Software Requirements Specification and Technology Neutral Process Design
¡Name of System/Project¿

Version: ¡versionNo¿

¡Organization/group/person authoring the document¿

February 17, 2015

Contents

1 Introduction

The requirements specification should ultimately contain sufficient information such that the system coudl be largely developed by a third party without further input. To this end the requirements must be precise and testable.

The requirements need not be fully specified up-front. One might start with the vision, scope and architectural requirements, perform an upfront software architecture engineering phase and then iteratively elicit the detailed requirements for a use case, build, test and deploy the use case before adding the detailed requirements for the next use case. Such an approach follows solid engineering phase for the core software infrastructure/architecture with an agile software development approach within which the application functionality is developed iteratively.

2 Vision

A short discussion of the project vision, i.e. what the client is trying to achieve with the project and the typical usage scenarios for the outputs of the project.

3 Background

A general discussion of what lead to the project including potentially

- business/research opportunities,
- opportunities to simplify/improve some aspect of life/work or community,
- problems your client is currently facing,
- ...

4 Architecture requirements

The software architecture requirements include the access and integration requirements, quality requirements and architectural constraints.

4.1 Access channel requirements

Specify the different access channels through which the system's services are to be accessed by humans and by other systems (e.g. Mobile/Android application clients, Restful web services clients, Browser clients, ...).

4.2 Quality requirements

Specify and quantify each of the quality requirements which are relevant to the system. Examples of quality requirements include performance, reliability, scalability, security, flexibility, maintainability, auditability/monitorability, integrability, cost, usability. Each of these quality requirements need to be either quantified or at least be specified in a testable way.

4.3 Integration requirements

This section specifies any integration requirements for any external systems. This may include

- the integration channel to be used,
- the protocols to be used,

- API specifications in the form of UML interfaces and/or technology-specific API specifications (e.g. WSDLs, CORBA IDLs, ...), and
- any quality requirements for the integration itself (performance, scalability, reliability, security, auditability, ...).

4.4 Architecture constraints

This specifies any constraints the client may specify on the system architecture include

- technologies which MUST be used,
- architectural patterns/frameworks which must be used (e.g. layering, Services Oriented Architectures, ...)
- ...

5 Functional requirements and application design

This section discusses the application functionality required by users (and other stakehodlers).

5.1 Use case prioritization

Consider a simple three-level prioritization with

Critical: A use case which is absolutely essential (ask whether the project should be canceled if that functionality could not be provided).

Important: The system would still be useful without some of the important use cases, but the client would get quantifiably less value from the system.

Nice-To-Have: Its a requirement but the value to the client/business is insignificant/not quantifiable.

5.2 Use case/Services contracts

For each use case/service specify

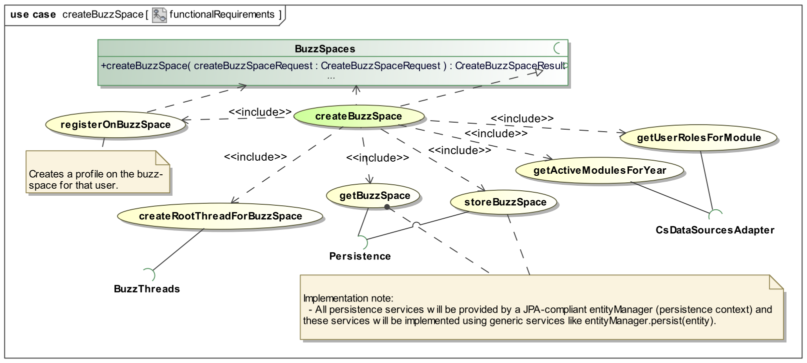
Pre-Conditions: the conditions under which the service may be refused (usually there is an exception associated with each pre-condition).

Post-Conditions: the conditions which must hold true after the servcie has been provided.

Request and Results Data Structures: Use class diagrams to specify the data structure requirements for the request and result objects (i.e. the inputs and outputs).

5.3 Required functionality

Use for each concrete use case a use case diagram with the required functionality in the form of includes and extends relationships to lower level use cases – this may be specified across levels of granularity.



Creates a profile on the buzz-space for that user.

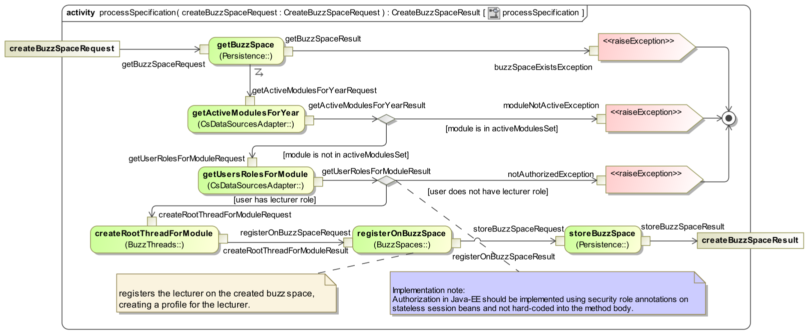
Implementation note:
- All persistence services will be provided by a JPA-compliant entityManager (persistence context) and these services will be implemented using generic services like entityManager.persist(entity).

Figure 1: An example of a UML use case diagram depicting the required functionality for a use case.

An example UML use case diagram specifying the functional requirements for a use case is show in Figure 1. We show the required functionality and the services domains represented by interfaces from which these services will be sourced.

5.4 Process specifications

For some of the use cases there may be requirements around the process which needs to be followed. If so, these requirements are typically specified via activity and/or sequence diagrams or alternatively via state charts.


activity processSpecification(createBuzzSpaceRequest : CreateBuzzSpaceRequest) : CreateBuzzSpaceResult [processSpecification]
createBuzzSpaceRequest
getBuzzSpaceRequest
getBuzzSpace (Persistence::)
getBuzzSpaceResult
<<raiseException>>
buzzSpaceExistsException
getActiveModulesForYearRequest
getActiveModulesForYear (CsDataSourcesAdapter::)
getActiveModulesForYearResult
moduleNotActiveException
<<raiseException>>
[module is in activeModulesSet]
[module is not in activeModulesSet]
getUserRolesForModuleRequest
getUsersRolesForModule (CsDataSourcesAdapter::)
getUserRolesForModuleResult
notAuthorizedException
<<raiseException>>
[user does not have lecturer role]
[user has lecturer role]
createRootThreadForModuleRequest
createRootThreadForModule (BuzzThreads::)
registerOnBuzzSpaceRequest
createRootThreadForModuleResult
registerOnBuzzSpace (BuzzSpaces::)
storeBuzzSpaceRequest
registerOnBuzzSpaceResult
storeBuzzSpace (Persistence::)
storeBuzzSpaceResult
createBuzzSpaceResult

registers the lecturer on the created buzz space, creating a profile for the lecturer.

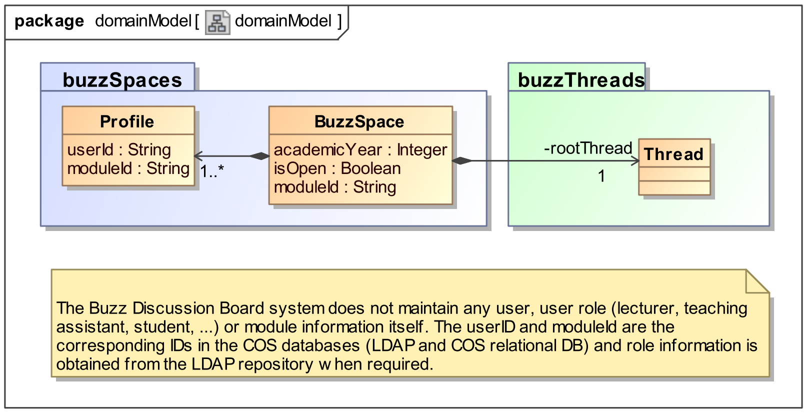
Implementation note:
Authorization in Java-EE should be implemented using security role annotations on stateless session beans and not hard-coded into the method body.

Figure 2: An example of a process specification for a use case.

An example UML of an activity diagram for a process specification is shown in Figure 2. The outer activity is the activity of the service for which we are doing the process design. It starts with a request. For each pre-condition there is a path leading to an exception being thrown and the service being aborted. If all pre-conditions are met, the service will return the return value. Note that each inner activity is a UML call operation, requesting a particular service from a particular interface, i.e. ultimately from some class realizing the service contract represented by that interface. This ensures decoupling of clients and services, i.e. that we can plug in any service provider implementing a services contract and that we are not locked into using a particular service provider (class).

5.5 Domain Model

Use UML class diagrams to specify the data structure requirements in a technology neutral way. These can ultimately be mapped onto different technologies like ERD diagrams/relational databases, XML schemas, Python/Java/C++/...objects, paper based or UI forms, ...but those are just different technology mappings and this would not be part of the requirements specification.



The Buzz Discussion Board system does not maintain any user, user role (lecturer, teaching assistant, student, ...) or module information itself. The userID and moduleId are the corresponding IDs in the COS databases (LDAP and COS relational DB) and role information is obtained from the LDAP repository w hen required.

Figure 3: An example of a UML class diagram for a domain model.

An example UML class diagram for a domain model is shown in Figure 3.

The domain model will show the domain objects, their attributes and potentially methods and the relationships between them (e.g. association, aggregation, composition, specialization, realization and containment). Dependency relationships are usually not explicitly shown.

6 Open Issues

Discuss in this section

- any aspects of the requirements which still need to be specified,
- around which clarification is still required, as well as
- any discovered inconsistencies in the requirements.