OIIE Use Case Architecture

Requirements, Description, and Views

Contributors:
Matt Selway, University of South Australia
Sandra Fabiano, Yokogawa

June 29, 2020

This document describes the OIIE Use Case Architecture, its requirements, and the representations to be used for each of its constituents: Use Cases, Scenarios, Events, and User Stories.

Version 1.0-RC2

Contents

Glossary

OGI Pilot
Oil & Gas Interoperability Pilot

OIIE
Open Industrial Interoperability Ecosystem

Document Versioning

Version	Date	Major Changes
1.0	2019-02-05	Initial description of the 3+1-layer architecture, in contrast to the original 2-layer architecture, and the naming and identification scheme for the components.
- RC2	2020-06-29	Updated template

Overview

The Open Industrial Interoperability Ecosystem (OIIE) is built around a set of interoperability use cases that describe how industrial systems are to interact to achieve functionality desired/required by organizations involved in asset lifecycle management. The use cases allow organizations (EPCs, O/Os, Software Vendors, etc.) to declare conformance to specific reusable chunks of functionality, with which its systems can interoperate. The set of use cases is incrementally extended to incorporate new functionality as each set is validated by pilots, such as the OGI Pilot, with the inclusion of new use cases guided by industry partners.

Each use case conforms to the OIIE Use Case Architecture, which defines a standardized breakdown of Use Cases into smaller reusable parts, as well as a top-level overview of a Use Case or group of connected Use Cases. This breakdown forms a 3+1 level architecture, totaling 4 main components: Use Cases, Scenarios, Events, and User Stories. Each of the first two components decompose into the next, i.e., Use Cases decompose into Scenarios and Scenarios decompose into Events, while the fourth, User Stories, forms the "+1" as they can cross the other layers to illustrate specific events or whole use cases as required to achieve their purpose. The components of the Use Case Architecture and the relationships between them are summarized in Figure 1.

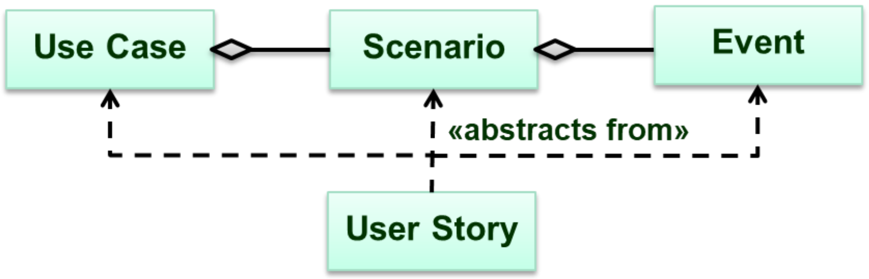

Figure 1 Relationships between Use Case Architecture Components

Use Cases describe common interactions and context to achieve an interoperability goal and are decomposed into Scenarios. Each Scenario provides additional details and requirements on how to achieve an interaction based on a specific group of Events. The Event descriptions detail specific message exchanges and their requirements but are general enough to support different realizations of the exchanges over different protocols and data formats. Finally, these three components are tied together by User Stories, which abstract from the underlying components to provide a higher-level overview of interactions and connect Use Cases in a logical flow. An overview of the Use Case Architecture is shown in Figure 2.

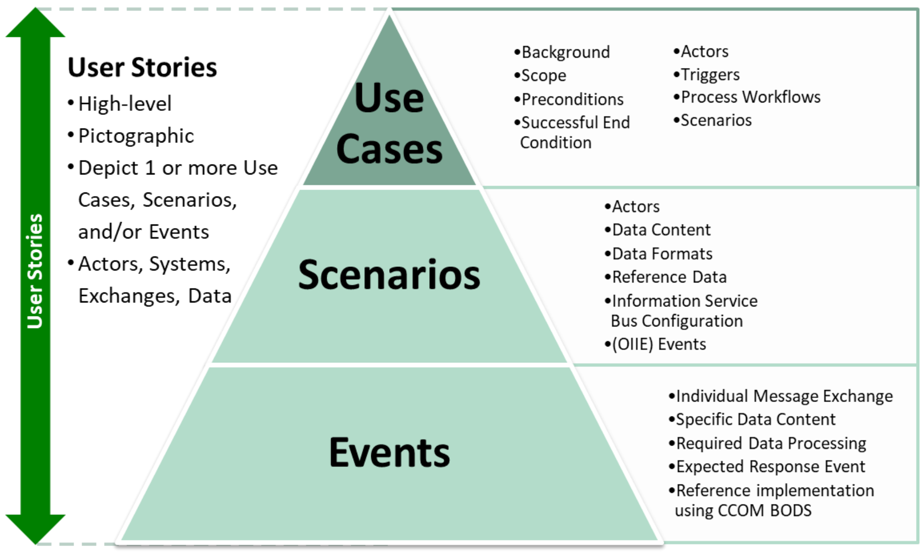

Figure 2 OIIE Use Case Architecture Overview

This document describes the Use Case Architecture itself and not any particular Use Cases, which are described separately (refer to the document: *02-List of Use Cases*). In the following, the requirements for the Use Case Architecture itself are described, followed by the differences between the current and previously used architecture of past OGI Pilot phases, and the specification of each architecture component is detailed.

Architectural Requirements

To support interoperability across the Oil & Gas industry (and process industries more generally), it is recognized that a consistent method for describing and specifying interoperability use cases is required. By describing use cases consistently, specific interoperability concerns can be addressed in a prioritized manner and participants know what to expect when taking part in different sets of interactions. To that end, the Use Case Architecture itself has been developed to fulfil certain requirements, including:

Reusability and Modularity
The components of the Use Case Architecture need to provide building blocks that can be reused and recombined in different contexts. This helps support bounded scopes for agreement, implementation, and conformance. In addition, the same building blocks can be reused to form novel use cases without requiring new implementation.

Low-level and High-level Views
The use cases must be described using means suitable for a wide range of audiences, including implementors, business managers, and those in-between. Therefore, it must provide low-level views that detail implementation concerns as well as high-level views that illustrate the overall use case(s) and how they fit together to meet business needs.

Industry/end-user participation
A long-term goal is to have industry support in the development of use cases to address common needs identified by industry partners. The architecture needs to be described such that an organization can propose a use case following the architecture that can then be validated by MIMOSA and incorporated into the set of OIIE Use Cases.

Interoperability Focused
The primary goal of OIIE Use Cases is interoperability between systems. Therefore, much of the description of a Use Case (and related components) needs to make system interactions explicit while situating them in context. However, it need *not* provide detailed descriptions of activities that may occur only within a system (or tightly integrated system-of-systems).

Updates to the Architecture

The OIIE Use Case Architecture (formerly OGI Use Case Architecture) has been used in past OGI Pilot phases. This document describes an updated version of this architecture over what was used previously. The following briefly lists the key changes to the previous architecture:

- From 2 Layer to 3+1 Layer Architecture. The original architecture only specified 2 layers: Use Cases and Scenarios.
- Addition of Events: previously Events were listed in Scenarios simply as the CCOM BODs that were required for the Scenario. The new version abstracts from CCOM BODs by introducing Events, which generalize specific message exchanges allowing different representations (not just CCOM BODs) and other forms of events. This change supports the reusability and modularity requirement.
- Addition of User Stories: User Stories were previously used informally to help illustrate the flow of Use Cases and Scenarios. By making the User Stories an explicit part of the architecture, it helps fulfil the requirement to make the use cases understandable to a wide range of audiences. Also, by formalizing the

notation (through rules and conventions) used for User Stories, it supports industry participation in the creation of consistent fully-described use cases.

Architecture Components

The Use Case Architecture identifies four components for describing use cases in a decomposable way: Use Cases, Scenarios, Events, and User Stories. This section describes each in turn and defines what each component must contain.

Note The current descriptions are a work in progress and will be progressively completed with more detail.

Note When referring to components of the Use Case Architecture, the terms "Use Case", "Scenario", and "Event" are capitalized. When using the same terms with a general meaning, lower case is used.

Use Cases

A Use Case provides a general description of interactions to achieve an interoperability goal within a specified scope and background context. The description includes the actors (systems or people) that are interacting, any preconditions and triggering events/conditions/use cases, the success case, a main success workflow (and possibly other workflows, e.g., exception flows, as required), and the Scenarios that are necessary to perform those workflows.

An example Use Case is "Asset Installation/Removal Updates" which describes the interactions between Operations (personnel) and Maintenance systems to perform a corrective maintenance task (removing an asset and installing a new asset) and the resulting publication of configuration updates (the asset removal/installation events) from the Maintenance Management Systems to Operations & Maintenance Systems. The Use Case also covers the situation where a Device Monitoring System senses the asset removal/installation and pushes the event to the Maintenance Management System to be reconciled against any current Work Orders. Only the publication of events is covered by OIIE Scenarios, as the Maintenance Personnel interactions are for illustration purposes and to provide context for the Scenarios. While there are four publication events, only two Scenarios are required due to reuse.

Scenarios

A Scenario provides a specific description of a group of events that achieves an interaction detailing data and configuration requirements; multiple scenarios may be required to achieve the goal of a use case and the same scenario may be reused by the same or in multiple use cases. Items included in the description of a Scenario are: the actors involved in the interaction (usually systems only; if a person is specified, it indicates a device that the person is using); the data content in general terms; required data format(s) such as the CCOM Business Object Document (BOD) format; the use of particular reference data libraries or items to ensure interoperability for the Scenario; any required configuration of the Information Service Bus (e.g., channel/topic configuration); any other infrastructure requirements (support systems that are required, etc.); and the Events required to achieve the Scenario.

For example, the Scenario "Publish Asset Removal/Installation events from MMS to O&M" describes the publication of asset removal or installation events from a Maintenance Management System to Operations & Maintenance systems. It specifies the need to exchange the functional location, asset, and time of the event (the data content requirements), the use of CCOM BODs (data format), the specific types of events (install and remove) from the MIMOSA reference data library, the use of the ISBM publish/subscribe mechanism and its configuration, and the specific Events used in the interaction. As Events may have multiple realizations, only one of which may be a CCOM BOD, the Scenario specifies that CCOM BODs are used explicitly, rather than only specifying the Events. If no restrictions were placed on the data format, any implementation of an Event can/should be supported.

Events

An Event describes an individual message exchange between systems, detailing data and processing requirements. This includes specific data content (in contrast to the general description of the Scenarios), any processing requirements placed on the recipient (e.g., if a flag is set to true, then behave in a certain way), and any expected response event such as a confirmation or a query result.

Events are still abstract in that they can be realized in multiple ways to support various mechanisms for exchange while adhering to the data and other requirements. Each Event is provided with a reference implementation, often described in CCOM BOD (XML) format. This allows events to be reused in different contexts and to support future exchange mechanisms. Moreover, remaining partially abstract allows Events to represent different types of events (note, lowercase 'e') where necessary.

An example Event is "Update Asset Configuration", which describes the data required to update the asset configuration information through one or more CCOM AssetSegmentEvent objects (which associate an asset to a functional location at a specific time). Its requirements state that there is no expected response event and confirmation is optional (as it is intended to be a published event), and that the receiving systems must update their asset configuration information with the new association (whatever it means for the receiving system). In contrast to the Scenario in which it is used, this Event does not specify any restriction on the type of AssetSegmentEvent that is to be updated, while the Scenario requires that it be either an "Install" or a "Remove". By not overly restricting the Event itself, the Event can be reused more easily in different contexts.

User Stories

A User Story provides a high-level graphical representation of interactions and events defined by one or more use cases and/or scenarios. They are designed to provide a business level overview of interactions and Use Cases across any level of the architecture (as necessary) using a simple graphical notation. The notation differentiates people, systems, and data/documents and connects them using arrows to illustrate interactions. A User Story consists of a number of frames, each frame illustrating a small portion of the Story and can be connected to preceding frames in various ways to illustrate continuity and/or use of data from a previous frame.

For example, a User Story may illustrate the various events and interactions (including person-to-person, system-to-system, person-to-system, business-to-business, etc.) involved in a series of related Use Cases, such as the triggering of a maintenance event based on condition data which leads to the removal of an asset and the installation of a new asset. In this way, User Stories can simply describe a logical sequence of related Use Cases, rather than following a trail of 'triggering events' defined in the Use Cases.

Naming/Identification Scheme for Use Case Components

The different components of the architecture (Use Cases, Scenarios, and Events, and User Stories) will be given consistent identifiers based on a naming scheme.

Considerations

There are several types of identification schemes that can be considered, for example: a simple numbered scheme (e.g., 1, 2, 3, …; a, b, c, …; etc.), using the "name" of the use case, (randomly) generated unique identifier, or "smart" identifiers that have parts representing different information categories.

The original Use Cases, for example, were specified using a simple numbering scheme in the order that they were defined. This can be problematic as the numbers are meaningless but imply an ordering. Coincidentally, this is the same order in which the Use Cases occurred. However, as new Use Cases are added, the occurrence order and Use Case number order diverge, possibly leading to confusion. Renumbering the Use Cases is not an appropriate solution, so the orderings will become inconsistent. The positive side of using a simple numbering system is that it is simple and provides a short and simple way of referencing another Use Case (also applies to Scenarios), such as UC1, UC13, etc.

Using the “name” of the use case as its identifier can make it clear and understandable. However, names can change over time if it is deemed that they do not quite represent the content—or someone just thinks of a “better” name. This will cause difficulties in tracking Use Cases (and other components), whereas as a consistent identifier will better support change management. For example, if a name change occurs, the identifier remains unchanged.

The third option is to generate identifiers using some well-known identifier scheme. As the Use Case Architecture is designed for human consumption, generating meaningless opaque identifiers is not appropriate. Moreover, such identification schemes often lead to large identifies, whereas it is convenient in many cases to use an abbreviated form of the identifiers, as long as it can be done without compromising the meaning, e.g., UC1 (for Use Case 1).

The final alternative is to use some form of “smart” identifier based on the different components of the identifier providing specific information, e.g., organization, category, system actor, etc. Such an approach can provide meaningful identifiers if the encoding is simple to understand. However, it can lead to long identifiers that cannot be conveniently abbreviated and, depending on the components of the identifier, it may need to change. For example, if the category were encoded in the identifier and the Use Cases were recategorized—which can easily occur as categorization schemes are often developed from a particular viewpoint at a particular time—the identifier would have to change or, if not, create an inconsistency.

Taking these considerations into account, the different alternatives may be more suited to different components of the Use Case Architecture. The following describe the naming/identifier schemes for each component.

Use Case Identifier Scheme

Use Cases will be identified using a simple numeric identifier. The numbers will be allocated in the order in which the Use Cases are defined and ***do not*** imply any ordering or dependencies between the Use Cases. Dependencies between Use Cases are explicitly listed as part of the Use Case content, and the flow of Use Cases can be obtained from the Use Stories.

Use Case can be identified using any of the following equivalent forms:

- Long form: Use Case 1, Use Case 2, …
- Abbreviated form: UC1, UC2, …
- Hyphenated abbreviation (for readability): UC-1, UC-2, …, UC-20

If the context is clearly referring to a Use Case, the number alone can be used to identify the Use Case.

Proposed or Pending Use Cases that have not yet been fully defined or accepted as OIIE Use Cases will ***not*** be assigned a numeric identifier. A numeric identifier will only be assigned once the Use Case has been defined to the extent that it includes all key details and has been accepted as a Use Case by an appropriate community.

Note The Use Cases defined before the update to the Use Case Architecture may not fulfil the criteria of including all key details; however, for consistency, their numeric identifiers will remain unchanged.

Scenario Identifier Scheme

Scenarios will be identified using a simple numeric identifier. The numbers will be allocated in the order in which the Scenarios are declared and ***do not*** imply any ordering or dependencies between Scenarios. Occurrences of Scenarios and, hence, their ordering, are determined by the process flows of the Use Cases.

In contrast to Use Cases, Scenarios may have their identifiers allocated ahead of time as placeholders before the full definition of the Scenario is complete. However, they must have the following basic details identified (although they may be updated later):

- The primary exchange method: Push, Pull or Publish

- The data being exchanged in general terms: for example, Asset Removal/Installation Events or As-Built Engineering data
- The primary source system(s)
- The primary target system(s)

These details are enough to give a general idea of the interoperability Scenario and allow Use Cases to meaningfully reference the Scenario before its complete requirements and secondary data exchanges have been fully defined.

Scenarios may be identified using any of the following equivalent forms:

- Long form: Scenario 1, Scenario 2, …
- Abbreviated form: S1, S2, …
- Hyphenated abbreviation (for readability): SC-1, SC-2

If the context is clearly referring to a Scenario, the number alone can be used to identify the Scenario.

Note Due to the reusability of Scenarios each Scenario identifier is unique and not specific to the Use Case(s) that make use of it. Therefore, there is no added discriminating value by combining the Scenario identifier with a Use Case identifier, as in, Use Case 5-Scenario 10 or UC5SC10 (or any other variation). Using such combined forms is only necessary if it important to identify both the Use Case and Scenario to which is being referred.

Event Identifier Scheme

Events will be identified by a name comprising the type of event and the type of data to be exchanged (based on the data requirements of the Event). In contrast to Use Cases and Scenarios, events are fine grained and reusable enough that their name can (and must) be unique. Moreover, due to the likely high-number of events and their reuse in different contexts, a name-based identifier is used to provide the reader with an indication of the purpose/content of the Event without needing to refer to an Event catalogue. The idea is to improve understanding in contexts in which Events are described, for example, when reading a Scenario description.

Events identifiers are constructed in the form: <*event type*> <*data type*>
where the *event type* is typically a verb describing what the event does or the action that will occur, while the *data type* is typically noun indicating the thing involved in or affected by the event or action.

Some example event identifiers are:

- Pull Segments
- Publish Asset Configuration Change
- Push Requests for Work

The following is a list of event types that are used in forming Event identifiers.

Event Type	Description
Push	A source system sends a message/data to a target system without the target system having previously requested the data. The source system must have a priori knowledge of the target, while the target may not know of the source system before receiving the message/data. There is typically an expectation of some form of processing to be performed by the target system and a confirmation or acknowledgement to be returned to the source system.

Event Type	**Description**
Pull	A target system queries a source system for a set of data to which the source system will respond. As the instigator, the target system must have a priori knowledge of the source, while the source system may not know of the target before receiving the query. A pull event typically implies the returned data will be stored/tracked/managed in the target system for an ***extended period of time***.
Get	A target system queries a source system for a set of data to which the source system will respond. As the instigator, the target system must have a priori knowledge of the source, while the source system may not know of the target before receiving the query. In contrast to a Pull event, a Get event typically implies the returned data will only be stored ***temporarily***: generally, until some processing has been completed.
Publish	A source system sends a message/data to any number of known or unknown target systems. Target systems may or may not store or otherwise process the data and there is no expectation of a response to be sent to the source system.

This list may be extended in the future to cover additional event types as required.

User Story Identifier Scheme

User Stories, specifically each User Story frame, will be identified by numeric identifier comprising three. The sequence of identifiers indicates the order in which the frames occur. Each frame captures a snapshot of interactions occurring generally (there may be exceptions to handle branching sequences) in the order the frames are defined but are not required to immediately precede/follow neighboring frames. Further, elements of User Story frames can be connected to show continuity. Such connectors (circles) are identified by an uppercase character, e.g., ‘A’, ‘B’, in alphabetic order. A set of User Stories (or frames) must ensure that each occurrence of a connector identifier represents the same thing to ensure correct continuity across frames.

User Stories (frames) can be identified using the following equivalent forms:

- Long form: User Story M001, User Story M100, User Story M220, …
- Short form: Story M001, Story M100, Story M220, …
- 3-digit identifier: M001, M100, M220, …

The identifiers for User Stories are grouped into categories to indicate distinct conceptual groupings. The identifiers are split approximately every 100 to leave space for new and expanded User Stories. The following categories are currently defined:

ID Range	**Category Description**	**Examples**
000-099	OIIE Configuration	RDL/ISDD selection, configuration of org/business unit/site breakdown structure
100-199	Capital Projects	Basic/detailed engineering, Make/Model selection, procurement, construction/asset installation
200-299	Handover, completion, startup, and commissioning	Handover of As-Built engineering information
300-399	Operations & Maintenance	Condition-Based Maintenance, remove/replace assets
400-499	Miscellaneous	Model/Asset information remediation

This list may be extended in the future to cover additional categories as required.