Teaching Room Occupancy Management System
Specifications for Sealand Republic University (SRU)

Date : October 29, 2024

Version : v1.0

Team Name : QEV

Team Members : CLERC Edgar, LATH Victor, ZHENG Qi

Subject A

Preface

The exported file is meant to be used by staffs, teachers and students, to make a schedule. The document is structured as follows:

- **Introduction** : Outlines the context and aims of the software project.
- **Client Level Specification** : Lists the functional specifications for the software, along with potential non-functional specifications. Each specification is described in natural language with a unique reference ID.
- **Detailed Specification** :
    - **Functional Specification Details** : Provides a structured breakdown for each specification, using formal fields such as Identifier, Title, Objective(s), Pre-condition(s), Post-condition(s), Input, Process, Output, and Error Handling.
    - **Data Format**: Specifies the syntax of data formats used by the software, adhering to the Augmented Backus-Naur Form (ABNF).
    - **Data Semantics**: Defines the data structure for the primary object type used in the software through abstract data type formalism, including temporal order and equivalence operations. This section also includes an abstract data type specification for a collection of such objects, with relevant operations and axioms.
- **Conclusion** : Emphasizes the relevance of the proposed specifications in achieving the project's goals and provides recommendations for the development team.

Introduction

Context : The Sealand Republic University (SRU) manages a large student and faculty population. The university wants to simplify the room occupancy tracking system to make scheduling and space management more efficient.

Objectives : The objective is to develop a command-line application for SRU's classroom management. This tool will enable users to :

- Query classroom schedules
- Check room availability
- Export schedule as iCalendar files
- Verify schedule quality : preventing overlapping booking in the same room, improving schedule reliability
- Visualize room utilization : provide a report on room occupancy rates and a capacity-based classification of rooms, for helping administrators to identify underutilized or over-utilized spaces for strategic planning.

Client level specification

Functional Specifications

- **Search for Rooms by Course**:
  Allow users to search for rooms associated with a given course by entering its identifier (e.g., ME01).

- **Maximum Room Capacity**:
  Provide users with the maximum capacity (in terms of seating) for a specified room upon request.

- **Room Availability**:
  Display time slots during which a specific room is free to help students plan group work sessions.

- **List of Available Rooms for a Given Time Slot**:
  Enable users to see which rooms will be available at a specific time on a given day.

- **iCalendar (RFC 5545) File Generation**:
  Generate an iCalendar file between two given dates for courses (lectures, practical sessions, tutorials) in which a user participates, allowing them to import it into their calendar.

- **Data Quality Verification**:
  Ensure that a time slot in a room is not double-booked or used by multiple courses simultaneously (no overlapping or double-booking of courses).

Non-Functional Specifications

- **Performance**:
  The software should respond to user requests within 2 seconds for simple operations (e.g., room capacity lookup) and within 5 seconds for complex operations (e.g., iCalendar file generation or room occupancy visualization).

- **Command Line Interface**:
  Provide a simple and intuitive command-line user interface with clear documentation for each function.

- **Robustness and Security**:
  The program should handle user input errors reliably without causing crashes. Ensure the security of timetable information by restricting access to non-relevant data.

- **Code Maintainability and Scalability**:
  The code should be well-structured, documented, and modular to facilitate system maintenance and future updates.

Detailed specification

- Functional Specifications

Specification 1

- **Identifier**: F1

- **Title**: Search for Rooms by Course

- **Objective(s)**: Allow users to retrieve all rooms and their scheduled time slots for a specified course, facilitating access to room schedules.

- **Pre-condition(s)**: The course identifier is valid and exists in the CRU timetable data.

- **Post-condition(s)**: The system displays a list of rooms associated with the course and their scheduled time slots.

- **Input**: Course identifier (e.g., ME01).

- **Process**:
    - The system parses the CRU data, filtering entries to match the course identifier.
    - For each match, it retrieves the associated room(s) and corresponding time slots.

- **Output**: A list showing the rooms, course type (CM, TD, TP), capacity, and schedule for the specified course.

- **Error handling**:
    - If the course identifier is invalid or missing from the data, return an error message stating, “Course identifier not found.”
    - If no rooms are associated with the course, inform the user: “No rooms scheduled for this course.”

Specification 2

- **Identifier**: F2

- **Title**: Retrieve Maximum Capacity of a Room

- **Objective(s)**: Provide users with the maximum seating capacity for a specified room to help them assess its suitability.

- **Pre-condition(s)**: The room identifier is valid and exists in the CRU timetable data.

- **Post-condition(s)**: The system displays the maximum capacity of the requested room.

- **Input**: Room identifier (e.g., P202).

- **Process**:
    - The system scans the CRU data for the specified room and extracts all associated capacities.
    - It then calculates the maximum capacity recorded for the room.

- **Output**: The maximum seating capacity for the specified room.

- **Error handling**:
    - If the room identifier is invalid or missing, return an error message stating, “Room identifier not found.”
    - If the room has no associated capacity data, inform the user: “No capacity data available for this room.”

Specification 3

- **Identifier**: F3

- **Title**: Check Room Availability by Time Slot

- **Objective(s)**: Allow users to view the availability of a specific room during specified time frames to plan group work or other activities.

- **Pre-condition(s)**: The room identifier is valid, and the time frame is correctly formatted.

- **Post-condition(s)**: The system displays all time slots during which the room is available in the specified time frame.

- **Input**: Room identifier and optional time frame (e.g., Thursday, 10:00-12:00).

- **Process**:
    - The system filters the CRU data for the specified room to identify all scheduled time slots.
    - It cross-checks the provided time frame to exclude any overlapping time slots.

- **Output**: A list of available time slots for the room within the specified time frame, organized by weekday and hours.

- **Error handling**:
    - If the room identifier is invalid, return an error message: “Room identifier not found.”
    - If no free time slots are available within the specified frame, inform the user: “No free time slots found for this room in the specified timeframe.”

Specification 4

- **Identifier**: F4

- **Title**: List Available Rooms for a Given Time Slot

- **Objective(s)**: Enable users to identify all rooms available for a specific time slot to facilitate flexible room bookings.

- **Pre-condition(s)**: The date and time slot are correctly formatted and fall within operational hours.

- **Post-condition(s)**: The system displays a list of rooms available during the specified time slot.

- **Input**: Specific day and time slot (e.g., Thursday, 13:00-16:00).

- **Process**:
    - The system parses the CRU data to identify rooms with scheduled courses during the specified time.
    - It then filters out those rooms, listing only the rooms with no occupancy during the specified time slot.

- **Output**: A list of available rooms for the specified time slot, including room identifiers, types, and capacities.

- **Error handling**:
    - If no rooms are found to be available, display the message: “No rooms available for the specified time slot.”
    - If the input format is incorrect or out of operational hours, return an error stating, “Invalid time slot format or outside operational hours.”

Specification 5

- **Identifier**: F5

- **Title**: Generate iCalendar File (RFC 5545)

- **Objective(s)**: Allow users to export their course schedules in iCalendar format for easy integration with personal calendar applications.

- **Pre-condition(s)**: User identifier and date range are valid, and the user is associated with scheduled courses.

- **Post-condition(s)**: An iCalendar (.ics) file is generated containing events representing the user's course schedule.

- **Input**: User identifier, course types to include (CM, TD, TP), and date range.

- **Process**:
    - The system retrieves all scheduled courses for the specified user, date range, and course types.
    - It generates an iCalendar file, creating one event per course with details on location, date, time, and description, formatted to RFC 5545 standards.

- **Output**: A downloadable iCalendar (.ics) file with the user's course schedule.

- **Error handling**:
    - If no courses are found for the user, display the message: "No courses found for the specified date range."
    - If the input format is incorrect, return an error: "Invalid input format."

Specification 6

- **Identifier**: F6

- **Title**: Verify Timetable Data Quality

- **Objective(s)**: Ensure data integrity by detecting scheduling conflicts, such as overlapping courses in the same room.

- **Pre-condition(s)**: The CRU timetable data is loaded and parsed.

- **Post-condition(s)**: A report is generated, highlighting any scheduling conflicts within the data.

- **Input**: None (automated or on-demand).

- **Process**:
    - The system scans the CRU data for each room and checks for overlapping time slots.
    - It flags conflicts where two or more courses are scheduled for the same room and time slot.

- **Output**: A report of detected scheduling conflicts, including course and room details with overlapping times.

- **Error handling**:
    - If no conflicts are found, display the message: “No scheduling conflicts detected.”
    - If data parsing fails, return an error stating: “Failed to parse CRU data.”

Non-Functional Specifications

Specification N1

- **Identifier**: N1

- **Title**: Performance Optimization

- **Objective(s)**: Ensure responsive and efficient system performance for user queries and tasks.

- **Pre-condition(s)**: System resources are within minimum requirements, and the CRU data file is accessible.

- **Post-condition(s)**: Users receive prompt responses, with simple queries completing in under 2 seconds and complex queries in under 5 seconds.

- **Input**: User requests for various functionalities (e.g., search, calendar generation).

- **Process**:
    - The system uses efficient data parsing, indexing, and caching where possible to minimize response time.
    - Implements lazy loading or preloading data to reduce processing time on complex queries.

- **Output**: Fast and responsive user experience.

- **Error handling**:
    - If system resources are insufficient, display a message: “System resources are low; performance may be affected.”
    - If a process exceeds the time limit, return an error: “Request timed out. Please try again.”

Specification N2

- **Identifier**: N2

- **Title**: Command-Line Interface Usability

- **Objective(s)**: Provide a user-friendly command-line interface for easy access to all functionalities.

- **Pre-condition(s)**: The system is installed, and users are familiar with the command-line interface.

- **Post-condition(s)**: Users can interact with the system efficiently using intuitive commands and parameters.

- **Input**: Commands and arguments for desired functionalities (e.g., `--get_capacity`, `--find_room`).

- **Process**:
    - The CLI parses each command and validates it against a list of known commands and options.
    - The system provides quick access to documentation or help using `--help`.
-
- **Output**: The requested information or result, displayed in a clear, readable format on the command line.

- **Error handling**:
    - If an unknown command is entered, display the message: “Unknown command. Use --help for available commands.”
    - If the input format is incorrect, display: “Invalid format. Refer to the help documentation.”

Specification N3

- **Identifier**: N3

- **Title**: Robustness and Data Security

- **Objective(s)**: Ensure stable operations, handle errors gracefully, and protect sensitive data.

- **Pre-condition(s)**: System is installed, and necessary files are available without corruption.

- **Post-condition(s)**: System handles errors without crashing and restricts unauthorized data access.

- **Input**: Various inputs, such as course and room identifiers, date ranges.

- **Process**:
    - Error handling is implemented to catch invalid inputs, file corruption, or permission issues.
    - Validation checks ensure only valid data is processed, while access controls limit data exposure.

- **Output**: Reliable system behavior with meaningful error messages.

- **Error handling**:
    - If a required file is missing, display an error: “Data file not found.”
    - If an unauthorized action is attempted, return “Permission denied.”

Specification N4

- **Identifier**: N4

- **Title**: Portability and Compatibility

- **Objective(s)**: Ensure the software can run across multiple operating systems (Windows, macOS, Linux) and supports compatibility CRU data format.

- **Pre-condition(s)**: The system is developed using cross-platform-compatible libraries and programming languages, and CRU data follows a supported structure.

- **Post-condition(s)**: The system runs without issues on all specified operating systems and correctly handles CRU data format updates.

- **Input**: Various command-line instructions and CRU data in compatible format versions.

- **Process**:
    - The system is built using cross-platform dependencies to avoid OS-specific limitations.
    - It includes input parsers that detect CRU data format versions and apply necessary adjustments for backward compatibility.

- **Output**: A functioning tool with consistent outputs regardless of the operating system or CRU format version used.

- **Error handling**:
    - If an unsupported operating system is detected, display an error: “Unsupported operating system.”
    - If an incompatible CRU data version is detected, return an error: “Unsupported CRU data version. Please use a compatible format.”

Data formats (ABNF)

file = 1*courseBlock

courseBlock = courseId CRLF 1*teachingTimeslot

courseId = '+' 2ALPHA 2DIGIT

teachingTimeslot = '1,' type ',' capacity ',' dayTime ',' subgroupIndex ',' roomName '//' CRLF

type = 1ALPHA 1DIGIT

capacity = 'P=' 1*3DIGIT

dayTime = 'H=' day WSP startTime '-' endTime

day = 'L' / 'MA' / 'ME' / 'J' / 'V' / 'S' / 'D'

startTime = 1*2DIGIT ':' 2DIGIT

endTime = 1*2DIGIT ':' 2DIGIT

subgroupIndex = 1ALPHA 1DIGIT

roomName = 'S=' 4(ALPHA / DIGIT)

Key Elements Explanation:

- **file**: Represents the entire structure, containing multiple **`courseBlock`**s.
- **courseBlock**: Defines a course block with a **`courseId`** and multiple **`teachingTimeslot`**s. Each course starts with the **+** character.
- **courseId**: The unique identifier for each course, consisting of **+** followed by two letters and two digits.
- **teachingTimeslot**: Defines each session's details. A teaching timeslot includes:
    - **type**: Specifies the type of the course.
    - **capacity**: The room capacity for this course, formatted as **P=** followed by digits.
    - **dayTime**: The day and time when this course occurs.
    - **subgroupIndex**: Specifies which subgroup this course is designated for.
    - **roomName**: The room in which this course is held, prefixed by **S=**.
- **day**: Indicates the day of the week using single letters or abbreviations.
- **startTime** and **endTime**: Represent the timeslot's start and end time in **`HH:MM`** format.

Data semantics

Abstract type TeachingTimeslot :

Title : TeachingTimeslot
Sort : TeachingTimeslot
References : String, Integer, Boolean

Description : The TeachingTimeslot defines a course session object, with each session containing specific information such as :

- courseType : String, representing the type of course
- capacity : Integer, indicating the maximum number of participants
- date : Date, indicating the day of the session
- startTime and endTime : Time, specifying the session's time range
- subgroupIndex : String, denoting the subgroup for the session
- roomName : String, representing the room where the session occurs

This type supports **Equivalence** and **Temporal Order** operations, which are used to compare the properties and order of two sessions.

Préconditions : We have defined the derived types Date and Time, which are subsets of the String type and meet the following range constraints :
Date = { d ∈ { "L", "MA", "ME", "J", "V", "S", "D" } }
Time = { t | t is a string representing time in the "HH:MM" format, with "00:00" ≤ t ≤ "23:59" }

---------- SIGNATURES ----------

CreateTeachingTimeslot : String x Integer x Date x Time x Time x String x String -> TeachingTimeslot
Equals : TeachingTimeslot x TeachingTimeslot → Boolean
Before : TeachingTimeslot x TeachingTimeslot → Boolean
After : TeachingTimeslot x TeachingTimeslot → Boolean
GetCourseType : TeachingTimeslot → String
GetCapacity : TeachingTimeslot → Integer
GetDate : TeachingTimeslot → Date
GetStartTime : TeachingTimeslot → Time
GetEndTime : TeachingTimeslot → Time
GetSubgroupIndex : TeachingTimeslot → String
GetRoomName : TeachingTimeslot → String

----------AXIOMES ----------

Equals(t1, t2) =
GetCourseType(t1) = GetCourseType(t2) ∧
GetCapacity(t1) = GetCapacity(t2) ∧
GetDate(t1) = GetDate(t2) ∧
GetStartTime(t1) = GetStartTime(t2) ∧
GetEndTime(t1) = GetEndTime(t2) ∧

GetSubgroupIndex(t1) = GetSubgroupIndex(t2) ∧
GetRoomName(t1) = GetRoomName(t2)

Before(t1, t2) =
(GetDate(t1) < GetDate(t2)) ∨
(GetDate(t1) = GetDate(t2) ∧ GetEndTime(t1) <=GetStartTime(t2))

After(t1, t2) =
(GetDate(t1) > GetDate(t2)) ∨
(GetDate(t1) = GetDate(t2) ∧ GetStartTime(t1) >= GetEndTime(t2))

GetCourseType(CreateTeachingTimeslot(courseType, capacity, date, startTime, endTime, subgroupIndex, roomName)) = courseType

GetCapacity(CreateTeachingTimeslot(courseType, capacity, date, startTime, endTime, subgroupIndex, roomName)) = capacity

GetDate(CreateTeachingTimeslot(courseType, capacity, date, startTime, endTime, subgroupIndex, roomName)) = date

GetStartTime(CreateTeachingTimeslot(courseType, capacity, date, startTime, endTime, subgroupIndex, roomName)) = startTime

GetEndTime(CreateTeachingTimeslot(courseType, capacity, date, startTime, endTime, subgroupIndex, roomName)) = endTime

GetSubgroupIndex(CreateTeachingTimeslot(courseType, capacity, date, startTime, endTime, subgroupIndex, roomName)) = subgroupIndex

GetRoomName(CreateTeachingTimeslot(courseType, capacity, date, startTime, endTime, subgroupIndex, roomName)) = roomName

Abstract type Calendar :

Title : Calendar
Sort : Calendar
References : boolean, Date, Time, Set

Description :
The Calendar type represents a calendar that organizes and manages teaching time slots (TeachingTimeslot). It is designed to store and manipulate these time slots efficiently while ensuring data integrity.
Préconditions : Date, time and roomName are defined in **TeachingTimeslot**

---------- SIGNATURES ----------

Create : → Calendar
IsEmpty : Calendar → boolean
AddTimeslot : Calendar x TeachingTimeslot → Calendar
IsInCalendar : Calendar x TeachingTimeslot → boolean
Conflicts : Calendar x TeachingTimeslot → boolean
GetTeachingTimeslotByRoom : Calendar x Room→ Calendar
GetFreeRoomByDateAndTime : Calendar x Date x Time → Set<RoomName>

----------AXIOMES ----------
IsEmpty(Create) = True
IsEmpty(AddTimeslot(Create,TeachingTimeslot)) = False
IsInCalendar(Create,TeachingTimeslot) = False
IsInCalendar(AddTimeslot(Create,T1),T2) = **If** T1 = T2 **then** True **else** False
Conflicts(AddTimeslot(Create,T1),T2) = **If** GetRoomName(T1) = GetRoomName(T2) ∧(Before(T1,T2)∨After(T1,T2) **then** True **else** False
GetTeachingTimeslotByRoom(AddTimeslot(Create,TeachingTimeslot), **GetRoomName(TeachingTimeslot′)) =** {TeachingTimeslot′ ∈ Create | GetRoomName(TeachingTimeslot′) = GetRoomName**(**TeachingTimeslot)} ∪ ∅
// Return a set of room name available with date and time
GetFreeRoomByDateAndTime(AddTimeslot(Create, T), date, time) =
{room | room ∈ Set<RoomName> ∧ ∀T′ ∈ AddTimeslot(Create, T): (Equals(GetDate(T′), date) ∧ Equals(GetStartTime(T′), time) ∧ Equals(GetRoomName(T′), room))} ∪ ∅

Conclusion

The specifications outlined in this document are designed to directly support the aims of the project by providing a clear, structured approach to the design and implementation of core functionalities. By defining precise data formats, functional requirements, and the abstract data type for the TeachingTimeslot and Calendar, this document offers a comprehensive framework that will enable the development team to align with project goals effectively.

Recommendations for the Development Team

1. Adhere to Specifications: Follow the detailed specifications closely during development to avoid misalignment with project goals. The structured fields in each functional specification (Identifier, Title, Objective(s), Pre-condition(s), Post-condition(s), Input, Process, Output, Error Handling) will serve as a checklist to ensure full coverage.
2. Data Validation: Given the reliance on specific data formats (as per ABNF), prioritize validation mechanisms to maintain data consistency. This will be crucial for interoperability and preventing errors in data handling.
3. Testing Equivalence and Temporal Order Operations: As defined in the Data Semantics section, equivalence and temporal order operations are central to the functionality. Develop rigorous tests for these operations to confirm accurate behavior in all scenarios.
4. Iterative Feedback and Revision: Regularly review and test each feature against the specifications, and gather feedback during development. Updates or enhancements to specifications may be necessary.
5. Documenting Changes: Keep the document updated with any necessary adjustments, ensuring traceability of changes for future reference and maintenance.

In summary, the proposed specifications provide a robust foundation for achieving the project's goals. Following this guidance will support a structured development process, ultimately delivering a high-quality, reliable software solution.