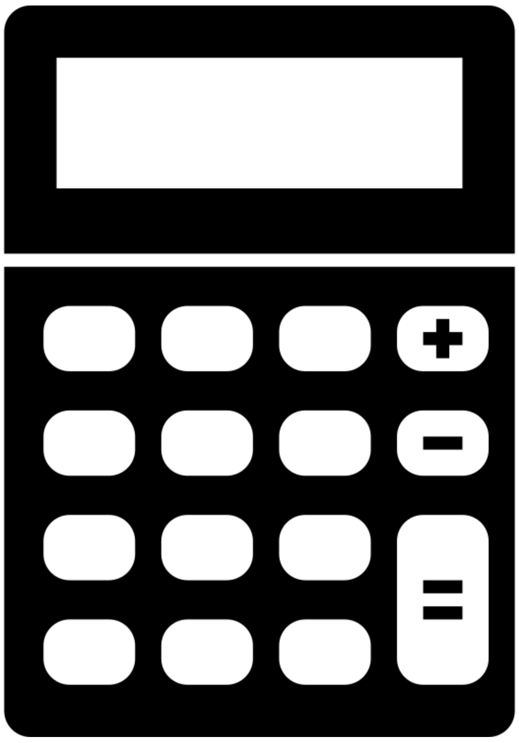
Architecture and Node documentation for Tilmann’s cijfer generator

Group 4: Rik van Velzen, Melissa van Leeuwen, Burhan Topaloglu



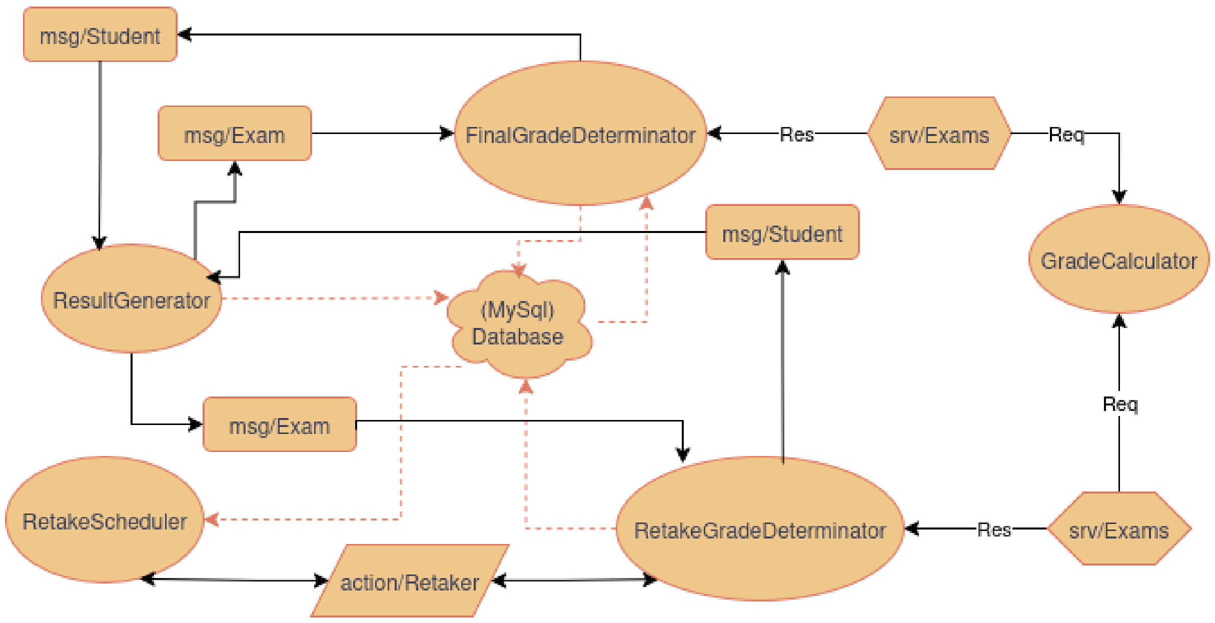
This Document details the architecture and node documentation of the ROS2-Jazzy package “Tilmann’s cijfer generator” which is assignment number 1 for ES1.

Firstly it will go over how the nodes interact and then it will detail how the nodes themselves work.

CONTENTS

Architecture

Flowchart



Exam Generation Process

- FinalGradeDeterminator requests students from the database.
- Students are published to ResultGenerator via add_students.
- ResultGenerator periodically publishes random exam results (Exam messages).

Final Grade Calculation

- FinalGradeDeterminator subscribes to these exam results.
- Once all grades for a student/course are received:
    - Calls GradeCalculator service.
    - Stores final grade in database.
    - Publishes remove_students to ResultGenerator.

Retake Process

- RetakeScheduler periodically checks the database for failed students.
- Sends retake requests to RetakeGradeDeterminator via Action Client.
- RetakeGradeDeterminator requests ResultGenerator to generate new exam results.
- Collects new grades, sends them to GradeCalculator, stores results.
- Publishes remove_students once retake is finished.

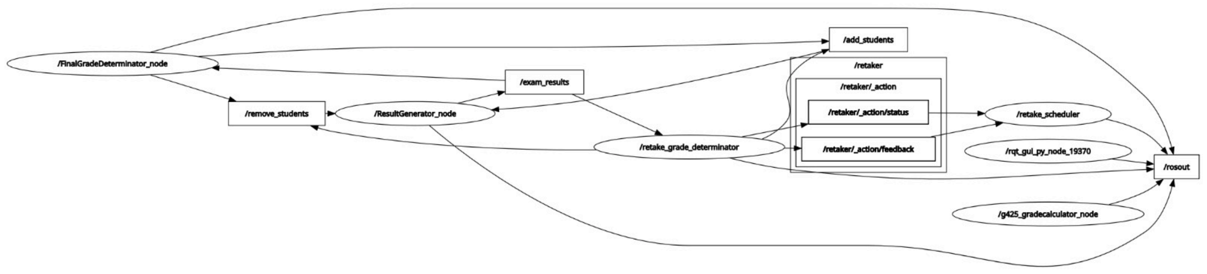
Grade Calculator

- Acts as a service endpoint used by both FinalGradeDeterminator and RetakeGradeDeterminator to compute grades.

Notes on each node:

- ResultGenerator is the only node producing exam grades.
- FinalGradeDeterminator handles first-time grade calculation.
- RetakeScheduler + RetakeGradeDeterminator handle failing students sequentially.
- GradeCalculator is a service used by both determinators for final grade computation.
- GGDatabase is central MySQL storage without using ROS messages.
- Add/Remove messages control which students ResultGenerator should generate grades for.

rqt_graph



Messages

Retaker.action

```
# Goal
g425_assign1_interfaces_pkg/Student student
---
# Result
bool success
string message
---
# Feedback
string status
```

Exams.srv

```
# Request
g425_assign1_interfaces_pkg/Student student
float32[] exam_grades
---
# Response
g425_assign1_interfaces_pkg/Student student
float32 final_grade
```

Student.msg

```
#Student name in full
string student_fullname
#Course name in full
string course_name
#Unique IDs to prevent duplicates and possibly make lookup simpler
int64 student_id
int32 course_id
int32 number_of_grades
#timestamp
builtin_interfaces/Time stamp
```

Exam.msg

```
builtin_interfaces/Time stamp
g425_assign1_interfaces_pkg/Student student
float32 exam_grade
```

Nodes

Node helper: Database

Type: mariadb database function class

The database puts these functions out:

```
std::tuple<bool, MYSQL*> setupConnection();
std::string getStudentName(int s_id);
std::string getCourseName(int c_id);
int getStudentId(const std::string& s_name);
int getCourseId(const std::string& c_name);
int getGradeAmountFromCourse(int c_id);
bool addGrade(const DBT_Grade& st_grade);
bool addFinalGrade(const DBT_FinalGrade& st_finalGrade);
std::vector<std::tuple<int, int>> getAllStudentCoursesRel();
std::vector<std::tuple<int, int>> getMissingFinalGrades();
std::vector<DBT_FinalGrade> getAllFinalGrades();
```

Method “setupConnection” is automatically called in the constructor and is only public for when the connection needs to be restarted.
These functions were tailored to the needs of the nodes, they can use these by creating an object and calling the method.

```
#include "g425_assign1_pkg/GGDatabase.hpp"
GGDatabase db;
std::vector<DBT_FinalGrade> finalGrades = db.getAllFinalGrades();
```

Result Generator

Node type:

- Publisher (publishes "exam_results" messages)
- Subscriber (receives "add_students" and "remove_students" commands)
- Timer (periodically triggers result generation)
- Database interface (stores generated exam results)

Main ROS objects used:

rclcpp::Publisher<Exam>	- publishes random exam results
rclcpp::Subscription<Student>	- listens for add/remove student requests
rclcpp::TimerBase	- executes result generation loop
GGDatabase	- records generated grades in the database

Role:

- Generates random exam results for student/course combinations that lack a final grade.
- Publishes Exam messages.
- Subscribes to messages to add/remove students for random result generation.

Node description:
Continuously generates random exam results for all active student/course combinations.

The node periodically (every EXAM_PUBLISH_INTERVAL seconds) publishes an "Exam" message with a random grade between MIN_MARK and MAX_MARK for a random student from the current list.
Each generated result is also stored in the database.

```
void publish_random_result() {
   if (students_.empty()) return;

   auto student = students_[rand() % students_.size()];
   float grade = (rand() % (max_mark_ - min_mark_ + 1)) + min_mark_;

   Exam exam;
   exam.stamp = now();
   exam.student = student;
   exam.exam_grade = grade;
   exam_pub_->publish(exam);

   db_.addGrade({student.student_id, student.course_id, grade});
}
```

It subscribes to:

- "add_students" to add new students to the generation list.
- "remove_students" to remove students once their final grade has been determined.

```
add_student_sub_ = create_subscription<Student>(
    "add_students", 10,
    [this](const Student &s) { students_.push_back(s); });

remove_student_sub_ = create_subscription<Student>(
    "remove_students", 10,
    [this](const Student &s) {
        students_.erase(std::remove_if(students_.begin(),
students_.end(),[&](auto &st){return st.student_id == s.student_id &&
st.course_id == s.course_id;}),students_.end());
});
```

Retake Scheduler

Node Type:

- Action Client (Scheduler)

Main ROS objects used:

rclcpp::TimerBase	for periodic execution
rclcpp_action::Client<Retaker>	to send goals
GGDatabase	for accessing grades and student info
std::queue and std::set	for managing scheduled retakes

Role:

- Periodically checks the database for students who failed (grades 10–54).
- Uses Action Client to request retakes via RetakeGradeDeterminator.
- Manages a queue of students needing retakes.

Node description:
Periodically checks a student database to find all students who have failed a course (final grade between 10 and 54). For each of these students, it schedules a new retake by sending a goal to the "retaker" Action Server.

```
// Action client for retake requests
retake_actionclient_ = rclcpp_action::create_client<Retaker>(this,
"retaker");
// Timer to periodically check for failed students
timer_ = create_wall_timer(std::chrono::seconds(schedule_time_),
   [this]() {
       retake_actionclient_->wait_for_action_server();
       // Fetch failed students from database
       for (auto &fg : db.getAllFinalGrades()) {
           if (fg.final_grade >= 10 && fg.final_grade <= 54) {
               Student s;
               s.student_id = fg.student_id;
               s.course_id = fg.course_id;
               s.student_fullname = db.getStudentName(fg.student_id);
               s.course_name = db.getCourseName(fg.course_id);
               s.number_of_grades =
db.getGradeAmountFromCourse(fg.course_id);
               // Send a retake goal
               auto goal_msg = Retaker::Goal();
               goal_msg.student = s;
               retake_actionclient_->async_send_goal(goal_msg);
           }
}
});
```

Retake Grade Determinator

Node type:

- Action Server (handles Retaker goals)
- Service Client (calls GradeCalculator)
- Subscriber (listens to exam_results)
- Publisher (add_students, remove_students)
- Timer-like loop (via rclcpp::Rate)

Main ROS objects used:

rclcpp_action::Server<Retaker>	for goal handling
rclcpp::Client<GradeCalculator>	for grade calculation
rclcpp::Publisher<Student>	for triggering/removing result generation
rclcpp::Subscription<ExamResults>	for receiving exam grades
GGDatabase	for storing new final results

Role:

- Uses an Action Server to handle retake requests.
- Requests ResultGenerator to start generating new exam results.
- Subscribes to Exam messages during retake.
- Sends grades to GradeCalculator service.
- Saves new final grades in the database.
- Publishes messages to remove students after retake completion.

Node description:
Manages the full process of re-determining a student's final grade for a course after a failed attempt.

Upon receiving a retake goal from the "retake_scheduler" Action Client, the node requests new random exam results by publishing to the "add_students" topic.

```
// Receive a retake goal
retake_actionserver_ = rclcpp_action::create_server<Retaker>(
   this, "retaker",
   [](auto, auto){ return
rclcpp_action::GoalResponse::ACCEPT_AND_EXECUTE; },
   [](auto){ return rclcpp_action::CancelResponse::ACCEPT; },
   [this](auto goal_handle) {
       auto student = goal_handle->get_goal()->student;
       collected_grades_.clear();
       collecting_ = true;

       // Ask ResultGenerator for new random results
      add_student_pub_->publish(student);
```

It then subscribes to the "exam_results" topic to collect new grades. Once enough results are received, it calls the GradeCalculator service to calculate a new final grade, adds this new result to the database (without overwriting the previous one), and finally publishes to "remove_students" to stop random generation.

```
// Listen for new exam results
exam_sub_ = create_subscription<ExamResults>("exam_results", 10, [this,
student](const ExamResults &msg) {
      if (!collecting_) return;
      collected_grades_.push_back(msg.exam_grade);
// Once enough results received → calculate new final grade
if (collected_grades_.size() >= student.number_of_grades) {
auto req = std::make_shared<GradeCalculator::Request>();
req->student = student;
req->exam_grades = collected_grades_;
grade_client_->async_send_request(req, [this](auto fut) {
 auto res = fut.get();
 db_.addFinalGrade({res->student.student_id,res->student.course_id,
res->final_grade});
 remove_student_pub_->publish(res->student);
 collecting_ = false;
 });
}
});
});

// ROS interfaces
add_student_pub_      = create_publisher<Student>("add_students", 10);
remove_student_pub_   = create_publisher<Student>("remove_students",
10);
grade_client_ = create_client<GradeCalculator>("GradeCalculator");
```

Final Grade Determinator

Node type:

- Subscriber (listens to "exam_results")
- Service Client (calls "GradeCalculator")
- Publisher (publishes to "add_students" and "remove_students")
- Timer (periodically checks the database for new students)

Main ROS objects used:

● rclcpp::Subscription<Exam>	for receiving exam results
● rclcpp::Client<Exams>	for requesting grade calculations
● rclcpp::Publisher<Student>	for starting/stopping result generation
● rclcpp::TimerBase	for periodic database checks
● GGDatabase	for retrieving and storing student and grade information

Role:

- Publishes messages to add students to ResultGenerator.
- Subscribes to Exam results.
- Collects exam grades until enough grades are received for a student/course.
- Calls GradeCalculator service to compute the final grade.
- Saves the final grade in the database.
- Publishes messages to remove students from ResultGenerator.

Node description:
Automatically determines the final grades for students and courses that do not yet have a result stored in the database.

It subscribes to the "exam_results" topic to receive exam grades generated by the ResultGenerator node.

```
exam_sub_ = create_subscription<Exam>(
   "exam_results", 10,
   [this](const Exam &msg) {
       auto &grades = student_grades_[{msg.student.student_id,
msg.student.course_id}];
       grades.push_back(msg.exam_grade);
```

Once enough grades have been received (based on the required
number retrieved from the database), the node calls the "GradeCalculator" service
to compute the final grade.

```
 if (grades.size() >= required_grades_[{msg.student.student_id,
msg.student.course_id}]) {
           auto req = std::make_shared<Exams::Request>();
           req->student = msg.student;
           req->exam_grades = grades;
//exam_client_ = create_client<Exams>("GradeCalculator");
           exam_client_->async_send_request(req, [this](auto future) {
                auto res = future.get();
```

The computed grade is then added to the database and a "remove_students" message is published to stop further result generation for that student/course combination.

```
                db_.addFinalGrade({res->student.student_id,
                                   res->student.course_id,
                                   res->final_grade});
                remove_student_pub_->publish(res->student);
//remove_student_pub_ = create_publisher<Student>("remove_students",
10);
```

GradeCalculator

Node type:

- Service Server

Main ROS objects used:

rclcpp::Service<Exams>	for handling service requests
Exams.srv	for exchanging student data and computed grade
rclcpp::Node base class	for ROS node setup

Role:

- Provides a service to calculate a final grade from exam results.
- Adds a bonus for specific students.
- Returns a grade between 10 and 100.

Node description:
The calculator waits for a service request “Exams.srv”.
It then calculates an average, adds bonus points, and sends a response.

```
calculatedFinalGrade = 0.0f;
calculatedFinalGrade +=
 request->exam_grades.empty() ? 0.0f : sum/request->exam_grades.size();
calculatedFinalGrade += BONUS_POINTS;
calculatedFinalGrade = clamp(calculatedFinalGrade, 10, 100);
```