The RAP is a five-stage process that is used at the beginning of each module in your course. The RAP ensures that students understand the fundamental concepts, definitions, and foundational knowledge they need to begin problem-solving.


Preparatory Materials Pre-Readings
Individual Readiness Assurance Test
Team Readiness Assurance Test
Appeals Process
Mini-lecture Clarification

The second and third steps of the RAP include an individual and a group test, called Readiness Assurance Tests (RATs). When implementing these tests in their classes, IIK decided to do so through scratch cards distributed to the students in class. There are three problems with this approach:

1. Inefficiency
2. Paper waste
3. Unavailability for remote students

Firstly, the current solution is inefficient in that a noticeable amount of time is used to physically distribute these scratch cards. In addition to this, there is no good way to efficiently know when to distribute the team RATs, i.e., when all members of a group are finished with their individual RATs. The time wasted on these inefficiencies could instead have been used for lectures and team-based learning activities.

Secondly, the current solution introduces unnecessary paper waste. A class of 100 students, organized into 20 groups of 5 people, results in 120 questionnaires (1 for each student + 1 for each team), in addition to 1 scratch card for each team, having to be printed out for each test. Many of IIK's classes have such tests every week, meaning that for a semester, several hundreds of scratch cards are printed and produced. Adding to this, questionnaires for students that are calling in sick or unable to attend the class physically are still printed, as the current solution for printing assumes all students are present.

Lastly, the current solution does not consider that modern teaching styles and team-based cooperation express the need for digital or hybrid solutions. The pandemic showed the importance of being agile and flexible when it comes to work and communication platforms. The current solution is only available in physical form for the students present in class.

2 Vision

Digital RAT System (DRS) is a digital version of the traditional RAT system used in subjects as a method for team-based learning. The DRS aims to increase efficiency and decrease the usage of paper. Teachers can create RATs, store them in a database, and publish them when wanting to conduct a RAT session for the students.

Through an application, the students will get access to the RATs digitally, allowing them to participate remotely from anywhere at the time when the RAT sessions are held. The DRS does not provide the team with a specified method for communicating during the tRATs. The teams are encouraged to communicate in whatever way they seem to fit themselves. After the iRAT is conducted and the whole team is finished, the team can start the tRAT, which facilitates collaboration and cooperation among group members into finding solutions to the current week's unit questions. During the tRAT the team gets instant feedback when they answer a question.

3 Objectives

- BO-1 - Improve time-effectiveness
  - Automating and digitalizing the RATs.
  - Reduce wasted time by 100%, by allowing a team to start their tRAT immediately once all members are done with their iRAT.
  - Reduce wasted time by 100%, by allowing teams to begin their class activities immediately after they have completed their tRAT.
  - Automated correction and registration of results.
  - Provide quantifiable data on participation for each RAT and the average time spent.
- BO-2 – Increase attendance rate by allowing remote participation
  - Allow for remote RAT participation, which is not possible today.
  - Measurable through attendance rate.
- BO-3 – Reduce operating and environmental costs
  - Phase out scratch cards (For TTM4115 this can reduce operating cost by ~$0.3 per scratch card, ~$5 per week).
  - Remove the need for printed RATs (For TTM4115 this can reduce paper usage by 100%, ~115 pages per week).
  - Operating costs in the form of labor will also be reduced as there is no need for manual registration of results.

4 Stakeholders

Stakeholder	Values	Attitudes	Interests	Constraints
Students	Efficient and well-structured RAT system. High learning outcome; Good opportunities to help each other during tRATs; Solid useability.	Expectations of a well-developed system.	HIGH Simple user experience. System is available from anywhere.	Mobile device with network connection necessary.
Teaching Assistants (TAs)	Easy overview over students' progress.	Strong enthusiasm to make the labs more efficient, interactive and get better overview of students' progress	MEDIUM Overview of students' progress; less administrative work	Mobile device with network connection necessary.

Professors/ Course Coordinator	Tracking task-progress. Adds value to course.	Enthusiastic to improve a system to help the learning-process and simplify it.	HIGH Overview of students' progress; adjust future syllabus based on students' results on RATs. Compare with previous years. Indicates difficulty of tasks	Mobile device with network connection necessary. Change from analog system to digital system
University Management	Well-educated students; Better external reputation.	Open and optimistic for the system	LOW Gives value to students, TAs and course coordinators for RATs.	Operational costs for database and server

5 Use Cases

5.1 Use case hierarchy

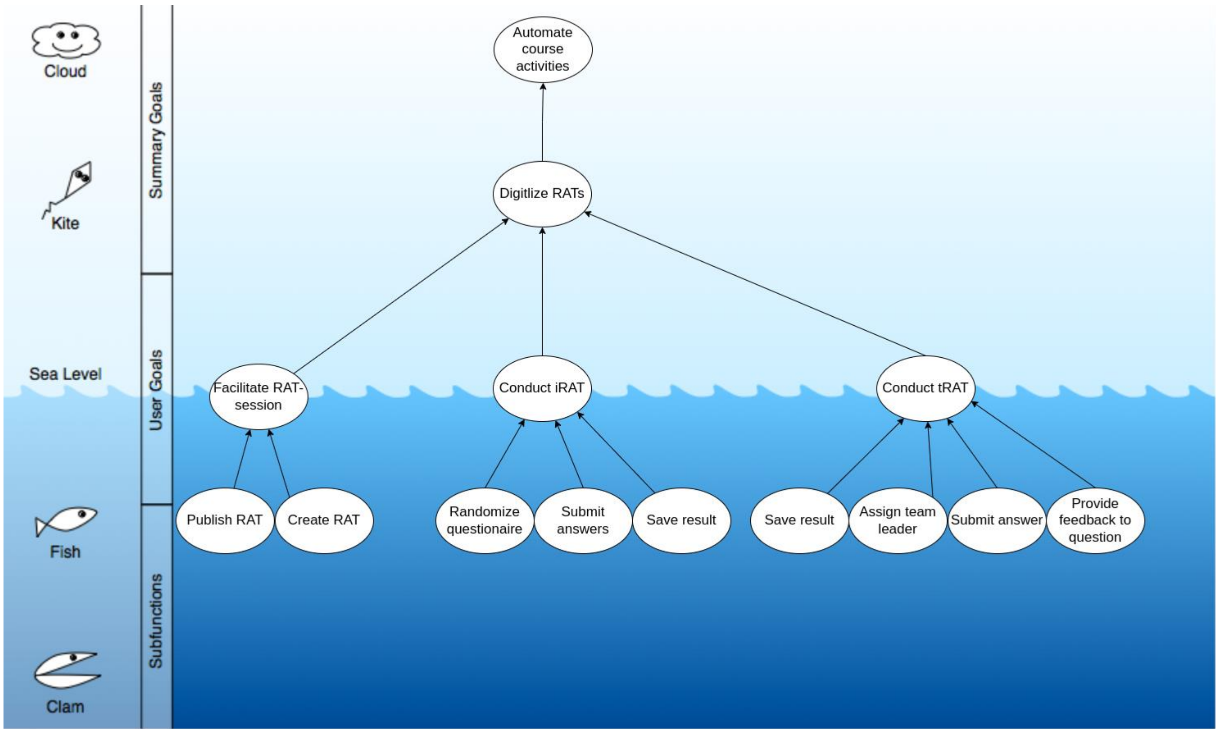


Figure 1. Use Case Hierarchy

5.2 User classes

The various user classes are identified as primary actors for the use cases for the DRS:

For the DRS and in our use case scenarios we have decided not to distinguish between professors, TAs or other people involved in managing class activities. We have made a general user class named

Teacher. For future development of the DRS, a distinction between the different roles regarding the people facilitating class activities and what they are allowed to do, should be considered. I.e., only teachers should be able to create RATs, but TAs should be able to start RATs.

Primary actor	**Use cases**
Student	1. Conduct a digital iRAT 2. Conduct a digital tRAT
Teacher	3. Facilitate RAT-session

5.3 Use case descriptions

ID and name:	**UC-1 Facilitate RAT-session**	**UC-2 Conduct iRAT**	**UC-3 Conduct tRAT**
Primary actor:	Teacher	Student	Student
Secondary actor(s):	Student, Database	Database	Teacher, Database
Description:	The teacher creates a RAT by entering questions, answer options and the correct answer into a predefined user interface. The system stores the RAT in the database, to be distributed to the students. When ready, the teacher publishes a specified RAT to the students on their page for a given time window, default 20 minutes if not specified otherwise.	A student accesses the RAT of the day and must finish it within a specified time limit. The student is shown a number of multiple-choice questions and chooses one answer per question. Upon completion or timeout, answers are sent to the database and the student is redirected to the tRAT waiting room. A student will receive the iRAT with questions and corresponding answers in randomized order.	After the entire team has delivered their iRAT, one student from the team is chosen as a team leader. The team leader is provided withthe questions and answers them on behalf of the team. Upon beginning the tRAT, a timer is started. When all questions are answered or the timer runs out, the answers are sent to the database.
Trigger:	Teacher indicates that they want to create a RAT	A digital iRAT is made available on the DRS and the student begins the test.	The final team member submits their iRAT or iRAT-timer expires.
Preconditions:	**PRE-1A** Database is established.	**PRE-2A** RAT is created and the student is logged in when the RAT is made available to the students.	**PRE-3A** RAT is created and the student is logged in when the RAT is made available to the students. **PRE-3B** iRAT is submitted and students are placed in tRAT

			waiting room.
Postcondition:	**POST-1A** RAT is stored in the database and can be made available for later sessions. **POST-1B** Global RAT-timer is started.	**POST-2A** Individual scores are stored in the “RAT” database after completion of the iRAT. **POST-2B** Students are sent to the tRAT waiting room.	**POST-3A** Team scores are stored in the “RAT” database after completion of the tRAT.
Normal flow:	**1.0 Host RAT-session** 1. The teacher navigates to the RAT page. 2. The DRS displays stored RATs and a “CREATE RAT” button. 3. The teacher presses the “Create RAT” button, chooses how many questions to fill in the name for the RAT, and is given an input field for each question and answer. 4. The teacher inputs a question, one correct answer, and three wrong answers. 5. When finished filling in the questions, the teacher saves the RAT and sends it to the server for storing. 6. The RAT is stored in the database. 7. The teacher refreshes the interface with a push of the button “REFRESH RATS”, this prompts the server to provide a list of available RATs in the database. 8. The teacher can choose the RAT they just created and publish it to the students.	**1.0 Conduct iRAT** 1. A student navigates to the RAT page. 2. The DRS displays the available RAT. 3. The student begins the RAT and is the iRAT displayed. 4. The student chooses one answer for each question by toggling a checkbox. 5. The student submits the answers. 6. The DRS confirms the submission with a pop-up display. 7. The DRS stores the answers, calculates a score, adds it to the database and updates the completion status. 8. The student is redirected to the tRAT waiting room.	**1.0 Conduct tRAT as team leader** 1. The DRS displays the tRAT to the team leader's device. 2. The student is provided withthe questions from the RAT, with the corresponding answer options. 3. The student is given instant feedback for each question, where only a correct answer will let them continue. Teams will only get credited if they answer correctly on the first try. 4. The student submits the answers or is forced to deliver the answers if the time runs out. 5. The DRS confirms the submission with a pop-up display. 6. The DRS stores the answers, calculates a score, adds it to the database and updates the completion status. 7. The student is sent to the log-in page.

Alternative flows:	**1.1 Host RAT session with previously created RAT** 1. A teacher navigates to the RAT page. 2. The DRS displays stored RATs and a "CREATE RAT" button. 3. The teacher chooses an existing RAT to be published forthe students. **1.2 Teacher wishes only to create a RAT** 1. Step 1-5 from normal flow	**1.1 iRAT is finished by timer running out** 1. Step 1-4 from normal flow 2. Time window passes 3. Step 6-8 from normal flow	**1.1 Conduct tRAT non-team leader** 1. The student is placed in a waiting room, while the team leader conducts the tRAT on behalf of the team. 2. The student is sent to the log-in page when tRAT is delivered.
Exceptions*:	**1.0 E1 Teacher exits the process** 1. Teacher exits the "create RAT" process before saving or choosing title.	**1.0 E1 Student exits the session** 1. DRS stores the temporary answers. 2. Upon re-entering, the student is able to obtain thei old answers. 3. If the timer expires, DRS stores the answers, calculates scores, adds them to the system database, and updates completion status.	**1.0 E1 Team leader exits their session** 1. DRS updates scores for the whole team for each question answered. 2. Upon re-entering, the team leader is provided the question where they left off. **2.0 E2 Team leader is unable to re-enter. New leader is selected** 1. DRS updates scores for the whole team for each question answered. 2. A student from the waiting room is manually chosen as leader by the teacher and is provided the question where the groupleft off.
Priority:	High	High	High
Frequency of use:	One RAT is hosted each week.	Approximately 100 students, divided into 20 groups. The students will access the iRATs in the same time window each week.	Approximately 100 students, divided into 20 groups. The students will access the tRATs in approximately the same time window each week.
Other information:	1. No RAT should be able to be named the same (this should be handled with exception handling)		

Assumptions:	Students have to be logged in when the RAT session starts to be able to access it.	1. Questions and answers for the RAT are given in random order for each individual student. The randomization is done client side. 2. The students can see the remaining time for the iRAT in the classroom. 3. Individual scores are made available for the students after completion.	1. The team leader is responsible for communicating the question-and-answer options to the other team members. 2. Students should be able to see the timer for the tRAT in the classroom. 3. Team scores are made available for the students after completion.

**We have specified exceptions for the different use cases. Not all these exceptions have been implemented in our sequence- and state diagrams, as we consider them too complex and not worth prioritizing at this point in time. The exceptions have been documented in the use case descriptions to highlight challenges with the DRS that should be addressedat a later point.*

6 Deployment Diagram

The physical view of the DRS, illustrated in Figure 2, provides an overview of how the system is configured with respect to its hardware and infrastructure. This view depicts the network topology that is necessary to support the software system. Specifically, the deployment diagram shows how software and hardware resources are deployed in our system, and how they communicate, with the components representing the software artifacts that are deployed on the hardware nodes.

The software artifacts include state machines, managers, UIs, an MQTT broker, and a local database. The nodes in the diagram represent the various hardware components, such as PCs, phones, and servers. Overall, this physical view gives a clear picture of how the system is designed and how the different components interact with one another to achieve its intended functionality

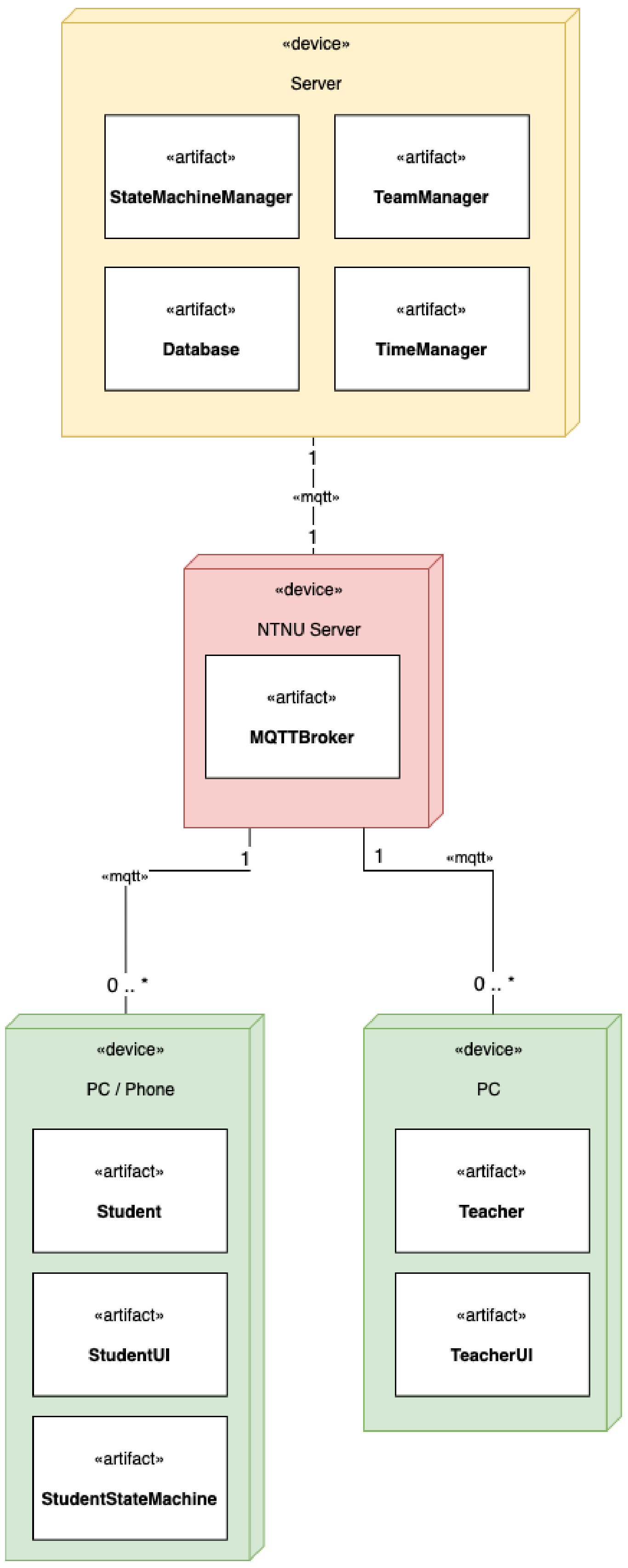


Figure 2. Deployment Diagram of DRS

7 Sequence Diagrams

This type of interaction diagram is designed to display how objects and components connect in a system over time. Sequence diagrams are an effective and intuitive way to describe communication between several communicating partners, through different scenarios, such as user interactions, system events, or software processes. There are four different sequence diagrams to show different interactions that the teacher and student can do. Figure 3 teacher would have to login to the DRS. This interaction must be successful for a user to continue with other interactions.

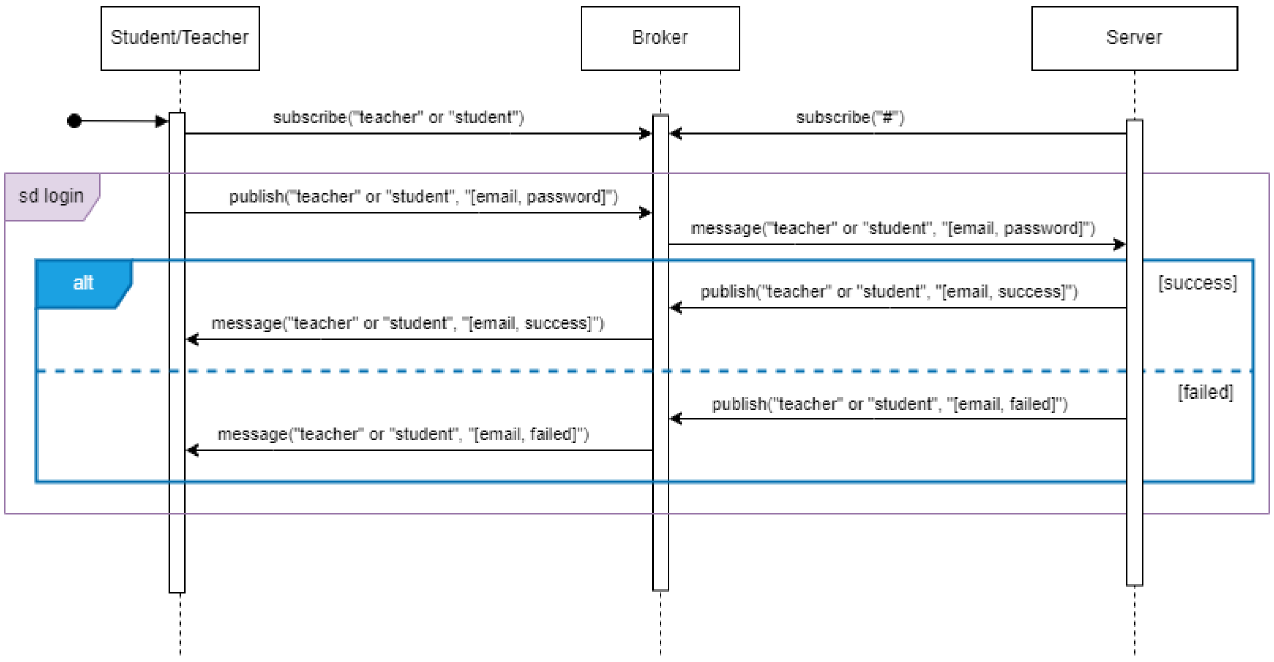

Student/Teacher
Broker
Server
subscribe("teacher" or "student")
subscribe("#")
sd login
publish("teacher" or "student", "[email, password]")
message("teacher" or "student", "[email, password]")
alt
[success]
publish("teacher" or "student", "[email, success]")
message("teacher" or "student", "[email, success]")
[failed]
publish("teacher" or "student", "[email, failed]")
message("teacher" or "student", "[email, failed]")

Figure 3. Sequence diagram for logging into the DRS

Figure 4 demonstrates how the systems communicate during an iRAT. The timer can run out before or after the student has finished the iRAT.

- [opt 1] The timer runs out before the student has finished, and any answers that have been submitted up to that point will be sent to the server.
- [opt 2] The timer runs out before at least one of the other students in the group is not finished.
- [opt 3] The entire team can finish before the timer runs out.

Only one of the opt fragments can happen during this sequence. If a student is finishing an iRAT at the same time as the timer runs out nothing should happen, and the student should continue as if they did not receive the t1_expired message.

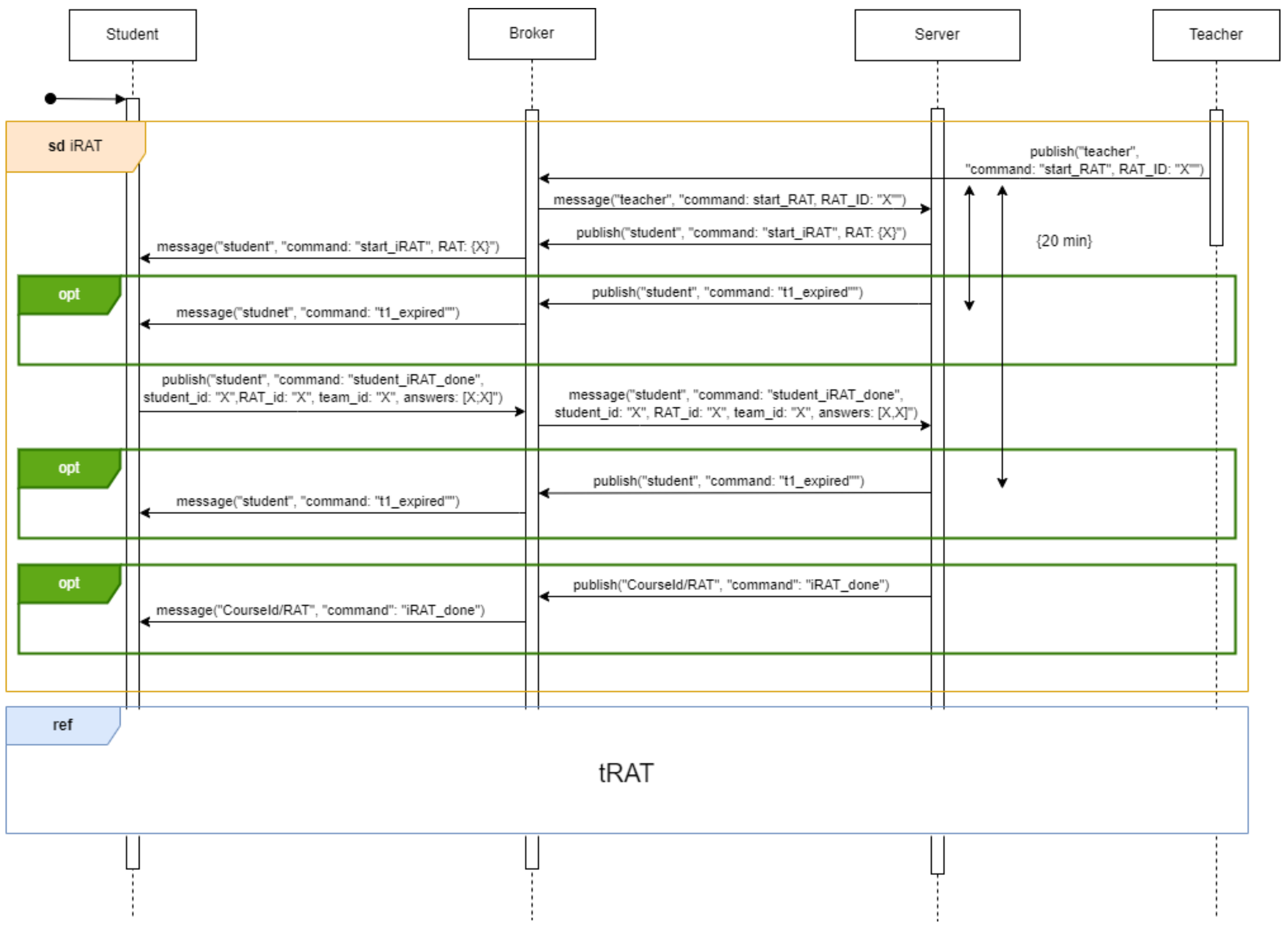

Student
Broker
Server
Teacher
sd iRAT
publish("teacher", "command: "start_RAT", RAT_ID: "X"")
message("teacher", "command: start_RAT, RAT_ID: "X"")
publish("student", "command: "start_iRAT", RAT: {X}")
message("student", "command: "start_iRAT", RAT: {X}")
{20 min}
opt
publish("student", "command: "t1_expired"")
message("studnet", "command: "t1_expired"")
publish("student", "command: "student_iRAT_done", student_id: "X",RAT_id: "X", team_id: "X", answers: [X;X]")
message("student", "command: "student_iRAT_done", student_id: "X", RAT_id: "X", team_id: "X", answers: [X,X]")
opt
publish("student", "command: "t1_expired"")
message("student", "command: "t1_expired"")
opt
publish("CourseId/RAT", "command": "iRAT_done")
message("CourseId/RAT", "command": "iRAT_done")
ref
tRAT

Figure 4. Sequence diagram for tRAT

The iRAT has to be finished by the entire group before the group can proceed with the tRAT. A team leader is chosen by the server and is communicated to the student when publishing the ‘start_tRAT’. This action of the team leader is shown in the ‘alt’ fragment below. The other team members will be sent into a waiting room. All the answers to the questions will be sent from the server when finished, so that the student can get an instant response without having to communicate back and forth with the server.

If the timer runs out before the team leader is finished with the tRAT, then the answers already submitted will be sent to the server. All the team members will receive either the “t2_expired” or “tRAT_done” message depending on whether the timer runs out or not.

Also, for this sequence nothing should happen differently for the system if the team sends the answers at the same time as the timer runs out.

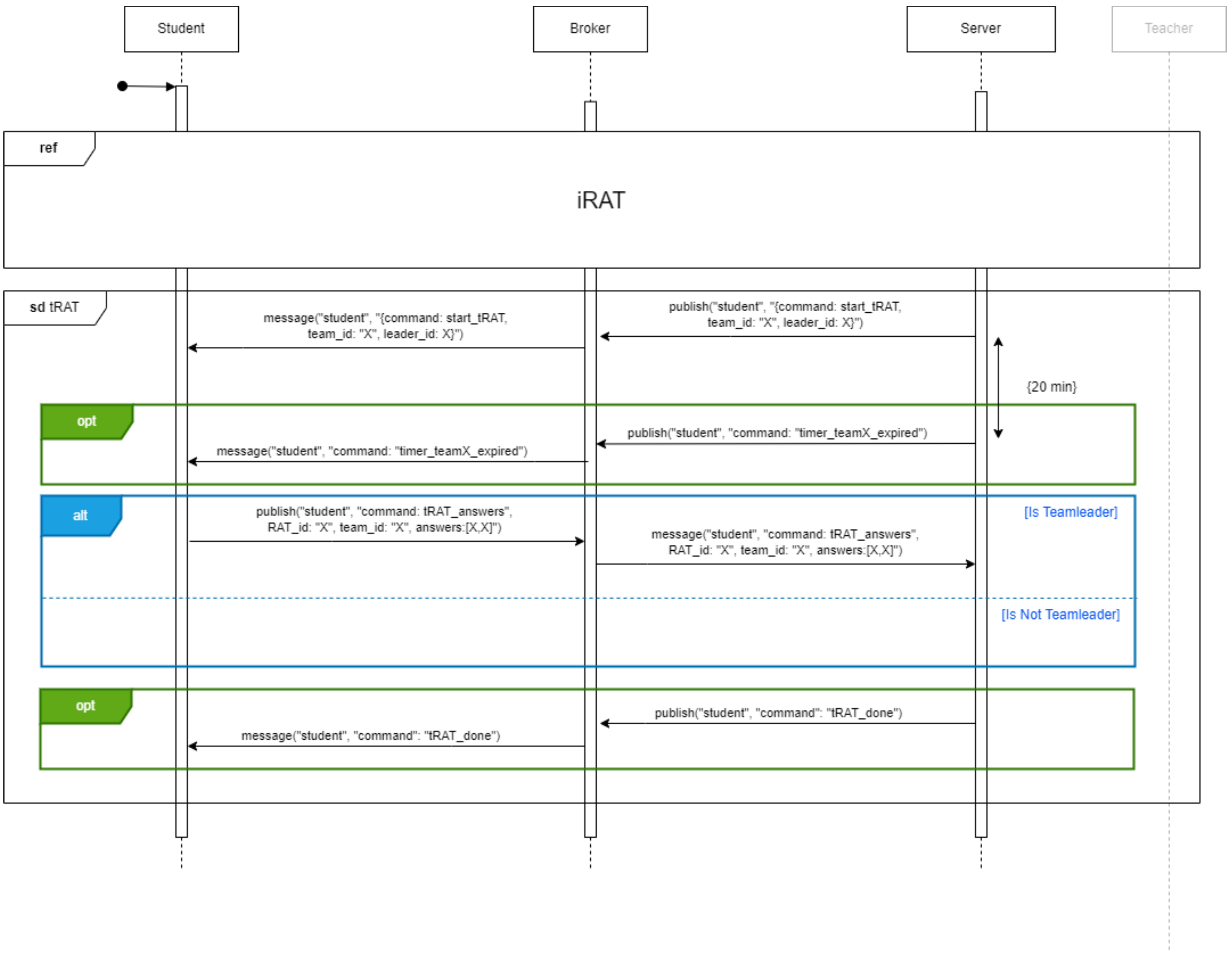

Student
Broker
Server
Teacher
ref
iRAT
sd tRAT
message("student", "{command: start_tRAT, team_id: "X", leader_id: X}")
publish("student", "{command: start_tRAT, team_id: "X", leader_id: X}")
{20 min}
opt
publish("student", "command: "timer_teamX_expired")
message("student", "command: "timer_teamX_expired")
alt
publish("student", "command: tRAT_answers", RAT_id: "X", team_id: "X", answers:[X,X]")
[Is Teamleader]
message("student", "command: tRAT_answers", RAT_id: "X", team_id: "X", answers:[X,X]")
[Is Not Teamleader]
opt
publish("student", "command": "tRAT_done")
message("student", "command": "tRAT_done")

Figure 5. Interactions for the tRAT

Figure 6 shows the teachers' interactions for creating a RAT, refreshing RATs and starting a RAT. The three opt fragments can happen in any order and a opt fragment can happen more than once in a row.

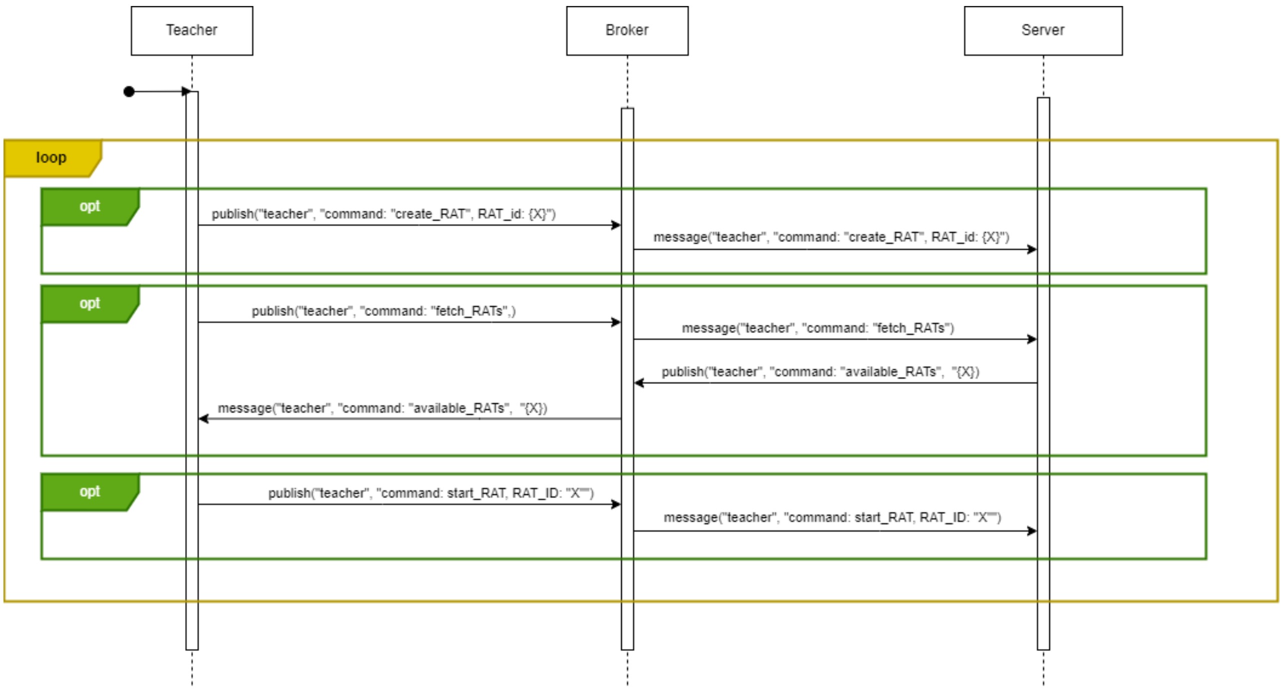

Teacher
Broker
Server
loop
opt
publish("teacher", "command: "create_RAT", RAT_id: {X}")
message("teacher", "command: "create_RAT", RAT_id: {X}")
opt
publish("teacher", "command: "fetch_RATs",)
message("teacher", "command: "fetch_RATs")
publish("teacher", "command: "available_RATs", "{X})
message("teacher", "command: "available_RATs", "{X})
opt
publish("teacher", "command: start_RAT, RAT_ID: "X"")
message("teacher", "command: start_RAT, RAT_ID: "X"")

Figure 6. Teachers' interactions

8 State Machines

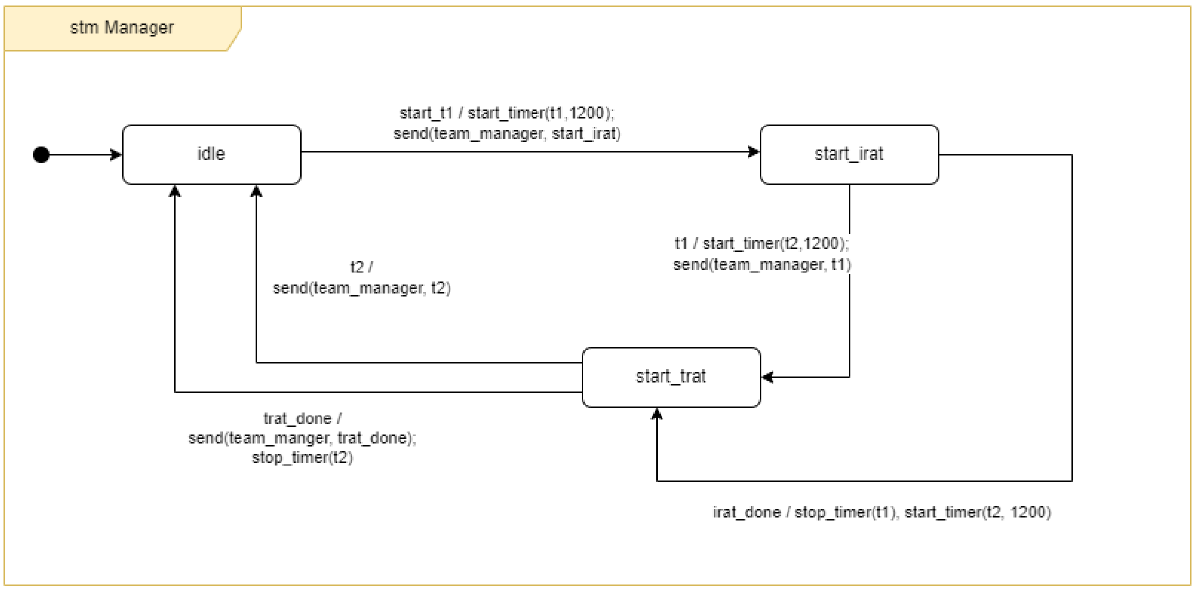

stm Manager
idle
start_irat
start_trat
start_t1 / start_timer(t1,1200);
send(team_manager, start_irat)
t1 / start_timer(t2,1200);
send(team_manager, t1)
t2 /
send(team_manager, t2)
trat_done /
send(team_manger, trat_done);
stop_timer(t2)
irat_done / stop_timer(t1), start_timer(t2, 1200)

Figure 7. Manager state machine controlling the overall flow of the RAT

The Manager state machine in Figure 7 handles the main logic of the system. It controls the entire flow from start to finish, including start and stop of RATs and timers and sending messages to team manager.

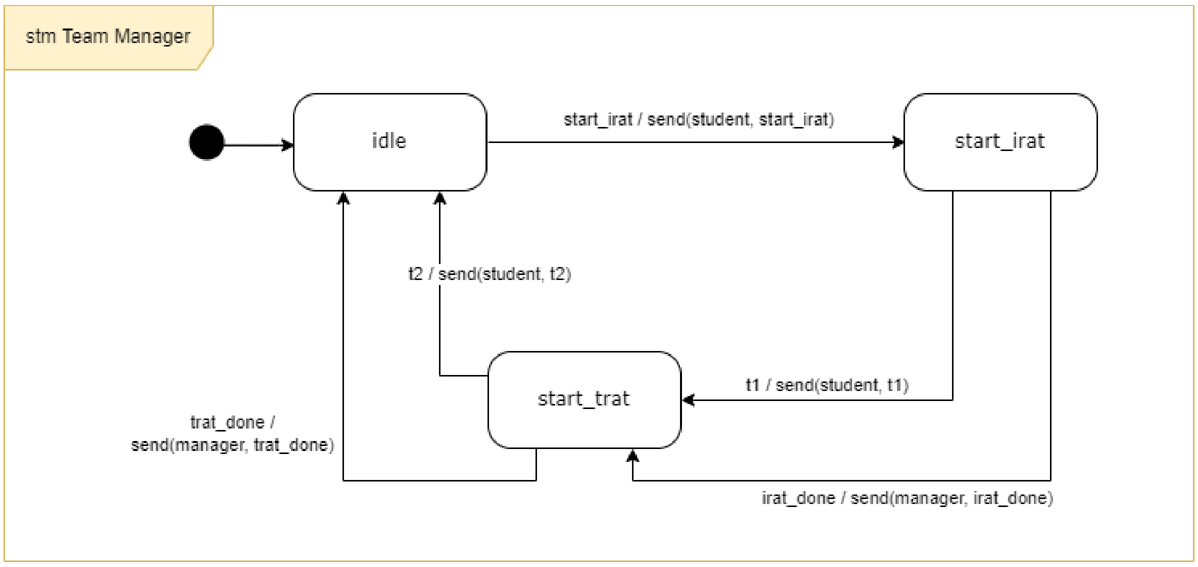


Figure 8. Team Manager state machine coordinating with the manager and the student state machines

The Team Manager state machine in Figure 8 handles the communication between manager and students.

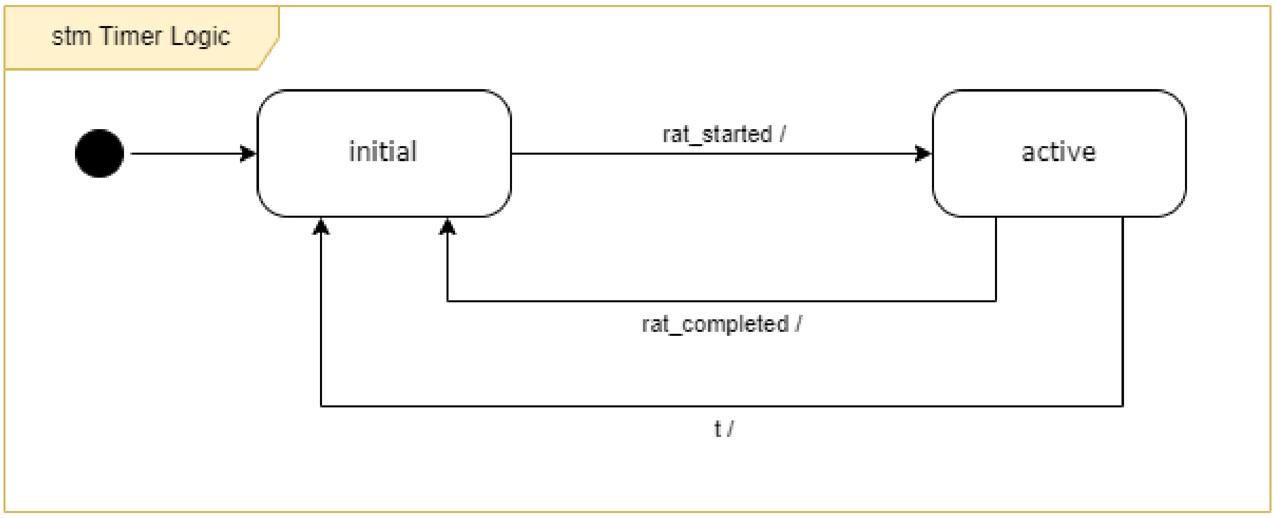


Figure 9. Timer Logic state machine handling the timers

The Timer Logic in Figure 9 is general and can be used to handle the timers for both iRAT and tRAT. The timer starts running when a RAT is started and stops either when the team is finished with the RAT, or the time limit is up.

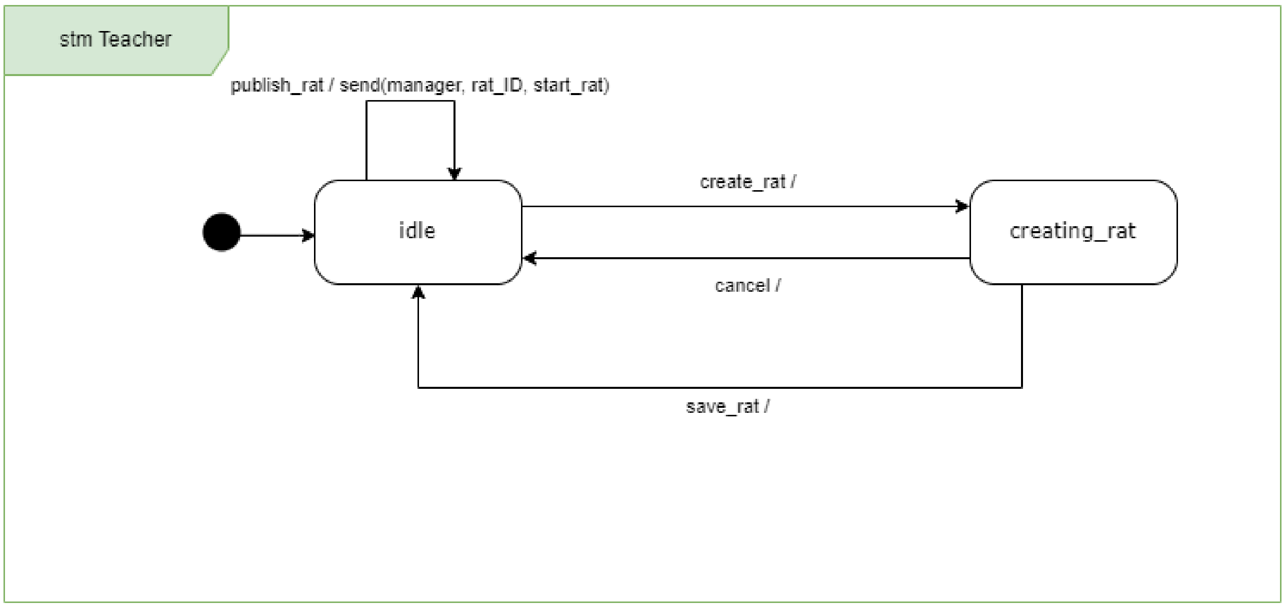


Figure 10. The teacher state machine (creates, saves and publishes a RAT)

The Teacher state machine in Figure 10 represents the process through which a teacher goes in order to create, save and publish a RAT.

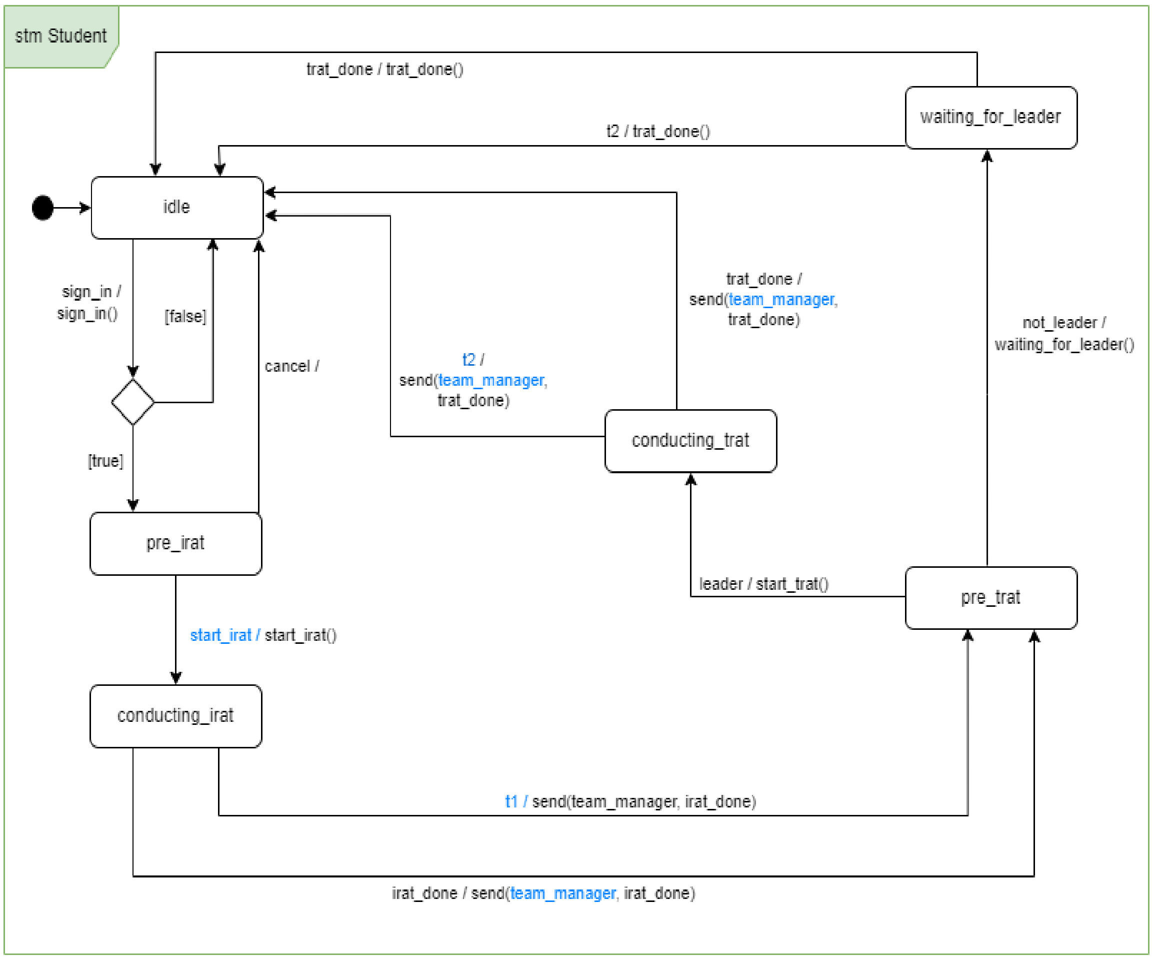

stm Student
trat_done / trat_done()
waiting_for_leader
t2 / trat_done()
idle
sign_in / sign_in()
[false]
cancel /
trat_done / send(team_manager, trat_done)
not_leader / waiting_for_leader()
t2 / send(team_manager, trat_done)
conducting_trat
[true]
pre_irat
leader / start_trat()
pre_trat
start_irat / start_irat()
conducting_irat
t1 / send(team_manager, irat_done)
irat_done / send(team_manager, irat_done)

Figure 11. Student conducting an iRAT and a tRAT

The Student state machine in Figure 11 handles the entire process of conducting an iRAT followed by a tRAT. When a student enters the "waiting_for_leader" state, it checks whether the team leader is already in the room. If the leader is not present, nothing happens. When the leader enters the waiting room, either via the timer t2 or if they finish the team rat before the timer runs out, the student enters back into the idle state. Although the implementation section does not address the decision tree, we chose to treat the necessity of having a real and secure authentication method in a real system only in the state machine.

9 Implementation

The implementation of the DRS system can be viewed as a minimum viable product (MVP). The demonstrated system works in accordance with our use cases, the current version contains just enough features to conduct a RAT session for a group of students. Below are some comments about the implementation:

- Implementation of the system features the functionality described in the use cases and structure given in the deployment diagram.
- The state machines are only partially implemented, mostly because of complexity and time constraints.
  - Implementation of the state machines for team_manager, manager, teacher and student are only partially or not implemented. Their state machines still describe the functionality for each software application and give a picture of how the different applications interact with the rest of the system and user input.
  - The decision to not implement certain state machines was based on considerations of time

constraints and complexity. In the case of the Teacher application, we determined that adding a state machine would unnecessarily complicate the code without providing significant value to the application's current functionality. However, if additional features were to be added in the future, state machines could prove useful in addressing potential complexity issues and enhancing overall functionality. Moving forward, it's important to carefully assess where state machines may be relevant and beneficial for the continued development of the application.

- Message topics and actual message data are slightly modified in the actual implementation, for easier implementation.
  - E.g., the topics are not divided as structured as described in the sequence diagram. This is because it was easier for the server to only subscribe to a general channel for all communication, and rather filter out data based on the commands sent and with the messages. We chose to base our actions in the code on data that is contained in the messages, rather than watching topics.
  - An overview of the MQTT-messages is provided as an appendix (see *Appendix A – MQTT-overview*).
- Exception handling is not implemented, e.g., when a user leaves a RAT session. The same goes for edge cases, which we chose not to prioritize and rather focus on making a system that shows the general functionality of the system.
- For further development, the code should be altered to be more dynamic and able to adapt to the system needs of the future.
- Data storing is done locally in a JSON-database. For future development, a more scalable and robust database solution should be implemented. This will ensure that the system can handle larger data volumes and handle concurrent user requests more efficiently. Therefore, it is recommended to explore other database options that are better suited to the specific needs of the application.

Appendix A - *MQTT-overview*

Message overview

1. *create_RAT*
2. *fetch_RATs*
3. *available_RATs*
4. *start_RAT*
5. *start_iRAT*
6. *student_iRAT_done*
7. *iRAT_done*
8. *start_tRAT*
9. *tRAT_answers*
10. *tRAT_done*
11. *timer_expired*

1. *create_RAT*

Command: “create_RAT”
Sender: Teacher
Receiver: Server
Channel: “ttm4115/team_5/teacher”
Function: Sends a message to the server with a RAT that is to be stored in the database.

JSON format:
```json
{
  "command": "create_RAT",
  "RAT": {
    "id": "5397c02f-3572-42c4-b861-4f6cfb891186",
    "name": "TestRAT",
    "size": 2.0,
    "subject": "TTM4115",
    "questions": {
      "1": {
        "question": "TestQ1",
        "correct": "correct",
        "a": "fail1",
        "b": "fail2",
        "c": "fail1"
      },
      "2": {
        "question": "TestQ2",
        "correct": "correct",
        "a": "fail1",
        "b": "fail2",
        "c": "fail3"
      }
    }
  }
}
```

2. *fetch_RATs*
Command: "fetch_RATs"
Sender: Teacher
Receiver: Server
Channel: "ttm4115/team_5/teacher"
Function: Requests an overview of the RATs in the database.
JSON format:
```json
{
  "command": "fetch_RATs"
}
```

3. *available_RATs*
Command: "available_RATs"
Sender: Server
Receiver: Teacher
Channel: "ttm4115/team_5/teacher"
Function: Sends an overview of the RATs in the database
JSON format:
```json
{
  "command": "available_RATs",
  "rat_info": {
    "75e60adc-e830-11ed-a05b-0242ac120003": "RAT_1",
    "8b07615d-fa1c-4e39-95f5-caddde20b113": "RAT_2",
    "3c087784-ca0c-4922-8642-a78b4fe06666": "RAT_3",
    "ca654113-c46c-4b94-98ed-b2af060d24c8": "RAT_4",
    "f0c0794d-0eaf-4937-b85b-7f46495e9040": "RAT_5"
  }
}
```

4. *start_RAT*

Command: “start_RAT”
Sender: Teacher
Receiver: Server
Channel: “ttm4115/team_5/teacher”
Function: Tells the server to start a RAT session.
JSON format:

```
{
  "command": "start_RAT",
  "RAT_ID": "75e60adc-e830-11ed-a05b-0242ac120003"
}
```

5. *start_iRAT*

Command: “start_iRAT”
Sender: Server
Receiver: Student
Channel: “ttm4115/team_5/student”
Function: Sends the chosen RAT to the students.
JSON format:

```
{
  "command": "start_iRAT",
  "RAT": {
    "name": "Demo RAT",
    "id": "d9b3a6c1-2eba-4767-be41-315bab3c5110",
    "size": 3,
    "subject": "TTM4115",
    "questions": {
      "1": {
        "question": "test1",
        "correct": "correct",
        "a": "fail1",
        "b": "fail2",
        "c": "fail3"
      },
      "2": {
        "question": "test2",
        "correct": "correct",
        "a": "fail1",
        "b": "fail2",
        "c": "fail3"
      },
      "3": {
        "question": "test3",
        "correct": "correct",
        "a": "fail1",
        "b": "fail2",
        "c": "fail3"
      }
    }
  }
}
```

6. *student_iRAT_done*

Command: “student_iRAT_done”
Sender: Student
Receiver: Server
Channel: “ttm4115/team_5/student”
Function: Tells the server that a student in team x, is done with their iRAT, and the questions they got correct.
JSON format:

```
{
   "command":"student_iRAT_done",
   "student_ID": "1"
   "team_ID": "1"
   "RAT_ID": "75e60adc-e830-11ed-a05b-0242ac120003"
   "answers": "[correct, a, correct]"
}
```

7. *iRAT_done*

Command: “iRAT_done”
Sender: Server
Receiver: Server*
Channel: “ttm4115/team_5/server”
Function: Tells the server that an entire team of students are done with their iRAT
JSON format:

```
{
   "command":"iRAT_done",
   "team_ID": "1"
}
```

8. *start_tRAT*

Command: “start_tRAT”
Sender: Server
Receiver: Student
Channel: “ttm4115/team_5/student”
Function: Starts the tRAT and chooses a team leader after the entire team is done with the iRAT
JSON format:

```
{
  "command": "start_tRAT",
  "team_ID": "1",
  "leader_ID": "1"
}
```

9. *tRAT_answers*

Command: “start_tRAT”
Sender: Student
Receiver: Server

Channel: “ttm4115/team_5/student”
Function: Sends the tRAT answers to the server,
JSON format:
```
{
  "command": "tRAT_answers",
  "RAT_id": "1",
  "team_id": "1",
  "answers": "[correct, correct, correct]"
}
```

10. *tRAT_done*
Command: “tRAT_done”
Sender: Server
Receiver: Server*
Channel: “ttm4115/team_5/teacher”
Function: Tells the server that a team is done with the tRAT.
JSON format:
```
{
  "command": "tRAT_done"
}
```

11. *timer_expired*
Command: “x_expired”**
Sender: Server
Receiver: Student
Channel: “ttm4115/team_5/student”
Function: Tells a student during the iRAT or the student leader during a tRAT that the RAT timer has expired, forcing the student to submit their answers
JSON format:
```
{
  "command": "x_expired"
}
```

Footnotes
* The team manager sends an MQTT message to the manager. We did this to allow for there to be different servers operating at the same time, but we implemented everything on the same server
** The “x” is the name of the timer. If we start the timer “t1” at the iRAT, the command will be: “t1_expired”. If we want to address the timer for team 1, the command will be “timer_team1_expired”.

Appendix B - *GitHub Repository*
https://github.com/mkarder/ttm4115_project

Appendix C – *Video Demo*
https://drive.google.com/file/d/1X2UtMWoZ0T2XcrY-_7LKMN34grmVToaE/view?usp=sharing