Z Specification for a Knowledge Management System

Shekar Sivasubramanian

CMU-LTI-16-002

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:
Dr. Eric Nyberg (Chair)
Dr. Jamie Callan
Dr. Robert Frederking
Kiran Hosakote

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
In Language and Information Technologies

©2016 Shekar Sivasubramanian

1. **Overview**
   The following document provides a Z specification of a Knowledge Management System (KMS). The specification provides composition of this system, along with a definition of the operators used in this system.

   The KMS is assumed to exist in the context of a software development organization. Such organizations do their work in the confines of a project. Projects have well defined time-frames, goals, and resources allocated to ensure completion. Resources include people who work on the project, modeled here as users. Projects have knowledge needs - such needs represent the need for artifacts or expertise that are useful for the completion of the project. Projects can create such artifacts by applying resources available to them. Alternately, they can use the KMS to determine if artifacts and expertise already exist to provide a head-start for the project. If the knowledge need is not fulfilled, the project will create the artifacts and can provide that to the KMS. Over time, the number of hits in the KMS will increase and the need for teams to keep creating artifacts can get displaced more efficiently by the KMS providing artifacts and expertise, within the context and domain of the software development organization. This may offer an economical and meaningful alternative to augment the traditional manner in which software development is approached, particularly in the context of large software organizations, where the commonality of engineering problems that need to be solved tend to be high, and the knowledge associated with the existence of a prior solution may be low. In addition, the KMS can promote greater focus and an economical viability to the application of expertise and work done in informal communities. The organization is mapped in line with contemporary concepts of a knowledge management. Typically, there are designated experts in an organization who are tracked. Likewise, the concept of Communities of Practice is captured with users and experts enrolling in such communities. Such Communities of Practice form one of the most important organizational entities to enrich knowledge in an informal context. Experts and Communities can review and enrich the artifacts in the KMS in an unsolicited manner. Knowledge enrichment can thus be captured in the system, and can offer nuances on a set of artifact. All knowledge items have an inherent characteristic that is an important thread to navigate across such items - the classification scheme. The classification scheme is kept simple, and used to classify artifacts, needs, experts, and communities. The classification scheme offers the flexibility to the users of the KMS to create a comprehensive, organization-wide ontology, or a very localized ontology for a specific phase of a project or part of a system.

   The KMS must have the ability to add and delete information in the system to ensure that the system can be maintained with accurate information. The KMS has various questions of interest that will help the software development organization:

   - Give knowledge artifacts in a project.
   - Give project needs.
   - Give one or more experts who can address my knowledge need.
   - Give one or more users who can address my knowledge need.

- Give one or more artifacts that can address my knowledge need.

This document provides the structural definition of the KMS, all primary data manipulation operations and specifications for the different types of queries. Questions for the future include the following. Some of these may require extending the Z specification of the KMS:

- Give an area where the maximum knowledge activity is taking place. This would represent artifacts the undergo the greatest number of enrichments and communities with the greatest number of enrollments.
- Give communities that have the highest enrollment of experts.
- Give knowledge artifacts that fulfill my project need that are assessed by a specific user at a specific level.
- At the completion of the project, all project needs must be fulfilled by the KMS. This implies that the need and items are classified in the KMS after the completion of the project.
- Give the most active communities. Such a query can help the organization understand what promotes activity in communities to emulate in other areas.
- Create a list of common needs across projects and their corresponding fulfillments.

2. **Type Definition**
The following are type definitions used by the Knowledge Management System.

$[DATE, TIME, TEXT, PASSWORD]$

$EXPERT\_CLASS ::= high \mid medium \mid low \mid none$

$NEED\_TYPE ::= artifact \mid community \mid user \mid expert$

$USER\_TYPE ::= normal \mid administrator$

$ARTIFACT\_RATING ::= excellent \mid useful \mid moderate$

$RETURN ::= OK \mid USER\_NOT\_FOUND \mid PROJECT\_NOT\_FOUND \mid$
$NEED\_NOT\_FOUND \mid ARTIFACT\_NOT\_FOUND \mid ASSESSMENT\_NOT\_FOUND \mid$
$COMMUNITY\_NOT\_FOUND \mid CLASSIFICATION\_NOT\_FOUND \mid$
$USER\_ALREADY\_AN\_EXPERT \mid ENRICHMENT\_NOT\_FOUND \mid$
$USER\_NOT\_AN\_EXPERT \mid EXPERT\_CLASS\_NONE \mid$
$PROJECT\_HAS\_KNOWLEDGE\_NEEDS \mid$
$PROJECT\_HAS\_NO\_KNOWLEDGE\_NEEDS \mid$
$KNOWLEDGE\_NEED\_NOT\_CLASSIFIED \mid$
$NO\_USER\_FOUND\_FOR\_KNOWLEDGE\_NEED \mid$
$NO\_EXPERT\_FOUND\_FOR\_KNOWLEDGE\_NEED \mid$
$PROJECT\_HAS\_NO\_USERS \mid$
$NO\_ARTIFACT\_FOUND\_FOR\_KNOWLEDGE\_NEED \mid$
$NO\_COMMUNITY\_FOUND\_FOR\_KNOWLEDGE\_NEED$

- [*TIME*]: This is a primitive type for time stamp that includes the date and time.
- [*TEXT*]: This is a descriptive text item that contains one or more characters.
- [*PASSWORD*]: This is a primitive type for an encrypted password.
- *BOOLEAN*: This is a generic boolean indicator.
- *EXPERT_CLASS*: This represents classification for a knowledge expert. *high* denotes a person with higher expertise, while *medium* is a person of moderate expertise, and *low* is a person of low expertise. *none* indicates that the fact that the user is not an expert.
- *NEED_TYPE*: This represents the type of need - it represents the desire to fulfill the need with a certain type of entity - which could be an artifact, an expert, a user, or a community.
- *USER_TYPE*: This represents different types of users of the system: *normal* is a routine user of the system, while *administrator* is an administrator of the system.
- *ARTIFACT_RATING*: This represents rating of an artifact, based on its assessment by a user. *excellent* denotes a high rating, while *useful* a moderate rating, while *moderate* denotes a low rating of the artifact.
- *RETURN*: Contains the return values for ensuring a robust system.
  - OK: Means that the operation was correct.
  - USER_NOT_FOUND: Implies that the user was not found.
  - PROJECT_NOT_FOUND: Implies that the project was not found.
  - NEED_NOT_FOUND: Implies that the knowledge need was not found for the project.
  - NEED_TYPE_NOT_ARTIFACT: Implies that the need type requested is not an artifact.
  - ARTIFACT_NOT_FOUND: Implies that the knowledge artifact was not found.
  - ASSESSMENT_NOT_FOUND: Implies that the knowledge assessment was not found.
  - COMMUNITY_NOT_FOUND: Implies that the community was not found.
  - CLASSIFICATION_NOT_FOUND: Implies that the classification was not found.
  - USER_ALREADY_AN_EXPERT: Implies that user is already designated as an expert.
  - USER_NOT_AN_EXPERT: Implies that user is not designated as an expert.
  - EXPERT_CLASS_NONE: Implies that the expert class has been entered as none.
  - ENRICHMENT_NOT_FOUND: Implies that the enrichment has not been found for the artifact.
  - PROJECT_HAS_KNOWLEDGE_NEEDS: Implies that the project still has knowledge needs associated with it.

- PROJECT_HAS_NO_KNOWLEDGE_NEEDS: Implies that the project has no knowledge needs associated with it.
- KNOWLEDGE_NEED_NOT_CLASSIFIED: Implies that the knowledge need for the project has not been classified.
- NO_USER_FOUND_FOR_KNOWLEDGE_NEED: Implies that no user has been found for a knowledge need.
- NO_EXPERT_FOUND_FOR_KNOWLEDGE_NEED: Implies that no user has been found for a knowledge need.
- PROJECT_HAS_NO_USERS: No users are in the project.
- NO_ARTIFACT_FOUND_FOR_KNOWLEDGE_NEED: Implies that no artifact has been found for a knowledge need.
- NO_COMMUNITY_FOUND_FOR_KNOWLEDGE_NEED: Implies that no community has been found for a knowledge need.

3. **Basic definitions**

(a) **Basic definition of** *USER*
The following is the definition of *USER* which defines the user in the knowledge management system.

USER

$$
\begin{aligned}
&user\_id : \mathbb{N} \\
&user\_name : TEXT \\
&password : PASSWORD \\
&user\_type : USER\_TYPE \\
&expert\_class : EXPERT\_CLASS \\
&create\_time : TIME
\end{aligned}
$$

The *USER* schema defines a user. Note that for the rest of this specification we will assume that no two elements of the set of all possible users are the same. This will be true since the id will be unique. The uses of the individual fields are as follows:

- **user_id** : a unique identifier. This will be used to identify user.
- **user_name** : is the name of the user.
- **user_type** : classifies the user as a *normal* user or *administrator*.
- **expert_class** : classifies the expert as a *high*, *medium*, *low* or *none*. *none* denotes that the user is not an expert.
- **create_time** : stores the time when the user information was created.

(b) **Basic definition of** *NEED*
The following is the definition of *NEED* which defines a knowledge need.

NEED

$need\_id : \mathbb{N}$
description : *TEXT*
need_type : *NEED_TYPE*
create_time : *TIME*

The *NEED* schema defines a need for knowledge. Note that for the rest of this specification we will assume that no two elements of the set of all possible needs are the same for a project. This will be true since the id will be unique within the context of a project. The uses of the individual fields are as follows:

- **need_id** : a unique identifier. This will be used to identify a knowledge need.
- **need_type** : a discrete set of values representing different types of needs requested - could be one of *artifact*, *expert*, *user*, or *community*.
- **description** : describes the knowledge need.
- **create_time** : stores the time when the knowledge need was created.

(c) **Basic definition of** *PROJECT*
The following is the definition of *PROJECT*, that is a placeholder for the definition of a project in the knowledge management system.

PROJECT

$project\_id : \mathbb{N}$
project_name : *TEXT*
description : *TEXT*
create_time : *TIME*
start_date : *DATE*
end_date : *DATE*

The *PROJECT* schema defines a project. Note that for the rest of this specification we will assume that no two elements of the set of all possible projects are the same. This will be true since the id will be unique. The uses of the individual fields are as follows:

- **project_id** : a unique identifier. This will be used to identify a project.
- **project_name** : is the name of the project.
- **description** : describes the project.
- **create_time** : stores the time when the project information was created.
- **start_date** : stores the start date of the project.
- **end_date** : stores the end date of the project.

(d) **Basic definition of** *ENRICHMENT*
The following is the definition of *ENRICHMENT* which defines information on enrichment of the artifacts.

ENRICHMENT

$enrichment\_id : \mathbb{N}$
$description : TEXT$
$create\_time : TIME$

The *ENRICHMENT* schema defines information on enrichment of an artifact. Note that for the rest of this specification we will assume that no two elements of the set of all possible enrichments are the same. This will be true since the id will be unique. The uses of the individual fields are as follows:

- **enrichment_id** : a unique identifier. This will be used to identify an enrichment.
- **description** : describes the enrichment.
- **create_time** : stores the time when the project information was created.

(e) **Basic definition of** *ARTIFACT*
The following is the definition of *ARTIFACT*, one of the basic units in the Knowledge Management System. An artifact stores information about a single artifact.

ARTIFACT

$artifact\_id : \mathbb{N}$
$artifact\_name : TEXT$
$description : TEXT$
$Enrichments : \mathbb{P}\, ENRICHMENT$
$location : TEXT$
$create\_time : TIME$

An artifact is a basic unit of knowledge that is stored in a knowledge management system. Note that for the rest of this specification we will assume that no two elements of the set of all possible artifacts are the same. This will be true since the id will be unique. The uses of the individual fields are as follows:

- **artifact_id** : a unique identifier. This will be used to identify an artifact.
- **artifact_name** : is the name of the artifact.
- **description** : is a description of the artifact.
- **Enrichments** : is the set of Enrichments that for the artifact.
- **location** : stores the location of the artifact. This is a reference to the location of the artifact. This has been kept open-ended for the purposes of the model as a *TEXT* type.
- **create_time** : stores the time when the artifact has been created.

(f) **Basic definition of** *ASSESSMENT*
The following is the definition of *ASSESSMENT*, which store information associated with an assessment of an artifact.

ASSESSMENT

$$
\begin{array}{l}
assessment\_id : \mathbb{N} \\
description : TEXT \\
artifact\_rating : ARTIFACT\_RATING \\
create\_time : TIME
\end{array}
$$

An assessment provides information on the quality of the artifact as seen by a user. Note that for the rest of this specification we will assume that no two elements of the set of all possible assessments are the same. This will be true since the id will be unique. The uses of the individual fields are as follows:

- **assessment_id** : a unique identifier. This will be used to identify an artifact.
- **description** : is a description of the artifact.
- **artifact_rating** : denotes the rating of the artifact.
- **create_time** : stores the time when the artifact has been created.

(g) **Basic definition of** *COMMUNITY*
The following is the definition of *COMMUNITY* which defines information on a community of practice.

COMMUNITY

$$
\begin{array}{l}
community\_id : \mathbb{N} \\
community\_name : TEXT \\
description : TEXT \\
create\_time : TIME
\end{array}
$$

The *COMMUNITY* schema defines information on a community. A community represents a community of practice in a knowledge management context. Note that for the rest of this specification we will assume that no two elements of the set of all possible enrichments are the same. This will be true since the id will be unique. The uses of the individual fields are as follows:

- **community_id** : a unique identifier. This will be used to identify a community.
- **community_name** : Name of the community.
- **description** : describes the community.
- **create_time** : stores the time when the project information was created.

(h) **Basic definition of** *CLASSIFICATION*
The following is the definition of *CLASSIFICATION* which defines a knowledge classification. A knowledge classification is the fundamental language of the system. For the current case, this has been abstracted as a simple representation of a unique id and a description. This can be extended to represent a complete ontology.

CLASSIFICATION

$$classification\_id : \mathbb{N}$$
$$description : TEXT$$
$$create\_time : TIME$$

The *CLASSIFICATION* schema defines a classification for the knowledge artifacts. Note that for the rest of this specification we will assume that no two elements of the set of all possible enrichments are the same. This will be true since the id will be unique. The uses of the individual fields are as follows:

- **classification_id** : a unique identifier. This will be used to identify a classification.
- **description** : describes the knowledge classification.
- **time_stamp** : stores the time when the classification information was created.

4. **Definition of Building Blocks**

   (a) **Definition of Knowledge Users**
   The following is the definition of the Knowledge Users. Knowledge Users represent all the people who use the KMS. They include normal and administrative users of the system.

Knowledge_Users

$$Users : \mathbb{P}\, USER$$

$$\forall m, n : Users \bullet m.user\_id = n.user\_id \Leftrightarrow m = n$$

The following applies to the *Knowledge_Users* schema.

- Users is a set of *USER*.
- No two users have the same id.

   (b) **Definition of Software Projects**
   The following is the definition of Software Projects. Software Projects is the abstraction to capture the domain of software projects in the context of a GDC. All projects that are performed in a GDC have a representation in this schema.

Software_Projects

$$Projects : \mathbb{P}\, PROJECT$$

$$\forall m, n : Projects \bullet m.project\_id = n.project\_id \Leftrightarrow m = n$$

The following applies to the *Software_Projects* Schema.

- Projects is a set of *PROJECT*.
- No two projects have the same id.

(c) **Definition of Knowledge Needs**
The following is the definition of the Knowledge Needs. Knowledge Needs represent needs for different aspects of knowledge in a project. A knowledge need forms the principal source of knowledge search for a project.

Knowledge_Needs

$$Needs : \mathbb{P}\, NEED$$
$$Projects : \mathbb{P}\, PROJECT$$
$$Project\_Has\_Needs : PROJECT \leftrightarrow NEED$$

$$\forall m, n : Needs \bullet m.need\_id = n.need\_id \Leftrightarrow m = n$$
$$\mathrm{dom}\, Project\_Has\_Needs \subseteq Projects$$
$$\mathrm{ran}\, Project\_Has\_Needs \subseteq Needs$$

The following applies to the *Knowledge_Needs* Schema.

- Needs is a set of *NEED*.
- No two needs have the same id.
- **Project_Has_Needs** : This model assumes that the knowledge management system supports knowledge needs of an existing project-based software organization. Each knowledge need arises from a project. One project may have many needs. There may be projects that have no needs. The same need does not arise from two projects. This is an accurate representation of a software organization, which may have needs from a project that has no knowledge of related needs in other projects. That may be well be a subsequent discovery process that the KMS must attempt to do.

(d) **Definition of Knowledge Artifacts**
The following is the definition of the Knowledge Artifacts. Knowledge Artifacts represents the electronic representation of all software artifacts of interest in a GDC. Whenever an artifact of interest is created in a software project, this artifact will be stored in the KMS for future reference. Enrichments to the artifact are also stored as part of this schema, for each software artifact.

Knowledge_Artifacts

$$Artifacts : \mathbb{P}\, ARTIFACT$$

$$\forall m, n : Artifacts \bullet m.artifact\_id = n.artifact\_id \Leftrightarrow m = n$$

The following applies to the *Knowledge_Artifacts* schema.

- Artifacts is a set of *ARTIFACT*.
- No two artifacts have the same id.

(e) **Definition of Knowledge Assessments**
The following is the definition of the Knowledge Assessment. Knowledge assessment represents the assessment of knowledge that is performed in the system.

Knowledge_Assessments

$$
\begin{array}{l}
Artifacts : \mathbb{P}\, ARTIFACT \\
Users : \mathbb{P}\, USER \\
Assessments : \mathbb{P}\, ASSESSMENT \\
Assessed\_By\_User : ASSESSMENT \leftrightarrow USER \\
Assessment\_Of\_Artifact : ASSESSMENT \leftrightarrow ARTIFACT \\
\hline
\forall\, m, n : Assessments \bullet m.assessment\_id = n.assessment\_id \Leftrightarrow m = n \\
\mathrm{dom}\, Assessed\_By\_User \subseteq Assessments \\
\mathrm{ran}\, Assessed\_By\_User \subseteq Users \\
\mathrm{dom}\, Assessment\_Of\_Artifact \subseteq Assessments \\
\mathrm{ran}\, Assessment\_Of\_Artifact \subseteq Artifacts
\end{array}
$$

The following applies to the *Knowledge_Assessment* Schema.

- Assessments is a set of *ASSESSMENT*.
- No two assessments have the same id.
- **Assessed_By_User** : Provides the assessment by a user.
- **Assessment_Of_Artifact** : Provides the assessment of an artifact.

(f) **Definition of Communities of Practice**
The following is the definition of Communities of Practice. Communities of Practice form informal networks of people who collaborate together on areas of mutual interest. It is widely recognized that such communities form an integral part to knowledge enrichment and distribution. This schema provides the abstraction for the Communities of Practice.

Communities_Of_Practice

$$
\begin{array}{l}
Communities : \mathbb{P}\, COMMUNITY \\
\hline
\forall\, m, n : Communities \bullet m.community\_id = n.community\_id \Leftrightarrow m = n
\end{array}
$$

The following applies to the *Communities_Of_Practice* schema.

- Communities is a set of *COMMUNITY*.
- No two communities have the same id.

(g) **Definition of User Enrollments**
The following is the definition of User Enrollments. User Enrollments captures the enrollment of users to communities and projects. This also captures the relation associated with users who create software artifacts.

User_Enrollments

$$
\begin{array}{l}
Knowledge\_Users \\
Software\_Projects \\
Knowledge\_Artifacts \\
Communities\_Of\_Practice \\
User\_Enrolls\_In\_Communities : USER \leftrightarrow COMMUNITY \\
User\_Enrolls\_In\_Projects : USER \leftrightarrow PROJECT \\
User\_Creates\_Artifacts : USER \leftrightarrow ARTIFACT \\
\hline
\mathrm{dom}\ User\_Enrolls\_In\_Communities \subseteq Users \\
\mathrm{ran}\ User\_Enrolls\_In\_Communities \subseteq Communities \\
\mathrm{dom}\ User\_Enrolls\_In\_Projects \subseteq Users \\
\mathrm{ran}\ User\_Enrolls\_In\_Projects \subseteq Projects \\
\mathrm{dom}\ User\_Creates\_Artifacts \subseteq Users \\
\mathrm{ran}\ User\_Creates\_Artifacts \subseteq Artifacts
\end{array}
$$

The following applies to the *User_Enrollments* schema.

- **User_Enrolls_In_Communities** : A user can belong to more than one community and a community can consist of several users.
- **User_Enrolls_In_Projects** : A user can belong to more than one project and a project can consist of several users.
- **User_Creates_Artifacts**: One or more users creates an artifact. Users can create multiple artifacts.

(h) **Definition of Ontology**

The following is the definition of Ontology. The Ontology provides the basic classification schema for an artifact. For this representation, the classification is represented by a unique id and a description. Different alternatives exist to design and implement a classification schema. Of interest here is the fact that the classification is uniquely identifiable, and will be used to classify different components of the KMS.

Ontology

$$
\begin{array}{l}
Classifications : \mathbb{P}\, CLASSIFICATION \\
\hline
\forall m, n : Classifications \bullet m.classification\_id = n.classification\_id \Leftrightarrow m = n
\end{array}
$$

The following applies to the *Ontology* schema.

- Classifications is a set of *CLASSIFICATION*.
- No two classifications have the same id.

(i) **Definition of Item Classifications**

The following is the definition of Item Classifications. Item Classifications provides

the ability to classify project needs, software artifacts, users, and communities using the same classification scheme.

Item_Classifications

$$
\begin{array}{l}
Knowledge\_Users \\
Communities\_Of\_Practice \\
Knowledge\_Artifacts \\
Knowledge\_Needs \\
Ontology \\
Artifact\_Is\_Classified : ARTIFACT \leftrightarrow CLASSIFICATION \\
User\_Is\_Classified : USER \leftrightarrow CLASSIFICATION \\
Need\_Is\_Classified : NEED \leftrightarrow CLASSIFICATION \\
Community\_Is\_Classified : COMMUNITY \leftrightarrow CLASSIFICATION \\
\hline
\mathrm{dom}\, Artifact\_Is\_Classified \subseteq Artifacts \\
\mathrm{ran}\, Artifact\_Is\_Classified \subseteq Classifications \\
\mathrm{dom}\, User\_Is\_Classified \subseteq Users \\
\mathrm{ran}\, User\_Is\_Classified \subseteq Classifications \\
\mathrm{dom}\, Need\_Is\_Classified \subseteq Needs \\
\mathrm{ran}\, Need\_Is\_Classified \subseteq Classifications \\
\mathrm{dom}\, Community\_Is\_Classified \subseteq Communities \\
\mathrm{ran}\, Community\_Is\_Classified \subseteq Classifications
\end{array}
$$

The following applies to the *Item_Classifications* schema.

- **Artifact_Is_Classified** : An artifact can have multiple classifications. A specific classification can apply to one or more artifacts.
- **User_Is_Classified** : A user can have multiple classifications, each of which represents the area of interest or expertise. A specific classification can apply to one or more users.
- **Need_Is_Classified** : A need can be classified in multiple ways. A specific classification can apply to one or more needs. The classification of the need allows the system to formally map the need to a uniform classification that can be used to locate artifacts and people.
- **Community_Is_Classified** : A community is classified to address multiple areas. One classification can apply to multiple communities.

5. **Definition of Knowledge_Management_System** The following is the definition of the Knowledge_Management_System. The representation has been devised to enhance readability and limit the scope of operations to relevant areas.

KMS

Knowledge_Users
Software_Projects
Knowledge_Needs
Knowledge_Artifacts
Knowledge_Assessments
Communities_Of_Practice
User_Enrollments
Ontology
Item_Classifications

The following are the components of the Knowledge Management System.

- Knowledge_Users represents users of the system.
- Software_Projects represents projects in the system.
- Knowledge_Needs represents project needs in the system.
- Knowledge_Artifacts represent the basic unit of knowledge in the system.
- Knowledge_Assessments represent assessment of knowledge in the system.
- Communities_Of_Practice represent communities of practice.
- User_Enrollments represents users enrolled in communities.
- Ontology provides the classification ontology for the knowledge management system.
- Item_Classifications provides classifications applied to different parts of the knowledge management system including artifacts, needs, communities, and users.

6. **Initialization Operation** The following is the initialization of the Knowledge_Management_System.

InitKMS

ΔKMS

$Users = \emptyset$
$Projects = \emptyset$
$Artifacts = \emptyset$
$Communities = \emptyset$
$Classifications = \emptyset$
$Assessments = \emptyset$
$Project\_Has\_Needs = \emptyset$
$User\_Enrolls\_In\_Communities = \emptyset$
$User\_Enrolls\_In\_Projects = \emptyset$
$User\_Creates\_Artifacts = \emptyset$
$Assessed\_By\_User = \emptyset$
$Assessment\_Of\_Artifact = \emptyset$
$Artifact\_Is\_Classified = \emptyset$
$User\_Is\_Classified = \emptyset$
$Need\_Is\_Classified = \emptyset$
$Community\_Is\_Classified = \emptyset$

This operation initializes the system. The initial state of the system contains no data. Therefore, all the sets and relations have been initialized to empty sets.

7. **Manipulation Operations**

   (a) **Operation: Add a user**

AddUser

$\Delta Knowledge\_Users$
$un? : TEXT$
$pw? : PASSWORD$
$ut? : USER\_TYPE$
$ts? : TIME$

$Users' = Users \cup \{n : USER \mid n.user\_id = \#Users + 1$
$\land\ n.user\_name = un? \land n.password = pw?$
$\land\ n.user\_type = ut? \land n.create\_time = ts? \land n.expert\_class = none\}$

This operation adds a user in the system. Note that a unique user id is created for the user as part of this operation. The expert class for the user is set to *none* to denote that the user is not an expert.

   (b) **Operation: Designate a user as an expert**

DesignateUserAsExpert

$\Delta Knowledge\_Users$
$user : USER$
$uid? : \mathbb{N}$
$ec? : EXPERT\_CLASS$

$\exists m : Users \bullet m.user\_id = uid? \land m.expert\_class = none$
$ec? \neq none$
$user = (\mu m : Users \mid m.user\_id = uid?)$
$user.expert\_class = ec? \land user.user\_id = user.user\_id$
$\land\ user.user\_name = user.user\_name \land user.password = user.password$
$\land\ user.user\_type = user.user\_type \land user.create\_time = user.create\_time$
$Users' = Users \setminus \{n : USER \mid n.user\_id = uid?\} \cup \{user\}$

This operation designates a user as an expert. The entered id must be a valid user id in the system. The user must not already be an expert. The specific expert class is assigned to the user. The operation is valid only if the entered expert class is not *none*.

(c) **Operation: Add a project**

AddProject

$\Delta Software\_Projects$
$pn? : TEXT$
$pd? : TEXT$
$ts? : TIME$
$sd? : DATE$
$ed? : DATE$

$Projects' = Projects \cup \{n : PROJECT \mid n.project\_id = \#Projects + 1$
$\land\ n.project\_name = pn? \land n.description = pd? \land n.start\_date = sd? \land$
$n.end\_date = ed? \land n.create\_time = ts?\}$

This operation adds a project in the system. Note that a unique project id is created for the project as part of this operation. An empty set of needs is created for the project.

(d) **Operation: Establish need for project**

EstablishNeedForProject

$\Delta Knowledge\_Needs$
$need : NEED$
$project : PROJECT$
$pid? : \mathbb{N}$
$nd? : TEXT$
$nt? : NEED\_TYPE$
$ts? : TIME$

$project = (\mu\, m : Projects \mid m.project\_id = pid?)$
$need = (\mu\, m : Needs \mid m.need\_id = \#Needs + 1)$
$Needs' = Needs \cup \{n : NEED \mid n.need\_id = \#Needs + 1$
$\land\ n.description = nd? \land n.need\_type = nt? \land n.create\_time = ts?\}$
$Project\_Has\_Needs' =$
$Project\_Has\_Needs \cup \{project \mapsto need\}$
$Projects' = Projects$

This operation establishes a knowledge need for a project. The entered project id must be a valid project id in the system. A knowledge need id established for the project.

(e) **Operation: Add an artifact**

AddArtifact

$\Delta User\_Enrollments$
$an? : TEXT$
$ad? : TEXT$
$ts? : TIME$
$user : USER$
$artifact : ARTIFACT$
$uid? : \mathbb{N}$

$user = (\mu\, m : Users \mid m.user\_id = uid?)$
$artifact = (\mu\, m : Artifacts \mid m.artifact\_id = \#Artifacts + 1)$
$Artifacts' = Artifacts \cup \{n : ARTIFACT \mid n.artifact\_id = \#Artifacts + 1$
$\land\ n.artifact\_name = an? \land n.description = ad? \land n.create\_time = ts?$
$\land\ n.Enrichments = \emptyset\}$
$User\_Creates\_Artifacts' =$
$User\_Creates\_Artifacts \cup \{user \mapsto artifact\}$
$User\_Enrolls\_In\_Communities' = User\_Enrolls\_In\_Communities$
$User\_Enrolls\_In\_Projects' = User\_Enrolls\_In\_Projects$
$Communities' = Communities$
$Projects' = Projects$
$Users' = Users$

This operation adds an artifact. Note that a unique artifact id is created for the artifact as part of this operation. An empty set of *Enrichments* is added to the project. The artifact is associated with the user who created this artifact. The user must be a valid user in the system.

(f) **Operation: Enrich Artifact**

EnrichArtifact

$$
\begin{array}{l}
\Delta Knowledge\_Artifacts \\
artifact : ARTIFACT \\
aid? : \mathbb{N} \\
nd? : TEXT \\
ts? : TIME \\
\hline
artifact = (\mu\, m : Artifacts \mid m.artifact\_id = aid?) \\
artifact.Enrichments = artifact.Enrichments \cup \\
\{n : ENRICHMENT \mid n.enrichment\_id = \#artifact.Enrichments + 1 \\
\land\ n.description = nd? \land n.create\_time = ts?\} \\
Artifacts' = Artifacts \setminus \{n : ARTIFACT \mid n.artifact\_id = aid?\} \cup \{artifact\}
\end{array}
$$

This operation enriches an artifact. The artifact id must be a valid id. A new enrichment information is added to the set of enrichments that forms part of the artifact.

(g) **Operation: Assess Artifact**

AssessArtifact

$$
\begin{array}{l}
\Delta Knowledge\_Assessments \\
user : USER \\
artifact : ARTIFACT \\
assessment : ASSESSMENT \\
uid? : \mathbb{N} \\
aid? : \mathbb{N} \\
ad? : TEXT \\
ar? : ARTIFACT\_RATING \\
ts? : TIME \\
\hline
user = (\mu\, m : Users \mid m.user\_id = uid?) \\
artifact = (\mu\, m : Artifacts \mid m.artifact\_id = aid?) \\
assessment = (\mu\, m : Assessments \mid m.assessment\_id = \#Assessments + 1) \\
Assessments' = Assessments \cup \{n : ASSESSMENT \mid n.assessment\_id = \#Assessments + 1 \\
\land\ n.description = ad? \land n.artifact\_rating = ar? \land n.create\_time = ts?\} \\
Assessed\_By\_User' = \\
Assessed\_By\_User \cup \{assessment \mapsto user\} \\
Assessment\_Of\_Artifact' = \\
Assessment\_Of\_Artifact \cup \{assessment \mapsto artifact\}
\end{array}
$$

This operation assesses a knowledge artifact. The entered user id must be a valid user id in the system. The entered artifact id must be a valid one for the system. After adding the assessment information, a relation is established between the assessment and the user, as well as the assessment and the artifact.

(h) **Operation: Add a community**

AddCommunity

$$\Delta Communities\_Of\_Practice$$
$$cn? : TEXT$$
$$cd? : TEXT$$
$$ts? : TIME$$

$$Communities' = Communities \cup$$
$$\{n : COMMUNITY \mid n.community\_id = \#Communities + 1$$
$$\land\ n.community\_name = cn? \land n.description = cd? \land n.create\_time = ts?\}$$

This operation adds a community. Note that a unique community id is created for the community as part of this operation.

(i) **Operation: Enroll User in Community**

EnrollUserInCommunity

$$\Delta User\_Enrollments$$
$$user : USER$$
$$community : COMMUNITY$$
$$uid? : \mathbb{N}$$
$$cid? : \mathbb{N}$$

$$user = (\mu\, m : Users \mid m.user\_id = uid?)$$
$$community = (\mu\, m : Communities \mid m.community\_id = cid?)$$
$$User\_Enrolls\_In\_Communities' =$$
$$User\_Enrolls\_In\_Communities \cup \{user \mapsto community\}$$
$$User\_Enrolls\_In\_Projects' = User\_Enrolls\_In\_Projects$$
$$User\_Creates\_Artifacts' = User\_Creates\_Artifacts$$
$$Communities' = Communities$$
$$Projects' = Projects$$
$$Users' = Users$$

This operation enrolls a user into an community. The user id and community id must both be valid for this operation.

(j) **Operation: Enroll User in Project**

$$
\begin{array}{|l}
\hline
\textit{EnrollUserInProject} \\
\Delta \textit{User\_Enrollments} \\
\textit{user} : \textit{USER} \\
\textit{project} : \textit{PROJECT} \\
\textit{uid}? : \mathbb{N} \\
\textit{pid}? : \mathbb{N} \\
\hline
\textit{user} = (\mu\, m : \textit{Users} \mid m.\textit{user\_id} = \textit{uid}?) \\
\textit{project} = (\mu\, m : \textit{Projects} \mid m.\textit{project\_id} = \textit{pid}?) \\
\textit{User\_Enrolls\_In\_Projects}' = \\
\textit{User\_Enrolls\_In\_Projects} \cup \{\textit{user} \mapsto \textit{project}\} \\
\textit{User\_Enrolls\_In\_Communities}' = \textit{User\_Enrolls\_In\_Communities} \\
\textit{User\_Creates\_Artifacts}' = \textit{User\_Creates\_Artifacts} \\
\textit{Communities}' = \textit{Communities} \\
\textit{Projects}' = \textit{Projects} \\
\textit{Users}' = \textit{Users} \\
\hline
\end{array}
$$

This operation enrolls a user into a project. The user id and project id must both be valid for this operation.

(k) **Operation: Add a classification**

$$
\begin{array}{|l}
\hline
\textit{AddClassification} \\
\Delta \textit{Ontology} \\
\textit{cd}? : \textit{TEXT} \\
\textit{ts}? : \textit{TIME} \\
\hline
\textit{Classifications}' = \textit{Classifications} \cup \\
\{n : \textit{CLASSIFICATION} \mid n.\textit{classification\_id} = \#\textit{Classifications} + 1 \\
\land\ n.\textit{description} = \textit{cd}? \land n.\textit{create\_time} = \textit{ts}?\} \\
\hline
\end{array}
$$

This operation adds a classification. Note that a unique classification id is created for the classification as part of this operation.

(l) **Operation: Classify artifact**

ClassifyArtifact

$\Delta Item\_Classifications$
$artifact : ARTIFACT$
$classification : CLASSIFICATION$
$aid? : \mathbb{N}$
$cid? : \mathbb{N}$

$artifact = (\mu\, m : Artifacts \mid m.artifact\_id = aid?)$
$classification = (\mu\, m : Classifications \mid m.classification\_id = cid?)$
$Artifact\_Is\_Classified' = Artifact\_Is\_Classified \cup \{artifact \mapsto classification\}$
$Classifications' = Classifications$
$Needs' = Needs$
$Projects' = Projects$
$Artifacts' = Artifacts$
$Users' = Users$
$Communities' = Communities$
$Project\_Has\_Needs' = Project\_Has\_Needs$
$User\_Is\_Classified' = User\_Is\_Classified$
$Need\_Is\_Classified' = Need\_Is\_Classified$
$Community\_Is\_Classified' = Community\_Is\_Classified$

This operation classifies the artifact. The artifact id and classification id must both be valid for this operation.

(m) **Operation: Classify need**

ClassifyNeed

$\Delta Item\_Classifications$
$need : NEED$
$classification : CLASSIFICATION$
$nid? : \mathbb{N}$
$cid? : \mathbb{N}$

$need = (\mu\, m : Needs \mid m.need\_id = nid?)$
$classification = (\mu\, m : Classifications \mid m.classification\_id = cid?)$
$Need\_Is\_Classified' = Need\_Is\_Classified \cup \{need \mapsto classification\}$
$Classifications' = Classifications$
$Needs' = Needs$
$Projects' = Projects$
$Artifacts' = Artifacts$
$Communities' = Communities$
$Project\_Has\_Needs' = Project\_Has\_Needs$
$User\_Is\_Classified' = User\_Is\_Classified$
$Artifact\_Is\_Classified' = Artifact\_Is\_Classified$
$Community\_Is\_Classified' = Community\_Is\_Classified$

This operation classifies the need for a project. The project id, the need id and classification id must both be valid for this operation.

(n) **Operation: Classify user**

$$
\begin{array}{|l}
\textit{ClassifyUser} \\
\hline
\Delta \textit{Item\_Classifications} \\
\textit{user} : \textit{USER} \\
\textit{classification} : \textit{CLASSIFICATION} \\
\textit{uid}? : \mathbb{N} \\
\textit{cid}? : \mathbb{N} \\
\hline
\textit{user} = (\mu\, m : \textit{Users} \mid m.\textit{user\_id} = \textit{uid}?) \\
\textit{classification} = (\mu\, m : \textit{Classifications} \mid m.\textit{classification\_id} = \textit{cid}?) \\
\textit{User\_Is\_Classified}' = \textit{User\_Is\_Classified} \cup \{\textit{user} \mapsto \textit{classification}\} \\
\textit{Classifications}' = \textit{Classifications} \\
\textit{Needs}' = \textit{Needs} \\
\textit{Projects}' = \textit{Projects} \\
\textit{Artifacts}' = \textit{Artifacts} \\
\textit{Users}' = \textit{Users} \\
\textit{Communities}' = \textit{Communities} \\
\textit{Project\_Has\_Needs}' = \textit{Project\_Has\_Needs} \\
\textit{Artifact\_Is\_Classified}' = \textit{Artifact\_Is\_Classified} \\
\textit{Need\_Is\_Classified}' = \textit{Need\_Is\_Classified} \\
\textit{Community\_Is\_Classified}' = \textit{Community\_Is\_Classified} \\
\hline
\end{array}
$$

This operation classifies the user. The user id and classification id must both be valid for this operation.

(o) **Operation: Classify community**

$$
\begin{array}{|l}
\textit{ClassifyCommunity} \\
\hline
\Delta \textit{Item\_Classifications} \\
\textit{community} : \textit{COMMUNITY} \\
\textit{classification} : \textit{CLASSIFICATION} \\
\textit{oid}? : \mathbb{N} \\
\textit{cid}? : \mathbb{N} \\
\hline
\textit{community} = (\mu\, m : \textit{Communities} \mid m.\textit{community\_id} = \textit{oid}?) \\
\textit{classification} = (\mu\, m : \textit{Classifications} \mid m.\textit{classification\_id} = \textit{cid}?) \\
\textit{Community\_Is\_Classified}' = \textit{Community\_Is\_Classified} \cup \{\textit{community} \mapsto \textit{classification}\} \\
\textit{Classifications}' = \textit{Classifications} \\
\textit{Projects}' = \textit{Projects} \\
\textit{Needs}' = \textit{Needs} \\
\textit{Artifacts}' = \textit{Artifacts} \\
\textit{Users}' = \textit{Users} \\
\textit{Communities}' = \textit{Communities} \\
\textit{Project\_Has\_Needs}' = \textit{Project\_Has\_Needs} \\
\textit{User\_Is\_Classified}' = \textit{User\_Is\_Classified} \\
\textit{Need\_Is\_Classified}' = \textit{Need\_Is\_Classified} \\
\textit{Artifact\_Is\_Classified}' = \textit{Artifact\_Is\_Classified} \\
\hline
\end{array}
$$

This operation classifies the community. The community id and classification id must both be valid for this operation.

8. **Delete Operations**

   (a) **Operation: Delete User**

$$
\begin{array}{|l}
\hline
\textit{DeleteUser} \\
\Delta KMS \\
user : USER \\
lassessment : \mathbb{P}\, ASSESSMENT \\
uid? : \mathbb{N} \\
\hline
user = (\mu\, m : Users \mid m.user\_id = uid?) \\
lassessment = Assessed\_By\_User^{\sim}(\!|\ \{user\}\ |\!) \\
User\_Enrolls\_In\_Communities' = \\
\{user\} \lhd User\_Enrolls\_In\_Communities \\
User\_Enrolls\_In\_Projects' = \\
\{user\} \lhd User\_Enrolls\_In\_Projects \\
User\_Is\_Classified' = \\
\{user\} \lhd User\_Is\_Classified \\
User\_Creates\_Artifacts' = \\
\{user\} \lhd User\_Creates\_Artifacts \\
Assessed\_By\_User' = Assessed\_By\_User \rhd \{user\} \\
Users' = Users \setminus \{user\} \\
Assessments' = Assessments \setminus lassessment \\
Projects' = Projects \\
Needs' = Needs \\
Artifacts' = Artifacts \\
Communities' = Communities \\
Classifications' = Classifications \\
Project\_Has\_Needs' = Project\_Has\_Needs \\
Artifact\_Is\_Classified' = Artifact\_Is\_Classified \\
Need\_Is\_Classified' = Need\_Is\_Classified \\
Community\_Is\_Classified' = Community\_Is\_Classified \\
\hline
\end{array}
$$

This operation deletes a user, provided the user is found. It removes the association with communities and projects that the user has enrolled into. The association with the created artifacts is removed. The association with assessment of artifacts is removed. The classification of the user is removed. Finally, the user is removed.

(b) **Operation: Remove expert designation from a user**

RemoveExpertDesignation

$\Delta Knowledge\_Users$
$user : USER$
$uid? : \mathbb{N}$

$user = (\mu\, m : Users \mid m.user\_id = uid? \land m.expert\_class \neq none)$
$user.expert\_class = none \land user.user\_id = user.user\_id$
$\land\ user.user\_name = user.user\_name \land user.password = user.password$
$\land\ user.user\_type = user.user\_type \land user.create\_time = user.create\_time$
$Users' = Users \setminus \{n : USER \mid n.user\_id = uid?\} \cup \{user\}$

This operation removes an expert designation from a user. The entered id is a valid user id in the system and is an expert.

(c) **Operation: Delete Need**

DeleteNeed

$\Delta Knowledge\_Needs$
$\Delta Item\_Classifications$
$need : NEED$
$nid? : \mathbb{N}$

$need = (\mu\, m : Needs \mid m.need\_id = nid?)$
$Need\_Is\_Classified' = \{need\} \lhd Need\_Is\_Classified$
$Project\_Has\_Needs' = Project\_Has\_Needs \rhd \{need\}$
$Needs' = Needs \setminus \{need\}$
$Projects' = Projects$
$Artifacts' = Artifacts$
$Users' = Users$
$Communities' = Communities$
$Classifications' = Classifications$
$User\_Is\_Classified' = User\_Is\_Classified$
$Artifact\_Is\_Classified' = Artifact\_Is\_Classified$
$Community\_Is\_Classified' = Community\_Is\_Classified$

This operation deletes a knowledge need for a project. The need must be associated with a valid project. Once the need is valid, the relationship to its classification is removed and the need is then deleted.

(d) **Operation: Delete Project**

$$
\begin{array}{|l}
\hline
\textit{DeleteProject} \\
\Delta \textit{Knowledge\_Needs} \\
\Delta \textit{Item\_Classifications} \\
\textit{project} : \textit{PROJECT} \\
\textit{Needset} : \mathbb{P}\, \textit{NEED} \\
\textit{pid}? : \mathbb{N} \\
\hline
\textit{project} = (\mu\, m : \textit{Projects} \mid m.\textit{project\_id} = \textit{pid}?) \\
\textit{Needset} = \textit{Project\_Has\_Needs}(\!|\ \{\textit{project}\}\ |\!) \\
\textit{Needs}' = \textit{Needs} \setminus \textit{Needset} \\
\textit{Need\_Is\_Classified}' = \textit{Needset} \lhd \textit{Need\_Is\_Classified} \\
\textit{Project\_Has\_Needs}' = \{\textit{project}\} \lhd \textit{Project\_Has\_Needs} \\
\textit{Projects}' = \textit{Projects} \setminus \{\textit{project}\} \\
\textit{Artifacts}' = \textit{Artifacts} \\
\textit{Users}' = \textit{Users} \\
\textit{Communities}' = \textit{Communities} \\
\textit{Classifications}' = \textit{Classifications} \\
\textit{User\_Is\_Classified}' = \textit{User\_Is\_Classified} \\
\textit{Artifact\_Is\_Classified}' = \textit{Artifact\_Is\_Classified} \\
\textit{Community\_Is\_Classified}' = \textit{Community\_Is\_Classified} \\
\hline
\end{array}
$$

This operation deletes all the needs and its associated classification. Then the project is deleted.

(e) **Operation: Delete Enrichment**

$$
\begin{array}{|l}
\hline
\textit{DeleteEnrichment} \\
\Delta \textit{Knowledge\_Artifacts} \\
\textit{artifact} : \textit{ARTIFACT} \\
\textit{aid}? : \mathbb{N} \\
\textit{eid}? : \mathbb{N} \\
\hline
\textit{artifact} = (\mu\, m : \textit{Artifacts} \mid m.\textit{artifact\_id} = \textit{aid}?) \\
\exists\, m : \textit{artifact.Enrichments} \bullet m.\textit{enrichment\_id} = \textit{eid}? \\
\textit{artifact.Enrichments} = \textit{artifact.Enrichments} \setminus \\
\{m : \textit{artifact.Enrichments} \mid m.\textit{enrichment\_id} = \textit{eid}?\} \\
\textit{Artifacts}' = \textit{Artifacts} \setminus \{n : \textit{ARTIFACT} \mid n.\textit{artifact\_id} = \textit{aid}?\} \cup \{\textit{artifact}\} \\
\hline
\end{array}
$$

This operation deletes a knowledge enrichment for an artifact. It removes the relations between the knowledge enrichment and the artifact, and removes the enrichment.

(f) **Operation: Delete Artifact**

DeleteArtifact

ΔKMS
$artifact : ARTIFACT$
$lassessment : \mathbb{P}\, ASSESSMENT$
$aid? : \mathbb{N}$

$artifact = (\mu\, m : Artifacts \mid m.artifact\_id = aid?)$
$lassessment = Assessment\_Of\_Artifact^{\sim}(\!|\ \{artifact\}\ |\!)$
$Artifact\_Is\_Classified' =$
$\{artifact\} \lhd Artifact\_Is\_Classified$
$User\_Creates\_Artifacts' = User\_Creates\_Artifacts \rhd \{artifact\}$
$Assessment\_Of\_Artifact' = Assessment\_Of\_Artifact \rhd \{artifact\}$
$Artifacts' = Artifacts \setminus \{artifact\}$
$Assessments' = Assessments \setminus lassessment$
$Users' = Users$
$Projects' = Projects$
$Needs' = Needs$
$Project\_Has\_Needs' = Project\_Has\_Needs$
$Communities' = Communities$
$User\_Enrolls\_In\_Communities' = User\_Enrolls\_In\_Communities$
$Classifications' = Classifications$
$Need\_Is\_Classified' = Need\_Is\_Classified$
$Community\_Is\_Classified' = Community\_Is\_Classified$
$User\_Is\_Classified' = User\_Is\_Classified$

This operation deletes an artifact. All classifications associated with artifact are removed. The user association is removed. The artifact assessment association is removed. The assessment is also removed. The enrichments are removed. Finally the artifact is removed.

(g) **Operation: Remove User Enrollment**

RemoveUserEnrollment

$\Delta User\_Enrollments$
$user : USER$
$community : COMMUNITY$
$uid? : \mathbb{N}$
$cid? : \mathbb{N}$

$user = (\mu\, m : Users \mid m.user\_id = uid?)$
$community = (\mu\, m : Communities \mid m.community\_id = cid?)$
$\{user \mapsto community\} \neq \emptyset$
$User\_Enrolls\_In\_Communities' = User\_Enrolls\_In\_Communities \setminus \{user \mapsto community\}$
$Users' = Users$
$Communities' = Communities$

This operation removes user enrollment into a community by removing the association between the community and the user.

(h) **Operation: Delete Community**

DeleteCommunity

$\Delta User\_Enrollments$
$\Delta Item\_Classifications$
$community : COMMUNITY$
$cid? : \mathbb{N}$

$community = (\mu\, m : Communities \mid m.community\_id = cid?)$
$Community\_Is\_Classified' = \{community\} \lhd Community\_Is\_Classified$
$User\_Enrolls\_In\_Communities' = User\_Enrolls\_In\_Communities \rhd \{community\}$
$Communities' = Communities \setminus \{community\}$
$Users' = Users$
$Projects' = Projects$
$Needs' = Needs$
$Project\_Has\_Needs' = Project\_Has\_Needs$
$Artifacts' = Artifacts$
$Artifact\_Is\_Classified' = Artifact\_Is\_Classified$
$Need\_Is\_Classified' = Need\_Is\_Classified$
$User\_Is\_Classified' = User\_Is\_Classified$

This operation deletes a community. This operations removes the classification associated with the community. It removes both user enrollments and expert enrollments in the community. Finally, the operation removes the community.

(i) **Operation: Delete Classification**

$$
\begin{array}{|l}
\textit{DeleteClassification} \\
\hline
\Delta \textit{Item\_Classifications} \\
\textit{cid}? : \mathbb{N} \\
\textit{classification} : \textit{CLASSIFICATION} \\
\hline
\textit{classification} = (\mu\, m : \textit{Classifications} \mid m.\textit{classification\_id} = \textit{cid}?) \\
\textit{User\_Is\_Classified}' = \textit{User\_Is\_Classified} \rhd \{\textit{classification}\} \\
\textit{Artifact\_Is\_Classified}' = \textit{Artifact\_Is\_Classified} \rhd \{\textit{classification}\} \\
\textit{Community\_Is\_Classified}' = \textit{Community\_Is\_Classified} \rhd \{\textit{classification}\} \\
\textit{Need\_Is\_Classified}' = \textit{Need\_Is\_Classified} \rhd \{\textit{classification}\} \\
\textit{Classifications}' = \textit{Classifications} \setminus \{\textit{classification}\} \\
\textit{Project\_Has\_Needs}' = \textit{Project\_Has\_Needs} \\
\textit{Users}' = \textit{Users} \\
\textit{Projects}' = \textit{Projects} \\
\textit{Needs}' = \textit{Needs} \\
\textit{Artifacts}' = \textit{Artifacts} \\
\textit{Communities}' = \textit{Communities} \\
\hline
\end{array}
$$

This operation deletes a classification. It removes the classifications associated with the user, artifact, the set of project needs, and community. Then, the classification is removed.

9. **Search Operations**

   (a) **Operation: List Needs for a Project**

$$
\begin{array}{|l}
\textit{ListNeedsForProjects} \\
\hline
\Xi \textit{KMS} \\
\textit{pid}? : \mathbb{N} \\
\textit{project} : \textit{PROJECT} \\
\textit{lneed}! : \mathbb{P}\, \textit{NEED} \\
\hline
\textit{project} = (\mu\, m : \textit{Projects} \mid m.\textit{project\_id} = \textit{pid}?) \\
\{\textit{project}\} \lhd \textit{Project\_Has\_Needs} \neq \emptyset \\
\textit{lneed}! = \textit{Project\_Has\_Needs}(\!|\, \{\textit{project}\} \,|\!) \\
\hline
\end{array}
$$

   This operation lists the needs for a project.

   (b) **Operation: List Users for a Need**

ListUsersForNeed

ΞKMS
$nid? : \mathbb{N}$
$need : NEED$
$luser! : \mathbb{P}\, USER$

$need = (\mu\, m : Needs \mid m.need\_id = nid? \land m.need\_type = user)$
$Need\_Is\_Classified(\!| \{need\} |\!) \neq \emptyset$
$luser! = User\_Is\_Classified^{\sim}(\!| (Need\_Is\_Classified(\!| \{need\} |\!)) |\!) \neq \emptyset$

This operation lists all the users for a need. The *need_id* is used to identify the classification that has classified the need. Using this classification, the inverse of the *User_Is_Classified* relation is traversed to locate the set of users.

(c) **Operation: List Experts for a Need**

ListExpertsForNeed

ΞKMS
$nid? : \mathbb{N}$
$need : NEED$
$luser : \mathbb{P}\, USER$
$lexpert! : \mathbb{P}\, USER$

$need = (\mu\, m : Needs \mid m.need\_id = nid? \land m.need\_type = expert)$
$Need\_Is\_Classified(\!| \{need\} |\!) \neq \emptyset$
$User\_Is\_Classified^{\sim}(\!| (Need\_Is\_Classified(\!| \{need\} |\!)) |\!) \neq \emptyset$
$luser = User\_Is\_Classified^{\sim}(\!| (Need\_Is\_Classified(\!| \{need\} |\!)) |\!)$
$lexpert! = \{x : luser \mid x.expert\_class \neq none\} \neq \emptyset$

This operation lists all the experts for a need. The *need_id* is used to identify the classification that has classified the need. Using this classification, the inverse of the *User_Is_Classified* relation is traversed to locate the set of users who have the *expert_class* set to a value other than *none*.

(d) **Operation: List Artifacts for a Need**

ListArtifactsForNeed

ΞKMS
$nid? : \mathbb{N}$
$need : NEED$
$lartifact! : \mathbb{P}\, ARTIFACT$

$need = (\mu\, m : Needs \mid m.need\_id = nid? \land m.need\_type = artifact)$
$Need\_Is\_Classified(\!| \{need\} |\!) \neq \emptyset$
$lartifact! = Artifact\_Is\_Classified^{\sim}(\!| (Need\_Is\_Classified(\!| \{need\} |\!)) |\!) \neq \emptyset$

This operation lists all the artifacts for a need. The *need_id* is used to identify the classification that has classified the need. Using this classification, the inverse of the *Artifact_Is_Classified* relation is traversed to locate the set of artifacts.

(e) **Operation: List Communities for a Need**

ListCommunitiesForNeed

$$
\begin{array}{l}
\Xi KMS \\
nid? : \mathbb{N} \\
need : NEED \\
lcommunity! : \mathbb{P}\, COMMUNITY \\
\hline
need = (\mu\, m : Needs \mid m.need\_id = nid? \land m.need\_type = community) \\
Need\_Is\_Classified(\!| \{need\} |\!) \neq \emptyset \\
lcommunity! = Community\_Is\_Classified^{\sim}(\!| (Need\_Is\_Classified(\!| \{need\} |\!)) |\!) \neq \emptyset
\end{array}
$$

This operation lists all the communities for a need. The *need_id* is used to identify the classification that has classified the need. Using this classification, the inverse of the *Community_Is_Classified* relation is traversed to locate set of communities.

(f) **Operation: List Projects with Similar Need**

ListProjectsWithSimilarNeed

$$
\begin{array}{l}
\Xi KMS \\
nid? : \mathbb{N} \\
need : NEED \\
nty? : NEED\_TYPE \\
lproject! : \mathbb{P}\, PROJECT \\
lclassification : \mathbb{P}\, CLASSIFICATION \\
lneed : \mathbb{P}\, NEED \\
\hline
need = (\mu\, m : Needs \mid m.need\_id = nid? \land m.need\_type = nty?) \\
Need\_Is\_Classified(\!| \{need\} |\!) \neq \emptyset \\
lclassification = Need\_Is\_Classified(\!| \{need\} |\!) \\
lneed = \mathrm{dom}(Need\_Is\_Classified \rhd lclassification) \setminus \{need\} \\
lproject! = \mathrm{dom}(Project\_Has\_Needs \rhd lneed)
\end{array}
$$

This operation lists all projects with a need similar to the need for the specific need type that is specified as an input. The classification of the need is used to located a set of classifications. Using this classification, the needs are traversed to locate all needs (other than the one mentioned), with a similar classification. These needs are used to locate projects associated with the needs.

(g) **Operation: List Communities Enrollment for Project**

ListCommunitiesEnrollmentForProject

$$\Xi KMS$$
$$pid? : \mathbb{N}$$
$$project : PROJECT$$
$$luser : \mathbb{P}\, USER$$
$$lcommunities! : \mathbb{P}\, COMMUNITY$$

$$project = (\mu\, m : Projects \mid m.project\_id = pid?)$$
$$luser = User\_Enrolls\_In\_Projects^{\sim}(\!|\ \{project\}\ |\!) \neq \emptyset$$
$$lcommunities! = User\_Enrolls\_In\_Communities(\!|\ luser\ |\!)$$

This operation lists all projects with a need similar to the need specified as an input. The classification of the need is used to located a set of classifications. Using this classification, the needs are traversed to locate all needs (other than the one mentioned), with a similar classification. These needs are used to locate projects associated with the needs.

10. **Error Handling: Entity Errors** The following defines error handling in the system.

    (a) Successful Completion

Success

$$result! : RETURN$$

$$result! = OK$$

This denotes successful completion of the operation.

(b) User not found.

UserNotFound

$$\Xi KMS$$
$$uid? : \mathbb{N}$$
$$result! : RETURN$$

$$\forall\, m : Users \bullet m.user\_id \neq uid?$$
$$\Rightarrow result! = USER\_NOT\_FOUND$$

This denotes that the user does not exist.

(c) Project not found.

ProjectNotFound

ΞKMS
$pid? : \mathbb{N}$
$result! : RETURN$

$\forall m : Projects \bullet m.project\_id \neq pid? \Rightarrow$
$result! = PROJECT\_NOT\_FOUND$

This denotes that the project does not exist.

(d) Need not found.

NeedNotFound

ΞKMS
$nid? : \mathbb{N}$
$result! : RETURN$

$\forall m : Needs \bullet m.need\_id \neq nid? \Rightarrow$
$result! = NEED\_NOT\_FOUND$

This denotes that the need does not exist.

(e) Artifact not found.

ArtifactNotFound

ΞKMS
$aid? : \mathbb{N}$
$result! : RETURN$

$\forall m : Artifacts \bullet m.artifact\_id \neq aid? \Rightarrow$
$result! = ARTIFACT\_NOT\_FOUND$

This denotes that the assessment does not exist.

(f) Artifact not found.

AssessmentNotFound

ΞKMS
$aid? : \mathbb{N}$
$result! : RETURN$

$\forall m : Assessments \bullet m.assessment\_id \neq aid? \Rightarrow$
$result! = ASSESSMENT\_NOT\_FOUND$

This denotes that the artifact does not exist.

(g) Community not found.

CommunityNotFound

$$\Xi KMS$$
$$cid? : \mathbb{N}$$
$$result! : RETURN$$

$$\forall m : Communities \bullet m.community\_id \neq cid? \Rightarrow$$
$$result! = COMMUNITY\_NOT\_FOUND$$

This denotes that the community does not exist.

(h) Classification not found.

ClassificationNotFound

$$\Xi KMS$$
$$cid? : \mathbb{N}$$
$$result! : RETURN$$

$$\forall m : Classifications \bullet m.classification\_id \neq cid? \Rightarrow$$
$$result! = CLASSIFICATION\_NOT\_FOUND$$

This denotes that the classification does not exist.

(i) Enrichment not found.

EnrichmentNotFound

$$\Xi KMS$$
$$artifact : ARTIFACT$$
$$aid? : \mathbb{N}$$
$$eid? : \mathbb{N}$$
$$result! : RETURN$$

$$artifact = (\mu m : Artifacts \mid m.artifact\_id = aid?)$$
$$\forall m : artifact.Enrichments \bullet m.enrichment\_id \neq eid? \Rightarrow$$
$$result! = ENRICHMENT\_NOT\_FOUND$$

This denotes that the classification does not exist.

(j) User is already designated as an expert.

UserAlreadyAnExpert

$$\Xi KMS$$
$$user : USER$$
$$uid? : \mathbb{N}$$
$$result! : RETURN$$

$$user = (\mu\, m : Users \mid m.user\_id = uid?)$$
$$user.expert\_class \neq none$$
$$\Rightarrow result! = USER\_ALREADY\_AN\_EXPERT$$

This denotes that the user is already an expert. Only users who are not already experts can be designated as experts.

(k) User is not an expert.

UserNotAnExpert

$$\Xi KMS$$
$$user : USER$$
$$uid? : \mathbb{N}$$
$$result! : RETURN$$

$$user = (\mu\, m : Users \mid m.user\_id = uid?)$$
$$user.expert\_class = none$$
$$\Rightarrow result! = USER\_NOT\_AN\_EXPERT$$

This error condition can occur when a user is not designated as an expert.

(l) Expert Class must not be *none*.

ExpertClassNone

$$\Xi KMS$$
$$ec? : EXPERT\_CLASS$$
$$result! : RETURN$$

$$ec? = none \Rightarrow result! = EXPERT\_CLASS\_NONE$$

Expert class has been defined as none. When designating a user as an expert, expert class cannot be defined as none.

11. **Error Handling: Logic Errors** The following defines error handling in the system.

(a) Project has knowledge needs.

ProjectHasKnowledgeNeeds

$$\Xi KMS$$
$$project : PROJECT$$
$$pid? : \mathbb{N}$$
$$result! : RETURN$$

$$project = (\mu\, m : Projects \mid m.project\_id = pid?)$$
$$\{project\} \lhd Project\_Has\_Needs \neq \emptyset$$
$$\Rightarrow result! = PROJECT\_HAS\_KNOWLEDGE\_NEEDS$$

This denotes that a project has knowledge needs. When a project is deleted, all needs must first be deleted before the project can be deleted.

(b) Project has no knowledge needs.

ProjectHasNoKnowledgeNeeds

$$\Xi KMS$$
$$project : PROJECT$$
$$pid? : \mathbb{N}$$
$$result! : RETURN$$

$$project = (\mu\, m : Projects \mid m.project\_id = pid?)$$
$$\{project\} \lhd Project\_Has\_Needs = \emptyset$$
$$\Rightarrow result! = PROJECT\_HAS\_NO\_KNOWLEDGE\_NEEDS$$

This error condition can occur when a project has no knowledge needs.

(c) Project has no users.

ProjectHasNoUsers

$$\Xi KMS$$
$$project : PROJECT$$
$$pid? : \mathbb{N}$$
$$result! : RETURN$$

$$project = (\mu\, m : Projects \mid m.project\_id = pid?)$$
$$User\_Enrolls\_In\_Projects^{\sim}(\!| \{project\} |\!) = \emptyset$$
$$\Rightarrow result! = PROJECT\_HAS\_NO\_USERS$$

This error condition can occur when a project has no knowledge needs.

(d) Need has not been classified.

NeedNotClassified

ΞKMS
$nid? : \mathbb{N}$
$need : NEED$
$result! : RETURN$

$need = (\mu\, m : Needs \mid m.need\_id = nid?)$
$Need\_Is\_Classified(\!| \{need\} |\!) = \emptyset$
$\Rightarrow result! = KNOWLEDGE\_NEED\_NOT\_CLASSIFIED$

This error condition can occur when a knowledge need is not classified. This is critical in listing different connections in the KMS that are made using the classification scheme. The need must be classified to ensure that related entities such as communities, users and artifacts can be reached using the classification scheme.

(e) No user fulfills the need.

NoUserFulfillsNeed

ΞKMS
$need : NEED$
$nid? : \mathbb{N}$
$result! : RETURN$

$need = (\mu\, m : Needs \mid m.need\_id = nid?)$
$User\_Is\_Classified^{\sim}(\!| (Need\_Is\_Classified(\!| \{need\} |\!)) |\!) = \emptyset$
$\Rightarrow result! = NO\_USER\_FOUND\_FOR\_KNOWLEDGE\_NEED$

This error condition can occur when a user is not found for the project need.

(f) No expert fulfills the need.

NoExpertFulfillsNeed

ΞKMS
$need : NEED$
$nid? : \mathbb{N}$
$luser : \mathbb{P}\, USER$
$result! : RETURN$

$need = (\mu\, m : Needs \mid m.need\_id = nid?)$
$luser = User\_Is\_Classified^{\sim}(\!| (Need\_Is\_Classified(\!| \{need\} |\!)) |\!) \neq \emptyset$
$\{x : luser \mid x.expert\_class \neq none\} = \emptyset$
$\Rightarrow result! = NO\_EXPERT\_FOUND\_FOR\_KNOWLEDGE\_NEED$

This error condition can occur when a user is not found for the project need.

(g) No artifact fulfills the need.

$$
\begin{array}{|l}
\textit{NoArtifactFulfillsNeed} \\
\hline
\Xi KMS \\
need : NEED \\
nid? : \mathbb{N} \\
result! : RETURN \\
\hline
need = (\mu\, m : Needs \mid m.need\_id = nid?) \\
Artifact\_Is\_Classified^{\sim}(\!| \ (Need\_Is\_Classified(\!| \ \{need\} \ |\!)) \ |\!) = \emptyset \\
\Rightarrow result! = NO\_ARTIFACT\_FOUND\_FOR\_KNOWLEDGE\_NEED \\
\hline
\end{array}
$$

This error condition can occur when an artifact is not found for the project need.

(h) No community fulfills the need.

$$
\begin{array}{|l}
\textit{NoCommunityFulfillsNeed} \\
\hline
\Xi KMS \\
need : NEED \\
nid? : \mathbb{N} \\
result! : RETURN \\
\hline
need = (\mu\, m : Needs \mid m.need\_id = nid?) \\
Community\_Is\_Classified^{\sim}(\!| \ (Need\_Is\_Classified(\!| \ \{need\} \ |\!)) \ |\!) = \emptyset \\
\Rightarrow result! = NO\_COMMUNITY\_FOUND\_FOR\_KNOWLEDGE\_NEED \\
\hline
\end{array}
$$

This error condition can occur when a community is not found for the project need.

12. **Robust Operations**
The following are robust operations performed by the Knowledge Management System.

- Robust initialization of Knowledge Management.

  $$RInitKMS \mathrel{\widehat{=}} InitKMS \land Success$$

  This is the robust initialization. Since there are no specific error conditions, the robust version returns a successful completion of initialization.

- Robust addition of a user.

  $$RAddUser \mathrel{\widehat{=}} (AddUser \land Success)$$

  This is the robust addition of a user. In this operation, the user is added with a successful completion. There are no specific error conditions.

- Robust designation of an expert.

  $$RDesignateUserAsExpert \mathrel{\widehat{=}} (DesignateUserAsExpert \land Success) \\ \lor UserNotFound \lor UserAlreadyAnExpert \lor ExpertClassNone$$

  This is the robust designation of a user as an expert. In this operation, either the user is successfully designated as an expert, or the user has not been found, or the user is already an expert.

- Robust addition of a project.

  $$RAddProject \mathrel{\widehat{=}} (AddProject \land Success)$$

  This is the robust addition of a project. In this operation, the project is added with a successful completion. There are no specific error conditions.

- Robust establishment of a need.

  $$REstablishNeedForProject \mathrel{\widehat{=}} \\ (EstablishNeedForProject \land Success) \lor ProjectNotFound \lor NeedNotFound$$

  This is the robust establishment of a need for a project. In this operation, either the need is successfully established for a project, or the project has not been found.

- Robust addition of an artifact.

  $$RAddArtifact \mathrel{\widehat{=}} (AddArtifact \land Success) \lor UserNotFound$$

  This is the robust addition of an artifact. In this operation, the artifact is added with a successful completion or the user has not been found in the system.

- Robust enrichment of an artifact.

  $$REnrichArtifact \mathrel{\widehat{=}} \\ (EnrichArtifact \land Success) \lor ArtifactNotFound$$

  This is the robust enrichment of an artifact. In this operation, either the artifact is successfully enriched, or the artifact has not been found.

- Robust assessment of an artifact.

  $$RAssessArtifact \mathrel{\widehat{=}} \\ (AssessArtifact \land Success) \lor ArtifactNotFound \lor UserNotFound$$

  This is the robust assessment of an artifact. In this operation, either the artifact is successfully enriched, or the artifact has not been found or the user has not been found.

- Robust enrollment of a user to a community.

  $$REnrollUserInCommunity \mathrel{\widehat{=}} (EnrollUserInCommunity \land Success) \lor UserNotFound \lor CommunityNotFound$$

  This is the robust enrollment of a user into a community. In this operation, either the enrollment is successful, or community does not exist or the user does not exist.

- Robust enrollment of a user to a project.

  $$REnrollUserInProject \mathrel{\widehat{=}} (EnrollUserInProject \land Success) \lor UserNotFound \lor ProjectNotFound$$

  This is the robust enrollment of a user into a project. In this operation, either the enrollment is successful, or project does not exist or the user does not exist.

- Robust addition of a classification.

  $$RAddClassification \mathrel{\widehat{=}} (AddClassification \land Success)$$

  This is the robust addition of an classification. In this operation, the classification is added with a successful completion. There are no specific error conditions.

- Robust classification of an need.

  $$RClassifyNeed \mathrel{\widehat{=}} (ClassifyNeed \land Success) \lor NeedNotFound \lor ClassificationNotFound$$

  This is the robust classification of a need. In this operation, either the classification is successful, or need does not exist or the classification does not exist.

- Robust classification of a user.

  $$RClassifyUser \mathrel{\widehat{=}} (ClassifyUser \land Success) \lor UserNotFound \lor ClassificationNotFound$$

  This is the robust classification of a user. In this operation, either the classification is successful, or user does not exist or the classification does not exist.

- Robust classification of a community.

  $$RClassifyCommunity \mathrel{\widehat{=}} (ClassifyCommunity \land Success) \lor CommunityNotFound \lor ClassificationNotFound$$

  This is the robust classification of a community. In this operation, either the classification is successful, or community does not exist or the classification does not exist.

- Robust deletion of user.

  $$RDeleteUser \mathrel{\widehat{=}} (DeleteUser \land Success) \\ \lor UserNotFound$$

  This is the robust designation of a user as an expert. In this operation, either the user is successfully deleted or the user has not been found.

- Robust removal of expert designation.

  $$RRemoveExpertDesignation \mathrel{\widehat{=}} (RemoveExpertDesignation \land Success) \\ \lor UserNotFound \lor UserNotAnExpert$$

  This is the robust removal of the designation of a user as an expert. In this operation, either the user is successfully de-assigned as an expert, or the user has not been found, or the user is not an expert.

- Robust deletion of a knowledge need.

  $$RDeleteNeed \mathrel{\widehat{=}} \\ (DeleteNeed \land Success) \lor NeedNotFound$$

  This is the robust deletion of an knowledge need. In this operation, either the deletion is successful, or knowledge need does not exist or the project does not exist.

- Robust deletion of a project.

  $$RDeleteProject \mathrel{\widehat{=}} \\ (DeleteProject \land Success) \lor ProjectNotFound \\ \lor ProjectHasKnowledgeNeeds$$

  This is the robust deletion of a project. In this operation, either the deletion is successful, or project does not exist or project has associated needs which have to be removed.

- Robust deletion of a knowledge enrichment.

  $$RDeleteEnrichment \mathrel{\widehat{=}} \\ (DeleteEnrichment \land Success) \lor \\ ArtifactNotFound \lor EnrichmentNotFound$$

  This is the robust deletion of an knowledge enrichment. In this operation, either the deletion is successful, or knowledge enrichment does not exist or the association does not exist between the artifact and the enrichment.

- Robust deletion of a knowledge artifact.

  $$RDeleteArtifact \mathrel{\widehat{=}} \\ (DeleteArtifact \land Success) \lor ArtifactNotFound$$

  This is the robust deletion of an knowledge artifact. In this operation, either the deletion is successful, or knowledge artifact does not exist.

- Robust removal of user enrollment into a community.

  $$RRemoveUserEnrollment \mathrel{\widehat=} (RemoveUserEnrollment \land Success) \lor CommunityNotFound \lor UserNotFound$$

  This is the robust removal of a user enrollment into a community. In this operation, either the removal is successful, or community does not exist or the user does not exist.

- Robust deletion of a community.

  $$RDeleteCommunity \mathrel{\widehat=} (DeleteCommunity \land Success) \lor CommunityNotFound$$

  This is the robust deletion of an community. In this operation, either the deletion is successful, or community does not exist.

- Robust deletion of a classification.

  $$RDeleteClassification \mathrel{\widehat=} (DeleteClassification \land Success) \lor ClassificationNotFound$$

  This is the robust deletion of an classification. In this operation, either the deletion is successful, or classification does not exist.

- Robust listing of project needs.

  $$RListNeedsForProjects \mathrel{\widehat=} (ListNeedsForProjects \land Success) \lor ProjectNotFound \lor ProjectHasNoKnowledgeNeeds$$

  This is the robust listing of needs for a project. Either the project needs are listed successfully, or the project is is invalid or the project has no knowledge needs.

- Robust listing of users for a project need.

  $$RListUsersForNeed \mathrel{\widehat=} (ListUsersForNeed \land Success) \lor NeedNotFound \lor NeedNotClassified \lor NoUserFulfillsNeed$$

  This is the robust listing of users for a project need. Either the set of users is found successfully, or the need is not found, the need is not classified or no expert is found.

- Robust listing of experts for a project need.

  $$RListExpertsForNeed \mathrel{\widehat=} (ListExpertsForNeed \land Success) \lor NeedNotFound \lor NeedNotClassified \lor NoUserFulfillsNeed \lor NoExpertFulfillsNeed$$

  This is the robust listing of experts for a project need. Either the set of experts is found successfully, or no user is found for the need or the need is not found, the need is not classified or no expert is found.

- Robust listing of artifacts for a project need.

  $$RListArtifactsForNeed \mathrel{\widehat=} (ListArtifactsForNeed \land Success) \lor NeedNotFound \lor NeedNotClassified \lor NoArtifactFulfillsNeed$$

  This is the robust listing of artifacts for a project need. Either the set of artifacts is found successfully, or the need is not found, the need is not classified or no artifact is found.

- Robust listing of community for a project need.

  $$RListCommunitiesForNeed \mathrel{\widehat=} (ListCommunitiesForNeed \land Success) \lor NeedNotFound \lor NeedNotClassified \lor NoCommunityFulfillsNeed$$

  This is the robust listing of communities for a project need. Either the set of communities is found successfully, or the need is not found, the need is not classified or no community is found.

- Robust listing of projects with a need similar to my need.

  $$RListProjectsWithSimilarNeed \mathrel{\widehat=} (ListProjectsWithSimilarNeed \land Success) \lor NeedNotFound \lor NeedNotClassified$$

  This is the robust listing of projects for a need. Either the set of projects is found successfully, or the need is not found, the need is not classified.

- Robust listing of communities enrolled by a project team.

  $$RListCommunitiesEnrollmentForProject \mathrel{\widehat=} (ListCommunitiesEnrollmentForProject \land Success) \lor ProjectNotFound \lor ProjectHasNoUsers$$

  This is the robust listing of projects for a need. Either the set of projects is found successfully, or the need is not found, the need is not classified.