Requirements Specification: Squirrel Marking System

Final Version

FritzSolms
Dept Computer Science, University of Pretoria

March 4, 2014

Contents

1 Project background

Lecturers make use of assistant lecturers and higher-level students to mark assessments like practicals, assignments, tests and exams (though exams and tests may only be marked by lecturers). The management of marking sheets, collection, aggregation and publication of marks results currently in a lot of manual labour. This is not only very inefficient, but may also result in integrity issues with lost marks and in errors being introduced during the collection and importing of marks as well as privacy issues with marks being, at times, visible to fellow students.

The above inefficiencies and other problems have led Jan Kroeze to propose and marks management system which can be accessed from mobile devices and web browsers.

2 Project vision and scope

The proposed system is a mark collection, aggregation and publication system which will allow lecturers to

- maintain course information,
- manage assessments,
- manage marks, and
- reporting.

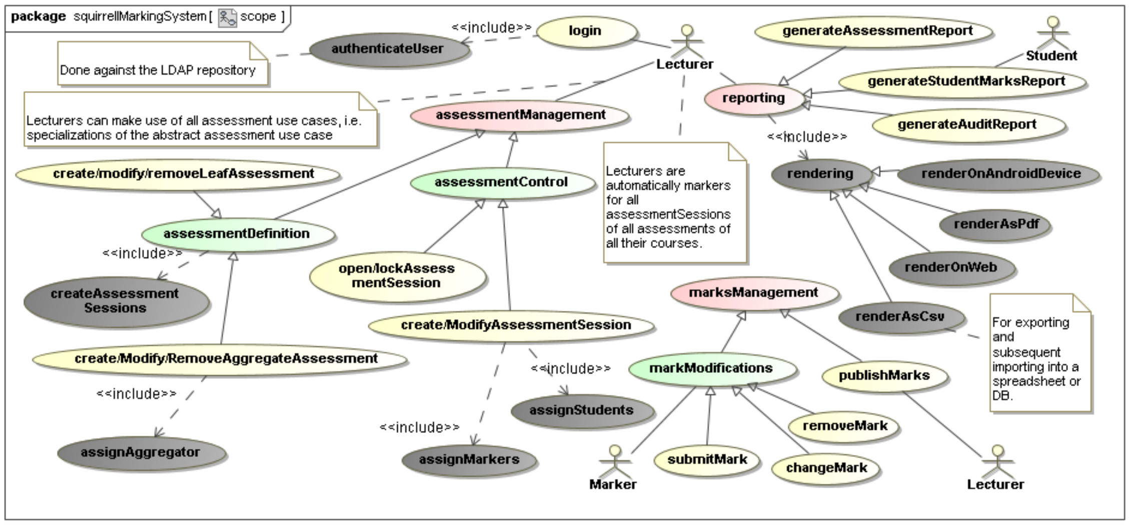

package squirrellMarkingSystem [scope]
<<include>>
login
authenticateUser
Done against the LDAP repository
Lecturer
generateAssessmentReport
Student
generateStudentMarksReport
reporting
generateAuditReport
<<include>>
Lecturers can make use of all assessment use cases, i.e. specializations of the abstract assessment use case
assessmentManagement
assessmentControl
create/modify/removeLeafAssessment
assessmentDefinition
<<include>>
createAssessment Sessions
open/lockAssess mentSession
Lecturers are automatically markers for all assessmentSessions of all assessments of all their courses.
rendering
renderOnAndroidDevice
renderAsPdf
renderOnWeb
renderAsCsv
For exporting and subsequent importing into a spreadsheet or DB.
marksManagement
create/ModifyAssessmentSession
create/Modify/RemoveAggregateAssessment
<<include>>
markModifications
publishMarks
assignStudents
<<include>>
removeMark
<<include>>
assignAggregator
assignMarkers
submitMark
changeMark
Marker
Lecturer

Figure 1: The scope of the system.

In particular, the system allow

- **administrators** to

  - downloading and updating of information around courses from the CS systems
    - including the lecturers, teaching assistants and students for a course,

- **lecturers** to
  - specify atomic leaf assessments (assessments which are given a single, atomic mark, like a question for a test or a practical which is gen a single atomic mark in the assessment system) and how these assessments are to be aggregated into higher level assessments (e.g. how the questions are aggregated into a test mark and how the test and practical marks are aggregated into a semester mark),
  - create assessment sessions with students and markers assigned to these sessions,
  - publish leaf and aggregated marks,
  - and generate assessment reports at any level of aggregation as well as audit reports and have these rendered either onto the screen, onto a PDF document or onto a CSV file for subsequent importing into a spreadsheet or a database.
- **markers** who have been assigned to mark leaf assessments of certain students[1]
  - to submit, modify or remove marks[2],
- and **students**
  - retrieve his or her marks for an assessment at any level of aggregation and have these rendered either onto the screen, onto a PDF document or onto a CSV file for subsequent importing into a spreadsheet or a database.

3 Stakeholders

The stakeholders of the system include the following

The Client The client is Jan Kroeze who originated the idea for the mini-project and who is the main source of the requirements information. Vreda Pieterse, who has been involved with teaching in the department for some time has also significantly contributed to the requirements and requirements validation.

Lecturers use the system to define assessments and aggregations of assessments and to retrieve marks.

Teaching asssistants use the system to capture marks.

Students use the system to retrieve their marks.

TechTeam is responsible with assisting with the CS system integration and has to deploy the system into production and who has to maintain the system.

WebTeam will be responsible for integrating the system into the web-front-end of the CS systems and for maintaining the web front-end.

[1] If a marker is assigned to an aggregate assessment, the marker may mark all components of that aggregate assessment,

[2] A full audit trail will be maintained by the system,

4 Architecture requirements

This section discusses the software architecture requirements – that is the requirements around the software infrastructure within which the application functionality is to be developed. The purpose of this infrastructure is to address the non-functional requirements. In particular, the architecture requirements will specify

- the architectural responsibilities which need to be addressed,
- the access and integration requirements for the system,
- the quality requirements, and
- the architecture constraints specified by the client.

4.1 Access and Integration Requirements

This section discusses

1. the requirements for the different channels through which the system can be accessed by humans and other system, and
2. the integration channels which must be supported by this system.

4.1.1 Access Channels

The system will be accessible by human users through the following channels:

1. From a web browser through a rich web interface. The system must be accessible from any of the widely used web browsers including all recent versions of *Mozilla Firefox*, *Google Chrome*, *Apple Safari* and *Microsoft Internet Explorer*.
2. From mobile *Android* devices using an *Android* application.

Other systems should be able to access the services offered by the system through either restful or SOAP-based web services.

The system should be accessible through both, human and system access channels.

4.1.2 Integration channels

This system will be able to access

- the CS LDAP server in order to retrieve student and personal details and class lists.
- the CS MySQL database to access course/module information.

Details of the respective data structures are given in appendix A.1. Further details regarding the integration to these databases and the use of the mock databases created for this project can be obtained from the *TechTeam*.

In addition, the system will allow manual integration through importing and exporting of CSV files. In particular, the system will support

- Importing of assessment entries from CSV files.
- Exporting of marks sheets to CSV files.

4.1.3 Quality requirements for acces and integration channels

All communication of sensitive data must be done securely using HTTPS.

4.2 Architectural responsibilities

The architectural responsibilities include the responsibilities of providing an infrastructure for

1. a web access channel,
2. a mobile access channel,
3. hosting and providing the execution environment for the services/business logic of the system,
4. persisting and providing access to domain objects,
5. providing an infrastructure for specifying and executing reports, and
6. integrating with an LDAP repository.

4.3 Quality requirements

The quality requirements are the requirements around the quality attributes of the systems and the services it provides. This includes requirements like performance, scalability, security, auditabilty, usability, and testability requirements.

4.3.1 Security

General security considerations

1. All system functionality is only accessible to users who can be authenticated through the LDAP system used by the department of Computer Science.
2. The system must make certain that any operation on any data is only allowed if the user may use the requested operation on the requested data object. Some services only require role based authorization for the service itself. However, the system must be able to constrain who is allowed to do what with which entity. For example, a student may see his or her results, but not those of any other students.

In particular

1. One or more persons may be assigned as marker for any assessment at any level of granularity. If the assessment is a leaf assessment, the marker may assign a mark to that assessment. If the assessment is an aggregate assessment, the marker may assign a mark to any leaf assessment of the aggregate assessment. Only markers and assessment owners may change the marks for an assessment.
2. The person creating an assessment is the assessment owner. Only the assessment owner may modify any aspects of the assessment itself including adding, changing or removing assessment components or the marks assigned to assessment components.

4.3.2 Auditability

One should be able to query for any entity, any changes made to that entity or any of its components. The information provided must include

- by whom the change was made,
- when the change was made, and
- the new and old value of the field(s) which were changed.

The system will provide only services to extract information from the audit log and will not allow the audit log to be modified.

4.3.3 Testability

All services offered by the system must be testable through unit tests which test

- that the service is provided if all pre-conditions are met (i.e. that no exception is raised except if one of the pre-conditions for the service is not met), and
- that all post-conditions hold true once the service has been provided.

4.3.4 Usability

1. An average student should be able to use the system without prior training.
2. The system must be developed using Internationalization in order to support multiple languages. Initially only English needs to be supported, but it must allow for translations to the other official languages of the University to be added at a later stage.

4.3.5 Scalability

1. The system must be able to scale to handle all assessments of all modules of the department of Computer Science.
2. The system must be able to operate effectively under th load of 100 concurrent users.

4.3.6 Performance requirements

The system does not have particularly stringent performance requirements.

1. All non-reporting operations should respond within less than 1 second.
2. Report queries should be processed in no more than 10 seconds.

4.4 Architecture constraints

The following architecture constraints have been introduced largely for maintainability reasons:

1. The system must be developed using the following technologies
   - The system must be developed using the *Django web framework.*
   - Persistence to a relational database must be done using the *Object-Relational Mapper* bundled with *Django.*
   - The unit tests should be developed using the *Django unittest module.*
2. The system must ultimately be deployed onto a *Django application server* rnning within the `cs.up.ac.za` *Apache* web server.
3. The mobile client must be running on an *Android* application
4. The system must be decoupled from the choice of database. The system will use the *MySQL* database.
5. The system must expose all system functionality as restful web services and hence may not have any application functionality within the presentation layer.
6. Web services must be published as either SOAP-based or Restful web services.

5 Application (functional) requirements

This section discusses the functional requirements for the *Squirrel Marking System.* Section 5.1 which discusses the domain objects introduces the core domain concepts and relationships between this concepts.

This is followed by the functional requirements specification for the concrete use cases of the system.

5.1 Domain objects

This section introduces core domain concepts and aspects of the requirements which cross-cut across different use cases. these concepts. These concepts are relevant for the understanding of the detailed use casse requirements.

5.1.1 Overview

The main domain objects are

- **persons** which may be assigned lecturer, marker or student roles with respect to different courses and assessments,
- **assessments** which can be aggregated which are aggregated into higher level assessments (meaning marks are aggregated into higher level marks), and for which different assessment sessions may be created, and
- mark allocations.

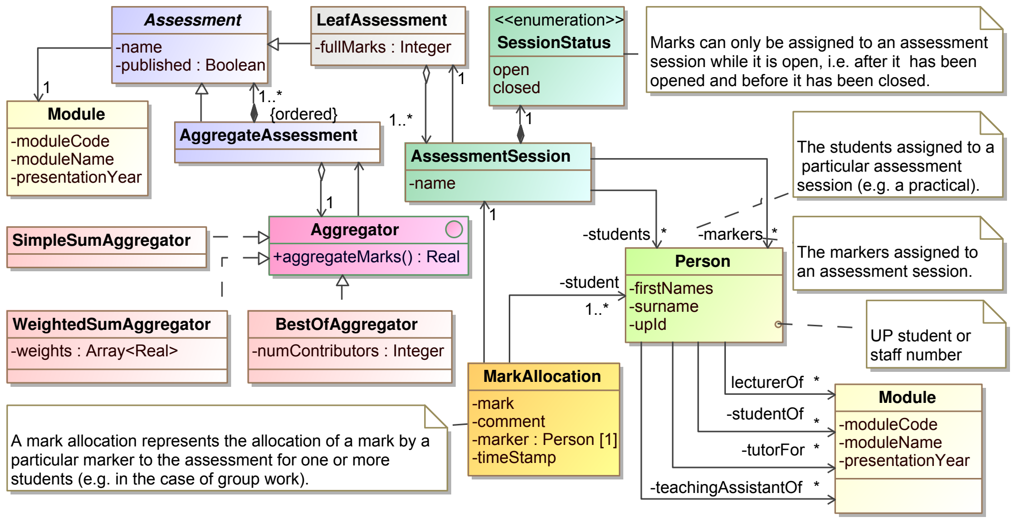


Figure 2: Overview of the data structures and relationships for the core domain objects of the system.

5.1.2 Persons

All persons known to the system must be persons which are registered within the Computer Science LDAP registry and will have to be authenticated against this registry. The person information their demographic details (name, student/personal no, . . .) including the courses assigned to lecturers and students is obtained from the CS-LDAP repository.

The system does should not have separate classes for students, markers and lecturers, but these are simply different roles people could play in the context of a course or an assessment session. A person who is a lecturer of one course could be assigned to be a marker of an assessment for another course and could register to be a student of a third course.

5.1.3 Assessment, assessment sessions and markers

Assessments are either leaf assessments or aggregate assessments. a leaf assessment is an assessment for which an atomic mark is allocated to and for which the full marks have been specified. Examples include a question in a test or exam and a mark for a practical component or for the entire practical (if only the full practical marks captured by the system). An aggregate assessment is an aggregation of lower level assessments which can themselves be aggregate assessments or leaf assessments. For example,

- the marks of the questions for a test are aggregated into a test mark,
- the marks across the tests and practicals could be aggregated into a semester mark, and
- the semester and exam marks could be aggregated into a course mark.

The default aggregator is the simple sum aggregation (i.e. the marks for the lower level assessment components are simply added up). But the lecturer may assign a different aggregator to an aggregate assessment. For now, the aggregators which need to be supported are

- a `WeightedSumAggregator` which assigns different weights to the components of the aggregate component, and
- a `BestOfAggregator` which selects only the best n components of the aggregate component and returns the sum of their allocated marks.

A leaf assessment (i.e. an assessment for which a mark is allocated by markers) has one or more assessment sessions. Each assessment session is assigned a number of markers and a subset of the students enrolled for the module. The associated markers may mark the assessment (including all its assessment components recursively) of those students allocated to that particular assessment session.

5.1.4 Assessment sessions, markers and mark allocations

By default any assessment associated with a module may only be marked by the persons who are assigned lecturers for that module. Thus by default an assessment has a single assessment session which contains all registered students for the course and only the lecturers as markers.

A lecturer can assign any person within the CS-LDAP system as an additional marker. Alternatively, a lecturer may create multiple assessment, each with a subset of the students and each with its assigned markers.

Markers may be assigned to be able to mark and assessment for a set of students. The marker will be able to supply marks for the allocated students for all the leaf assessments contained within any of the assessments assigned to him or her.

Assessment sessions can be opened and closed for markling. By default, assessment sessions are closed and need to be opened for marking by one of the lecturers of the course. Marks can only be allocated by assigned markers and only whilst the assessment session is open.

Any marker assigned to an assessment session for an aggregate assessment may mark all its components. Any markers assigned to additional assessment session for specific components are in addition to the markers assigned at the higher level of aggregation.

5.2 Login

All system functionalities are only accessible to users who have been able to log into the system.

5.2.1 Service contract

The service request for the `login` use case contains the authentication credentials (username and password). If the LDAP system is available and if the user could be authenticated, the response contains the user's name and uid (student or staff number) as well as the information of the different roles the user has been assigned on different modules.

Note that its is the service's responsibility to construct the required response object.

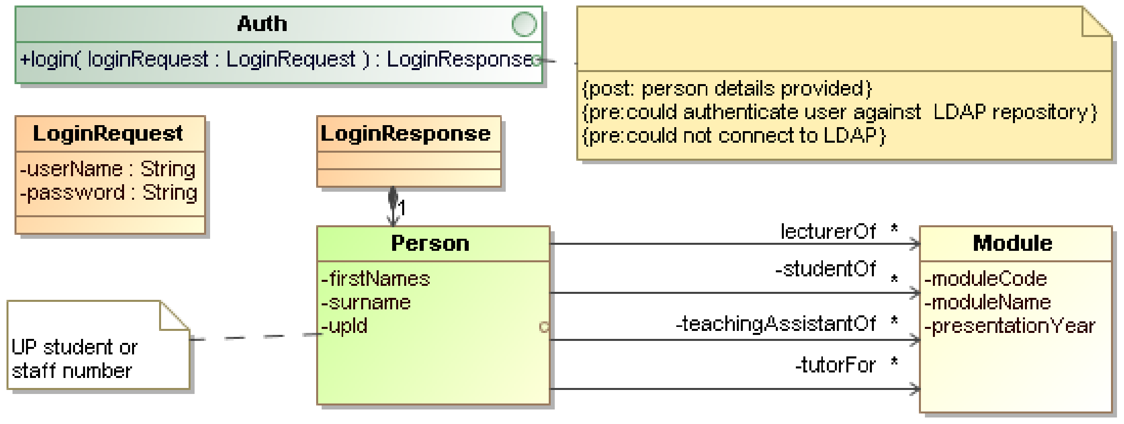


Figure 3: The service contract for the login use case.

5.2.2 Functional requirements

The functional requirements for the `login` use case include the authentication against LDAP and the sourcing of required information from the LDAP repository as well as the ultimate construction of the result object.

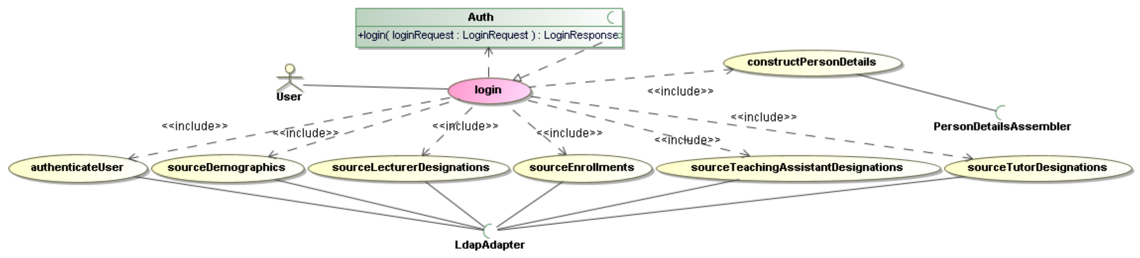


Figure 4: The functional for the login use case.

5.2.3 Process specification

The provided user authentication credentials are used to authenticate the user against the LDAP repository. If this is unsuccessful, a corresponding exception is raised. otherwise the person's demographics as well as the designated lecturer, teaching-assistant, tutor and student roles with respect of different modules is sourced from the LDAP repository and the person details are constructed as required by the services contract specified in section 5.2.1.

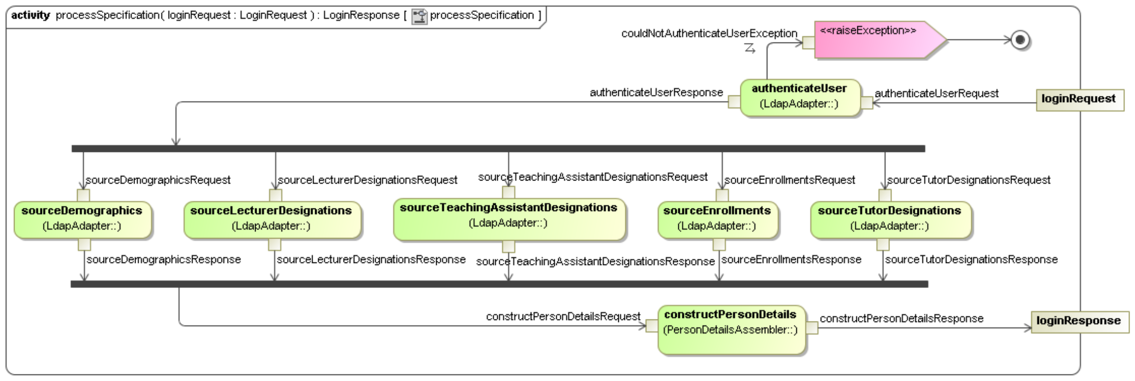

activity processSpecification(loginRequest : LoginRequest) : LoginResponse [processSpecification]
couldNotAuthenticateUserException
<<raiseException>>
authenticateUserResponse
authenticateUser (LdapAdapter::)
authenticateUserRequest
loginRequest
sourceDemographicsRequest
sourceLecturerDesignationsRequest
sourceTeachingAssistantDesignationsRequest
sourceEnrollmentsRequest
sourceTutorDesignationsRequest
sourceDemographics (LdapAdapter::)
sourceLecturerDesignations (LdapAdapter::)
sourceTeachingAssistantDesignations (LdapAdapter::)
sourceEnrollments (LdapAdapter::)
sourceTutorDesignations (LdapAdapter::)
sourceDemographicsResponse
sourceLecturerDesignationsResponse
sourceTeachingAssistantDesignationsResponse
sourceEnrollmentsResponse
sourceTutorDesignationsResponse
constructPersonDetailsRequest
constructPersonDetails (PersonDetailsAssembler::)
constructPersonDetailsResponse
loginResponse

Figure 5: The process specification for the login use case.

5.3 Assessment use cases

The system will allow lecturers to create, modify and remove aggregate and leaf assessments. An assessment is by default not published implying that students cannot, b default, query their marks for an assessment. Once a lecturer has published an assessment, students can query marks for that assessment or any of its components.

Leaf assessments are assessments to which an atomic mark is assigned. Aggregate assessments are not assigned marks. Instead the mark is calculated by aggregating the marks of the component assessments. How these are aggregated depends on the type of aggregator assigned to the aggregate assessment. Examples are simple summation aggregators, weighted sum aggregators or best-of aggregators.

5.3.1 Create leaf assessment

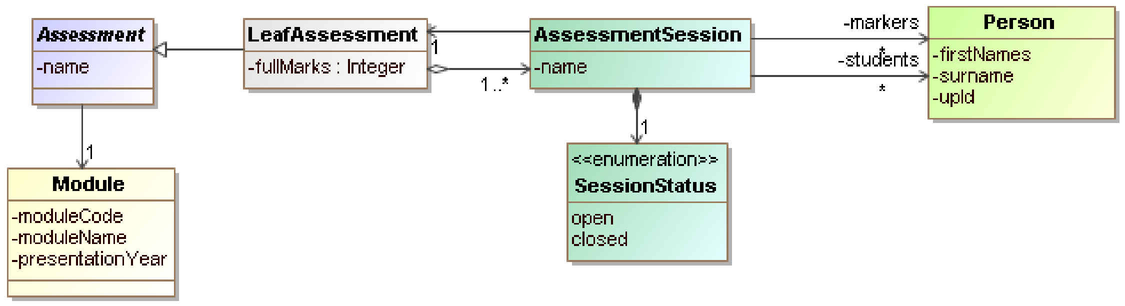


Figure 6: The structure of a leaf assessment.

Creating a leaf assessment requires the lecturer to specify

- the course for which this is an assessment,
- a name for the assessment,
- and the full marks for the assessment (e.g. a practical which counts out of 20), and
- one or more assessment sessions which have a name and is either open or closed for marking and which have a subset of the enrolled students as well as potentially a set of markers assigned for the assessment session[3].

The data structure created by the use case is shown in figure 6.

5.3.2 Create aggregate assessment

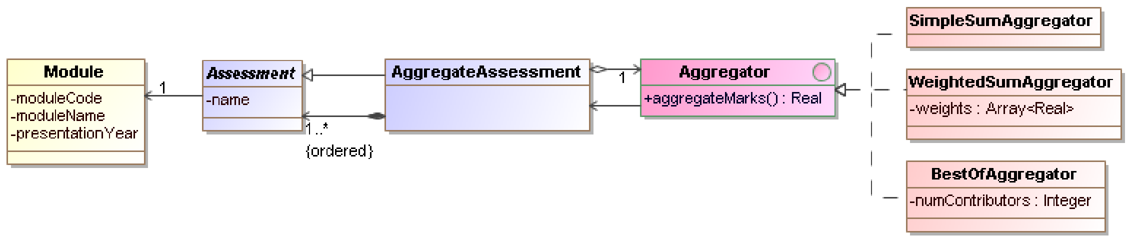


Figure 7: The structure of a aggregate assessment.

When creating aggregate assessments, lecturers need to specify

1. the course for which this is an assessment,
2. the assessment name,
3. the asssessments which are to be the components of the aggregate assessment, and
4. an aggregator which determines how the marks of the component assessments are aggregated into a mark for the aggregate assessment.

The data structure created by the use case is shown in figure 7.

5.4 Marks management

Lecturers can open and close assessment sessions for marking. A person may only assign a mark for a leaf assessment for a specific student if

- the assessment session for that student is open,
- the person is assigned by one of the lecturers of the course as a marker for the assessment session for that assessment and student, or

[3]By default there is a single assessment session for all enrolled students with only the course lecturers being assigned as markers and assessment sessions are, by default closed.

- if the person is assigned as marker for one of the aggegate assessments containing the leaf assessment with such an assessment session.

The system will prevent a marker from assigning a mark which exceeds the fullMarks for the leaf assessment.

5.5 Reporting

The reporting use cases purely provide raw and processed information from the system without altering any information within the system (except adding some audit log entries for the report generation itself). This includes the generation of assessment reports, student makes reports and audit reports. All report can be rendered either onto an Android device, a eb interface, a PDF document or a CSV file for later importing into a database or spreadsheet.

5.5.1 Generate assessment report

Assessment reports can be generated at any level of aggregation. All marks are aggregated to the level at which the assessment report is requested. In addition a statistical analysis of the marks is done which includes the calculation of the class average as well as a frequency analysis.

The assessment report request has the information required to identify the asessment for which an assessment report is requested as well as the details for the frequency analysis required.

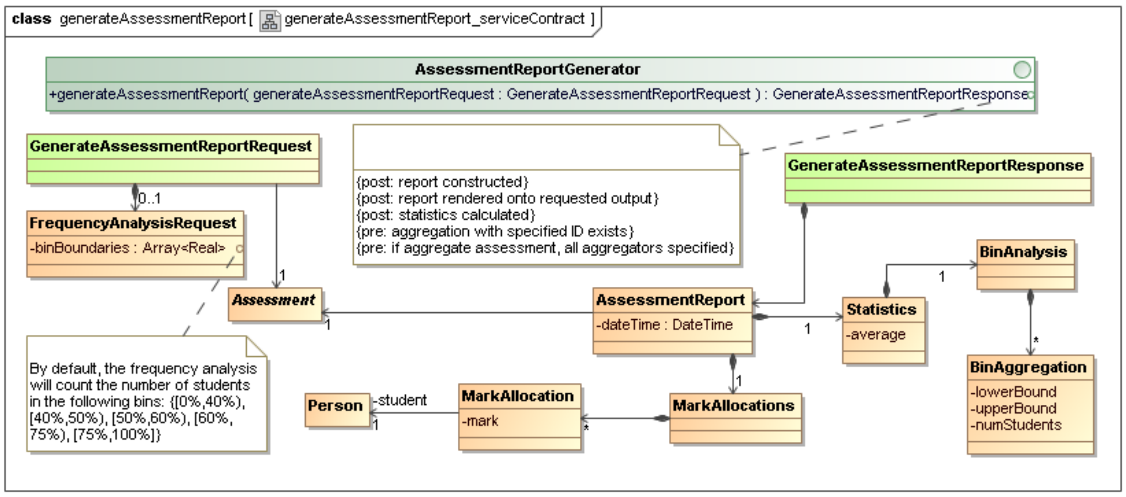

class generateAssessmentReport [generateAssessmentReport_serviceContract]
AssessmentReportGenerator
+generateAssessmentReport(generateAssessmentReportRequest : GenerateAssessmentReportRequest) : GenerateAssessmentReportResponse
GenerateAssessmentReportRequest
0..1
FrequencyAnalysisRequest
-binBoundaries : Array<Real>
{post: report constructed}
{post: report rendered onto requested output}
{post: statistics calculated}
{pre: aggregation with specified ID exists}
{pre: if aggregate assessment, all aggregators specified}
GenerateAssessmentReportResponse
BinAnalysis
1
Assessment
1
1
AssessmentReport
-dateTime : DateTime
1
Statistics
-average
*
BinAggregation
-lowerBound
-upperBound
-numStudents
By default, the frequency analysis will count the number of students in the following bins: {[0%,40%), [40%,50%), [50%,60%), [60%, 75%), [75%,100%]}
Person
-student
1
MarkAllocation
-mark
*
1
MarkAllocations

Figure 8: The service contract for the login use case.

The response object contains the information of the report rendered in the required format (e.g. on a web page, a PDF document or a CSV file). It contains information about which assessment this is an assessment report, a date/time stamp, the mark allocations for the different students of the module and the results of the simple statistical analysis which includes the class average as well as the results of the frequency analysis.

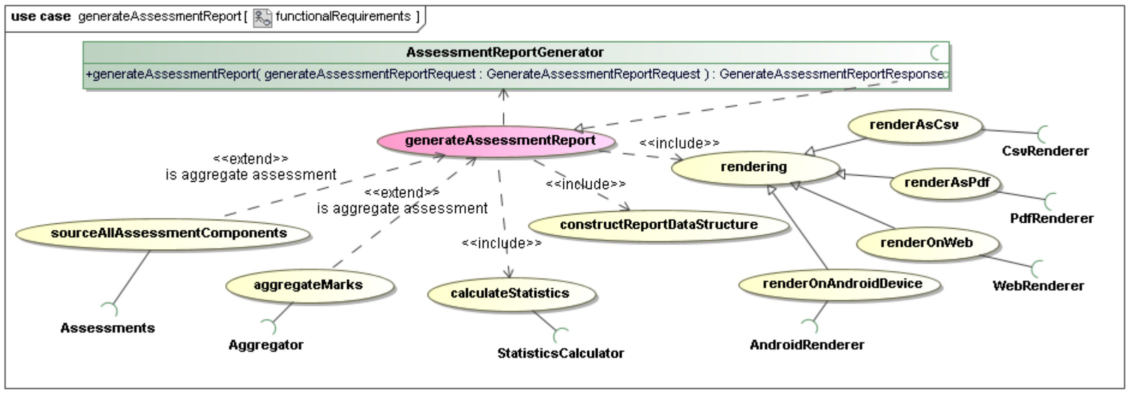

use case generateAssessmentReport [functionalRequirements]
AssessmentReportGenerator
+generateAssessmentReport(generateAssessmentReportRequest : GenerateAssessmentReportRequest) : GenerateAssessmentReportResponse
generateAssessmentReport
<<extend>> is aggregate assessment
<<extend>> is aggregate assessment
<<include>>
<<include>>
<<include>>
sourceAllAssessmentComponents
aggregateMarks
calculateStatistics
constructReportDataStructure
rendering
renderAsCsv
renderAsPdf
renderOnWeb
renderOnAndroidDevice
Assessments
Aggregator
StatisticsCalculator
AndroidRenderer
CsvRenderer
PdfRenderer
WebRenderer

Figure 9: The functional for the generate assessment-report use case.

The functional requirements for the `generateAssessmentReport` use case include the sourcing and aggregation of all assessment components, the calculation of statistics, the construction of the report data object and ultimately the rendering of the report onto either a user interface, a PDF document or a CSV file. The latetr is done by appropriate report renderers.

5.5.2 Generate student marks report

Students can use the systm to generate themselves marks reports at any level of granularity. The marks report will contain, in a tree structure, all leaf and aggregated marks up to the level of aggregation for which the report was requested.

Like all other reports, the student marks report can also be rendered onto web and Android based UIs as well as PDF and CSV files.

The student marks report request has sufficient information to identify both, the student and the assessment for which a marks report is required.

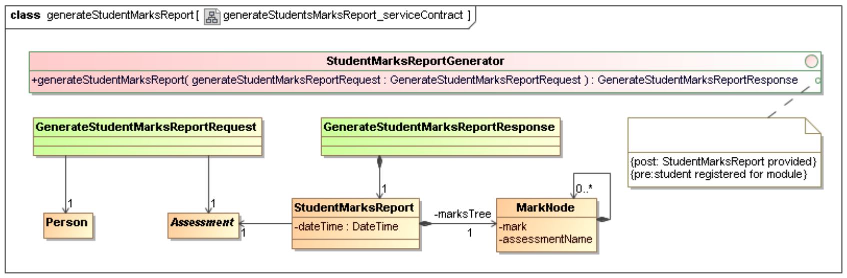

class generateStudentMarksReport [generateStudentsMarksReport_serviceContract]
StudentMarksReportGenerator
+generateStudentMarksReport(generateStudentMarksReportRequest : GenerateStudentMarksReportRequest) : GenerateStudentMarksReportResponse
GenerateStudentMarksReportRequest
GenerateStudentMarksReportResponse
{post: StudentMarksReport provided}
{pre:student registered for module}
Person
Assessment
StudentMarksReport
-dateTime : DateTime
-marksTree
MarkNode
-mark
-assessmentName
1
1
1
1
1
0..*

Figure 10: The service contract for the generate student marks report use case.

The response object has a date/time stamp and the marks tree which contains the assessment name and allocated mark as well as all contributing (aggregated) assessments and their marks.

5.5.3 Generate audit report

Lecturers may request audit reports for any assessment which will contain for any audit event

- A date/time stamp.
- The userId for the session within which the event occured.
- The action which was performed by the user.

The audit events will include

- all assessment creations, modifications and removals,
- all assessment session creations, modifications and removals,
- all mark submissions, modifications and removals,
- any requests to open/close assessment sessions,
- any requests to publish marks,
- any requests for any reports including assessment reports, students marks reports and audit reports.

6 Requirements for the development process used

- All changes to produced artifacts must be made through the `git` repositories assigned to the project.
- All functionality must be unit-tested.

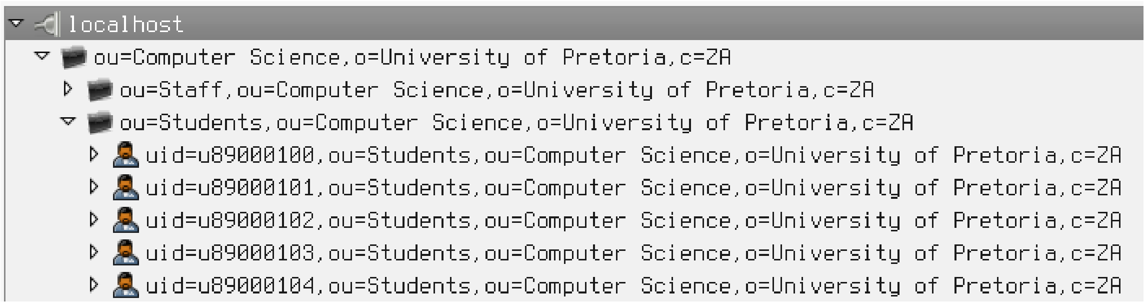
A Appendices

A.1 External database structures

The relational database structure containing the courses/module information has the following structure:

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
code	varchar(20)	YES		NULL	
name	varchar(255)	NO			
lecturer	varchar(255)	NO		0	
description	text	YES		NULL	
semester	smallint(6)	NO		0	
has_webct	tinyint(4)	YES		NULL	
year_group	int(11)	YES		NULL	
hidden	tinyint(3) unsigned	NO		0	
last_updated	datetime	NO		0000-00-00 00:00:00	
discussion_board	tinyint(4)	YES		NULL	
tutors_allowed	tinyint(2)	YES		NULL	

The structure of the LDAP repository is shown in the following figure



```
localhost
  ou=Computer Science,o=University of Pretoria,c=ZA
    ou=Staff,ou=Computer Science,o=University of Pretoria,c=ZA
    ou=Students,ou=Computer Science,o=University of Pretoria,c=ZA
      uid=u89000100,ou=Students,ou=Computer Science,o=University of Pretoria,c=ZA
      uid=u89000101,ou=Students,ou=Computer Science,o=University of Pretoria,c=ZA
      uid=u89000102,ou=Students,ou=Computer Science,o=University of Pretoria,c=ZA
      uid=u89000103,ou=Students,ou=Computer Science,o=University of Pretoria,c=ZA
      uid=u89000104,ou=Students,ou=Computer Science,o=University of Pretoria,c=ZA
```

For example, an LDAP search for stud_COS301 returns:

```
dn:            cn=stud_COS301,ou=Modules,ou=Groups,ou=Computer Science,o=University of Pretoria,c=ZA
objectClass:   top
objectClass:   posixGroup
cn:            stud_COS301
gidNumber:     10093
memberUid:     u11061015
memberUid:     u12214834
memberUid:     u11063612
memberUid:     u11002566
memberUid:     u12019837
memberUid:     u11371910
memberUid:     u12148858
memberUid:     u29557373
memberUid:     u11247143
```

and an LDAP search for u29052735 returns:

```
dn:              uid=u29052735,ou=students,ou=Computer Science,o=University of Pretoria,c=ZA
objectClass:     inetOrgPerson
objectClass:     posixAccount
objectClass:     top
uidNumber:       26526
gidNumber:       10001
loginShell:      /bin/bash
title:           Mr
initials:        CJ
st:              8911085042083
sn:              van Rooyen
homeDirectory:   /home/cs/students/u29052735
employeeNumber:  29052735
uid:             u29052735
mail:            nexusdk@gmail.com
cn:              Neels
```

and an LDAP search for memberuid=u29052735 returns:

```
cn:   stud_COS333
cn:   stud_COS301
cn:   stud_COS326
cn:   stud_COS216
cn:   stud_COS330
```