Functional Specification of ProjectAdministration

Specify authors in Ampersand with: META "authors" "<author names>"

26 November 2015

Contents

Chapter 1

Introduction

This is a small demonstration script that uses the basic &-features. It is used as an example in our GitBook, in . Keep this script in the sentinel's Shouldsucceed, ensuring that readers of the GitBook always have something that actually works.

This document[1] defines the functionality of an information system called 'ProjectAdministration'. It defines the database and the business services of ProjectAdministration by means of business rules[2]. Those rules are listed in chapter **??**, ordered by theme. , ordered by theme.

The diagnosis in chapter **??** is meant to help the authors identify shortcomings in their Ampersand script.

[1]This document was generated at 26-11-2015 on 04:04:55, using Ampersand v3.2.0[master:4b6fc5c*], build time: 06-Nov-15 20:29:59 Ame.

[2]Rule based design characterizes the Ampersand approach, which has been used to produce this document.

Chapter 2

Shared Language

This chapter defines the natural language, in which functional requirements of 'ProjectAdministration' can be discussed and expressed. The purpose of this chapter is to create shared understanding among stakeholders. The language of 'ProjectAdministration' consists of concepts and basic sentences. All functional requirements are expressed in these terms. When stakeholders can agree upon this language, at least within the scope of 'ProjectAdministration', they share precisely enough language to have meaningful discussions about functional requirements. All definitions have been numbered for the sake of traceability.

2.1 Projects

This pattern describes an administration of persons who work on projects. For this reason, it introduces the concepts Project and Person.

The sequel introduces the language of Projects.

At this point, the definitions of *Project*, *Person*, and *Assignment* are given.

Definition 1: planned set of interrelated tasks to be executed over a fixed period and within certain cost and other limitations *Project*

In order to administer project participants, the system must register information about them. For that reason, we introduce the concept Person.

Definition 2: A person is any human being *Person*

In order to allow a planner to allocate participants to projects, we introduce the concept of assignment. This will allow us to express rules such as: a person may register his hours on a project from the start date mentioned on his assignment to that project.

Definition 3: An assignment links one person to one project from a given start date. *Assignment*

In order to refer to a project, it must be identifiable, which means that it must be possible to select or find it in the set of existing projects. We choose to use the project's name for that.

In order to refer to a person (in the system), (s)he must be identifiable, which means that it must be possible to select him or her from the set of registered people. We choose to use the person's email-address for that.

As a matter of definition, we choose to consider the project leader of a project to not be a (working) member of a project. Therefore, we need a rule that ensures this is the case.

While it is possible that a project lacks a projectleader, this is an undesired situation. Planners are given the job to find a new projectleader for such projects. The projects that are in need for a projectleader must therefore be signalled.

When a member of some project becomes the project leader of that project, it cannot be a project member any more. This is a consequence of the choice that project leaders are not considered to be members of the projects they lead. Whenever this is the case, the membership is automatically removed.

We say that a person works with another person if there is a project which they share. This means that either person can be a member or a project leader (since there may be multiple project leaders) of a specific project. Therefore, we need a rule that populates the relation 'workswith'. in appropriate cases

We say that a person works with another person if there is a project which they share. This means that either person can be a member or a project leader (since there may be multiple project leaders) of a specific project. Therefore, we need a rule that depopulates the relation 'workswith' in appropriate cases.

In order to become a project leader, you need an assignment as project leader. Therefore, we need a rule that creates such structures, and populates them.

Whenever a project participant is discharged from his task, the corresponding Assignment needs to be deleted. This is done by means of an automated rule.

Chapter 3

Diagnosis

This chapter provides an analysis of the Ampersand script of *'ProjectAdministration'*. This analysis is intended for the author(s) of this script. It can be used to complete the script or to improve possible flaws.

ProjectAdministration does not specify which roles may change the contents of which relations.

ProjectAdministration assigns rules to roles. The following table shows the rules that are being maintained by a given role.

Rule	Planner	ExecEngine
Every project must have a projectleader	*no*	
Projectleaders are not members of a team		*no*
Works with (populate)		*no*
Works with (depopulate)		*no*
Create Assignment		*no*
Delete Assignment		*no*

Concepts Project, ProjectName, ProjectStatus, Description, Date, PersonName, PersonStatus, and Email remain without a purpose.

The purpose of relations projectName, projectStatus, projectDescription, projectStartDate, projectStarted, pl, member, personName, personStatus, personEmail, workswith, project, assignee, pplStartDate, and pplStarted is not documented.

All concepts defined in this document are used in relations.

Relations projectStatus, projectDescription, projectStartDate, projectStarted, personName, personStatus, pplStartDate, and pplStarted are not used in any rule.

Figure **??** shows a conceptual diagram with all relations.

Rules are defined, the meaning of which is documented by means of computer generated language:

- *Projects are identifiable by their names*

  line 72:1, file ProjectAdministration.adl

- *People are identifiable by their email-address*

  line 76:1, file ProjectAdministration.adl

- *Project leaders are not considered members of the projects they lead.*

  line 80:1, file ProjectAdministration.adl

- *Every project must have a projectleader*

  line 86:1, file ProjectAdministration.adl

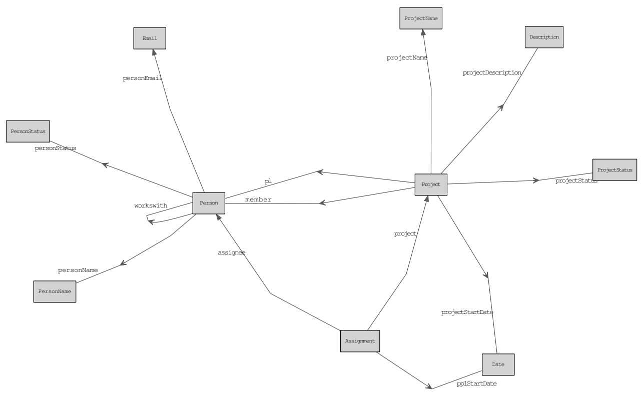

Figure 3.1: Concept diagram of relations in Projects

- *Projectleaders are not members of a team*
  line 94:1, file ProjectAdministration.adl
- *Works with (populate)*
  line 101:1, file ProjectAdministration.adl
- *Works with (depopulate)*
  line 108:1, file ProjectAdministration.adl
- *Create Assignment*
  line 114:1, file ProjectAdministration.adl
- *Delete Assignment*
  line 123:1, file ProjectAdministration.adl

The table below shows for each theme (i.e. process or pattern) the number of relations and rules, followed by the number and percentage that have a reference. Relations declared in multiple themes are counted multiple times.

Theme	Relations	With reference	%	Rules	Entire context	%
Projects	15	0	0%	9	0	0%
Entire context	15	0	0%	9	0	0%

TODO: Inleiding bij de rol-regel tabel

role	rule	from
Planner	Every project must have a projectleader	Projects
ExecEngine	Projectleaders are not members of a team	Projects
ExecEngine	Works with (populate)	Projects
ExecEngine	Works with (depopulate)	Projects
ExecEngine	Create Assignment	Projects
ExecEngine	Delete Assignment	Projects

This script contains work in progress. The following tables provide details with line numbers from the original script files.

rule	location	#tasks
Works with (populate)	line 101:1, file ProjectAdministration.adl	40
Create Assignment	line 114:1, file	6

Person	Person
p10001	p10003
p10001	p10002
p10002	p10012
p10002	p10005
p10002	p10003
p10002	p10001
p10003	p10012

Rule‘Create Assignment’**??**says:

This rule contains work (for ExecEngine)The following table shows the items that require attention.

Project	Person
1970.13	p10008
2013.01	p10012
2014.01	p10012
2014.03	p10012
2014.04	p10011
2014.05	p10007

The population in this script violates 0 invariants and 2 process rules.

- *Rule Works with (populate)*

  Total number of work items: 40

Table 3.7: Tasks yet to be performed by ExecEngine

Person	**Person**
p10001	p10012
p10001	p10005
p10001	p10004
p10001	p10003
p10001	p10002
p10002	p10012
p10002	p10005
p10002	p10003
p10002	p10001
p10003	p10012
p10003	p10005
p10003	p10004
p10003	p10002
p10003	p10001
p10004	p10012
p10004	p10005
p10004	p10003
p10004	p10001
p10005	p10012
p10005	p10004
p10005	p10003
p10005	p10002
p10005	p10001
p10006	p10010
p10006	p10009
p10006	p10008
p10008	p10010
p10008	p10009
p10008	p10006
p10009	p10010

Person	Person
p10009	p10008
p10009	p10006
p10010	p10009
p10010	p10008
p10010	p10006
p10012	p10005
p10012	p10004
p10012	p10003
p10012	p10002
p10012	p10001

- *Rule Create Assignment*

  Total number of work items: 6

Table 3.8: Tasks yet to be performed by ExecEngine

Project	Person
1970.13	p10008
2013.01	p10012
2014.01	p10012
2014.03	p10012
2014.04	p10011
2014.05	p10007

Chapter 4

Conceptual Analysis

This chapter defines the formal language, in which functional requirements of ‘ProjectAdministration’ can be analysed and expressed.The purpose of this formalisation is to obtain a buildable specification. This chapter allows an independent professional with sufficient background to check whether the agreements made correspond to the formal rules and definitions.

This is a small demonstration script that uses the basic &-features. It is used as an example in our GitBook, in . Keep this script in the sentinel’s Shouldsucceed, ensuring that readers of the GitBook always have something that actually works.

4.1 Projects

This pattern describes an administration of persons who work on projects. For this reason, it introduces the concepts Project and Person.

Figure **??** shows a conceptual diagram of this pattern.

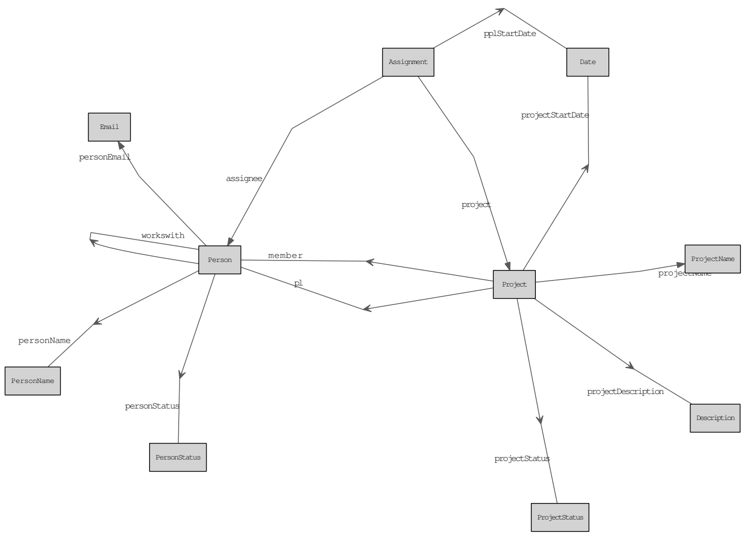

Figure 4.1: Concept diagram of the rules in Projects

The definitions of concepts can be found in the glossary.

4.1.1 Declared relations

This section itemizes the declared relations with properties and purpose.

The following function has been defined

$$\mathit{projectName} : \quad \mathit{Project} \rightarrow \mathit{ProjectName} \tag{4.1}$$

A project must have one name
The following univalent relation has been defined

$$\mathit{projectStatus} : \quad \mathit{Project} \times \mathit{ProjectStatus} \tag{4.2}$$

A project can have one status, such as 'in progress', or 'completed'
The following univalent relation has been defined

$$\mathit{projectDescription} : \quad \mathit{Project} \times \mathit{Description} \tag{4.3}$$

A project can have a description, e.g. stating the result it aims to achieve
The following univalent relation has been defined

$$\mathit{projectStartDate} : \quad \mathit{Project} \times \mathit{Date} \tag{4.4}$$

The start date of a project can be specified
The following symmetric, antisymmetric, univalent, and injective relation has been defined

$$\mathit{projectStarted} : \quad \mathit{Project} \times \mathit{Project} \tag{4.5}$$

Projects can have the property of having been started
The following relation has been defined

$$\mathit{pl} : \quad \mathit{Project} \times \mathit{Person} \tag{4.6}$$

A project can have any number of project leaders
The following relation has been defined

$$\mathit{member} : \quad \mathit{Project} \times \mathit{Person} \tag{4.7}$$

A person can be assigned to work within a project
The following univalent relation has been defined

$$\mathit{personName} : \quad \mathit{Person} \times \mathit{PersonName} \tag{4.8}$$

A person can have (at most) one name
The following univalent relation has been defined

$$\mathit{personStatus} : \quad \mathit{Person} \times \mathit{PersonStatus} \tag{4.9}$$

A person can have a status
The following function has been defined

$$\mathit{personEmail} : \quad \mathit{Person} \rightarrow \mathit{Email} \tag{4.10}$$

A person can have an email-address
The following relation has been defined

$$\mathit{workswith} : \quad \mathit{Person} \times \mathit{Person} \tag{4.11}$$

A person can work with another person (in some project)

The following function has been defined

$$project : \quad Assignment \to Project \tag{4.12}$$

Every Assignment must apply to one project

The following function has been defined

$$assignee : \quad Assignment \to Person \tag{4.13}$$

Every Assignment must apply to one person

The following univalent relation has been defined

$$pplStartDate : \quad Assignment \times Date \tag{4.14}$$

The date at which the Assignment started may be known

The following symmetric, antisymmetric, univalent, and injective relation has been defined

$$pplStarted : \quad Assignment \times Assignment \tag{4.15}$$

A Assignment may have the property that it has been started

4.1.2 Rules

This section itemizes the rules with a reference to the shared language of stakeholders for the sake of traceability.

In order to refer to a project, it must be identifiable, which means that it must be possible to select or find it in the set of existing projects. We choose to use the project's name for that.
Therefore **??** exists:
Using relations **??** (projectName), this is formalized as

$$\text{projectName}; \text{projectName}^{\smile} \vdash I_{[\text{Project}]} \tag{4.16}$$

Figure **??** shows a conceptual diagram of this rule.

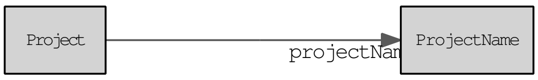

Figure 4.2: Concept diagram of rule Projects are identifiable by their names

In order to refer to a person (in the system), (s)he must be identifiable, which means that it must be possible to select him or her from the set of registered people. We choose to use the person's email-address for that.
Therefore **??** exists:
Using relations **??** (personEmail), this is formalized as

$$\text{personEmail}; \text{personEmail}^{\smile} \vdash I_{[\text{Person}]} \tag{4.17}$$

Figure **??** shows a conceptual diagram of this rule.

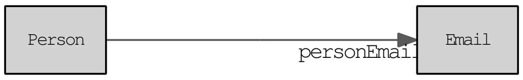

Figure 4.3: Concept diagram of rule People are identifiable by their email-address

As a matter of definition, we choose to consider the project leader of a project to not be a (working) member of a project. Therefore, we need a rule that ensures this is the case. Therefore **??** exists:

Using relations **??** (pl), **??** (member), this is formalized as

$$\mathrm{pl} \vdash \overline{\mathrm{member}} \tag{4.18}$$

Figure **??** shows a conceptual diagram of this rule.

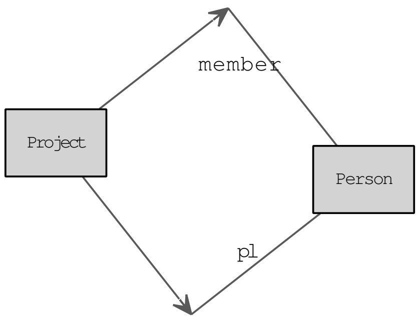

Figure 4.4: Concept diagram of rule Project leaders are not considered members of the projects they lead.

Chapter 5

Process Analysis

This is a small demonstration script that uses the basic &-features. It is used as an example in our GitBook, in . Keep this script in the sentinel's Shouldsucceed, ensuring that readers of the GitBook always have something that actually works.

ProjectAdministration does not specify which roles may change the contents of which relations.

ProjectAdministration assigns rules to roles. The following table shows the rules that are being maintained by a given role.

Role	Rule
Planner	Every project must have a projectleader
ExecEngine	Projectleaders are not members of a team Works with (populate) Works with (depopulate) Create Assignment Delete Assignment

5.1 Projects

This pattern describes an administration of persons who work on projects. For this reason, it introduces the concepts Project and Person.

Rule: Projects are identifiable by their names In order to refer to a project, it must be identifiable, which means that it must be possible to select or find it in the set of existing projects. We choose to use the project's name for that.
We use relations **??**(*projectName*).
This means:

$$\text{projectName}; \text{projectName}^{\smile} \vdash I_{[\text{Project}]} \tag{5.1}$$

Rule: People are identifiable by their email-address In order to refer to a person (in the system), (s)he must be identifiable, which means that it must be possible to select him or her from the set of registered people. We choose to use the person's email-address for that.
We use relations **??**(*personEmail*).
This means:

$$\text{personEmail}; \text{personEmail}^{\smile} \vdash I_{[\text{Person}]} \tag{5.2}$$

Rule: Project leaders are not considered members of the projects they lead. As a matter of definition, we choose to consider the project leader of a project to not be a (working) member of a project. Therefore, we need a rule that ensures this is the case.
We use relations **??** (*pl*) and **??** (*member*)

This means:

$$\text{pl} \vdash \overline{\text{member}} \tag{5.3}$$

Rule: Every project must have a projectleader While it is possible that a project lacks a projectleader, this is an undesired situation. Planners are given the job to find a new projectleader for such projects. The projects that are in need for a projectleader must therefore be signalled.
We use relations **??**(*pl*).
Activities that are defined by this rule are finished when:

$$I_{[\text{Project}]} \vdash \text{pl};\text{pl}^{\smile} \tag{5.4}$$

Rule: Projectleaders are not members of a team When a member of some project becomes the project leader of that project, it cannot be a project member any more. This is a consequence of the choice that project leaders are not considered to be members of the projects they lead. Whenever this is the case, the membership is automatically removed.
We use relations **??** (*pl*) and **??** (*member*)
Activities that are defined by this rule are finished when:

$$\text{pl} \vdash \overline{\text{member}} \tag{5.5}$$

Rule: Works with (populate) We say that a person works with another person if there is a project which they share. This means that either person can be a member or a project leader (since there may be multiple project leaders) of a specific project. Therefore, we need a rule that populates the relation 'workswith'. in appropriate cases
We use relations **??** (*pl*), **??** (*member*), and **??** (*workswith*)
Activities that are defined by this rule are finished when:

$$(\text{pl} \cup \text{member})^{\smile};(\text{pl} \cup \text{member}) - I_{[\text{Person}]} \vdash \text{workswith} \tag{5.6}$$

Rule: Works with (depopulate) We say that a person works with another person if there is a project which they share. This means that either person can be a member or a project leader (since there may be multiple project leaders) of a specific project. Therefore, we need a rule that depopulates the relation 'workswith' in appropriate cases.
We use relations **??** (*pl*), **??** (*member*), and **??** (*workswith*)
Activities that are defined by this rule are finished when:

$$\text{workswith} \vdash (\text{pl} \cup \text{member})^{\smile};(\text{pl} \cup \text{member}) - I_{[\text{Person}]} \tag{5.7}$$

Rule: Create Assignment In order to become a project leader, you need an assignment as project leader. Therefore, we need a rule that creates such structures, and populates them.
We use relations **??** (*pl*), **??** (*project*), and **??** (*assignee*)
Activities that are defined by this rule are finished when:

$$\text{pl} \vdash \text{project}^{\smile};\text{assignee} \tag{5.8}$$

Rule: Delete Assignment Whenever a project participant is discharged from his task, the corresponding Assignment needs to be deleted. This is done by means of an automated rule.
We use relations **??** (*pl*), **??** (*member*), **??** (*project*), and **??** (*assignee*)
Activities that are defined by this rule are finished when:

$$\text{project}^{\smile};\text{assignee} \vdash \text{pl} \cup \text{member} \tag{5.9}$$

Chapter 6

Data structure

This chapter contains the result of the data analysis. It is structured as follows:

We start with the classification model, followed by a list of all relations, that are the foundation of the rest of the analysis. Finally, the logical and technical data model are discussed.

6.1 Rules

TODO: explain section

6.1.1 Process rules

TODO: explain process rules

Process rule: ***Every project must have a projectleader***

While it is possible that a project lacks a projectleader, this is an undesired situation. Planners are given the job to find a new projectleader for such projects. The projects that are in need for a projectleader must therefore be signalled.

$$I_{[\mathrm{Project}]} \vdash \mathrm{pl};\mathrm{pl}^{\smile} \tag{6.1}$$

Process rule: ***Projectleaders are not members of a team***

When a member of some project becomes the project leader of that project, it cannot be a project member any more. This is a consequence of the choice that project leaders are not considered to be members of the projects they lead. Whenever this is the case, the membership is automatically removed.

$$\mathrm{pl} \vdash \overline{\mathrm{member}} \tag{6.2}$$

Process rule: ***Works with (populate)***

We say that a person works with another person if there is a project which they share. This means that either person can be a member or a project leader (since there may be multiple project leaders) of a specific project. Therefore, we need a rule that populates the relation 'workswith'. in appropriate cases

$$(\mathrm{pl} \cup \mathrm{member})^{\smile};(\mathrm{pl} \cup \mathrm{member}) - I_{[\mathrm{Person}]} \vdash \mathrm{workswith} \tag{6.3}$$

Process rule: ***Works with (depopulate)***

We say that a person works with another person if there is a project which they share. This means that either person can be a member or a project leader (since there may be multiple project leaders) of a specific project. Therefore, we need a rule that depopulates the relation 'workswith' in appropriate cases.

$$\text{workswith} \vdash (\text{pl} \cup \text{member})^{\smile}; (\text{pl} \cup \text{member}) - I_{[\text{Person}]} \tag{6.4}$$

Process rule: ***Create Assignment***

In order to become a project leader, you need an assignment as project leader. Therefore, we need a rule that creates such structures, and populates them.

$$\text{pl} \vdash \text{project}^{\smile}; \text{assignee} \tag{6.5}$$

Process rule: ***Delete Assignment***

Whenever a project participant is discharged from his task, the corresponding Assignment needs to be deleted. This is done by means of an automated rule.

$$\text{project}^{\smile}; \text{assignee} \vdash \text{pl} \cup \text{member} \tag{6.6}$$

6.1.2 Invariants

TODO: explain invariants

Invariant: ***Projects are identifiable by their names***

In order to refer to a project, it must be identifiable, which means that it must be possible to select or find it in the set of existing projects. We choose to use the project's name for that.

$$\text{projectName}; \text{projectName}^{\smile} \vdash I_{[\text{Project}]} \tag{6.7}$$

Violations of this rule will result in an error message for the user: "TODO".

Invariant: ***People are identifiable by their email-address***

In order to refer to a person (in the system), (s)he must be identifiable, which means that it must be possible to select him or her from the set of registered people. We choose to use the person's email-address for that.

$$\text{personEmail}; \text{personEmail}^{\smile} \vdash I_{[\text{Person}]} \tag{6.8}$$

Violations of this rule will result in an error message for the user: "TODO".

Invariant: ***Project leaders are not considered members of the projects they lead.***

As a matter of definition, we choose to consider the project leader of a project to not be a (working) member of a project. Therefore, we need a rule that ensures this is the case.

$$\text{pl} \vdash \overline{\text{member}} \tag{6.9}$$

Violations of this rule will result in an error message for the user: "TODO".

6.2 Logical data model

The functional requirements have been translated into a data model. This model is shown by figure **??**.

There are three entity types.The details of each entity type are described (in alphabetical order) in the following two tables:

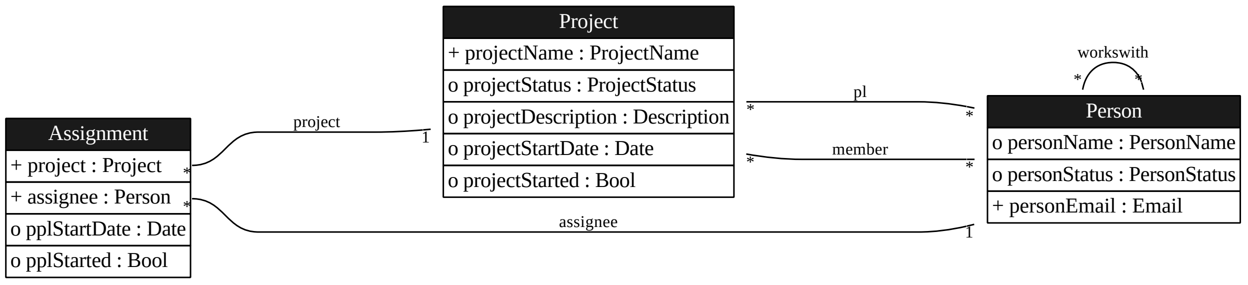

Figure 6.1: Logical data model of ProjectAdministration

Table 6.1: Logical entity types

Concept	Meaning	Type
Assignment	An assignment links one person to one project from a given start date. In order to allow a planner to allocate participants to projects, we introduce the concept of assignment. This will allow us to express rules such as: a person may register his hours on a project from the start date mentioned on his assignment to that project.	
Person	A person is any human being In order to administer project participants, the system must register information about them. For that reason, we introduce the concept Person.	
Project	planned set of interrelated tasks to be executed over a fixed period and within certain cost and other limitations	

Table 6.2: Other attributes

Concept	Meaning	Type
Date		
Description		
Email		
PersonName		
PersonStatus		
ProjectName		
ProjectStatus		
SESSION		

6.2.1 Entity type: *Assignment*

In order to allow a planner to allocate participants to projects, we introduce the concept of assignment. This will allow us to express rules such as: a person may register his hours on a project from the start date mentioned on his assignment to that project.

This entity type has the following attributes:

Attribute	Type	
Id	Assignment	Primary key
project	Project	Mandatory
assignee	Person	Mandatory
pplStartDate	Date	Optional
pplStarted	Bool	Optional

Assignment has the following associations:

1. project (from Assignment to Project).
2. assignee (from Assignment to Person).

Attribute	Type	
Id	Person	Primary key
personName	PersonName	Optional
personStatus	PersonStatus	Optional
personEmail	Email	Mandatory

Person has the following associations:

1. pl (from Project to Person).
2. member (from Project to Person).
3. workswith (from Person to Person).
4. assignee (from Assignment to Person).

6.2.3 Entity type: ***Project***

This entity type has the following attributes:

Attribute	Type	
Id	Project	Primary key
projectName	ProjectName	Mandatory
projectStatus	ProjectStatus	Optional
projectDescription	Description	Optional
projectStartDate	Date	Optional
projectStarted	Bool	Optional

Project has the following associations:

1. pl (from Project to Person).
2. member (from Project to Person).
3. project (from Assignment to Project).

6.3 Technical datamodel

The functional requirements have been translated into a technical data model. This model is shown by figure **??**.

The technical datamodel consists of the following 14 tables:

6.3.1 Table: Assignment

This table has the following 5 fields:

- **Assignment**
  This attribute is the primary key.
  `SQLVarchar 255`, Mandatory, Unique.
- **tgt_project**
  This attribute implements the relation $Assignment \xrightarrow{project} Project$.
  `SQLVarchar 255`, Optional.
- **tgt_assignee**
  This attribute implements the relation $Assignment \xrightarrow{assignee} Person$.
  `SQLVarchar 255`, Optional.
- **tgt_pplStartDate**
  This attribute implements the relation $Assignment \xrightarrow{pplStartDate} Date$.
  `SQLDate`, Optional.

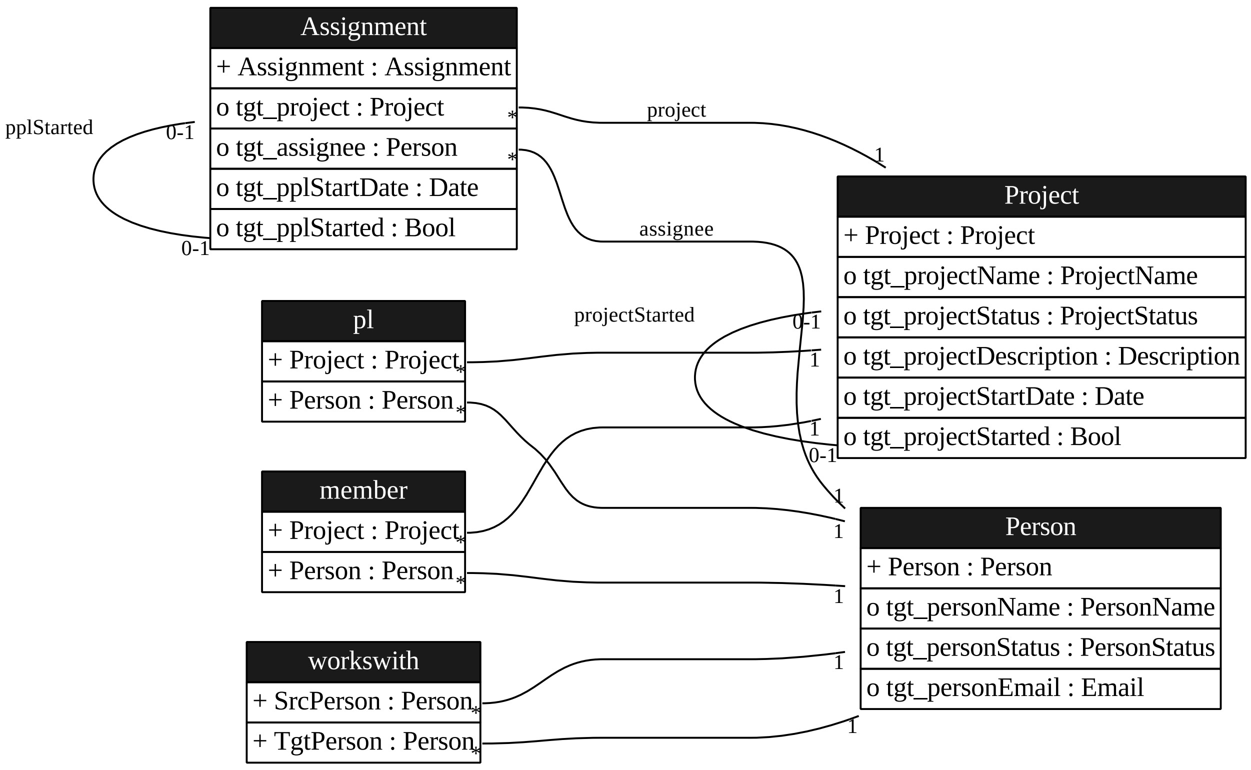

Figure 6.2: Technical data model of ProjectAdministration

- **tgt_pplStarted**
  This attribute implements the relation $Assignment \xrightarrow{pplStarted} Assignment$.
  `SQLVarchar 255`, Optional, Unique.

6.3.2 Table: Date

This table has the following 1 fields:

- **Date**
  This attribute is the primary key.
  `SQLDate`, Mandatory, Unique.

6.3.3 Table: Description

This table has the following 1 fields:

- **Description**
  This attribute is the primary key.
  `SQLText`, Mandatory, Unique.

6.3.4 Table: Email

This table has the following 1 fields:

- **Email**
  This attribute is the primary key.
  `SQLVarchar 255`, Mandatory, Unique.

6.3.5 Table: member

This is a link-table, implementing the relation $Project \xrightarrow{member} Person$. It contains the following columns:

- **Project**
  This attribute is a foreign key to Project
  `SQLVarchar 255`, Mandatory.
- **Person**
  This attribute implements the relation $Project \xrightarrow{member} Person$.
  `SQLVarchar 255`, Mandatory.

6.3.6 Table: Person

This table has the following 4 fields:

- **Person**
  This attribute is the primary key.
  `SQLVarchar 255`, Mandatory, Unique.
- **tgt_personName**
  This attribute implements the relation $Person \xrightarrow{personName} PersonName$.
  `SQLVarchar 255`, Optional.
- **tgt_personStatus**
  This attribute implements the relation $Person \xrightarrow{personStatus} PersonStatus$.
  `SQLVarchar 255`, Optional.
- **tgt_personEmail**
  This attribute implements the relation $Person \xrightarrow{personEmail} Email$.
  `SQLVarchar 255`, Optional.

6.3.7 Table: PersonName

This table has the following 1 fields:

- **PersonName**
  This attribute is the primary key.
  `SQLVarchar 255`, Mandatory, Unique.

6.3.8 Table: PersonStatus

This table has the following 1 fields:

- **PersonStatus**
  This attribute is the primary key.
  `SQLVarchar 255`, Mandatory, Unique.

6.3.9 Table: pl

This is a link-table, implementing the relation $Project \xrightarrow{pl} Person$. It contains the following columns:

- **Project**
  This attribute is a foreign key to Project
  `SQLVarchar 255`, Mandatory.
- **Person**
  This attribute implements the relation $Project \xrightarrow{pl} Person$.
  `SQLVarchar 255`, Mandatory.

6.3.10 Table: Project

This table has the following 6 fields:

- **Project**
  This attribute is the primary key.
  `SQLVarchar 255`, Mandatory, Unique.
- **tgt__projectName**
  This attribute implements the relation $Project \xrightarrow{projectName} ProjectName$.
  `SQLVarchar 255`, Optional.
- **tgt__projectStatus**
  This attribute implements the relation $Project \xrightarrow{projectStatus} ProjectStatus$.
  `SQLVarchar 255`, Optional.
- **tgt__projectDescription**
  This attribute implements the relation $Project \xrightarrow{projectDescription} Description$.
  `SQLText`, Optional.
- **tgt__projectStartDate**
  This attribute implements the relation $Project \xrightarrow{projectStartDate} Date$.
  `SQLDate`, Optional.
- **tgt__projectStarted**
  This attribute implements the relation $Project \xrightarrow{projectStarted} Project$.
  `SQLVarchar 255`, Optional, Unique.

6.3.11 Table: ProjectName

This table has the following 1 fields:

- **ProjectName**
  This attribute is the primary key.
  `SQLVarchar 255`, Mandatory, Unique.

6.3.12 Table: ProjectStatus

This table has the following 1 fields:

- **ProjectStatus**
  This attribute is the primary key.
  `SQLVarchar 255`, Mandatory, Unique.

6.3.13 Table: SESSION

This table has the following 1 fields:

- **SESSION**
  This attribute is the primary key.
  `SQLVarchar 255`, Mandatory, Unique.

6.3.14 Table: workswith

This is a link-table, implementing the relation $Person \xrightarrow{workswith} Person$. It contains the following columns:

- **SrcPerson**
  This attribute is a foreign key to Person
  `SQLVarchar 255`, Mandatory.

- **TgtPerson**
  This attribute implements the relation $Person \xrightarrow{workswith} Person$.
  `SQLVarchar 255`, Mandatory.

6.4 Logical data model

Concept	C	R	U	D
Assignment		Project		
Assignment			Person	
Date		Started projects		
Description		Started projects		
Description		Unstarted projects		
Description		Project		
Email		People		
Email		Project		
Email	Person	Person		Person
Person		Started projects		
Person		Unstarted projects		
Person	People	People		People
Person		Project		
Person		Person		
PersonName		People		
PersonName		Project		
PersonName	Person	Person		Person
PersonStatus		Project		
PersonStatus		Person		
Project		Started projects		
Project	Unstarted projects	Unstarted projects		Unstarted projects
Project		People		
Project		Project		
ProjectName		Started projects		
ProjectName		Unstarted projects		
ProjectName		Project		

Chapter 7

Interface: “Started projects”

Dit hoofdstuk bevat de documentatie voor de interface “Started projects”.

De interface is beschikbaar voor alle rollen.

Voor deze interface hoeven geen regels gecontroleerd te worden.

CRUD matrix:

Concept	C	R	U	D
Project		√		
ProjectName		√		
Person		√		
Description		√		
Date		√		

Interfacestructuur:

Interface voor een waarde van type “Project”.

Een lijst van 0 of meer velden van type “Project”. (niet editable)

De bijbehorende Ampersand expressie is:

`V[SESSION*Project];projectStarted`

Elk veld bestaat uit 7 deelvelden:

1 Projects

Een verplicht veld van type “Project”. (niet editable)

De bijbehorende Ampersand expressie is:

`I[Project]`

2 Name

Een optioneel veld van type “ProjectName”. (niet editable)

De bijbehorende Ampersand expressie is:

`projectName`

3 Description

Een optioneel veld van type “Description”. (niet editable)

De bijbehorende Ampersand expressie is:

`projectDescription`

4 Projectleider

Een lijst van 0 of meer velden van type “Person”. (niet editable)

De bijbehorende Ampersand expressie is:

`pl`

5

Een lijst van 0 of meer velden van type “Person”. (niet editable)

De bijbehorende Ampersand expressie is:

`pl`

6 Start

Een optioneel veld van type “Date”. (niet editable)

De bijbehorende Ampersand expressie is:

`projectStartDate`

7 Started

Een optioneel veld van type “Project”. (niet editable)

De bijbehorende Ampersand expressie is:

`projectStarted`

Chapter 8

Interface: “Unstarted projects”

Dit hoofdstuk bevat de documentatie voor de interface “Unstarted projects”.

De interface is beschikbaar voor alle rollen.

Voorafgaand aan het afsluiten van een transactie (commit), moet aan de volgende regels voldaan zijn:

CRUD matrix:

Concept	C	R	U	D
Project	✓	✓		✓
ProjectName		✓		
Person		✓		
Description		✓		

Interfacestructuur:

Interface voor een waarde van type “Project”.

Een lijst van 0 of meer velden van type “Project”. (niet editable)

De bijbehorende Ampersand expressie is:

`V[SESSION*Project];(I[Project] - projectStarted)`

Elk veld bestaat uit 5 deelvelden:

1 Name

Een optioneel veld van type “ProjectName”. (niet editable)

De bijbehorende Ampersand expressie is:

`projectName`

2 Description

Een optioneel veld van type “Description”. (niet editable)

De bijbehorende Ampersand expressie is:

`projectDescription`

3 Projectleider

Een lijst van 0 of meer velden van type “Person”. (niet editable)

De bijbehorende Ampersand expressie is:

`pl`

4

Een lijst van 0 of meer velden van type “Person”. (niet editable)

De bijbehorende Ampersand expressie is:

`pl`

5 Started

Een optioneel veld van type “Project”. (editable)

De bijbehorende Ampersand expressie is:

`projectStarted`

Chapter 9

Interface: “People”

Dit hoofdstuk bevat de documentatie voor de interface “People”.

De interface is beschikbaar voor alle rollen.

Voorafgaand aan het afsluiten van een transactie (commit), moet aan de volgende regels voldaan zijn:

- Works with (populate)
- Delete Assignment
- Project leaders are not considered members of the projects they lead.
- Projectleaders are not members of a team
- Works with (depopulate)

CRUD matrix:

Concept	C	R	U	D
Project		√		
Person	√	√		√
Email		√		
PersonName		√		

Interfacestructuur:

Interface voor een waarde van type “Person”.

Een lijst van 1 of meer velden van type “Person”. (niet editable)

De bijbehorende Ampersand expressie is:

`V[SESSION*Person]`

Elk veld bestaat uit 4 deelvelden:

1 Person

Een verplicht veld van type “Person”. (niet editable)

De bijbehorende Ampersand expressie is:

`I[Person]`

2 Name

Een optioneel veld van type “PersonName”. (niet editable)

De bijbehorende Ampersand expressie is:

`personName`

3 Email

Een optioneel veld van type “Email”. (niet editable)

De bijbehorende Ampersand expressie is:

```
personEmail
```

4 Projects

Een lijst van 0 of meer velden van type “Project”. (editable)

De bijbehorende Ampersand expressie is:

```
member~
```

Chapter 10

Interface: “Project”

Dit hoofdstuk bevat de documentatie voor de interface “Project”.

De interface is beschikbaar voor alle rollen.

Voor deze interface hoeven geen regels gecontroleerd te worden.

CRUD matrix:

Concept	C	R	U	D
Project		√		
ProjectName		√		
Person		√		
Email		√		
Assignment		√		
Description		√		
PersonName		√		
PersonStatus		√		

Interfacestructuur:

Interface voor een waarde van type “Project”.

Een verplicht veld van type “Project”. (niet editable)

De bijbehorende Ampersand expressie is:

`I[Project]`

Dit veld bestaat uit 4 deelvelden:

1 Name

Een optioneel veld van type “ProjectName”. (niet editable)

De bijbehorende Ampersand expressie is:

`projectName`

2 Current PL

Een lijst van 0 of meer velden van type “Person”. (niet editable)

De bijbehorende Ampersand expressie is:

`pl`

3 Description

Een optioneel veld van type “Description”. (niet editable)

De bijbehorende Ampersand expressie is:

`projectDescription`

4 Administration

Een verplicht veld van type “Project”. (niet editable)

De bijbehorende Ampersand expressie is:

```
I[Project]
```

Dit veld bestaat uit 2 deelvelden:

4.1 Project leaders

Een lijst van 0 of meer velden van type “Person”. (niet editable)

De bijbehorende Ampersand expressie is:

```
project~;assignee /\ pl
```

Elk veld bestaat uit 3 deelvelden:

4.1.1 Name

Een optioneel veld van type “PersonName”. (niet editable)

De bijbehorende Ampersand expressie is:

```
personName
```

4.1.2 Status

Een optioneel veld van type “PersonStatus”. (niet editable)

De bijbehorende Ampersand expressie is:

```
personStatus
```

4.1.3 Email

Een optioneel veld van type “Email”. (niet editable)

De bijbehorende Ampersand expressie is:

```
personEmail
```

4.2 Project members

Een lijst van 0 of meer velden van type “Person”. (niet editable)

De bijbehorende Ampersand expressie is:

```
project~;assignee /\ member
```

Elk veld bestaat uit 3 deelvelden:

4.2.1 Name

Een optioneel veld van type “PersonName”. (niet editable)

De bijbehorende Ampersand expressie is:

```
personName
```

4.2.2 Status

Een optioneel veld van type “PersonStatus”. (niet editable)

De bijbehorende Ampersand expressie is:

```
personStatus
```

4.2.3 Email

Een optioneel veld van type “Email”. (niet editable)

De bijbehorende Ampersand expressie is:

```
personEmail
```

Chapter 11

Interface: “Person”

Dit hoofdstuk bevat de documentatie voor de interface “Person”.

De interface is beschikbaar voor alle rollen.

Voorafgaand aan het afsluiten van een transactie (commit), moet aan de volgende regels voldaan zijn:

- People are identifiable by their email-address

CRUD matrix:

Concept	C	R	U	D
Person		√		
Email	√	√		√
Assignment			√	
PersonName	√	√		√
PersonStatus		√		

Interfacestructuur:

Interface voor een waarde van type “Person”.

Een verplicht veld van type “Person”. (niet editable)

De bijbehorende Ampersand expressie is:

`I[Person]`

Dit veld bestaat uit 4 deelvelden:

1 Name

Een optioneel veld van type “PersonName”. (editable)

De bijbehorende Ampersand expressie is:

`personName`

2 Status

Een optioneel veld van type “PersonStatus”. (niet editable)

De bijbehorende Ampersand expressie is:

`personStatus`

3 Email

Een optioneel veld van type “Email”. (editable)

De bijbehorende Ampersand expressie is:

`personEmail`

4 Works with

Een lijst van 0 of meer velden van type “Person”. (niet editable)

De bijbehorende Ampersand expressie is:

```
worksswith
```