OpenReq Ontology

Author(s): UH, UPC

Project co-funded by the European Commission under the H2020 Programme.

Abstract
OpenReq is a project that aims to enhance requirements engineering activities. This document specifies the common ontology of OpenReq. The ontology defines the key concepts and their relationships as well as provides a blueprint for the data. In practical software engineering scenarios, the ontology is realized and used in the JSON-based messages exchanged between the different microservices in the OpenReq infrastructure. Towards this end, the OpenReq JSON schema adhering to the OpenReq ontology is provided. The microservices of OpenReq also rely on the ontology in their internal data to some extent.

This document is a shortened version of D5.3 OpenReq ontologies and pattern catalogue.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

OpenReq is a project that aims to enhance requirements engineering activities. The focus areas cover the domain of requirements engineering over the entire life-cycle, including the areas from requirements identification to release planning. In order to form a common ground, shared understanding must be formed between the life-cycle phases and stakeholders. Towards this end, the OpenReq ontology has been created.

The OpenReq ontology serves the interoperability between microservices, specification of external interfaces or APIs, and define the core data in OpenReq. The most practical need is to enable the interoperability of the different microservices of OpenReq infrastructure. Likewise, it is important that external services, including OpenReq trials and the OpenReq open call participants, can have access to the OpenReq infrastructure using stable, standardized data on the interfaces. Another important aspect is to agree on what concepts and data is managed in the OpenReq infrastructure even to the extent that data should follow a relatively similar structure adhering to the ontology in different microservices.

We apply UML class diagrams and descriptions in natural language to represent the OpenReq ontology. The use of UML in ontology representation is quite usual and well-established, being reported as the language used to express RE ontologies in 23,9% of existing approaches [1]. Class diagrams (particularly in UML) are a widespread notation that usually will not require any kind of training to understand it for most software engineers. To operationalize the ontology, we also represent a JSON schema adhering to the OpenReq ontology. JSON objects adhering to the OpenReq JSON schema are used in the messages that are sent to the interfaces of the OpenReq microservices in OpenReq infrastructure.

The OpenReq ontology has been commonly agreed in the OpenReq project. The core part or a subset of the ontology is stable and has been provided with formalization by a mapping to a logical representation especially in OpenReq Dependency engine. The formalization focuses on and around the concept of a requirement, while other key concepts, such as stakeholders have not been covered. In addition, as the OpenReq ontology aims to define the minimal set of common concepts needed in OpenReq infrastructure, it is possible to extend the OpenReq ontology for different purposes.

This document covers the OpenReq ontology. The creation of the ontology has been a co-creative and iterative process but the process is not covered here---the details about the process of creating and agreeing the ontology is reported in a separate publication [2]. We have developed only JSON-based operationalizations of the OpenReq ontology. Because all interfaces currently rely only on JSON and there have been no use for other formats, we do not cover other possible formats, such as ReqIF.

The rest of this document is organized as follows. Section 2 describes the OpenReq ontology by the class diagram in UML with a graphical outline of the concepts and relationships among them, then the textual definitions of these concepts and a simple application example. In addition, Appendix shows an informative example state transition of requirement states. This document is a shortened version of D5.3 OpenReq ontologies and pattern catalogue.

2. OpenReq ontology

A UML class diagram representation of the OpenReq ontology is shown in **Figure 1**. The lower part of the figure represents the key concepts, their properties and relationships using the UML class diagram notation. The upper part of the figure represents the enumerations, which are used in the UML diagram as types of some of the properties. The concepts of ontology are described in natural language after the UML diagram. Finally, a simple example of ontology application is provided.

2.1. Class diagram

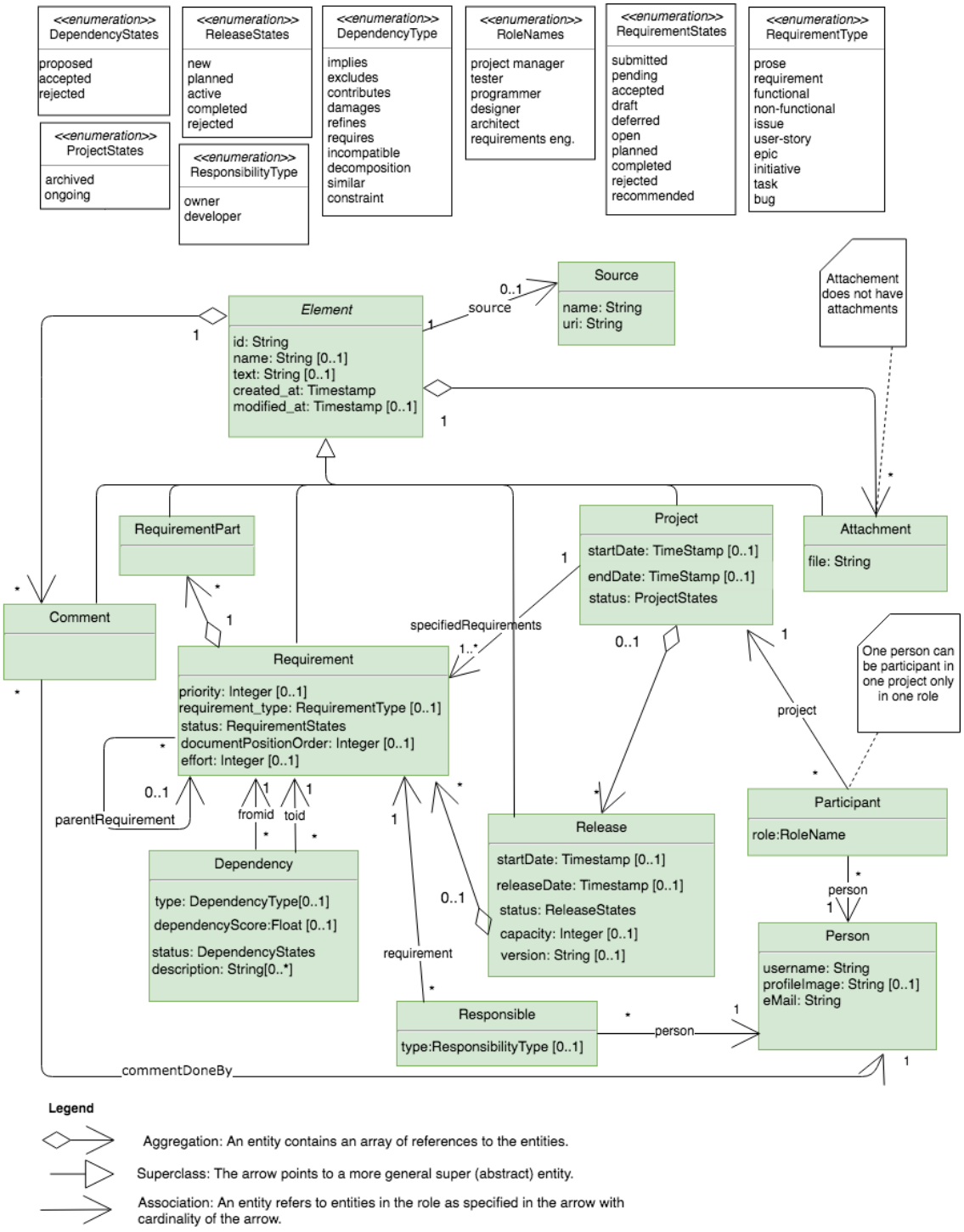

Figure 1: UML Class diagram of the OpenReq ontology.

2.2. Concepts

The concepts are defined in the following. We describe the associations only to the direction that are navigable.

Element

Concept	***Description***
<<abstract>> Element	Concepts in the OpenReq ontology that share a common set of properties and for which source can be specified (Source), comments can be defined (Comment), and files can be attached to them (Attachment)."
Properties	
id: string [1..1]	The unique identifier of an Element. Not a null value or an empty string.
name: string [0..1]	The name or title of an Element.
text: string [0..1]	The textual description or content of an Element. Note that Attachments (see below) can be used for other than plain text format descriptions.
created_at: timestamp [1..1]	The creation time of an Element in milliseconds. So called Unix time.
modified_at: timestamp [0..1]	The latest modification time of an Element in milliseconds. So called Unix time.
Associations	
source [0..1]	Association to the Source of an element, if any.
comments [0..*]	Aggregation of Comments associated to an Element, if any. See the Comment concept below for additional details.
attachments [0..*]	Aggregation of Attachments associated to the element, if any. See the Attachment concept below for additional details.

Source

Concept	***Description***
Source	The origins from which an Element, such as a Requirement, was obtained.
Properties	
name: string [1..1]	The human understandable name or other identification of Source.
uri: string [1..1]	The URI of a Source, such as a web-address.

Attachment

(subclass of Element) Attachment	A computer file related with an Element to give additional information about it An attachment has no attachments.
Properties	
id: string [1..1]	*(inherited from Element)*
name: string [0..1]	*(inherited from Element)*
text: string [0..1]	*(inherited from Element)*
created_at: timestamp [1..1]	*(inherited from Element)*
modified_at: timestamp [0..1]	*(inherited from Element)*
file: string [1..1]	The path to the attachment file.
Associations	
source [0..1]	*(inherited from Element)*
comments [0..*]	*(inherited from Element)*

Comment

(subclass of Element) Comment	A remark expressing an opinion or a concern about an Element.
Properties	
id: string [1..1]	*(inherited from Element)*
name: string [0..1]	*(inherited from Element)*
text: string [0..1]	*(inherited from Element)*
created_at: timestamp [1..1]	*(inherited from Element)*
modified_at: timestamp [0..1]	*(inherited from Element)*
Associations	
source[0..1]	*(inherited from Element)*
comments [0..*]	*(inherited from Element)*
attachments [0..*]	*(inherited from Element)*
commentDoneBy [1..1]	The Person who created the Comment.

Requirement

(subclass of Element) Requirement	A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents. (IEEE Standard Glossary)
Properties	
id: string [1..1]	*(inherited from Element)*
name: string [0..1]	*(inherited from Element)*
text: string [0..1]	*(inherited from Element)*
created_at: timestamp [1..1]	*(inherited from Element)*
modified_at: timestamp [0..1]	*(inherited from Element)*
priority: integer [0..1]	The importance of the Requirement in comparison to other Requirements according to given criteria. (IREB Glossary)
requirement_type: RequirementType [0..1]	The type of the Requirement. Requirements types allow to classify requirements by common functions, features or uses, etc. See different values in RequirementType enumerations.
status: RequirementStates [1..1]	The state of the Requirement in its life cycle. See different values in RequirementStates enumeration and **Appendix 1**.
documentPositionOrder: Integer [0..1]	The position of the Requirement as ascending number when Requirements are ordered and order has relevance, such as in a document file.
effort: integer [0..1]	The analyzed effort for implementing the Requirement. The unit of measure is not specified as it depends on the application scenario.
Associations	
source [0..1]	*(inherited from Element)*
comments [0..*]	*(inherited from Element)*
attachments [0..*]	*(inherited from Element)*
requirementParts [0..*]	Aggregation of RequirementParts out of which the requirement consists of. This aggregation provides a mechanism for specifying requirement fragments or additional information for the requirement.
parentRequirement [0...1]	The parent Requirement of the current Requirement for hierarchical structure in which the parent and child are tied together and cannot be understood without each other. For a general parent-child relationship, Dependency should be used.

RequirementType enumeration

prose	Not a Requirement, but other information related to requirements. For example, section headings or introduction text in the document-based requirements.
requirement	A general Requirement.
functional	A functional Requirement
non-functional	A non-functional Requirement
initiative	A high-level Requirement or a grouping of high level requirements.
epic	A high level Requirement.
user-story	A high level Requirement.
issue	A Requirement item typically in an issue tracker
task	A detailed or technical Requirement.
bug	Not a Requirement, but a software bug that can induce a Requirement.

RequirementsStates enumeration

draft	A Requirement is being edited.
submitted	A Requirement has been just created, but it has not been reviewed or analysed.
recommended	A Requirement has been recommended from other source, e.g., by some automatic tools.
pending	A Requirement is lacking information.
open	A Requirement has been analyzed to be acceptable.
deferred	A Requirement has been put off to a later time or postponed.
accepted	A Requirement has been accepted to be included in the project to be decided whether or not to be developed in a Release.
planned	A Requirement has been assigned to a Release.
completed	A Requirement has been implemented.
rejected	A Requirement that has been decided not be included in a Release or being implemented.

Project

(subclass of Element) Project	A set of interrelated activities, carefully planned usually by a project team, to be executed over a fixed period of time and within certain cost and other limitations of resources, to implement a certain software system. (based on Oxford Dictionary + Business Dictionary)
Properties	
id: string [1..1]	*(inherited from Element)*
name: string [0..1]	*(inherited from Element)*
text: string [0..1]	*(inherited from Element)*
created_at: timestamp [1..1]	*(inherited from Element)*
modified_at: timestamp [0..1]	*(inherited from Element)*
startDate: timeStamp [0..1]	The start time of the Project in milliseconds. So called Unix time.
endDate: timeStamp [0..1]	The end time of the Project in milliseconds. So called Unix time.
status: ProjectStates [1..1]	The state of the Project in its life cycle. See different values in ProjectStates enumeration.
Associations	
source [0..1]	*(inherited from Element)*
comments [0..*]	*(inherited from Element)*
attachments [0..*]	*(inherited from Element)*
specifiedRequirements [1..*]	The set of Requirements that have been specified for the project.
Releases [0..*]	The Releases planned to be delivered for a Project.

ProjectStates enumeration

archived	A Project that is finished.
ongoing	An active Project.

Release

(subclass of Element) release	Deliverable planned for a Project, in which a subset of the Requirements of the Project are implemented.
Properties	
id: string [1..1]	*(inherited from Element)*
name: string [0..1]	*(inherited from Element)*
text: string [0..1]	*(inherited from Element)*
created_at: timestamp [1..1]	*(inherited from Element)*
modified_at: timestamp [0..1]	*(inherited from Element)*
version: string [0..1]	The version of a Release. It is recommended to conform to Maven artefact versioning[1], such as 1.2.3 or 2.3-alpha.
startDate: timeStamp [0..1]	The start time of a Release in milliseconds. So called Unix time.
releaseDate: timestamp [0..1]	The release time of Release in milliseconds. So called Unix time.
status:ReleaseStates [1..1]	The state of the Release in its life cycle. See different values in ReleaseStates enumerations.
capacity: integer [0..1]	The maximum effort capacity of the Release. The unit of measure is not specified as it depends on the application scenario.
Associations	
source [0..1]	*(inherited from Element)*
comments [0..*]	*(inherited from Element)*
attachments [0..*]	*(inherited from Element)*
requirements [0…*]	The Requirements assigned to the Release

[1] https://maven.apache.org/ref/3.6.0/maven-artifact/apidocs/org/apache/maven/artifact/versioning/ComparableVersion.html

ReleaseStates enumeration

new	A Release in which the planned set of included Requirements has not been decided.
planned	A Release in which the set of included Requirements has been decided.
active	A Release in which a subset of the included Requirements are being developed.
completed	A Release in which its included Requirements have been implemented. Requirements that were not fully implemented, e.g., in a time-boxed development, have been removed.
rejected	A Release that will not be developed.

RequirementPart

(subclass of Element) requirementPart	A part of which a Requirement is composed of (not a sub-requirement). RequirementParts are an extension mechanism for specifying additional information related to a Requirement. For example, RequirementPart can be used to include fragments of Requirement content or additional properties of Requirement.
Properties	
id: string [1..1]	*(inherited from Element)*
name: string [0..1]	*(inherited from Element)*
text: string [0..1]	*(inherited from Element)*
created_at: timestamp [1..1]	*(inherited from Element)*
modified_at: timestamp [0..1]	*(inherited from Element)*
Associations	
source [0..1]	*(inherited from Element)*
comments [0..*]	*(inherited from Element)*
attachments [0..*]	*(inherited from Element)*

Dependency

Dependency	A relationship between Requirements. All relationships are binary relations between two Requirements.
Properties	
type: DependencyType [1..1]	The type of the Dependency. See DependencyType enumeration.
dependencyScore: float [0..1]	An estimation of the reliability of the Dependency. The estimation is needed, e.g, when dependency is extracted by automation, such as natural language processing.
status: DependencyStates [0..1]	The state of the Dependency in its life cycle. See different values in DependencyStates enumeration.
description: string [0..*]	A description or additional note of the Dependency. For example, the algorithm or source of the Dependency.
Associations	
fromid [1..1]	The requirement from which the Dependency originates.
toid [1..1]	The requirement to which the Dependency points to.

DependencyStates enumeration

proposed	An identified Dependency that has not been accepted or rejected. For example, an automatically identified Dependency has state “proposed” until it has been analyzed following agreed practices.
accepted	A proposed Dependency that was accepted or an explicitly identified Dependency. For example, all dependencies that have been marked by engineers and possibly analyzed are accepted.
rejected	A proposed Dependency that was rejected. The state is used to store rejected states in order that the same Dependencies are not proposed again.

DependencyType enumeration

implies	A implies B. A is related to B and if A is developed, B needs to be developed in an earlier release for A to be functional.
requires	A requires B. A is related to B and if A is developed, B needs to be developed in the same or earlier release for A to be functional.
contributes	A contributes to B. A is not required for B, but A adds the value of B.
excludes	A excludes B. If A is selected to be in any release, B must not be in any release. Excludes is a global dependency, not between releases.
incompatible	A is incompatible with B. If A is selected to be in a release, B must not be selected to the same release.
damages	A damages B. A does not exclude B but A decreases the value of B.
refines	A refines B. B is defined in more detail by another requirement A.
decomposition	A decomposes B. A is a child of B
similar	A is similar with B. A symmetric relationship between two requirements.
constraint	A general constraint can express other dependencies using a dedicated constraint language (see, e.g., [3]) in the Dependency description attribute.

Person

person	Someone that is Participant in a Project and is Responsible in some way of the Requirements.
Properties	
username: string [1..1]	The visible username of the Person.
profileImage: string [0..1]	The path to the profile image file of the Person.
email: string [1..1]	The e-mail address of the Person.

Participant

participant	The participation of a Person in a Project. A Person may participate in different Projects. On the other hand, one Project may have as participants different Persons. A Person can participate only in one Role in one Project.
Properties	
Role: RoleName [1..1]	The participation role. See different values in Rolename enumeration.
Associations	
project [1..1]	The Project in which the Person participates.
person [1..1]	The Person that participates in the Project.

Rolenames enumeration

project manager	The Person is responsible of the Project.
tester	The Person is responsible for testing the satisfaction of one or more requirements of the Project.
programmer	The Person is responsible for implementing on one or more requirements of the Project.
designer	The Person is responsible for design of the Project.
architect	The Person is responsible for the architecture of the Project.
requirements engineer	The Person is responsible for the engineering of the requirements of the Project.

Responsible

responsible	The responsibility of a Person on a Requirement. A Person may have different responsibilities in different Requirements. On the other hand, one Requirement may have different Persons involved each having different responsibility.
Properties	
type: ResponsibilityType [0..1]	The type of the responsibility. See different values in ResponsibilityType enumeration.
Associations	
requirement [1..1]	The Requirement on which the Person is responsible.
person [1..1]	The Person that is responsible on the Requirement.

ResponsibilityType enumeration

owner	The Person is the responsible for managing the Requirement.
developer	The Person is the responsible for implementing the Requirement.

2.3. An example application of the OpenReq ontology

A simple map example project consisting of three requirements, and one dependency and comment adhering to the OpenReq ontology is shown in **Figure 2**.

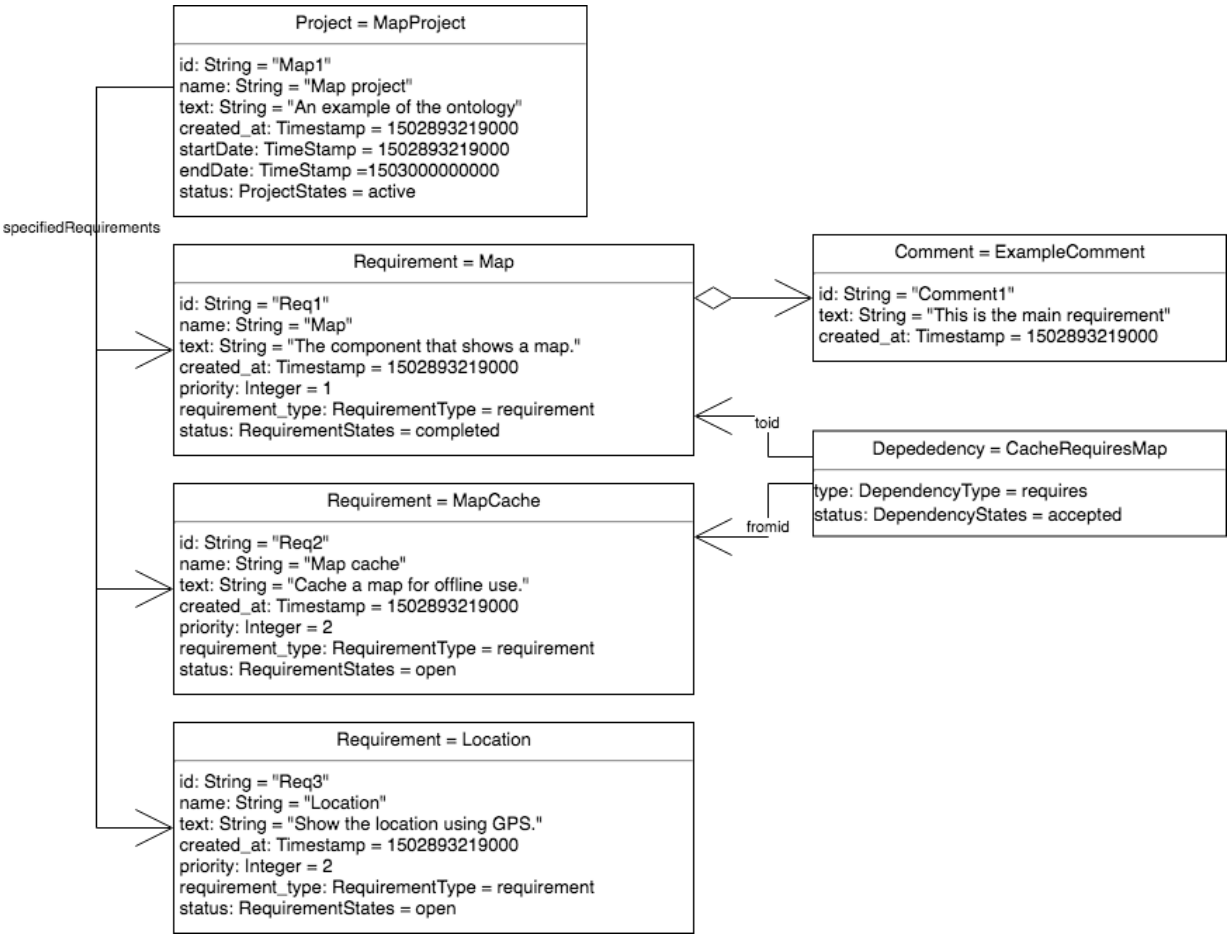

Figure 2: An example of ontology application as an UML instance diagram.

3. References

[1] Dermeval, D., Vilela, J., Bittencourt, I.I., Isotani, S, Brito, P & Silva, A., Applications of ontologies in requirements engineering: a systematic review of the literature, Requirements Engineering, 21: 405, 2016.

[2] Quer C., Franch, X., Palomares, C., Falkner, A., Felfernig, A., Fucci, D., Maalej, W., Nerlich, J., Raatikainen, M., Schenner, G., Stettinger, M. & Tiihonen, J., Reconciling Practice and Rigour in Ontology-Based Heterogeneous Information Systems Construction, The Practice of Enterprise Modeling. Springer International Publishing (LNBIP vol. 335), 2018

[3] Felfernig, A., Spöcklberger, J., Samer, R., Stettinger, M., Atas, M., Tiihonen, J. T. & Raatikainen, M., Configuring Release Plans, the 20th Configuration Workshop. CEUR-WS.org, Vol-2220, 2018.

Appendix: Example Requirement state transition (informative)

An example of a Requirement state transition is shown in **Figure 3**. The transition model is informative because the ontology does not enforce any model.

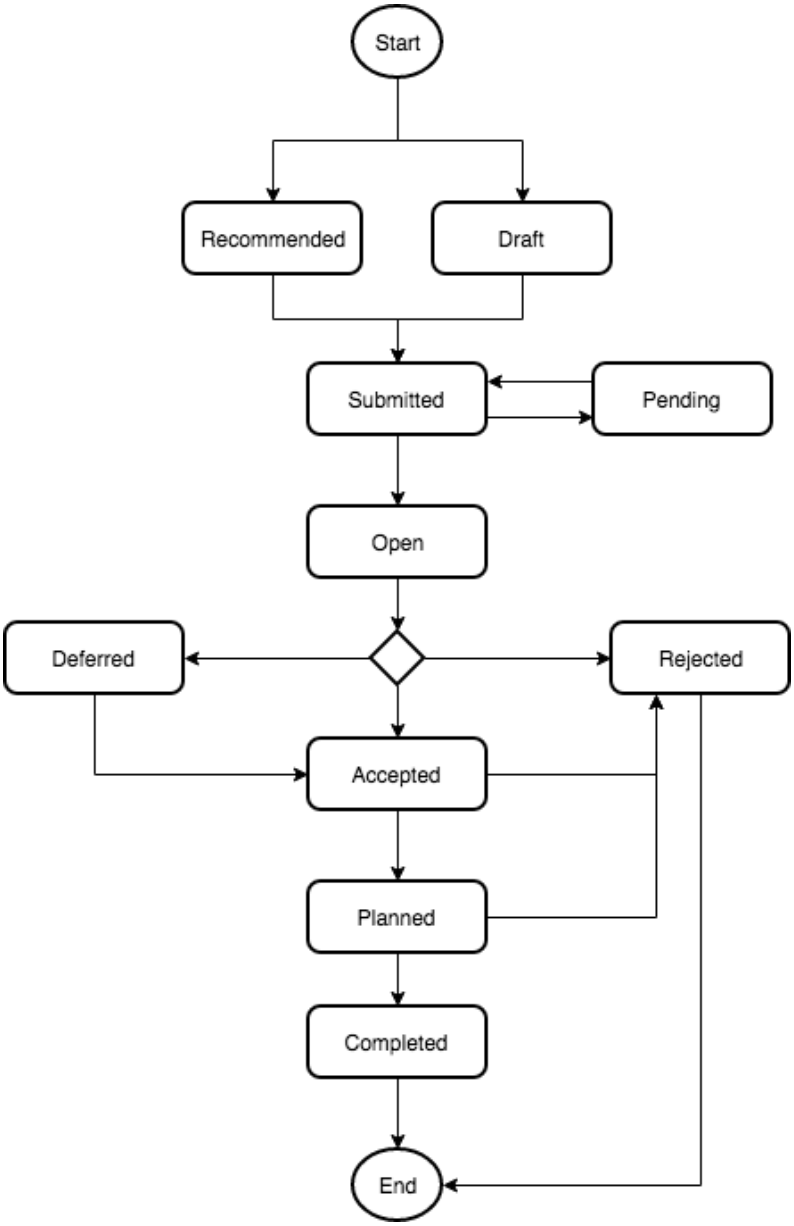

Figure 3: Example Requirement state transition.