# OASIS Committee Note

# OSLC PROMCODE Use Cases Version 1.0

## Committee Note 01

## 24 June 2021

**This stage:**
https://docs.oasis-open.org/oslc-promcode/usecase/v1.0/cn01/usecase-v1.0-cn01.html (Authoritative)
https://docs.oasis-open.org/oslc-promcode/usecase/v1.0/cn01/usecase-v1.0-cn01.pdf

**Previous stage:**
N/A

**Latest stage:**
https://docs.oasis-open.org/oslc-promcode/usecase/v1.0/usecase-v1.0.html (Authoritative)
https://docs.oasis-open.org/oslc-promcode/usecase/v1.0/usecase-v1.0.pdf

**Latest editor's draft:**
http://tools.oasis-open.org/version-control/browse/wsvn/oslc-promcode/shape/trunk/usecase.html

**Technical Committee:**
OASIS OSLC Lifecycle Integration for Project Management of Contracted Delivery (OSLC PROMCODE) TC

**Chair:**
Tom Kamimura ( tomk@nanzan.jp ), Nanzan University

**Editors:**
Mikio Aoyama (amikio@nanzan.jp), Nanzan University

  

Yoshio Horiuchi (hoy@jp.ibm.com), IBM
Tom Kamimura (tomk@nanzan.jp), Nanzan University
Shinji Matsuoka (matuoka.sinji@jp.fujitsu.com), Fujitsu Limited
Shigeaki Matsumoto (shigeaki.m@nec.com), NEC Corporation
Masaki Wakao (wakao@jp.ibm.com), IBM
Kazuo Yabuta (yabuta@nanzan.jp), Nanzan University
Hiroyuki Yoshida (yhryuki@nanzan-u.ac.jp), Nanzan University

### Related work:

This document is related to:

- *OSLC PROMCODE Version 1.0*. Edited by Mikio Aoyama, Yoshio Horiuchi, Tom Kamimura, Shinji Matsuoka, Shigeaki Matsumoto, Masaki Wakao, Kazuo Yabuta, and Hiroyuki Yoshida. Latest stage: https://docs.oasis-open.org/oslc-promcode/promcode/v1.0/promcode-spec.html.

### Abstract:

This document describes use cases, scenarios and flows of detailed activities of scenarios of the OASIS PROMCODE specification to provide the reader with understanding of some real situations of project management in contracted software delivery which are modeled by the OASIS PROMCODE specification.

### Status:

This is a Non-Standards Track Work Product.

This document was last revised or approved by the OASIS OSLC Lifecycle Integration for Project Management of Contracted Delivery (OSLC PROMCODE) TC on the above date. The level of approval is also listed above. Check the “Latest stage” location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=oslc-promcode#technical.

TC members should send comments on this specification to the TC’s email list. Others should send comments to the TC’s public comment list oslc-promcode-comment@lists.oasis-open.org, after subscribing to it by following the instructions at the “Send A Comment” button on the TC’s web page at https://www.oasis-open.org/committees/oslc-promcode/.

This specification is provided under the RF on Limited Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC’s web page (https://www.oasis-open.org/committees/oslc-promcode/ipr.php).

### Citation format:

When referencing this specification, the following citation format should be used:

  

**[OSLC-PROMCODE-UseCases-v1.0]**
*OSLC PROMCODE Use Cases Version 1.0*. Edited by Mikio Aoyama, Yoshio Horiuchi, Tom Kamimura, Shinji Matsuoka, Shigeaki Matsumoto, Masaki Wakao, Kazuo Yabuta, and Hiroyuki Yoshida. 24 June 2021. OASIS Committee Note 01. https://docs.oasis-open.org/oslc-promcode/usecase/v1.0/cn01/usecase-v1.0-cn01.html. Latest stage: https://docs.oasis-open.org/oslc-promcode/usecase/v1.0/usecase-v1.0.html.

# Notices



All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable

produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for above guidance.

# Table of Contents



  

# 1. Introduction

This document describes use cases, scenarios and flow of detailed activities of scenarios of the PROMCODE specification [OSLCPROMCODE] to provide the reader with understanding of real situation of project management in contracted software delivery with the OASIS PROMCODE specification model. It is intended to provide the reader with understanding of real situations of project management in contracted software delivery with the OASIS PROMCODE model. There are three chapters in this document.

In the next chapter 2. Use Cases and Scenarios, the scope of use cases and an assumed environment for use cases are described first, and then, three use cases are described. These are project planning, project execution and control and project closing. The use case of project execution and control are broken into eight use case scenarios. These use cases and the use case scenarios are described in detail.

The chapter3. An Implementation of the PROMCODE specification with LDPC describes one implementation of the OASIS PROMCODE specification [OSLCPROMCODE] that uses LDPC that accepts POST operations to create resources and supports dynamic service discovery. Known existing implementations so far follow the implementation described.

The last chapter 4. Systems and Data Flow for Scenarios, detailed flow of activities in the use cases and use case scenarios are described in an implementation of the PROMCODE model. The chapter first describes assumption of the implementation. Then, it will describe each of use cases and scenarios with details of flow of activities including sequence diagrams.

# 2. Use Cases and Scenarios

This chapter describes use cases and use case scenarios, or scenarios for short, for the PROMCODE Specification. The next chapter gives detailed systems and data flow of the scenarios described in this chapter.

The vocabulary used in this chapter is based on commonly used terms in many real project management activities in contracted delivery. It is also consistent with global standards such as [PMBOK5] and [ISO21500].

In this chapter and in Appendix A, the terms *acquirer* and *supplier* are used to refer to the project manager responsible for each organization to manage the project.

## 2.1 Scope of Use Cases

There are three use cases as shown in Fig. 1 Scope of Use Cases : Project Planning, Project Execution and Control, and Project Closing. Each use case can contain multiple use case scenarios, each of which represents a single path of sequential activities to achieve its goal. Each of these use cases and use case scenarios is described in the subsequent sections. Details of each use case scenario depend on various elements such as relationships between an acquirer and a supplier, how much information is shared between them, and tools used and project management environment of an acquirer and a supplier. In particular, an environment in which PROMCODE servers and clients are configured has a major influence on the detailed sequence of activities of interaction between an acquirer and a supplier.

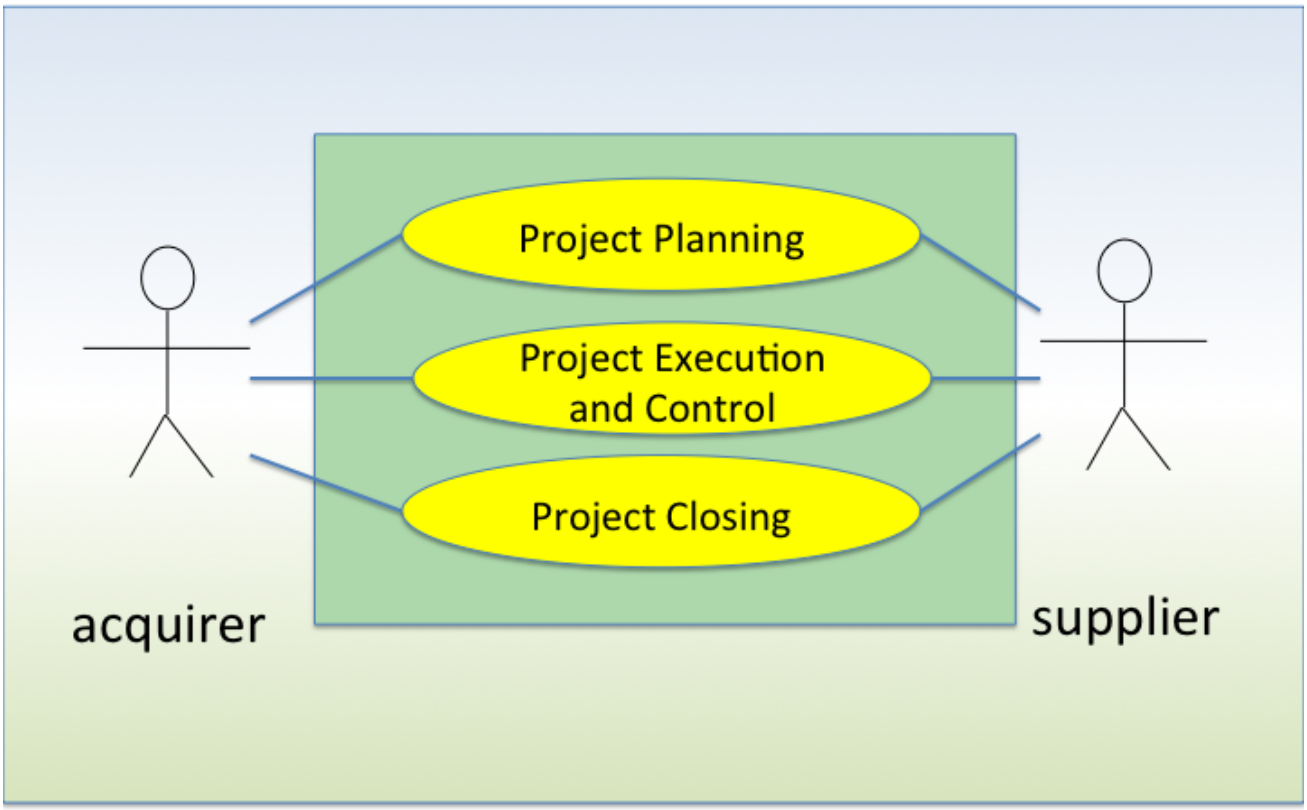


Fig. 1 Scope of Use Cases

## 2.2 Environment for Use Cases

There is a variety of environments that influence interaction between an acquirer and a supplier in reality. Among them, the following two environments seem typical. One is an environment that has a common PROMOCODE server available that can be accessed by both an acquirer and a supplier. This environment is called shared server environment. Each of an acquirer and a supplier uses its own tool to manage their work, and information sharing between them is done by using the common PROMCODE server. The second typical environment is non-shared environment. Non-shared server environment is an environment where there is no common server available between an acquirer and a supplier, and information sharing is done by sending and receiving information explicitly. The two common server environments explained are shown in Fig. 2 Environment of Use Cases .

  

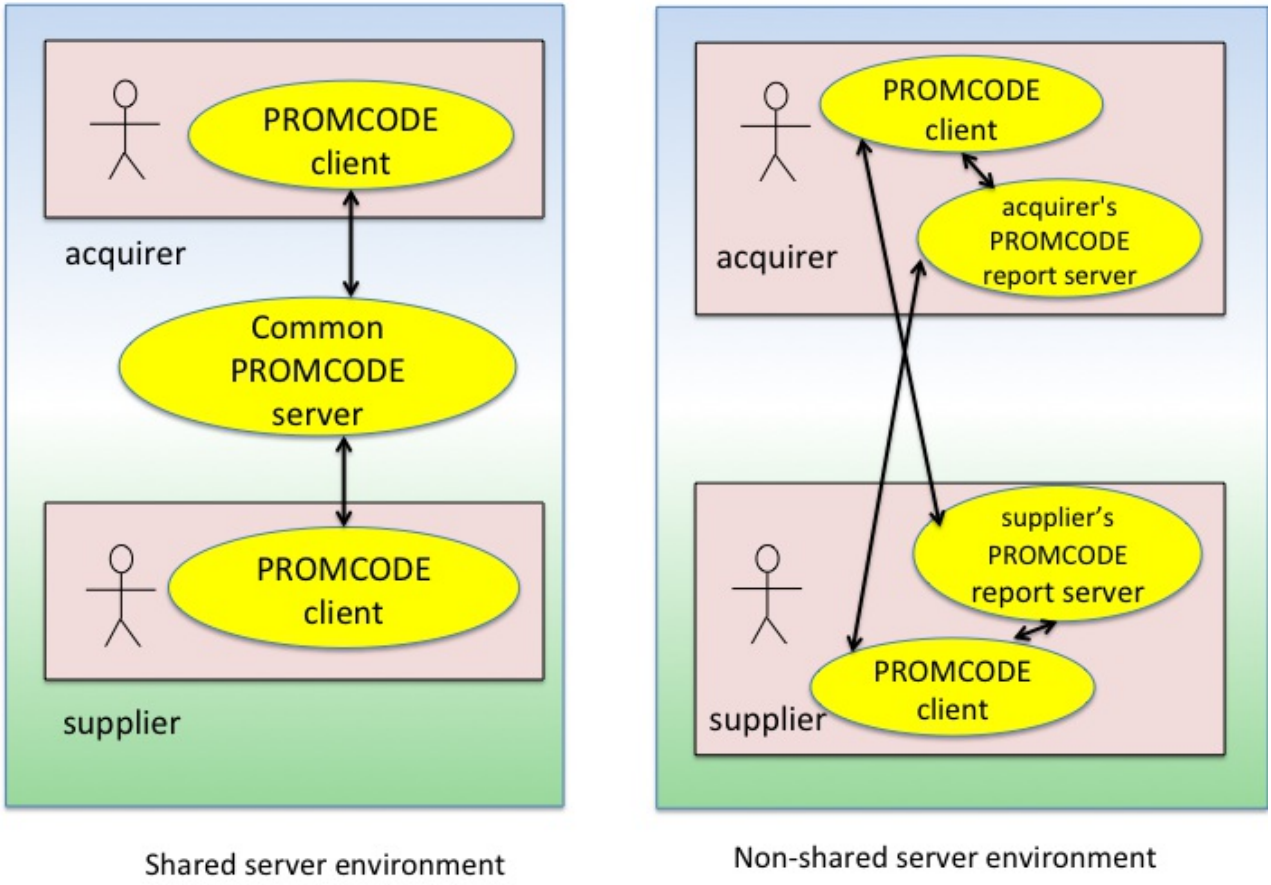


Fig. 2 Environment of Use Cases

In this chapter as well as in Appendix A, we primarily use the shared server environment since sequences of activities are simpler than those in non-shared environment. For non-shared server environment, similar sequences can be used to guide activities of an acquirer and a suppler. When helpful, differences on the sequences of activities in non-shared server environment will be described in the relevant portions in the sequences.

In the shared server environment, there is a common PROMOCODE server that can be accessed by both an acquirer and a supplier. In this chapter and in Appendix A, we also assume that the common server supports all the resources and resource operations described in the PROMCODE specification.

Also assumed is that the common PROMCODE server supports POST operations to create resources and PUT operations to update resources. Under this assumption, all actions in use cases and use case scenarios can be described using HTTP operations of the resources. In reality, however, the data used for project management may be created, modified and stored into a server using proprietary tools that do not support HTTP POST nor PUT operations, but the data may be made accessible as PROMCODE resources through GET operations. This is particularly common in the non-shared server environment. Note that as described in , PROMCODE servers have only the GET operations as '**MUST**'-level operations.

  

## 2.3 Project Planning

This use case has a single use case scenario. Systems and data flow of this use case is described in 4.3.1 Project Planning.

### Preconditions

Project definition is already completed. A plan is defined to agree on activities by an acquirer and a supplier and to track the progress with the reports by the supplier. In some cases, a single plan may be used for activities from the project initiation to the closure. In other cases, multiple plans may be used for planning some aspects of progress such as scheduling or quality. Also, a project may be decomposed into a sequence of "phases" and a plan may be created for each phase. In all cases, a plan defines resources to be tracked and agreed on between the acquirer and the supplier.

A plan includes scope items that define the work by the supplier, work items to describe the work by the supplier, artifacts to be produced by the supplier and target measure of the artifacts. A plan does not necessarily have to have all three types of managed items. It must have scope items since they provide binding between the acquirer and the supplier. A plan to track the scheduling may have only scope items and work items, and a plan to track the quality of artifacts may have scope items and artifacts. An acquirer and a supplier also agree on metric and unit size of scope items.

### Scenario

1. An acquirer creates a `Project` resource and a `Plan` resource. As described above, it is possible to create more than one `Plan` resource. For example, one may create a scheduling `Plan` and a quality `Plan` and then report the progress for each plan. For the use case and scenario discussion in this chapter, however, only a simple case where a single plan is used to cover both scheduling and quality aspects of the whole project is explained. In this case, a single `Report` resource is created to report the progress on scheduling and quality at each reporting timing.

2. The acquirer then creates resources that need to be included in the single `Plan` resource such as `ScopeItem` resources, `WorkItem` resources, `Artifact` resources with `Measure` resources as their target values using acquirer's project management tool.

3. The acquirer publishes these resources to the PROMCODE server using the project management tool it uses.

4. The PROMCODE server accepts PROMCODE resources and generates the URL for each resource.

5. The acquirer obtains URLs of these resources and notifies the supplier of the URLs when needed.

  

In non-shared server environment, steps 3-5 will be different. In a typical case, sharing is done using a `Plan` resource and a `Report` resource. Each of an acquirer and a supplier uses its own PROMCODE server that can be accessed by its partner. Then, the first step is for the acquirer to create a `Project` resource and a `Plan` resource and publish them on its PROMCODE server. Then, the suppler reads them from the acquirer's server. A supplier then creates corresponding copies of `Project` resource and `Plan` resource in its own server. A `source` attribute is used to identify the original resource for each copy resource. All other attributes of a copy of `Project` resource should be the same as those of the original. The content of the copy of a `Plan` resource should be the same as those of the original. A `Plan` resource has a collection of `ScopeItem` resources, `WorkItem` resources and `Artifact` resources. When the acquirer's server does not provide access to these resources, the supplier creates copies of these resources in its PROMCODE server. In that case, `source` attribute can be included in each copy to indicate its original source resource. `Measure` resource linked from an `Artifact` resource can be created without the `source` attribute. If the supplier's server does not support explicit creation of these resources, the supplier creates a copy of `Plan` resource into the server that ensures that these resources are included in the supplier's server by collaborating with the supplier's project management tool. The `Plan` resource in the supplier's server can use the `source` attribute to indicate that it is a copy of the original `Plan` in the acquirer's server.

## 2.4 Project Execution and Control

In the project execution and control use case, there are eight scenarios, i.e., Project Start, Status Reporting, Review and Actions for Scope Items, Review and Actions for Schedule Problems, Review and Actions for Quality Problems, Risk Management, Issue Management and Plan Change. As in the case of Project Planning section of 7.2, scenario descriptions in the subsequent subsections assume the use of common PROMCODE full server shared by the acquirer and the supplier. For non-shared environment, description will be given on the different portions when relevant.

### 2.4.1 Project Start

Project start scenario is the first step of the project execution and control use case. Systems and data flow of the scenario are described in 4.3.2 Project Start.

#### Preconditions

A supplier may use a project management tool different from that of the acquirer. Each tool needs to convert project information to PROMCODE resources and should be able to enter PROMCODE resources to the PROMCODE server and to retrieve them from the server.

Before the project starts, an acquirer and a supplier need to agree on rules and guidance to run and manage the project. This includes reporting frequency and timing, and criteria used for raising and managing issues and risks.

#### Scenario

  

1. An acquirer fills in `actualStartDate` of the `Project` resource with the acquirer's project management tool and submits it to the PROMCODE server.

2. The PROMCODE server updates the PROMCODE resource.

3. A supplier retrieves the resource and imports the information to supplier's project management tool.

4. The supplier starts execution and control of the project using its own project management tool that understands the information.

In non-shared environment, sharing is done by first the acquirer publishing the information in its server and the supplier reading the information. Then, the supplier enters the information to its own server using the supplier's project management tool.

### 2.4.2 Status Reporting

Status reporting scenario describes reporting activities that take place periodically at pre-agreed timing during the project execution. Reporting is done by using a `Report` resource that describes the project status against the `Plan` resource linked by the `correspondsTo` property. Based on the agreement between the acquirer and the supplier, a `Report` resource may describe the information of all the `ManagedItem` resources collected in the `Plan` resource, or may describe a subset of such `ManagedItem` resources. For the sake of simplicity, a `Report` resource that collects all the `ManagedItem` resources collected in the `Plan` resource is described in 4. Systems and Data Flow for Scenarios. Details of systems and data flow of the status reporting scenario are described in 4.3.3 Status Reporting.

#### Preconditions

The acquirer and the supplier agree on reporting information and its timing. Each time the supplier reports the information, a new `Report` resource is created in the PROMCODE server. The acquirer retrieves it to review the information. As in the previous sections, the acquirer and the supplier may use their own propriety project tools for their own management activities.

#### Scenario

1. The supplier creates a `Report` resource at agreed timing based on the agreed plan. A `Report` resource collects a subset of the resources collected by the `Plan` resource with updated properties of some of the resources in the subset. Specifically, for `ScopeItem` resources, `actualSize` property might be calculated, for `WorkItem` resources, `actualStartDate` and `actualEndDate` properties might be updated, and for `Artifact` resources, the `Report` resource may include a set of `Measurement` resources that measure the `Artifact` resources. The supplier enters the `Report` resource into the PROMCODE server.

2. The acquirer's project management tool retrieves the `Report` resource and the

  

resources included in the `Report` resource from the PROMCODE server.

In non-shared environment, this is done by the supplier creating a new `Report` resource into its own server, and the acquirer reading it. After the acquirer reads `Report` resource, a copy of the `Report` resource is created with `source` attribute, and values and properties of resources that are linked from the `Report` resource and are already in the acquirer's server are updated. The `source` attribute of such resources are used to map a copy resource to its original resource. Also, a copy of `Measurement` resource is created without `source` attribute in the acquirer's server for each `Measurement` resource in the `Report`.

### 2.4.3 Review and Actions for Scope Items

Review and actions for scope items is concerned with activities during the project execution when the value of scope item size becomes calculated. Systems and data flow of the scenario are described in 4.3.4 Review and Actions for Scope Items

#### Preconditions

Status reporting is done from the supplier to the acquirer at pre-agreed timing.

#### Scenario

1. The acquirer reviews the value of `actualSize` of each `ScopeItem` resource that is available from the `Report` resource, and compares it with the value of `plannedSize` of the `ScopeItem` resource.

2. Based on the comparison of all such `ScopeItem` resources, the acquirer takes one of the following actions.

    1. No formal action, and the project stays with the current plan.

    2. Escalate the situation to stakeholders for possible plan change. A plan change triggered from the change of a `ScopeItem` resource is likely to result in the change of a contract between the acquirer and the supplier with possible major impact on them. Therefore, the situation needs to be considered with extreme caution.

3. If a plan change is necessary, it will trigger the scenario of the plan change. See details in the 2.4.8 Plan Change scenario.

### 2.4.4 Review and Actions for Schedule Problems

Review and actions for schedule problem scenario is concerned with activities during the project execution that review and take actions on potential schedule problems. Systems and data flow of the scenario are described in 4.3.5 Review and Actions for Schedule Problems.

#### Preconditions

Status reporting is done from the supplier to the acquirer at pre-agreed timing.

**Scenario**

1. The acquirer reviews the difference between the previous `Report` resource and the current `Report` resource, and raises a concern if the following is observed.
    - Condition 1: Progress is not sufficient.
    - Condition 2: Risk of not meeting a schedule emerges with the current pace of progress. The acquirer may use past project data for risk identification.
2. The acquirer interacts with the supplier on further update.
    - Reasons for delay
    - Outlook of meeting a schedule
3. Based on the interaction, the acquirer takes one of the following actions.
    1. No formal action, but continue to monitor the situation.
    2. Take an action to mitigate the problem raised by the supplier without plan change. This may include new risk to be identified that results in creation of a `Risk` resource.
    3. Escalate the situation to stakeholders for possible plan change.
4. If a plan change is necessary, it will trigger the scenario of the plan change. See details in the 2.4.8 Plan Change scenario.

### 2.4.5 Review and Actions for Quality Problems

This scenario is concerned with activities during the project execution that review and take actions on potential quality problems. Systems and data flow of the scenario are described in 4.3.6 Review and Actions for Quality Problems.

**Preconditions**

Status reporting is done from suppliers to the acquirer at pre-agreed timing.

**Scenario**

1. The acquirer compares the previous `Report` resource and current `Report` resource and reviews the difference.
2. The acquirer raises a concern if the current level of quality is not sufficient and there is a risk of not meeting quality goals.

  

3. The acquirer interacts with the supplier on further update.
    1. Reasons of the current problem
    2. Outlook of meeting a goal
    3. Assess the impact to the overall project.
4. Based on the interaction, the acquirer takes one of the following actions.
    1. Stay with the current plan and monitor the situation.
    2. Take an action to mitigate the problem raised by the supplier without plan change. This may include new risk to be identified that results in creation of a `Risk` resource.
    3. Escalate the situation to stakeholders for possible plan change.
5. If plan change is necessary, it will trigger the scenario of the plan change. See details in the 2.4.8 Plan Change scenario.

### 2.4.6 Risk Management

Risk is an event that may or may not happen in the future with adverse effect to the project. Risk may typically be concerned with project cost, project schedule, project scope, quality, and so forth. Systems and data flow of the scenario are described in 4.3.7 Risk Management.

#### Preconditions

Risks are examined by reviewing the information in previous and current `Report` resources including resources linked from the `Report` resources such as `ScopeItem` resources, `WorkItem` resources, `Artifact` resources and `Measurement` resources with their values of properties. The acquirer identifies a collection of risks to be managed in the project and maintains the collection with `RiskCollection` resource.

#### Scenario

1. The acquirer evaluates each risk, such as cost overrun, schedule delay and shortage of skills for the project.
2. The acquirer asks a supplier to create a `Risk` resource by examining the information contained in the previous and current `Report` resources in the PROMCODE server.
3. The supplier creates a `Risk` resource for each identified risk and registers it in the `RiskCollection` resource in the PROMCODE server. The `Risk` resource has links to other resources that caused the risk to be raised.
4. The supplier notifies the registration of `Risk` resources to the acquirer.

  

5. The acquirer obtains the information on each `Risk` resource in the `RiskCollection` resource from the PROMCODE server.

6. The acquirer reviews a risk represented by each `Risk` resource in the `RiskCollection` resource and creates a risk mitigation plan. A risk mitigation plan can include monitoring the situation, actions to remove or reduce the risk, or closure of the risk for now. `Risk` resources reviewed include all `Risk` resources including non-closed `Risk` resources registered at previous risk review activities.

In non-shared environment, identifying, reporting and managing risks may be done by exchanging `Risk` and `Issue` resources.

### 2.4.7 Issue Management

In contrast to risk, issue is an event that already took place that requires attention and resolution to minimize negative impact to the project. Issue management scenario can take place after status reporting scenario, or it can take place on as needed basis during project execution. Systems and data flow of the scenario are described in 4.3.8 Issue Management.

#### Preconditions

The acquirer determines a collection of issues to be managed for the project and maintains the collection with `IssueCollection` resource.

#### Scenario

1. The acquirer evaluates each issue and determines the issues to be managed in the project. The acquirer interacts with the supplier for the evaluation if necessary.

2. The acquirer asks a supplier to create an `Issue` resource for each issue to be managed by examining the information contained in the previous and current `Report` resources in the PROMCODE server.

3. The supplier creates `Issue` resources and registers them in the `IssueCollection` resource on the PROMCODE server. Each `Issue` resource has links to other resources that caused the issue to be raised.

4. The supplier notifies the acquirer of the registration of the `Issue` resources.

5. The acquirer obtains the information on each registered `Issue` resource in the `IssueCollection` resource from the PROMCODE server.

6. The acquirer reviews the situation represented by each `Issue` resource in the `IssueCollection` resource and create an issue management plan to address it. An issue management plan can include mitigation actions, monitoring the situation, and closure. `Issue` resources to be reviewed include all non-closed `Issue` resources including `Issue` resources registered at previous issue review activities.

  

7. An `Issue` resource is closed if the acquirer decides that neither resolution action nor monitoring is needed for it.

Activities in non-shared environment are similar to those in Risk Management.

### 2.4.8 Plan Change

Plan change scenario occurs when change on some parts of a current plan becomes necessary. Typical cases may be a change on scope items, a change on work items, a change on artifacts, and a change on the collections of resources to be reported. Reasons for the change could be a change of requirements, possible schedule delay, and possible quality issues. Some of these changes might impact the project with the change such as `plannedEndDate` of the `Project` resource. Systems and data flow of the scenario are described in 4.3.9 Plan Change.

#### Preconditions

All stakeholders of the project agree on the change of a current project plan.

#### Scenario

1. The acquirer decides to change a current plan. The acquirer notifies the supplier of the plan change and proposes a revised plan.
2. A supplier reviews a revised plan. The acquirer interacts with the supplier for the change, and updates it if necessary.
3. The acquirer and the supplier agree.
4. The acquirer creates a new `Plan` resource to represent a revised plan and enters it to the PROMCODE server.
5. A supplier retrieves the new `Plan` resource from the PROMCODE server.

## 2.5 Project Closing

The use case for project closing has a single scenario closes the project. Detailed sequence of operations is described in 4.3.10 Project Closing.

#### Preconditions

All the target of `Artifact` resources has been achieved. All the `Risk` resources and `Issue` resources are closed.

#### Scenario

1. A supplier registers `actualEndDate` of the `Project` resource and `actualSize` of

  

all the `ScopeItem` resources to the PROMCODE server.

2. The supplier notifies the registration of the data to the acquirer.
3. The acquirer obtains the published actual data from PROMCODE server.
4. The acquirer preserves the whole project information for future work such as project analysis.

In non-shared environment, the scenario is modified by the supplier publishing information into its own server and the acquirer retrieving it from the server.

# 3. An Implementation of the PROMCODE specification with LDPC

This chapter describes one implementation that uses LDPC that accepts POST operations to create resources and that supports dynamic service discovery. Known existing implementations so far follow the implementation described here.

## 3.1 Containers

The implementation introduces an LDP container for each PROMCODE resource type. Therefore, the following containers are implemented.

*ProjectContainer, ScopeItemContainer, WorkItemContainer, ArtifactContainer, MeasureContainer, MeasurementContainer, RiskContainer, IssueContainer, RiskCollectionContainer, IssueCollectionContainer, PlanContainer*, and *ReportContainer.*

For class X, XContainer, the container for X, is defined with the following resource shape.

- **Name:** `XContainer`
- **Type URI:** `http://open-services.net/ns/promcode#XContainer`
- **Summary:** Shape resource of a `XContainer` resource.

**XContainer Properties**

| *Prefixed Name* | *Occurs* | *Read-only* | *Value-type* | *Representation* | *Range* | *Description* |
|---|---|---|---|---|---|---|
| `ldp:contains` | Zero-or-many | unspecified | Resource | Reference | `oslc_promcode:X` | `ldp:contains` is a relation between a `XContainer` resource and a resource of class `X`. |
| `rdf:type` | One-or-many | unspecified | Resource | Reference | `rdfs:Class` | The resource type URIs. |

A resource of X can be created by sending an `HTTP POST` request to the `XContainer` resource.

There is no container for *ManagedItem* and *ManagedItemCollection*, since they are abstract types that define common attributes and properties for their subclass resource types.

## 3.2 Service Discovery

There are two approaches described in the OSLC Core Version 3.0 Discovery specification [OSLCCore3-Discovery].

Theimplementation supports the dynamic resource discovery, namely the dynamic

  

incremental discovery as described in OSLC Core 3.0 [OSLCCore3].

### 3.2.1 Root Resource

The ProjectContainer resource serves as a root resource for the implementation. The ProjectContainer resource is an LDP Container for Project resources.

### 3.2.2 Links among containers and PROMCODE resources

When the implementation uses the dynamic incremental discovery, it must provide a container resource for each PROMCODE resource it supports.

The diagram below illustrates the containers and relationships with the PROMCODE resources.

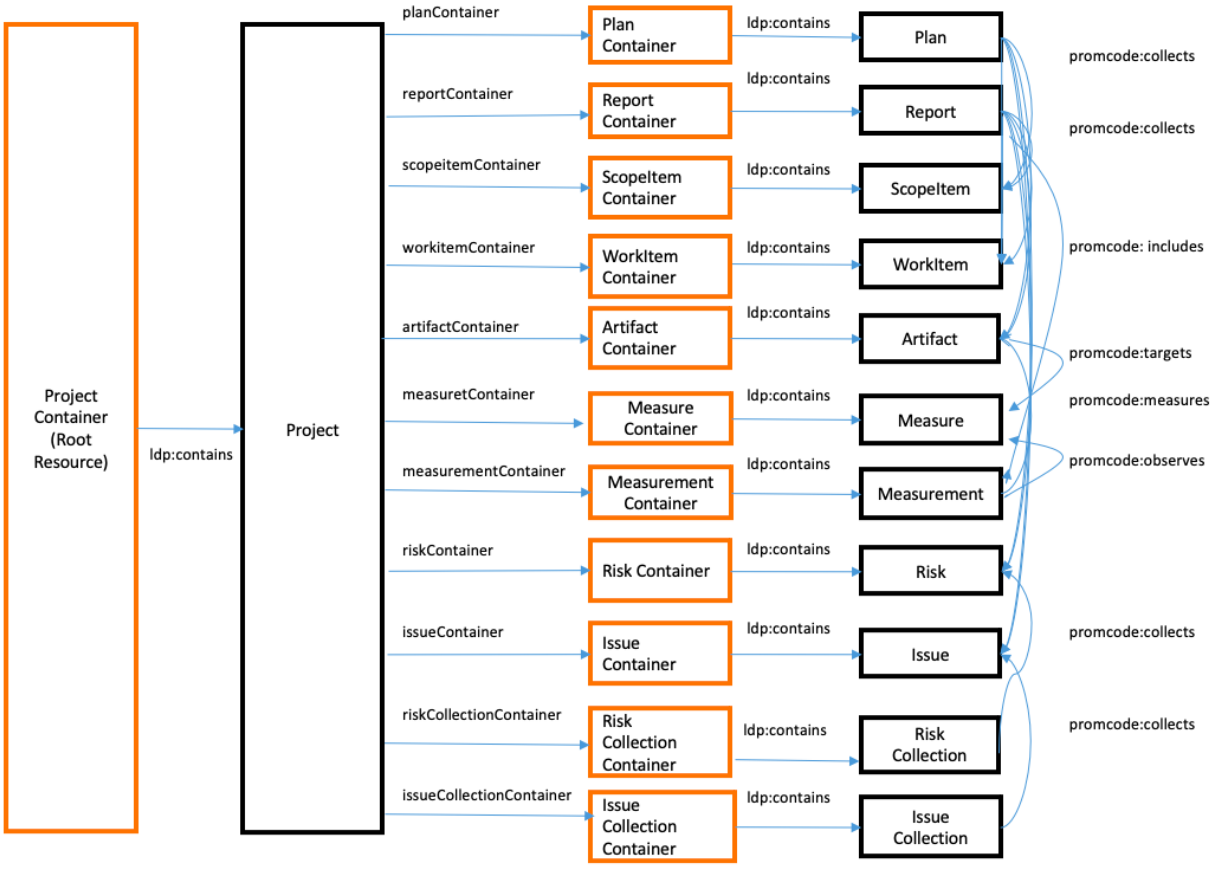


Fig. 3 PROMCODE LDP Containers

Black boxes are domain resources and orange boxes are the container resources. To enable the relationships, a project resource needs to have a property to link it to the container for resources of type that is implemented in the project. For example, if an implementation of a project supports plan resources, the project resource has a property `planContainer` to link the resource to `PlanContainer` resource.

In the sample implementation, it is assumed that all containers are implemented as LDP Basic Containers.

### 3.2.3 Discovery

Steps with dynamic incremental discovery proceed as follows. A PROMCODE client sends an HTTP request to a PROMCODE server to access the [ProjectContainer], the top-level resource the server manages. After getting a list of [Project] resources from the container, it can select the [Project] resource of interest. Then, the client can determine if there is a link from the [Project] resource to each of container resources as in Fig. 3 PROMCODE LDP Containers . If a link to a container exists, resources created by the container are supported by the server. If not, the resources are not supported. Once the existence of each container is assured, the client can send an HTTP OPTIONS message to the container to get a list of all the services supported by the container. Similarly, once a domain resource is created, the client can send an HTTP OPTIONS message to the resource to get a list of all the services supported by the resource. In this way, a PROMCODE client communicates with a PROMCODE server to get the information on all the resource types the server supports and all the services supported for each of such resource types.

  

# 4. Systems and Data Flow for Scenarios

This chapter describes detailed flows of activities including sequential diagrams among tools and users for each use case scenario described in Chapter 1 for the implementation described in Chapter 2.

## 4.1 Tools

In this chapter, the following tools are used to illustrate the systems and data flow sequences.

PROMCODE server
: A PROMCODE full server that supports OSLC PROMCODE services defined in the PROMCODE Specification. In addition to being a full server, the server also supports the POST and PUT methods to create and update PROMCODE resources as explained in 2.2 Environment for Use Cases. For creation of resources, the server supports LDP containers. The LDP container of the Project resource can be retrieved by getting /pm URL of the server. LDP containers for other PROMCODE resources can be retrieved from the Project resource. For example:

EXAMPLE 1: Example of a Project resource which contains LDP containers of PROMCODE Resources

```
<http://example.com/pm/projects/100>
  a oslc_promcode:Project ;
  dcterms:identifier "100" ;
  dcterms:title "PJ1" ;
  oslc_promcode:plannedStartDate "2017-03-01T00:00:00Z" ;
  oslc_promcode:plannedEndDate "2017-12-31T00:00:00Z" ;
  oslc_promcode:metricOfScopeItemSize <http://example.com/ns/m
etric#FunctionSize> ;
  oslc_promcode:unitOfScopeItemSize <http://example.com/ns/uni
t#FunctionPoint> ;
  example:planContainer <http://example.com/pm/projects/100/pl
anContainer> ;
  example:reportContainer <http://example.com/pm/projects/100/
reportContainer> ;
  example:scopeItemContainer <http://example.com/pm/projects/1
00/scopeItemContainer> ;
  example:workItemContainer <http://example.com/pm/projects/10
0/workItemContainer> ;
  example:artifactContainer <http://example.com/pm/projects/10
0/artifactContainer> ;
  example:measurementContainer <http://example.com/pm/projects
/100/measurementContainer> ;
```

```
  example:measureContainer <http://example.com/pm/projects/100
/measureContainer> ;
  example:riskContainer <http://example.com/pm/projects/100/ri
skContainer> ;
  example:issueContainer <http://example.com/pm/projects/100/i
ssueContainer> ;
  example:riskCollectionContainer <http://example.com/pm/proje
cts/100/riskCollectionContainer> ;
  example:issueCollectionContainer <http://example.com/pm/proj
ects/100/issueCollectionContainer> .
```

Both the acquirer and the supplier have an access to the common PROMCODE server.

PROMCODE client
: A PROMCODE client that provides the user with the capability of invoking the following operations. The acquirer and the supplier must install a PROMCODE client in each environment:

- Create a project and a plan
- Update a project
- Update a plan
- Create a report
- Update a report
- List projects
- List plans of a project
- Create a risk collection
- Create a risk
- Close a risk
- Create an issue collection
- Create an issue
- Close an issue

## 4.2 Assumption (Precondition)

- Initial `ScopeItem`, `WorkItem` and `Artifact` have already been defined as a result of prior agreement between the acquirer and the supplier.
- `Metric` and `Unit` for `metricOfScopeItemSize` and `unitOfScopeItemSize` of the project have been defined as in the following example.
- `Metric` and `Unit` for `Measure` of each `Artifact` have been defined as in the following examples.

EXAMPLE 2: ScopeItemSize

```
Metric: http://example.com/ns/metric#FunctionSize
```

  

```
Unit: http://example.com/ns/unit#FunctionPoint
```

EXAMPLE 3: Measure for Source Code artifact

```
Metric: http://example.com/ns/metric#Sloc
Unit: http://example.com/ns/unit#Loc
```

EXAMPLE 4: Measure for Test Case artifact

```
Metric: http://example.com/ns/metric#Defects
Unit: http://example.com/ns/unit#Defect
```

- Status values for `Issue` and `Risk` resources have been defined as in the following example.

EXAMPLE 5: Status for Issue and Risk

```
Status Value: http://example.com/ns/status#open
Status Value: http://example.com/ns/status#inprogress
Status Value: http://example.com/ns/status#closed
```

## 4.3 Scenarios

### 4.3.1 Project Planning

This scenario illustrates how the PROMCODE services can be used in the process of project planning. The corresponding use case scenario is described in 2.3 Project Planning.

Project planning is the first phase in the lifecycle of a project.

#### *4.3.1.1 Scenario*

The first step of project planning is to create a `Project` resource and a `Plan` resource on the PROMCODE server through the PROMCODE client.

1. The acquirer executes "Create a project and a plan" operation and enters the following information into the PROMCDE client.
    - Project name : "PJ1"
    - Plan name : "PN1"

  

- Planned start date : "2017/3/1"
- Planned end date: "2017/12/31"
- Metric and Unit linked from the project : "Function Size" and "1 Function Point".

2. The PROMCODE client submits an HTTP GET request to the PROMCODE Sever and retrieves the `ProjectContainer` resource. A `ProjectContainer` resource has already been created in the beginning by default.

EXAMPLE 6: Request - retrieving a LDP container for the Project resource

```
GET /pm/ HTTP/1.1
Host: example.com
Accept: text/turtle
```

EXAMPLE 7: Response - retrieving a LDP container for the Project resource

```
HTTP/1.1 200 OK
Content-Type: text/turtle; charset=UTF-8
Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type", <
http://www.w3.org/ns/ldp#Resource>; rel="type"
Allow: OPTIONS,HEAD,GET,POST,PUT,PATCH
Accept-Post: text/turtle, application/ld+json
Content-Length: 250
ETag: W/'123456780'

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .
@prefix example: <http://example.com/ns/> .

<http://example.com/pm/> a example:ProjectContainer .
```

3. The PROMCODE client creates the content of the new `Project` resource to be created with title "PJ1" in RDF (e.g. Turtle format) with the information set in step 1.

4. The PROMCODE client submits an HTTP POST request with the content created in step 3 to the LDP container URL for the Project resource.

EXAMPLE 8: Request - creating a Project resource

```
POST /pm/ HTTP/1.1
```

  

```
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:Project ;
  dcterms:title "PJ1" ;
  oslc_promcode:plannedStartDate "2017-03-01T00:00:00Z" ;
  oslc_promcode:plannedEndDate "2017-12-31T00:00:00Z" ;
  oslc_promcode:metricOfScopeItemSize <http://example.com/ns/m
etric#FunctionSize> ;
  oslc_promcode:unitOfScopeItemSize <http://example.com/ns/uni
t#FunctionPoint> .
```

5. The PROMCODE server creates `Project` resource PJ1 and returns the URL to the PROMCODE client.

EXAMPLE 9: Response - creating a Project resource

```
HTTP/1.1 201 Created
Location: http://example.com/pm/projects/100
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

6. The PROMCODE client submits an HTTP GET request to the PROMCODE server, retrieves PJ1, and extracts containers.

EXAMPLE 10: Request - container retrieval after resource created

```
GET /pm/projects/100 HTTP/1.1
Host: example.com
Accept: text/turtle
```

EXAMPLE 11: Response - container retrieval after resource created

```
HTTP/1.1 200 OK
Content-Type: text/turtle; charset=UTF-8
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
```

```
Allow: OPTIONS,HEAD,GET,PUT,PATCH,DELETE
Accept-Post: text/turtle, application/ld+json
Content-Length: 250
ETag: W/'123456781'

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .
@prefix example: <http://example.com/ns/> .

<http://example.com/pm/projects/100>
  a oslc_promcode:Project ;
  dcterms:identifier "100" ;
  dcterms:title "PJ1" ;
  oslc_promcode:plannedStartDate "2017-03-01T00:00:00Z" ;
  oslc_promcode:plannedEndDate "2017-12-31T00:00:00Z" ;
  oslc_promcode:metricOfScopeItemSize <http://example.com/ns/m
etric#FunctionSize> ;
  oslc_promcode:unitOfScopeItemSize <http://example.com/ns/uni
t#FunctionPoint> ;
  example:planContainer <http://example.com/pm/projects/100/pl
anContainer> ;
  example:reportContainer <http://example.com/pm/projects/100/
reportContainer> ;
  example:scopeItemContainer <http://example.com/pm/projects/1
00/scopeItemContainer> ;
  example:workItemContainer <http://example.com/pm/projects/10
0/workItemContainer> ;
  example:artifactContainer <http://example.com/pm/projects/10
0/artifactContainer> ;
  example:measurementContainer <http://example.com/pm/projects
/100/measurementContainer> ;
  example:measureContainer <http://example.com/pm/projects/100
/measureContainer> ;
  example:riskContainer <http://example.com/pm/projects/100/ri
skContainer> ;
  example:issueContainer <http://example.com/pm/projects/100/i
ssueContainer> ;
  example:riskCollectionContainer <http://example.com/pm/proje
cts/100/riskCollectionContainer> ;
  example:issueCollectionContainer <http://example.com/pm/proj
ects/100/issueCollectionContainer> .
```

7. The PROMCODE client creates the content of the new `Plan` resource with title "PN1" in RDF with the information set in step 1.

  

8. The PROMCODE client submits an HTTP POST request with the content created in step 7 to the LDP container URL for the Project resource.

EXAMPLE 12: Request - creating a Plan resource

```
POST /pm/projects/100/planContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:Plan ;
  dcterms:title "PN1" ;
  oslc_promcode:belongsTo <http://example.com/pm/projects/100>
 .
```

9. The PROMCODE server creates `Plan` PN1, and returns its URL to the acquirer.

EXAMPLE 13: Response - creating a Plan resource

```
HTTP/1.1 201 Created
Location: http://example.com/pm/plans/200
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

The next step is to create `ScopeItem` resources, `WorkItem` resources, and `Artifact` resources for the `Plan` resource PN1.

10. The acquirer creates all `ScopeItem` resources, `WorkItem` resources, and `Artifact` resources for PN1. In many cases, information on these resources and their relationships is summarized with spreadsheet like the following.
    A color in the heading rows indicates the type of resources described by each column; green for `ScopeItem` resources, orange for `Artifact` resources and blue for `WorkItem` resources. In this sample spreadsheet, a very simple case is considered in which a single `Artifact` is produced for each `ScopeItem`, i.e., there is a single `Artifact` resource that is linked to each `ScopeItem` resource with `producedFor` link. Similarly, for each `ScopeItem` resource and therefore for each `Artifact` resource, it is assumed that there is a single coding `WorkItem` resource and also a single testing `WorkItem` resource, respectively. Then, an entire row that has a

  

`ScopeItem` resource also contains the `Artifact` resource that are linked to the `ScopeItem` resource, and two `WorkItem` resources that are linked to the `Artifact` resource, i.e., the `WorkItem` resource for testing and the `WorkItem` resource for unit test.

| Item | | | | Asset | | | | | | Coding | | | | Unit Test | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ID | Title | Size (Function Point) | | Program code | | Measure - code size (Kloc) | | Measure - # of defects | | Start | | End | | Start | | End | |
| | | Planned | Actual | Description | file name | Planned | Actual | Planned | Actual | Panned | Actual | Panned | Actual | Planned | Actual | Planned | Actual |
| 1.1 | UI for making a reservation | 20 | | UI code for making a reservation | | 10 | | 50 | | 2017/3/5 | | 2017/3/31 | | 2017/4/15 | | 2017/5/15 | |
| 1.2 | UI for updating a reservation | 10 | | UI code for updating a reservation | | 5 | | 25 | | 2017/3/15 | | 2017/4/15 | | 2017/4/15 | | 2017/5/15 | |

Fig. 4 Data in the PROMCODE client

11. The PROMCODE client submits an HTTP GET request to the PROMCODE server to retrieve PN1 and the associated `Project` resource PJ1. PROMCODE client then extracts URLs to the containers stored in PJ1.

EXAMPLE 14: Request - retrieving Plan resource PN1

```
GET /pm/plans/200 HTTP/1.1
Host: example.com
Accept: text/turtle
```

EXAMPLE 15: Response - retrieving Plan resource PN1

```
HTTP/1.1 200 OK
Content-Type: text/turtle; charset=UTF-8
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Allow: OPTIONS,HEAD,GET,PUT,PATCH,DELETE
Accept-Post: text/turtle, application/ld+json
Content-Length: 250
ETag: W/'123456782'

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/plans/200>
  a oslc_promcode:Plan ;
  dcterms:identifier "200" ;
  dcterms:title "PN1" ;
  oslc_promcode:belongsTo <http://example.com/pm/projects/100>
 .
```

EXAMPLE 16: Request - retrieving project PJ1

```
GET /pm/projects/100 HTTP/1.1
Host: example.com
```

```
Accept: text/turtle
```

EXAMPLE 17: Response - retrieving project PJ1

```
HTTP/1.1 200 OK
Content-Type: text/turtle; charset=UTF-8
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Allow: OPTIONS,HEAD,GET,PUT,PATCH,DELETE
Accept-Post: text/turtle, application/ld+json
Content-Length: 250
ETag: W/'123456781'

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .
@prefix example: <http://example.com/ns/> .

<http://example.com/pm/projects/100>
  a oslc_promcode:Project ;
  dcterms:identifier "100" ;
  dcterms:title "PJ1" ;
  oslc_promcode:plannedStartDate "2017-03-01T00:00:00Z" ;
  oslc_promcode:plannedEndDate "2017-12-31T00:00:00Z" ;
  oslc_promcode:metricOfScopeItemSize <http://example.com/ns/m
etric#FunctionSize> ;
  oslc_promcode:unitOfScopeItemSize <http://example.com/ns/uni
t#FunctionPoint> ;
  example:planContainer <http://example.com/pm/projects/100/pl
anContainer> ;
  example:reportContainer <http://example.com/pm/projects/100/
reportContainer> ;
  example:scopeItemContainer <http://example.com/pm/projects/1
00/scopeItemContainer> ;
  example:workItemContainer <http://example.com/pm/projects/10
0/workItemContainer> ;
  example:artifactContainer <http://example.com/pm/projects/10
0/artifactContainer> ;
  example:measurementContainer <http://example.com/pm/projects
/100/measurementContainer> ;
  example:measureContainer <http://example.com/pm/projects/100
/measureContainer> ;
  example:riskContainer <http://example.com/pm/projects/100/ri
skContainer> ;
  example:issueContainer <http://example.com/pm/projects/100/i
ssueContainer> ;
```

  

```
  example:riskCollectionContainer <http://example.com/pm/proje
cts/100/riskCollectionContainer> ;
  example:issueCollectionContainer <http://example.com/pm/proj
ects/100/issueCollectionContainer> .
```

12. The PROMCODE client converts the contents to `ScopeItem` resources, `WorkItem` resources, and `Artifact` resources in RDF. The PROMCODE client also creates the contents of `Measure` resources in RDF that will be linked from each `Artifact` resource by its `targets` property with inline representation.

13. The PROMCODE client submits a set of HTTP POST requests to containers to create `ScopeItem` resources, `WorkItem` resources, and `Artifact` resources. The PROMCODE server creates these `ScopeItem` resources, `WorkItem` resources, and `Artifact` resources, and returns the URLs to the PROMCODE client.

The following code shows an example for each resource type.

EXAMPLE 18: Request - creating a ScopeItem resource

```
POST /pm/projects/100/scopeItemContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:ScopeItem ;
  dcterms:title "SI1" ;
  dcterms:description "UI for making a reservation" ;
  oslc_promcode:plannedSize "20" .
```

EXAMPLE 19: Response - creating a ScopeItem resource

```
HTTP/1.1 201 Created
Location: http://example.com/pm/scopeitems/1010
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

EXAMPLE 20: Request - creating a Measure resource

```
POST /pm/projects/100/measureContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:Measure ;
  dcterms:title "ME1" ;
  dcterms:description "Code size (kloc)" ;
  oslc_promcode:valueOfMeasure "10" ;
  oslc_promcode:metricOfMeasure <http://example.com/ns/metric#
Sloc> ;
  oslc_promcode:unitOfMeasure <http://example.com/ns/unit#Loc>
 .
```

EXAMPLE 21: Response - creating a Measure resource

```
HTTP/1.1 201 Created
Location: http://example.com/pm/measures/6010
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

EXAMPLE 22: Request - creating an Artifact resource

```
POST /pm/projects/100/artifactContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:Artifact ;
  dcterms:title "AR1" ;
  dcterms:description "Source Code for UI for making a reserva
tion" ;
  oslc_promcode:producedFor <http://example.com/pm/scopeitem/1
```

  

```
010> ;
  oslc_promcode:targets <http://example.com/pm/measures/6010>,
 <http://example.com/pm/measures/6011> .
```

EXAMPLE 23: Response - creating an Artifact resource

```
HTTP/1.1 201 Created
Location: http://example.com/pm/artifacts/2010
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

EXAMPLE 24: Request - creating a WorkItem resource

```
POST /pm/projects/100/workItemContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:WorkItem ;
  dcterms:title "WI1" ;
  dcterms:description "Implement UI for making a reservation"
;
  oslc_promcode:plannedStartDate "2017-03-05T09:00:00Z" ;
  oslc_promcode:plannedEndDate "2017-03-31T18:00:00Z" ;
  oslc_promcode:requiredBy <http://example.com/pm/artifacts/20
10> .
```

EXAMPLE 25: Response - creating a WorkItem resource

```
HTTP/1.1 201 Created
Location: http://example.com/pm/workitems/3010
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

14. The PROMCODE client adds `ScopeItem` resources, `WorkItem` resources, and `Artifact` resources as the values of `collects` property of `Plan` resource PN1 so

  

that the supplier can retrieve the information on these resources from PN1.

15. The PROMCODE client submits an HTTP PUT request to update PN1.

EXAMPLE 26: Request - updating Plan resource PN1

```
PUT /pm/plans/200 HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle
If-Match: W/'123456782'

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/plans/200>
  a oslc_promcode:Plan ;
  dcterms:identifier "200" ;
  dcterms:title "PN1" ;
  oslc_promcode:belongsTo <http://example.com/pm/projects/100>
 ;
  oslc_promcode:collects <http://example.com/pm/scopeitems/101
0>, <http://example.com/pm/scopeitems/1011>,
                <http://example.com/pm/artifacts/2010>, <http:
//example.com/pm/artifacts/2011>,
                <http://example.com/pm/workitems/3010>, <http:
//example.com/pm/workitems/3011>,
                <http://example.com/pm/workitems/3012>, <http:
//example.com/pm/workitems/3013> .
```

EXAMPLE 27: Response - updating Plan resource PN1

```
HTTP/1.1 204 No Content
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
ETag: W/'123456783'
```

16. The acquirer notifies the supplier of the URL of PN1 for a review by the supplier.

#### *4.3.1.2 Summary Sequence Diagram*

The following sequence diagram summarizes Project Planning scenario:

  

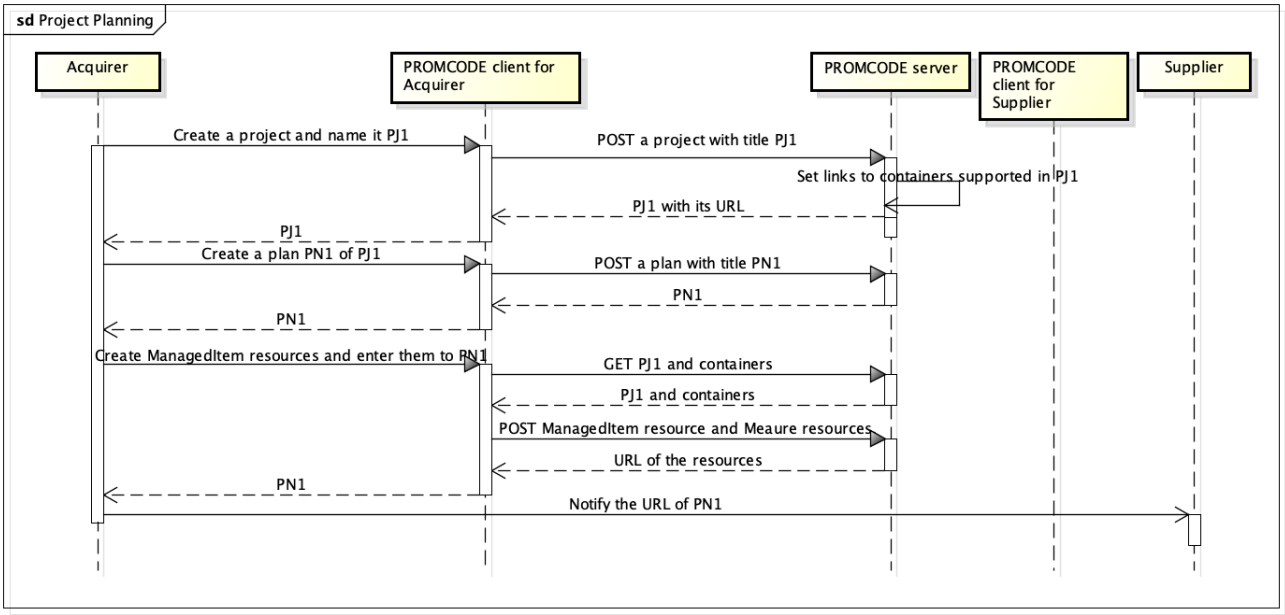


Fig. 5 Project Planning

### 4.3.2 Project Start

This scenario illustrates how the PROMCODE services can be used for starting a project. The corresponding use case scenario is described in 2.4.1 Project Start.

Project start is the first phase of the project execution and control use case.

#### *4.3.2.1 Scenario*

The first step of project start is to set the actual start date of `Project` resource PJ1.

1. The acquirer sets the URL of PJ1 to the PROMCODE client.
2. The PROMCODE client submits an HTTP GET request to the PROMCODE server and retrieves PJ1.
3. The acquirer executes "Update a project" operation and sets the actual start date with the PROMCODE client.
4. The PROMCODE client submits an HTTP PUT request to the PROMCODE server.

EXAMPLE 28: Request - updating Project resource PJ1

```
PUT http://example.com/pm/projects/100 HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle
```

  

```
If-Match: W/'123456781'

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .
@prefix example: <http://example.com/ns/> .

<http://example.com/pm/projects/100>
  a oslc_promcode:Project ;
  dcterms:identifier "100" ;
  dcterms:title "PJ1" ;
  oslc_promcode:plannedStartDate "2017-03-01T00:00:00Z" ;
  oslc_promcode:plannedEndDate "2017-12-31T00:00:00Z" ;
  oslc_promcode:actualStartDate "2017-03-01T00:00:00Z" ;
  oslc_promcode:metricOfScopeItemSize <http://example.com/ns/m
etric#FunctionSize> ;
  oslc_promcode:unitOfScopeItemSize <http://example.com/ns/uni
t#FunctionPoint> ;
  example:planContainer <http://example.com/pm/projects/100/pl
anContainer> ;
  example:reportContainer <http://example.com/pm/projects/100/
reportContainer> ;
  example:scopeItemContainer <http://example.com/pm/projects/1
00/scopeItemContainer> ;
  example:workItemContainer <http://example.com/pm/projects/10
0/workItemContainer> ;
  example:artifactContainer <http://example.com/pm/projects/10
0/artifactContainer> ;
  example:measurementContainer <http://example.com/pm/projects
/100/measurementContainer> ;
  example:measureContainer <http://example.com/pm/projects/100
/measureContainer> ;
  example:riskContainer <http://example.com/pm/projects/100/ri
skContainer> ;
  example:issueContainer <http://example.com/pm/projects/100/i
ssueContainer> ;
  example:riskCollectionContainer <http://example.com/pm/proje
cts/100/riskCollectionContainer> ;
  example:issueCollectionContainer <http://example.com/pm/proj
ects/100/issueCollectionContainer> .
```

EXAMPLE 29: Response - updating Project resource PJ1

```
HTTP/1.1 204 No Content
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
```

  

```
ETag: W/'123456784'
```

5. The acquirer notifies the supplier of the URL of PN1 to review it.

When the supplier receives the notification from the acquirer, the supplier reviews `Plan` resource PN1 using PROMCODE client and starts its execution.

6. The supplier sets the URL of PN1 to the PROMCODE client.
7. The PROMCODE client submits an HTTP GET request to the PROMCODE server and retrieves PN1.

EXAMPLE 30: Request - retrieving Plan resource PN1

```
GET /pm/plans/200 HTTP/1.1
Host: example.com
Accept: text/turtle
```

EXAMPLE 31: Response - retrieving Plan resource PN1

```
HTTP/1.1 200 OK
Content-Type: text/turtle; charset=UTF-8
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Allow: OPTIONS,HEAD,GET,PUT,PATCH,DELETE
Accept-Post: text/turtle, application/ld+json
Content-Length: 250
ETag: W/'123456783'

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/plans/200>
  a oslc_promcode:Plan ;
  dcterms:identifier "200" ;
  dcterms:title "PN1" ;
  oslc_promcode:belongsTo <http://example.com/pm/projects/100>
 ;
  oslc_promcode:collects <http://example.com/pm/scopeitems/101
0>, <http://example.com/pm/scopeitems/1011>,
                <http://example.com/pm/artifacts/2010>, <http:
//example.com/pm/artifacts/2011>,
                <http://example.com/pm/workitems/3010>, <http:
//example.com/pm/workitems/3011>,
```

  

```
                <http://example.com/pm/workitems/3012>, <http:
//example.com/pm/workitems/3013> .
```

8. The PROMCODE client submits an HTTP GET request to the PROMCODE server and retrieves `ScopeItem` resources for `Plan` resource PN1.

EXAMPLE 32: Request - retrieving a ScopeItem resource

```
GET /pm/scopeitems/1010 HTTP/1.1
Host: example.com
Accept: text/turtle
```

EXAMPLE 33: Response - retrieving a ScopeItem resource

```
HTTP/1.1 200 OK
Content-Type: text/turtle; charset=UTF-8
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Allow: OPTIONS,HEAD,GET,PUT,PATCH,DELETE
Accept-Post: text/turtle, application/ld+json
Content-Length: 250
ETag: W/'123456785'

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/scopeitems/1010>
  a oslc_promcode:ScopeItem ;
  dcterms:title "SI1" ;
  dcterms:description "UI for making a reservation" ;
  oslc_promcode:plannedSize "20" .
```

9. The supplier reviews the plan PN1 and starts its execution.

#### *4.3.2.2 Summary Sequence Diagram*

The following sequence diagram summarizes Project Start scenario:

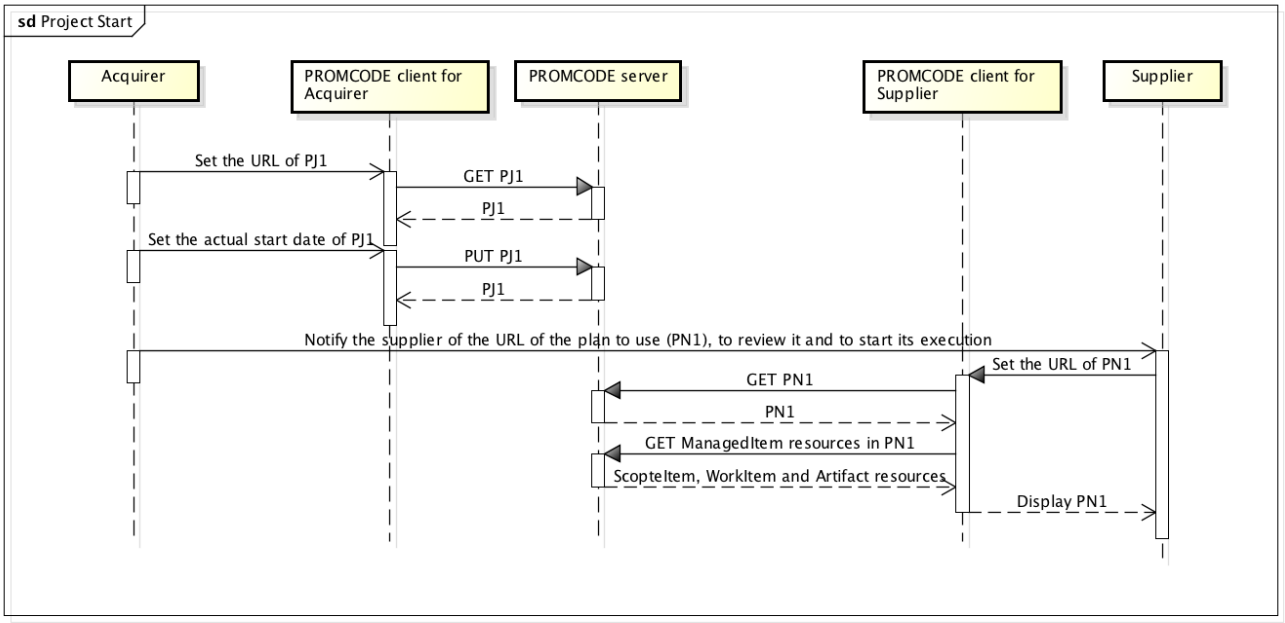


Fig. 6 Project Start

### 4.3.3 Status Reporting

This scenario illustrates how the PROMCODE services can be used in the process of status reporting. The corresponding use case scenario is described in 2.4.2 Status Reporting.

#### *4.3.3.1 Scenario*

First, the supplier retrieves `Plan` resource PN1 and `ScopeItem` resources, `WorkItem` resources and `Artifact` resources that are collected in PN1.

1. The supplier inputs the URL of PN1 to the PROMCODE client.
2. The PROMCODE client submits an HTTP GET request to the PROMCODE server and retrieves PN1 and associated `Project` resource PJ1.
3. The PROMCODE client parses PN1 and obtains URLs of resources collected in PN1, i.e., `ScopeItem` resources, `WorkItem` resources and `Artifact` resources and `Measure`.
4. The PROMCODE client then submits a GET request to the PROMCODE server and retrieves the `ScopeItem` resources, `WorkItem`, resources, `Artifact` resources.
5. The PROMCODE client parses the resources and displays them.

Next, the supplier selects resources, and collects them into a `Report` resource.

6. The supplier selects resources and executes "Create a report" operation on the

  

PROMCODE client.

7. The PROMCODE client creates a content of the new `Report` resource with title "RP1" in RDF.

8. The PROMCODE client extracts the LDP container URL for the `Report` resource from PJ1, and submits an HTTP POST request to the container with the content created in step 7.

EXAMPLE 34: Request - creating a Report resource

```
POST /pm/projects/100/reportContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:Report ;
  dcterms:title "RP1" ;
  dcterms:date "2017-03-15T18:00:00Z" ;
  oslc_promcode:belongsTo <http://example.com/pm/projects/100>
 ;
  oslc_promcode:collects <http://example.com/pm/scopeitems/101
0>, <http://example.com/pm/scopeitems/1011>,
                <http://example.com/pm/artifacts/2010>, <http:
//example.com/pm/artifacts/2011>,
                <http://example.com/pm/workitems/3010>, <http:
//example.com/pm/workitems/3011>,
                <http://example.com/pm/workitems/3012>, <http:
//example.com/pm/workitems/3013> .
```

9. The PROMCODE server returns the URL of `Report` resource RP1.

EXAMPLE 35: Response - creating a Report resource

```
HTTP/1.1 201 Created
Location: http://example.com/pm/reports/300
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

  

Next, the supplier updates `Report` resource RP1.

10. The supplier executes "Update a report" operation with the PROMCODE client.
11. The supplier updates the `ScopeItem` resources, `WorkItem` resources, and `Artifact` resources, and creates `Measurement` resources and `Measure` resources, based on the current status of the project.
12. The PROMCODE client creates a content of the new `Measure` resource in RDF with the update by the supplier.
13. The PROMCODE client extracts the LDP container URL for the `Measure` resource from `Project` resource PJ1, and submits HTTP POST request to the container with the content created in step 12.

EXAMPLE 36: Request - creating a Measure resource

```
POST /pm/projects/100/measureContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:Measure ;
  dcterms:title "ME3" ;
  dcterms:description "Code size (kloc)" ;
  oslc_promcode:valueOfMeasure "15" ;
  oslc_promcode:metricOfMeasure <http://example.com/ns/metric#
Sloc> ;
  oslc_promcode:unitOfMeasure <http://example.com/ns/unit#Loc>
 .
```

EXAMPLE 37: Response - creating a Measure resource

```
HTTP/1.1 201 Created
Location: http://example.com/pm/measures/6012
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

14. The PROMCODE client creates a content of the new `Measurement` resource in RDF.

  

15. The PROMCODE client extracts the LDP container URL for the `Measurement` resource from `Project` resource PJ1, and submits HTTP POST request to the container with the content created in step 12.

EXAMPLE 38: Request - creating a Measurement resource

```
POST /pm/projects/100/measurementContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:Measurement ;
  dcterms:date "2017-03-15T18:00:00Z" ;
  oslc_promcode:measures <http://example.com/pm/artifacts/2010
> ;
  oslc_promcode:observes <http://example.com/pm/measures/6012>
 .
```

EXAMPLE 39: Response - creating a Measurement resource with Measure resources

```
HTTP/1.1 201 Created
Location: http://example.com/pm/measurements/5010
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

16. The PROMCODE client submits an HTTP PUT request to the PROMCODE server to update `Report` resource RP1 to include the `Measurement` resource as well as to update other resources in the report.

EXAMPLE 40: Report resource

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/reports/300>
  a oslc_promcode:Report ;
  dcterms:identifier "300" ;
```

  

```
  dcterms:title "RP1" ;
  dcterms:date "2017-03-15T18:00:00Z" ;
  oslc_promcode:belongsTo <http://example.com/pm/projects/100>
 ;
  oslc_promcode:collects <http://example.com/pm/scopeitems/101
0>, <http://example.com/pm/scopeitems/1011>,
                <http://example.com/pm/artifacts/2010>, <http:
//example.com/pm/artifacts/2011>,
                <http://example.com/pm/workitems/3010>, <http:
//example.com/pm/workitems/3011>,
                <http://example.com/pm/workitems/3012>, <http:
//example.com/pm/workitems/3013> ;
  oslc_promcode:includes <http://example.com/pm/measurements/5
010> .
```

EXAMPLE 41: ScopeItem resource

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/scopeitems/1010>
  a oslc_promcode:ScopeItem ;
  dcterms:identifier "1010" ;
  dcterms:title "SI1" ;
  dcterms:description "UI for making a reservation" ;
  oslc_promcode:plannedSize "20" .
```

EXAMPLE 42: WorkItem resource

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/workitems/3010>
  a oslc_promcode:WorkItem ;
  dcterms:identifier "3010" ;
  dcterms:title "WI1" ;
  dcterms:description "Implement UI for making a reservation"
;
  oslc_promcode:plannedStartDate "2017-03-05T09:00:00Z" ;
  oslc_promcode:actualStartDate "2017-03-05T09:00:00Z" ;
```

  

```
  oslc_promcode:plannedEndDate "2017-03-31T18:00:00Z" ;
  oslc_promcode:requiredBy <http://example.com/pm/artifacts/20
10> .
```

17. The supplier notifies the acquirer of the URL of RP1.

Upon the notification, the acquirer starts reviewing the status of the project.

18. The acquirer gets the URL of RP1, then set it to the PROMCODE client.
19. The PROMCODE client submits an HTTP GET request to the PROMCODE server and retrieves RP1.
20. The PROMCODE client parses RP1 and gets the list of collected resources, i.e., `ScopeItem` resources, `WorkItem` resources and `Artifact` resources.
21. The PROMCODE client then submits an HTTP GET request to the PROMCODE server and retrieves the `ScopeItem` resources, `WorkItem` resources, `Artifact` resources, and `Measurement` resources.
22. The acquirer reviews the resources.

#### *4.3.3.2 Summary Sequence Diagram*

The following sequence diagram summarizes Status Reporting scenario:

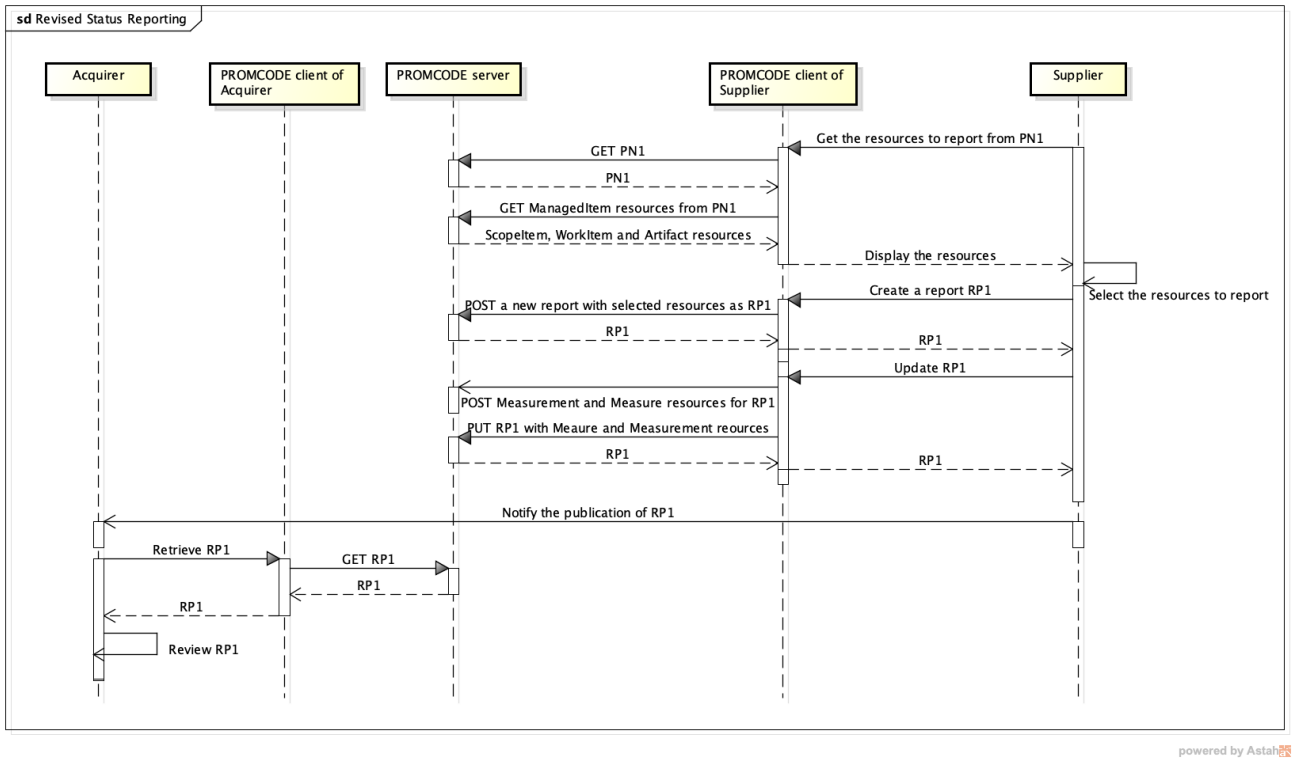


  

Fig. 7 Status Reporting

### 4.3.4 Review and Actions for Scope Items

The corresponding use case scenario is described in 2.4.3 Review and Actions for Scope Items. This scenario doesn't have any interaction with systems.

### 4.3.5 Review and Actions for Schedule Problems

The corresponding use case scenario is described in 2.4.4 Review and Actions for Schedule Problems. This scenario doesn't have any interaction with systems.

### 4.3.6 Review and Actions for Quality Problems

The corresponding use case scenario is described in 2.4.5 Review and Actions for Quality Problems This scenario doesn't have any interaction with systems.

### 4.3.7 Risk Management

This scenario illustrates how the PROMCODE services can be used in the process of risk management. The corresponding use case scenario is described in 2.4.6 Risk Management

#### *4.3.7.1 Scenario*

First the acquirer evaluates `Report` resource RP1 and finds possible risks.

1. The acquirer sets the URL of RP1 to PROMCODE client
2. The PROMCODE client submits an HTTP GET request to the PROMCODE server to retrieve RP1 and linked resources including `Project` resource PJ1, and `Plan` resource PN1.
3. The PROMCODE client parses RP1 and obtains the list of collected resources, i.e., `ScopeItem` resources, `WorkItem` resources and `Artifact` resources, `Measurement` resources included in RP1 and `Measure` resources linked from those resources.
4. The acquirer then reviews the resources and identifies risks.

Next, the acquirer creates and registers a `RiskCollection` resource RC1 if it is not created yet.

5. The acquirer starts "Create a risk collection" operation on the PROMCODE client.
6. The PROMCODE client extracts the LDP container URL for the `RiskCollection` resource from PJ1.

  

7. The PROMCODE client creates the content of the new `RiskCollection` resource with title "RC1" in RDF, and submits an HTTP POST request with the content to the container.

EXAMPLE 43: Request - creating a RiskCollection resource

```
POST /pm/projects/100/riskCollectionContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:RiskCollection ;
  dcterms:title "RC1" ;
  oslc_promcode:belongsTo <http://example.com/pm/projects/100>
 .
```

8. The PROMCODE server returns the URL of `RiskCollection` RC1 to the PROMCODE client.

EXAMPLE 44: Response - creating a RiskCollection resource

```
HTTP/1.1 201 Created
Location:http://example.com/pm/riskcollections/800
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

9. The PROMCODE client displays the URL of `RiskCollection` RC1.

The acquirer notifies the supplier of the URL of `RiskCollection` resource RC1, and asks the supplier to register the risks.

10. The acquirer notifies the supplier of the URL of RC1.

The supplier obtains the risks and then creates `Risk` resources and registers them into `RiskCollection` resource RC1.

11. The supplier sets the URL of RC1 to the PROMCODE client.

12. The supplier executes "Create a risk" operation on the PROMCODE client.

  

13. The PROMCODE client creates a content of the new `Risk` resource with title "RS1" in RDF.

14. The PROMCODE client extracts the LDP container URL for the `Risk` resource from `Project` resource PJ1, and submits an HTTP POST request with RS1 to the container.

EXAMPLE 45: Request - creating a Risk resource

```
POST /pm/projects/100/riskContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:Risk ;
  dcterms:title "RS1" ;
  dcterms:description "Work load for UI for making a reservati
on may exceed estimation" ;
  oslc_promcode:raisedDate "2017-03-15T20:00:00Z" ;
  oslc_promcode:identifiedFor <http://example.com/pm/scopeitem
/1010> ;
  oslc_promcode:stateOfRisk <http://example.com/ns/status#open
> .
```

15. The PROMCODE server returns the URL of RS1 to the PROMCODE client.

EXAMPLE 46: Response - creating a Risk resource

```
HTTP/1.1 201 Created
Location:http://example.com/pm/risks/8000
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

16. The PROMCODE client submits an HTTP PUT request to the PROMCODE server to update `RiskCollection` resource RC1 to include `Risk` resource RS1.

EXAMPLE 47: Request - updating a RiskCollection resource

```
PUT http://example.com/pm/riskcollections/800 HTTP/1.1
```

  

```
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle
If-Match: W/'123456786'

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/riskcollections/800>
  a oslc_promcode:RiskCollection ;
  dcterms:identifier "800" ;
  dcterms:title "RC1" ;
  oslc_promcode:belongsTo <http://example.com/pm/projects/100>
 ;
  oslc_promcode:collects <http://example.com/pm/risks/8000> .
```

EXAMPLE 48: Response - updating a RiskCollection resource

```
HTTP/1.1 204 No Content
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
ETag: W/'123456787'
```

17. The supplier repeats the steps from 11 to 16 for each risk identified. When all `Risk` resources are created and registered to RC1, the supplier notifies the acquirer that RC1 is updated.

The acquirer reviews `Risk` resources in the `RiskCollection` resource RC1 and creates a new `Issue` resource if necessary.

18. The acquirer sets the URL of RC1 to the PROMCODE client.
19. The PROMCODE client submits an HTTP GET request to the PROMCODE server, and retrieves RC1.
20. The PROMCODE server returns RC1 to the PROMCODE client.
21. The acquirer selects RS1 on the PROMCODE client.
22. The PROMCODE client submits an HTTP GET request to the PROMCODE server, and retrieves RS1.
23. The PROMCODE server returns RS1 to the PROMCODE client.
24. The acquire reviews RS1 and creates a risk mitigation plan.

  

25. One of the actions of the plan can be to raise an issue. In that case, the acquirer can create a new `Issue` resource by following the steps of Issue Management scenario.

26. If the risk is resolved and no longer needed, the acquirer closes `Risk` RS1 by executing "Close a risk" operation on the PROMCODE client.

27. In that case, the PROMCODE client submits an HTTP PUT request to the PROMCODE server to update the `stateOfRisk` property of RS1.

EXAMPLE 49: The content of Risk resource RS1

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/risks/8000>
  a oslc_promcode:Risk ;
  dcterms:identifier "8000" ;
  dcterms:title "RS1" ;
  dcterms:description "Work load for UI for making a reservati
on may exceed estimation" ;
  oslc_promcode:raisedDate "2017-03-15T20:00:00Z" ;
  oslc_promcode:identifiedFor <http://example.com/pm/scopeitem
/1010> ;
  oslc_promcode:stateOfRisk <http://example.com/ns/status#clos
ed> .
```

28. The PROMCODE server returns the closed RS1 to the PROMCODE client.

29. Repeat steps from 21 to 28 for each `Risk` resource RSi.

#### *4.3.7.2 Summary Sequence Diagram*

The following sequence diagram summarizes Risk Management scenario:

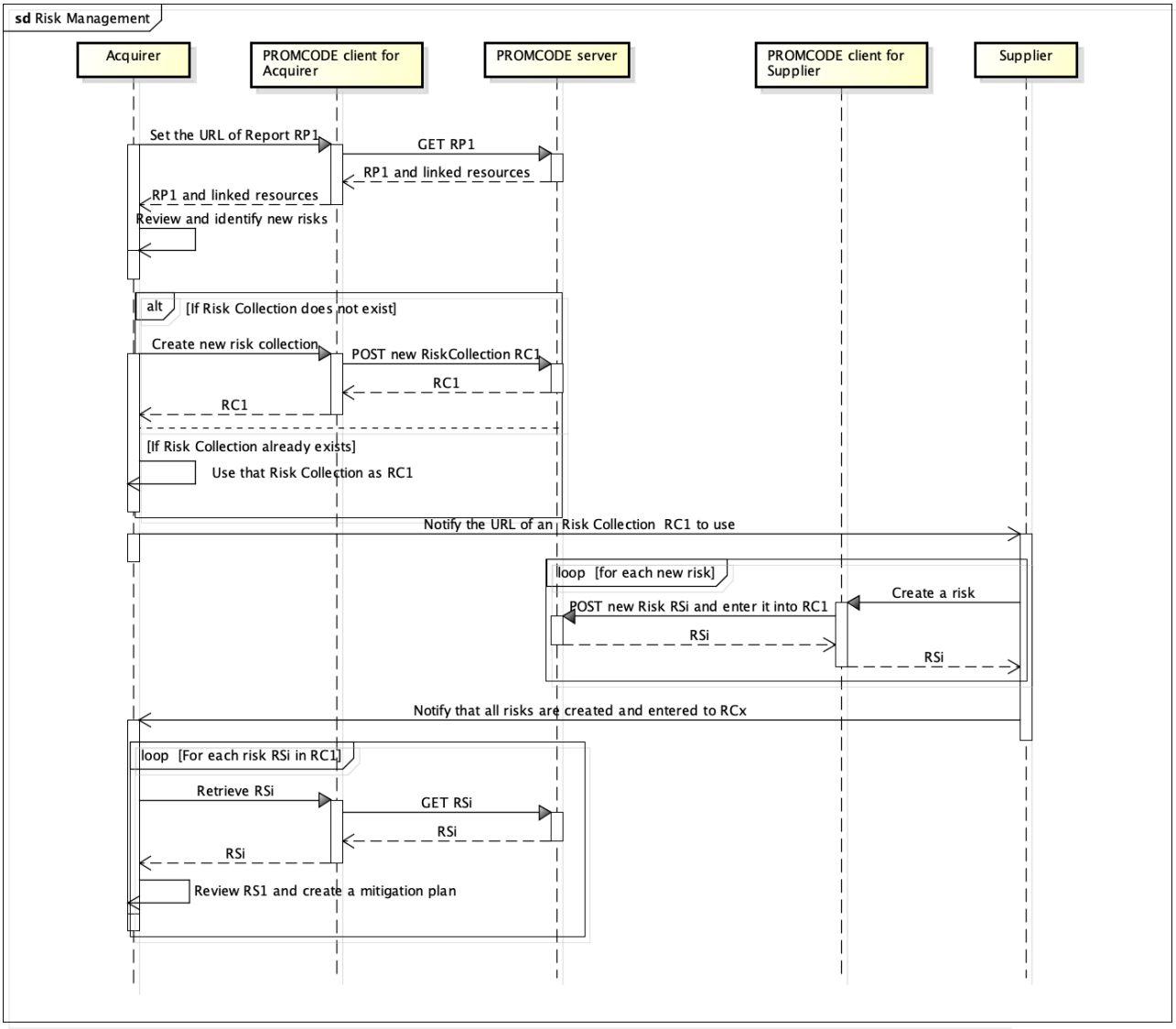


Fig. 8 Risk Management

## 4.3.8 Issue Management

This scenario illustrates how the PROMCODE services can be used in the process of issue management. The corresponding use case scenario is described in 2.4.7 Issue Management

### *4.3.8.1 Scenario*

First, the acquirer registers an `IssueCollection` resource.

1. The acquirer executes "Create an issue collection" operation on the PROMCODE client if the `IssueCollection` resource has not been created yet.

2. The PROMCODE client creates a content of the new `IssueCollection` resource

with title "IC1" in RDF.

3. The PROMCODE client extracts the LDP container URL for the `IssueCollection` resource from `Project` resource PJ1, and submits an HTTP POST request with the content created in step 2 to the container.

EXAMPLE 50: Request - creating an IssueCollection resource

```
POST /pm/projects/100/issueCollectionContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:IssueCollection ;
  dcterms:title "IC1" ;
  oslc_promcode:belongsTo <http://example.com/pm/projects/100>
 .
```

4. The PROMCODE server returns the URL of `IssueCollection` resource IC1 to the PROMCODE client.

EXAMPLE 51: Response - creating an IssueCollection resource

```
HTTP/1.1 201 Created
Location:http://example.com/pm/issuecollections/710
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

5. The PROMCODE client displays the URL of IC1.

6. The acquirer notifies the supplier of the URL of IC1.

If the supplier already identified issues during the project execution, then registers `Issue` resources in `IssueCollection` resource IC1.

7. The supplier sets the URL of IC1 to the PROMCODE client.

8. The supplier executes "Create an issue" operation for each identified issue on the PROMCODE client.

  

9. The PROMCODE client creates a content of the new `Issue` resource with title "IS1" in RDF.

10. The PROMCODE client extracts the LDP container URL for the `Issue` resource from `Project` resource PJ1, and submits an HTTP POST request to the container with the content created in step 9.

EXAMPLE 52: Request - creating an Issue resource

```
POST /pm/projects/100/issueContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:Issue ;
  dcterms:title "IS1" ;
  dcterms:description "Estimation for UI for making a reservat
ion needs to be updated" ;
  oslc_promcode:raisedDate "2017-03-16T17:00:00Z" ;
  oslc_promcode:raisedBy <http://example.com/pm/scopeitem/1010
> ;
  oslc_promcode:stateOfIssue <http://example.com/ns/status#ope
n> .
```

11. The PROMCODE server returns the URL of `Issue` resource IS1 to the PROMCODE client.

EXAMPLE 53: Response - creating an Issue resource

```
HTTP/1.1 201 Created
Location:http://example.com/pm/issues/7010
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

12. The PROMCODE client submits an HTTP PUT request to the PROMCODE server to update `IssueCollection` resource IC1 to include the `Issue` resource IS1.

EXAMPLE 54: Request - updating an IssueCollection resource

  

```
PUT http://example.com/pm/issuecollections/710 HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle
If-Match: W/'123456788'

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/issuecollections/710>
  a oslc_promcode:IssueCollection ;
  dcterms:identifier "710" ;
  dcterms:title "IC1" ;
  oslc_promcode:belongsTo <http://example.com/pm/projects/100>
 ;
  oslc_promcode:collects <http://example.com/pm/issues/7010> .
```

EXAMPLE 55: Response - updating an IssueCollection resource

```
HTTP/1.1 204 No Content
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
ETag: W/'123456789'
```

13. The supplier repeats steps from 8 to 12 for each newly identified issue to create `Issue` resource and register it into IC1. When all new issues are handled, the supplier notifies the acquirer that IC1 is updated.

The acquirer reviews `Issue` resources in the `IssueCollection` resource IC1. If it determines that a plan change is needed, the plan change scenario is triggered.

14. The acquirer sets the URL of IC1 to get the resource to the PROMCODE client.
15. The PROMCODE client submits an HTTP GET request to the PROMCODE server, and retrieves IC1.
16. The PROMCODE server returns IC1 to the PROMCODE client.
17. The PROMCODE client displays IC1.
18. The acquirer selects `Issue` resource IS1 on the PROMCODE client.
19. The PROMCODE client submits an HTTP GET request to the PROMCODE server, and retrieves IS1.

  

20. The PROMCODE server returns IS1 to the PROMCODE client.
21. The acquire reviews IS1.
22. The acquire creates an issue management plan to address IS1. In some cases, it may result in the plan change, triggering the execution of the Plan Change scenario.
23. If IS1 is resolved and is no longer necessary to keep it in the issue management process, the acquirer can execute "Close an issue" operation on the PROMCODE client.
24. In that case, the PROMCODE client changes the `stateOfIssue` property of IS1 to "Closed", and submits an HTTP PUT request to the PROMCODE server.

EXAMPLE 56: The content of Issue resource IS1

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/issues/7010>
  a oslc_promcode:Issue ;
  dcterms:identifier "7010" ;
  dcterms:title "IS1" ;
  dcterms:description "Estimation for UI for making a reservat
ion needs to be updated" ;
  oslc_promcode:raisedDate "2017-03-16T17:00:00Z" ;
  oslc_promcode:raisedBy <http://example.com/pm/scopeitem/1010
> ;
  oslc_promcode:stateOfIssue <http://example.com/ns/status#clo
sed> .
```

25. The PROMCODE server returns the closed `Issue` resource IS1 to the PROMCODE client.
26. The acquirer repeats the steps from 18 to 25 for each `Issue` resource ISi.

#### *4.3.8.2 Summary Sequence Diagram*

The following sequence diagram summarizes Issue Management scenario:

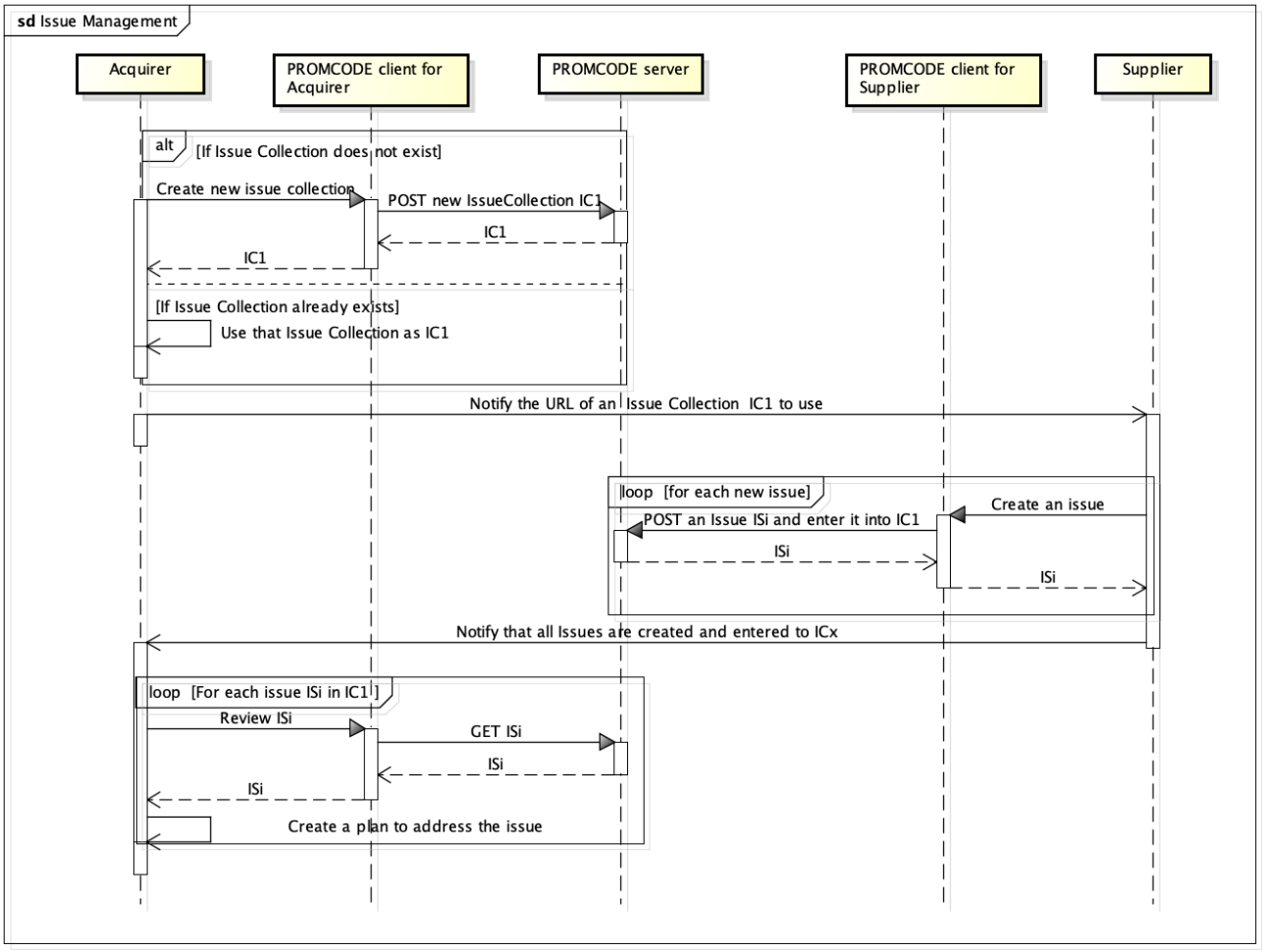


Fig. 9 Issue Management

### 4.3.9 Plan Change

This scenario illustrates how the PROMCODE services can be used in the process of the plan change. The corresponding use case scenario is described in 2.4.8 Plan Change.

The PROMCODE specification does not specify how changes on resources are implemented. However, implementation on how changes on resources are made is needed to understand the details of the sequence of steps. Therefore, the following rules are used in the appendix.

1. When there is a change on the content of a resource for any resource except a `Plan` resource and a `Measure` resource, the content of the original resource is replaced with a new content. When this is done, the description property of the resource is updated to include the information on the change such as what property is changed, the old value and the new value with the change, and the date and time of the change.

2. When there is a need to do plan change, a new Plan resource is created with its description property containing the information on the details of the plan change from a

  

current Plan resource to the new Plan resource. Therefore, the content of a current Plan resource is never modified. A change of a plan may involve a change on the content of resources that are linked from the current `Plan` resource. In that case, a new `Plan` resource should include the information on the change in its description property. The reason for treating `Plan` resources uniquely is because it is easier to keep all the plans managed this way with capability of recovering the content of an old plan. For `Measure` resources, the Appendix decided to treat them similarly to Decimal resources.

Alternatively, changes of resources can be modeled more systematically using [OSLC-ConfigM]. In the Appendix, however, a simple and tactical implementation is used as described above to avoid adding complexity of the systematic framework to the description of the scenario .

#### *4.3.9.1 Scenario*

First, the acquirer obtains `Plan` resource PN1 and collected resources, i.e., `ScopeItem`, `WorkItem` and `Artifact` resources.

1. The acquirer inputs the URL of PN1 to PROMCODE client.
2. The PROMCODE client submits an HTTP GET request to the PROMCODE server and retrieves PN1 and associated `Project` resource PJ1.
3. The PROMCODE client parses PN1 and gets the list of collected resources, i.e., `ScopeItem` resources, `WorkItem` resources and `Artifact` resources.
4. The PROMCODE client then submits an HTTP GET request to the PROMCODE server and retrieve the `ScopeItem` resources, the `WorkItem` resources, and the `Artifact` resources.

Next, the acquirer updates the `ScopeItem` resources, `WorkItem` resources, `Artifact` resources and `Measure` resources for the change of plan. In some cases, it may need to change some properties of the project PJ1 such as `plannedEndDate`. In the example to follow, a simple case is considered where the change is only on `target` property of an `Artifact` resource.

5. The acquirer updates the resources on PROMCODE client.

Next, the acquirer publishes the `Plan` resource to the PROMCODE server.

6. The acquirer executes "Update a plan" operation on the PROMCODE client.
7. The PROMCODE client submits an HTTP POST request to the PROMCODE server to create a `Measure` resource with new target value of code size.

> EXAMPLE 57: The content of the Measure resource

  

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:Measure ;
  dcterms:title "ME10" ;
  dcterms:description "Code size (kloc)" ;
  oslc_promcode:valueOfMeasure "15" ;
  oslc_promcode:metricOfMeasure <http://example.com/ns/metric#
Sloc> ;
  oslc_promcode:unitOfMeasure <http://example.com/ns/unit#Loc>
 .
```

EXAMPLE 58: Response - creating a Measure resource

```
HTTP/1.1 201 Created
Location: http://example.com/pm/measures/6013
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

8. The PROMCODE client submits an HTTP PUT request to the PROMCODE server to update an `Artifact` resource with new target value of code size.

EXAMPLE 59: The content of the Artifact resource

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<http://example.com/pm/artifacts/2010>
  a oslc_promcode:Artifact ;
  dcterms:title "AR1" ;
  dcterms:description "Source Code for UI for making a reserva
tion. Updated one of targets from http://example.com/pm/measur
es/6010
to http://example.com/pm/measures/6013 on XX/XX/XX" ;
  oslc_promcode:producedFor <http://example.com/pm/scopeitem/1
010> ;
  oslc_promcode:targets <http://example.com/pm/measures/6013>,
 <http://example.com/pm/measures/6011> .
```

EXAMPLE 60: Response - updating an Artifact resource

```
HTTP/1.1 204 No Content
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
ETag: W/'123456791'
```

9. The PROMCODE client creates a content of the new `Plan` resource with title "PN2" in RDF.

10. The PROMCODE client extracts the LDP container URL for the `Plan` resource from `Project` resource PJ1, and submits an HTTP POST request with the content created in step 9 to the container.

EXAMPLE 61: Request - creating an updated Plan resource

```
POST /pm/projects/100/planContainer HTTP/1.1
Host: example.com
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_promcode: <http://open-services.net/ns/promcode#>
 .

<>
  a oslc_promcode:Plan ;
  dcterms:title "PN2" ;
  dcterms:description "Plan PN2 is a revised plan of http://ex
ample/pm/plans/200 with the change of one of targets of Artifa
ct http://example.com/pm/artifacts/2010
 from http://example.com/pm/measures/6010 to http://example.co
m/pm/measures/6013 on XX/XX/XX" ;
  oslc_promcode:belongsTo <http://example.com/pm/projects/100>
 ;
  oslc_promcode:collects <http://example.com/pm/scopeitems/101
0>, <http://example.com/pm/scopeitems/1011>,
                <http://example.com/pm/artifacts/2010>, <http:
//example.com/pm/artifacts/2011>,
                <http://example.com/pm/workitems/3010>, <http:
//example.com/pm/workitems/3011>,
                <http://example.com/pm/workitems/3012>, <http:
//example.com/pm/workitems/3013> .
```

11. The PROMCODE server returns the URL of `Plan` resource PN2.

EXAMPLE 62: Response - creating an updated Plan resource

```
HTTP/1.1 201 Created
Location: http://example.com/pm/plans/201
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

12. The PROMCODE client displays the URL of `Plan` resource PN2 so that the acquirer can get the URL.
13. Now a new `Plan` resource PN2 is created and is ready for review by the supplier. The acquirer notifies the supplier of the URL of PN2.

Upon the notification, the supplier starts reviewing the change of the plan.

14. The supplier sets the URL of PN2 to the PROMCODE client.
15. The PROMCODE client submits an HTTP GET request to the PROMCODE server and retrieves PN2.
16. The PROMCODE client parses PN2 and gets the list of collected resources, i.e., `ScopeItem` resources, `WorkItem` resources and `Artifact` resources.
17. The PROMCODE client then submits an HTTP GET request to the PROMCODE server and retrieves the `ScopeItem` resources, the `WorkItem` resources and the `Artifact` resources.
18. The supplier reviews the resources.
19. The supplier notifies the acquirer of the result of the review.

If necessary, the acquirer further updates the plan by going through the scenario again.

#### *4.3.9.2 Summary Sequence Diagram*

The following sequence diagram summarizes Plan Change scenario:

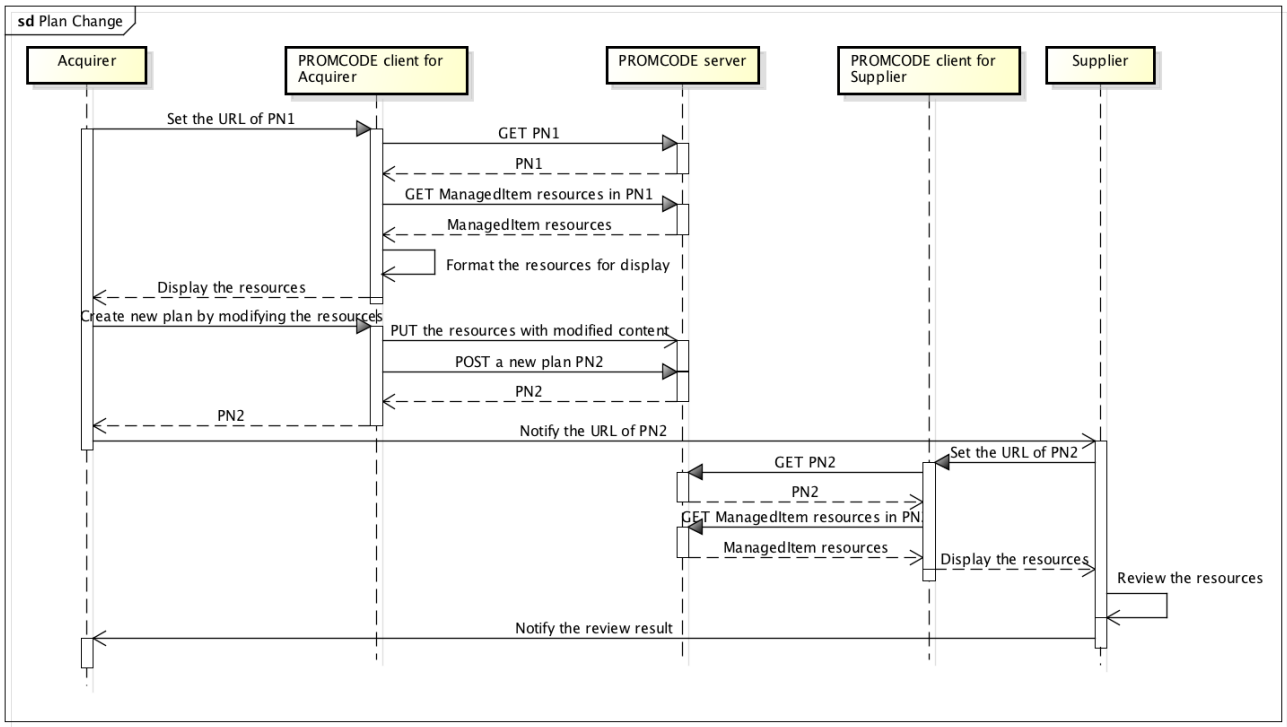


Fig. 10 Plan Change

### 4.3.10 Project Closing

This scenario illustrates how the PROMCODE 1services can be used in the process of project closing. The corresponding use case is described in 2.5 Project Closing.

#### *4.3.10.1 Scenario*

1. The supplier sets URL of `Project` resource PJ1 to the PROMCODE client.
2. The PROMCODE client submits an HTTP GET request to the PROMCODE server and retrieves PJ1.
3. The supplier executes "Update a project" operation and sets the actual end date and the actual size of the project with the PROMCODE client.
4. The PROMCODE client submits an HTTP PUT request to the PROMCODE server.
5. The supplier notifies the acquirer of the URL of PJ1.
6. The acquirer sets the URL of PJ1 to PROMCODE client.
7. The PROMCODE client submits an HTTP GET request to the PROMCODE server and retrieves PJ1.
8. The acquirer confirms the actual end date and the actual size.

  

# Appendix A. Acknowledgments

The PROMCODE TC would like to thank the members of the PROMCODE Consortium for their constructive discussions during the development of the specification. Thanks also go to Arthur Ryman and Kazuhiko Funakoshi who helped in creating technical content at an early stage of the project.

  

# Appendix B. References

## B.1 Informative references


[ISO21500]
*ISO 21500:2012, Guidance on Project Management*. URL: https://www.iso.org/standard/50003.html

[OSLC-ConfigM]
Nick Crossly. *OSLC Configuration Management 1.0*. URL: https://raw.githack.com/oslc-op/oslc-specs/master/specs/config/oslc-config-mgt.html

[OSLCCore3]
Jim Amsden; Martin Sarabura. *OSLC Core 3.0*. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/oslc-core.html

[OSLCCore3-Discovery]
Jim Amsden; Martin Sarabura. *OSLC Core Version 3.0. Part2: Discovery*. URL: https://docs.oasis-open-projects.org/oslc-op/core/v3.0/discovery.html

[OSLCPROMCODE]
Mikio Aoyama et al.. *OSLC PROMCODE Version 1.0*. URL: https://docs.oasis-open.org/oslc-promcode/promcode/v1.0/promcode-spec.html

[PMBOK5]
*A Guide to Project Management Body of Knowledge, 5th ed.*. URL: http://www.pmi.org/


  