CARLETON UNIVERSITY

cuPID

(Carleton University Project Partner Identifier)

Requirements Analysis Document

Team Insomnia

Dabeluchi Ndubisi

Charlie Li

Pierre Seguin

Submitted to:

Dr. Christine Laurendeau

COMP 3004 Object-Oriented Software Engineering

School of Computer Science

Carleton University

September-21-205

Contents

Tables

Figures

SECTION 1: Introduction

1.1 Purpose

The Carleton University Project Partner Identifier (cuPID) system is used by professors or instructors to create fairly matched teams for their projects. Its goal is aimed at fixing the problems associated with matching university students into project teams based on skill set, personality and work habits in order to maximize compatibility between the students. The cuPID system aims to fix this problem by automatically matching students into teams that will best suit them without loss of compatibility of the team members as a whole. Based on the fact that compatibility is not a characteristic that can accurately be measured, the cuPID system strives to ensure a team compatibility match that is within a 2.5% margin of error based on the discrepancies in skill set, personality and work habits of the team members.

The intended users of the cuPID system are professors or instructors (Administrators) of a particular academic course that requires the execution of a project within its curriculum. In order to use the cuPID system, the instructors or the professors of this course will create a project and set the specifications of which they intend to bind the project teams. The students add themselves to the project, and the Administrators launches the cuPID algorithm to automatically create teams consisting of project partners who are compatible with each other based on their specified qualifications.

1.2 Overview

The following document is created to introduce the specifications and pre-development design decision for the cuPID system. The format of this document will aid in constructing a thorough mental model of the modus operandi of the cuPID system. The structure of the document is highlighted in details in the next subsequent paragraphs. The aim of this is to give the reader a good outline of what the document plans to achieve and also equip him with background information of the essence of each of the proposed sections.

The ***Functional Requirements*** section which is contained in section 2 (proposed system) of this document is structured to highlight the key feature based interactions that occur between the cuPID system and the environment. In the scope of this project, the environment is only the user that may be using the cuPID system either as a StudentUser or an AdministratorUser. The essence of this section is to provide a list of proposed features that the cuPID system must possess in agreement with the client. Table 1 provides a detailed list of the functional requirements of the cuPID project.

The ***Non-Functional Requirements*** section, also contained in section 2 (proposed system) of this document aims to provide the reader with a set of system based constraints attached to the cuPID system. These constraints are not directly related to the key features of the cuPID system, but their existence is of paramount importance in the sense that they enhance specific aspects of the system which may be directly or indirectly beneficial to the user. Table 2 provides a detailed list of the Non-Functional requirements of the cuPID project.

The ***System Model,*** contained in section 2 (proposed system) of this document aims to provide the user with a better understanding of how the system operates. It starts off with the ***Use Case Model***. The relevance of this model is to provide the reader with a better understanding of how the functional requirements of the cuPID project translates to a direct interaction that occurs between the user and the cuPID system. In this model, participators, entry and exit conditions, along with interaction flow of a particular feature is highlighted. This should help the reader formulate a conceptual model of how the cuPID system operates.

The next model contained in the system model is the ***Object Model***. The aim of this section is to provide the reader with a good understanding of the functional entities that make up the cuPID system. Care has been taken to outline the high level attributes of each Entity that makes up the system. It should be noted that these attributes are subject to change as, this is only a high level estimate of the general functionality of the system. The object model provides the reader with UML class diagrams which depict the relationships between entities of the system, and a Data dictionary for each entity which provides a detailed explanation of the attributes of an entity and how the entity functions in the cuPID ecosystem.

The final system model is the ***Dynamic Model***. This model is set to illustrate the interaction that occurs between entities in the system for a particular Use case. The portrayal of these illustrations is aided using UML sequence diagrams. These diagrams provide a chronological illustration of the sequence of a particular interaction that occurs between the system and the user.

Finally, an abbreviations section is provided to the reader. This serves as an index for high level jargon used in the document. The user is advised to visit this section on the onset that he/she finds an abbreviation that eludes his/her understanding.

SECTION 2: Proposed System

2.1 Overview

This section highlights the key features and entities of the cuPID system, and how these entities interact with one another in order to provide said features. The cuPID system is designed to be project centric, and hence, all key features revolve around the project in question. In the cuPID system, the StudentUsers will have the ability to interact with the available projects created by the AdministratorUsers in various ways which include adding and removing themselves from a project. The AdministratorUsers will have the ability to create projects and also launch the Project Partner Identification (PPID) algorithm on the StudentUsers in the project in order to create teams. It will be important to point out that the system is to be designed in order to maximize user experience without the expense of key functionalities or efficiency of the system as a whole.

In the aim of eliminating potential ambiguities that may arise, care has been taken to categorize the features of the cuPID system into Functional and Non-Functional requirements.

This section highlights these requirements in detail. Table 1 highlights the Functional requirements of the system, and Table 2 highlights the Non-Functional requirements of the system.

2.2 Functional Requirements

Functional requirements in the cuPID system describes the key interactions between the cuPID system and the environment. Since the cuPID system is designed to be self contained at the moment of writing this document, the environment in this case comprises only of the user. Table 1 highlights the Functional requirements of the cuPID system. Unique numbers have been assigned to each of the requirements to ease traceability.

Table 1 - Functional Requirements

FR-01	StudentUsers must be able to interact with his/her unique PPP. StudentUsers must have the ability to:
FR-01-01	edit or create his/her unique PPP
FR-01-02	outline personal qualifications in his/her PPP
FR-01-03	outline qualifications that he/she will prefer in potential teammates that he/she may be matched with.
FR-02	StudentUsers must be able to interact with available Projects. StudentUsers must have the ability to:

	FR-02-01	view a list of available projects
	FR-02-02	see the description of any project they wish to add themselves to.
	FR-02-03	add themselves to a project
	FR-02-04	remove themselves from a project
FR-03		AdministratorUsers must be able to interact with Projects. AdministratorUsers must have the ability to:
	FR-03-01	view their list of created projects
	FR-03-02	see the description of the projects that they created
	FR-03-03	create projects
	FR-03-04	edit project configurations (manage/modify essential project data including but not limited to team size)
	FR-03-05	launch PPID algorithm on StudentUsers registered in a particular project
FR-04		AdministratorUsers must have the ability to view the match summary and detailed results of the PPID algorithm.

2.3 Non-Functional Requirements

The Non-Functional requirements of the cuPID system highlight the constraints or added functionalities added to the cuPID system that are not directly related to the functional behavior of the system listed in the sub-section above. These constraints are added in order to enhance the aspects of the system including but not limited to: usability, reliability, performance, and supportability of the system as a whole. Table 2 highlights the Non-Functional requirements of the cuPID system. Unique numbers have been assigned to each of the requirements to ease traceability.

Table 2 - Non-Functional Requirements

NFR-01	**Usability**: Detailed How-To Guide must be provided to the user.
NFR-02	**Usability**: Where applicable, keyboard shortcuts for particular actions must be available to the user.
NFR-03	**Usability**: Feedback should be given to user when time is required for loading algorithm or fetching data.
NFR-04	**Usability**: Helpful hints in the form of tool tips must be available to user via an option on the screen to provide information about the consequence of a particular action that the user can take.
NFR-05	**Reliability**: Information contained in projects created by AdministratorUsers must be persistent on exiting the application.
NFR-06	**Reliability**: All input received from user must be filtered to prevent harmful data that may compromise the system.
NFR-07	**Reliability**: A StudentUser must be able to interact with his/her own PPP
NFR-08	**Reliability:** The system must be able to restart itself on the event of a crash that occurs during the system activities.
NFR-09	**Performance**: Queries to the database must return result within 1 second at least 95% of the time.
NFR-10	**Performance**: Algorithm must not take more than 5 seconds to match StudentUsers into teams of any project.
NFR-11	**Performance**: The system must be able to respond to user interaction while it is running algorithm to create teams.
NFR-12	**Supportability**: Online documentation of the system must be provided in order to provide information for open source extensibility of the system.
NFR-13	**Supportability**: System must be able to support international conventions such as language, units, and number formats.

NFR-14	**Supportability**: Core system code and database should be extensible in order to provide an API for future web application version.
NFR-15	**Implementation**: Must run with single host on Ubuntu 14.04 LTS platform
NFR-16	**Implementation**: All source code must be written in C++.
NFR-17	**Implementation**: User Interface tests and Unit tests for system functionality must be developed with system to aid testing team.
NFR-18	**Interface**: System must provide functionality to print match results generated by the algorithm.
NFR-19	**Interface**: The system must provide functionality to import user qualifications from facebook.
NFR-20	**Interface**: The system must be able to allow users to login to the cuPID system via social network APIs like facebook or twitter.
NFR-21	**Operation**: The system must allow only one user to be logged in at any given time.
NFR-22	**Operation**: The AdministratorUser must not be able to launch the PPID algorithm if the number of registered StudentUsers is less than the team size specification for the project.
NFR-23	**Operation**: StudentUsers should not be able to register themselves to a project if they do not have a profile.
NFR-24	**Packaging**: The cuPID software must be available for installation via download or CD.
NFR-25	**Packaging**: The system must provide option to install dependencies (such as sqlite) during installation process.
NFR-26	**Packaging**: The system must provide a means of updating the software, and all updates must be compatible with any previous documents used by the system.
NFR-27	**Legal**: In compliance with the Carleton University privacy act, on no occasion must the system provide private user information to another user of the system.

NFR-28	**Legal**: As required by local privacy laws, StudentUsers must be able to delete their profiles and all private information provided by the user must never remain in the the system.
NFR-29	**Legal**: By accessibility laws of WCAG 2.0, cuPID must be able to be integrated with a screen reader for the visually impaired.

2.4 System Models

This section will describe the nature of the cuPID system in terms of its structure and its dynamic interoperations.

As a simple guide to this section, this document will illustrate the interaction scenarios between system and user (actor) through use case model. With functionalities of the system defined, the functionalities of the use case will be abstracted into object models, which comprises a data dictionary and class diagrams for the entities that make up the cuPID system. The object model will focus on providing the reader with an overview of the entities that make up the cuPID system and how they relate to one another. Lastly, this section will provide the dynamic model of the cuPID system, which captures the interactions involved between entities of the cuPID system in response to specific actions initiated by the user.

2.4.1 Use Case Models

The basis of the functionality of this system requires two typical users to create content for the system, administrator and student user. A Use Case is an abstraction of a particular scenario. In this section, the specific interaction user(s) or actor(s) undertake, will be investigated and defined precisely. Each use case describes an action that takes place between the actor(s) and system from the entry condition, to the flow of events and requirements needed to initiate key processes, to the exit conditions of the use case.

This section will first introduce the general functionalities that the actors can interface with, proceeding with a dissection of its generality in to lower level use cases. Each use case will have its own diagram to describe the relationship with the system as well as a text description capturing crucial details such as: participating actors, flow of event, entry/exit conditions, quality requirements, and traceability tags in reference of the functional and nonfunctional requirements.

Use Case Overview

The high level use case diagram for the cuPID system is illustrated below in Figure 1. The essence of this high level diagram is to abstract all functionalities of the system to the main activities that can easily describe the cuPID system. Table 3 highlights the high level use case descriptions for the diagram illustrated below.

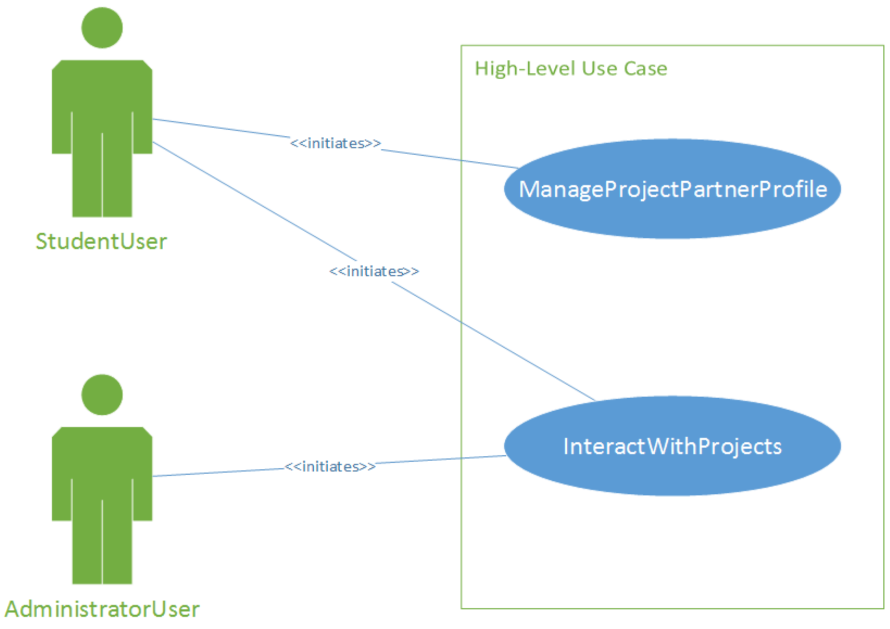

Figure 1 - High-Level Use Case Diagram

Table 3 - High-Level Use Case Descriptions

UC-01	**ManageProjectPartner Profile**	The StudentUser modifies the information in his Project Partner Profile
UC-02	**InteractWithProjects**	The StudentUser or AdministratorUser interacts with available projects

The high level use case diagram in Figure-1 is broken down into more detailed subcases so as to provide a more detailed understanding of how the functionalities of the system relate with one another. Figure 2 illustrates the detailed use case diagram for the StudentUser ManageProjectPartnerProfile use case. The StudentUser initiates the high-level Use case ManageProjectParnterProfile (UC-01). This use case then includes other use case in order to produce a good dependency and abstraction of the cuPID system. Table 4 highlights the use case descriptions for the StudentUser ManageProjectPartnerProfile Use Case diagram illustrated below.

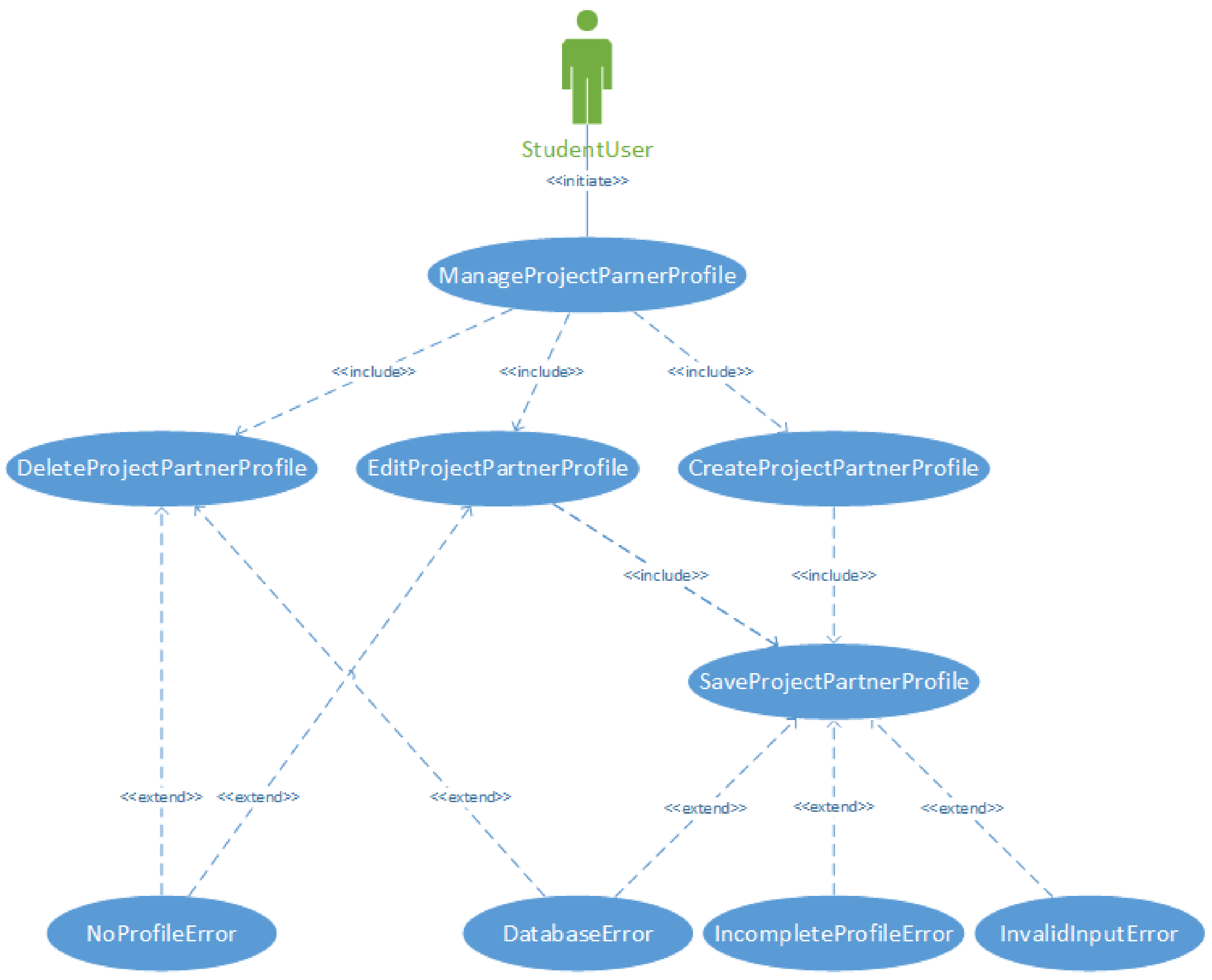

Figure 2 - StudentUser Detailed Use Case Diagram For: ManageProjectPartnerProfile (UC-01)

Table 4 - StudentUser Detailed Use Case Descriptions For: ManageProjectPartnerProfile

UC-03	**DeleteProjectPartnerProfile**	The StudentUser chooses to remove their current PPP from the system.
UC-04	**EditProjectPartnerProfile**	The StudentUser chooses to edit his/her project partner profile
UC-05	**CreateProjectPartnerProfile**	The StudentUser chooses to create a new project partner profile
UC-06	**SaveProjectPartnerProfile**	The StudentUser chooses this option to save any changes made to their PPP either via editing or creating
UC-15	**NoProfileError**	The StudentUser will trigger this error if there was an attempt to access his/her non-existent PPP.
UC-16	**IncompleteProfileError**	The StudentUser triggers this use case when the PPP is missing required information
UC-17	**InvalidInputError**	This error is triggered when the system receives an invalid input from the StudentUser or AdministratorUser.
UC-18	**DatabaseError**	This error is triggered when something wrong happens during a database transaction

Figure 3 illustrates the detailed use case diagram for the StudentUser and AdministratorUser InteractWithProjects use case. The StudentUser or AdministratorUser initiates the high-level use case InteractWithProjects (UC-02) This then includes other use cases depending on the type of user that initiated the interaction. If the User is a StudentUser, then *FetchProjectsList* use case (UC-07), *UnregisterFromProject* use case (UC-11), and *RegisterForProject* use case (UC-12) are included. In the case that the initiator is an AdministratorUser, then *FetchProjectsList* use case (UC-07), *CreateProject* use case (UC-09), *EditProjectConfigurations* use case (UC-10), and *LaunchPPIDAlgorithm* use case (UC-08) are included. Table 5 highlights the use case descriptions for the StudentUser ManageProjectPartnerProfile Use Case diagram illustrated below.

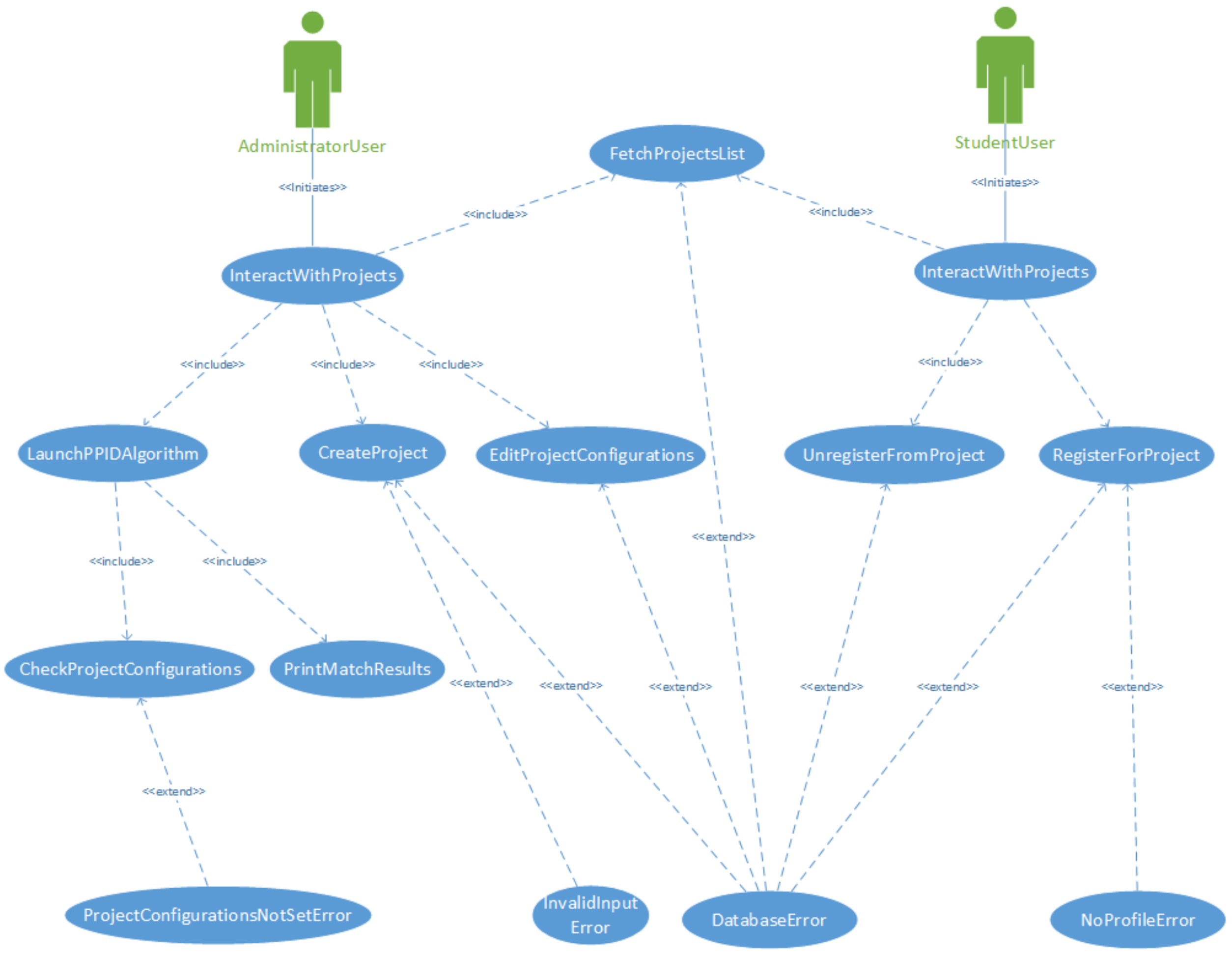

Figure 3 - StudentUser and AdministratorUser Detailed Use Case Diagram For: InteractWithProjects (UC-02)

Table 5 - StudentUser and AdministratorUser Detailed Use Case Description For: InteractWithProjects

UC-07	**FetchProjectsList**	The StudentUser or AdministratorUser views a list of projects available to him/her.
UC-08	**LaunchPPIDAlgorithm**	The AdministratorUser launches the algorithm that groups all the students into teams for a specific project
UC-09	**CreateProject**	The AdministratorUser chooses to create a new project.
UC-10	**EditProjectConfigurations**	The AdministratorUser chooses to edit the configurations of a particular project he/she created.
UC-11	**UnRegisterFromProject**	The StudentUser chooses to unregister himself/herself from a particular project.
UC-12	**RegisterForProject**	The StudentUser chooses to register himself/herself for a particular project.
UC-13	**CheckProjectConfigurations**	The system checks to see that the configurations for the project are set.
UC-14	**PrintMatchResults**	Once the PPID algorithm is launched, the results of the algorithm are presented to the user.
UC-19	**ProjectConfigurationsNotSetError**	This Error if the PPID algorithm is launched by the AdministratorUser without setting the Configurations for the project.
UC-17	**InvalidInputError**	(Refer to Table 4 for description of Use case)
UC-18	**DatabaseError**	(Refer to Table 4 for description of Use case)
UC-15	**NoProfileError**	(Refer to Table 4 for description of Use case)

Use Case Flow of Events

UC-01 - ManageProjectPartnerProfile

The StudentUser modifies the information in his Project Partner Profile

Use Case Identifier	UC-01
Name	**ManageProjectPartnerProfile**
Participating actors	Initiated by StudentUser
Flow of events	1. The StudentUser opens the manage profile service within the application. 2. The system provides the StudentUser with three options: create PPP, edit PPP and delete PPP. 3. If the StudentUser decides to create a PPP, the system provides the StudentUser with a form for creating a new PPP (**include** use case **CreateProjectPartnerProfile**). 4. If the StudentUser decides to edit his/her profile, the system provides the StudentUser with an editable form containing the StudentUser's current PPP information. The StudentUser can then edit the fields he/she wishes to change (**include** use case **EditProjectPartnerProfile**). 5. If the StudentUser decides to delete his/her profile, the system deletes the StudentUser's PPP from the system (**include** use case **DeleteProjectPartnerProfile**).
Entry conditions	● StudentUser must be logged in to the cuPID system.
Exit conditions	● The StudentUser modifies the information in his/her PPP.
Quality requirements	● Options provided by the system must be visible to the StudentUser.
Traceability	● FR-01, FR-01-01, FR-01-02, FR-01-03 ● NFR-01, NFR-02, NFR-03, NFR-04, NFR-07, NFR-08, NFR-13, NFR-19, NFR-28, NFR-29

UC-02 - InteractWithProjects

The StudentUser or AdministratorUser interacts with available projects

Use Case Identifier	UC-02
Name	**InteractWithProjects**
Participating actors	Initiated by StudentUser or AdministratorUser
Flow of events	1. The StudentUser or AdministratorUser chooses to interact with projects available to him/her. 2. The system provides a list of projects available to the StudentUser or Administrator User (**include** use case **FetchProjectsList**). The system provides the descriptions of projects to the StudentUser or AdministratorUser along with the list. 3. The system provides the StudentUser with the option to participate in a particular project (**include** use case **RegisterForProject** and **UnregisterFromProject**). 4. The system provides the AdministratorUser with the option to create a project (**include** use case **CreateProject**) 5. The system provides the AdministratorUser with the option to edit the configurations of a project (**include** use case **EditProjectConfigurations**). 6. The system provides the AdministratorUser with the option to launch PPID algorithm for the project (**include** use case **LauchPPIDAlgorithm**)
Entry conditions	● StudentUser or AdministratorUser is logged in to cuPID.
Exit conditions	● The information concerning project(s) is modified.
Quality requirements	● All possible actions that can be initiated by the type of User for the project must be visible to the User.
Traceability	● FR-02, FR-02-01, FR-02-02, FR-02-03, FR-02-04, FR-03, FR-03-01, FR-03-02, FR-03-3, FR-03-04, FR-03-05, FR-04 ● NFR-02, NFR-03, NFR-04, NFR-05, NFR-09, NFR-10, NFR-11, NFR-13, NFR-18, NFR-29

UC-03 - DeleteProjectPartnerProfile

The StudentUser chooses to remove their current PPP from the system.

Use Case Identifier	UC-03
Name	**DeleteProjectPartnerProfile**
Participating actors	Initiated by StudentUser
Flow of events	1. The StudentUser chooses to delete his PPP. 2. The system prompts the StudentUser to confirm his actions

	3. The StudentUser confirms that he wishes to delete his PPP 4. The system deletes the user's profile and all data relating to his registered projects. 5. The system notifies user that this operation succeed.
Entry conditions	● StudentUser is currently viewing his PPP
Exit conditions	● The StudentUser will no longer have a PPP
Quality requirements	Not Applicable
Traceability	● FR-01-01, NFR-03, FR-07 ● NFR-08, NFR-09, NFR-19, NFR-28, NFR-29

UC-04 - EditProjectPartnerProfile

The StudentUser chooses to edit his/her project partner profile

Use Case Identifier	UC-04
Name	**EditProjectPartnerProfile**
Participating actors	Initiated by StudentUser
Flow of events	1. The StudentUser chooses to edit his/her PPP 2. The System provides the StudentUser with a form containing previous information of the StudentUser's PPP where he/she can modify his/her qualifications and also qualifications required of potential teammates. 3. The StudentUser selects the submits the form to save his/her changes to the PPP. 4. The system saves the StudentUser's modifications to his/her PPP (**include** use case **SaveProjectPartnerProfile**).
Entry conditions	● StudentUser is currently viewing his PPP
Exit conditions	● The PPP of the StudentUser is updated.
Quality requirements	● If validation error occurs during the save process, the error is displayed to the StudentUser .
Traceability	● FR-01-01, FR-01-02, FR-01-03 ● NFR-01, NFR-02, NFR-03, NFR-04, NFR-06, NFR-07, NFR-09, NFR-13, NFR-19, NFR-27, NFR-28, NFR-29

UC-05 - CreateProjectPartnerProfile

The StudentUser chooses to create a new project partner profile

Use Case Identifier	UC-05
Name	**CreateProjectPartnerProfile**
Participating actors	Initiated by StudentUser

Flow of events	1. The StudentUser chooses the option to create a new PPP 2. The System creates a new PPP for the StudentUser and provides the StudentUser with a form where he/she enters his/her qualifications and also qualifications required of potential teammates. 3. The StudentUser submits the form to save his/her changes to the PPP (**include** use case **SaveProjectPartnerProfile**).
Entry conditions	● StudentUser currently has no PPP and wishes to create one.
Exit conditions	● The StudentUser has created a new PPP.
Quality requirements	● If validation error occurs during the save process, the detailed error is displayed to the StudentUser,
Traceability	● FR-01-01, FR-01-02, FR-01-03 ● NFR-01, NFR-02, NFR-03, NFR-04, NFR-06, NFR-07, NFR-09, NFR-13, NFR-19, NFR-27, NFR-28, NFR-29

UC-06 SaveProjectPartnerProfile

The StudentUser chooses this option to save any changes made to their PPP either via editing or creating

Use Case Identifier	UC-06
Name	**SaveProjectPartnerProfile**
Participating actors	Initiated by StudentUser
Flow of events	1. The user submits the PPP form provided to him/her by the system. 2. The system checks the validity of the changes made to the PPP by the StudentUser. 3. The system saves the changes made by the StudentUser in the database.
Entry conditions	● The StudentUser has finished making changes to his/her profile and wishes to save his changes.
Exit conditions	● The changes made by StudentUser to his/her PPP is saved in the database.
Quality requirements	● StudentUser should be notified when save action is completed.
Traceability	● FR-01-01, FR-01-02, FR-01-03 ● NFR-02, NFR-03, NFR-06, NFR-07, NFR-08, NFR-09, NFR-19, NFR-28, NFR-29

UC-07 - FetchProjectList

The StudentUser or AdministratorUser views a list of projects available to him/her.

Use Case Identifier	UC-07
Name	**FetchProjectsList**
Participating actors	Initiated by StudentUser or AdministratorUser
Flow of events	1. The StudentUser or AdministratorUser chooses to view a list of projects. 2. Depending on the type of User, the system provides a different list of project to the User. For StudentUsers, the system provides a list of all the projects that have been created in the cuPID system. For AdministratorUsers, the system provides a list of all the projects created by the AdministratorUser. 3. The StudentUser or AdministratorUser can select a project from the projects presented to him by the system.
Entry conditions	● The StudentUser or AdministratorUser is logged in to cuPID.
Exit conditions	● The StudentUser or AdministratorUser is provided with a list of projects available to him/her.
Quality requirements	● A long list of projects should be paginated
Traceability	● FR-02-01, FR-03-01 ● NFR-02, NFR-03, NFR-04, NFR-05, NFR-08, NFR-09, NFR-13, NFR-29

UC-08 - LaunchPPIDAlgorithm

The AdministratorUser launches the algorithm that groups all the students into teams for a specific project

Use Case Identifier	UC-08
Name	**LaunchPPIDAlgorithm**
Participating actors	Initiated by AdministratorUser
Flow of events	1. The AdministratorUser selects the option to launch the PPID algorithm on the current project.

	2. The system checks that the configurations of the project have been set (**include** use case **CheckProjectConfigurations**). 3. The system presents the results of the PPID matching algorithm to the AdministratorUser (**inlcude** use case **PrintMatchResults**). 4. The AdministratorUser can print the results of the cuPID matching algorithm.
Entry conditions	● The AdministratorUser is currently viewing a project containing StudentUsers currently registered.
Exit conditions	● The PPID algorithm is launched to create teams for the project with specified configurations and results of the algorithm are presented to the user.
Quality requirements	● The AdministratorUser should be informed when algorithm has completed.
Traceability	● FR-03-05 ● NFR-03, NFR-02, NFR-05, NFR-08, NFR-10, NFR-11, NFR-13, NFR-18

UC-09 - CreateProject

The AdministratorUser chooses to create a new project.

Use Case Identifier	UC-09
Name	**CreateProject**
Participating actors	Initiated by AdministratorUser
Flow of events	1. AdministratorUser selects option to create a new project. 2. The system creates a new project and presents the AdministratorUser with a form where he enters the description of the project and can also set the configurations of the project. 3. The AdministratorUser fill in the information for the project. 4. The system saves the newly created project in the database.
Entry conditions	● AdministratorUser is logged in to cuPID.
Exit conditions	● A new project is created with the AdministratorUser registered as the creator
Quality requirements	Not Applicable

Traceability	● FR-03-03 ● NFR-01, NFR-02, NFR-04, NFR-05, NFR-06, NFR-08, NFR-09, NFR-13, NFR-29

UC-10 - EditProjectConfigurations

The AdministratorUser chooses to edit the configurations of a particular project he/she created.

Use Case Identifier	UC-10
Name	**EditProjectConfigurations**
Participating actors	Initiated by AdministratorUser
Flow of events	1. The AdministratorUser selects the option the edit the configuration of a particular project 2. The system provides the user with all the current configurations of the project. 3. The AdministratorUser changes the configurations of the project. 4. The system saves the new project configurations made by the AdministratorUser.
Entry conditions	● AdministratorUser is currently viewing a project which he/she created
Exit conditions	● The configurations of the project have been modified.
Quality requirements	Not Applicable
Traceability	● FR-03-04 ● NFR-01, NFR-02, NFR-04, NFR-05, NFR-06, NFR-08, NFR-09, NFR-13, NFR-29

UC-11 - UnRegisterFromProject

The StudentUser chooses to unregister himself/herself from a particular project

Use Case Identifier	UC-11
Name	**UnregisterFromProject**
Participating actors	Initiated by StudentUser
Flow of events	1. The studentUser selects the option to unregister himself from

	the project 2. The system removes the StudentUser from the list of StudentUsers registered in the project.
Entry conditions	● StudentUser is viewing a project that he is currently registered in.
Exit conditions	● StudentUser is now removed from the list of StudentUsers registered for the project
Quality requirements	● The project is unflagged and is returned back to visual state to inform user that he is not registered for the project.
Traceability	● FR-02-04 ● NFR-03, NFR-05, NFR-08, NFR-09, NFR-29

UC-12 - RegisterForProject

The StudentUser chooses to register himself/herself for a particular project.

Use Case Identifier	UC-12
Name	**RegisterForProject**
Participating actors	Initiated by StudentUser
Flow of events	1. The StudentUser selects the option to register himself for the project 2. The system checks that the studentUser has a valid PPP. 3. The system adds the StudentUser to the list of StudentUsers registered for the project.
Entry conditions	● StudentUser is viewing a project that he is not currently registered in.
Exit conditions	● StudentUser is now included in list of StudentUsers registered for the project
Quality requirements	The project is flagged visually to let StudentUser know that he is registered in the project
Traceability	● FR-02-04 ● NFR-01, NFR-02, NFR-03, NFR-04, NFR-05, NFR-08, NFR-09, NFR-29

UC-13 - CheckProjectConfigurations

The system checks to see that the configurations for the project are set.

Use Case Identifier	UC-13
Name	**CheckProjectConfigurations**
Participating actors	Initiated by AdministratorUser
Flow of events	1. The system checks that the configurations for the project required by the cuPID algorithm have been properly set.
Entry conditions	● The AdministratorUser has selected the option to create teams for a particular project.
Exit conditions	● The configuration of the current project have been verified to be valid.
Quality requirements	Not Applicable
Traceability	● FR 03-04 ● NFR-03, NFR-08, NFR-09, NFR-13

UC-14 - PrintMatchResults

Once the PPID algorithm is launched, the results of the algorithm are presented to the user.

Use Case Identifier	UC-14
Name	**PrintMatchResults**
Participating actors	Initiated by AdministratorUser
Flow of events	1. The system retrieves the results of the PPID matching algorithm on completion. 2. The system presents the AdministratorUser with a match summary of the algorithm along with a detailed summary of the match results.
Entry conditions	● The AdministratorUser has selected the option to launch the PPID matching algorithm on the current project.
Exit conditions	● The AdministratorUser is presented with a detailed summary of the match results by the algorithm.

Quality requirements	● The Summary of the match results must be presented in a well formatted manner to the AdministratorUser.
Traceability	● FR-04 ● NFR-03, NFR-05, NFR-08, NFR-09, NFR-13, NFR-18, NFR-29

UC-15 - NoProfileError

The StudentUser will trigger this error if there was an attempt to access his/her non-existent PPP.

Use Case Identifier	UC-15
Name	**NoProfileError**
Participating actors	Initiated by StudentUser
Flow of events	1. The system notifies the StudentUser that his profile is not created and he is provided with the option to create his/her PPP 2. The user acknowledges his profile is incomplete
Entry conditions	● The StudentUser initiates an action that required a valid PPP
Exit conditions	● The StudentUser has been notified that his profile is non-existent.
Quality requirements	● The StudentUser must be provided with a descriptive error message and possible solutions to the error should also be provided.
Traceability	● FR-01-01 ● NFR-03, NFR-08, NFR-09

UC-16 - IncompleteProfileError

The StudentUser triggers this use case when the PPP is missing required information

Use Case Identifier	UC-16
Name	**IncompleteProfileError**
Participating actors	Initiated by StudentUser
Flow of events	1. The system returns a list of fields to the user in order to notify them of the fields that are missing 2. The StudentUser receives the notification and is prompted to fix his errors.
Entry conditions	● The StudentUser has made changes to his profile and selects the option to save his changes.
Exit conditions	● The StudentUser is notified that his PPP is incomplete.

Quality requirements	● The StudentUser must be provided with a descriptive error message and possible solutions to the error should also be provided.
Traceability	● FR-01-01 ● NFR-03, NFR-06, NFR-08

UC-17 - InvalidInputError

This error is triggered when the system receives an invalid input from the StudentUser or AdministratorUser

Use Case Identifier	UC-17
Name	**InvalidInputError**
Participating actors	Initiated by StudentUser or AdministratorUser
Flow of events	1. The system inspects each field of the form provided by the User for any inputs that are not in the expected format 2. The system returns a list of all fields that it has found with this error to notify the User 3. The User receives the notification and is prompted to fix his errors
Entry conditions	● The StudentUser or AdministratorUser is currently modifying a form and chooses to save the information entered in the form.
Exit conditions	● The StudentUser or AdministratorUser is notified that an invalid input was found in the submitted form.
Quality requirements	● The StudentUser or AdministratorUser must be provided with a descriptive error message and possible solutions to the error should also be provided.
Traceability	● FR-01-01, FR-02-04 ● NFR-03, NFR-06, NFR-08, NFR-13

UC-18 - DatabaseError

This error is triggered when something wrong happens during a database transaction

Use Case Identifier	UC-18
Name	**DatabaseError**
Participating actors	Initiated by StudentUser or AdministratorUser
Flow of events	1. The StudentUser or AdministratorUser invoked a database call through a variety of use cases 2. The system makes a database call, but the database call ends unexpectedly. 3. The system notifies the StudentUser or AdministratorUser that an error occurred while attempting to access the database.

	4. The system notifies the StudentUser or AdministratorUser that this error occured and advises the user to try again later
Entry conditions	● The user triggered a database call
Exit conditions	● The StudentUser or AdministratorUser is notified that an error occured in the database during the interaction initiated by the user.
Quality requirements	● The StudentUser or AdministratorUser must be provided with a descriptive error message and possible solutions to the error should also be provided.
Traceability	● FR-01-01, FR-02, FR-03-01, FR-03-02, FR-03-03, FR-03-03, FR-03-04 , FR-04 ● NFR-03, NFR-06, NFR-08, NFR-09, NFR-25

UC-19 - ProjectConfigurationsNotSetError

This Error if the PPID algorithm is launched by the AdministratorUser without setting the Configurations for the project.

Use Case Identifier	UC-19
Name	**ProjectConfigurationsNotSetError**
Participating actors	Initiated by AdministratorUser
Flow of events	1. The AdministratorUser attempted to Create teams without setting the project configurations needed by the algorithm. 2. The system verifies that the project configurations have not been set and notifies the AdministratorUser to set the project configurations required by the PPID algorithm.
Entry conditions	● The AdministratorUser attempts to launch the PPID algorithm without setting the configurations of the project.
Exit conditions	● The AdministratorUser is notified that the project configurations needed for the algorithm have not been set.
Quality requirements	● The AdministratorUser must be provided with a descriptive error message and possible solutions to the error should also be provided.
Traceability	● FR-03, FR-03-04, FR-03-05 ● NFR-03, NFR-06, NFR-08, NFR-09

2.4.2 Object Models

The purpose of this section is to provide the user with a concrete model of the entities that make up the cuPID system. The object model illustrates the system entities and how they relate with one another. To ease the understanding of this model, Figure 4 presents the reader with a UML class diagram that illustrates the user entities of the cuPID system. Figure 5 illustrates the relationships between the intrinsic entities of the cuPID system. Finally, Table 6 presents the reader with a complete data dictionary of all the entities of the cuPID system. Each entry in the data dictionary highlights the definition of the entity along with the attributes and associations attached to the specific entity. To ensure proper traceability, all Entity objects have been assigned a unique identifier, and the use cases in which they trace to are highlighted in the traceability column.

Data Dictionary

The data dictionary represented in Table 6 provides the reader with entities that make up the cuPID system. The definition of these entities along with their roles in the system are also specified in the dictionary. Finally, the attributes and associations that make up each entity are specified.

Table 6 - Data Dictionary For Entity Objects Of The cuPID system

ID	Entity Object	Attributes/ Associations	Definition	Traceability
EO-01	*User*	● firstName ● lastName	A user that accesses the cuPID system.	All Use cases UC-01 - UC-19
EO-02	*Administrator User*	● projects ● numberOfProjects Created	A user that has the ability to create projects and initiate the PPID algorithm.	UC-02, UC-07, UC-08, UC-09, UC-10,
EO-03	*StudentUser*	● PPProfile ● registeredProjects	A user that has only one Project Partner Profile used by the PPID algorithm to find a team match and a list of projects he/she is assigned to.	UC-02, UC-03, UC-04, UC-05, UC-06, UC-11, UC-12
EO-04	*Project*	● projectConfigurations ● registeredPPPs ● title ● description	An entity that contains a title, a description, a list of project configurations and a list of PPPs of the StudentUsers' who registered for the project.	UC-07, UC-08, UC-09, UC-10, UC-11, UC-12, UC-13, UC-14, UC-17, UC-18, UC-19

EO-05	*Configuration*	● parameter ● value	An entity that specifies the settings required by the PPID algorithm.	UC-08, UC-09, UC-10, UC-13, UC-19
EO-06	*CupidSystem*	● PPIDAlgorithm ● projects ● profiles	The system that manages the creation of teams containing StudentUsers using the PPID algorithm initiated by an AdministratorUser. It also keeps track of all the projects and profiles currently created in the system.	All Use cases UC-01 - UC-19
EO-07	*PPIDAlgorithm*	● algorithmConfigurations	An entity that is used to create teams of StudentUsers based on the configurations it is initialized with.	UC-08, UC-13, UC-14, UC-19
EO-08	*Project* *Partner Profile*	● StudentUser ● biography ● personalQualifications ● teammateQualifications	An entity that contains a bio and two list of qualifications: the StudentUser's personal qualifications and the qualifications of a teammate that the StudentUser would like to work with.	UC-01, UC-03, UC-04, UC-05, UC-06, UC-15, UC-16, UC-18
EO-09	*Qualifications*	● parameter ● value	An entity that demonstrates capabilities of a user.	UC-01, UC-04, UC-05
EO-10	*MatchReport*	● log ● result	An entity that contains the results of launching the PPID algorithm.	UC-08, UC-14

Class Diagram

The class diagrams of the entities involved with the cuPID system are presented below. Care has been taken to separate the entities intrinsic to the cuPID system from the external entities such as Users. Figure 4 presents the UML class diagram for the entity objects that represent users of the cuPID system. Figure 5 presents the UML class diagrams for the entities involved in the cuPID system.

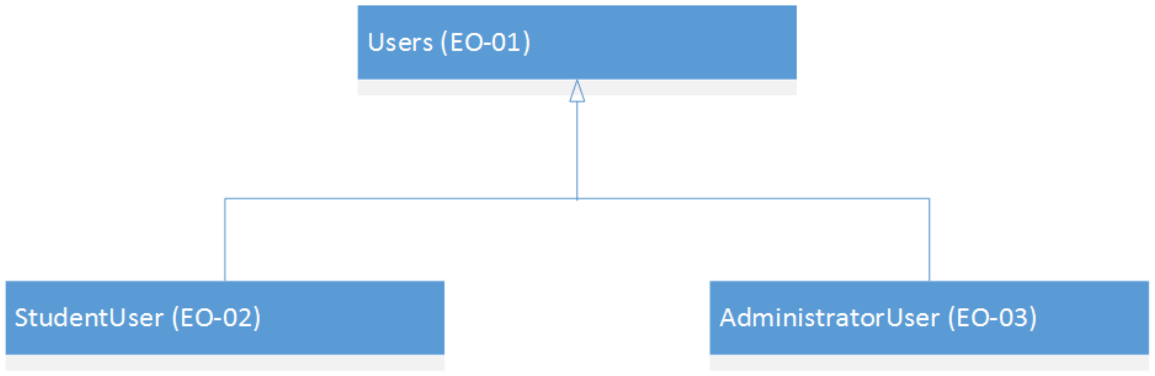

Figure 4 - UML Class Diagram of the entities representing users of cuPID system

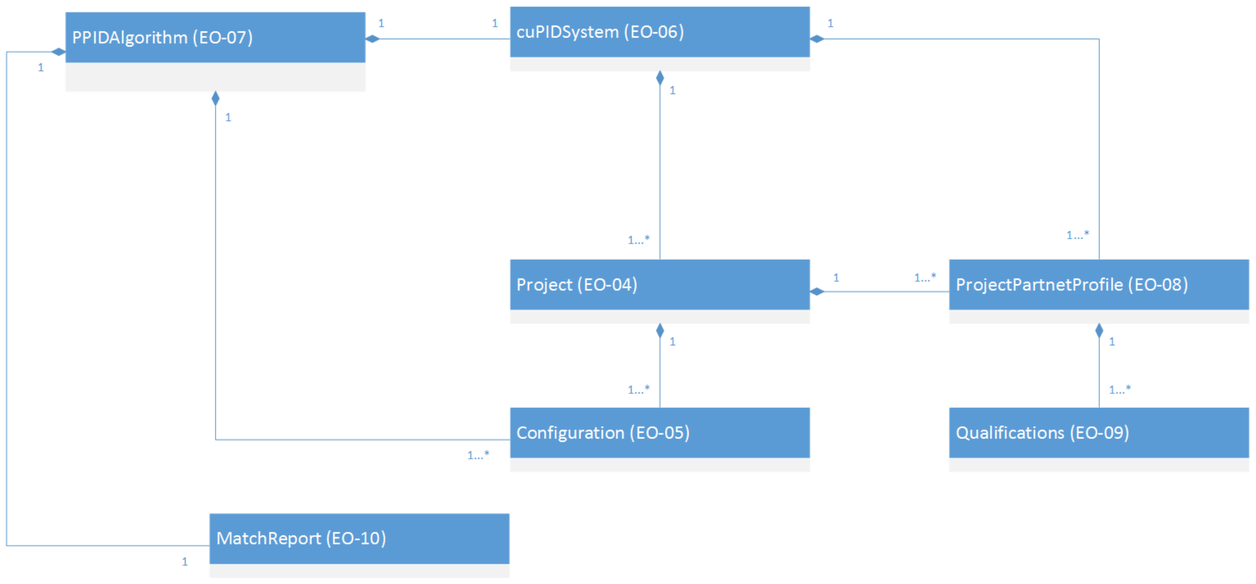

Figure 5 - UML Class Diagram of the entities involved with the cuPID system

2.4.3 Dynamic Models

In this section, this document will be elaborating on the external point-of-view of the proposed system in reaction to activities initiated by users of the system. The objective of this section is to precisely capture the behaviour of the system in terms of interactions involving actors. The approach to this presentation will involve two formats, state machines and sequence diagrams. In the presentation of state machines, each diagram is used to model the state of a given entity in the cuPID system in response to interactions with the cuPID system.

Sequence Diagrams on the other hand, represent the behavior of the system as described by the flow of events of the corresponding use case. Hence, care has been taken to relate each sequence diagram to a unique Use case that it is associated with. The diagrams depicted below interact with other objects (control, entity, and boundary) across the horizontal axis while time stretches vertically downwards.

State Machines

The state machine diagrams represent the object's state associated with an interaction between the system and an external actor (StudentUser or AdministratorUser). The possible states of the object are captured in the state machine and transition arrows depict the cause of change from one state to another. A high level state diagram of the cuPID system is provided to give a high level understanding of how the system responds to user interactions. Figure 6 illustrates the high level state diagram of the cuPID system. The state diagrams of entities that maintain state in the cuPID system are also provided along with descriptions for ease of understanding. In order to ease traceability, all UCs related to the sequence diagram have been included in their figure caption. Also each State machine diagram has been assigned a unique identifier.

SM-01 cuPIDSystem (EO-06) State Machine

This state machine diagram outlines the high level state of the cuPID system. When the system is activated, it begins *Waiting for User input*. Once it receives an interactions from the user it executes this input/interaction. The result of this interaction could cause the system to retrieve user content from the database or save the new user content to the database. On completion, the system returns back to the *Waiting for User input* state. Figure 6 illustrates the state machine diagram of the cuPID system.

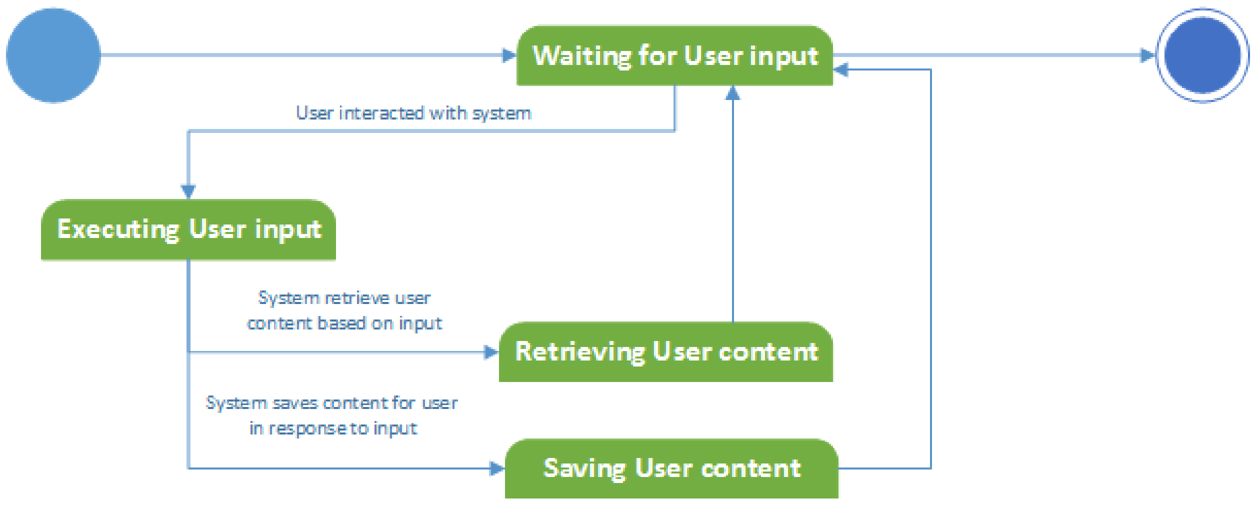
Waiting for User input
User interacted with system
Executing User input
System retrieve user content based on input
Retrieving User content
System saves content for user in response to input
Saving User content

Figure 6 - State Machine Diagram of cuPID System (UC-01 & UC-02)

SM-02 Project Entity (EO-04) State Machine

This state machine diagram outlines the state of the *Project* entity as the cuPID system responds to user interaction related to a *Project.* The default state of a project is the *Open for Registration* state. This state is a composite state as other states make it up. The *Open for Registration* state captures the states that the *Project* undergoes in response to the AdministratorUser creating or editing a *Project.* Hence, once a *Project* has been created, the *Project* is then *Open for Registration.* In this state, StudentUsers can then register into the project (as captured by the *Register Student* state), or unregister from the project (as captured by the *Unregister Student* state). In the *Open for Registration* state, an AdministratorUser can then launch the PPID algorithm on the *Project*. On launching the algorithm on the *Project*, the

Project goes into a *Locked for Team Generation* state. This is to depict that StudentUsers are not allowed to register while the algorithm is in the process of generating teams for the project. Finally, a *Project* goes into an *Inactive* state when the cuPID system is shutdown. Figure 7 illustrates the state machine diagram of the *Project* object.

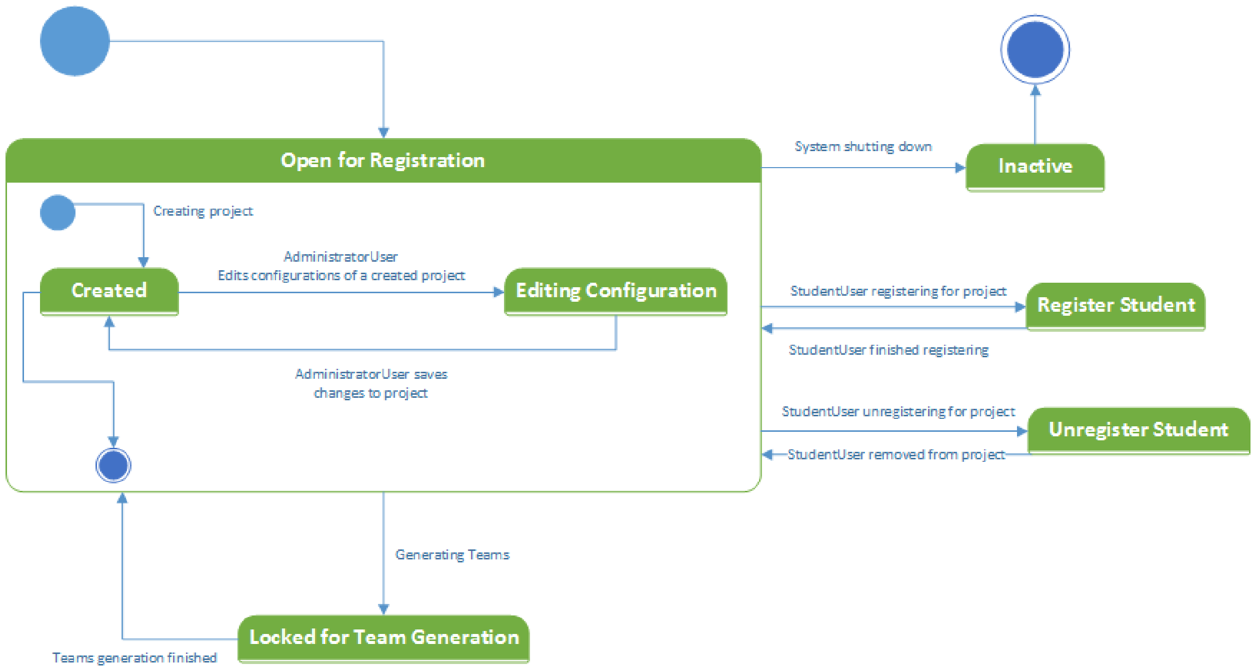
Open for Registration
Creating project
Created
AdministratorUser
Edits configurations of a created project
Editing Configuration
AdministratorUser saves
changes to project
System shutting down
Inactive
StudentUser registering for project
Register Student
StudentUser finished registering
StudentUser unregistering for project
Unregister Student
StudentUser removed from project
Generating Teams
Locked for Team Generation
Teams generation finished

Figure 7 - State Machine Diagram of Project entity (UC-02, UC-08, UC-09, UC-10, UC-11, UC-12)

SM-03 Project Partner Profile Entity (EO-08) State Machine

This state machine diagram outlines the state of the *PPP* entity as the cuPID system responds to user interaction related to a *PPP.* The default state of the *PPP* entity is *Created*. This is when the StudentUser has created his/her PPP. From this state, a StudentUser can then choose to edit his/her PPP, and then the *PPP* entity transitions into the *Editing* state. On completion of the *Editing* state, the *PPP* transitions into a *Saving* state, due to the StudentUser choosing the option to commit his/her changes to their PPP. Once changes have been saved, the *PPP* entity transitions back to the default state which is *Created.* The Final state for the *PPP* entity is the *Deleted* state, and this is caused by the StudentUser choosing the option to delete his/her PPP from the system. Figure 8 illustrates the state machine diagram of the *PPP* object.

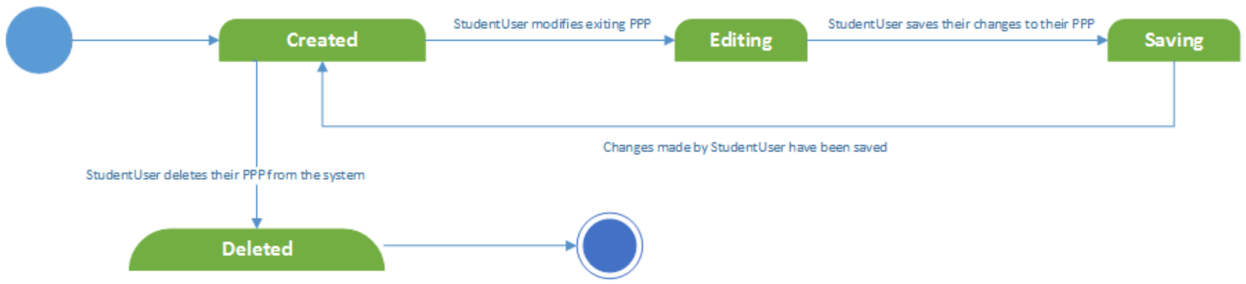
Created
StudentUser modifies exiting PPP
Editing
StudentUser saves their changes to their PPP
Saving
Changes made by StudentUser have been saved
StudentUser deletes their PPP from the system
Deleted

Figure 8 - State Machine Diagram of PPP entity (UC-01, UC-03, UC-04, UC-05, UC-06)

SM-04 PPID Algorithm Entity (EO-07) State Machine

This state machine diagram outlines the state of the *PPID Algorithm* entity as the cuPID system responds to user interaction related to a *PPID Algorithm.* The *PPID Algorithm* entity begins with the *Created* default state. When the AdministratorUser launches the Algorithm on a particular project, the *PPID Algorithm* state transitions to a *Generating Teams for Project* state. In this state, the *PPID Algorithm* loads the *Project's* configurations, matches the *PPP*s of the StudentUser's into their best possible teams and exits this state on completion. Finally, the *PPID Algorithm* transitions to a *Generate Report* state, where it's results are printed to the AdministratorUser. Figure 9 illustrates the state machine diagram of the *PPID Algorithm* object.

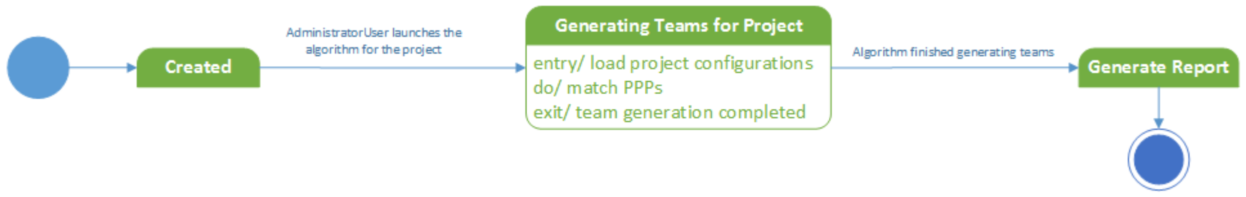
Created
AdministratorUser launches the algorithm for the project
Generating Teams for Project
entry/ load project configurations
do/ match PPPs
exit/ team generation completed
Algorithm finished generating teams
Generate Report

Figure 9 - State Machine Diagram of PPID Algorithm (UC-08, UC-13, UC-14)

Sequence Diagrams

The following diagrams may contain abstract notions and requires some background in computers to understand. The diagrams below, are modeled after the sequence of interactions among objects needed to realize the use case. Each blue text box represents an object (boundary, entity, or control) associated with each use case scenario.

SD-01 Sequence Diagram for ManageProjectPartnerProfile (UC-01)

This sequence diagram depicts the flow of events for the *ManagePPP* use case (UC-01). The StudentUser selects the option to manage his/her PPP. Since there are three options, the system provides the StudentUser with the option to: *CreatePPP* use case (UC-05), *EditPPP* use case (UC-04) or *DeletePPP* use case (UC-03). In order to promote simplicity of this sequence diagram, the included use cases (UC-03, UC-04 & UC-05) are represented as objects that are invoked in this high level use case diagram. Please refer to Figure 14 for sequence diagram for *EditPPP*, Figure 15 for sequence diagram for *CreatePPP*, and finally Figure 13 illustrates the sequence diagram for *DeletePPP*.

The Flow is as follows: The StudentUser is provided with a menu where he can choose an option to *create, edit* or *delete* his/her PPP. Depending on the option selected by the StudentUser, a different use case is invoked. Care has been taken to preserve traceability in the Sequence diagram below. The sequence diagram below illustrates these interactions.

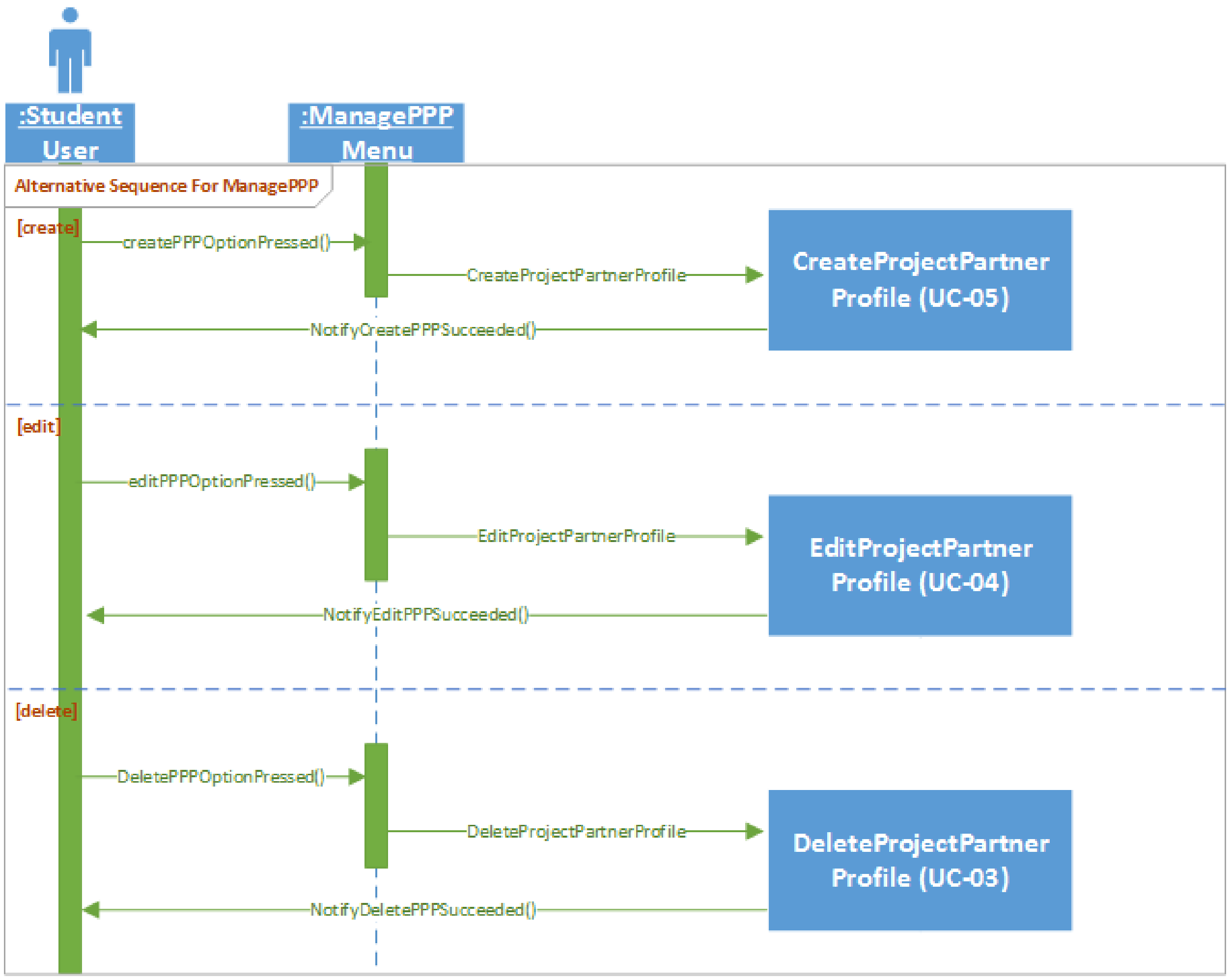
:Student User
:ManagePPP Menu
Alternative Sequence For ManagePPP
[create]
createPPPOptionPressed()
CreateProjectPartnerProfile
CreateProjectPartner Profile (UC-05)
NotifyCreatePPPSucceeded()
[edit]
editPPPOptionPressed()
EditProjectPartnerProfile
EditProjectPartner Profile (UC-04)
NotifyEditPPPSucceeded()
[delete]
DeletePPPOptionPressed()
DeleteProjectPartnerProfile
DeleteProjectPartner Profile (UC-03)
NotifyDeletePPPSucceeded()

Figure 10 - SD-01 Sequence Diagram for ManageProjectPartnerProfile (UC-01)

SD-02 Sequence Diagram for InteractWithProjects For AdministratorUser (UC-02)

In this sequence diagram is initiated by the AdministratorUser selecting the option to interact with projects. A *ProjectManagerControl* is created in order to retrieve a list of all the projects available to the AdministratorUser (Please refer to Figure 17 for the sequence diagram of FetchProjectsList (UC-07)). A ProjectListView boundary object is then presented to the AdministratorUser where he/she can then select a project he/she wishes to interact with. Once a project has been selected, the AdministratorUser now has 3 options. This is captured in the sequence diagram by an alternative sequence flow box. The sequence diagrams for the respective options are included as Object boxes in the the diagram below for ease of readability. Please refer to Figure 19 for the sequence diagram of CreateProject (UC-09), Figure 20 for the sequence diagram for EditProjectConfigurations (UC-10), and Figure 18 for the sequence diagram for LaunchPPIDAlgorithm. The sequence diagram below illustrates these interactions.

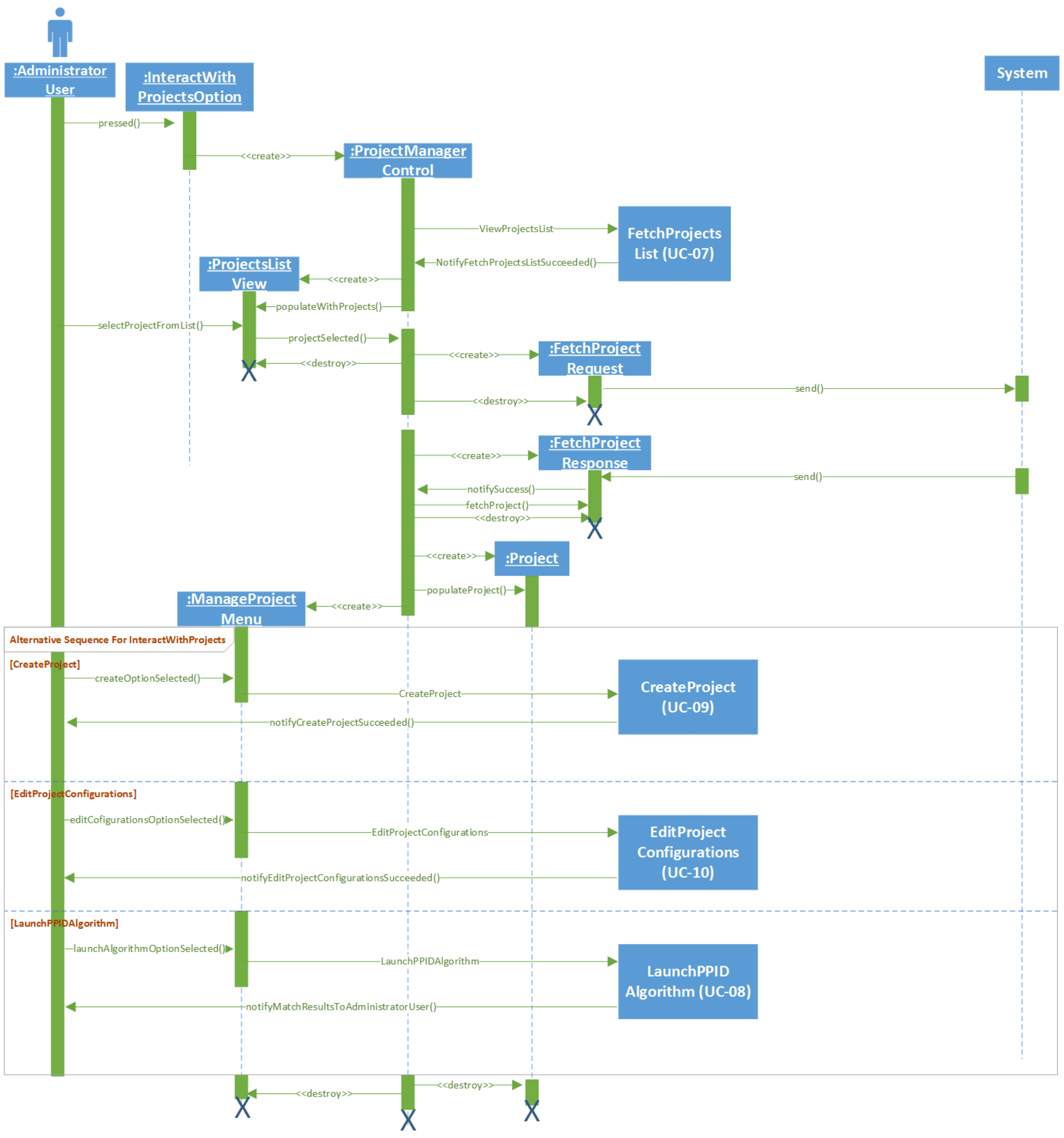

Figure 11 - SD-02 Sequence Diagram for InteractWithProjects For AdministratorUsers (UC-02)

SD-03 Sequence Diagram for InteractWithProjects For StudentUser (UC-02)

In this sequence diagram is initiated by the StudentUser selecting the option to interact with projects. A *ProjectManagerControl* is created in order to retrieve a list of all the projects available to the StudentUser (Please refer to Figure 17 for the sequence diagram of *FetchProjectsList* (UC-07)). A *ProjectListView* boundary object is then presented to the

StudentUser where he/she can then select a project he/she wishes to interact with. Once a project has been selected, the StudentUser now has 2 options. This is captured in the sequence diagram by an alternative sequence flow box. The sequence diagrams for the respective options are included as Object boxes in the the diagram below for ease of readability. Please refer to Figure 22 for the sequence diagram of *RegisterForProject* (UC-12), and Figure 21 for the sequence diagram for *UnregisterFromProject* (UC-11). The sequence diagram below illustrates these interactions. `

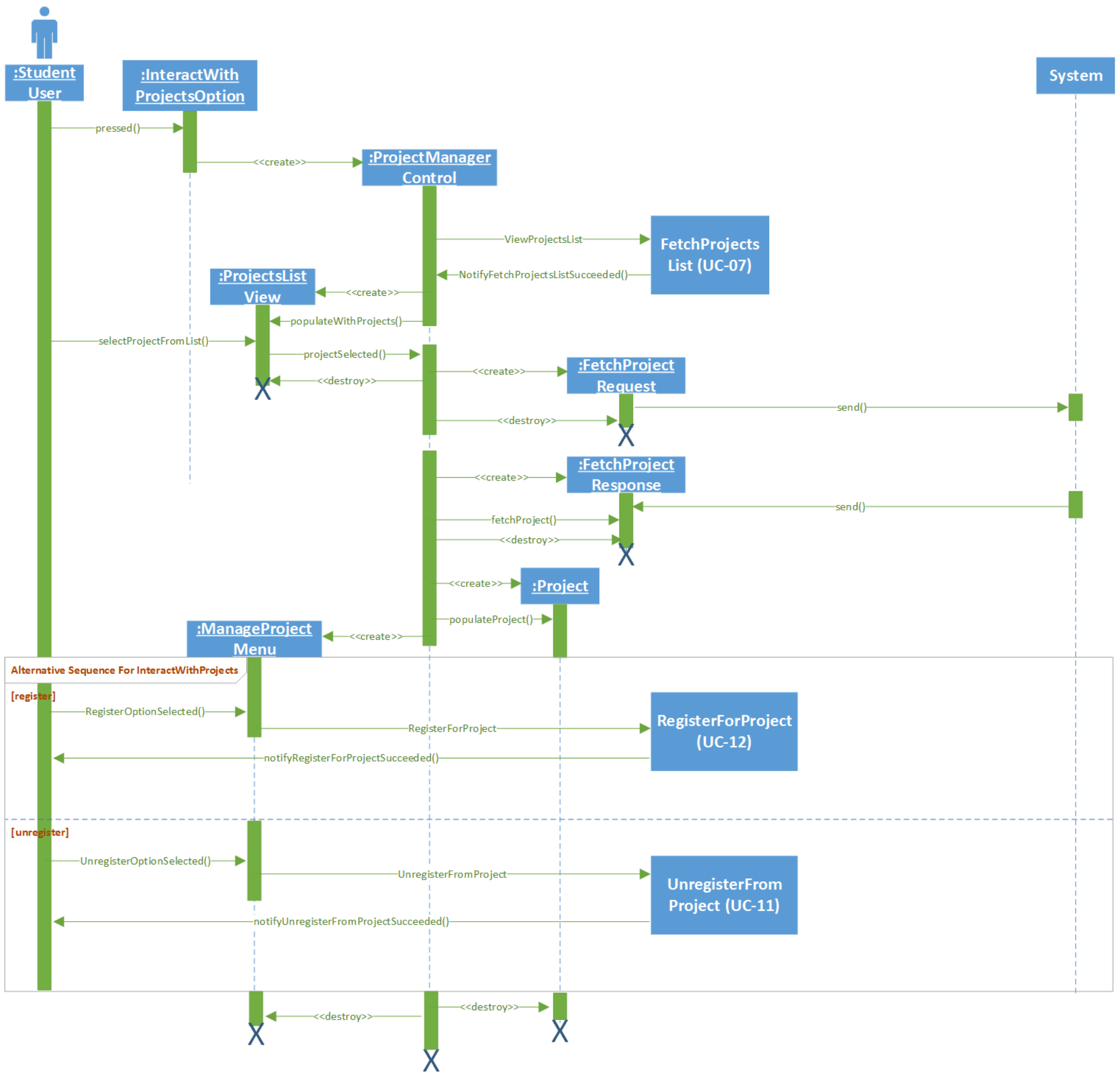

Figure 12 - SD-03 Sequence Diagram for InteractWithProjects For StudentUser (UC-02)

SD-04 Sequence diagram for DeleteProjectPartnerProfile (UC-03)

In the sequence diagram below, the StudentUser selects the option to Delete his/her existing PPP. A *DeletePPPControl* is created and a *ConfirmDeleteOption* is presented to the StudentUser to confirm his/her action to delete the PPP. On confirmation, the request to delete the PPP is created and sent to the System in order to delete the PPP from the database. On successful deletion, the system notifies the *DeletePPPControl* and the current instance of the StudentUser’s PPP is also deleted by the DeletePPPControl. The sequence diagram below illustrates these interactions.

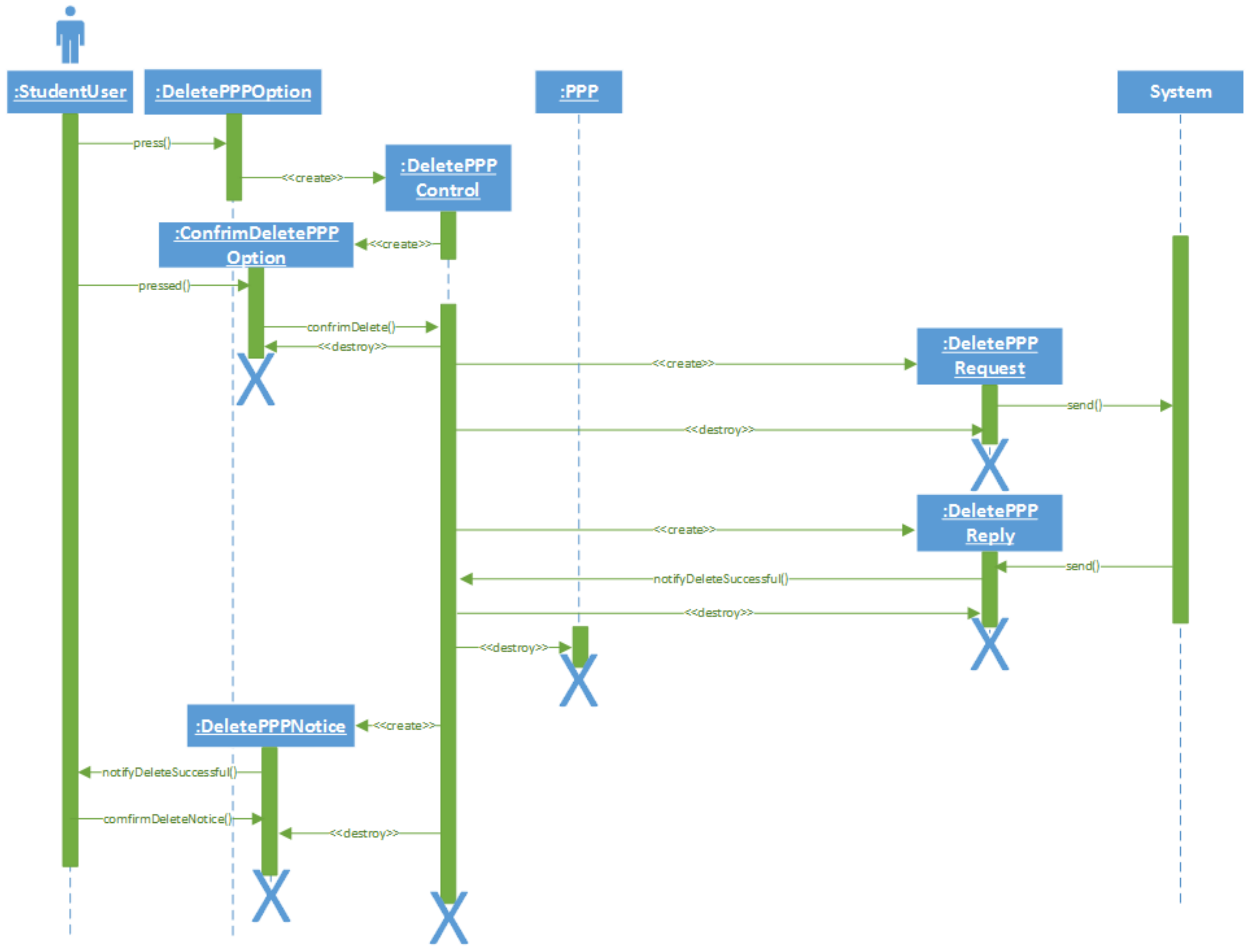

Figure 13 - SD-04 Sequence Diagram for DeleteProjectPartnerProfile (UC-03)

SD-05 Sequence Diagram for EditProjectPartnerProfile (UC-04)

In the sequence diagram below, the StudentUser selects the option to edit his PPP. An *EditPPPControl* object is created, and it sends a request to retrieve the StudentUser's PPP information from the System. On Successful retrieval of the StudentUser's PPP information, then the *EditPPPControl* creates a *PPP* entity as a proxy to represent the PPP information. An editable PPP form is presented to the StudentUser populated with his/her current PPP information. The StudentUser can then make changes to the form and submit. On submitting the form, the *EditPPPControl* invokes the *SavePPP* (UC -06) sequence diagram (SD-07) in order to save the changes made by the StudentUser. On successful save, the *EditPPPControl* is notified, and an *EditPPPSuccessNotice* is presented to the StudentUser to inform him/her that the edit operation was a success. The sequence diagram below illustrates these interactions.

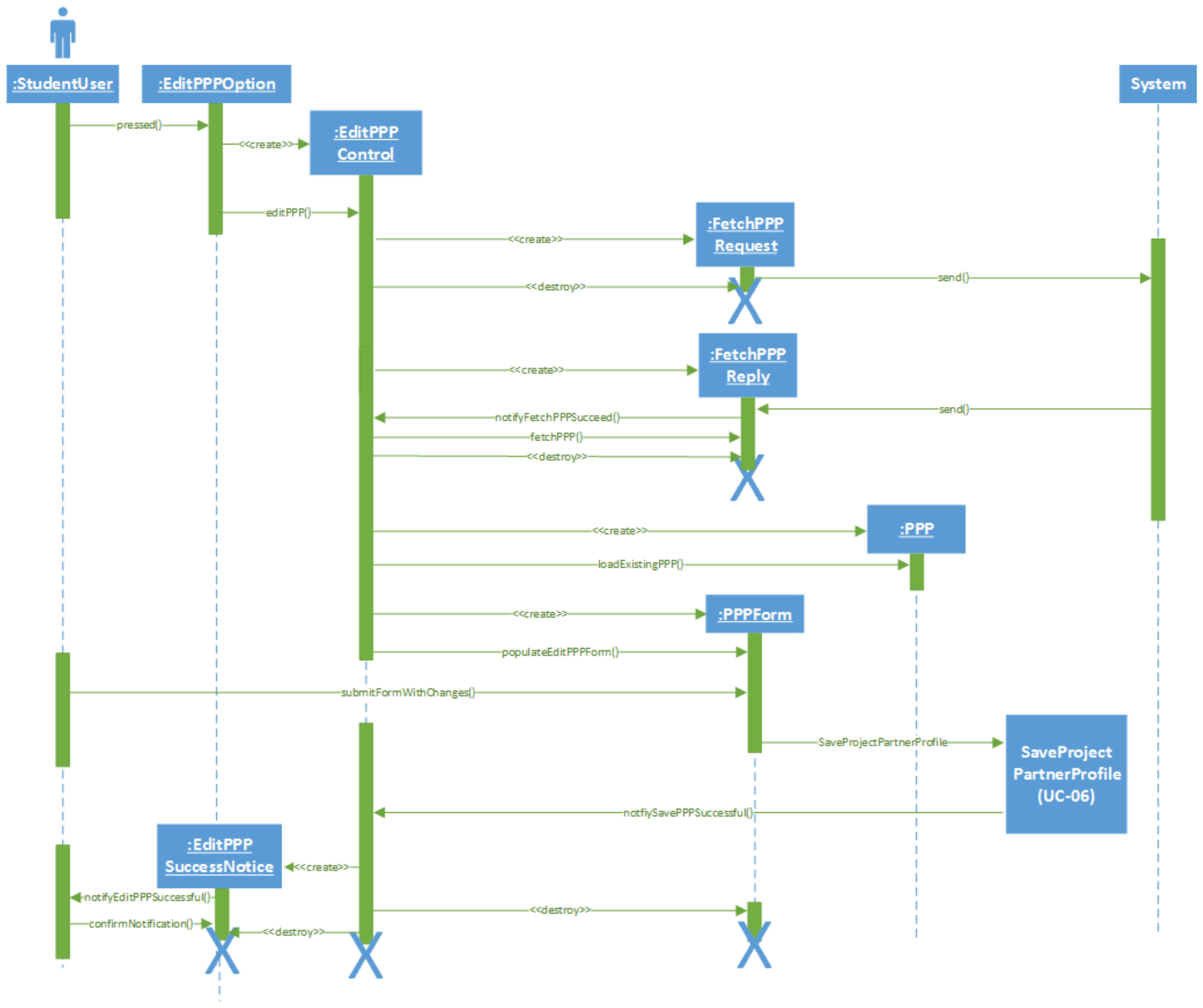

Figure 14 - SD-05 Sequence Diagram for EditProjectPartnerProfile (UC-04)

SD-06 Sequence diagram for CreateProjectPartnerProfile (UC-05)

In the sequence diagram below, the StudentUser selects the option to create a PPP. The entry condition of this UC is that the StudentUser currently doesn't have any PPP registered in the system. On selecting the *CreatePPPOption*, a *CreatePPPControl*. This then creates a new PPP proxy entity and a PPP form for the StudentUser to fill. On submitting the form, the PPP proxy entity is populated with the information from the the form and the SaveProjectPartnerProfile (Figure 16) sequence (SD-07) is invoked by the *CreatePPPControl.* On successful save, the *CreatePPPControl* is notified and the StudentUser is notified that his/her CreatePPP operation was successful. The sequence diagram below illustrates these interactions.

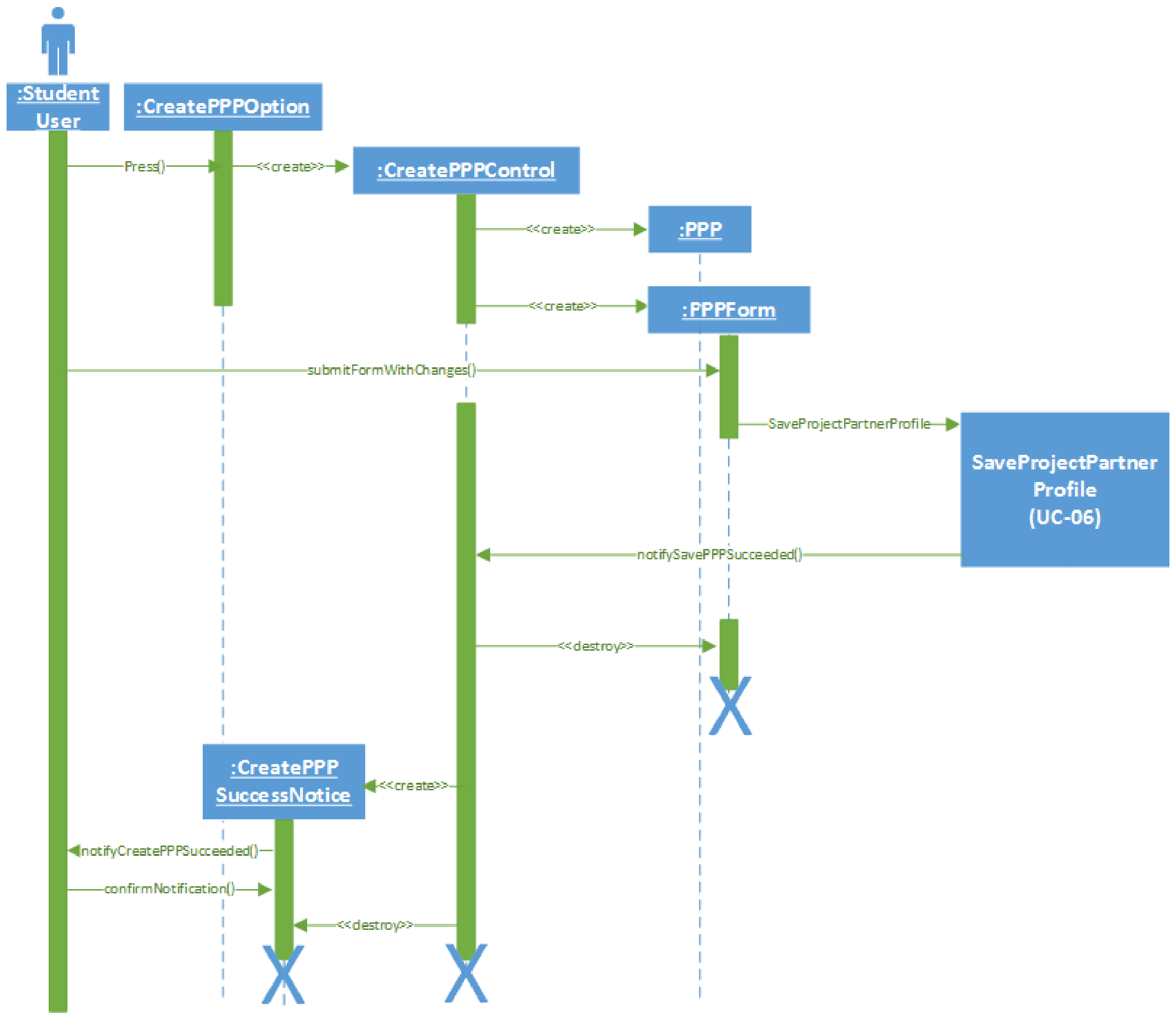

Figure 15 - SD-06 Sequence Diagram for CreateProjectPartnerProfile (UC-05)

SD-07 Sequence diagram for use case SaveProjectPartnerProfile (UC-06)

In the sequence diagram below, the StudentUser selects the option to save his/her PPP after making changes. A *SavePPPControl* object is created, and subsequently, a *FormValidatorControl* is created. The *FormValidatorControl* validates the updated PPP information from *EditPPP* (UC-04) or *CreatePPP* (UC-05) for any invalid formats or missing fields. On success, *FormValidatorControl* returns a success notice to the *SavePPPControl.* The *SavePPPControl* then initiates a *SavePPPRequest* to the system. . The sequence diagram below illustrates these interactions.

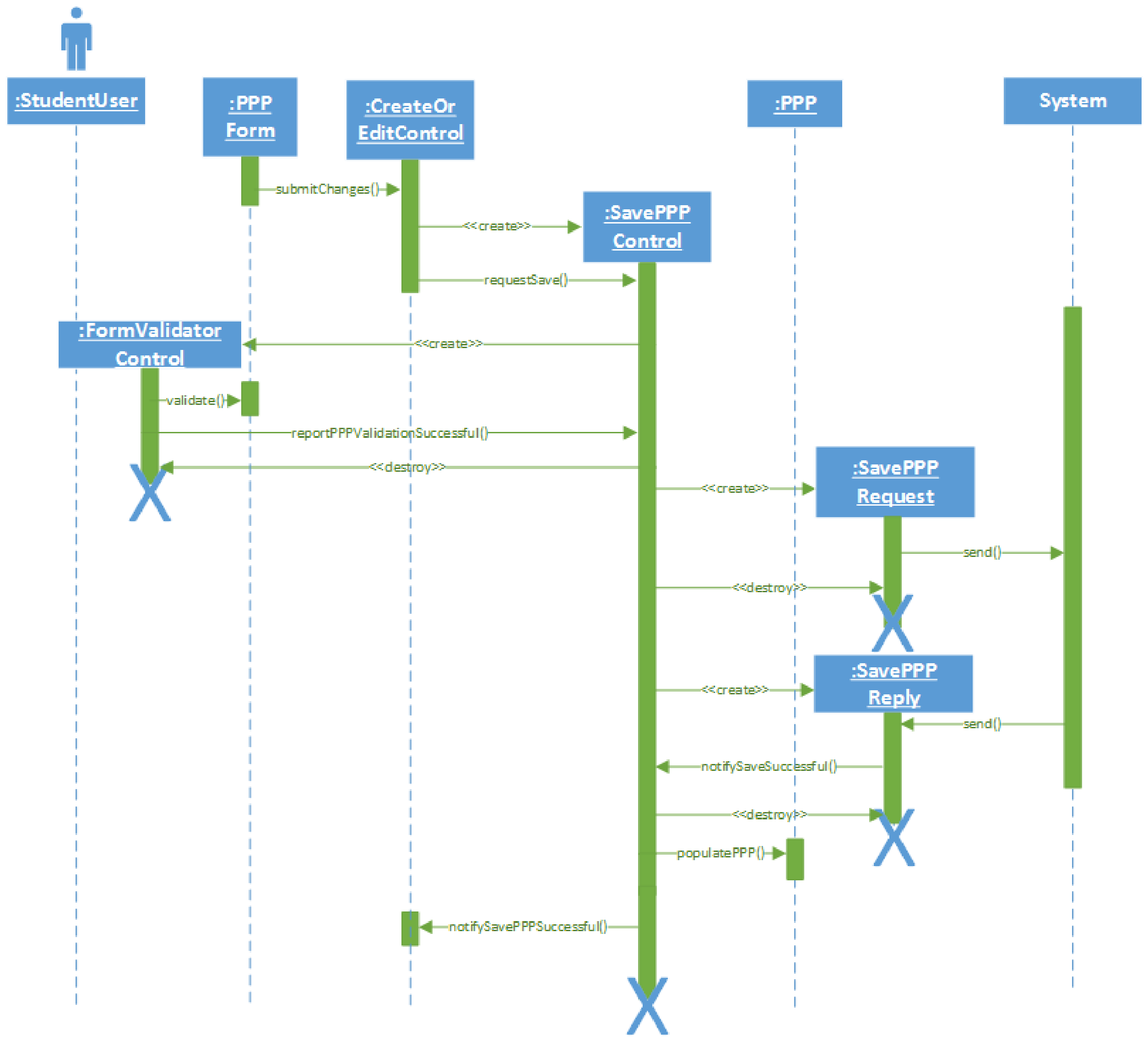

Figure 16 - SD-07 Sequence Diagram For SaveProjectPartnerProfile (UC-06)

SD-08 Sequence diagram for FetchProjectsList (UC-07)

The sequence diagram below depicts the flow of events when a StudentUser or an AdministratorUser chooses the option to interact with projects (UC-02). The *ProjectManagerControl* in the sequence diagram (SD-02 & SD-03) for InteractWithProjects (UC-02) Figure (11 or 12) invokes this *FetchProjectsList* sequence which determines the project available for viewing for both types of users. Depending on the type of user, the *FetchProjectsList* retrieves a different list of projects to the *ProjectManagerControl*. If the User is an AdministratorUser, a list of projects that he/she created is returned to him/her. If the User is a StudentUser, then the list of all the created projects in the cuPID system is returned to him/her.

Based on the fact that this sequence flow is included by the InteractWithProjects (UC-02) sequence, note that the StudentUser/AdministratorUser is not active in this sequence. The sequence diagram below illustrates these interactions.

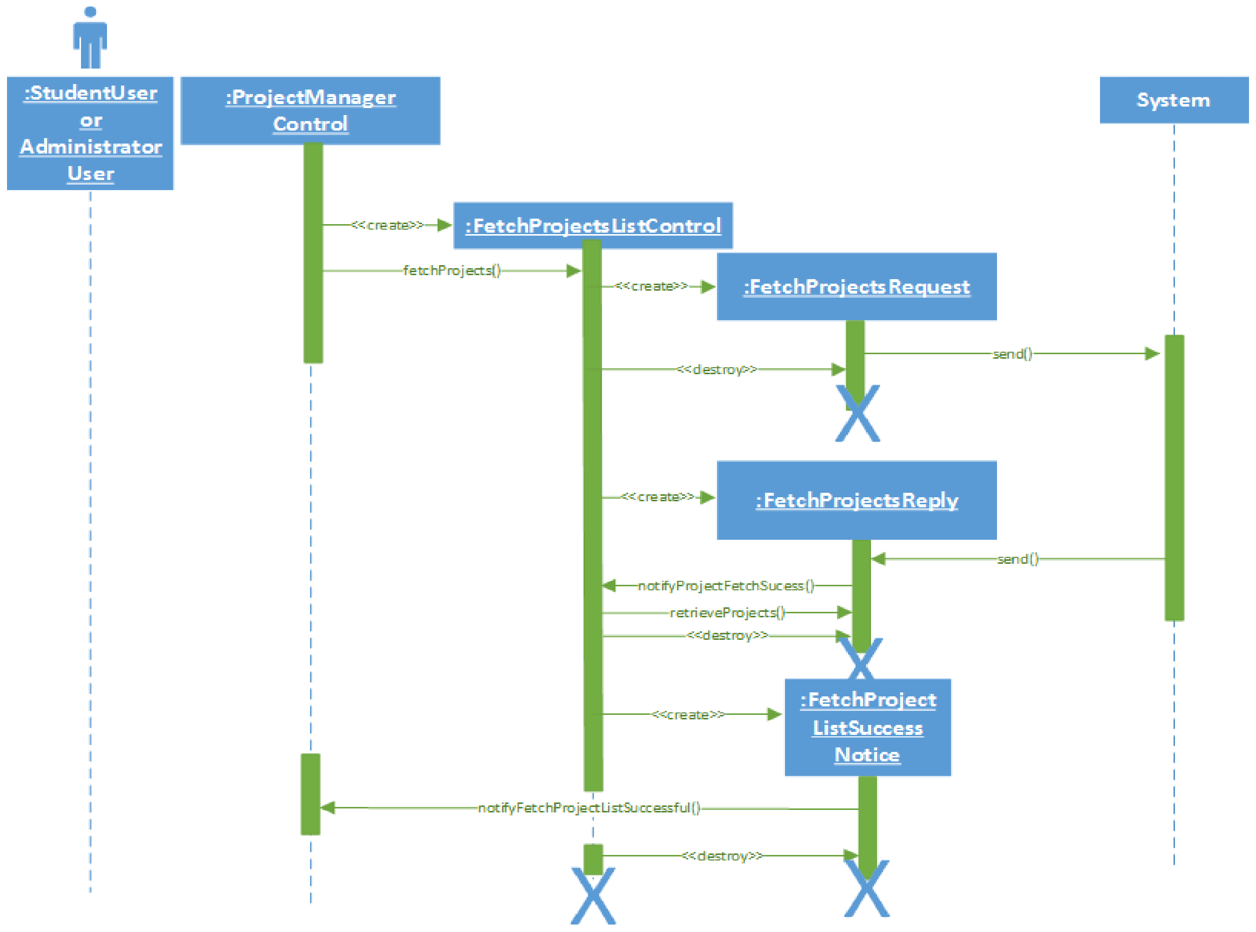

Figure 17 - SD-08 Sequence Diagram For FetchProjectsList (UC-07)

SD-09 Sequence diagram for LaunchPPIDAlgorithm (UC-08)

In the sequence diagram below, the AdministratorUser selected the option to start the PPID algorithm. A *LaunchPPPAlgorithmControl* is created to facilitate the matching algorithm's procedure. This control first retrieves the project configurations from the *Project* entity, then invokes the CheckProjectConfigurations (UC-13) sequence diagram (SD-14) to validate them. On successful validation, the *LaunchPPPAlgorithmControl* proceeds to get the registered valid PPP entities from *Project* and creates a *PPIDAlgorithm* entity to load the profiles in. When the algorithm for matching results the raw results, *LaunchPPPAlgorithmControl* invokes PrintMatchResults (UC-14) sequence diagram (SD-15) to format the summary and detailed report. A boundary view object is returned from *PrintMatchResult*, which is subsequently displayed to the AdministratorUser for viewing. The sequence diagram below illustrates these interactions.

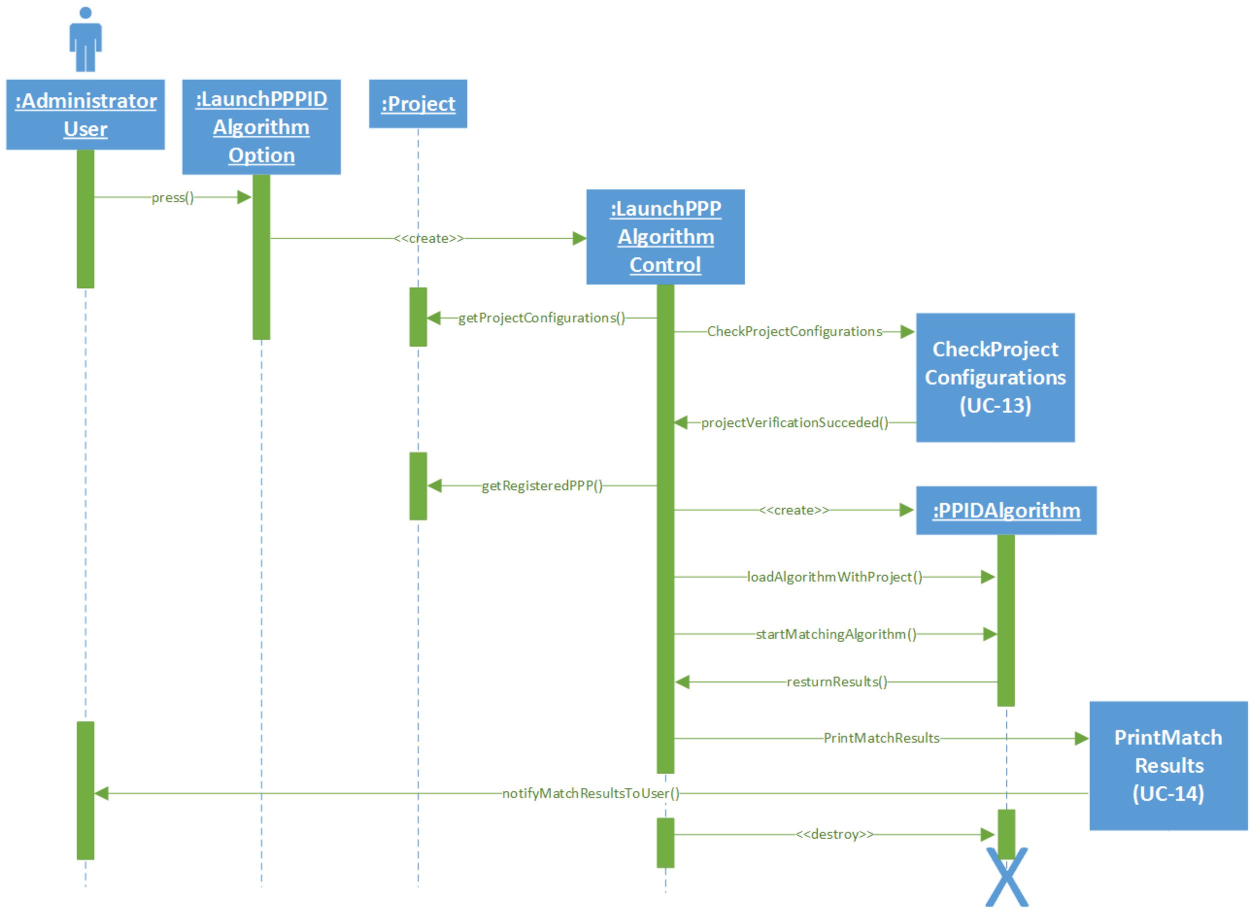
:Administrator User
:LaunchPPPID Algorithm Option
:Project
:LaunchPPP Algorithm Control
press()
<<create>>
getProjectConfigurations()
CheckProjectConfigurations
CheckProject Configurations (UC-13)
projectVerificationSucceded()
getRegisteredPPP()
<<create>>
:PPIDAlgorithm
loadAlgorithmWithProject()
startMatchingAlgorithm()
resturnResults()
PrintMatchResults
PrintMatch Results (UC-14)
notifyMatchResultsToUser()
<<destroy>>
X

Figure 18 - SD-09 Sequence Diagram for LaunchPPIDAlgorithm (UC-08)

SD-10 Sequence diagram for CreateProject (UC-09)

In the sequence diagram below, the AdministratorUser chooses the option to create a new project. A *CreateProjectControl* is created and it prepares the new *Project* entity and its configurations through the *ProjectForm*. When the AdministratorUser enters in all the project configurations and properties of the *Project*, the *ProjectForm* is run through a *ProjectFormValidator* to find and catch any invalid inputs. On success, the new project configurations from the *ProjectForm* are populated into the *Project* entity, then the *CreateProjectControl* creates a *SaveProjectRequest* object to save the project information to the system. On a successful save operation, *CreateProjectControl* creates a *CreateProjectSuccessNotice* and displays it to the AdministratorUser. The sequence diagram below illustrates these interactions.

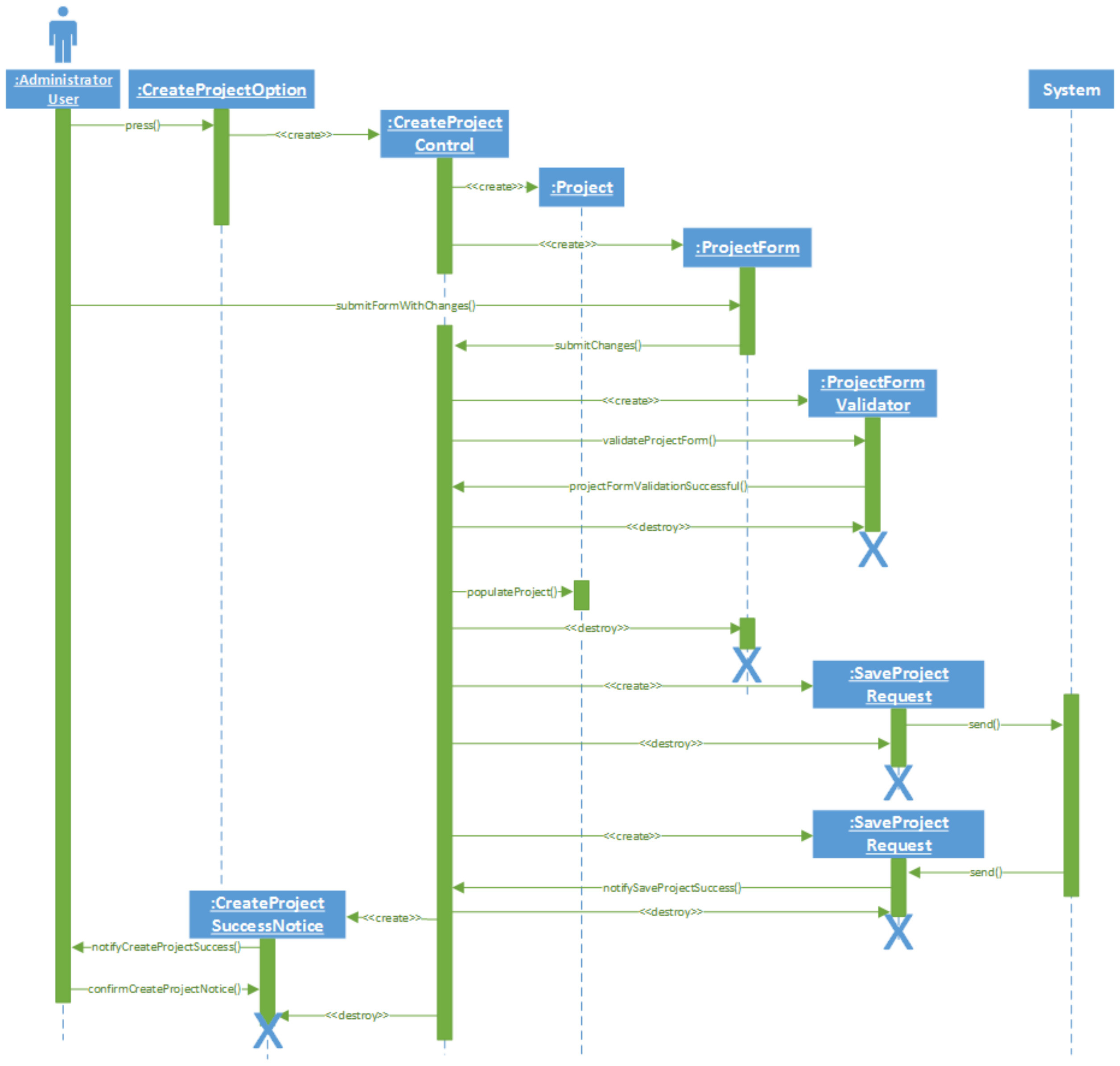

Figure 19 - SD-10 Sequence Diagram For CreateProject (UC-09)

SD-11 Sequence diagram for EditProjectConfigurations (UC-10)

In the sequence diagram below, an AdministratorUser chooses the option to edit project configurations. An *EditProjectConfigurationsControl* is created to facilitate this process. The existing *Project* entity that was opened by *ProjectManagerControl* in SD-02 Figure 11, is retrieved and loaded into the *EditProjectConfigurationForm*. The AdministratorUser then can make changes to the configurations of that particular project and submits the changes back to the control. These changes are then realized on the *Project* entity object, then a *SaveProjectRequest* is created to save the entity's state. On a successful save, the *EditProjectConfigurationsControl* creates a *EditProjectConfigurationSuccessNotice* to notify the AdministratorUser that his configurations changes has been successfully saved. The sequence diagram below illustrates these interactions.

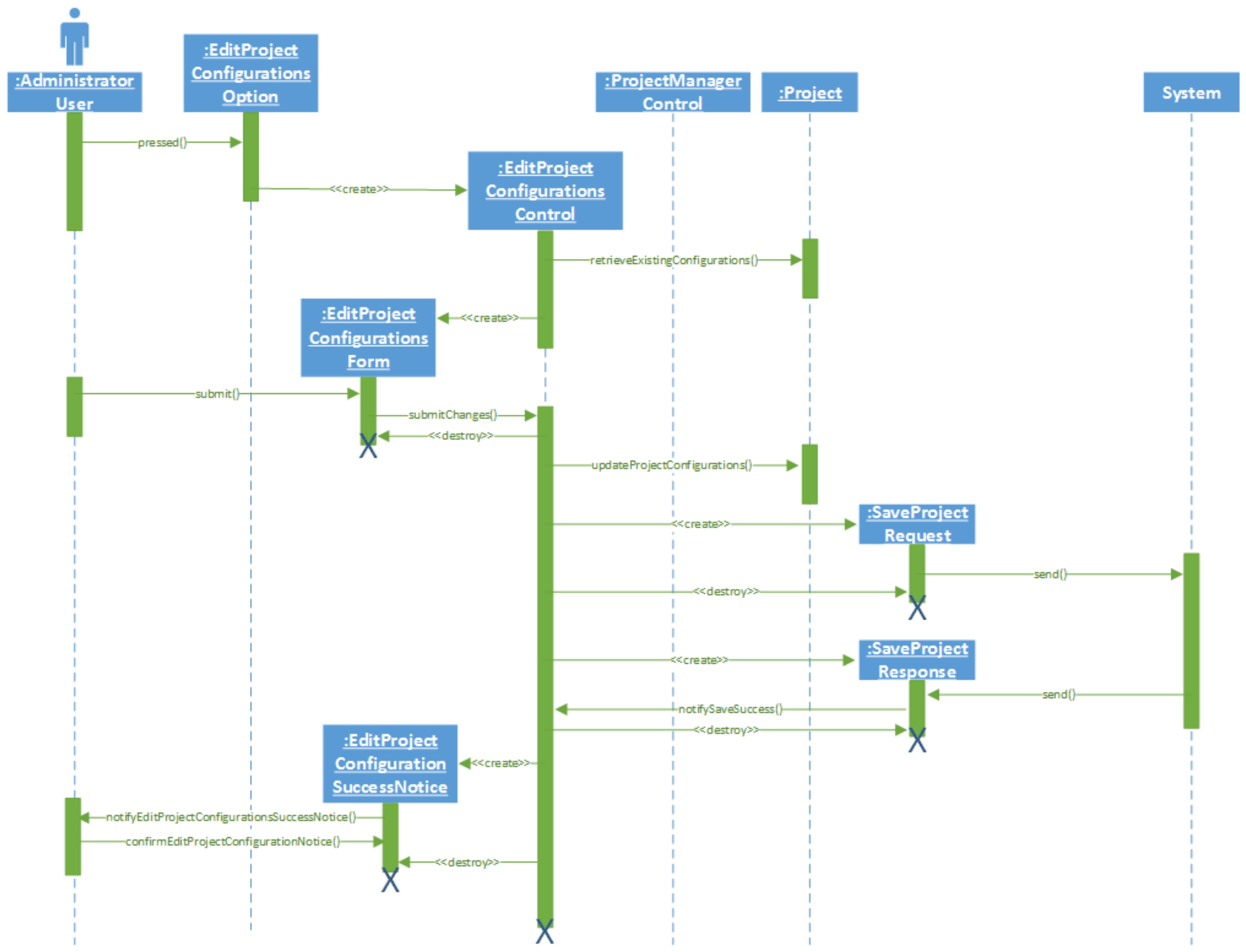
:Administrator User
:EditProject Configurations Option
:ProjectManager Control
:Project
System
pressed()
<<create>>
:EditProject Configurations Control
retrieveExistingConfigurations()
<<create>>
:EditProject Configurations Form
submit()
submitChanges()
<<destroy>>
updateProjectConfigurations()
<<create>>
:SaveProject Request
send()
<<destroy>>
<<create>>
:SaveProject Response
send()
notifySaveSuccess()
<<destroy>>
:EditProject Configuration SuccessNotice
<<create>>
notifyEditProjectConfigurationsSuccessNotice()
confirmEditProjectConfigurationNotice()
<<destroy>>

Figure 20 - SD-11 Sequence Diagram For EditProjectConfigurations (UC-10)

SD-12 Sequence diagram for UnregisterFromProject (UC-11)

In the sequence diagram below, the StudentUser selects the option to remove himself/herself from a project. The *UnRegisterProjectControl* is created, it creates a request to the system in order to retrieve the ProjectPartnerProfile. On successful retrieval, the *UnRegisterProjectControl* then removes the ProjectPartnerProfile from the Project entity and sends a notification to the user. The sequence diagram below illustrates these interactions.

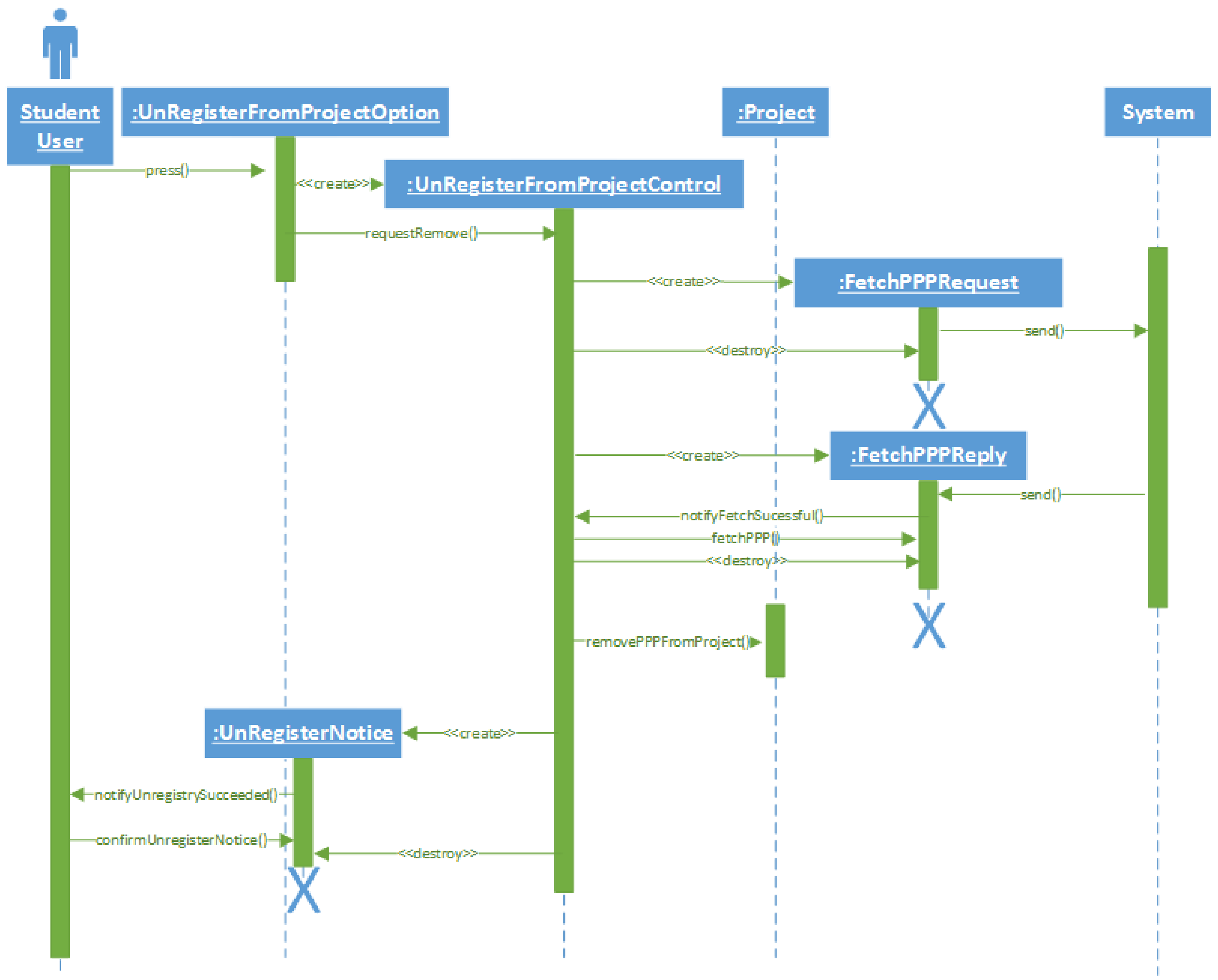

Figure 21 - SD-12 Sequence Diagram For UnregisterFromProject (UC-11)

SD-13 Sequence diagram for RegisterForProject (UC-12)

In the sequence diagram below, a StudentUser is registering for a project. A *RegisterForProjectControl* is created to facilitate the registration project. Subsequently, a fetch request to retrieve the StudentUser's PPP is activated. On successful system reply, the PPP is retrieved then registered into the *Project* entity. The *RegisterForProjectControl* then creates a *RegisterNotice* to notify the StudentUser that he has register into the project. The sequence diagram below illustrates these interactions.

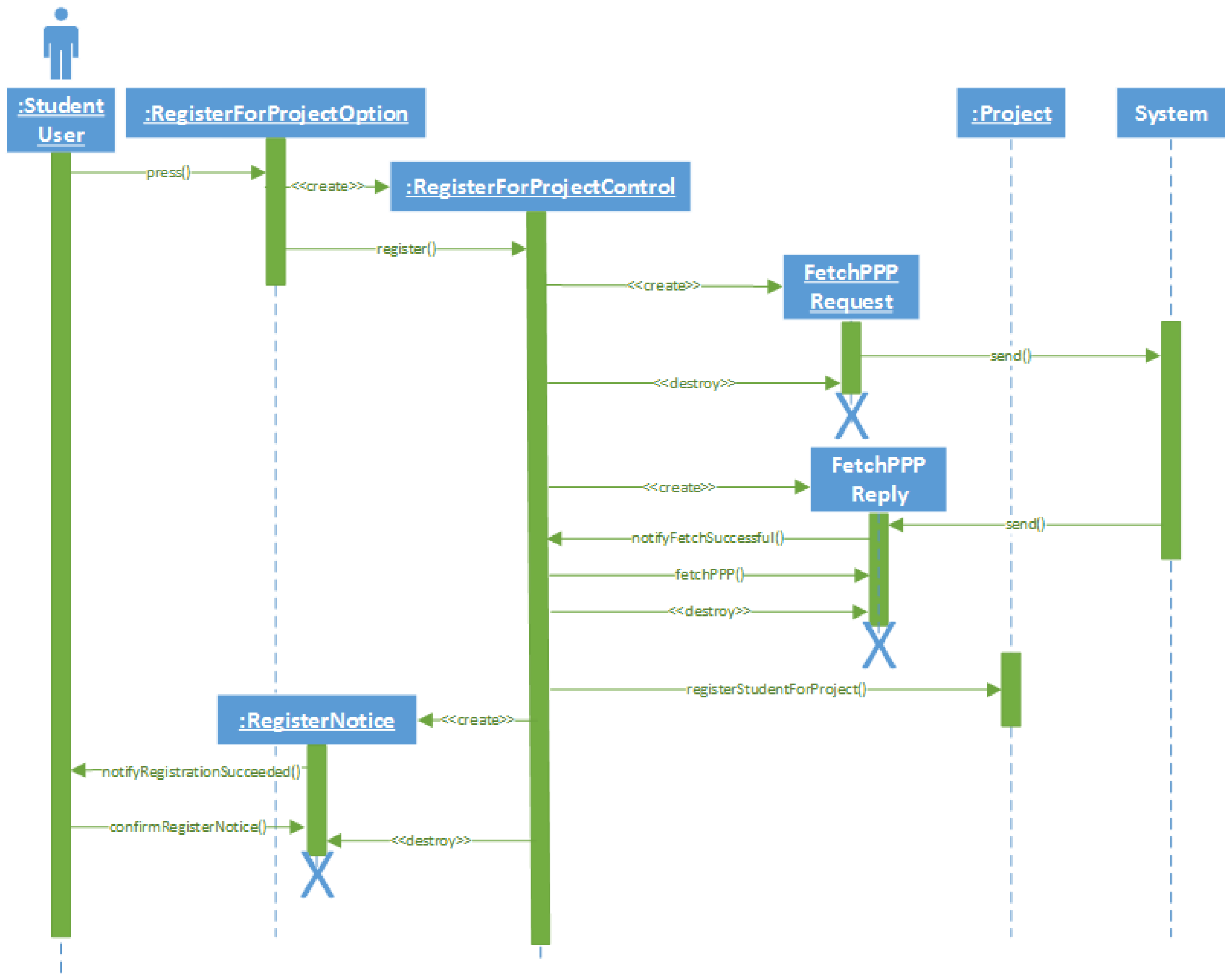

Figure 22 - SD-13 Sequence Diagram For RegisterForProject (UC-12)

SD-14 Sequence diagram for CheckProjectConfigurations (UC-13)

In the sequence diagram below, an AdministratorUser selected to launch the PPID Algorithm and a check for the project configurations has been invoked. A *CheckProjectConfigurationControl* is created which gets the configuration settings from the *Project* entity. When the configurations have been retrieved, a *ConfigurationsValidator* object is created to verify the project configurations. On successful verification, the *CheckProjectConfigurationControl* returns the green light back to *LaunchPPIDAlgorithmControl* and the algorithm is ready to continue. Based on the fact that this sequence is initiated after the AdministratorUser chooses to launch the PPID algorithm on a particular project, the AdministratorUser isn't participating in this interaction. The sequence diagram below illustrates these interactions.

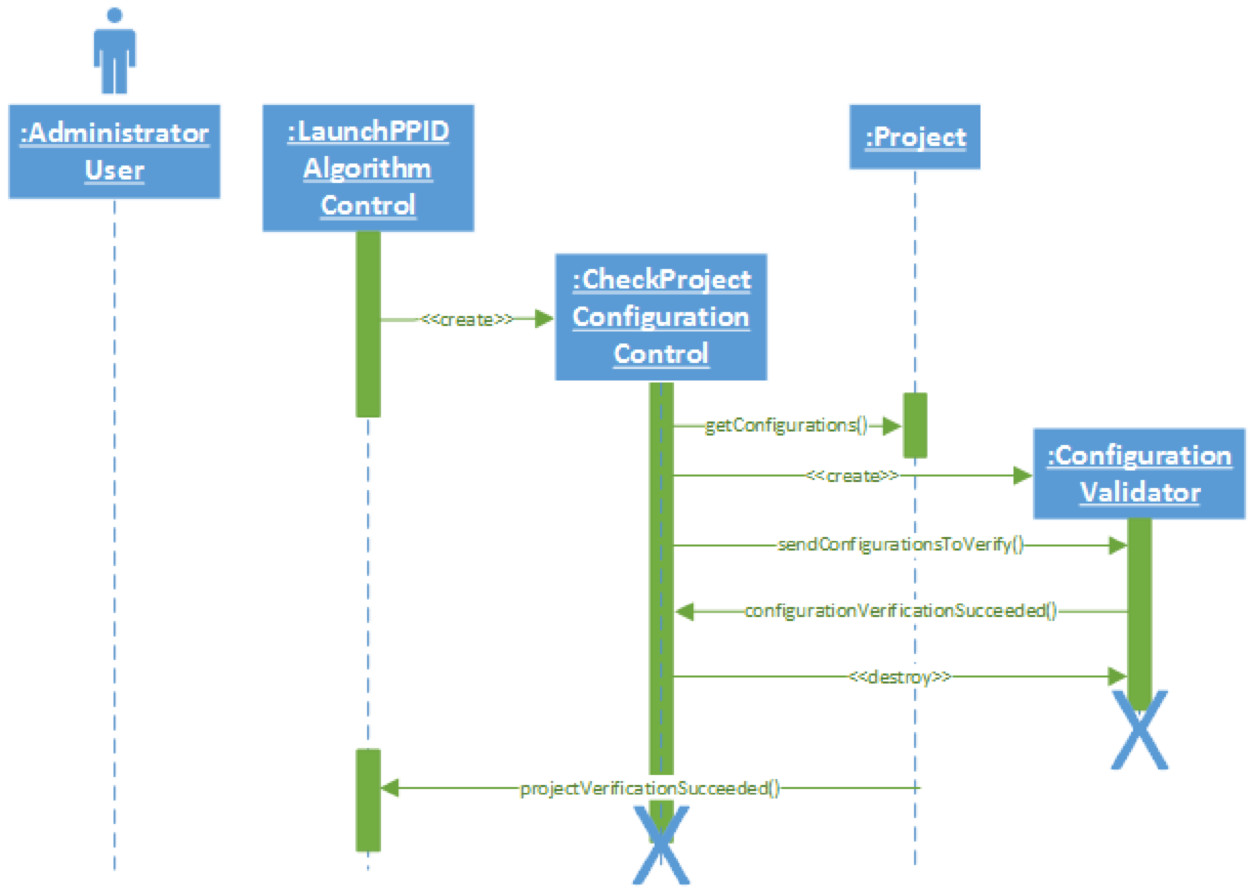
:Administrator User
:LaunchPPID Algorithm Control
:Project
:CheckProject Configuration Control
<<create>>
getConfigurations()
:Configuration Validator
<<create>>
sendConfigurationsToVerify()
configurationVerificationSucceeded()
<<destroy>>
projectVerificationSucceeded()

Figure 23 - SD-14 Sequence Diagram For CheckProjectConfigurations (UC-13)

SD-15 Sequence diagram for PrintMatchResults (UC-14)

In the sequence diagram below, the *LaunchPPIDAlgorithmControl* creates the *PrintMatchResultControl* and tells it to print the match results. This sequence is included by the LaunchPPIDAlgorithm sequence diagram SD-09. The *PrintMatchResultControl* retrieves the raw algorithm results then creates a *DetailedReportGenerator* and sends it the raw algorithm results. The *DetailedReportGenerator* generates a detailed report and sends back to the *PrintMatchResultControl*. The *PrintMatchReultControl* creates a *SummaryReportGenerator*, passes the raw algorithm results. The *SummaryReportGenerator* generates a summary report and sends it to the *PrintMatchResultControl.* The *PrintMatchResultControl* creates a *MatchReportResultView* which notifies the user of the match results. The sequence diagram below illustrates these interactions.

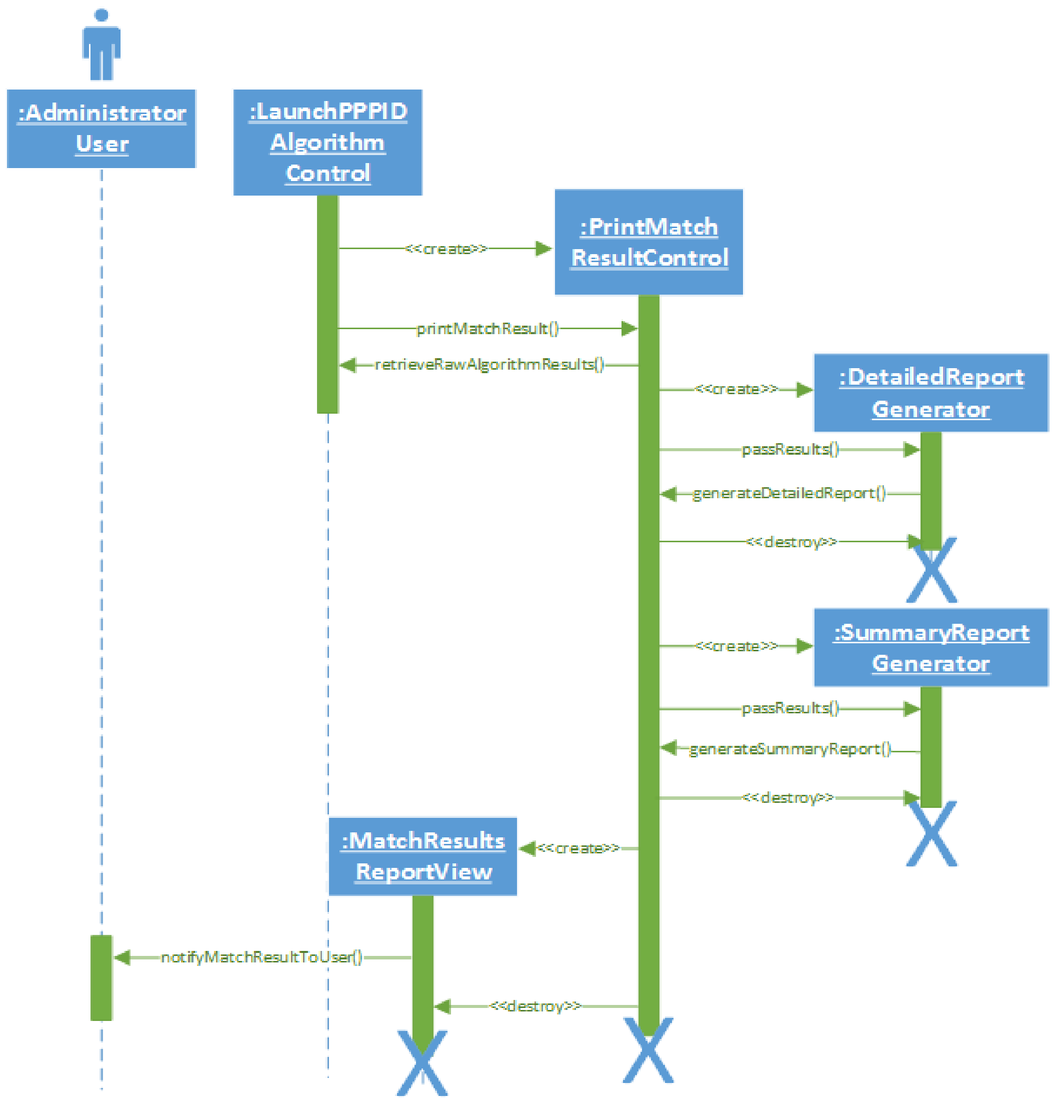

Figure 24 - SD-15 Sequence Diagram For PrintMatchResults (UC-14)

SD-16 Sequence diagram for NoProfileError (UC-15) Extending Edit/DeletePPP (UC-04/UC-03)

In the sequence diagram below, the StudentUser is requesting the projectPartnerProfile by either pressing Edit or Delete PPP, an *EditOrDeletePPPControl* is created, the *EditOrDeletePPPControl* creates a *FetchOrDeleteRequest. the FetchOrDeleteRequest* sends to the system. The *EditOrDeletePPPControl* creates a *FetchOrDeleteReply* which waits for the system to reply. The system sends an error reply. The *FetchOrReply* notifies the *EditOrDeletePPPControl* that the profile doesn't exist. The *EditOrDeletePPPControl* creates a *NoProfileErrorNotice. NoProfileErrorNotice* notifies the user that the profile doesn't exist. The sequence diagram below illustrates these interactions.

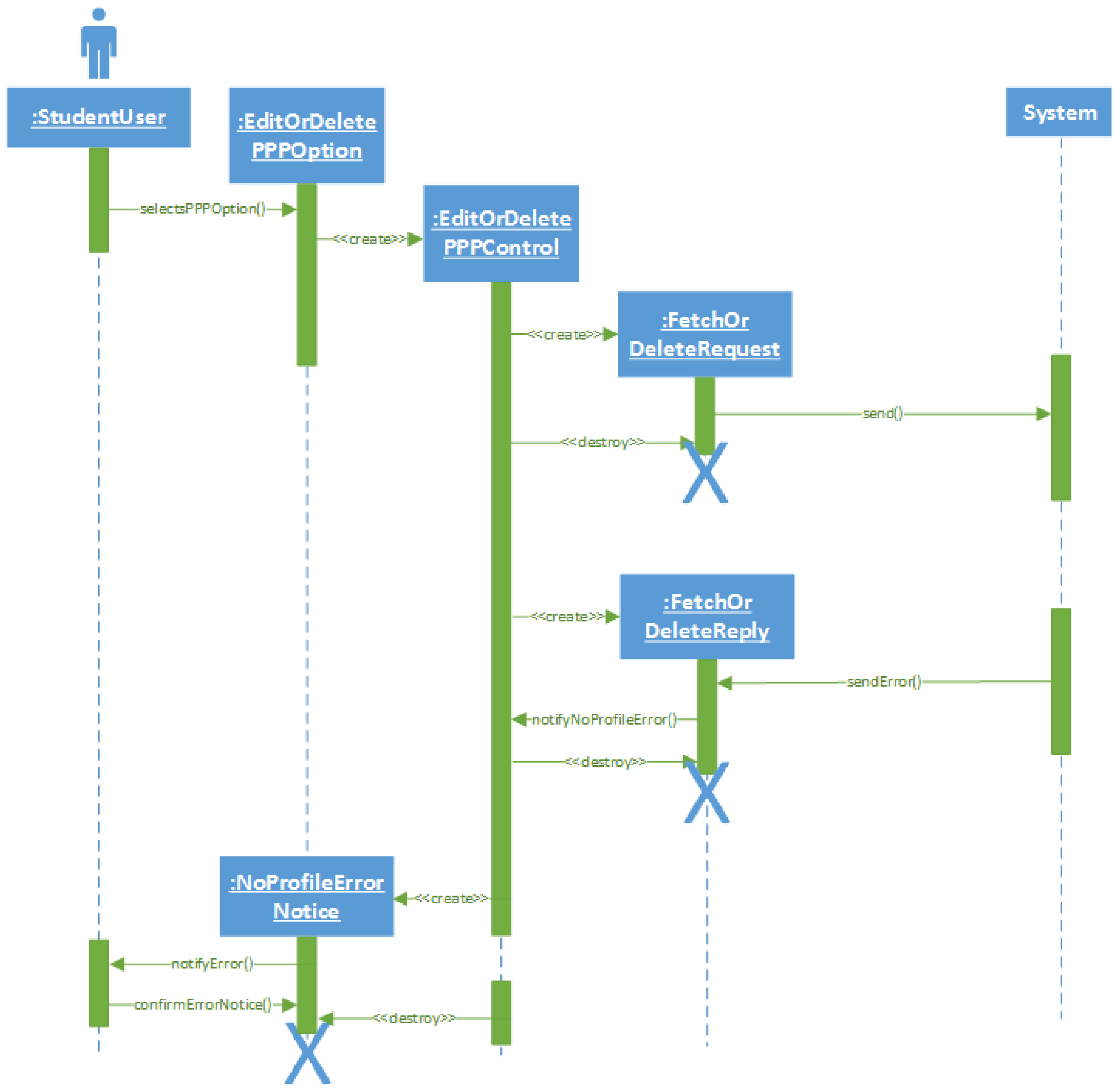

Figure 25 - SD-16 Sequence Diagram For NoProfileError (UC-15) Extending Edit/DeletePPP (UC-04/UC-03)

SD-17 Sequence diagram for NoProfileError (UC-15) Extending RegisterForProject (UC-12)

In the sequence diagram below, the StudentUser select the option to register for a project. The RegisterForProjecOption creates a *RegisterForProjectControl* which creates a *FetchPPPRequest*. The *FetchPPPRequest* sends a request to the system. The *RegisterForProjectContro*l creates a *FetchPPPReply*. The system sends a reply error to the FetchPPPReply, which sends a no profile error notification to the *RegisterForProjectControl.* The *RegisterForProjectControl* creates a NoProfileErrorNotice which sends an error notification to the StudentUser. Finally, the StudentUser acknoledges the error notice. The sequence diagram below illustrates these interactions.

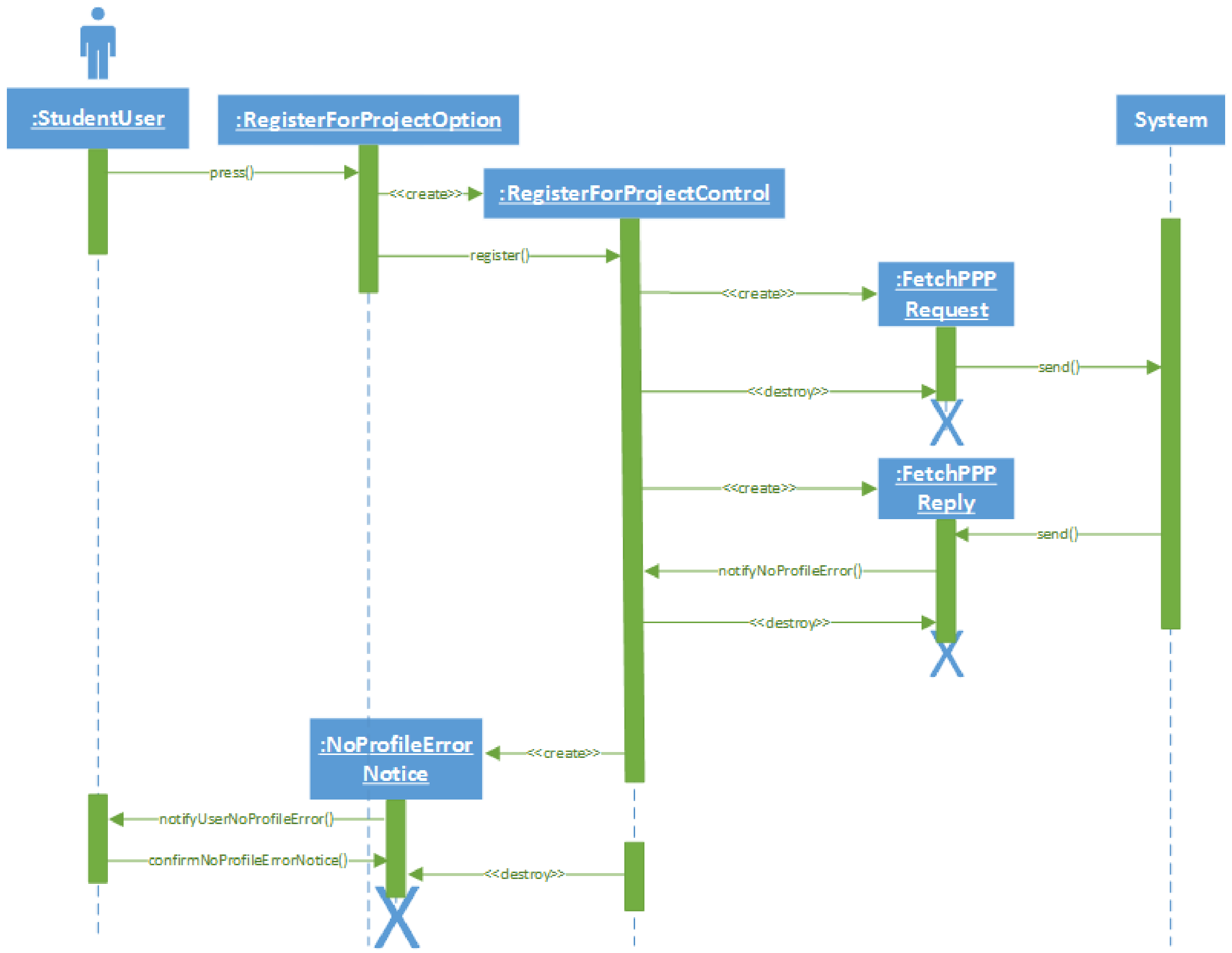

Figure 26 - SD-17 Sequence Diagram For NoProfileError (UC-15) Extending RegisterForProject (UC-12)

SD-18 Sequence diagram for IncompleteProfileError (UC-16) and InvalidInputError (UC-17) Extending SaveProjectPartnerProfile (UC-06)

In the sequence diagram below, the StudentUser submits his PPP to be saved in the system. A *SaveProfileControl* is created to facilitate the save process. This control creates a *FormValidatorControl* object that verifies the completeness and inputs of the form. The validator can return two alternative paths for the sequence. *FormValidatorControl* could return an *IncompleteProfileError* in the case that the form had missing fields and is incomplete. Alternatively, the validator could return *InvalidInputError* in case fields did not contain the format in which was expected. This is done to prevent harmful inputs from compromising the system (NFR-06). *SaveProfileControl* will create the necessary specified notice object to pass back to the user. The sequence diagram below illustrates these interactions.

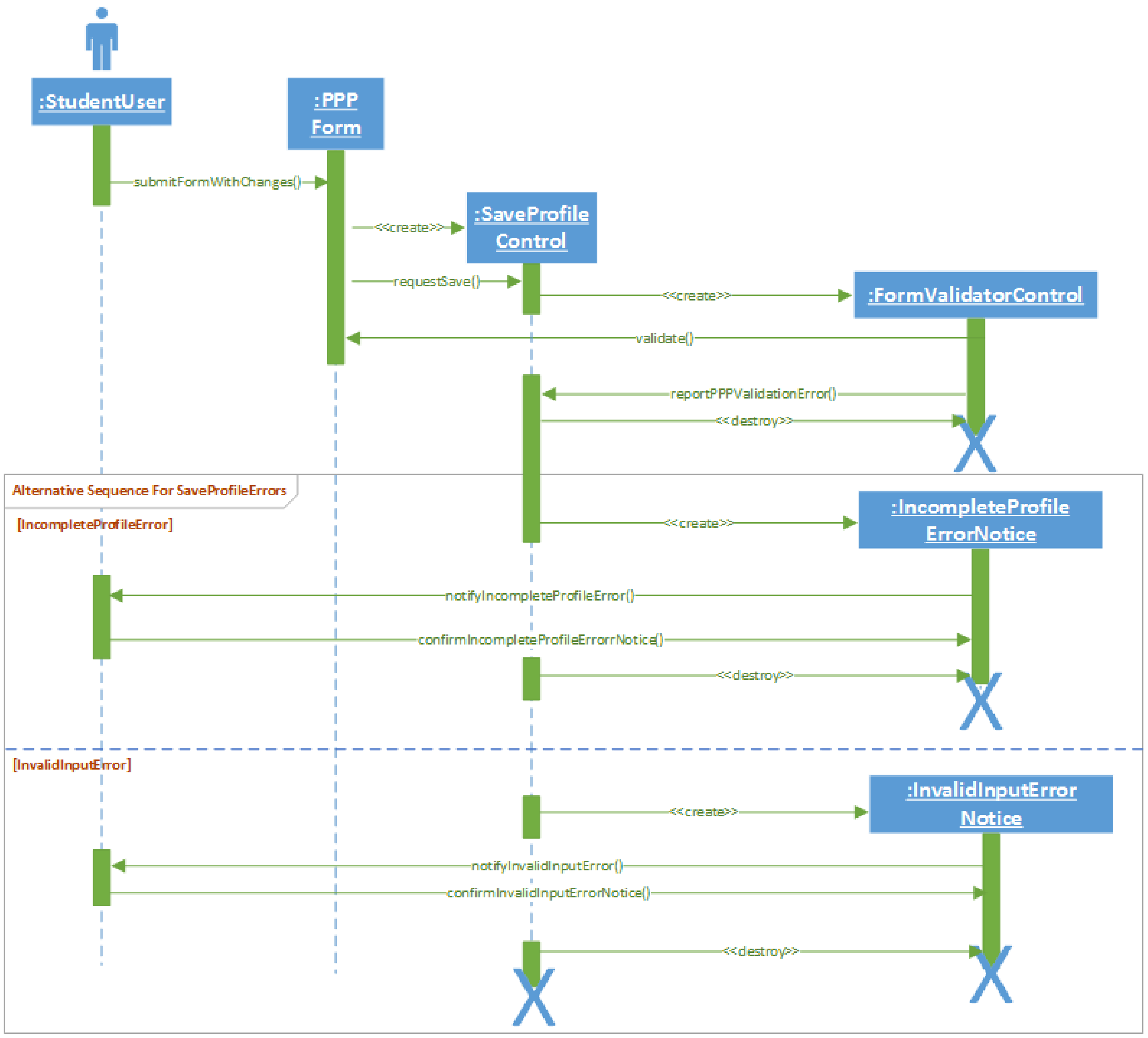
:StudentUser
:PPP Form
submitFormWithChanges()
<<create>>
:SaveProfile Control
requestSave()
<<create>>
:FormValidatorControl
validate()
reportPPPValidationError()
<<destroy>>
Alternative Sequence For SaveProfileErrors
[IncompleteProfileError]
<<create>>
:IncompleteProfile ErrorNotice
notifyIncompleteProfileError()
confirmIncompleteProfileErrorrNotice()
<<destroy>>
[InvalidInputError]
<<create>>
:InvalidInputError Notice
notifyInvalidInputError()
confirmInvalidInputErrorNotice()
<<destroy>>

Figure 27 - SD-18 Sequence Diagram For IncompleteProfileError (UC-16) and InvalidInputError (UC-17) Extending SaveProjectPartnerProfile (06)

SD-19 Sequence diagram for InvalidInputError (UC-17) Extending CreateProject (UC-09)

In the sequence diagram below, the AdministratorUser attempts to create a project, and on validating the form submitted by the AdministratorUser, the system encounters an invalid input provided by the user. On encountering the invalid input, the system notifies the AdministratorUser with an Invalid input error notice. The sequence diagram below illustrates these interactions.

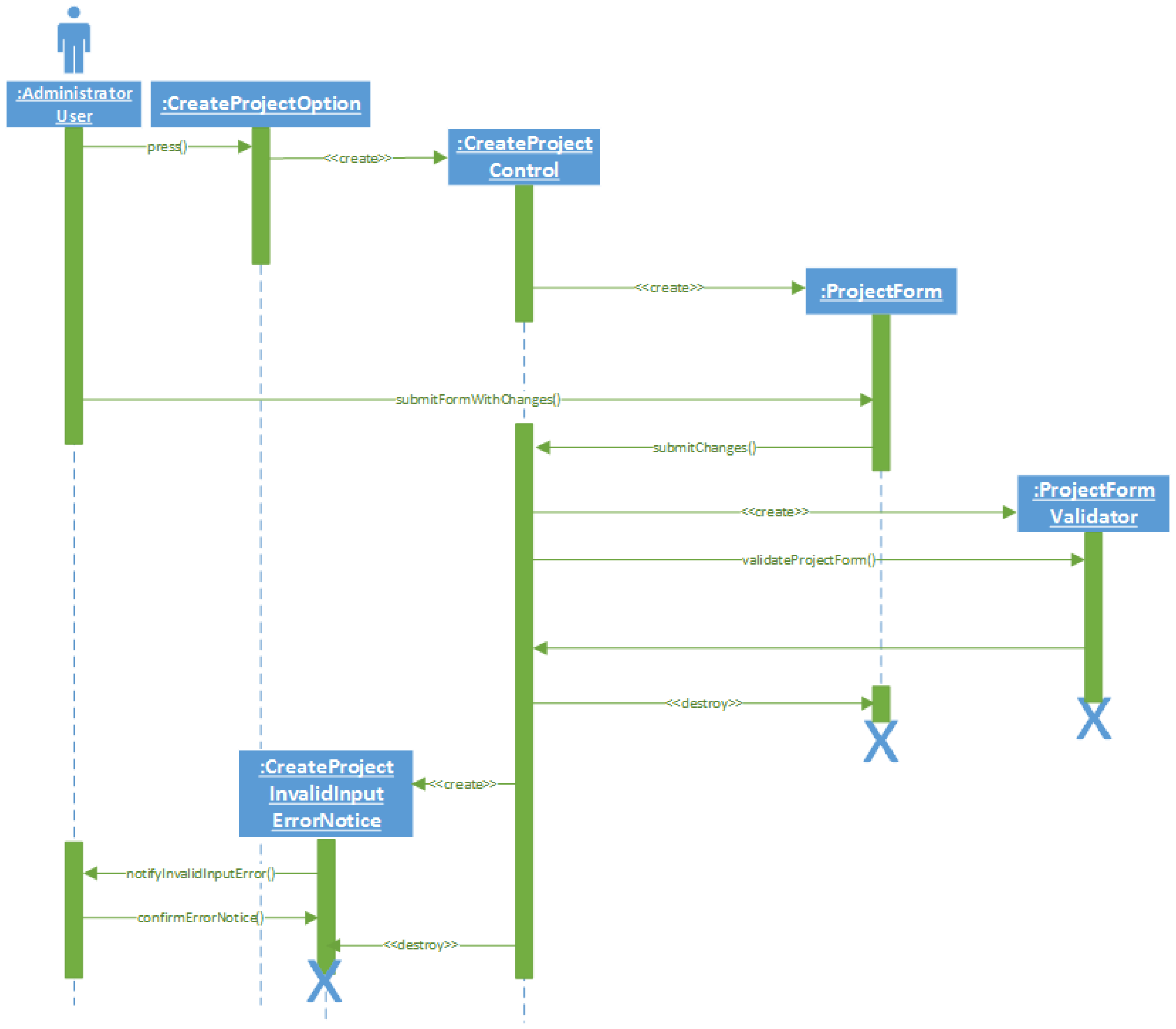

Figure 28 - SD-19 Sequence Diagram For InvalidInputError (UC-17) Extending CreateProject (UC-09)

SD-20 Sequence diagram for DatabaseError (UC-18)

In the sequence diagram below, the AdministratorUser or StudentUser can initiate the invoking option which creates an *InvokingControl*. The *InvokingControl* creates an

InvokingRequest that sends to the system. The *InvokingControl* also creates an *InvokingReply*. The system sends a database error to the *InvokingReply*, *InvokingReply* sends a database error notification to the *InvokingControl*. The *InvokingControl* creates a *DatabaseErrorNotice* and in turn the *DatabaseErrorNotice* sends the database error notification to the AdministratorUser or StudentUser which acknowledges the error notice. The sequence diagram below illustrates these interactions. For the sake of clarifications of ambiguities that may arise, all instance of the possible invoking controls and options are listed below:

- ***InvokingControl:*** *DeletePPPControl SD-04 UC-03, SaveProjectControl SD-07 UC-06, CreateProjectControl SD-10 UC-09, EditProjectConfigurationsControl SD-11 UC-10, UnregisterFromProjectControl SD-12 UC-11, FetchProjectsListControl SD-08 UC-07, RegisterForProjectControl SD-13 UC-12*
- ***InvokingOption:*** *DeletePPPOption SD-04 UC-03, PPPForm SD-07 UC-06, CreateProjectOption SD-10 UC-09, EditProjectConfigurationsOption SD-11 UC-10, UnregisterFromProjectOption SD-12 UC-11, InteractWithProjectOption SD-03 UC-02, RegisterForProjectOption SD-13 UC-12.*

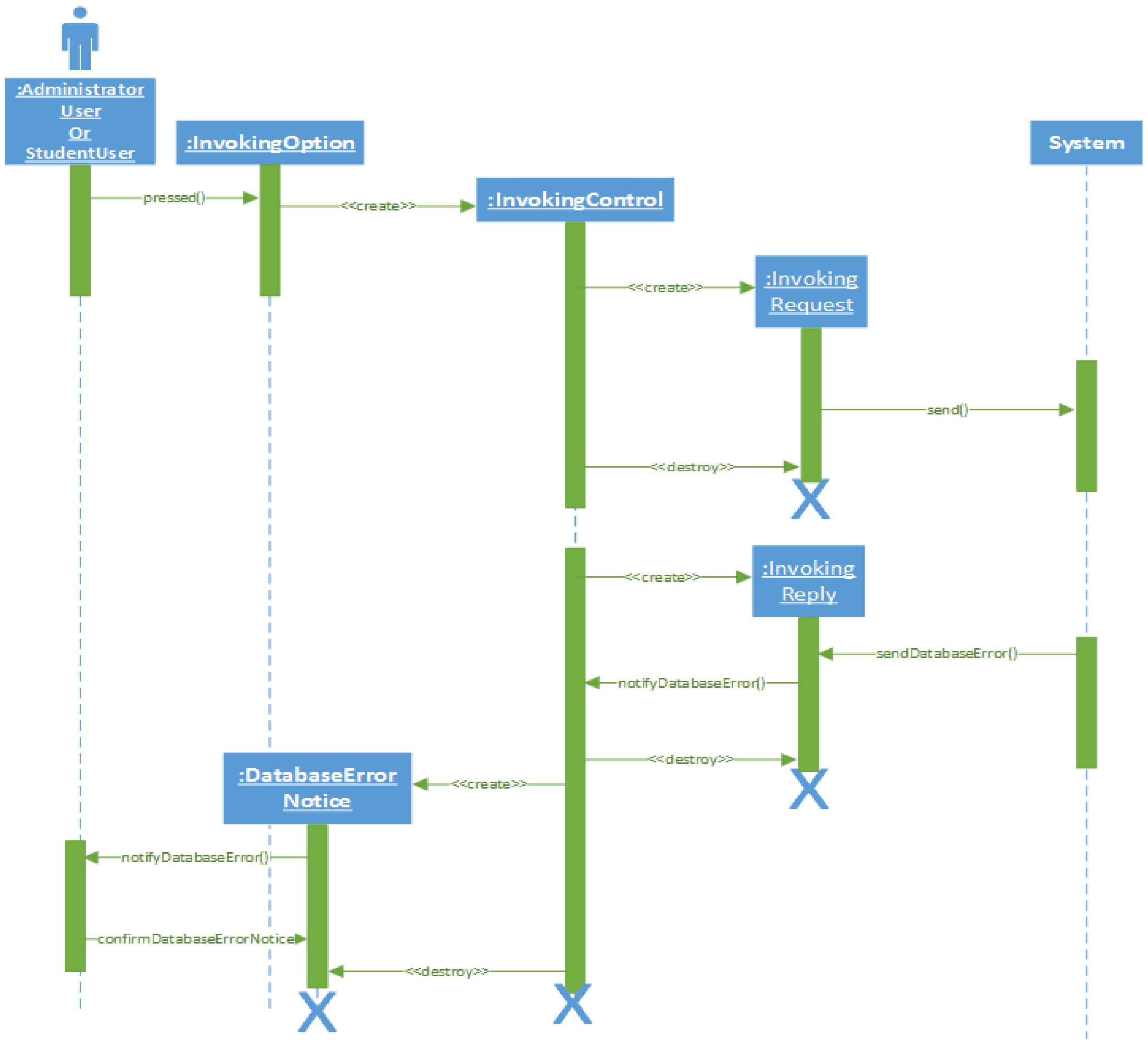

Figure 29 - SD-20 Sequence Diagram For DatabaseError (UC-18)

CheckProjectConfigurations (UC-13)

In the sequence diagram below, the AdministatorUser had launched the PPID algorithm. *LaunchPPIDAlgorithmControl* creates a *CheckProjectConfigurationControl* to facilitate the configuration check procedure. When this control object fetches the non-existent project configurations to run through the *ConfigurationValidator*, a verification error will result from the project. This error is then propagated through the *CheckProjectConfigurationControl* into the *LaunchPPIDAlgorithmControl* and then through to the AdministratorUser in an *IncompleteProjectConfigurationNotice*. The sequence diagram below illustrates these interactions.

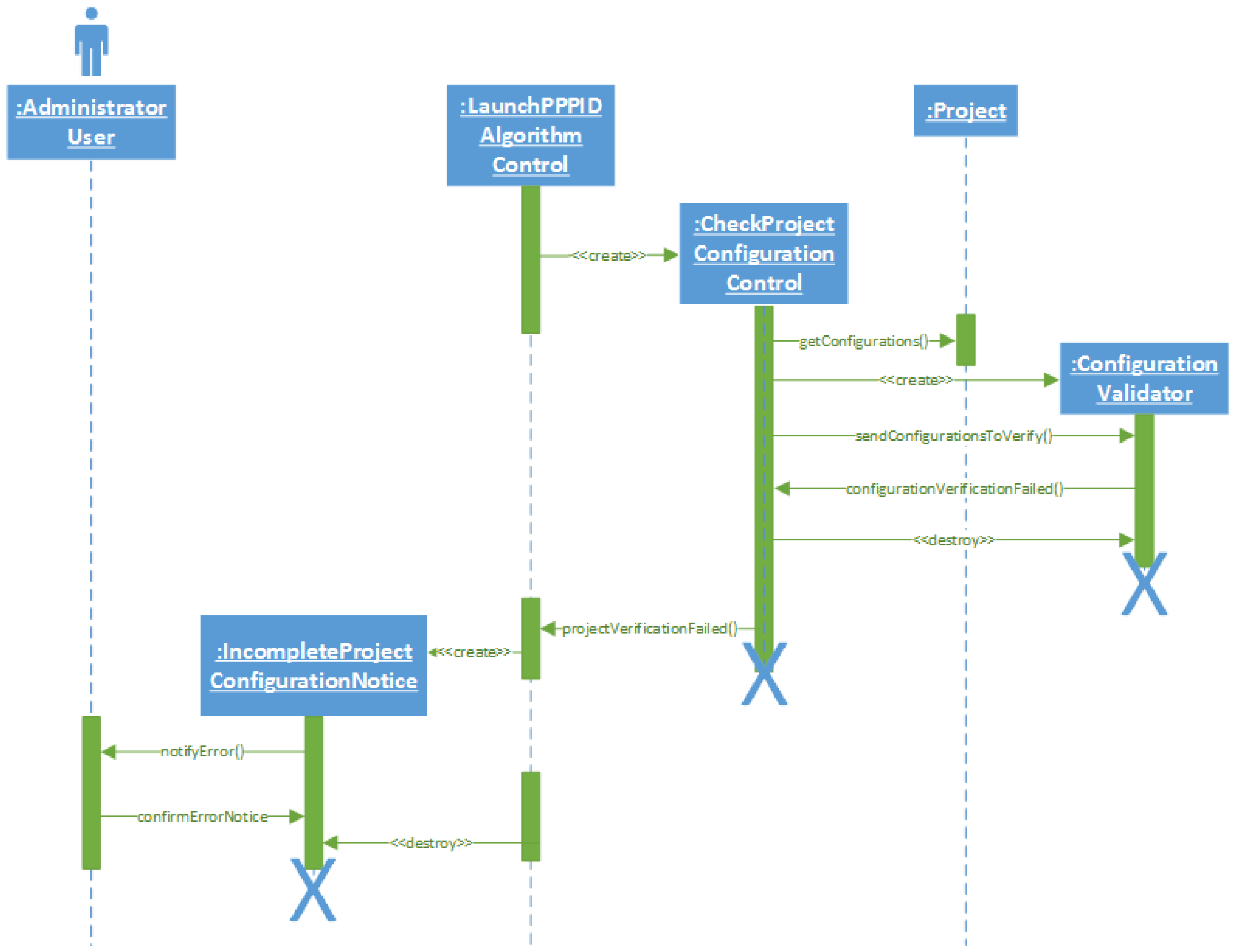
:Administrator User
:LaunchPPPID Algorithm Control
:Project
:CheckProject Configuration Control
<<create>>
getConfigurations()
:Configuration Validator
<<create>>
sendConfigurationsToVerify()
configurationVerificationFailed()
<<destroy>>
projectVerificationFailed()
:IncompleteProject ConfigurationNotice
<<create>>
notifyError()
confirmErrorNotice
<<destroy>>

Figure 30 - SD-21 Sequence Diagram for ProjectConfigurationsNotSetError (UC-19) Extending CheckProjectConfigurationsError (UC-13)

SECTION 3: Abbreviations Used In cuPID Analysis Document

This section of the document outlines all abbreviations used during the course of this analysis document. The essence of this section is to provide the user with the full representation of what an abbreviation stands for during the course of reading the document. The reader should use this section as required to identify the meaning of any abbreviations that elude during his/her course of reading this document. Table 6 shows all the abbreviations used in this document and their respective meanings.

Table 7 - Abbreviations Used in cuPID Analysis Document

Abbreviation	Full Meaning
FR	Functional Requirements
NFR	Non-Functional Requirements
PPP	Project Partner Profile
PPID	Project Partner Identification
cuPID	Carleton University Project Partner Identifier
UC	Use Case
EO	Entity Object
SM	State Machine
SD	Sequence Diagram
UML	Unified Modeling Language