**SANDIA REPORT**
SAND2015-8819
Unlimited Release
September, 2015

# IDC Use Case Realizations *Working Draft for IDC Discussions*

J. Mark Harris, Shack Burns, Ben Hamlet, Mark Montoya, Rudy Sandoval

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

| | |
|---|---|
| Telephone: | (865) 576-8401 |
| Facsimile: | (865) 576-5728 |
| E-Mail: | reports@osti.gov |
| Online ordering: | http://www.osti.gov/scitech |

| | |
|---|---|
| Telephone: | (800) 553-6847 |
| Facsimile: | (703) 605-6900 |
| E-Mail: | orders@ntis.gov |
| Online order: | http://www.ntis.gov/search |

SAND2015-8819
Unlimited Release
September, 2015

# IDC Use Case Realizations
# *Working Draft for IDC Discussions*

J. Mark Harris, Shack Burns, Ben Hamlet, Mark Montoya, Rudy Sandoval
Next Generation Monitoring Systems
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS0401

**Abstract**

This document contains 4 use case realizations generated from the model contained in Rational Software Architect.

This page intentionally left blank

## TABLE OF CONTENTS

Use Case Hierarchy ........................................................................6
UCR-02 System Detects Event..............................................................9
UCR-02.08 System Refines Event Location ..........................................29
UCR-03.03 Refines Event ....................................................................45
UCR-08.05 Views Event History ...........................................................79

# Use Case Hierarchy

The IDC Use Case Hierarchy is shown here. The use cases highlighted in yellow are the use case realizations that appear in this document.

| | |
|---|---|
| **1** | **System Acquires Data** |
| 1.1 | System Receives Station Data |
| 1.2 | System Receives Bulletin Data |
| 1.3 | System Automatically Distributes Data |
| 1.4 | System Acquires Meteorological Data |
| 1.5 | System Synchronizes Acquired Station Data |
| 1.6 | System Synchronizes Processing Results |
| **2** | **System Detects Event** |
| 2.1 | System Determines Waveform Data Quality |
| 2.2 | System Enhances Signals |
| 2.3 | System Detects Events using Waveform Correlation |
| 2.4 | System Detects Signals |
| 2.5 | System Measures Signal Features |
| 2.6 | System Builds Events using Signal Detections |
| 2.7 | System Resolves Event Conflicts |
| 2.8 | System Refines Event Location |
| 2.9 | System Refines Event Magnitude |
| 2.10 | System Evaluates Moment Tensor |
| 2.11 | System Finds Similar Events |
| 2.12 | System Predicts Signal Features |
| **3** | **Analyzes Events** |
| 3.1 | Selects Data for Analysis |
| 3.2 | Refines Event |
| 3.2.1 | Determines Waveform Data Quality |
| 3.2.2 | Enhances Signals |
| 3.2.3 | Detects Signals |
| 3.2.4 | Measures Signal Features |
| 3.2.5 | Refines Event Location |
| 3.2.6 | Refines Event Magnitude |
| 3.2.7 | Evaluates Moment Tensor |
| 3.2.8 | Compares Events |
| 3.3 | Scans Waveforms and Unassociated Detections |
| 3.4 | Builds Event |
| 3.5 | Marks Processing Stage Complete |
| **4** | **Reports Event of Interest** |
| **5** | **Provides Data to Customers** |
| 5.1 | Requests System Data |

| | |
|---|---|
| 5.2 | Approves Events for External Release |
| 5.3 | Views System Results |

## 6 Configures System

| | |
|---|---|
| 6.1 | Controls Data Acquisition |
| 6.2 | Configures Station Usage |
| 6.3 | Defines Processing Sequence |
| 6.4 | Configures Data Acquisition |
| 6.5 | Configures Processing Components |
| 6.6 | Configures System Messages |
| 6.7 | Views System Configuration History |
| 6.8 | Configures Analysis Interfaces |
| 6.9 | Configures System Permissions |

## 7 Monitors Performance

| | |
|---|---|
| 7.1 | Analyzes Mission Performance |
| 7.2 | Monitors System Performance |
| 7.3 | Monitors Station State-of-Health |
| 7.4 | System Monitors Mission Performance |
| 7.5 | Monitors Mission Processing |

## 8 Supports Operations

| | |
|---|---|
| 8.1 | Accesses the System |
| 8.2 | Controls the System |
| 8.3 | Exports Data |
| 8.4 | Imports Data |
| 8.5 | Views Event History |
| 8.6 | Maintains Operations Log |
| 8.7 | Provides Analyst Feedback |
| 8.8 | Views Analyst Feedback |
| 8.9 | Views Analyst Performance Metrics |
| 8.10 | Views Security Status |
| 8.11 | Views Messages |

## 9 Tests System

| | |
|---|---|
| 9.1 | Performs Software Component Testing |
| 9.2 | Creates Test Data Set |
| 9.3 | Replays Test Data Set |
| 9.4 | Replays Analyst Actions |

## 10 Maintains System

| | |
|---|---|
| 10.1 | Performs System Backups |
| 10.2 | Performs System Restores |
| 10.3 | Installs Software Update |
| 10.4 | System Monitors Security |

## 11 Performs Research

| | |
|---|---|
| 11.1 | Analyzes Special Events |

11.2 Develops New Algorithms and Models
11.3 Determines Optimal Processing Component Configuration
11.4 Performs Multiple Event Location

**12 Performs Training**

12.1 Configures Data for Training Subsystem
12.2 Trains Analysts

**13 Operates Standalone Subsystem**

13.1 Conducts Site Survey
13.2 Performs Standalone Analysis

**14 IDC Unique**

14.2 System Assesses Event Consistency
14.3 Assesses Event Consistency
14.4 System Screens Event
14.5 Controls Monitoring Stations

IDC Use Case Realization Report

# UCR-02 System Detects Event

## USE CASE DESCRIPTION

This use case describes how the System pipeline processes the raw seismic, hydroacoustic, and infrasound waveform data from a time interval to form event hypotheses. The System first checks the quality of arriving waveform data and creates data quality control masks for waveform sections containing data that is unsuitable for processing (see ‘System Determines Waveform Data Quality’ UC). The System then processes waveforms to enhance signal content while reducing noise (see ‘System Enhances Signals’ UC).

The System detects signals (see ‘System Detects Signals’ UC), measures features on the signal detections (see ‘System Measures Signal Features’ UC), and then uses the signal detections and feature measurements to build both single station and network event hypotheses (see ‘System Builds Events using Signal Detections’ UC). The System uses channel based waveform correlation techniques to form single-station or network event hypotheses (see ‘System Detects Events using Waveform Correlation’ UC). The System measures signal features for the event hypotheses on waveform channels across the network (see ‘System Measures Signal Features’ UC). The System predicts signal detections and their features for events (see 'System Predicts Signal Features' UC). The System uses similarity parameters to search for historic events similar to the new event hypothesis (see 'System Finds Similar Events' UC).

After forming event hypotheses, the System resolves conflicting event hypotheses (see ‘System Resolves Event Conflicts’ UC) and then refines each event hypothesis’ location (see ‘System Refines Event Location’ UC) and magnitude (see ‘System Refines Event Magnitude’ UC). The System evaluates the moment tensor for an event (see 'System Evaluates Moment Tensor' UC).

The System pipeline follows a sequence configured by the System Maintainer when pipeline processing raw waveform data to form event hypotheses (see ‘Configures Processing Sequence’ UC).

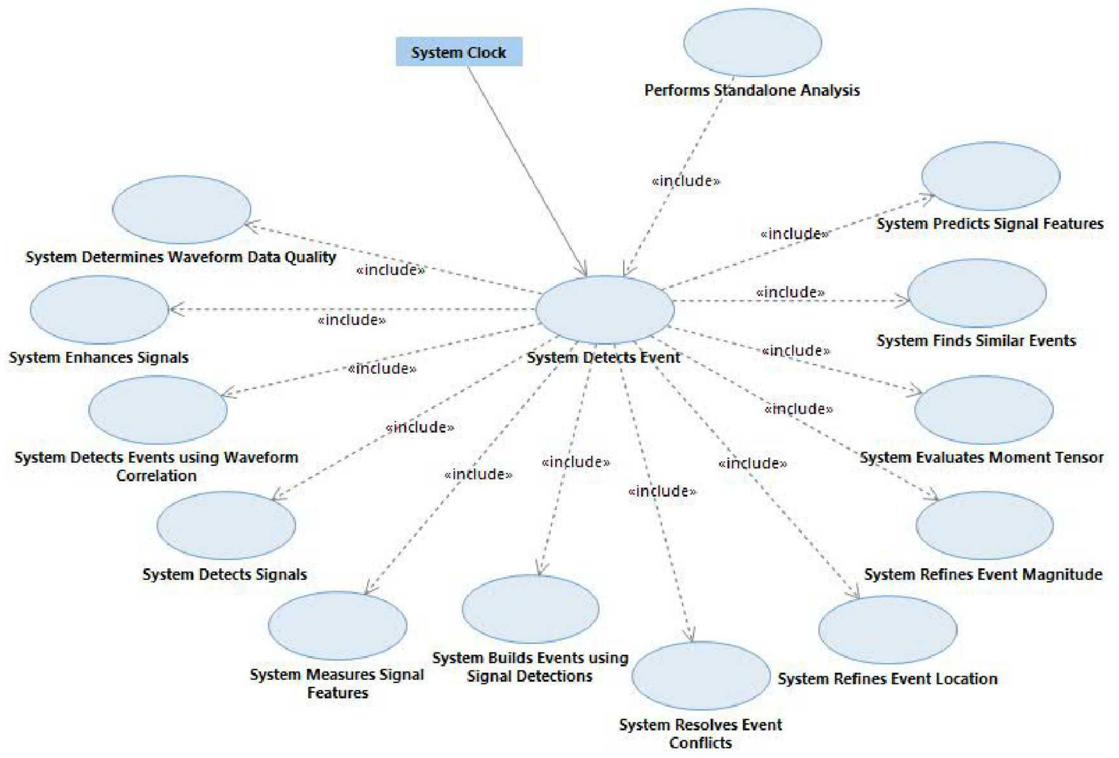
## ARCHITECTURE DESCRIPTION

The Processing Sequence Control mechanism is responsible for executing processing sequences previously defined by the System Maintainer (see “Defines Processing Sequence” UCR). Processing Sequence classes contain Processing Steps and Flows. These objects form a graph with Data References travelling on Flows between Processing Steps. Processing Sequence Control executes a Processing Sequence whenever a Triggering Condition initiating the Processing Sequence is satisfied. Several types of Processing Step exist in the System. A Processing Component Invocation Step invokes a control class to perform processing. Control classes invoked in this manner all realize a common interface named Processing Control IF which allows the Processing Sequence Control mechanism to invoke them. Control Flow Steps define loops around one or more Processing Steps, branches to select which of several

Processing Steps to execute, and Processing Steps that Processing Sequence Control can execute in parallel. A Processing Sequence Invocation Step invokes one Processing Sequence as part of another Processing Sequence.

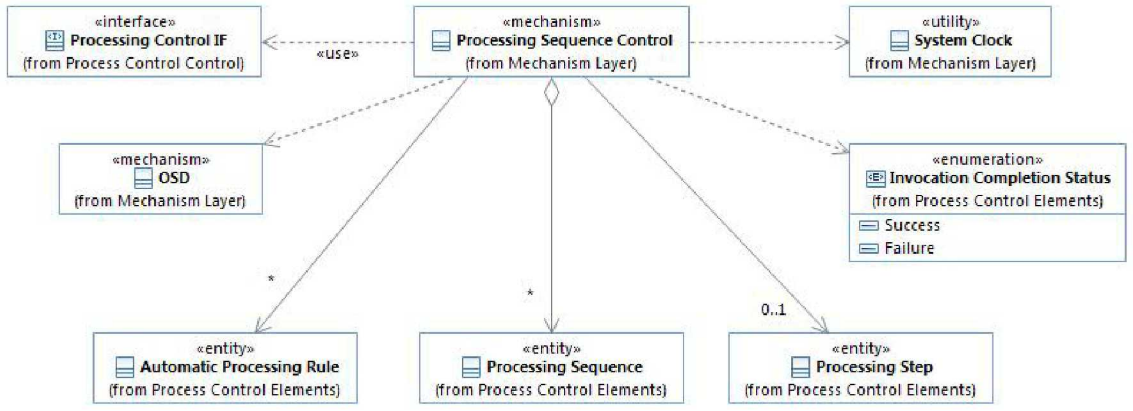
The Processing Control IF realizations may store data in the OSD, causing OSD callbacks to Processing Sequence Control. Processing Sequence Control places data references to the received data on Flows, making the data available to subsequence Processing Steps.

## USE CASE DIAGRAM


System Clock
Performs Standalone Analysis
«include»
System Predicts Signal Features
System Determines Waveform Data Quality
System Detects Event
System Finds Similar Events
System Enhances Signals
System Detects Events using Waveform Correlation
System Evaluates Moment Tensor
System Detects Signals
System Refines Event Magnitude
System Measures Signal Features
System Builds Events using Signal Detections
System Resolves Event Conflicts
System Refines Event Location

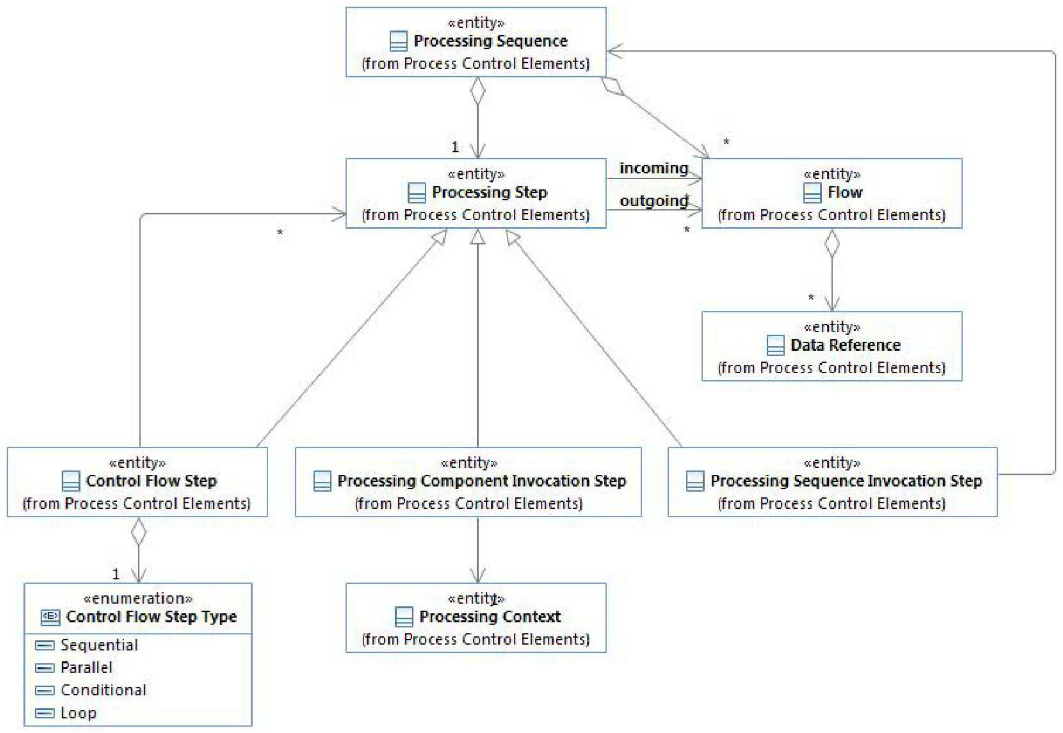
## CLASS DIAGRAMS

### Classes - Processing Sequence Control


«interface» Processing Control IF (from Process Control Control)
«use»
«mechanism» Processing Sequence Control (from Mechanism Layer)
«utility» System Clock (from Mechanism Layer)
«mechanism» OSD (from Mechanism Layer)
«enumeration» Invocation Completion Status (from Process Control Elements)
Success
Failure
*
*
0..1
«entity» Automatic Processing Rule (from Process Control Elements)
«entity» Processing Sequence (from Process Control Elements)
«entity» Processing Step (from Process Control Elements)

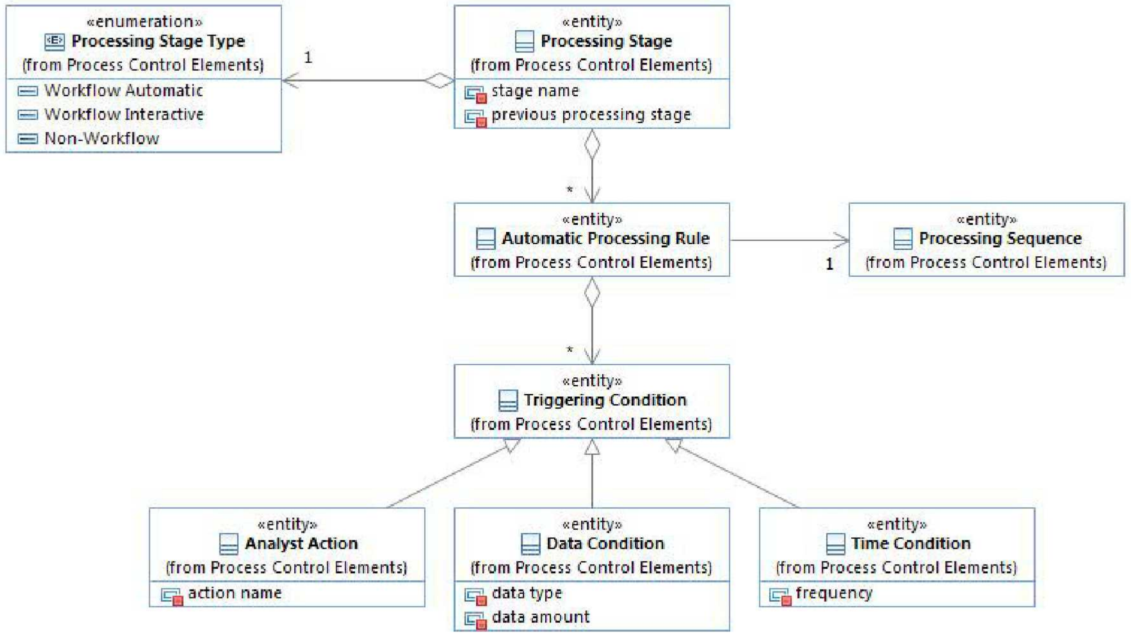
This diagram shows the Processing Sequence Control class and related classes. Processing Sequence Control periodically evaluates Automatic Processing Rules to initiate new Processing Sequences, executes the Processing Steps within a Processing Sequence, and monitors processing performed by realizations of the Processing Control IF interface.

## Classes - Processing Sequence


«entity»
Processing Sequence
(from Process Control Elements)
1
*
incoming
outgoing
*
*
«entity»
Processing Step
(from Process Control Elements)
«entity»
Flow
(from Process Control Elements)
*
«entity»
Data Reference
(from Process Control Elements)
«entity»
Control Flow Step
(from Process Control Elements)
«entity»
Processing Component Invocation Step
(from Process Control Elements)
«entity»
Processing Sequence Invocation Step
(from Process Control Elements)
1
«enumeration»
Control Flow Step Type
Sequential
Parallel
Conditional
Loop
1
«entity»
Processing Context
(from Process Control Elements)

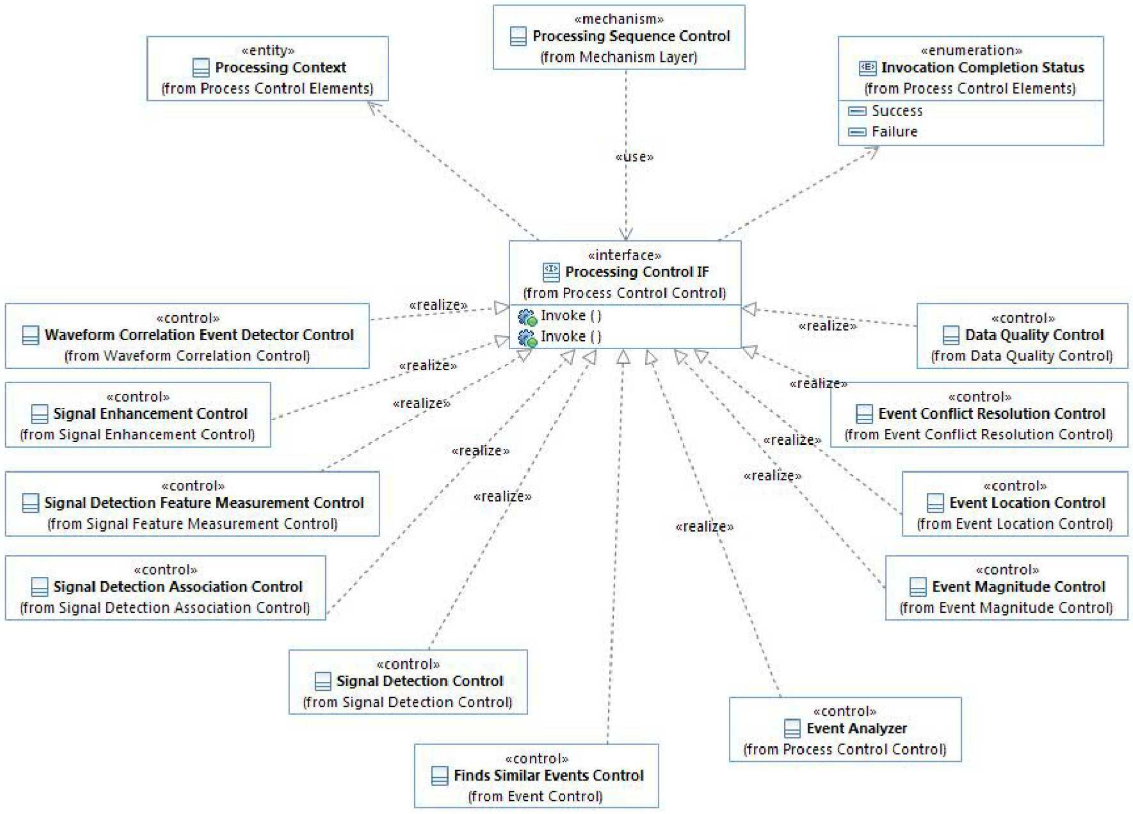
This diagram shows the structure of a Processing Sequence. A Processing Sequence consists of a graph of Processing Steps ordered based on their incoming and outgoing Flows. The System Maintainer configures Processing Sequences (see “Defines Processing Sequence” UCR) and the Processing Sequence Control mechanism is responsible for executing the steps of the graph in the proper order. Each step in the graph invokes a Processing Component, invokes another Processing Sequence or acts as a Control Flow Step. Control Flow Steps are the only steps that contain child steps. The child steps represent operands for the control flow. The number of children varies according the type of control flow. For example, a "Conditional" control flow step might always have two children: one for the "true" branch and one for the "false" branch. On the other hand, a "Sequential" or "Parallel" control flow step could have a variable number of children.

## Classes - Automatic Processing Rule


«enumeration»
Processing Stage Type
(from Process Control Elements)
Workflow Automatic
Workflow Interactive
Non-Workflow
1
«entity»
Processing Stage
(from Process Control Elements)
stage name
previous processing stage
*
«entity»
Automatic Processing Rule
(from Process Control Elements)
1
«entity»
Processing Sequence
(from Process Control Elements)
*
«entity»
Triggering Condition
(from Process Control Elements)
«entity»
Analyst Action
(from Process Control Elements)
action name
«entity»
Data Condition
(from Process Control Elements)
data type
data amount
«entity»
Time Condition
(from Process Control Elements)
frequency

This class shows the structure of an Automatic Processing Rule. An Automatic Processing Rule combines one or more Triggering Conditions with the Processing Sequence executed by the Processing Sequence Control when the System's state satisfies those conditions. Several types of Triggering Conditions exist in the System, including Analyst Action conditions satisfied when the Analyst takes some action, Data Conditions satisfied when certain types or amounts of data are acquired or created on the System, and Time Conditions satisfied when a certain amount of time has elapsed since the conditions was previously satisfied. The System Maintainer can configure Automatic Processing Rules for each Processing Stage (see "Defines Processing Sequence" UCR).

## Classes - Processing Control IF


«mechanism»
Processing Sequence Control
(from Mechanism Layer)
«entity»
Processing Context
(from Process Control Elements)
«enumeration»
Invocation Completion Status
(from Process Control Elements)
Success
Failure
«use»
«interface»
Processing Control IF
(from Process Control Control)
Invoke ()
Invoke ()
«realize»
«control»
Waveform Correlation Event Detector Control
(from Waveform Correlation Control)
«control»
Data Quality Control
(from Data Quality Control)
«control»
Signal Enhancement Control
(from Signal Enhancement Control)
«control»
Event Conflict Resolution Control
(from Event Conflict Resolution Control)
«control»
Signal Detection Feature Measurement Control
(from Signal Feature Measurement Control)
«control»
Event Location Control
(from Event Location Control)
«control»
Signal Detection Association Control
(from Signal Detection Association Control)
«control»
Event Magnitude Control
(from Event Magnitude Control)
«control»
Signal Detection Control
(from Signal Detection Control)
«control»
Event Analyzer
(from Process Control Control)
«control»
Finds Similar Events Control
(from Event Control)

This diagram shows the Processing Control IF interface, the classes realizing its interface, and its dependencies. Each instance of the Processing Component Invocation Step class (see "Classes - Processing Sequence Control") is configured to invoke one of the classes realizing the Processing Control IF through its Invoke() method. The UCRs included by System Detects Event define behaviors for these classes.

## CLASS DESCRIPTIONS

***<<control>> Event Analyzer***

Responsible for analyzing and updating events with information indicating whether further (e.g. iterative) automated processing should be performed. Instantiations of this class implement processing sequence control logic that is unavailable in the other classes realizing the Processing Control IF.

Event Analyzer implementations have the option to update the processing configuration parameters used by subsequent Processing Steps. This is similar to how human Analysts select processing parameters for algorithms they invoke, except that Event Analyzer does not directly invoke additional Processing Steps. Instead, Event Analyzer stores information in the OSD and then relies on callbacks to the Processing Sequence Control mechanism to initiate additional processing. Event Analyzer does this by creating a copy of a data object (e.g. a new version of an event hypothesis, etc.), updating the processing configuration parameters on that data object, and then storing the data object and its associated processing configuration parameters in the

OSD. OSD callbacks may result in Automatic Processing Rules being satisfied, causing the Processing Sequence Control mechanism to invoke additional Processing Sequences using the updated processing configuration parameters. The data objects stored in the OSD are also Event Analyzer's outputs, so the Processing Sequence Control Mechanism may set data references to these objects on the outgoing Flows for the Processing Step that invoked Event Analyzer.

***<<control>> Event Location Control***
Responsible for controlling the event location computation. Retrieves necessary data, invokes the appropriate Event Locator to compute the new location, and stores the result.

***<<control>> Signal Detection Association Control***
Control class responsible for controlling signal detection association calculations. Retrieves configuration from the OSD, invokes the appropriate Signal Detection Associator Plugin, computes quality metrics, and stores the new or modified events in the OSD.

***<<control>> Signal Detection Control***
Responsible for controlling automatic signal detection. Retrieves necessary data, invokes plugins to detect signals on waveform data and refine signal onset time, and stores the results.

***<<control>> Waveform Correlation Event Detector Control***
Responsible for controlling waveform correlation event detection computations. Retrieves necessary data, invokes the appropriate Waveform Correlation Event Detector Plugin to detect new events, and stores the results.

***<<entity>> Analyst Action***
Special type of Triggering Condition that is based on an action performed by an Analyst. The set of available actions is predefined by the system.

***<<entity>> Automatic Processing Rule***
Represents a Processing Sequence and the set of Triggering Conditions for initiating it.

***<<entity>> Control Flow Step***
A specialized kind of Processing Step that is used to represent control flow between other Processing Steps. The Control Flow Step is represented by the type of control flow operation (e.g. Parallel, Conditional, etc.) and operands to which it applies (i.e. other Processing Steps).

***<<entity>> Data Condition***
Special type of Triggering Condition that is based on the availability of data (e.g. 100 signal detections, 15% of all waveforms for the interval).

***<<entity>> Data Reference***
Represents a reference to data that passes between Processing Steps on a Flow.

***<<entity>> Flow***
Represents control and data flow between two Processing Steps.

***<<entity>> Processing Component Invocation Step***
A specialized kind of Processing Step that represents an invocation of a specific Processing Component.

***<<entity>> Processing Context***
Represents the context in which data is being stored and/or processed. This includes the processing session (e.g. processed by Analyst vs. processed by System). For Analyst processing, may identify the Analyst work session. For System processing, may identify the Processing Sequence and/or Processing Step being executed (including a way to identify a particular Processing Sequence and Processing Step among the many possible instantiations), the visibility for the results (private vs. global), and the lifespan of the data (transient vs. persistent). This information is needed by the Processing Sequence Control to manage the execution of Processing Sequences, which may execute in the context of an Analyst refining an event or in the context of the system initiating automatic processing. It is also needed by the Object Storage and Distribution (OSD) mechanism to determine how to store and distribute the data.

***<<entity>> Processing Sequence***
A user-configurable set of Processing Steps to be executed by the Processing Sequence Control mechanism. Each Processing Step may invoke a Processing Component or another Processing Sequence. Special steps are used to specify control flow (e.g. conditional logic, parallelism, etc.).

***<<entity>> Processing Sequence Invocation Step***
A specialized kind of Processing Step that represents an invocation of a specific Processing Sequence.

***<<entity>> Processing Stage***
Represents a named stage of data processing, which may be part of the System Maintainer-defined workflow or an Analyst-defined stage outside the workflow. All Processing Results are associated to a Processing Stage. The previous processing stage indicates the stage to be used as the default starting point when creating new processing results in the stage (e.g. when refining an event in the stage).

***<<entity>> Processing Step***
Represents a single step within a Processing Sequence. A Processing Step may invoke a Processing Component or invoke a Processing Sequence, and may optionally specify parameter overrides for the invoked component/sequence. Special kinds of Processing Steps known as Control Flow Steps are used to specify control flow between Processing Steps.

***<<entity>> Time Condition***
Special type of Triggering Condition that is based on time (e.g. every 5 minutes).

***<<entity>> Triggering Condition***
Represents a condition which must be satisfied in order to trigger a Processing Sequence.

***<<interface>> Processing Control IF***
Defines the interface implemented by all <<control>> classes in the system that are controlled by the Processing Sequence Control <<mechanism>>.

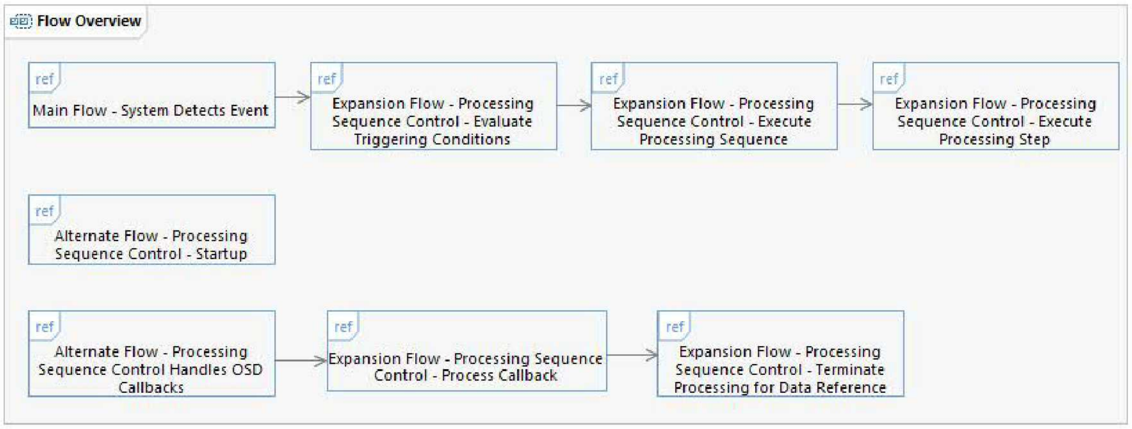
***<<mechanism>> OSD***
Represents the Object Storage and Distribution mechanism for storing and distributing data objects internally within the system.

***<<mechanism>> Processing Sequence Control***
Mechanism for executing and controlling processing sequences configured by the System Maintainer.

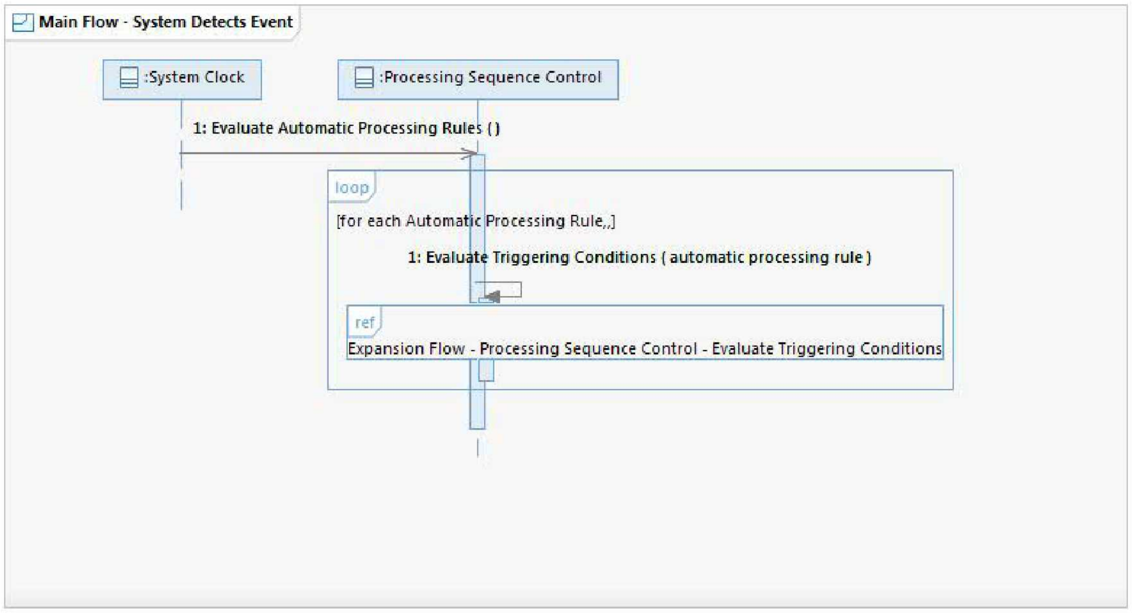
***<<utility>> System Clock***
Represents the mechanism to schedule, reschedule, and cancel callbacks.

## SEQUENCE DIAGRAMS

### Flow Overview


Flow Overview
ref
Main Flow - System Detects Event
ref
Expansion Flow - Processing Sequence Control - Evaluate Triggering Conditions
ref
Expansion Flow - Processing Sequence Control - Execute Processing Sequence
ref
Expansion Flow - Processing Sequence Control - Execute Processing Step
ref
Alternate Flow - Processing Sequence Control - Startup
ref
Alternate Flow - Processing Sequence Control Handles OSD Callbacks
ref
Expansion Flow - Processing Sequence Control - Process Callback
ref
Expansion Flow - Processing Sequence Control - Terminate Processing for Data Reference

### Main Flow - System Detects Event

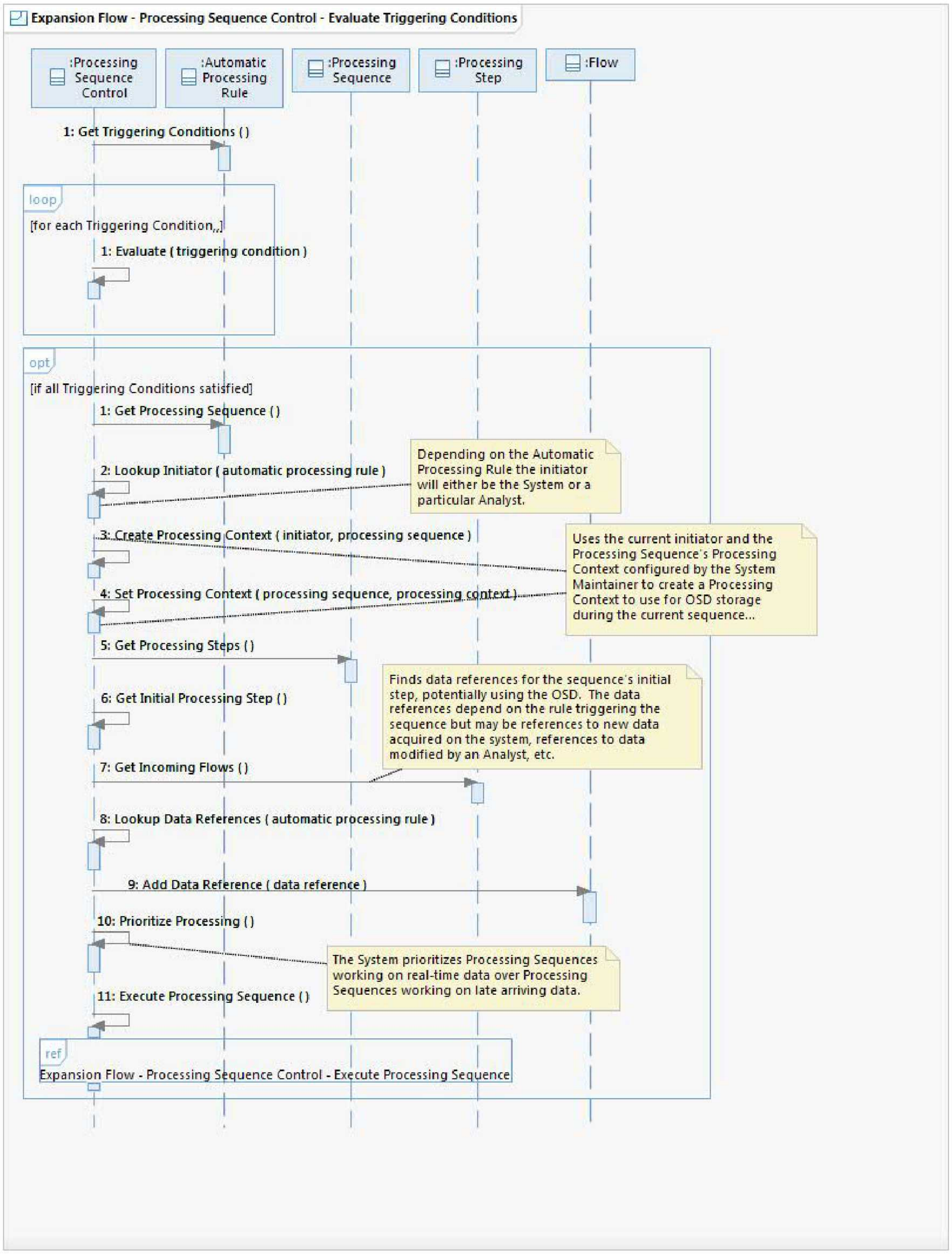

Main Flow - System Detects Event
:System Clock
:Processing Sequence Control
1: Evaluate Automatic Processing Rules ()
loop
[for each Automatic Processing Rule,,]
1: Evaluate Triggering Conditions ( automatic processing rule )
ref
Expansion Flow - Processing Sequence Control - Evaluate Triggering Conditions

This flow shows how Processing Sequence Control periodically evaluates the Automatic Processing Rules to trigger Processing Sequence execution.

*Operation Descriptions*

None

## Expansion Flow - Processing Sequence Control - Evaluate Triggering Conditions


Expansion Flow - Processing Sequence Control - Evaluate Triggering Conditions
:Processing Sequence Control
:Automatic Processing Rule
:Processing Sequence
:Processing Step
:Flow
1: Get Triggering Conditions ()
loop
[for each Triggering Condition,,]
1: Evaluate ( triggering condition )
opt
[if all Triggering Conditions satisfied]
1: Get Processing Sequence ()
2: Lookup Initiator ( automatic processing rule )
Depending on the Automatic Processing Rule the initiator will either be the System or a particular Analyst.
3: Create Processing Context ( initiator, processing sequence )
Uses the current initiator and the Processing Sequence's Processing Context configured by the System Maintainer to create a Processing Context to use for OSD storage during the current sequence...
4: Set Processing Context ( processing sequence, processing context )
5: Get Processing Steps ()
6: Get Initial Processing Step ()
Finds data references for the sequence's initial step, potentially using the OSD. The data references depend on the rule triggering the sequence but may be references to new data acquired on the system, references to data modified by an Analyst, etc.
7: Get Incoming Flows ()
8: Lookup Data References ( automatic processing rule )
9: Add Data Reference ( data reference )
10: Prioritize Processing ()
The System prioritizes Processing Sequences working on real-time data over Processing Sequences working on late arriving data.
11: Execute Processing Sequence ()
ref
Expansion Flow - Processing Sequence Control - Execute Processing Sequence

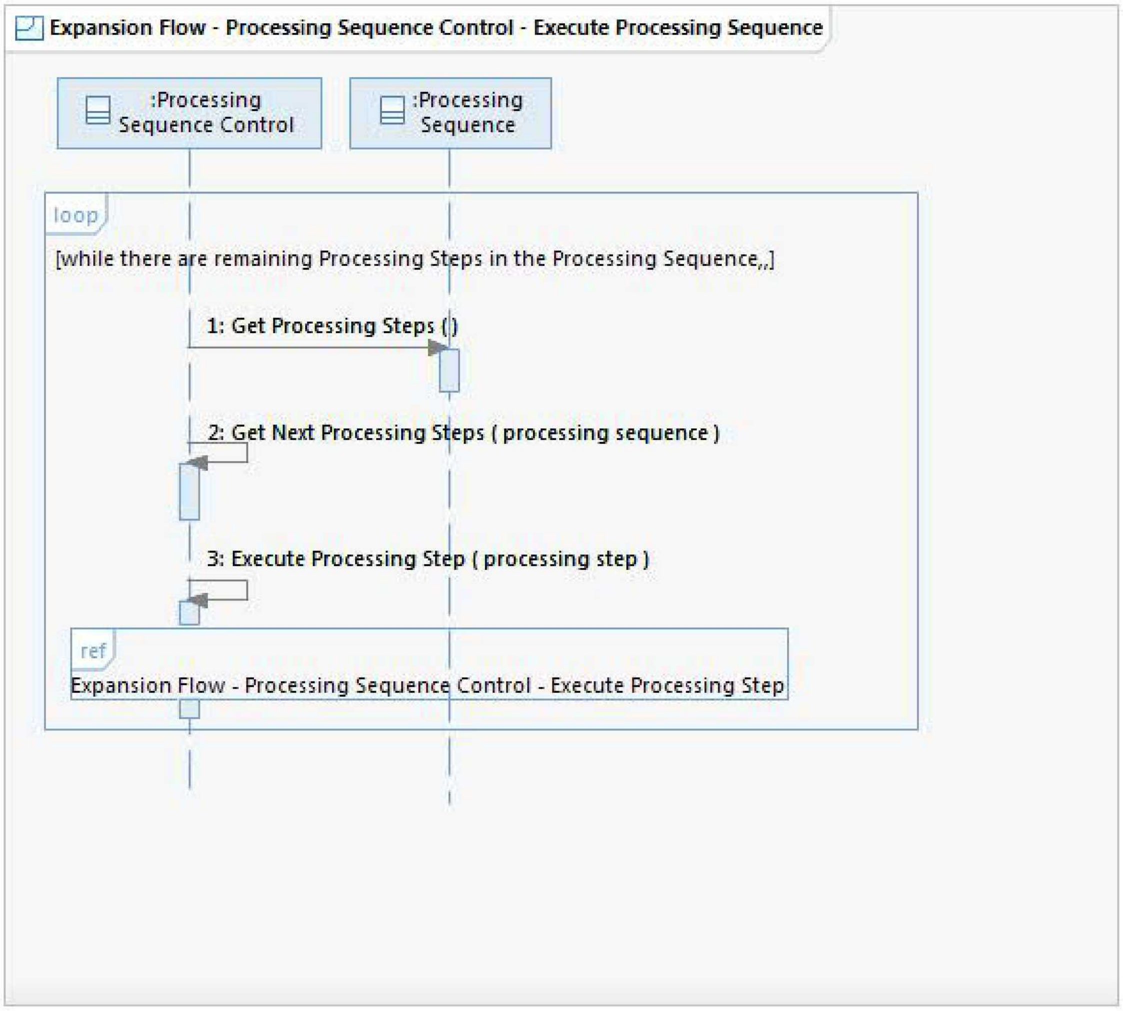
This flow shows Processing Sequence Control evaluating the Triggering Conditions for an Automatic Processing Rule. If all of the Triggering Conditions are satisfied then Processing

Sequence Control executes the Processing Sequence associated to the Automatic Processing Rule.

*Operation Descriptions*

None

## Expansion Flow - Processing Sequence Control - Execute Processing Sequence

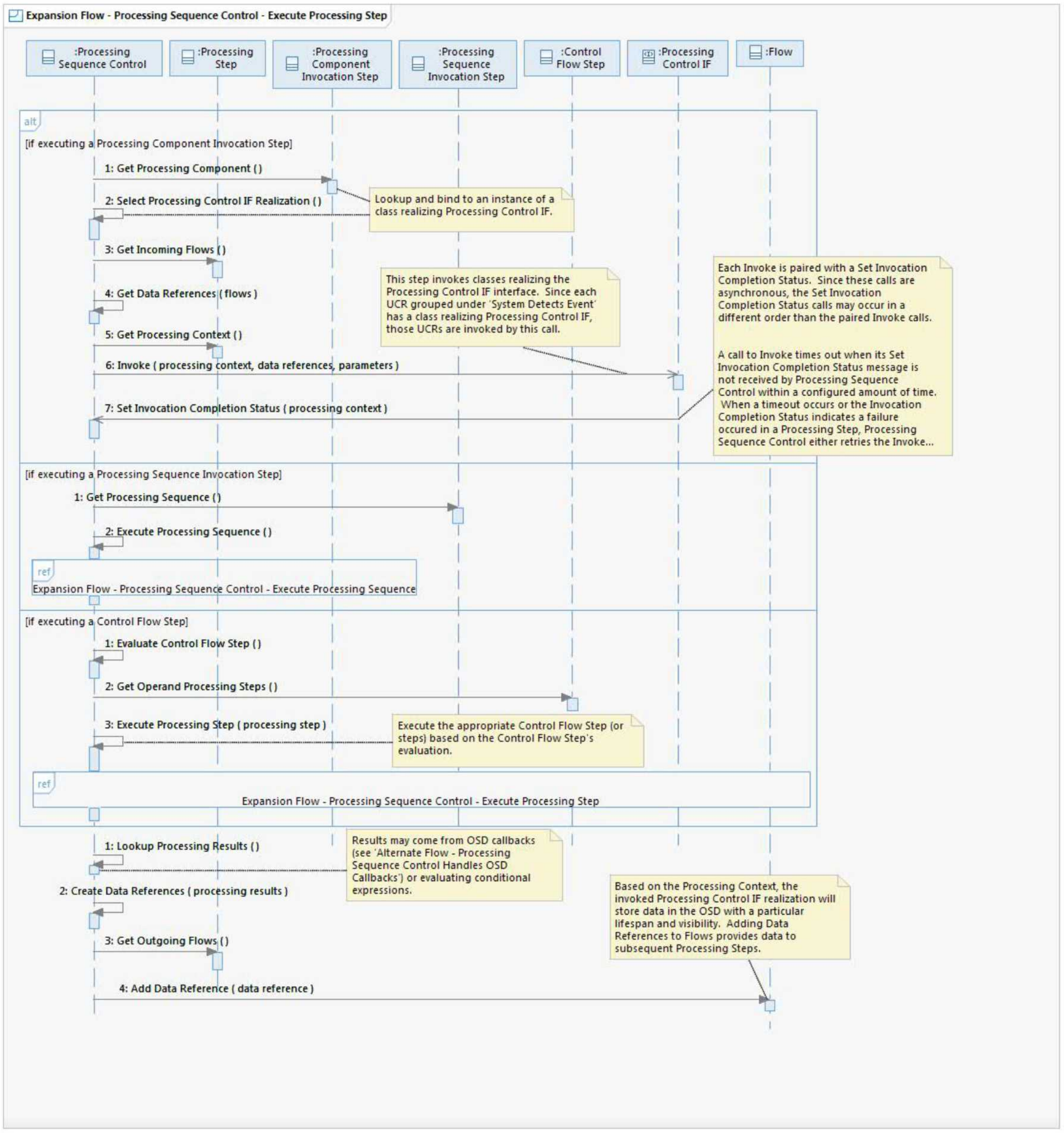

Expansion Flow - Processing Sequence Control - Execute Processing Sequence
:Processing Sequence Control
:Processing Sequence
loop
[while there are remaining Processing Steps in the Processing Sequence,,]
1: Get Processing Steps ()
2: Get Next Processing Steps ( processing sequence )
3: Execute Processing Step ( processing step )
ref
Expansion Flow - Processing Sequence Control - Execute Processing Step

This flow shows how Processing Sequence Control executes a processing sequence by iteratively executing the Processing Steps in the sequence.

*Operation Descriptions*

None

## Expansion Flow - Processing Sequence Control - Execute Processing Step



Expansion Flow - Processing Sequence Control - Execute Processing Step
:Processing Sequence Control
:Processing Step
:Processing Component Invocation Step
:Processing Sequence Invocation Step
:Control Flow Step
:Processing Control IF
:Flow
alt
[if executing a Processing Component Invocation Step]
1: Get Processing Component ()
2: Select Processing Control IF Realization ()
3: Get Incoming Flows ()
4: Get Data References ( flows )
5: Get Processing Context ()
6: Invoke ( processing context, data references, parameters )
7: Set Invocation Completion Status ( processing context )
[if executing a Processing Sequence Invocation Step]
1: Get Processing Sequence ()
2: Execute Processing Sequence ()
ref
Expansion Flow - Processing Sequence Control - Execute Processing Sequence
[if executing a Control Flow Step]
1: Evaluate Control Flow Step ()
2: Get Operand Processing Steps ()
3: Execute Processing Step ( processing step )
ref
Expansion Flow - Processing Sequence Control - Execute Processing Step
1: Lookup Processing Results ()
2: Create Data References ( processing results )
3: Get Outgoing Flows ()
4: Add Data Reference ( data reference )

Lookup and bind to an instance of a class realizing Processing Control IF.

This step invokes classes realizing the Processing Control IF interface. Since each UCR grouped under 'System Detects Event' has a class realizing Processing Control IF, those UCRs are invoked by this call.

Each Invoke is paired with a Set Invocation Completion Status. Since these calls are asynchronous, the Set Invocation Completion Status calls may occur in a different order than the paired Invoke calls.

A call to Invoke times out when its Set Invocation Completion Status message is not received by Processing Sequence Control within a configured amount of time. When a timeout occurs or the Invocation Completion Status indicates a failure occured in a Processing Step, Processing Sequence Control either retries the Invoke...

Execute the appropriate Control Flow Step (or steps) based on the Control Flow Step's evaluation.

Results may come from OSD callbacks (see 'Alternate Flow - Processing Sequence Control Handles OSD Callbacks') or evaluating conditional expressions.

Based on the Processing Context, the invoked Processing Control IF realization will store data in the OSD with a particular lifespan and visibility. Adding Data References to Flows provides data to subsequent Processing Steps.

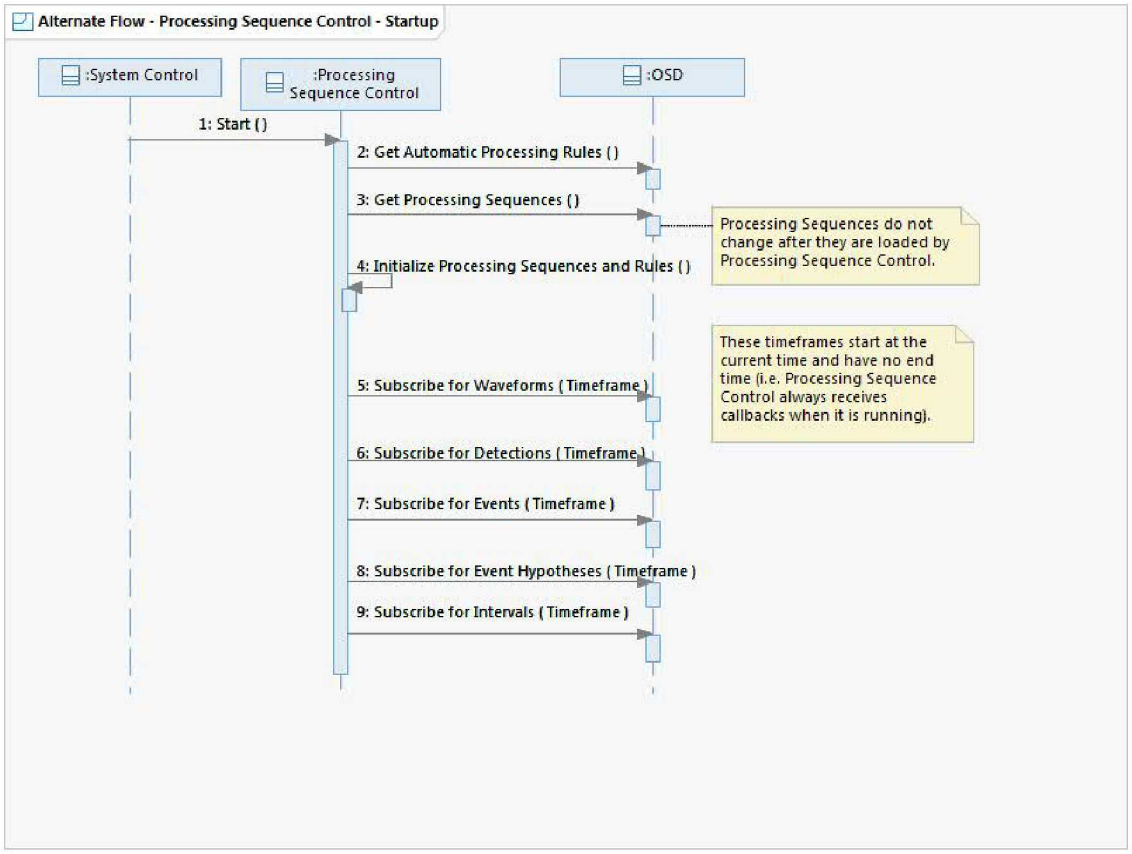
This flow shows how Processing Sequence Control executes a Processing Step. Processing Sequence Control executes a Processing Component Invocation Step by binding to the appropriate realization of the Processing Control IF interface, getting the Data References for the Processing Step, and invoking the Processing Control IF with the Data References. Processing Sequence Control monitors Invocation Completion Status to determine when the Processing Control IF invocation completes. Processing Sequence Control executes a Processing Sequence Invocation Step by initiating execution of that sequence. Processing Sequence Control executes a Control Flow Step by evaluating the Control Flow Step's conditions to determine which of the operand Processing Steps need to be executed and then executing those Processing Steps.

Regardless of Processing Step type, after Processing Sequence Control executes the Processing Step it sets Data References to the Processing Step's results on the step's outgoing Flows.

*Operation Descriptions*

None

## Alternate Flow - Processing Sequence Control - Startup



Alternate Flow - Processing Sequence Control - Startup
:System Control
:Processing Sequence Control
:OSD
1: Start ()
2: Get Automatic Processing Rules ()
3: Get Processing Sequences ()
Processing Sequences do not change after they are loaded by Processing Sequence Control.
4: Initialize Processing Sequences and Rules ()
These timeframes start at the current time and have no end time (i.e. Processing Sequence Control always receives callbacks when it is running).
5: Subscribe for Waveforms ( Timeframe )
6: Subscribe for Detections ( Timeframe )
7: Subscribe for Events ( Timeframe )
8: Subscribe for Event Hypotheses ( Timeframe )
9: Subscribe for Intervals ( Timeframe )

The flow shows how System Control starts Processing Sequence Control. Processing Sequence Control loads each Processing Sequence and subscribes for data updates from the OSD. Processing Sequence Control subscribes for updates to either provide data to Processing Sequences (see 'Alternate Flow - Processing Sequence Control Handles OSD Callbacks' or to remove data from Processing Sequences (see 'Alternate Flow - Processing Sequence Control Terminates Processing for Data Reference').

*Operation Descriptions*

***Operation: OSD::Subscribe for Detections()***

Subscribes for updates regarding signal detection creations, modifications, and associations occurring within the specified timeframe. This includes updates for new or modified unassociated signal detections.

***Operation: OSD::Subscribe for Waveforms()***

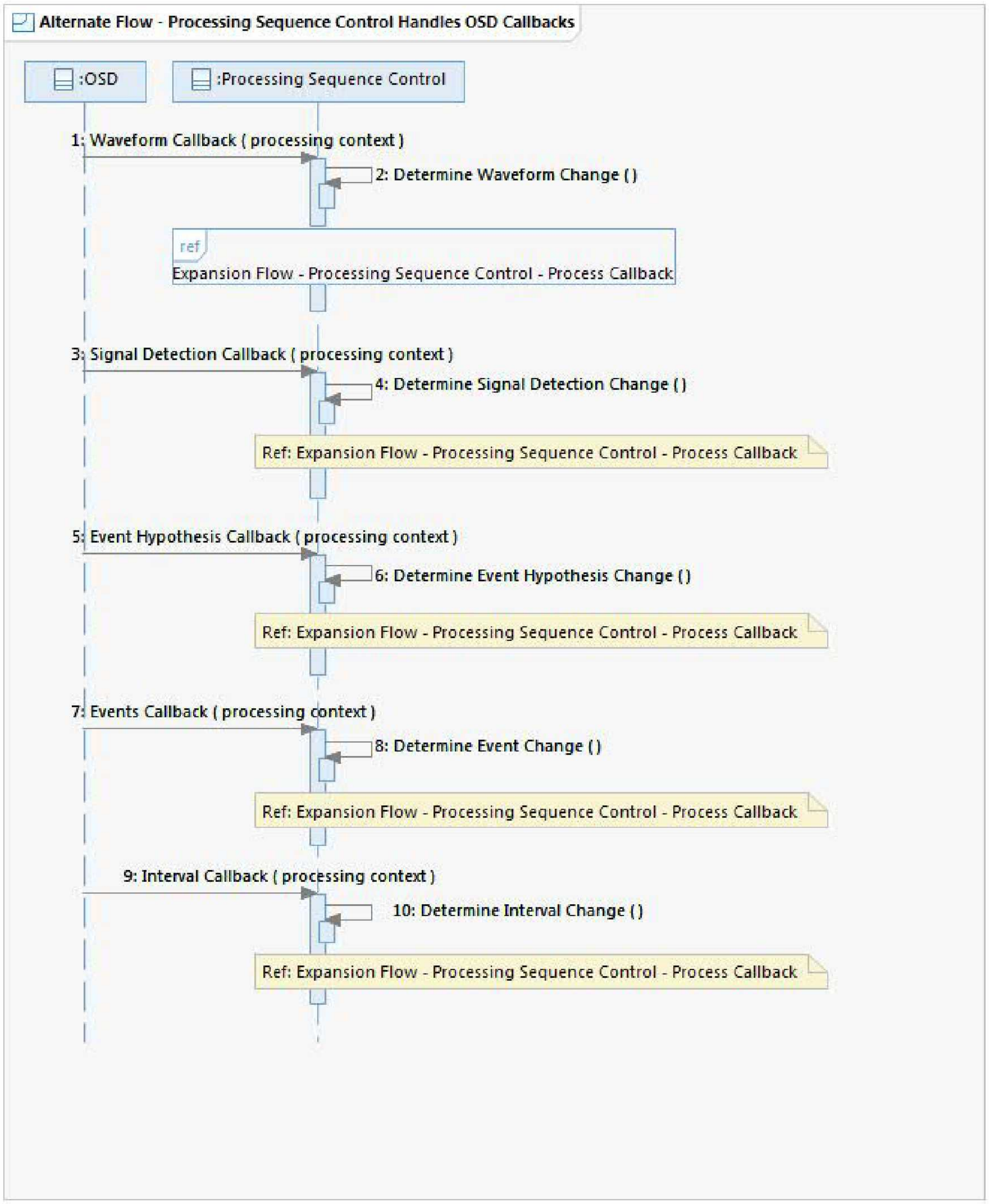
Subscribes for updates regarding raw and derived waveforms occurring within a specified timeframe. This includes information about what waveforms have been acquired by the System as well as what derived waveforms have been formed, but does not include the actual waveform data.

***Operation: OSD::Subscribe for Events()***
Subscribes for changes to Event objects within the given timeframe.  Callbacks are invoked on subscribers any time an Event within the timeframe is added or modified.

***Operation: Processing Sequence Control::Initialize Processing Sequences and Rules()***
Initializes the Automatic Processing Rules and the associated Processing Sequences so that Processing Sequence Control can evaluate the rules and initiate the sequences when the System state satisfies those rules.

***Operation: OSD::Subscribe for Intervals()***
Subscribes for changes to Interval objects that overlap with the given timeframe.  Interval objects track the active analysts and completion status of intervals corresponding to processing stages and processing activities within processing stages.  Callbacks are invoked on subscribers any time the set of active analysts or completion status for an Interval changes.

## Alternate Flow - Processing Sequence Control Handles OSD Callbacks

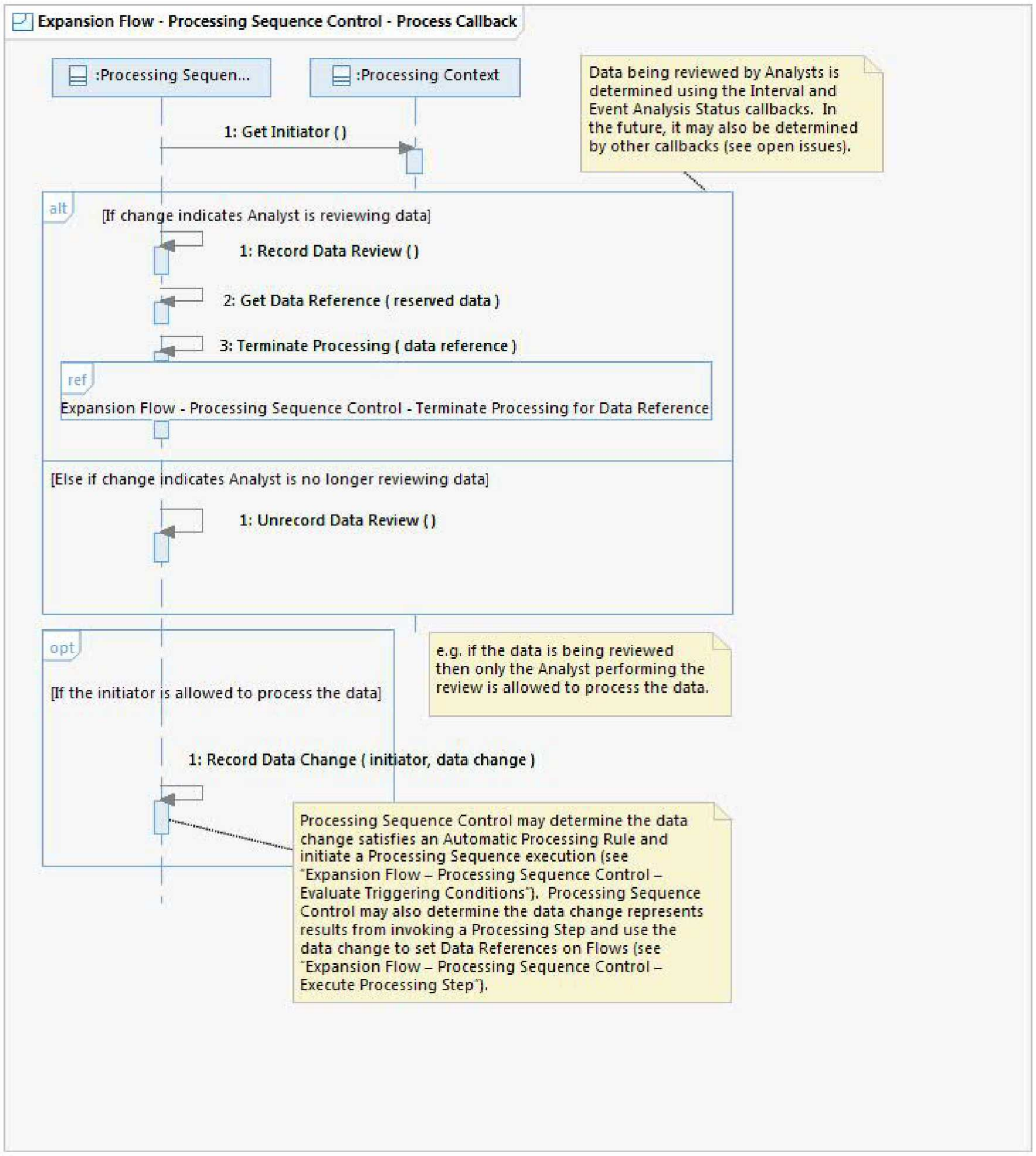

Alternate Flow - Processing Sequence Control Handles OSD Callbacks
:OSD
:Processing Sequence Control
1: Waveform Callback ( processing context )
2: Determine Waveform Change ()
ref
Expansion Flow - Processing Sequence Control - Process Callback
3: Signal Detection Callback ( processing context )
4: Determine Signal Detection Change ()
Ref: Expansion Flow - Processing Sequence Control - Process Callback
5: Event Hypothesis Callback ( processing context )
6: Determine Event Hypothesis Change ()
Ref: Expansion Flow - Processing Sequence Control - Process Callback
7: Events Callback ( processing context )
8: Determine Event Change ()
Ref: Expansion Flow - Processing Sequence Control - Process Callback
9: Interval Callback ( processing context )
10: Determine Interval Change ()
Ref: Expansion Flow - Processing Sequence Control - Process Callback

This flow shows how Processing Sequence Control processes OSD callbacks. OSD callbacks occur when data is stored in the OSD with any lifespan and visibility settings. In addition to supporting pipeline processing, this allows the initiation and running of Processing Sequences as a result of Analyst interactions with data that the Analyst has not selected to store permanently and/or make globally visible.

*Operation Descriptions*

None

## Expansion Flow - Processing Sequence Control - Process Callback



Expansion Flow - Processing Sequence Control - Process Callback
:Processing Sequen...
:Processing Context
1: Get Initiator ()
Data being reviewed by Analysts is determined using the Interval and Event Analysis Status callbacks. In the future, it may also be determined by other callbacks (see open issues).
alt
[If change indicates Analyst is reviewing data]
1: Record Data Review ()
2: Get Data Reference ( reserved data )
3: Terminate Processing ( data reference )
ref
Expansion Flow - Processing Sequence Control - Terminate Processing for Data Reference
[Else if change indicates Analyst is no longer reviewing data]
1: Unrecord Data Review ()
opt
[If the initiator is allowed to process the data]
e.g. if the data is being reviewed then only the Analyst performing the review is allowed to process the data.
1: Record Data Change ( initiator, data change )
Processing Sequence Control may determine the data change satisfies an Automatic Processing Rule and initiate a Processing Sequence execution (see "Expansion Flow – Processing Sequence Control – Evaluate Triggering Conditions"). Processing Sequence Control may also determine the data change represents results from invoking a Processing Step and use the data change to set Data References on Flows (see "Expansion Flow – Processing Sequence Control – Execute Processing Step").

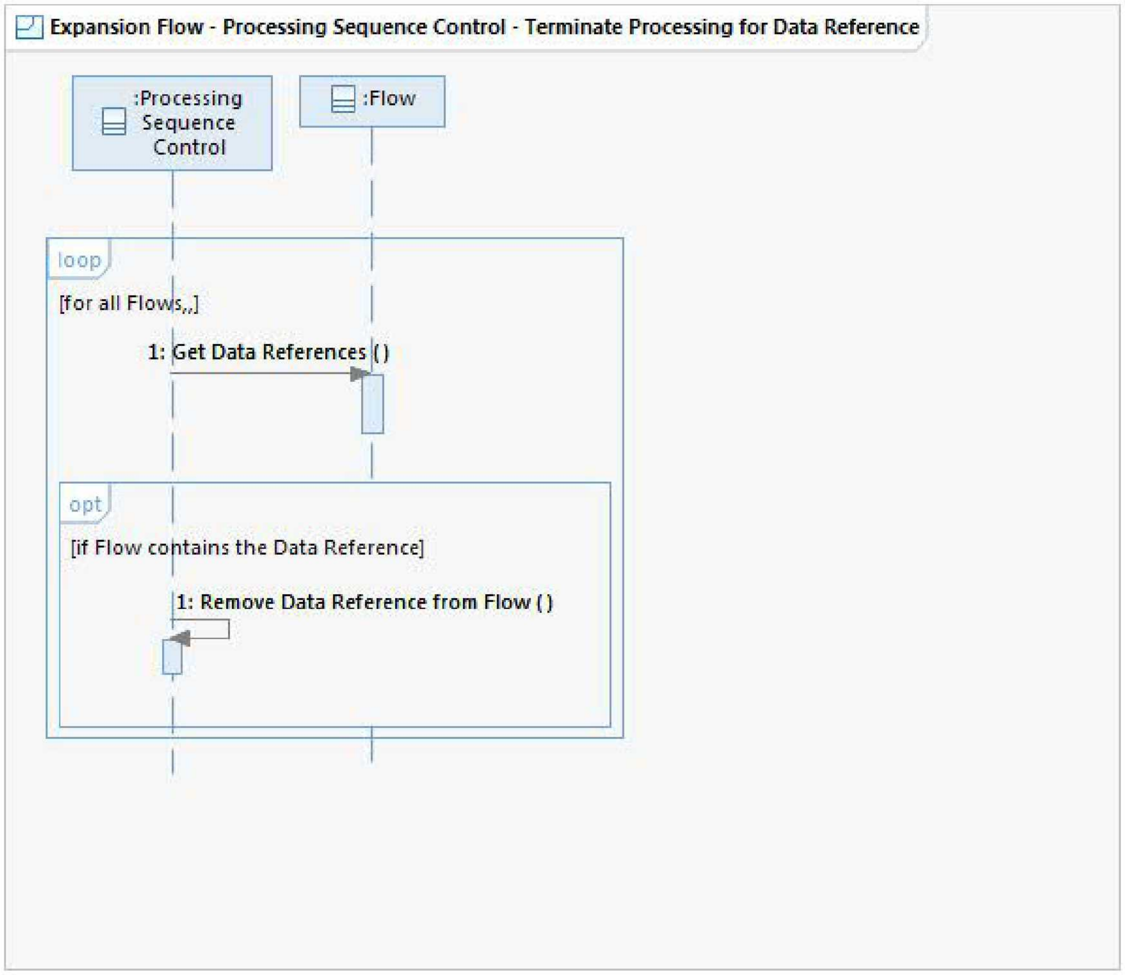
This flow shows how Processing Sequence Control processes data changes from OSD callbacks. Processing Sequence Control monitors data changes to determine which Processing Sequences are allowed to processing which types of (e.g. Processing Sequences for System processing can only use data that is not open for Analyst review). Processing Sequence Control records data changes and uses those changes to initiate Processing Sequence execution (see “Expansion Flow – Processing Sequence Control – Evaluate Triggering Conditions”) and to pass data between the

Processing Steps in a Processing Sequence (see “Expansion Flow – Processing Sequence Control – Execute Processing Step”).

*Operation Descriptions*

None

## Expansion Flow - Processing Sequence Control - Terminate Processing for Data Reference


Expansion Flow - Processing Sequence Control - Terminate Processing for Data Reference
:Processing Sequence Control
:Flow
loop
[for all Flows,,]
1: Get Data References ()
opt
[if Flow contains the Data Reference]
1: Remove Data Reference from Flow ()

This flow shows how Processing Sequence Control terminates processing for data that has been opened for Analyst review (e.g. the Analyst reserved an event for analysis (see “Refines Event” UCR), the Analyst opened a time interval to scan waveforms and unassociated signal detections (see “Scans Waveforms and Unassociated Detections” UCR) by removing all Data References to the opened data. This flow is invoked when Processing Sequence Control receives a callback from the OSD indicating the event or time interval is open for Analyst review (see “Alternate Flow - Processing Sequence Control Handles OSD Callbacks”). This flow does not terminate or interrupt any Processing Steps that are processing the data when this flow is invoked. Those Processing Steps continue processing, and Processing Sequence Control removes the data from further processing when the Processing Steps complete their processing. The results of running these Processing Steps have no effect on the data opened for analysis.

*Operation Descriptions*
None

## STATE MACHINE DIAGRAMS

None

## NOTES

**General:**

1. The Event Analyzer control class appears in this UCR as a control class that can be used to implement processing sequence control logic that is both too specific for the Processing Sequence Control mechanism to implement and which is not available in other control classes. Since the details of this additional logic do not appear in any UC, the analysis model will not further describe Event Analyzer.

**IDC Unique:**

1. Event Screening Control also realizes Processing Control IF, but it not shown in the Processing Control IF Class Diagram.

## OPEN ISSUES

1. The current model has a dependency (and a potential race condition) between processing OSD callbacks containing data changes and (see “Alternate Flow – Processing Sequence Control Handles OSD Callbacks”) processing step execution within a sequence (see “Expansion Flow – Processing Sequence Control – Execute Processing Step”). Explicitly modelling either the processing steps, Processing Control IF’s invoke() operation, or both returning data references (claim checks) with their results might be a more clear solution.

This page intentionally left blank

IDC Use Case Realization Report

# UCR-02.08 System Refines Event Location

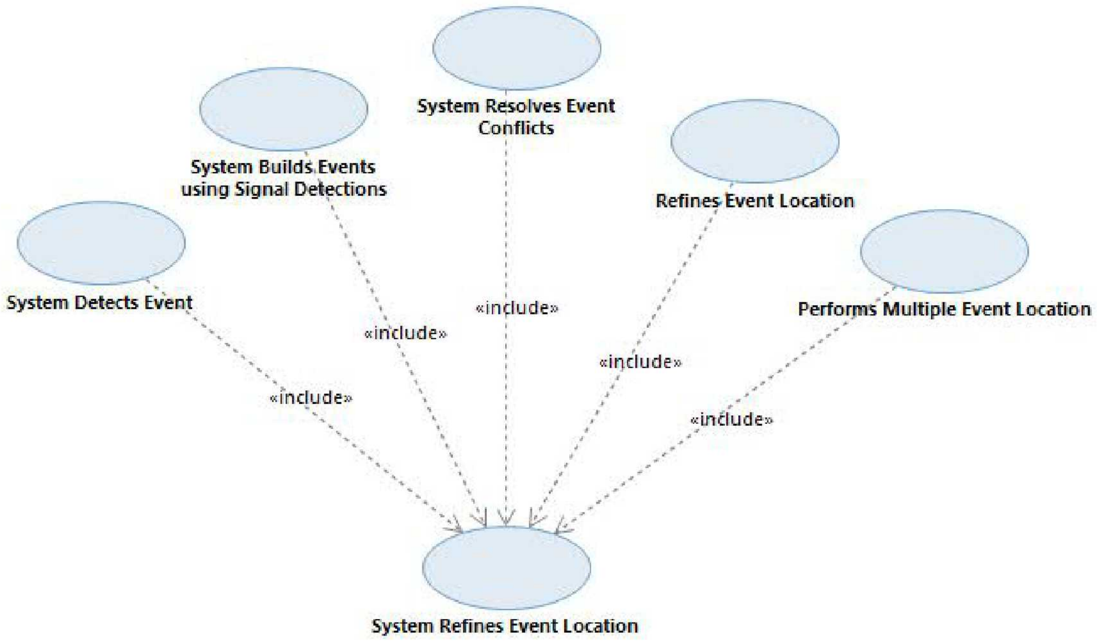
## USE CASE DESCRIPTION

This use case describes how the System refines event hypothesis location solutions using single event or multiple event algorithms. Event locations can be absolute or relative. The System locates events by finding the event location minimizing the difference between signal detection feature measurements and signal detection feature predictions (see 'System Measures Signal Features' UC). The System references both empirical knowledge from past events and geophysical models to form the signal detection feature predictions. The System also computes an uncertainty bound for each event hypothesis location solution describing a region bounding the event hypothesis' hypocenter and origin time at a particular confidence level. The System creates a variety of location solutions for each event hypothesis. These location solutions vary from one another in either the input parameters the System uses or in the location solution components the System restrains to fixed values (e.g. depth) during event location calculations. The System computes location solutions using input parameters configured by the System Maintainer (see ‘Configures Processing Components’ UC). The Analyst has the option to override input parameters originally configured by the System Maintainer (see 'Refines Event Location' UC).

## ARCHITECTURE DESCRIPTION

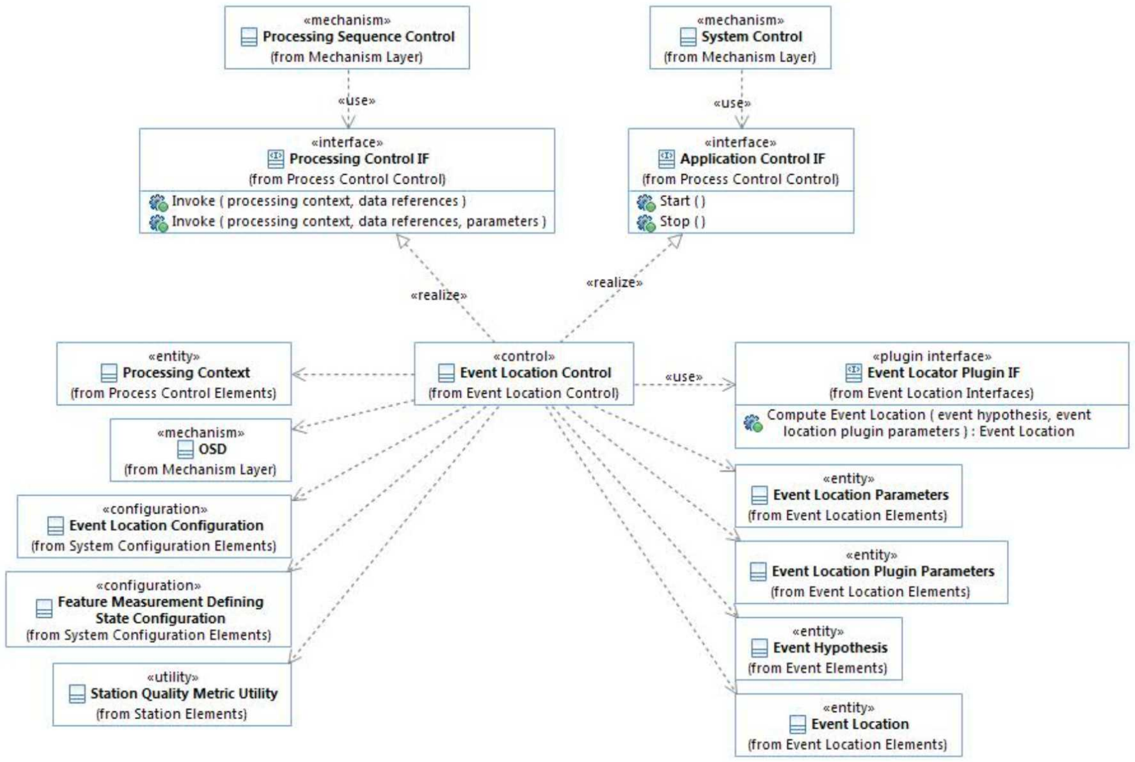
The Event Location Control class is responsible for controlling event location computations. Event Location Control may be invoked by Processing Sequence Control as part of executing a step in a Processing Sequence (see "System Detects Event" UCR), or manually invoked by an Analyst as part of refining an event (see "Refines Event Location" UCR). Event Location Control uses an Event Locator Plugin to perform the event location calculations. Various Event Locator plugins exist in the system, each realizing a common plugin interface. The specific Event Locator Plugin used varies dynamically at runtime based on the Event Location Parameters. When invoked from Processing Sequence Control, Event Locator Control uses Event Location Configuration to build up the Event Location Parameters, selects and invokes the appropriate Event Locator Plugin based on those parameters, updates the Event Hypothesis with the results, and stores the Event Hypothesis via the OSD mechanism. When invoked interactively the Analyst has the option to override the Event Location Parameters. Note that the stored Event Hypothesis is only accessible to OSD clients executing within compatible Processing Contexts (e.g. changes made within an Analyst work session may only be accessible within that session until the Analyst saves the event hypothesis with a global context).

## USE CASE DIAGRAM


System Resolves Event Conflicts
System Builds Events using Signal Detections
Refines Event Location
System Detects Event
Performs Multiple Event Location
«include»
«include»
«include»
«include»
«include»
System Refines Event Location

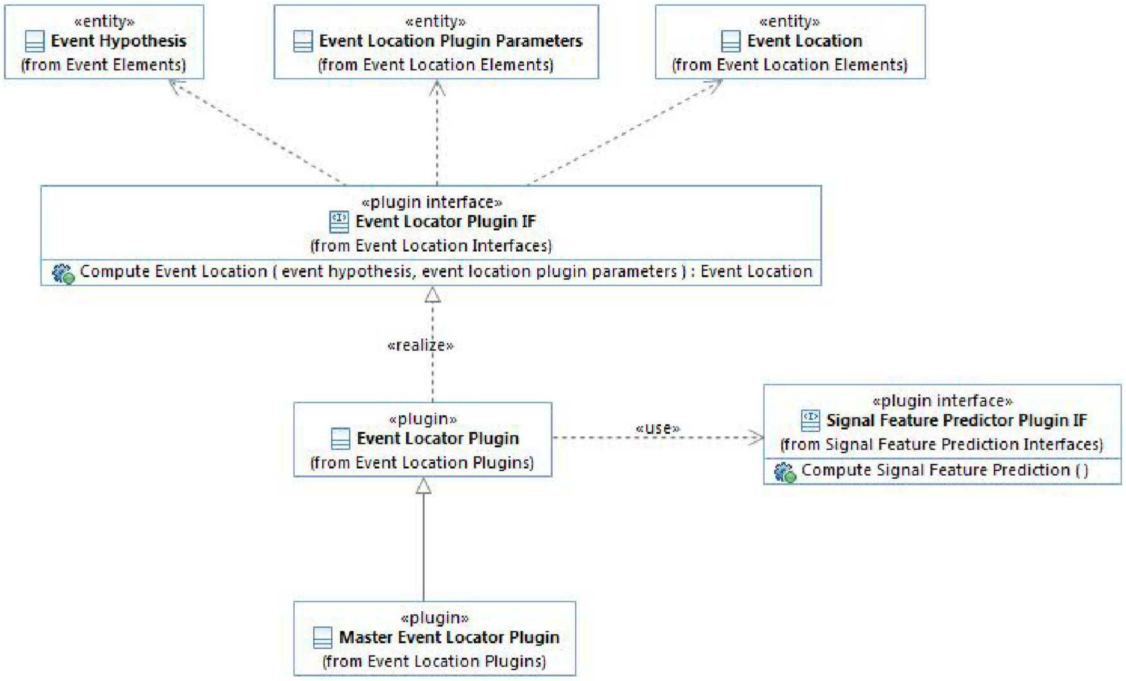
## CLASS DIAGRAMS

### Classes - Event Location Control


«mechanism» Processing Sequence Control (from Mechanism Layer)
«mechanism» System Control (from Mechanism Layer)
«use»
«use»
«interface» Processing Control IF (from Process Control Control)
Invoke ( processing context, data references )
Invoke ( processing context, data references, parameters )
«interface» Application Control IF (from Process Control Control)
Start ()
Stop ()
«realize»
«realize»
«entity» Processing Context (from Process Control Elements)
«control» Event Location Control (from Event Location Control)
«use»
«plugin interface» Event Locator Plugin IF (from Event Location Interfaces)
Compute Event Location ( event hypothesis, event location plugin parameters ) : Event Location
«mechanism» OSD (from Mechanism Layer)
«configuration» Event Location Configuration (from System Configuration Elements)
«configuration» Feature Measurement Defining State Configuration (from System Configuration Elements)
«utility» Station Quality Metric Utility (from Station Elements)
«entity» Event Location Parameters (from Event Location Elements)
«entity» Event Location Plugin Parameters (from Event Location Elements)
«entity» Event Hypothesis (from Event Elements)
«entity» Event Location (from Event Location Elements)

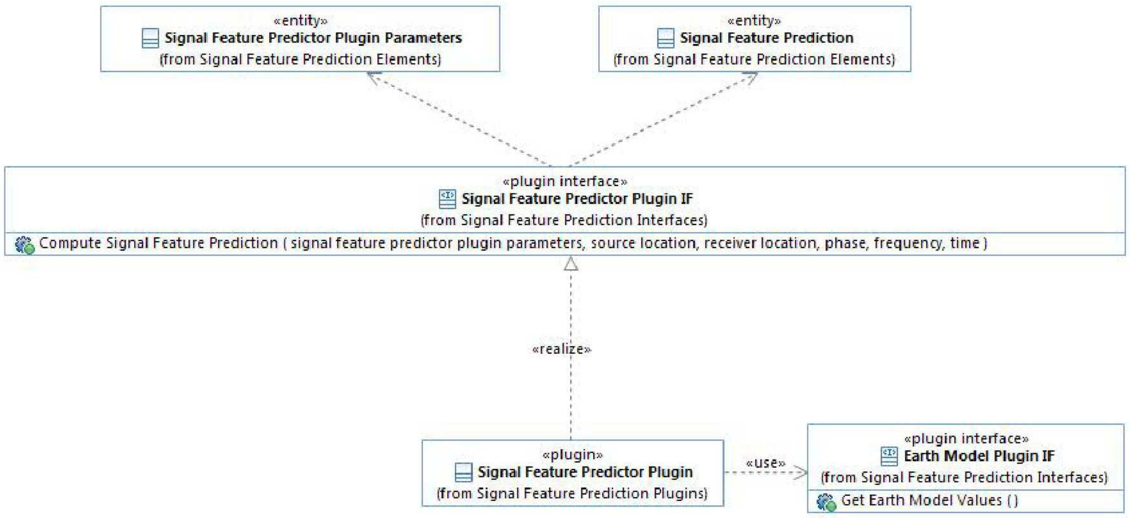
This diagram shows the Event Location Control class and related classes.

## Classes - Event Locator Plugin IF


«entity»
Event Hypothesis
(from Event Elements)
«entity»
Event Location Plugin Parameters
(from Event Location Elements)
«entity»
Event Location
(from Event Location Elements)
«plugin interface»
Event Locator Plugin IF
(from Event Location Interfaces)
Compute Event Location ( event hypothesis, event location plugin parameters ) : Event Location
«realize»
«plugin»
Event Locator Plugin
(from Event Location Plugins)
«use»
«plugin interface»
Signal Feature Predictor Plugin IF
(from Signal Feature Prediction Interfaces)
Compute Signal Feature Prediction ( )
«plugin»
Master Event Locator Plugin
(from Event Location Plugins)

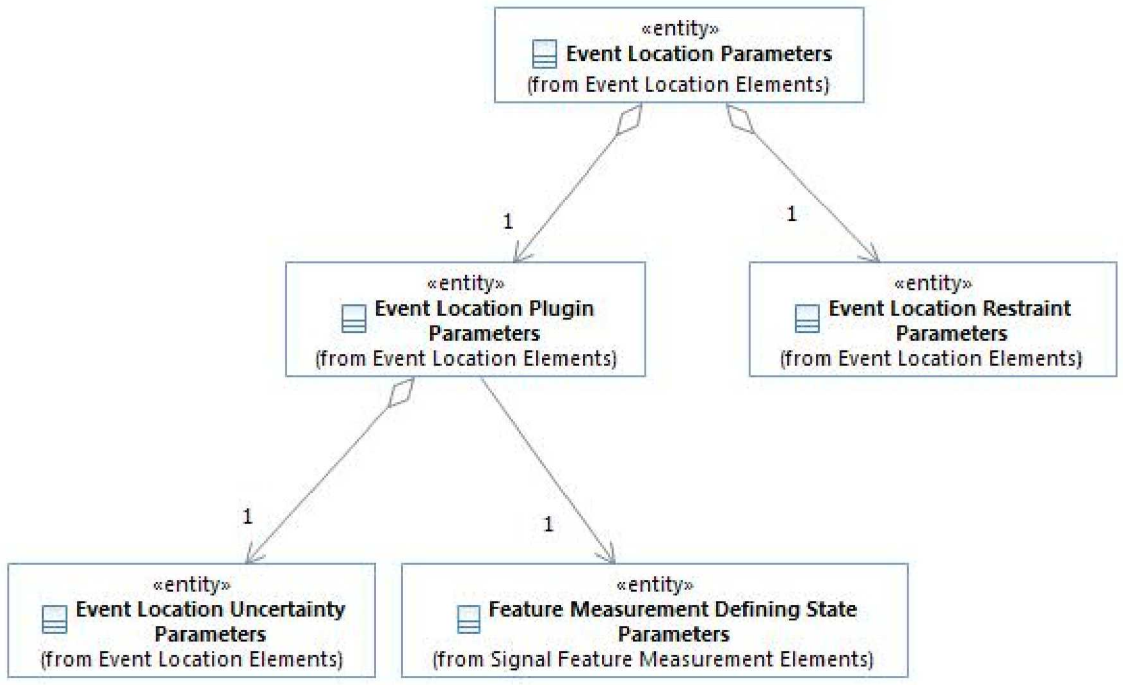
This diagram shows the Event Locator Plugin IF interface, which defines the common interface that all Event Locator Plugins must realize. An Event Locator Plugin may access plugins implementing the Signal Feature Predictor Plugin IF. For example, a locator may use a Signal Feature Predictor Plugin when calculating feature measurement residuals.

## Classes - Signal Feature Predictor Plugin IF


«entity»
Signal Feature Predictor Plugin Parameters
(from Signal Feature Prediction Elements)
«entity»
Signal Feature Prediction
(from Signal Feature Prediction Elements)
«plugin interface»
Signal Feature Predictor Plugin IF
(from Signal Feature Prediction Interfaces)
Compute Signal Feature Prediction ( signal feature predictor plugin parameters, source location, receiver location, phase, frequency, time )
«realize»
«plugin»
Signal Feature Predictor Plugin
(from Signal Feature Prediction Plugins)
«use»
«plugin interface»
Earth Model Plugin IF
(from Signal Feature Prediction Interfaces)
Get Earth Model Values ( )

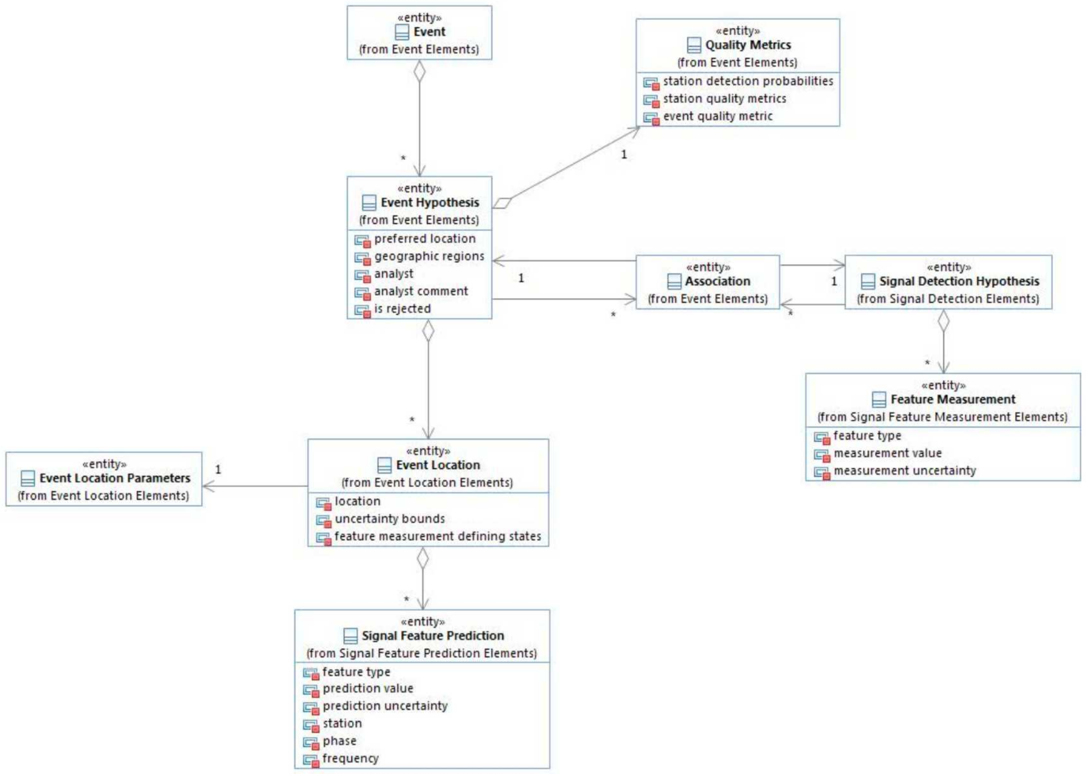
This diagram shows the Signal Feature Predictor plugin and related elements. A Signal Feature Predictor Plugin may access plugins implementing the Earth Model Plugin IF. Note that this dependency is optional and may not apply for some Signal Feature Predictor Plugin implementations.

## Classes - Event Location Parameters


«entity»
Event Location Parameters
(from Event Location Elements)
1
1
«entity»
Event Location Plugin Parameters
(from Event Location Elements)
«entity»
Event Location Restraint Parameters
(from Event Location Elements)
1
1
«entity»
Event Location Uncertainty Parameters
(from Event Location Elements)
«entity»
Feature Measurement Defining State Parameters
(from Signal Feature Measurement Elements)

This diagram shows details of the Event Location Parameters class. The Event Location Parameters class is used by Event Location Control to determine general behavior of location calculations and the Event Location Plugin Parameters class is provided as input to the Event Locator Plugin to control specific algorithm behavior. Event Location Control creates the parameters from the Event Location Configuration preconfigured by the System Maintainer (see "Configures Processing Components" UCR). The Analyst may override the parameters (see "Refines Event Location" UCR).

## Classes - Event Hypothesis


«entity» Event (from Event Elements)
«entity» Quality Metrics (from Event Elements)
station detection probabilities
station quality metrics
event quality metric
«entity» Event Hypothesis (from Event Elements)
preferred location
geographic regions
analyst
analyst comment
is rejected
«entity» Association (from Event Elements)
«entity» Signal Detection Hypothesis (from Signal Detection Elements)
«entity» Feature Measurement (from Signal Feature Measurement Elements)
feature type
measurement value
measurement uncertainty
«entity» Event Location Parameters (from Event Location Elements)
«entity» Event Location (from Event Location Elements)
location
uncertainty bounds
feature measurement defining states
«entity» Signal Feature Prediction (from Signal Feature Prediction Elements)
feature type
prediction value
prediction uncertainty
station
phase
frequency
1
*

This diagram shows the portions of an Event Hypothesis that are used to compute an Event Location as well as the portions that are computed by the algorithm. The event location algorithm analyzes Feature Measurements of Signal Detection Hypotheses associated to one or more Event Hypotheses. The algorithm uses the station location associated with each Signal Detection Hypothesis as well as features of the detection to compute the Event Location, which is the primary output of the event location algorithm. Multiple different Event Locations (each with a different set of Event Location Restraints, as stored in the Event Location Parameters associated to that Event Location) may be computed for each Event Hypothesis. Event Location Control marks one of these Event Locations as the preferred location for an Event Hypothesis. The Event Location Parameters class captures the parameters that were provided as input to the event location algorithm, enabling subsequent Analysts to recompute the location with the same parameters.

## CLASS DESCRIPTIONS

***<<configuration>> Event Location Configuration***

Default event location configuration as configured by the System Maintainer. Contains configuration about which Event Locator Plugins the System should invoke, as well as the types of location uncertainty bounds and restrained locations the System should compute. The Analyst may override the Event Location Parameters computed from this configuration.

***<<configuration>> Feature Measurement Defining State Configuration***

Represents all signal detection feature measurement defining state configuration in the system. This includes all configurations used to determine which signal detection feature measurements are by default defining and non-defining for various types of system calculations.

***<<control>> Event Location Control***
Responsible for controlling the event location computation. Retrieves necessary data, invokes the appropriate Event Locator Plugin to compute the new location, and stores the result.

***<<entity>> Association***
Represents an association between a Signal Detection Hypothesis and an Event Hypothesis.

***<<entity>> Event***
Represents information about an Event. Keeps track of all the Event Hypotheses for the event, which hypothesis is the preferred one for each processing stage, the active analysts for the event (i.e. whether the event is under "active review"), whether the event is "complete" for each processing stage, and other event-related information.

***<<entity>> Event Hypothesis***
Represents geophysical information about an Event as determined by an Analyst or through pipeline processing. There can be multiple hypotheses of the same Event (e.g. different associated signal detection hypotheses, different location solutions).

***<<entity>> Event Location***
Represents a computed location for an event.

***<<entity>> Event Location Parameters***
Represents the parameters that are used by Event Location Control. This includes which Event Locator Plugin to invoke as well as the types of restrained event locations that Event Location Control will invoke the plugin to compute. Initially set by the System based on the Event Location Configuration defined by the System Maintainer, but the Analyst may select to override parameter values when refining events.

***<<entity>> Event Location Plugin Parameters***
Represents all parameters passed to an Event Locator Plugin. This includes parameters describing default feature measurement defining states and the types of uncertainty bounds the plugin should compute. May also include parameters specific to the plugin being invoked.

***<<entity>> Event Location Restraint Parameters***
Represents restraints on the location coordinate spaces (lat, lon, depth or time) for the event location computation. Restraints indicate which coordinate spaces are restrained and the associated restrained value (or value range) to be used for that coordinate.

***<<entity>> Event Location Uncertainty Parameters***
Represents the type of uncertainty bound (Confidence, Coverage or K-Weighted), confidence level and scaling factor for the locator to use when computing event location uncertainty.

***<<entity>> Feature Measurement***
Represents the value and uncertainty of a measured feature of a signal detection.

***<<entity>> Feature Measurement Defining State Parameters***
Represents defining state parameters for feature measurements. The parameters include the following for each feature measurement for each type of calculation (e.g. location, magnitude, etc.)
- Whether the feature measurement is initially defining or non-defining
- Whether an algorithm is free to toggle the defining state
- Whether an Analyst is free to toggle the defining state

***<<entity>> Processing Context***
Represents the context in which data is being stored and/or processed. This includes the processing session (e.g. processed by Analyst vs. processed by System). For Analyst processing, may identify the Analyst work session. For System processing, may identify the Processing Sequence and/or Processing Step being executed (including a way to identify a particular Processing Sequence and Processing Step among the many possible instantiations), the visibility for the results (private vs. global), and the lifespan of the data (transient vs. persistent). This information is needed by the Processing Sequence Control to manage the execution of Processing Sequences, which may execute in the context of an Analyst refining an event or in the context of the system initiating automatic processing. It is also needed by the Object Storage and Distribution (OSD) mechanism to determine how to store and distribute the data.

***<<entity>> Quality Metrics***
Represents quality metrics for a single event hypothesis. This includes the event quality metric, station quality metrics at the time the event occurred, and station probabilities of detecting the event.

***<<entity>> Signal Detection Hypothesis***
Represents geophysical information about a Signal Detection as determined by an Analyst or through pipeline processing. There can be multiple hypotheses of the same Signal Detection (e.g. different onset times, different phase labels).

***<<entity>> Signal Feature Prediction***
Represents a predicted signal feature (e.g., travel time, azimuth, slowness, amplitude, probability of detection) and the associated uncertainties.

***<<entity>> Signal Feature Predictor Plugin Parameters***
Represents the parameters used by an invocation of a Signal Feature Predictor Plugin. This includes parameters that apply to all Signal Feature Predictor Plugins and may also include plugin specific parameters.

***<<interface>> Application Control IF***

Defines the interface implemented by all <<control>> classes in the system that are controlled by System Control.

***<<interface>> Processing Control IF***
Defines the interface implemented by all <<control>> classes in the system that are invoked by the Processing Sequence Control <<mechanism>>.

***<<mechanism>> OSD***
Represents the Object Storage and Distribution mechanism for storing and distributing data objects internally within the system.

***<<mechanism>> Processing Sequence Control***
Mechanism for executing and controlling processing sequences configured by the System Maintainer.

***<<plugin interface>> Earth Model Plugin IF***
Standard interface for all Earth Model plugins. All Earth Model plugins in the system realize this interface.

***<<plugin interface>> Event Locator Plugin IF***
Standard interface for all Event Locator plugins. All Event Locator plugins in the system realize this interface.

***<<plugin interface>> Signal Feature Predictor Plugin IF***
Standard interface for all Signal Feature Predictor plugins. All Signal Feature Predictor plugins in the system realize this interface. Plugins that implement Signal Feature Predictor IF may predict different types of signal features, such as travel time, azimuth, slowness, amplitude, and probability of detection.

***<<plugin>> Event Locator Plugin***
Abstract base class for Event Locator Plugins that may be plugged in to the system behind the Event Locator Plugin IF plugin interface. Event Locator Plugins are responsible for calculating event locations. Configuration for specific plugin implementations include the settings for controlling their behavior (e.g. settings for max number of iterations, which Signal Feature Predictor Plugin and Earth Model Plugin to use, etc.).

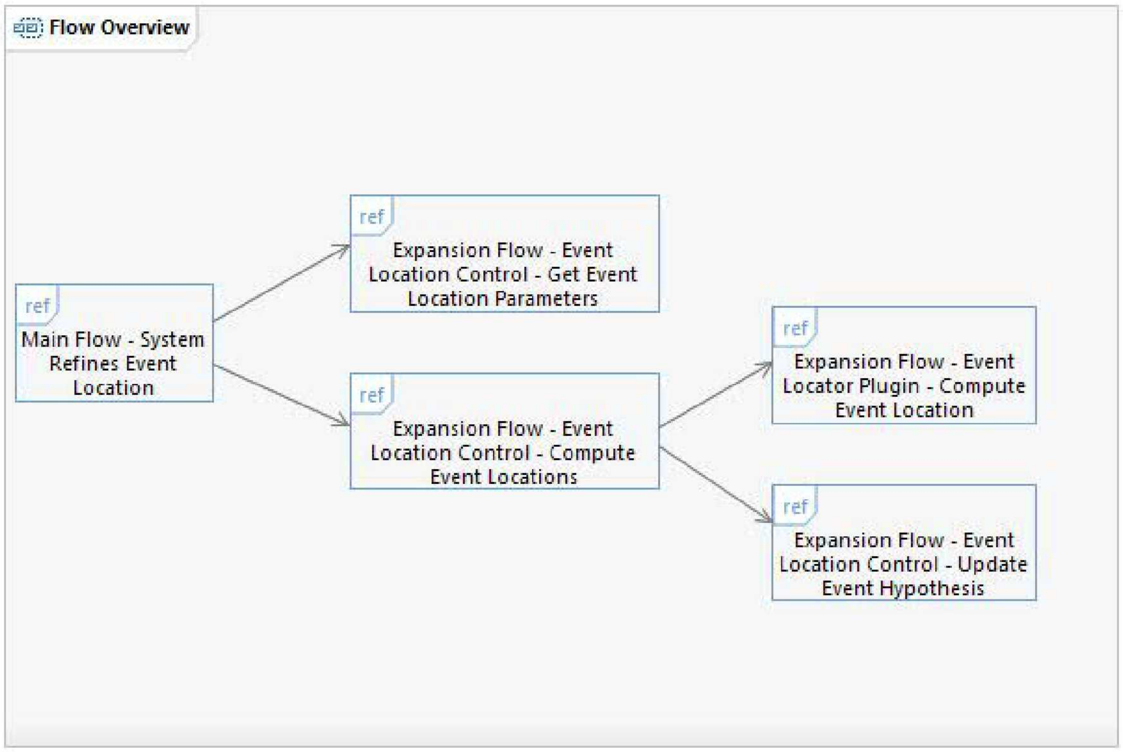
***<<plugin>> Master Event Locator Plugin***
Specialization of Event Locator Plugin that locates an event based on a specified "master event" (i.e. Master Event Relocation). Configuration for this plugin includes information required to select which master events to use (potentially based on geographic region), which signal detections and feature measurements to use, etc. Parameters for this plugin may include a particular master event, the location and location uncertainty of the master event, and feature measurements and associated uncertainties pertinent to location computation for signal detections associated to the master event that also exists on the event under refinement (i.e. signal detections with the same channel and phase).

***<<plugin>> Signal Feature Predictor Plugin***
Abstract class that represents any/all of the Signal Feature Predictor Plugins that may be plugged in to the system behind the Signal Feature Predictor Plugin IF plugin interface. Signal Feature Predictor Plugins are responsible for calculating signal feature predictions.

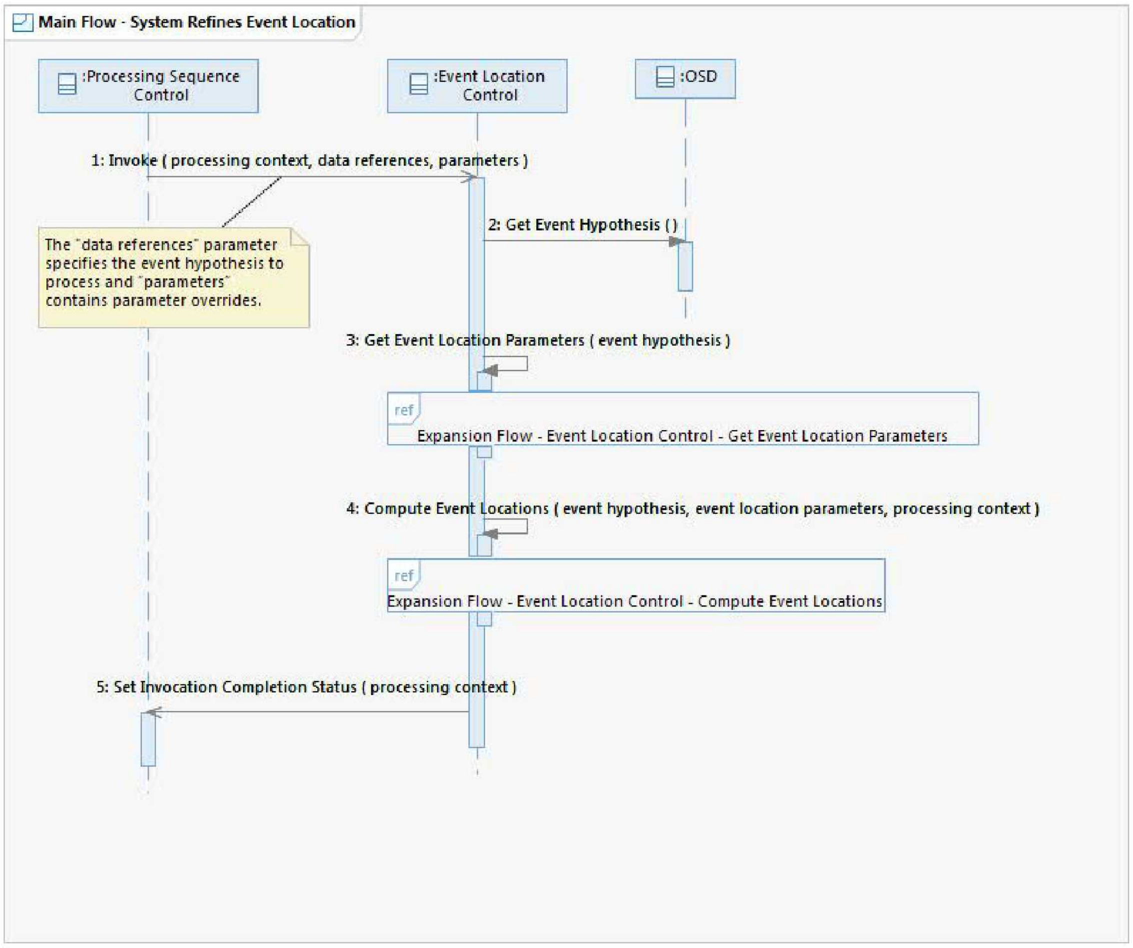
***<<utility>> Station Quality Metric Utility***
Utility class that computes the station quality metric.

## SEQUENCE DIAGRAMS

### Flow Overview


Flow Overview
ref
Main Flow - System Refines Event Location
ref
Expansion Flow - Event Location Control - Get Event Location Parameters
ref
Expansion Flow - Event Location Control - Compute Event Locations
ref
Expansion Flow - Event Locator Plugin - Compute Event Location
ref
Expansion Flow - Event Location Control - Update Event Hypothesis

## Main Flow - System Refines Event Location

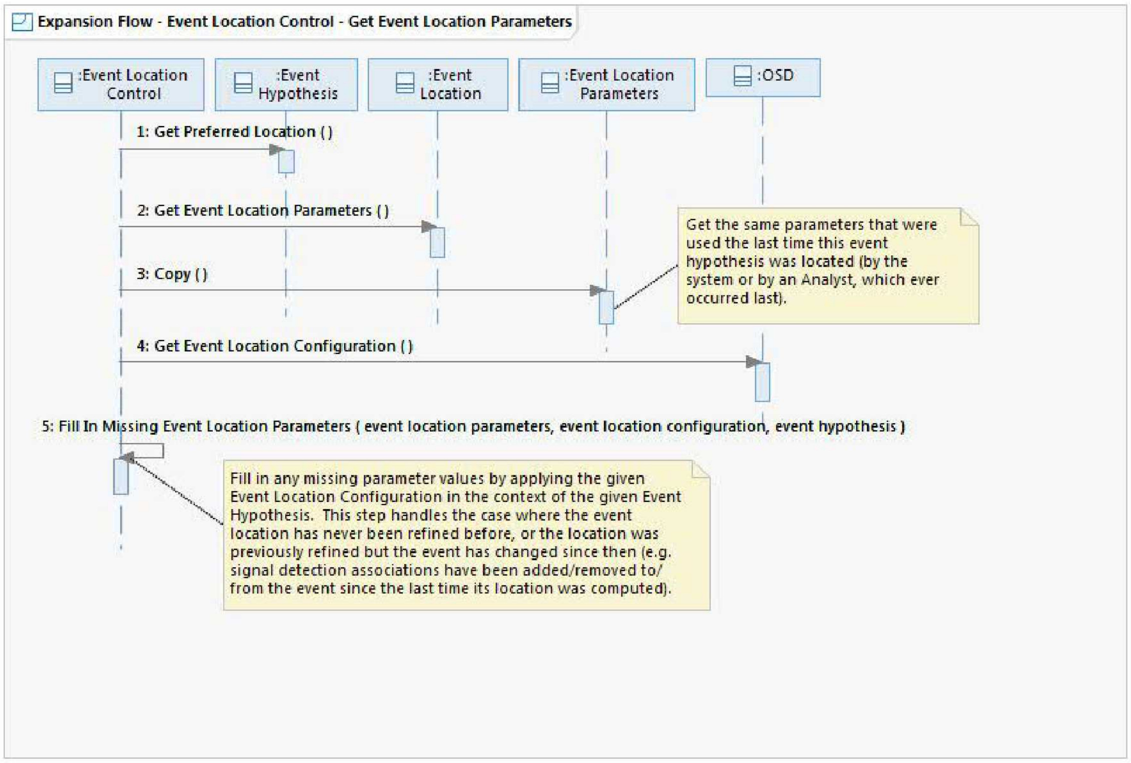

Main Flow - System Refines Event Location
:Processing Sequence Control
:Event Location Control
:OSD
1: Invoke ( processing context, data references, parameters )
The "data references" parameter specifies the event hypothesis to process and "parameters" contains parameter overrides.
2: Get Event Hypothesis ( )
3: Get Event Location Parameters ( event hypothesis )
ref
Expansion Flow - Event Location Control - Get Event Location Parameters
4: Compute Event Locations ( event hypothesis, event location parameters, processing context )
ref
Expansion Flow - Event Location Control - Compute Event Locations
5: Set Invocation Completion Status ( processing context )

This flow shows how the system refines event location. This flow is stimulated by the Processing Sequence Control mechanism as part of executing an automatic processing sequence. The precise triggering conditions for such sequences are configured by the System Maintainer (see "Defines Processing Sequence" UCR). For more information about the Processing Sequence Control mechanism see "System Detects Event" UCR.

*Operation Descriptions*

None

## Expansion Flow - Event Location Control - Get Event Location Parameters

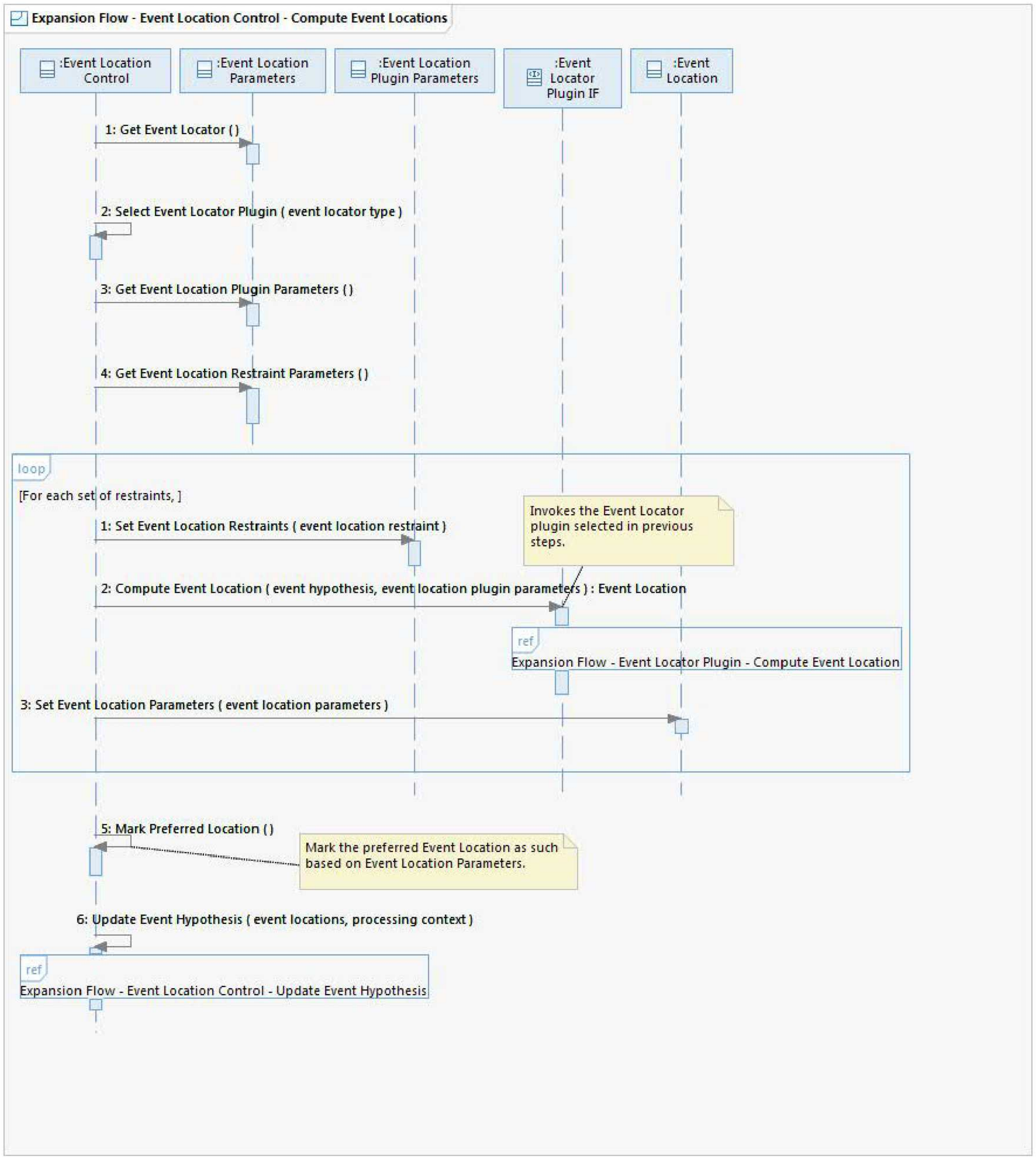

Expansion Flow - Event Location Control - Get Event Location Parameters
:Event Location Control
:Event Hypothesis
:Event Location
:Event Location Parameters
:OSD
1: Get Preferred Location ()
2: Get Event Location Parameters ()
3: Copy ()
4: Get Event Location Configuration ()
5: Fill In Missing Event Location Parameters ( event location parameters, event location configuration, event hypothesis )
Get the same parameters that were used the last time this event hypothesis was located (by the system or by an Analyst, which ever occurred last).
Fill in any missing parameter values by applying the given Event Location Configuration in the context of the given Event Hypothesis. This step handles the case where the event location has never been refined before, or the location was previously refined but the event has changed since then (e.g. signal detection associations have been added/removed to/ from the event since the last time its location was computed).

This flow shows how the Event Location Control class builds up the Event Location Parameters. Note that this flow may also be invoked directly from the Refines Event Location Display (see "Refines Event Location" UCR).

*Operation Descriptions*

None

## Expansion Flow - Event Location Control - Compute Event Locations


Expansion Flow - Event Location Control - Compute Event Locations
:Event Location Control
:Event Location Parameters
:Event Location Plugin Parameters
:Event Locator Plugin IF
:Event Location
1: Get Event Locator ( )
2: Select Event Locator Plugin ( event locator type )
3: Get Event Location Plugin Parameters ( )
4: Get Event Location Restraint Parameters ( )
loop
[For each set of restraints, ]
1: Set Event Location Restraints ( event location restraint )
Invokes the Event Locator plugin selected in previous steps.
2: Compute Event Location ( event hypothesis, event location plugin parameters ) : Event Location
ref
Expansion Flow - Event Locator Plugin - Compute Event Location
3: Set Event Location Parameters ( event location parameters )
5: Mark Preferred Location ( )
Mark the preferred Event Location as such based on Event Location Parameters.
6: Update Event Hypothesis ( event locations, processing context )
ref
Expansion Flow - Event Location Control - Update Event Hypothesis

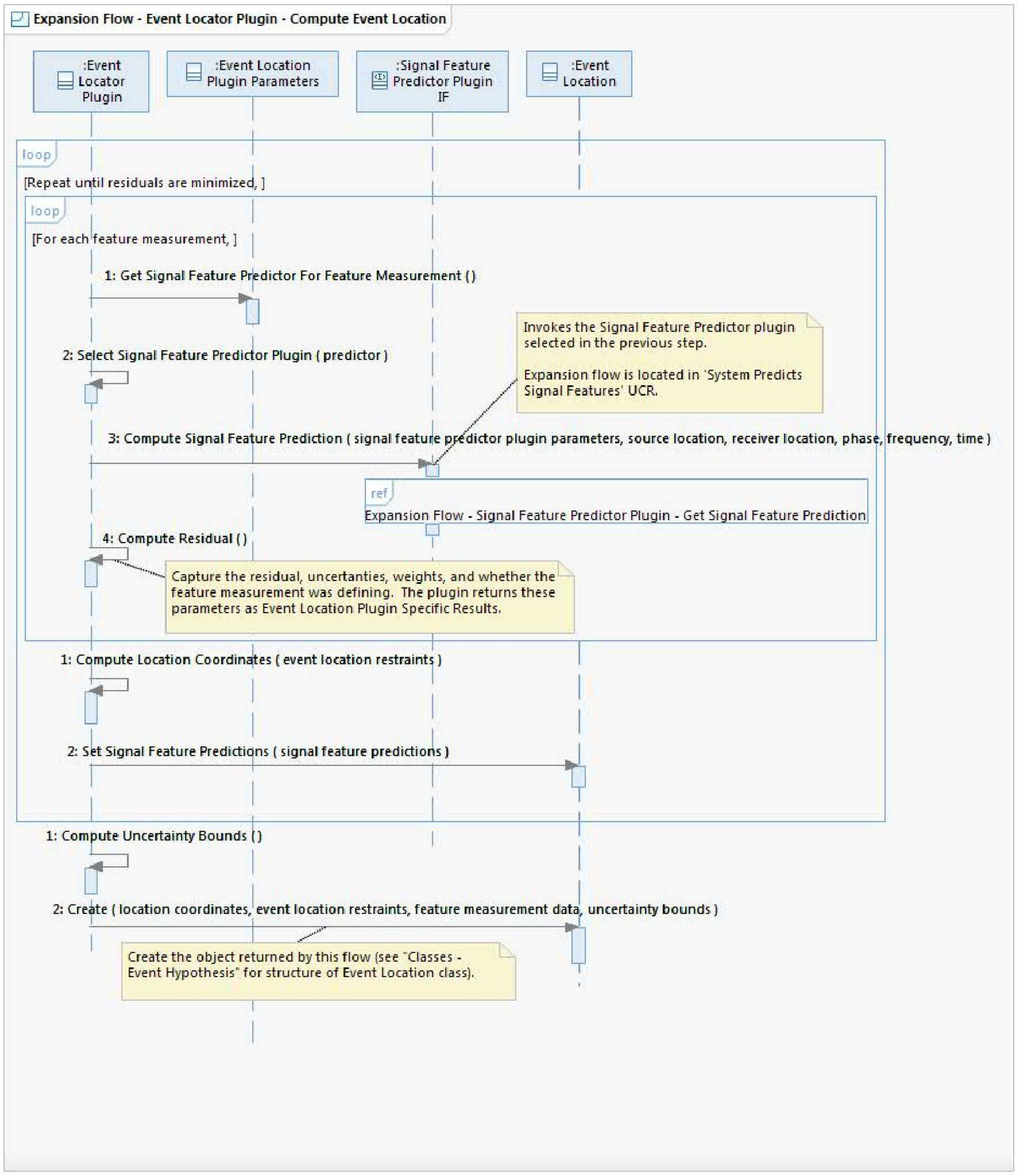
This flow shows how the Event Location Control invokes the Event Locator Plugin to compute locations for an event hypothesis. The control class selects which Event Locator Plugin to invoke based on the event locator type specified in the Event Location Parameters and then invokes the Event Locator Plugin through the Event Locator Plugin IF plugin interface. The Event Locator Plugin returns an Event Location. Event Location Control updates the Event Hypothesis with all of the computed Event Locations. Note that this flow may also be invoked directly from the Refines Event Location Display (see "Refines Event Location" UCR).

*Operation Descriptions*

***Operation: Event Location Control::Select Event Locator Plugin()***

Select and bind to the specific event locator plugin that corresponds to the given event location algorithm.

## Expansion Flow - Event Locator Plugin - Compute Event Location


Expansion Flow - Event Locator Plugin - Compute Event Location
:Event Locator Plugin
:Event Location Plugin Parameters
:Signal Feature Predictor Plugin IF
:Event Location
loop
[Repeat until residuals are minimized, ]
loop
[For each feature measurement, ]
1: Get Signal Feature Predictor For Feature Measurement ()
2: Select Signal Feature Predictor Plugin ( predictor )
Invokes the Signal Feature Predictor plugin selected in the previous step.
Expansion flow is located in 'System Predicts Signal Features' UCR.
3: Compute Signal Feature Prediction ( signal feature predictor plugin parameters, source location, receiver location, phase, frequency, time )
ref
Expansion Flow - Signal Feature Predictor Plugin - Get Signal Feature Prediction
4: Compute Residual ()
Capture the residual, uncertanties, weights, and whether the feature measurement was defining. The plugin returns these parameters as Event Location Plugin Specific Results.
1: Compute Location Coordinates ( event location restraints )
2: Set Signal Feature Predictions ( signal feature predictions )
1: Compute Uncertainty Bounds ()
2: Create ( location coordinates, event location restraints, feature measurement data, uncertainty bounds )
Create the object returned by this flow (see "Classes - Event Hypothesis" for structure of Event Location class).

This flow notionally shows how a particular Event Locator Plugin might compute an Event Location for an Event Hypothesis. The flow shown here may not apply to all Event Locator Plugins. In this example, the Event Locator Plugin iteratively computes residuals between observed vs. predicted feature measurements and updates the location coordinates until the residuals no longer improve. To compute Signal Feature Predictions the Event Locator Plugin uses a Signal Feature Predictor Plugin. The specific predictor used may vary for each prediction based on parameters specified in the Event Location Plugin Parameters class.

As a possible variation of this flow for performing Master Event Location, the flow might look essentially the same except that instead of using a Signal Feature Predictor to get the Signal Feature Predictions the Event Locator might use the Feature Measurements associated with a designated "master event".

*Operation Descriptions*

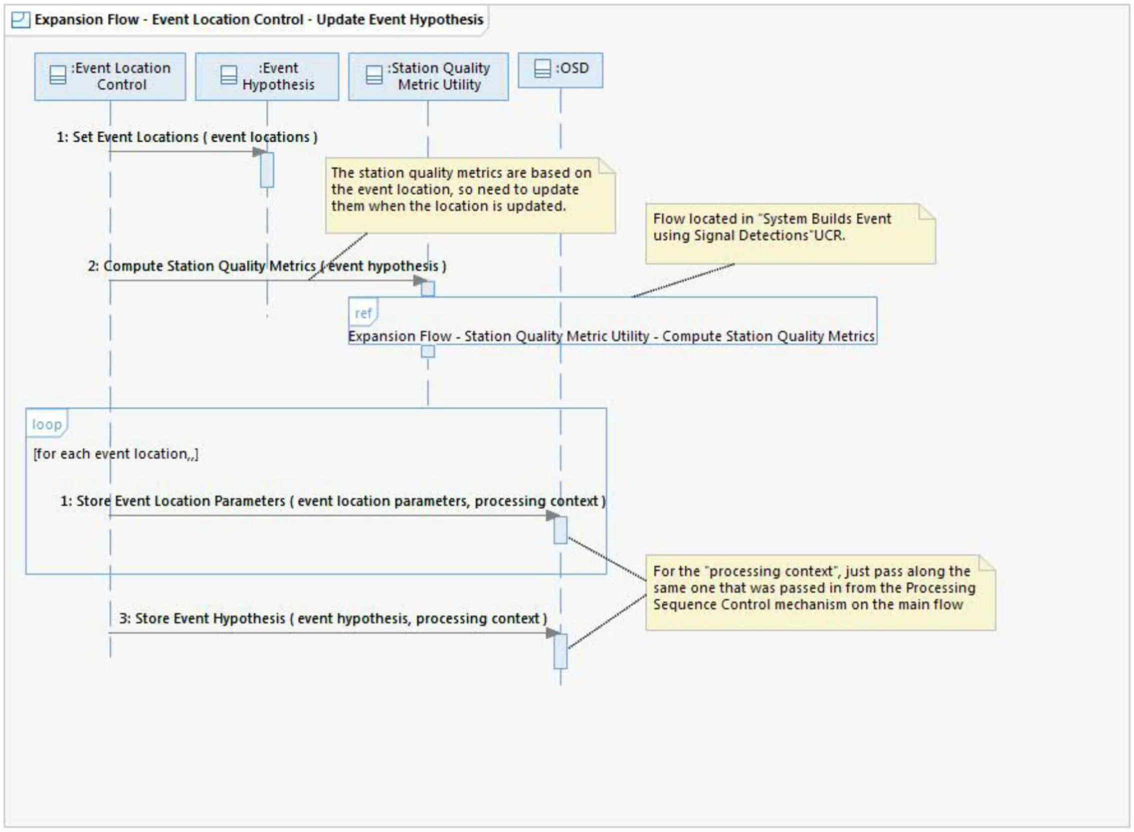
***Operation: Event Locator Plugin::Select Signal Feature Predictor Plugin()***

Select and bind to the specific Signal Feature Predictor plugin that corresponds to the given signal feature predictor.

***Operation: Event Locator Plugin::Compute Residual()***

Compute the difference between the signal feature measurement and the signal feature prediction, as well as the residual uncertainty.

## Expansion Flow - Event Location Control - Update Event Hypothesis


Expansion Flow - Event Location Control - Update Event Hypothesis
:Event Location Control
:Event Hypothesis
:Station Quality Metric Utility
:OSD
1: Set Event Locations ( event locations )
The station quality metrics are based on the event location, so need to update them when the location is updated.
Flow located in "System Builds Event using Signal Detections"UCR.
2: Compute Station Quality Metrics ( event hypothesis )
ref
Expansion Flow - Station Quality Metric Utility - Compute Station Quality Metrics
loop
[for each event location,,]
1: Store Event Location Parameters ( event location parameters, processing context )
For the "processing context", just pass along the same one that was passed in from the Processing Sequence Control mechanism on the main flow
3: Store Event Hypothesis ( event hypothesis, processing context )

This flow shows how the Event Location Control class updates the Event Hypothesis passed in from Processing Sequence Control with the Event Location objects returned by the Event Locator Plugin. Event Location Control also stores the Event Location Parameters used for each Event Location and recomputes the station quality metrics based on the updated event location.

*Operation Descriptions*

***Operation: OSD::Store Event Hypothesis()***

Store the given Event Hypothesis with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

## STATE MACHINE DIAGRAMS

None

## NOTES

1. This UCR shows the system refining the location for one event hypothesis at a time. However, in the actual implementation it may be desirable to support relocating a batch of events at a time, for efficiency. Batching is not shown in this UCR since it is considered a design/implementation optimization.

2. Computation of the event quality metric after an event is relocated is configurable (see 'Defines Processing Sequence' UC). Configuration could include selection of a minimum change in location that would result in recomputing the event quality metric or selection of when to recompute the event quality metric based on the cause for the event relocation.

## OPEN ISSUES

None

This page intentionally left blank

IDC Use Case Realization Report

# UCR-03.03 Refines Event

## USE CASE DESCRIPTION

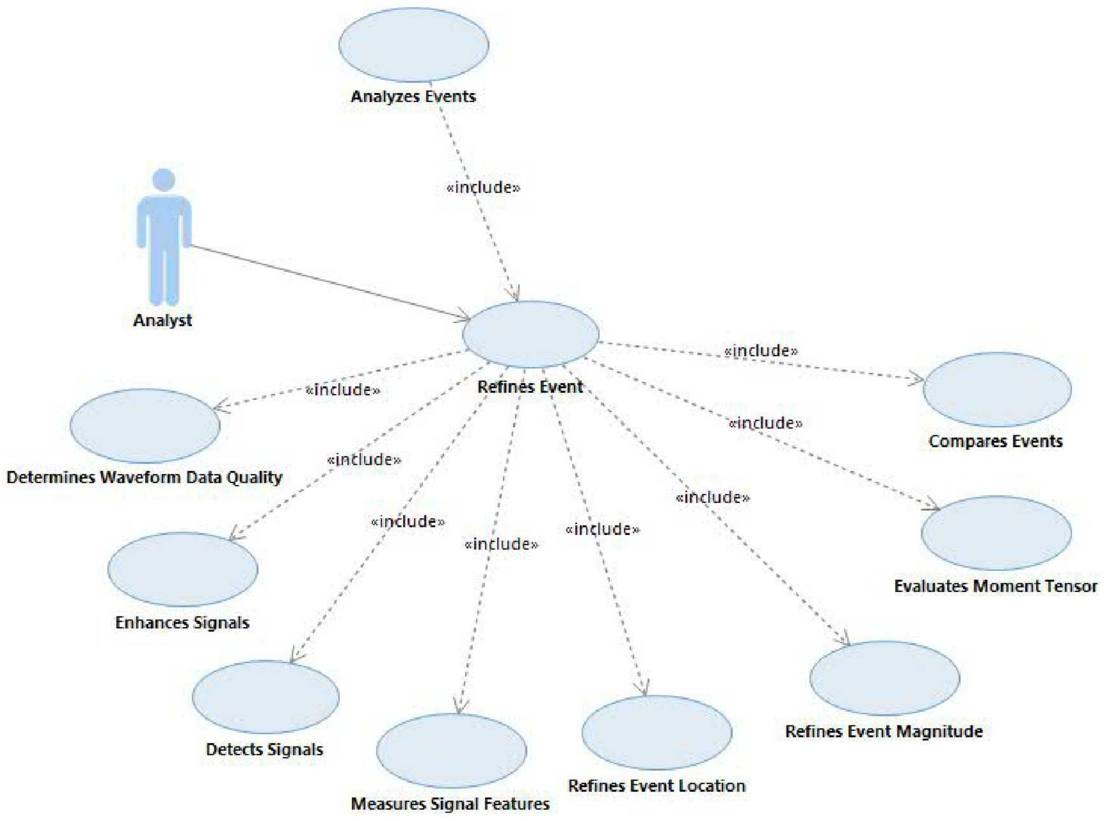
This architecturally significant use case describes how the Analyst refines an event hypothesis. The Analyst checks waveform quality (see 'Determines Waveform Data Quality' UC). For waveforms of sufficient quality, the Analyst enhances signals and suppresses noise on waveforms for relevant stations (see 'Enhances Signals' UC), adds and associates missing detections, and modifies or unassociates detections already associated with the event hypothesis (see 'Detects Signals' UC). The Analyst rejects event hypotheses that are invalid. For valid event hypotheses, the Analyst measures signal features associated with the detections (see 'Measures Signal Features' UC) and evaluates the moment tensor ('Evaluates Moment Tensor' UC). The Analyst uses these signal features to refine the location (see 'Refines Event Location' UC) and magnitude (see 'Refines Event Magnitude' UC) of the event hypothesis. The Analyst compares events to determine how similar events were constructed (see 'Compares Events' UC). The Analyst repeats these steps until satisfied with the results. Analysts may provide feedback for previous Analysts during any of these steps (see 'Provides Analyst Feedback' UC).

This use case is architecturally significant because it captures the interplay between all of the Analyst activities.

## ARCHITECTURE DESCRIPTION

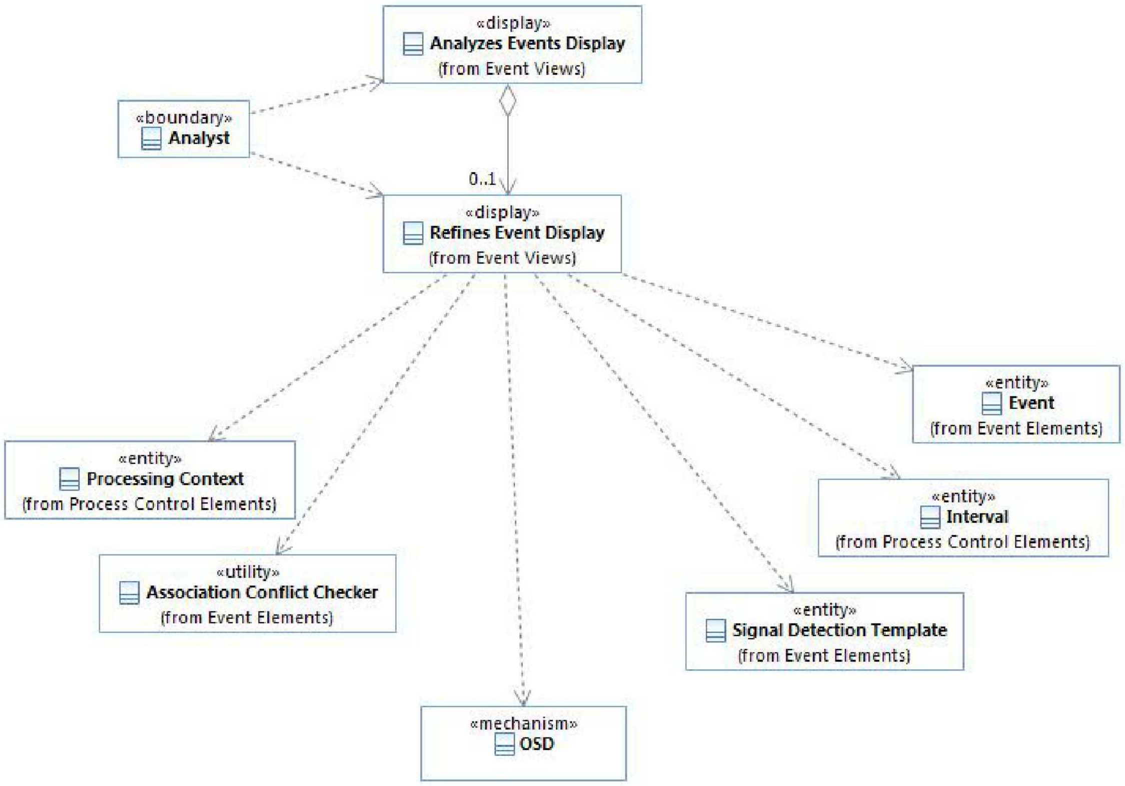
The Analyst refines an Event by selecting an Event on the Analyzes Events Display to open the Refines Event Display for the Event. The Refines Event Display retrieves the latest Event Hypothesis for the Event in the current processing stage (or the preferred Hypothesis from previous stage if no Hypothesis exists in the current stage) to use as a starting point, creates a local copy of it for the current processing stage, and provides the Analyst with the ability to refine it (depicted in included use cases). As the Analyst refines the Event the Event Hypothesis is updated and stored transiently in a private context via the OSD mechanism to make it available to the Processing Sequence Control mechanism for further automatic processing (the Processing Sequence Control mechanism is described in "System Detects Event" UCR). To save their changes, the Analyst selects to save the Event Hypothesis, which the display handles by storing the Event Hypothesis in a global context to persistent storage via the OSD.

## USE CASE DIAGRAM



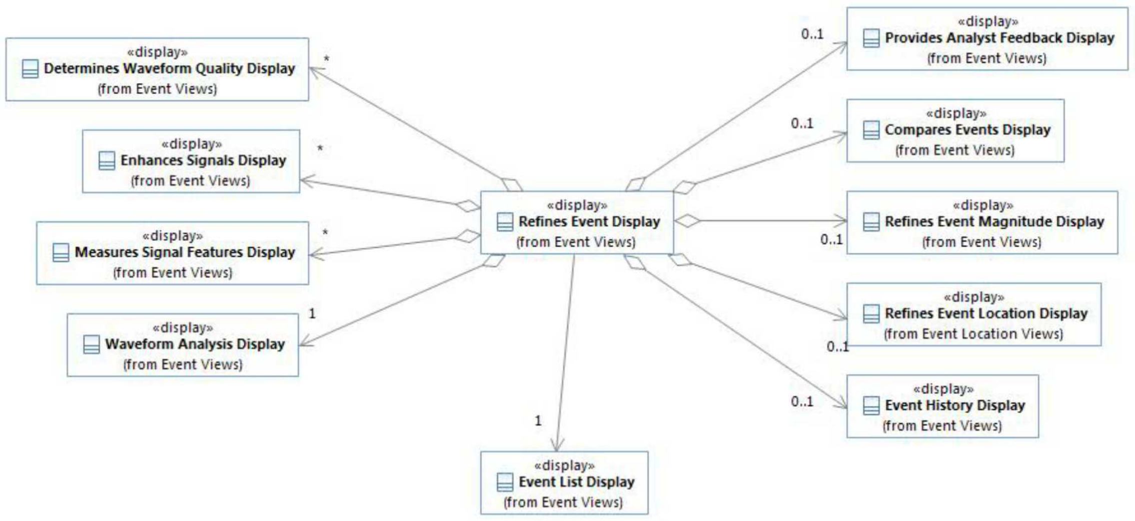
## CLASS DIAGRAMS

### Classes - Refines Event Display



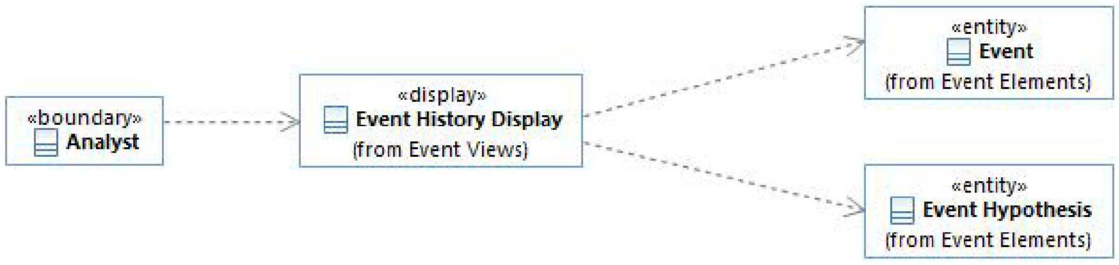
This diagram shows the Refines Event Display and related classes pertaining to this realization. The Analyst opens the Refines Event Display from the Analyzes Events Display. This display subscribes for the Event being refined in order to warn the Analyst if the Event is under active review by another Analyst in the same interval. The display subscribes for Intervals in order to warn the Analyst if the interval containing the Event is under active review by another Analyst (see "Scans Waveforms and Unassociated Detections" UCR). The Refines Event Display uses the Association Conflict Checker class to check for association conflicts with other Events whenever the current Event is modified or another Event in the interval is saved. The display also provides the Analyst with the ability to create Signal Detection Templates, which it stores in the OSD. Refines Event Display stores the refined Event in the OSD.

## Classes - Refines Event Display - Sub-displays



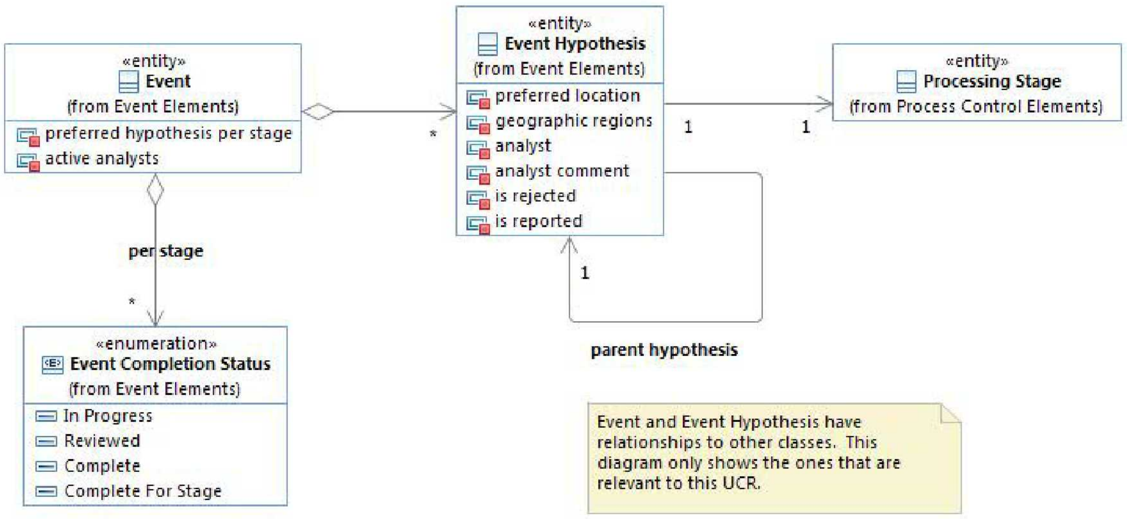
This class shows sub-displays of the Refines Event Display. The Refines Event Display creates and manages these sub-displays based on Analyst actions. Analyst interactions with these sub-displays are described in the corresponding UCRs; however, in general, the OSD mechanism is used to synchronize information between the displays. When the Analyst first opens the Event, the Refines Event Display creates a new Event Hypothesis and stores it in the OSD (in a private context, not visible to other Analysts). The Refines Event Display then subscribes for changes to this Event Hypothesis via the OSD. As the Analyst interacts with the various sub-displays, those Analyst actions may trigger processing on the privately stored Event Hypothesis; however, the sub-displays do not have knowledge of which processing steps will be performed since the processing sequences to be executed in response to Analyst actions are configurable (see "Defines Processing Sequence" UCR) and known only by the Processing Sequence Control mechanism. The Refines Event Display is informed of any processing performed on the Event Hypothesis via OSD callbacks.

## Classes - Event History Display



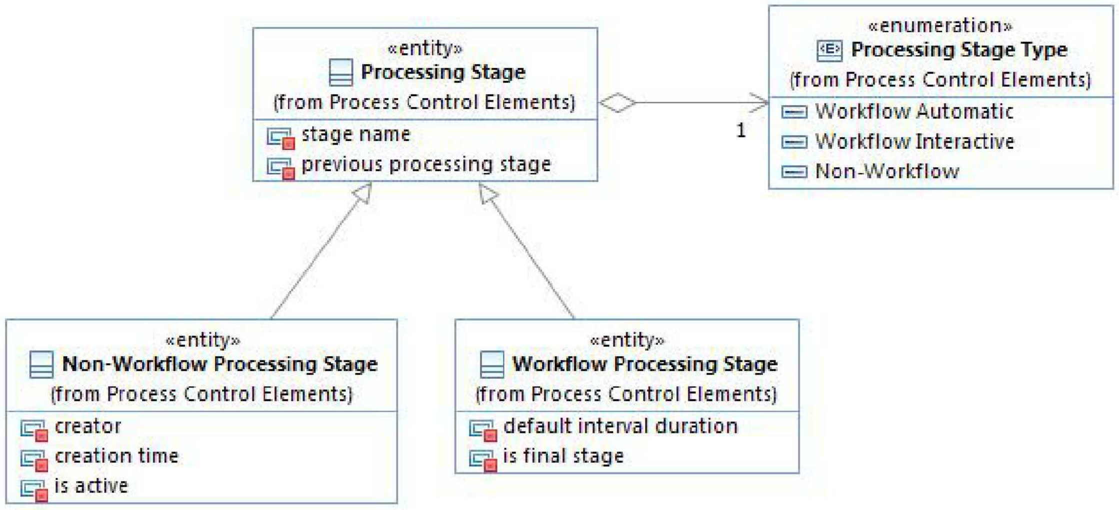
This diagram shows the Event History Display and related classes.

## Classes - Event



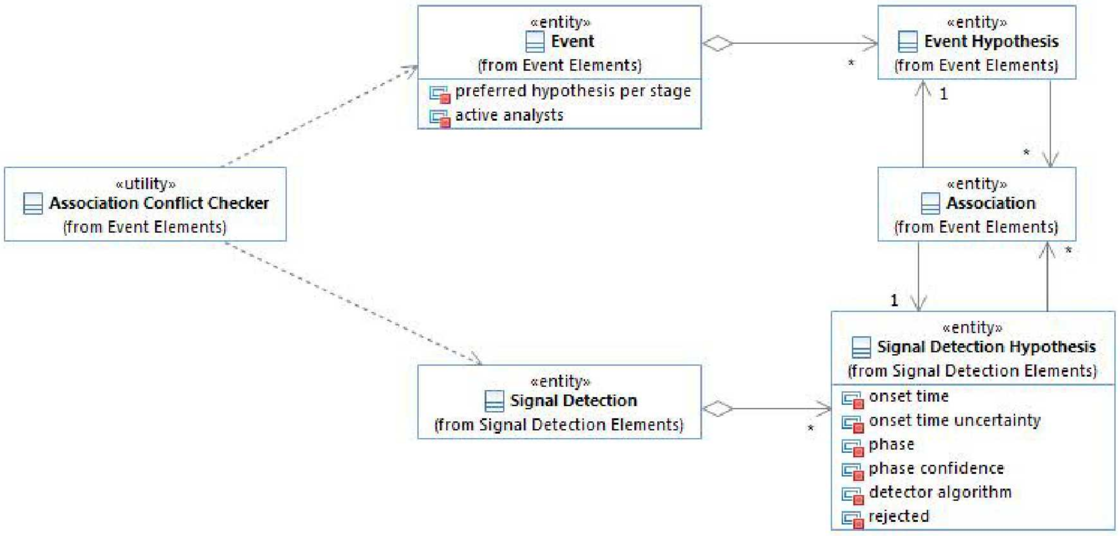
This diagram shows details of the Event and Event Hypothesis classes relevant to this UCR. Refinement of an Event results in a new Event Hypothesis. The Analyst potentially creates multiple Event Hypotheses for a given Event during a single processing stage, and designates one of them as the "preferred" Event Hypothesis for the Event for that stage (each stage can have a different preferred Hypothesis for the Event). Each Event also has an Event Completion Status, which reflects the Analyst's determination of the level of completeness of the Event within the stage. The Analyst specifies an Event Completion Status of "In Progress", "Reviewed" or "Complete" when saving the Event. The transition to "Complete For Stage" is covered in a separate UCR (see "Marks Processing Stage Complete" UCR).

## Classes - Processing Stages


«entity»
Processing Stage
(from Process Control Elements)
stage name
previous processing stage
1
«enumeration»
Processing Stage Type
(from Process Control Elements)
Workflow Automatic
Workflow Interactive
Non-Workflow
«entity»
Non-Workflow Processing Stage
(from Process Control Elements)
creator
creation time
is active
«entity»
Workflow Processing Stage
(from Process Control Elements)
default interval duration
is final stage

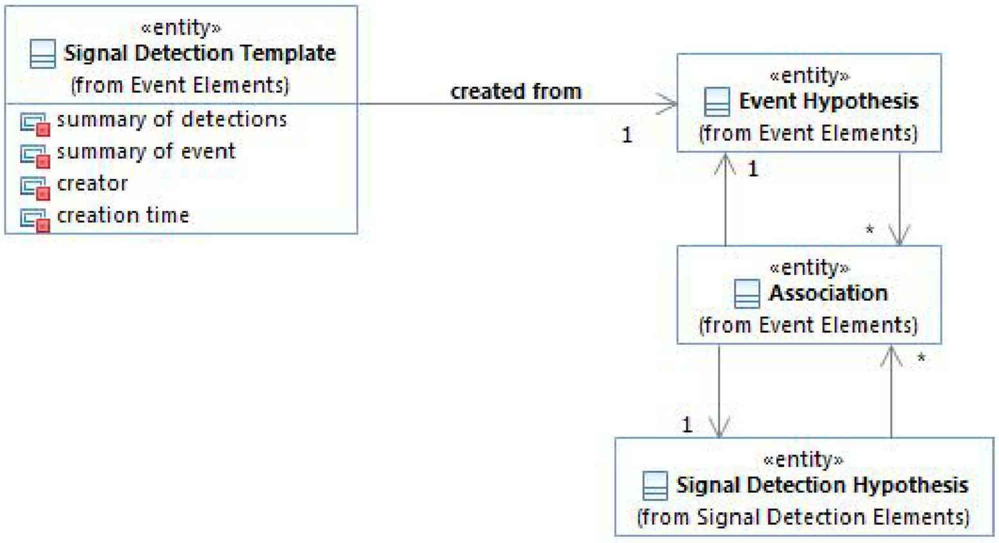
This diagram shows the Processing Stage class hierarchy and how Non-Workflow Processing Stages fit into it. Processing results (e.g. Event and Signal Detection Hypotheses) are linked to the Processing Stage in which they were created, which may be any of the three types of Processing Stage (Interactive, Automatic or Non-Workflow). Workflow Processing Stages are defined by the System Maintainer as part of the overall Processing Configuration (see "Defines Processing Sequence" UCR), whereas Non-Workflow Processing Stages are defined by individual Analysts (see "Alternate Flow - Analyst Defines Non-Workflow Processing Stage" in this UCR).

## Classes - Association Conflict Checker



This diagram shows details of the Association Conflict Checker class. The class retrieves Events, Associations, and Signal Detections from the OSD and checks for the case where more than one Event has a preferred Event Hypothesis for the stage that has a related Association to a Signal Detection Hypothesis for the same Signal Detection.

## Classes - Signal Detection Template



This diagram shows details for the Signal Detection Template class. The Analyst may create a Signal Detection Template via the Refines Event Display based on the Signal Detection Hypotheses associated to the current Event Hypothesis.

## CLASS DESCRIPTIONS

***<<boundary>> Analyst***
Represents the Analyst actor.

***<<display>> Analyzes Events Display***
Display that provides the Analyst with the ability to analyze data within a specified time interval in order to find or refine Events.

***<<display>> Compares Events Display***
Display that provides the Analyst with the ability to compare Events.

***<<display>> Refines Event Display***
Display that provides the Analyst with the ability to refine an Event. Each saved refinement of the Event results in a new Event Hypothesis.

***<<display>> Refines Event Location Display***
Provides the Analyst with the ability to enter Event location parameters and initiate computation of a location for an Event Hypothesis.

***<<display>> Waveform Analysis Display***
Displays a set of waveforms and provides the Analyst with the ability to interact with them (e.g. create/modify/reject Signal Detections, associate/unassociate detections and Events).

***<<entity>> Association***
Represents an association between a Signal Detection Hypothesis and an Event Hypothesis.

***<<entity>> Event***
Represents information about an Event. Keeps track of all the Event Hypotheses for the Event, which Event Hypothesis is the preferred one for each processing stage, the active analysts for the Event (i.e. whether the Event is under "active review"), whether the Event is "complete" for each processing stage, and other Event-related information.

***<<entity>> Event Hypothesis***
Represents geophysical information about an Event as determined by an Analyst or through pipeline processing. There can be multiple Event Hypotheses for the same Event (e.g. different associated Signal Detection Hypotheses, different location solutions).

***<<entity>> Interval***
Class for tracking the status of interactive or automatic processing on a specific timeframe of data. Specialized intervals exist for Processing Stage, Processing Activity, and Processing Sequence.

***<<entity>> Non-Workflow Processing Stage***

Represents a Processing Stage that is not part of the System Maintainer-defined Analyst workflow. Analysts may define such stages at any time to store their Processing Results, but the stored results in such stages are not used by Analysts or the system during workflow processing.

***<<entity>> Processing Context***
Represents the context in which data is being stored and/or processed. This includes the processing session (e.g. processed by Analyst vs. processed by System). For Analyst processing, may identify the Analyst work session. For System processing, may identify the Processing Sequence and/or Processing Step being executed (including a way to identify a particular Processing Sequence and Processing Step among the many possible instantiations), the visibility for the results (private vs. global), and the lifespan of the data (transient vs. persistent). This information is needed by the Processing Sequence Control to manage the execution of Processing Sequences, which may execute in the context of an Analyst refining an Event or in the context of the system initiating automatic processing. It is also needed by the Object Storage and Distribution (OSD) mechanism to determine how to store and distribute the data.

***<<entity>> Processing Stage***
Represents a named stage of data processing, which may be part of the System Maintainer-defined workflow or an Analyst-defined stage outside the workflow. All Processing Results are associated to a Processing Stage. The previous processing stage indicates the stage to be used as the default starting point when creating new processing results in the stage (e.g. when refining an event in the stage).

***<<entity>> Signal Detection***
Represents information about a Signal Detection and keeps track of all the Signal Detection Hypotheses for the Signal Detection. Represents information about a Signal Detection and keeps track of all the Signal Detection Hypotheses for the Signal Detection. For an unassociated Signal Detection the preferred Signal Detection Hypothesis is the most recently created Signal Detection Hypothesis. For an associated Signal Detection the preferred Signal Detection Hypothesis is the one associated to a preferred Event Hypothesis.

***<<entity>> Signal Detection Hypothesis***
Represents geophysical information about a Signal Detection as determined by an Analyst or through pipeline processing. There can be multiple Signal Detection Hypotheses for the same Signal Detection (e.g. different onset times, different phase labels).

***<<entity>> Signal Detection Template***
A template that represents the pattern of Signal Detections for an Event (i.e. channels detected, relative positions for each detection, phases, etc.). An Analyst may apply the template to quickly build new Events that match the pattern of detections. Signal Detection Associator Plugins (see "System Builds Events using Signal Detections" UCR) may also use Signal Detection Templates to build new Event Hypotheses and associate additional detections to Event Hypotheses matching the template. Also includes summary information about the original Event from which the template was created (e.g. Event location, magnitude, etc.), as an aid to the Analyst in finding and applying a relevant template.

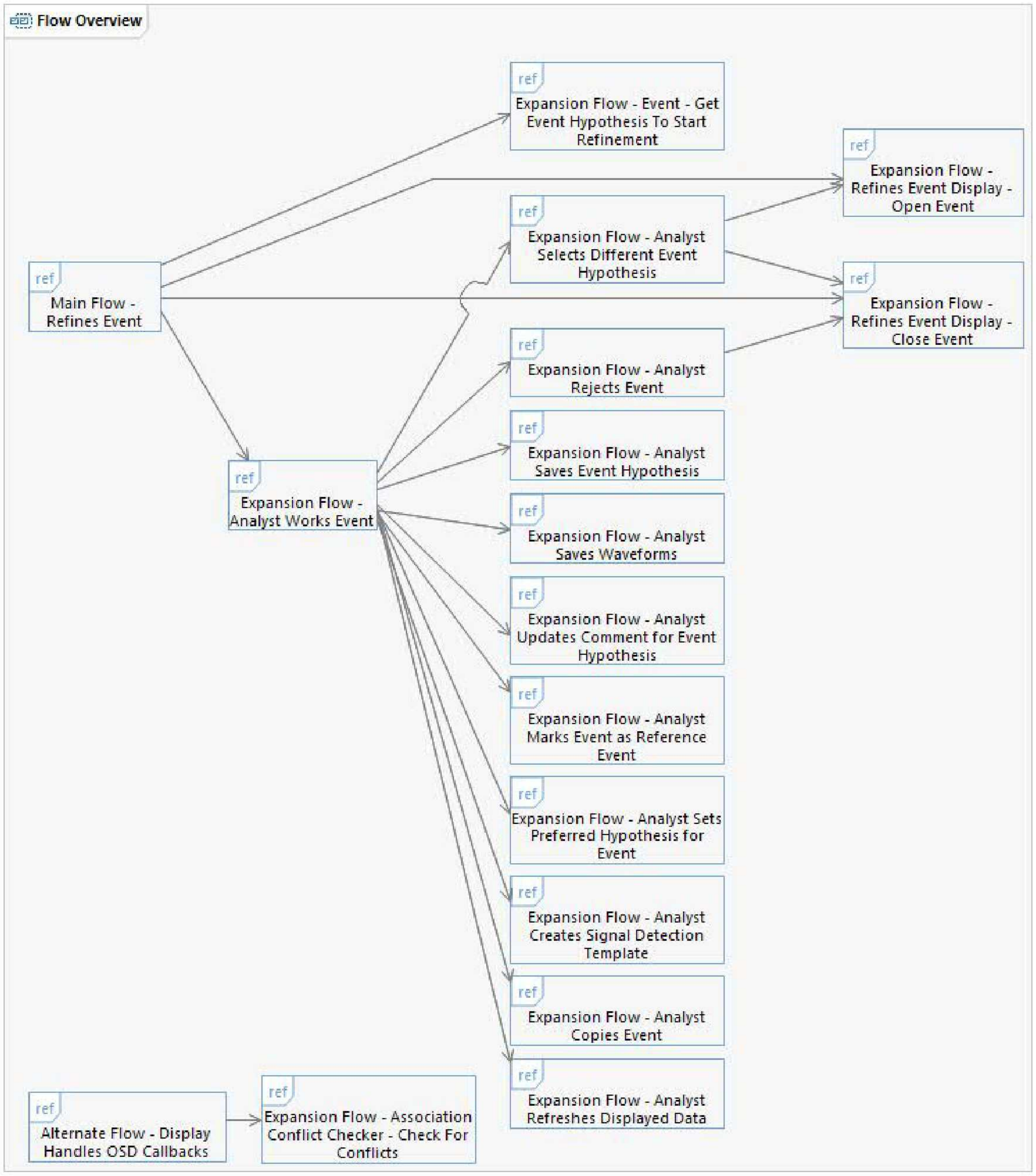
***<<entity>> Workflow Processing Stage***
Represents a Processing Stage that is part of the System Maintainer-defined Analyst workflow.

***<<mechanism>> OSD***
Represents the Object Storage and Distribution mechanism for storing and distributing data objects internally within the system.

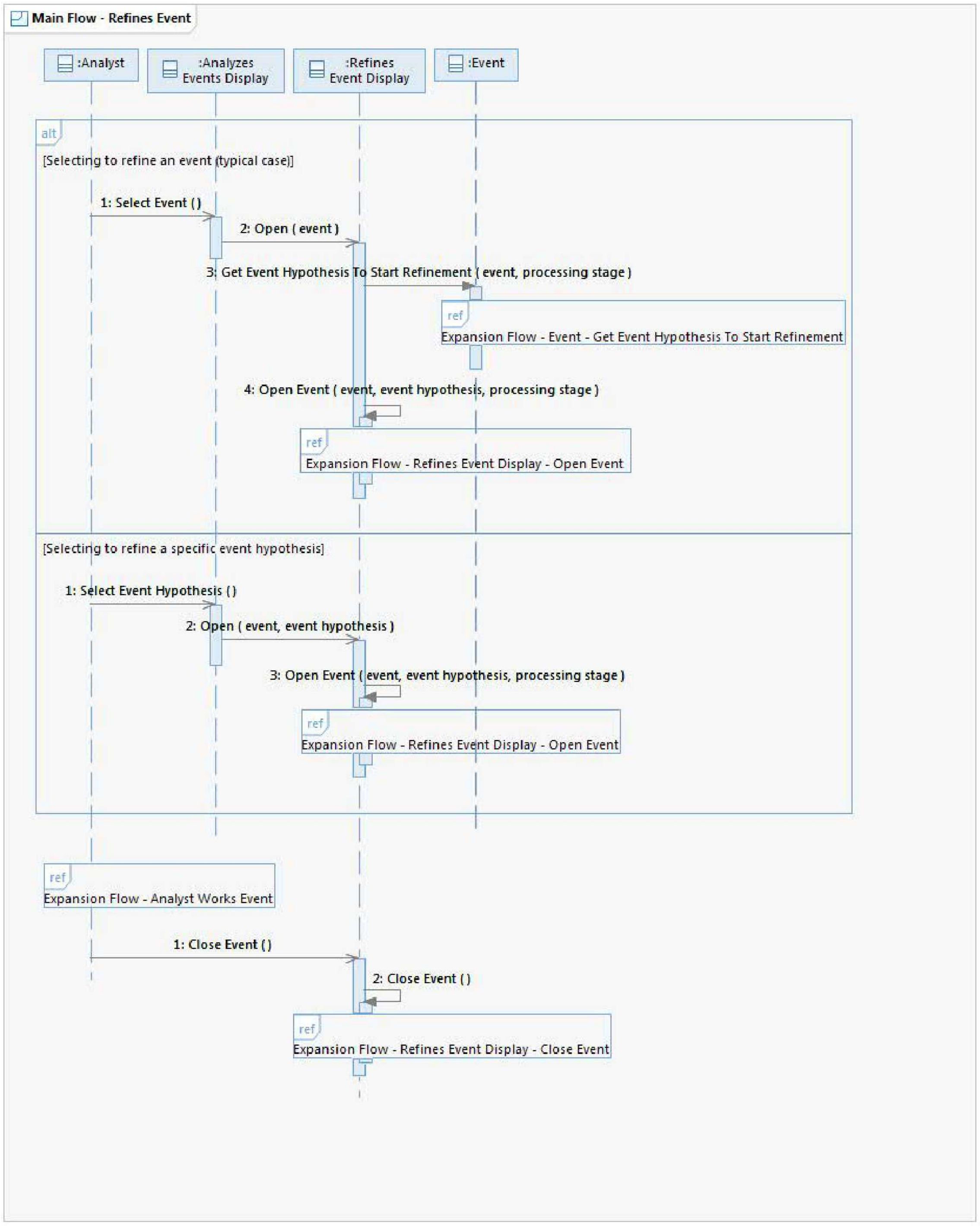
***<<utility>> Association Conflict Checker***
Utility class for checking that preferred Event Hypotheses within a processing stage do not share any Signal Detections.

## SEQUENCE DIAGRAMS

### Flow Overview


Flow Overview
ref Expansion Flow - Event - Get Event Hypothesis To Start Refinement
ref Expansion Flow - Refines Event Display - Open Event
ref Expansion Flow - Analyst Selects Different Event Hypothesis
ref Main Flow - Refines Event
ref Expansion Flow - Refines Event Display - Close Event
ref Expansion Flow - Analyst Rejects Event
ref Expansion Flow - Analyst Saves Event Hypothesis
ref Expansion Flow - Analyst Works Event
ref Expansion Flow - Analyst Saves Waveforms
ref Expansion Flow - Analyst Updates Comment for Event Hypothesis
ref Expansion Flow - Analyst Marks Event as Reference Event
ref Expansion Flow - Analyst Sets Preferred Hypothesis for Event
ref Expansion Flow - Analyst Creates Signal Detection Template
ref Expansion Flow - Analyst Copies Event
ref Expansion Flow - Analyst Refreshes Displayed Data
ref Alternate Flow - Display Handles OSD Callbacks
ref Expansion Flow - Association Conflict Checker - Check For Conflicts

## Main Flow - Refines Event


Main Flow - Refines Event
:Analyst
:Analyzes Events Display
:Refines Event Display
:Event
alt
[Selecting to refine an event (typical case)]
1: Select Event ()
2: Open ( event )
3: Get Event Hypothesis To Start Refinement ( event, processing stage )
ref
Expansion Flow - Event - Get Event Hypothesis To Start Refinement
4: Open Event ( event, event hypothesis, processing stage )
ref
Expansion Flow - Refines Event Display - Open Event
[Selecting to refine a specific event hypothesis]
1: Select Event Hypothesis ()
2: Open ( event, event hypothesis )
3: Open Event ( event, event hypothesis, processing stage )
ref
Expansion Flow - Refines Event Display - Open Event
ref
Expansion Flow - Analyst Works Event
1: Close Event ()
2: Close Event ()
ref
Expansion Flow - Refines Event Display - Close Event

This flow shows the main flow for refining an Event. The Refines Event Display is typically opened with an Event, in which case the display automatically determines which Event Hypothesis to use as a starting point (the Analyst can select a different Hypothesis as a starting point - see "Expansion Flow - Analyst Selects Different Event Hypothesis"). The Analyst may also open the Refines Event Display with a specific Event Hypothesis to refine.  In this case the display uses that Hypothesis as the starting point.

### *Operation Descriptions*

***Operation: Refines Event Display::Open Event()***

Open the given Event for refinement in the current processing stage, using the given Event Hypothesis as a starting point.

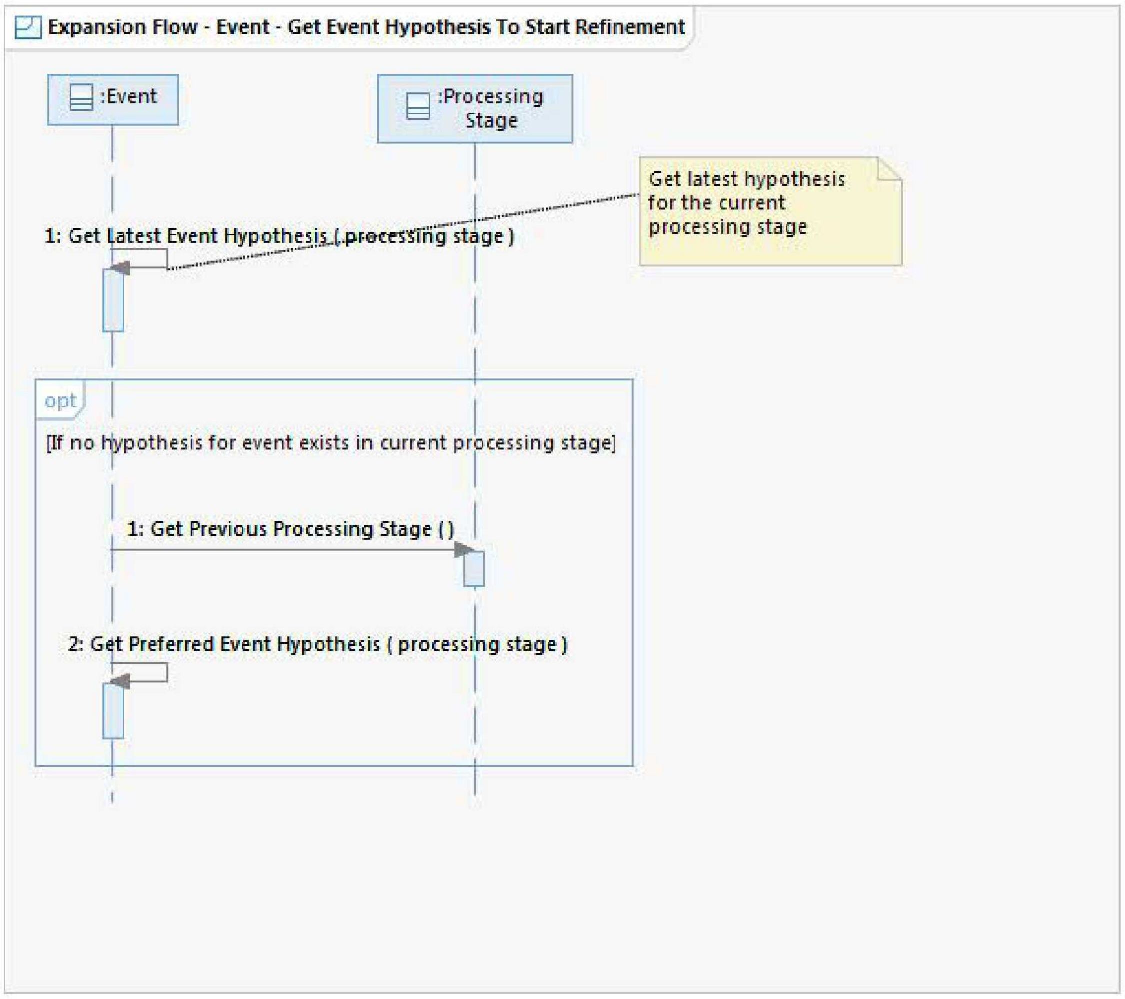
***Operation: Event::Get Event Hypothesis To Start Refinement()***

Return an Event Hypothesis to use as a starting point for refining the Event in the given processing stage.  If the processing stage is an Interactive Processing Stage, look in the previous stage for a preferred Hypothesis in that stage.  If none exists, continue the search in next previous stage, etc. until a preferred Hypothesis is found.  Note that the stage order is defined in the Processing Configuration class, which is defined by the System Maintainer.

***Operation: Refines Event Display::Open Event()***

Open the given Event for refinement in the current processing stage, using the given Event Hypothesis as a starting point.

## Expansion Flow - Event - Get Event Hypothesis To Start Refinement

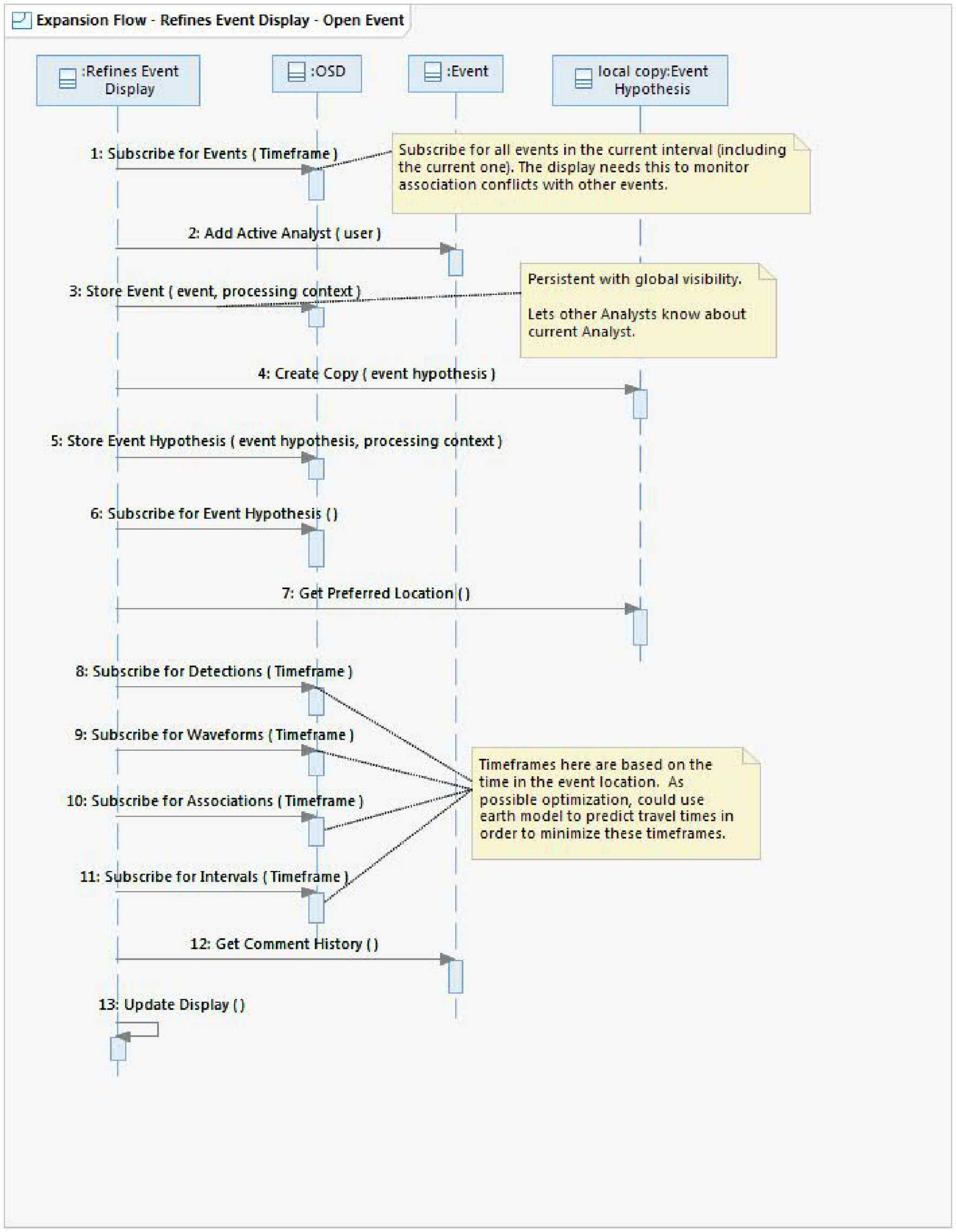

Expansion Flow - Event - Get Event Hypothesis To Start Refinement
:Event
:Processing Stage
Get latest hypothesis for the current processing stage
1: Get Latest Event Hypothesis ( processing stage )
opt
[If no hypothesis for event exists in current processing stage]
1: Get Previous Processing Stage ( )
2: Get Preferred Event Hypothesis ( processing stage )

This flow shows how the Event class finds an Event Hypothesis to use as a starting point for refinement in the current stage, which may be a workflow or non-workflow stage. Note that, in the case of a workflow stage the "previous processing stage" is configured by the System Maintainer (see "Defines Processing Sequence" UCR) whereas in the case of a non-workflow stage the "previous processing stage" is set by the Analyst when they define the stage (see "Selects Data for Analysis" UCR).

*Operation Descriptions*

None

## Expansion Flow - Refines Event Display - Open Event


Expansion Flow - Refines Event Display - Open Event
:Refines Event Display
:OSD
:Event
local copy:Event Hypothesis
1: Subscribe for Events ( Timeframe )
Subscribe for all events in the current interval (including the current one). The display needs this to monitor association conflicts with other events.
2: Add Active Analyst ( user )
3: Store Event ( event, processing context )
Persistent with global visibility.
Lets other Analysts know about current Analyst.
4: Create Copy ( event hypothesis )
5: Store Event Hypothesis ( event hypothesis, processing context )
6: Subscribe for Event Hypothesis ( )
7: Get Preferred Location ( )
8: Subscribe for Detections ( Timeframe )
9: Subscribe for Waveforms ( Timeframe )
10: Subscribe for Associations ( Timeframe )
11: Subscribe for Intervals ( Timeframe )
Timeframes here are based on the time in the event location. As possible optimization, could use earth model to predict travel times in order to minimize these timeframes.
12: Get Comment History ( )
13: Update Display ( )

This flow shows how the Refines Event Display opens an Event for refinement. The Event Hypothesis to use as a starting point is an input to this flow. The system keeps track of all the analysts that are working an Event ("active analysts") and warns if the Event is under active review by another analyst or overlaps an interval that is under active review by another analyst (see "Alternate Flow - Display Handles OSD Callbacks"). The display creates a new Event Hypothesis instance based on the passed-in Event Hypothesis. The display subscribes for detections and waveforms around the Event in order to display them. The comment history is also displayed.

### *Operation Descriptions*

***Operation: OSD::Subscribe for Events()***
Subscribe for changes to Event objects within the given timeframe. Callbacks are invoked on subscribers any time an Event within the timeframe is added or modified.

***Operation: OSD::Subscribe for Detections()***
Subscribe for updates regarding Signal Detection creations, modifications, and associations occurring within the specified timeframe. This includes updates for new or modified unassociated Signal Detections.

***Operation: OSD::Subscribe for Waveforms()***
Subscribe for updates regarding raw and derived waveforms occurring within a specified timeframe. This includes information about what waveforms have been acquired by the System as well as what derived waveforms have been formed, but does not include the actual waveform data.

***Operation: Event::Add Active Analyst()***
Add the given Analyst to the set of active Analysts for the Event. If the Event has active analysts it is said to be under "active review".

***Operation: OSD::Store Event()***
Store the given Event with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

***Operation: Event Hypothesis::Create Copy()***
Create a copy of the given Event Hypothesis. The copy has all of the same information as the original (e.g. same detections, location, etc.), with the following exceptions:

- The copy points to the original as its parent
- The copy starts out with an empty Analyst comment

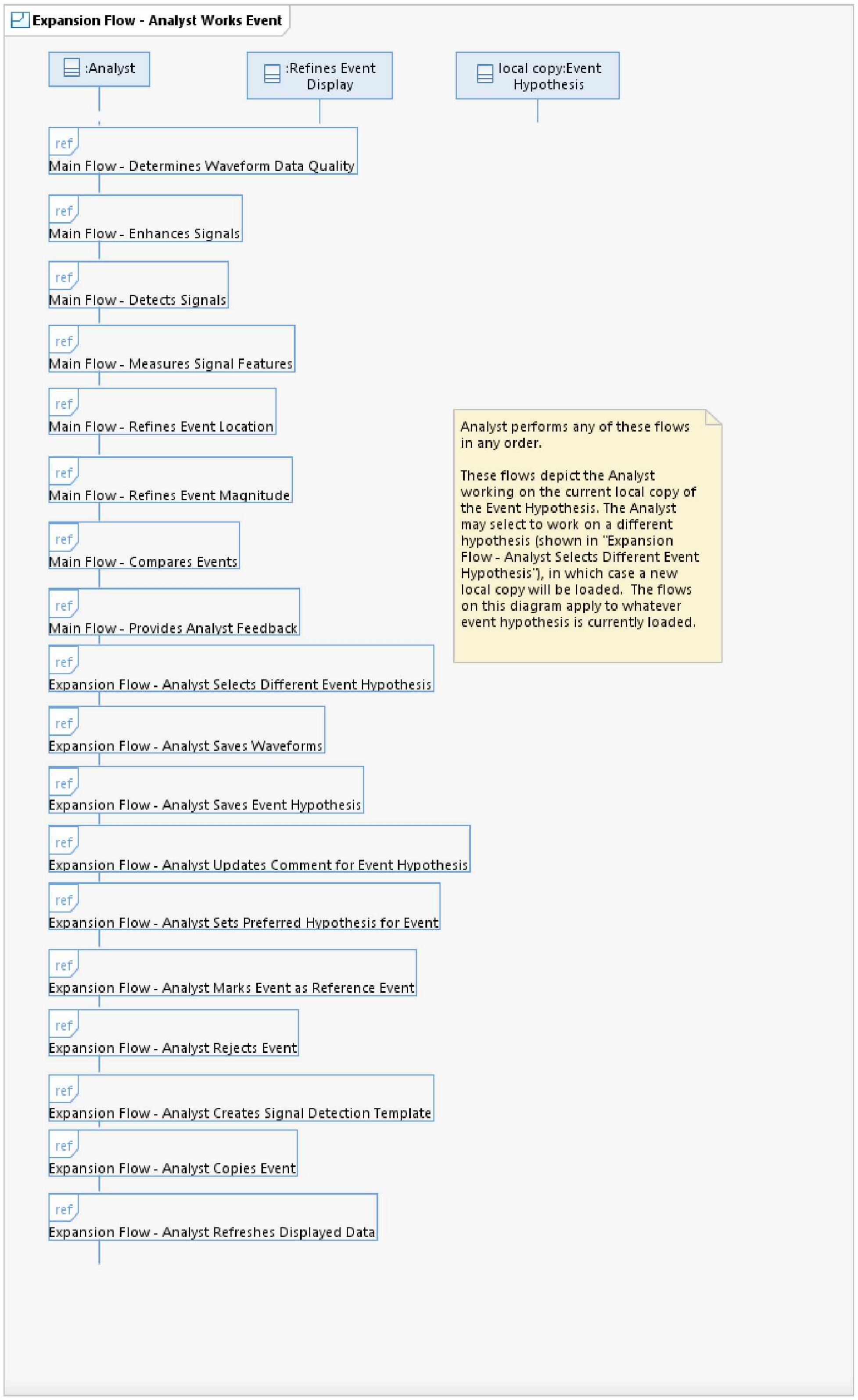
***Operation: OSD::Store Event Hypothesis()***
Store the given Event Hypothesis with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

***Operation: Refines Event Display::Update Display()***
Update the display of the Event Hypothesis being refined to reflect any changes that occurred. Indicate items that are out-of-date or inconsistent (e.g. beam may be out-of-date after refining Event location).

***Operation: Event::Get Comment History()***
Return all Analyst-entered comments associated with the Event.

***Operation: OSD::Subscribe for Intervals()***
Subscribe for changes to Interval objects that overlap with the given timeframe. Interval objects track the active analysts and completion status of intervals corresponding to processing stages and processing activities within processing stages. Callbacks are invoked on subscribers any time the set of active analysts or completion status for an Interval changes.

## Expansion Flow - Analyst Works Event


Expansion Flow - Analyst Works Event
:Analyst
:Refines Event Display
local copy:Event Hypothesis
ref
Main Flow - Determines Waveform Data Quality
ref
Main Flow - Enhances Signals
ref
Main Flow - Detects Signals
ref
Main Flow - Measures Signal Features
ref
Main Flow - Refines Event Location
ref
Main Flow - Refines Event Magnitude
ref
Main Flow - Compares Events
ref
Main Flow - Provides Analyst Feedback
ref
Expansion Flow - Analyst Selects Different Event Hypothesis
ref
Expansion Flow - Analyst Saves Waveforms
ref
Expansion Flow - Analyst Saves Event Hypothesis
ref
Expansion Flow - Analyst Updates Comment for Event Hypothesis
ref
Expansion Flow - Analyst Sets Preferred Hypothesis for Event
ref
Expansion Flow - Analyst Marks Event as Reference Event
ref
Expansion Flow - Analyst Rejects Event
ref
Expansion Flow - Analyst Creates Signal Detection Template
ref
Expansion Flow - Analyst Copies Event
ref
Expansion Flow - Analyst Refreshes Displayed Data

Analyst performs any of these flows in any order.

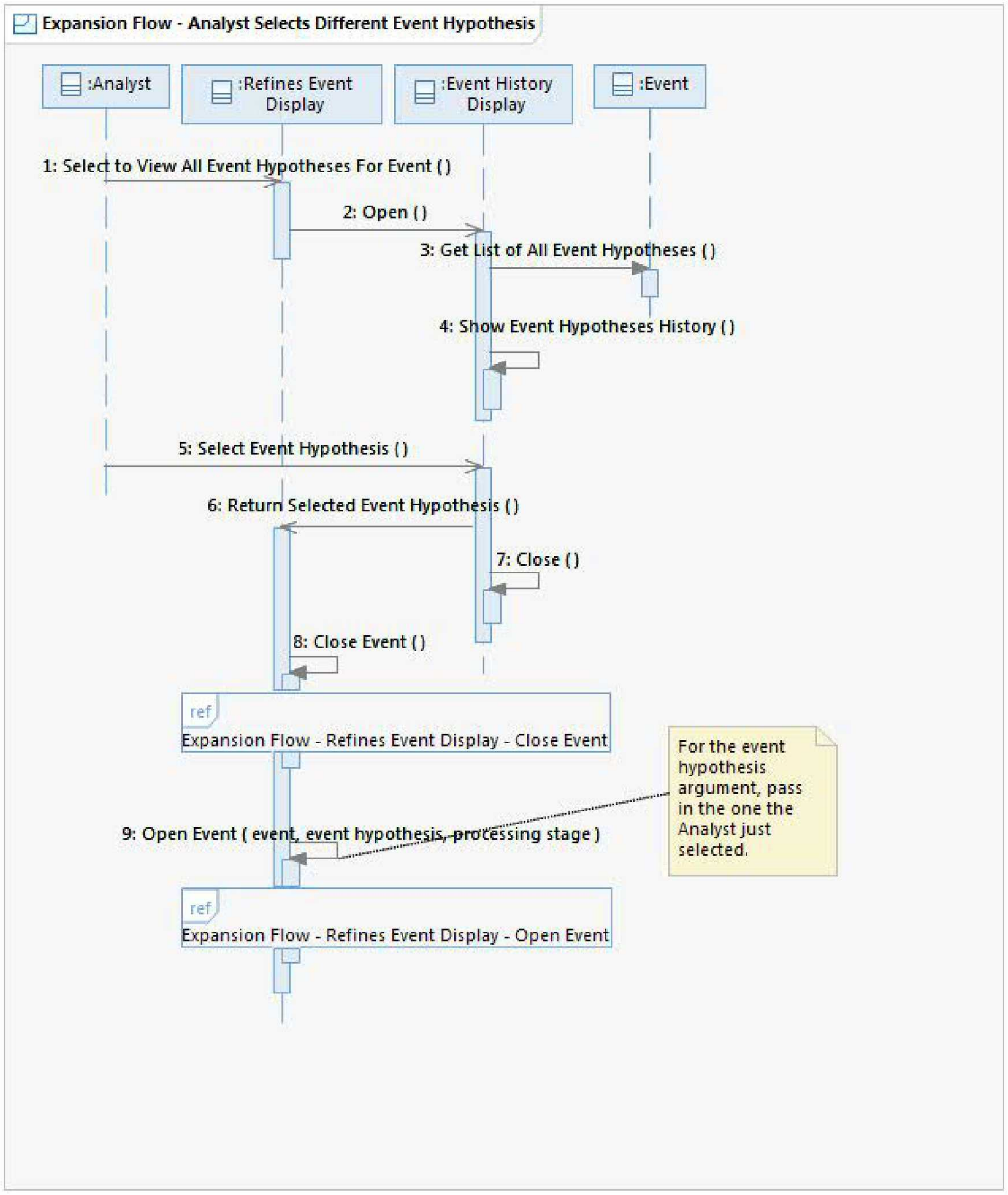
These flows depict the Analyst working on the current local copy of the Event Hypothesis. The Analyst may select to work on a different hypothesis (shown in "Expansion Flow - Analyst Selects Different Event Hypothesis"), in which case a new local copy will be loaded. The flows on this diagram apply to whatever event hypothesis is currently loaded.

This flow shows the actions an Analyst can perform as part of refining an Event.

*Operation Descriptions*

None

## Expansion Flow - Analyst Selects Different Event Hypothesis


Expansion Flow - Analyst Selects Different Event Hypothesis
:Analyst
:Refines Event Display
:Event History Display
:Event
1: Select to View All Event Hypotheses For Event ( )
2: Open ( )
3: Get List of All Event Hypotheses ( )
4: Show Event Hypotheses History ( )
5: Select Event Hypothesis ( )
6: Return Selected Event Hypothesis ( )
7: Close ( )
8: Close Event ( )
ref
Expansion Flow - Refines Event Display - Close Event
9: Open Event ( event, event hypothesis, processing stage )
For the event hypothesis argument, pass in the one the Analyst just selected.
ref
Expansion Flow - Refines Event Display - Open Event

This flow shows the Analyst selecting to refine a different Event Hypothesis for the Event (other than the current one).

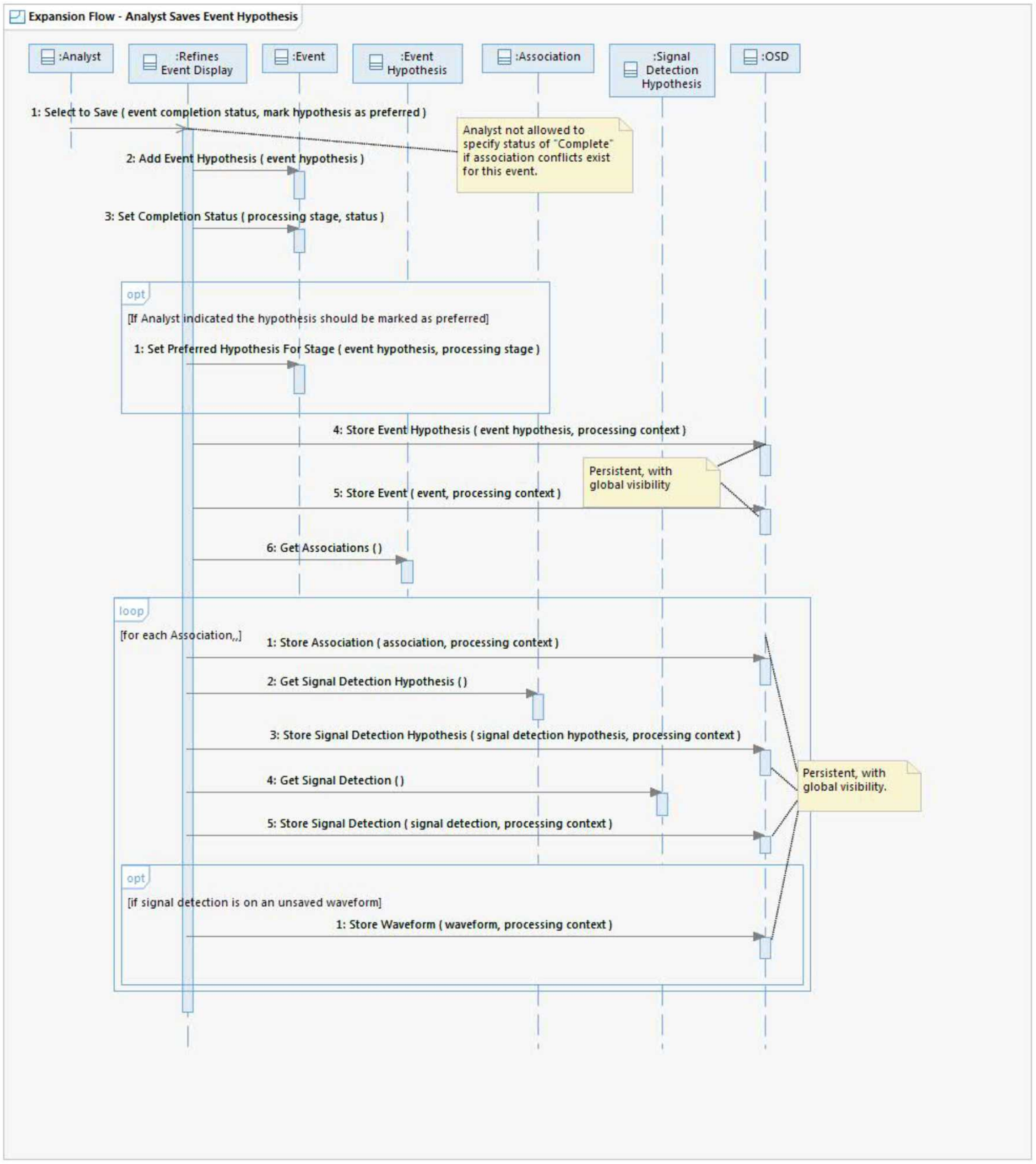
*Operation Descriptions*

***Operation: Refines Event Display::Open Event()***

Open the given Event for refinement in the current processing stage, using the given Event Hypothesis as a starting point.

***Operation: Event::Get List of All Event Hypotheses()***
Return a list of all the Event Hypotheses for the given Event, including summary information such as the processing stage and which Event Hypotheses have been designated as preferred.

## Expansion Flow - Analyst Saves Event Hypothesis


Expansion Flow - Analyst Saves Event Hypothesis
:Analyst
:Refines Event Display
:Event
:Event Hypothesis
:Association
:Signal Detection Hypothesis
:OSD
1: Select to Save ( event completion status, mark hypothesis as preferred )
Analyst not allowed to specify status of "Complete" if association conflicts exist for this event.
2: Add Event Hypothesis ( event hypothesis )
3: Set Completion Status ( processing stage, status )
opt
[If Analyst indicated the hypothesis should be marked as preferred]
1: Set Preferred Hypothesis For Stage ( event hypothesis, processing stage )
4: Store Event Hypothesis ( event hypothesis, processing context )
Persistent, with global visibility
5: Store Event ( event, processing context )
6: Get Associations ( )
loop
[for each Association,,]
1: Store Association ( association, processing context )
2: Get Signal Detection Hypothesis ( )
3: Store Signal Detection Hypothesis ( signal detection hypothesis, processing context )
4: Get Signal Detection ( )
5: Store Signal Detection ( signal detection, processing context )
Persistent, with global visibility.
opt
[if signal detection is on an unsaved waveform]
1: Store Waveform ( waveform, processing context )

This flow shows how the Analyst saves the Event Hypothesis they are refining to persistent storage and makes it visible to other Analysts. Once the Event Hypothesis is saved to persistent storage it can never be modified again (but the Analyst can create a new Event Hypothesis to further refine the Event). When saving the Event, the Analyst specifies the completion status for the Event (see Event Completion Status class on diagram "Classes - Event") and whether to mark the Hypothesis as the preferred one for the Event.

*Operation Descriptions*

***Operation: OSD::Store Event Hypothesis()***

Store the given Event Hypothesis with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

***Operation: OSD::Store Event()***

Store the given Event with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

***Operation: Event::Set Completion Status()***

Set the Event Completion Status of an Event in the given processing stage to the given value.

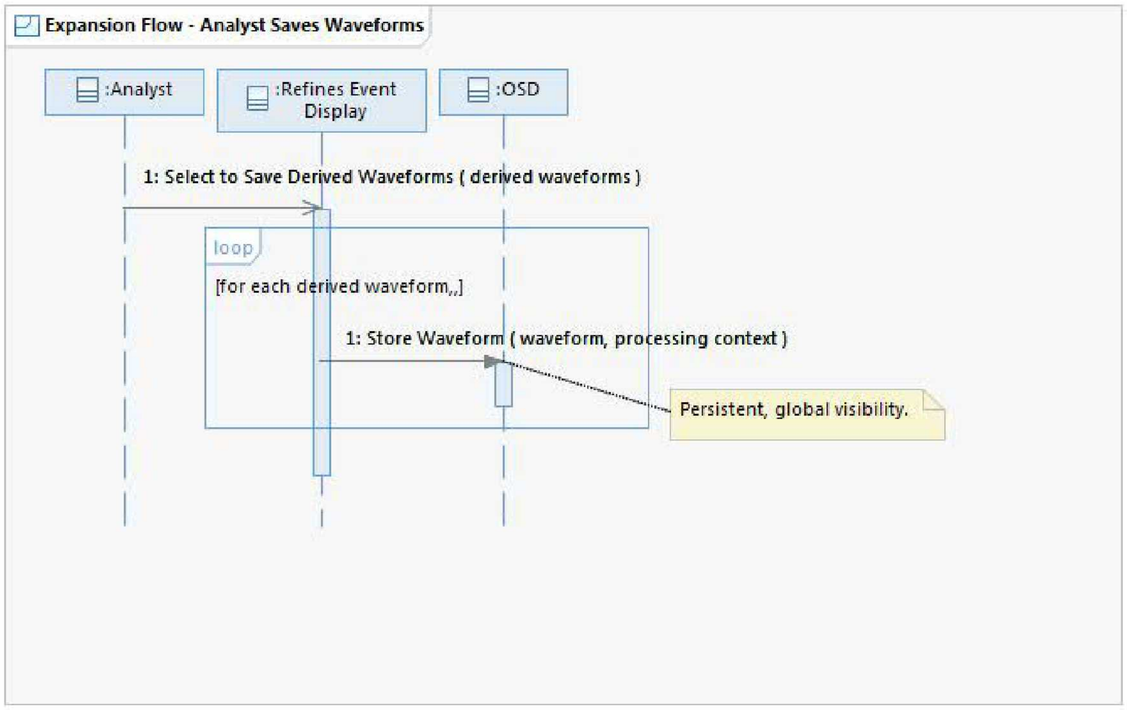
***Operation: OSD::Store Waveform()***

Store the given Waveform with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

***Operation: OSD::Store Signal Detection()***

Store the given Signal Detection with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

## Expansion Flow - Analyst Saves Waveforms


Expansion Flow - Analyst Saves Waveforms
:Analyst
:Refines Event Display
:OSD
1: Select to Save Derived Waveforms ( derived waveforms )
loop
[for each derived waveform,,]
1: Store Waveform ( waveform, processing context )
Persistent, global visibility.

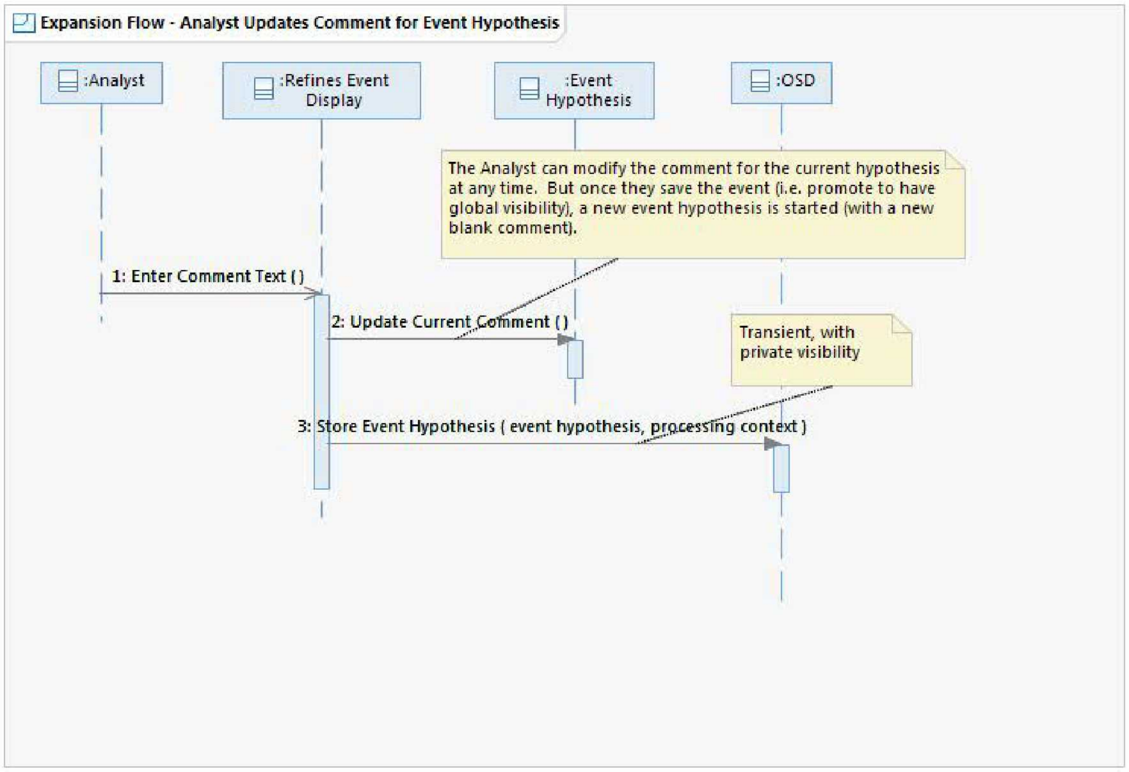
This flow shows Refines Event Display saving derived waveforms on Analyst request. The Analyst may select to save derived waveforms even when there are no signals detected on the waveforms.

### *Operation Descriptions*

***Operation: OSD::Store Waveform()***

Store the given Waveform with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

## Expansion Flow - Analyst Updates Comment for Event Hypothesis


Expansion Flow - Analyst Updates Comment for Event Hypothesis
:Analyst
:Refines Event Display
:Event Hypothesis
:OSD
The Analyst can modify the comment for the current hypothesis at any time. But once they save the event (i.e. promote to have global visibility), a new event hypothesis is started (with a new blank comment).
1: Enter Comment Text ( )
2: Update Current Comment ( )
Transient, with private visibility
3: Store Event Hypothesis ( event hypothesis, processing context )

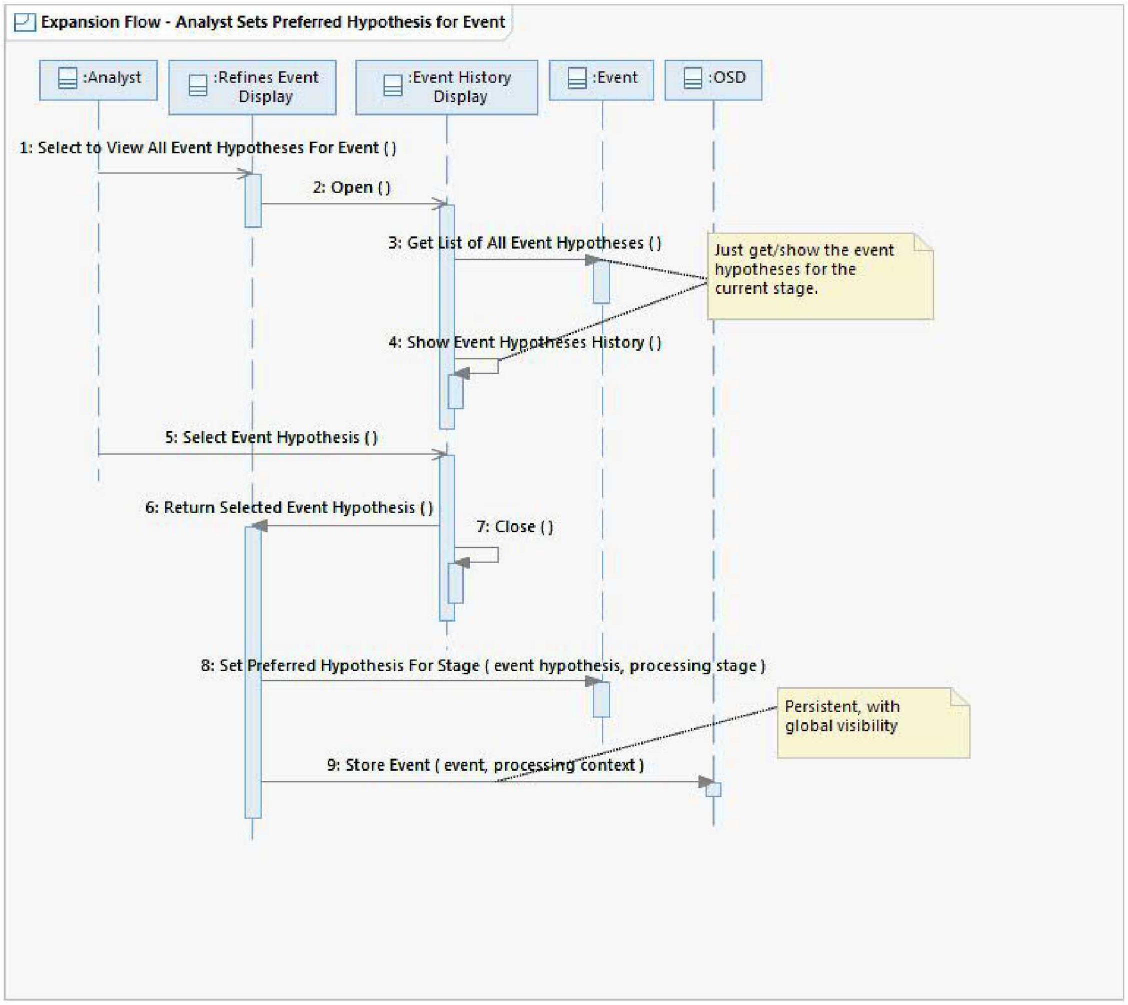
This flow shows how the Analyst updates comments on the Event Hypothesis.

*Operation Descriptions*

***Operation: OSD::Store Event Hypothesis()***

Store the given Event Hypothesis with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

## Expansion Flow - Analyst Sets Preferred Hypothesis for Event


Expansion Flow - Analyst Sets Preferred Hypothesis for Event
:Analyst
:Refines Event Display
:Event History Display
:Event
:OSD
1: Select to View All Event Hypotheses For Event ()
2: Open ()
3: Get List of All Event Hypotheses ()
Just get/show the event hypotheses for the current stage.
4: Show Event Hypotheses History ()
5: Select Event Hypothesis ()
6: Return Selected Event Hypothesis ()
7: Close ()
8: Set Preferred Hypothesis For Stage ( event hypothesis, processing stage )
Persistent, with global visibility
9: Store Event ( event, processing context )

This flow shows the Analyst designating an Event Hypothesis as preferred for a specified processing stage. Marking an Event Hypothesis as preferred causes the Event to be immediately stored in a global context (visible to other analysts), but does not change its Event Completion Status.

*Operation Descriptions*

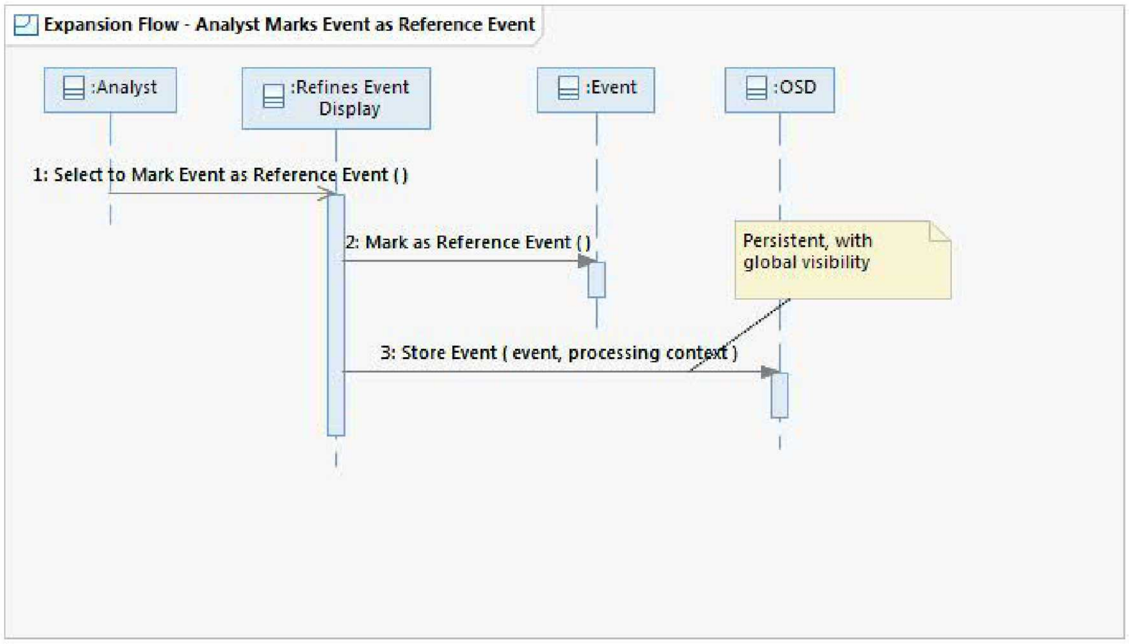
***Operation: OSD::Store Event()***

Store the given Event with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

***Operation: Event::Get List of All Event Hypotheses()***

Return a list of all the Event Hypotheses for the given Event, including summary information such as the processing stage and which Event Hypotheses have been designated as preferred.

## Expansion Flow - Analyst Marks Event as Reference Event


Expansion Flow - Analyst Marks Event as Reference Event
:Analyst
:Refines Event Display
:Event
:OSD
1: Select to Mark Event as Reference Event ()
2: Mark as Reference Event ()
Persistent, with global visibility
3: Store Event ( event, processing context )

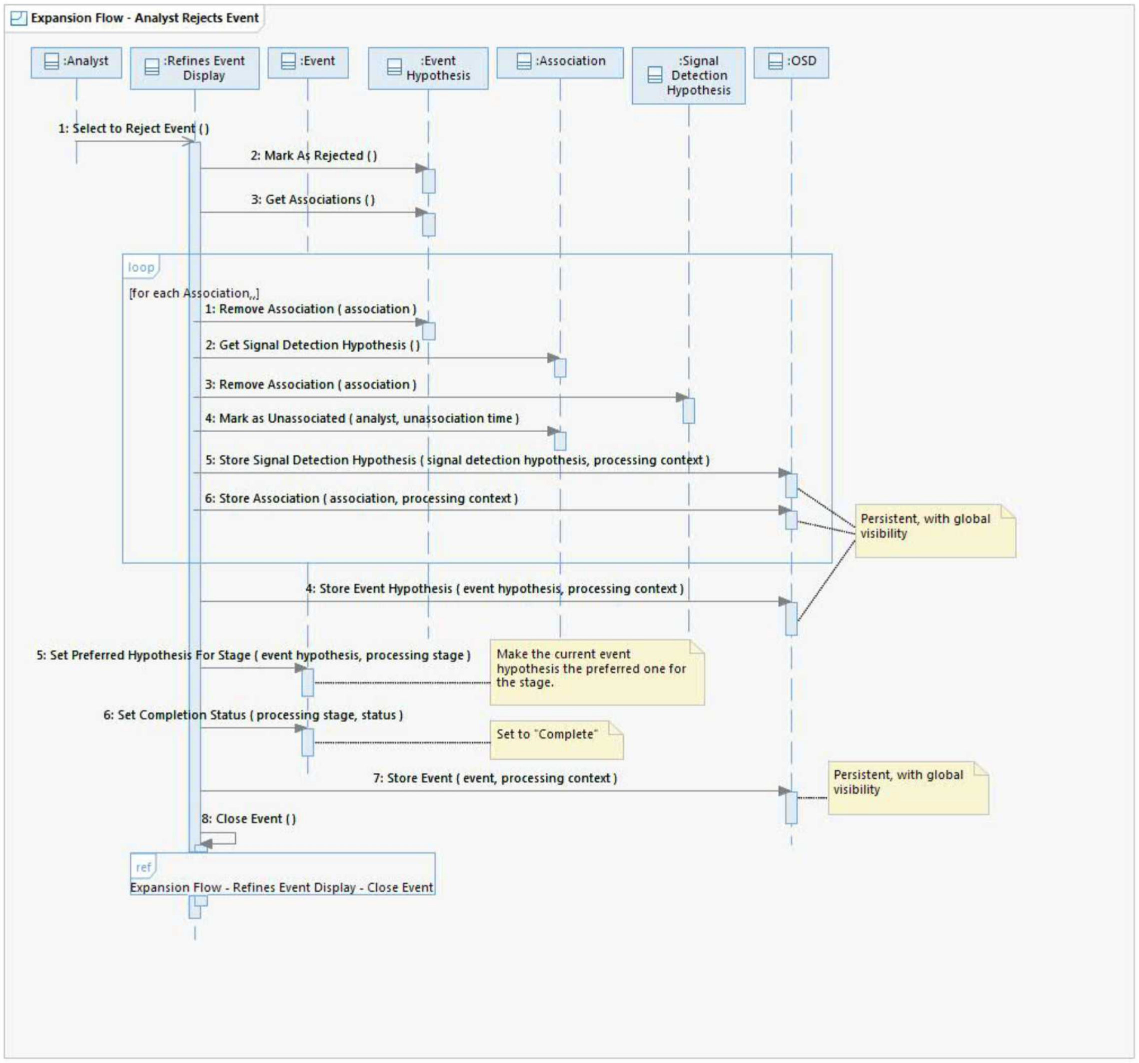
This flow shows the Analyst marking an Event as a reference Event. Although not shown, they may also unmark a previously marked Event in a similar manner. Note that it is the Event that is marked (not the Event Hypothesis), since the mark logically applies to the Event rather than any specific Hypothesis.

*Operation Descriptions*

***Operation: OSD::Store Event()***

Store the given Event with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

## Expansion Flow - Analyst Rejects Event


Expansion Flow - Analyst Rejects Event
:Analyst
:Refines Event Display
:Event
:Event Hypothesis
:Association
:Signal Detection Hypothesis
:OSD
1: Select to Reject Event ( )
2: Mark As Rejected ( )
3: Get Associations ( )
loop
[for each Association,,]
1: Remove Association ( association )
2: Get Signal Detection Hypothesis ( )
3: Remove Association ( association )
4: Mark as Unassociated ( analyst, unassociation time )
5: Store Signal Detection Hypothesis ( signal detection hypothesis, processing context )
6: Store Association ( association, processing context )
Persistent, with global visibility
4: Store Event Hypothesis ( event hypothesis, processing context )
5: Set Preferred Hypothesis For Stage ( event hypothesis, processing stage )
Make the current event hypothesis the preferred one for the stage.
6: Set Completion Status ( processing stage, status )
Set to "Complete"
7: Store Event ( event, processing context )
Persistent, with global visibility
8: Close Event ( )
ref
Expansion Flow - Refines Event Display - Close Event

This flow shows how the Refines Event Display handles rejecting an Event. Rejecting an Event is accomplished by unassociating all Signal Detection Hypotheses from the current Event Hypothesis, making the current Event Hypothesis the preferred Hypothesis for the current stage, saving the Event, Event Hypothesis, Signal Detection Hypothesis, and Association objects, and closing the Refines Event Display.

### *Operation Descriptions*

***Operation: OSD::Store Event Hypothesis()***

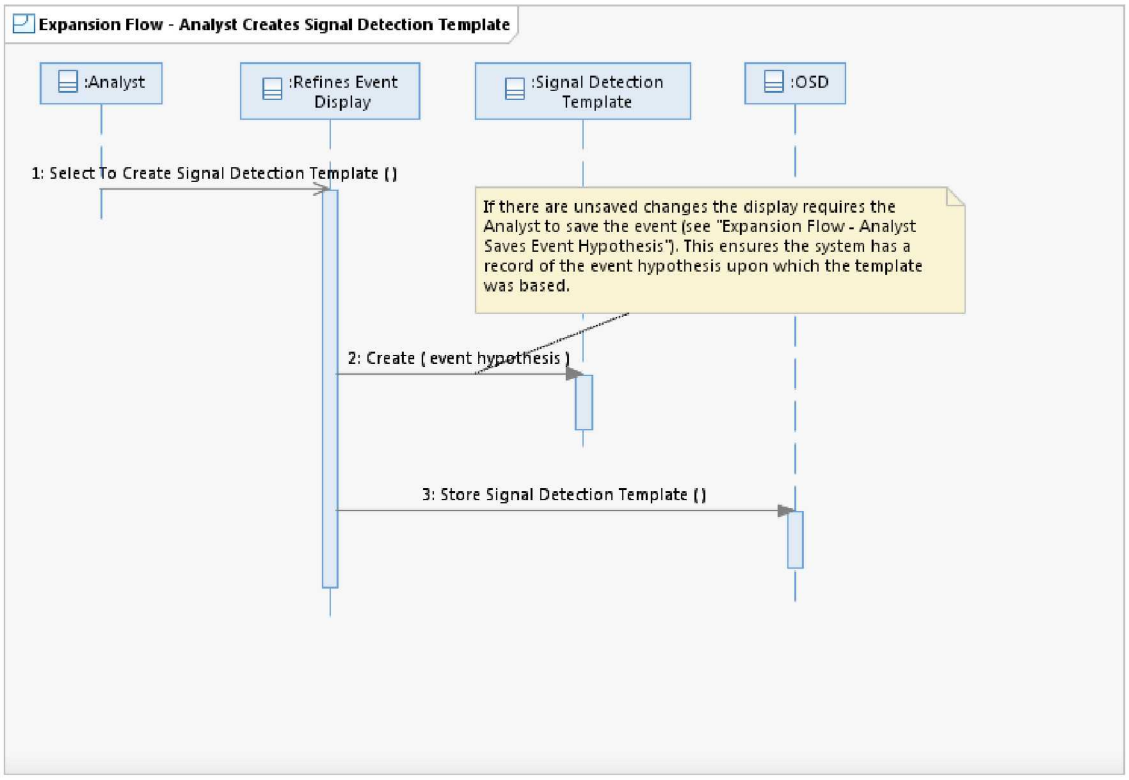
Store the given Event Hypothesis with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

***Operation: OSD::Store Event()***

Store the given Event with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

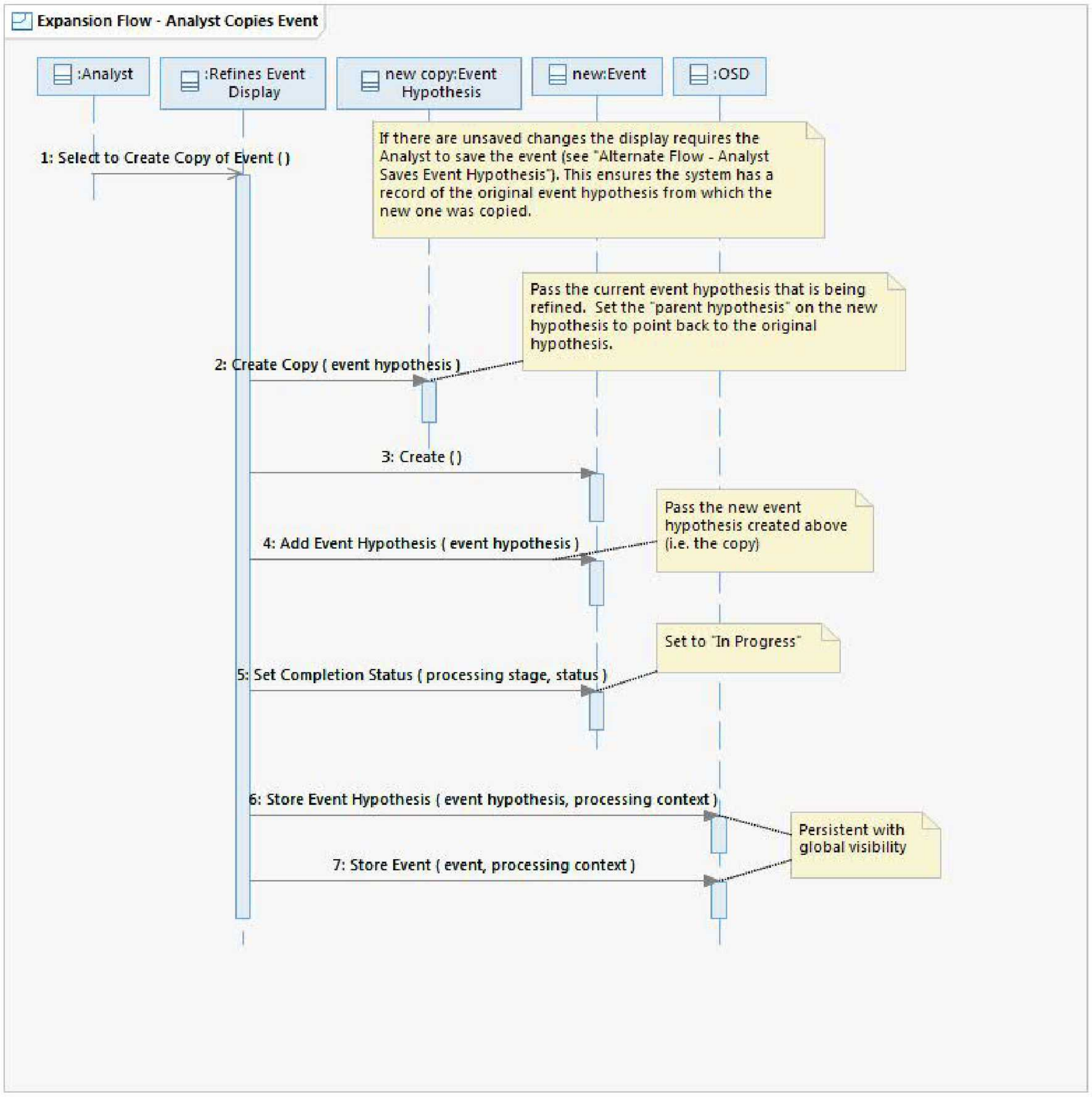
***Operation: Event::Set Completion Status()***
Set the Event Completion Status of an Event in the given processing stage to the given value.

## Expansion Flow - Analyst Creates Signal Detection Template


Expansion Flow - Analyst Creates Signal Detection Template
:Analyst
:Refines Event Display
:Signal Detection Template
:OSD
1: Select To Create Signal Detection Template ( )
If there are unsaved changes the display requires the Analyst to save the event (see "Expansion Flow - Analyst Saves Event Hypothesis"). This ensures the system has a record of the event hypothesis upon which the template was based.
2: Create ( event hypothesis )
3: Store Signal Detection Template ( )

The flow shows the Analyst creating a new Signal Detection Template from the current Event Hypothesis. Stored templates may be applied by the Analyst when building a new Event or associating additional detections to an existing Event Hypothesis (see "Builds Event" UCR).
*Operation Descriptions*
None

## Expansion Flow - Analyst Copies Event


Expansion Flow - Analyst Copies Event
:Analyst
:Refines Event Display
new copy:Event Hypothesis
new:Event
:OSD
1: Select to Create Copy of Event ()
If there are unsaved changes the display requires the Analyst to save the event (see "Alternate Flow - Analyst Saves Event Hypothesis"). This ensures the system has a record of the original event hypothesis from which the new one was copied.
Pass the current event hypothesis that is being refined. Set the "parent hypothesis" on the new hypothesis to point back to the original hypothesis.
2: Create Copy ( event hypothesis )
3: Create ()
Pass the new event hypothesis created above (i.e. the copy)
4: Add Event Hypothesis ( event hypothesis )
Set to "In Progress"
5: Set Completion Status ( processing stage, status )
6: Store Event Hypothesis ( event hypothesis, processing context )
Persistent with global visibility
7: Store Event ( event, processing context )

This flow shows the Analyst creating a copy of the Event they are currently refining. Initially, the copy will have related Association objects linking it to all of the Signal Detection Hypotheses as the original Event Hypothesis (resulting in conflicts which will be detected by the Association Conflict Checker - see "Alternate Flow - Display Handles OSD Callbacks"). The Analyst will need to refine each Event individually to manually remove the conflicts.

*Operation Descriptions*

***Operation: Event Hypothesis::Create Copy()***

Create a copy of the given Event Hypothesis. The copy has all of the same information as the original (e.g. same detections, location, etc.), with the following exceptions:

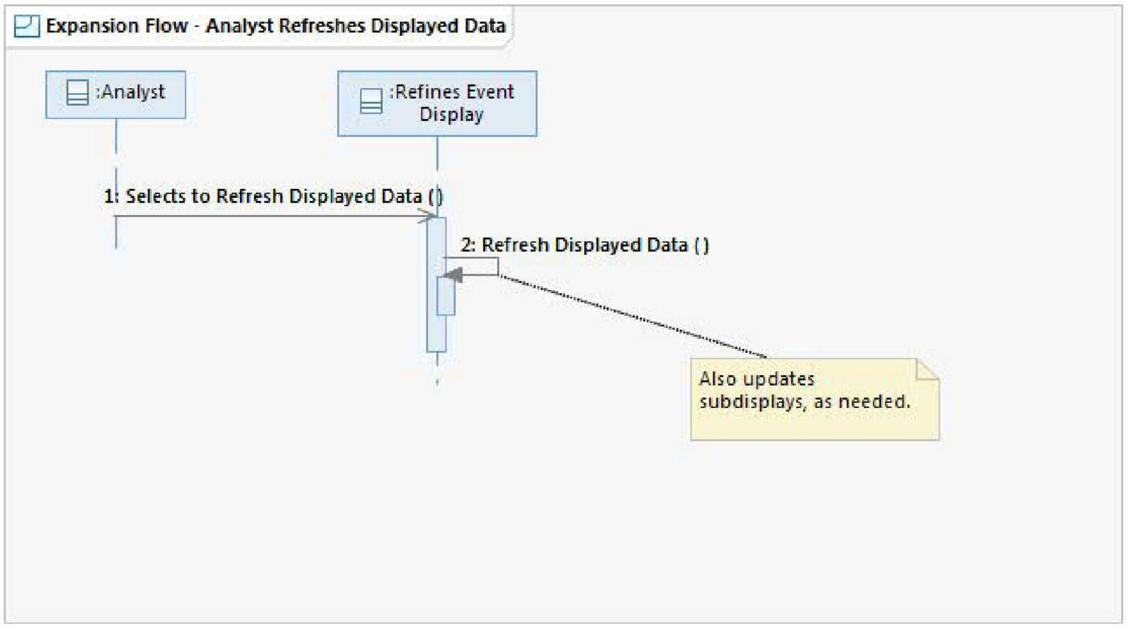
- The copy points to the original as its parent
- The copy starts out with an empty Analyst comment

***Operation: OSD::Store Event Hypothesis()***
Store the given Event Hypothesis with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

***Operation: OSD::Store Event()***
Store the given Event with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

***Operation: Event::Set Completion Status()***
Set the Event Completion Status of an Event in the given processing stage to the given value.

## Expansion Flow - Analyst Refreshes Displayed Data

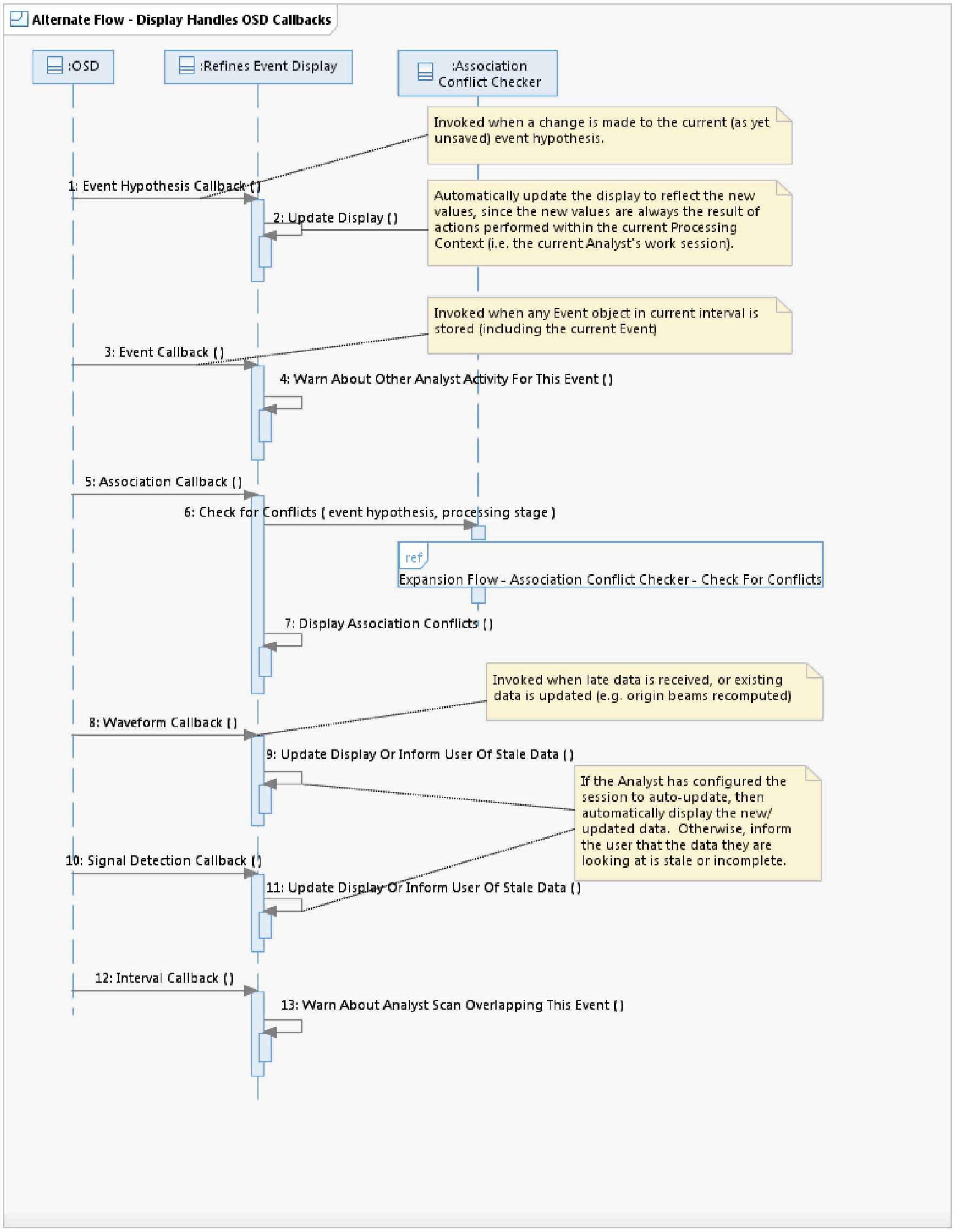

Expansion Flow - Analyst Refreshes Displayed Data
:Analyst
:Refines Event Display
1: Selects to Refresh Displayed Data ()
2: Refresh Displayed Data ()
Also updates subdisplays, as needed.

This flow shows how the Analyst refreshes his/her display to show the latest waveforms, Signal Detections and Signal Detection associations. The Analyst needs this capability since late-arriving waveforms, Signal Detections and Signal Detection associations made by other Analysts are not automatically displayed to the Analyst. The Analyst is notified when there is new data that is not shown on the display (see "Alternate Flow - Display Handles OSD Callbacks"). The Analyst may refresh the display to show that data at any time via this flow. Note that the display does not need to retrieve the new data from the OSD since it already has it due to subscriptions with the OSD (see "Expansion Flow - Refines Event Display - Open Event").

*Operation Descriptions*

***Operation: Refines Event Display::Refresh Displayed Data()***
Update displayed waveforms and Signal Detections to reflect the current state within the processing stage.

## Alternate Flow - Display Handles OSD Callbacks


Alternate Flow - Display Handles OSD Callbacks
:OSD
:Refines Event Display
:Association Conflict Checker
Invoked when a change is made to the current (as yet unsaved) event hypothesis.
1: Event Hypothesis Callback ()
2: Update Display ()
Automatically update the display to reflect the new values, since the new values are always the result of actions performed within the current Processing Context (i.e. the current Analyst's work session).
Invoked when any Event object in current interval is stored (including the current Event)
3: Event Callback ()
4: Warn About Other Analyst Activity For This Event ()
5: Association Callback ()
6: Check for Conflicts ( event hypothesis, processing stage )
ref
Expansion Flow - Association Conflict Checker - Check For Conflicts
7: Display Association Conflicts ()
Invoked when late data is received, or existing data is updated (e.g. origin beams recomputed)
8: Waveform Callback ()
9: Update Display Or Inform User Of Stale Data ()
If the Analyst has configured the session to auto-update, then automatically display the new/updated data. Otherwise, inform the user that the data they are looking at is stale or incomplete.
10: Signal Detection Callback ()
11: Update Display Or Inform User Of Stale Data ()
12: Interval Callback ()
13: Warn About Analyst Scan Overlapping This Event ()

This flow shows how the Refines Event Display handles various callbacks from the OSD. The Refines Event Display subscribes for the current Event Hypothesis in order to monitor updates to it as a result of executed Processing Sequences. The display subscribes for all Events in the interval in order to monitor other Analyst activity on the current Event and to check for association conflicts with other Events. The display subscribes for Signal Detections and Waveforms in order to display updates to that information made by other analysts. The display subscribes for Intervals to determine if an interval that overlaps the current Event is under active review by another analyst.

### *Operation Descriptions*

***Operation: Refines Event Display::Event Callback()***

Callback invoked any time there is a change in the subscribed Event (e.g. a new Event Hypothesis for the Event is saved, or the preferred Hypothesis for a processing stage changes).

***Operation: Refines Event Display::Warn About Other Analyst Activity For This Event()***

Warn the current Analyst about another Analyst working on the current Event.

***Operation: Refines Event Display::Signal Detection Callback()***

Invoked any time the set of Signal Detections that fall within the current time interval changes. The callback indicates what changed.

***Operation: Refines Event Display::Waveform Callback()***

Invoked any time new raw or derived waveforms overlapping the time of the Event are received (e.g. late data, beams).

***Operation: Refines Event Display::Event Hypothesis Callback()***

Callback invoked whenever portions of the Event Hypothesis are changed by the system. This callback can only occur as part of automatic processing sequences executed by the Processing Sequence Control mechanism, since changes made by other Analysts are stored in separate Event Hypotheses.

***Operation: Refines Event Display::Update Display()***

Update the display of the Event Hypothesis being refined to reflect any changes that may have occurred. Indicate items that are out-of-date or inconsistent (e.g. beam may be out-of-date after refining Event location).

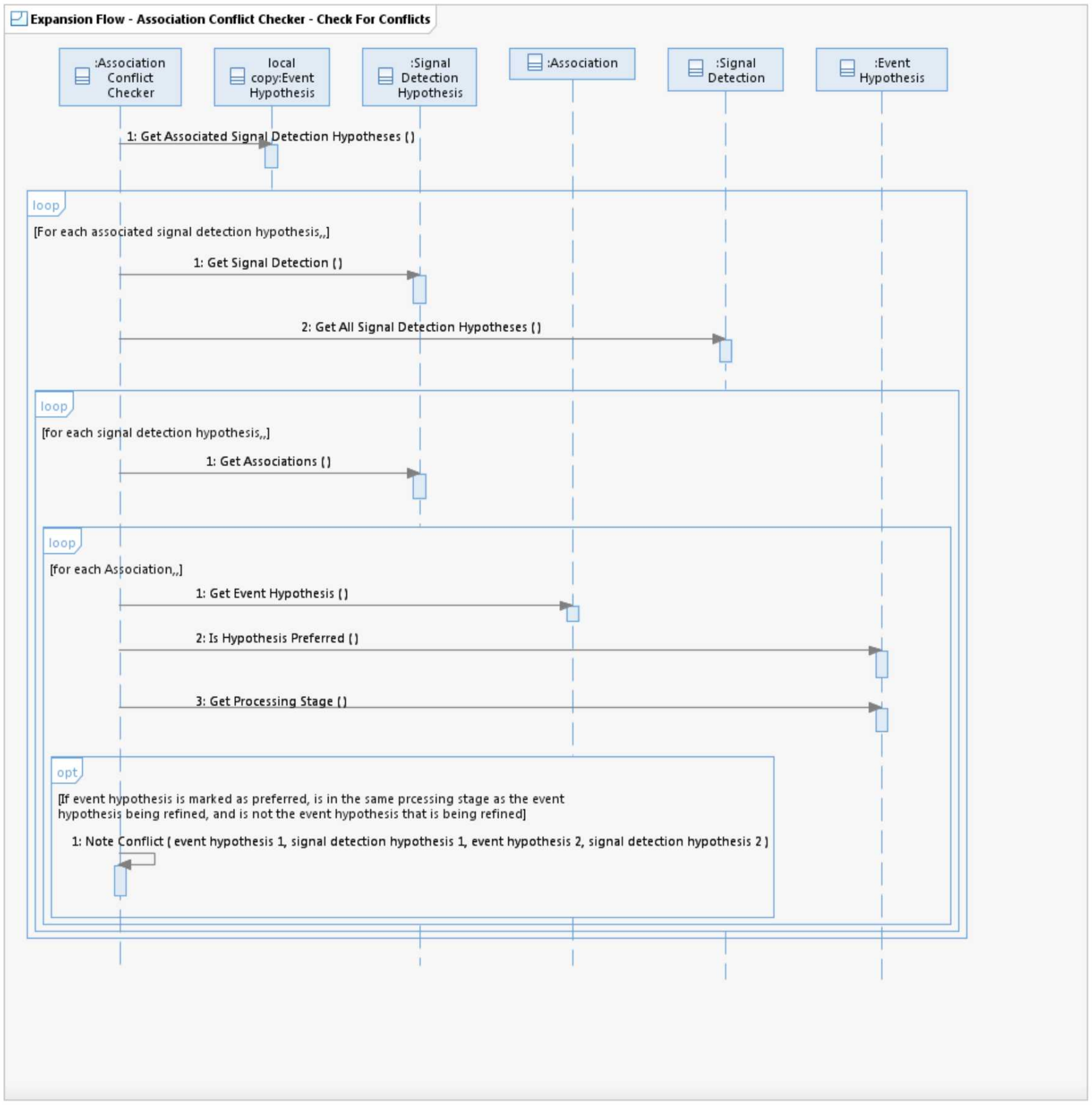
***Operation: Association Conflict Checker::Check for Conflicts()***

Check the given Event Hypothesis against the other Events in the processing stage for association conflicts. Only check for conflicts between Event Hypotheses marked as preferred. A conflict exists if a Signal Detection is associated to more than one preferred Hypothesis in the processing stage.

***Operation: Refines Event Display::Warn About Analyst Scan Overlapping This Event()***

Warn the Analyst if the interval that overlaps the current Event being refined is under active review by another Analyst.

## Expansion Flow - Association Conflict Checker - Check For Conflicts



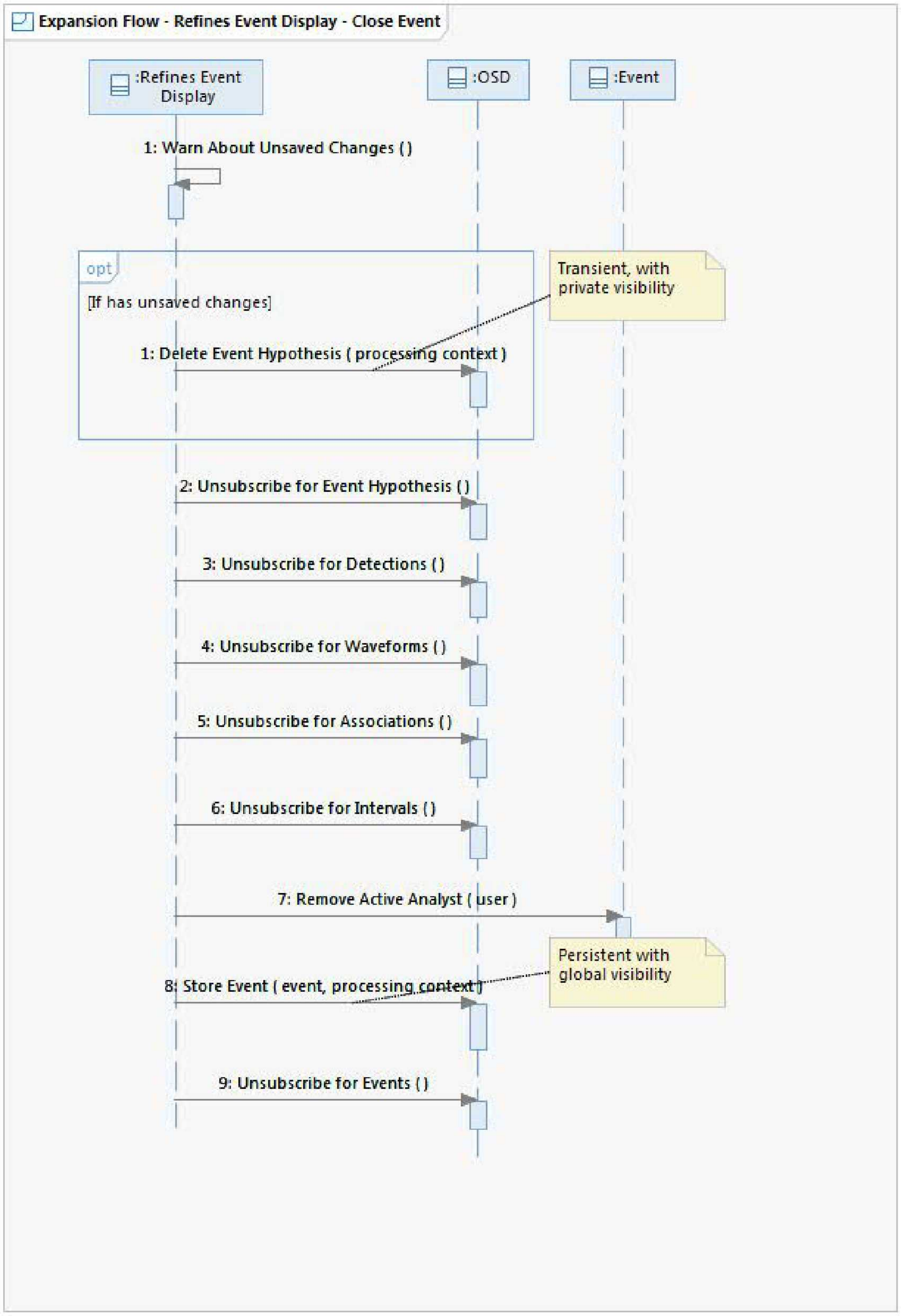
Expansion Flow - Association Conflict Checker - Check For Conflicts
:Association Conflict Checker
local copy:Event Hypothesis
:Signal Detection Hypothesis
:Association
:Signal Detection
:Event Hypothesis
1: Get Associated Signal Detection Hypotheses ( )
loop
[For each associated signal detection hypothesis,,]
1: Get Signal Detection ( )
2: Get All Signal Detection Hypotheses ( )
loop
[for each signal detection hypothesis,,]
1: Get Associations ( )
loop
[for each Association,,]
1: Get Event Hypothesis ( )
2: Is Hypothesis Preferred ( )
3: Get Processing Stage ( )
opt
[If event hypothesis is marked as preferred, is in the same prcessing stage as the event hypothesis being refined, and is not the event hypothesis that is being refined]
1: Note Conflict ( event hypothesis 1, signal detection hypothesis 1, event hypothesis 2, signal detection hypothesis 2 )

This flow shows how the Association Conflict Checker checks for association conflicts between the local (unsaved) Event Hypothesis and other Event Hypotheses. The local Event Hypothesis is input to this flow. The flow returns a list of association conflicts. Note that, by definition, a conflict can only exist with an Event Hypothesis that is marked as preferred. Note also that each Event can have at most one Event Hypothesis marked as preferred for each processing stage.

*Operation Descriptions*

None

## Expansion Flow - Refines Event Display - Close Event


Expansion Flow - Refines Event Display - Close Event
:Refines Event Display
:OSD
:Event
1: Warn About Unsaved Changes ( )
opt
[If has unsaved changes]
1: Delete Event Hypothesis ( processing context )
Transient, with private visibility
2: Unsubscribe for Event Hypothesis ( )
3: Unsubscribe for Detections ( )
4: Unsubscribe for Waveforms ( )
5: Unsubscribe for Associations ( )
6: Unsubscribe for Intervals ( )
7: Remove Active Analyst ( user )
Persistent with global visibility
8: Store Event ( event, processing context )
9: Unsubscribe for Events ( )

This flow shows what the Refines Event Display does upon closing the current Event.

*Operation Descriptions*

***Operation: OSD::Store Event()***

Store the given Event with the given lifespan (persistent vs. transient) and visibility (private vs. global) as specified by the given Processing Context and notify relevant subscribers via callbacks.

## STATE MACHINE DIAGRAMS

None

## NOTES

1. In this UCR and all of its child UCRs (e.g. "Refines Event Location", "Refines Event Magnitude", etc.), the display classes store computed results to transient storage via the OSD mechanism as the Event is refined in order to trigger execution of configured processing sequences. These processing sequences are configured by the System Maintainer (see "Defines Processing Sequence" UCR), and are automatically executed by the Processing Sequence Control mechanism in response to OSD callbacks (the Processing Sequence Control mechanism is shown in "System Detects Events" UCR). These changes to the Event Hypothesis are stored with private visibility such that the changes are accessible only to the Analyst work session where the changes are being made; other Analysts cannot see the updates until the Analyst chooses to save the Event Hypothesis, at which time the Event Hypothesis is stored to persistent storage via the OSD mechanism. The storage of an Event Hypothesis to persistent storage may trigger additional processing sequences, as defined by the System Maintainer.

2. The Analyst may undo/redo editing operations while refining the Event, but only back to the last save.

3. See "Marks Processing Stage Complete" UCR for a state machine diagram for Event Completion Status.

4. This UCR covers creation of Signal Detection templates. The Analyst applies such templates when building new Events (see "Builds Event" and "Scans Waveforms and Unassociated Detections" UCRs).

## OPEN ISSUES

None

This page intentionally left blank

IDC Use Case Realization Report

# UCR-08.05 Views Event History

## USE CASE DESCRIPTION

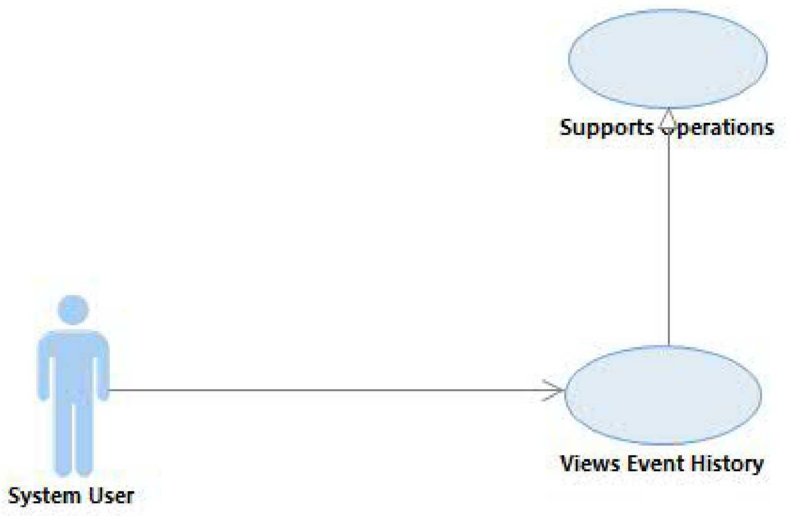
This architecturally significant use case describes how the System User observes the change history of a given event. The change history is a series of one or more saved event hypotheses. System Users view all the event hypotheses and the set of location solutions for each hypothesis. The System User views the relationship between event hypotheses including the preferred hypothesis for each processing stage. The event change history persists across work sessions for subsequent review.

This use case is architecturally significant because it involves storing multiple event hypotheses for each event containing provenance information including waveforms, defining signal detections, feature measurements and parameter settings.

## ARCHITECTURE DESCRIPTION

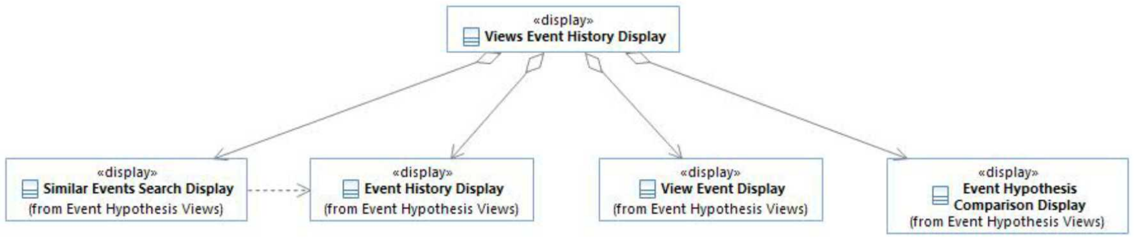
The System User opens the Views Event History Display to select and view an Event and the related Event Hypotheses. The System User selects an Event using the Event Search Display. The System User selects an event and opens the Event History Display. The Event History Display shows all the Event Hypotheses for an Event and the relationships between hypotheses stored by the OSD. The System User can select to view the information for an individual Event Hypothesis using the View Event Display or compare multiple hypotheses using the Event Hypothesis Comparison Display.

## USE CASE DIAGRAM



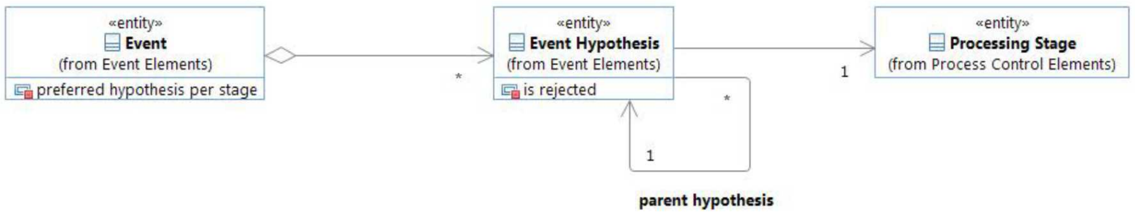
## CLASS DIAGRAMS

### Classes - Displays


«display»
Views Event History Display
«display»
Similar Events Search Display
(from Event Hypothesis Views)
«display»
Event History Display
(from Event Hypothesis Views)
«display»
View Event Display
(from Event Hypothesis Views)
«display»
Event Hypothesis Comparison Display
(from Event Hypothesis Views)

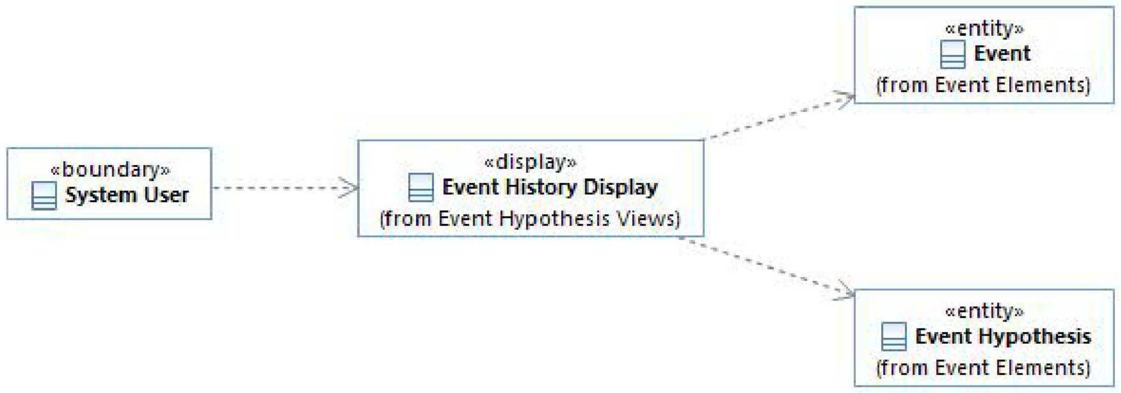
This diagram shows the display classes for selecting and viewing Events and Event Hypotheses. The System User uses the Event Search Display to select an Event. The Event History Display shows the relationships between Event Hypotheses for an Event. The View Event Display shows detailed information about an Event Hypothesis. The System User uses the Event Hypothesis Comparison Display to compare multiple hypotheses.

## Classes - Event



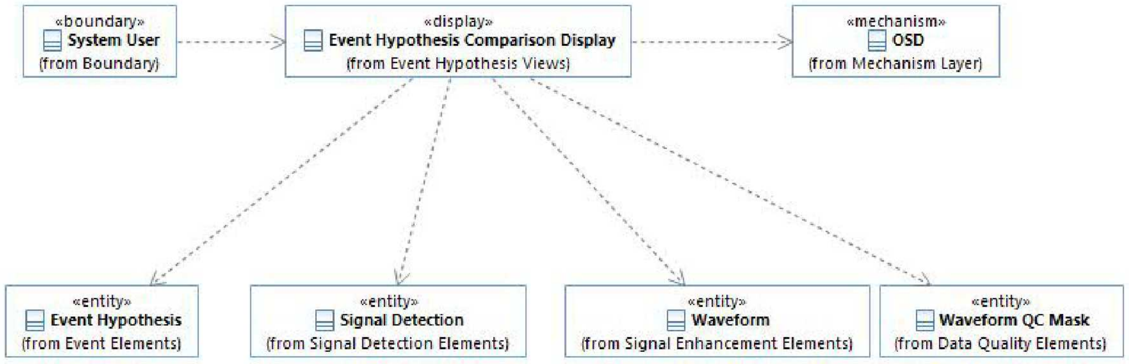
This diagram shows the relationships between Events and Event Hypotheses and the relationships between hypotheses. Each hypothesis is related to its parent hypothesis that was the basis for the child hypothesis. The parent hypothesis can be for the same event or a different event.

## Classes - Event History Display



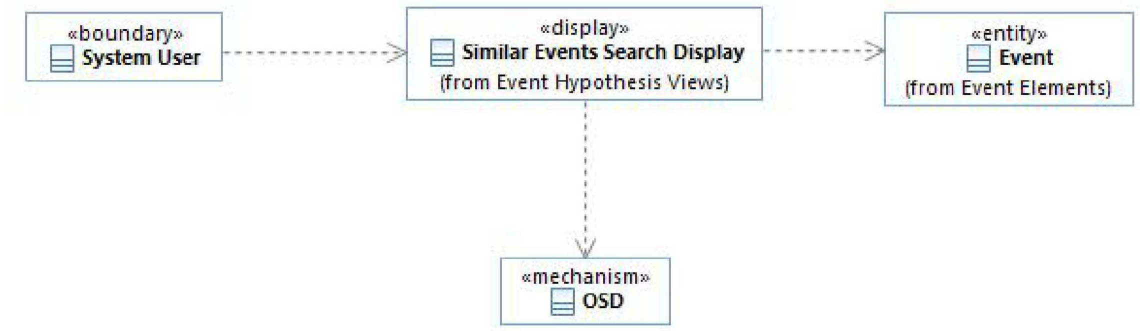
This diagram shows the Event History Display and related classes.

## Classes - Event Hypotheses Comparison Display



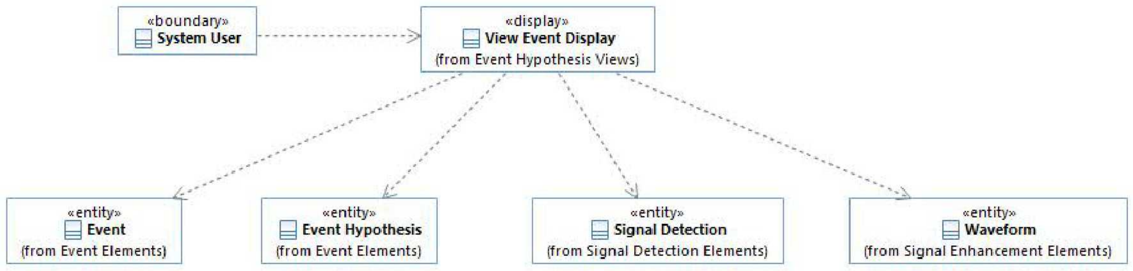
This diagram shows the Event Hypothesis Comparison Display and related classes.

## Classes - Event Search Display



This diagram shows the Event Search Display. The System User uses the Event Search Display to select Events and retrieve the Events from the OSD.

## Classes - View Event Display



This diagram shows the View Event Display and related classes that provide information about Events and Event Hypotheses.

## CLASS DESCRIPTIONS

***<<boundary>> System User***
Represents the System User actor.

***<<entity>> Event***
Represents information about an Event. Keeps track of all the Event Hypotheses for the event, which hypothesis is the preferred one for each processing stage, the active analysts for the event (i.e. whether the event is under "active review"), whether the event is "complete" for each processing stage, and other event-related information.

***<<entity>> Event Hypothesis***
Represents geophysical information about an Event as determined by an Analyst or through pipeline processing. There can be multiple hypotheses of the same Event (e.g. different associated signal detection hypotheses, different location solutions).

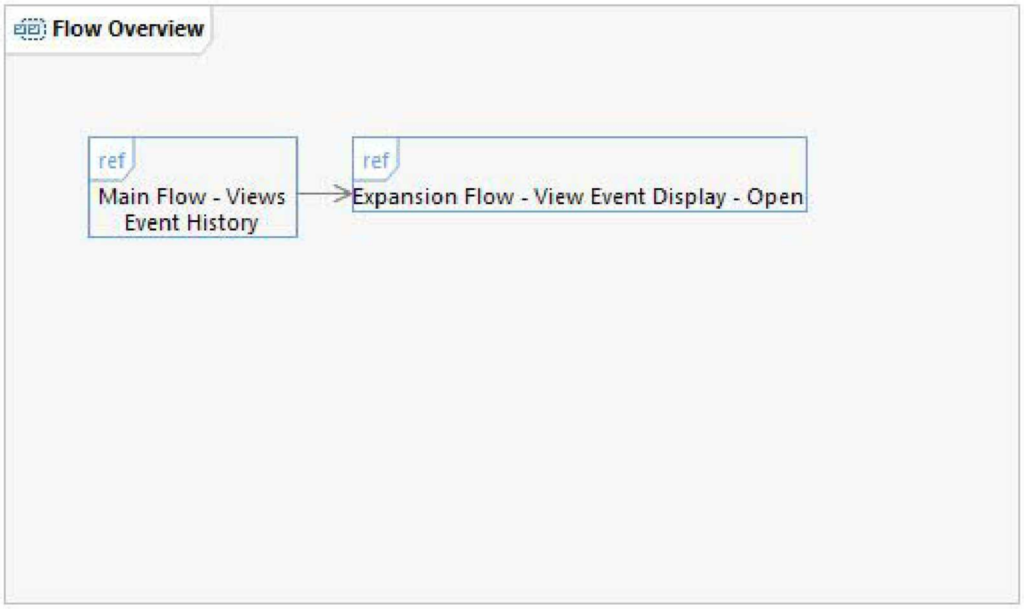
***<<entity>> Processing Stage***
Represents a named stage of data processing, which may be part of the System Maintainer-defined workflow or an Analyst-defined stage outside the workflow. All Processing Results are associated to a Processing Stage. The previous processing stage indicates the stage to be used as

the default starting point when creating new processing results in the stage (e.g. when refining an event in the stage).
***<<entity>> Signal Detection***
Represents information about a Signal Detection and keeps track of all the Signal Detection Hypotheses for the Signal Detection. Represents information about a Signal Detection and keeps track of all the Signal Detection Hypotheses for the Signal Detection. For an unassociated Signal Detection the preferred hypothesis is the most recently created hypothesis. For an associated Signal Detection the preferred hypothesis is the one associated to a preferred Event Hypothesis.

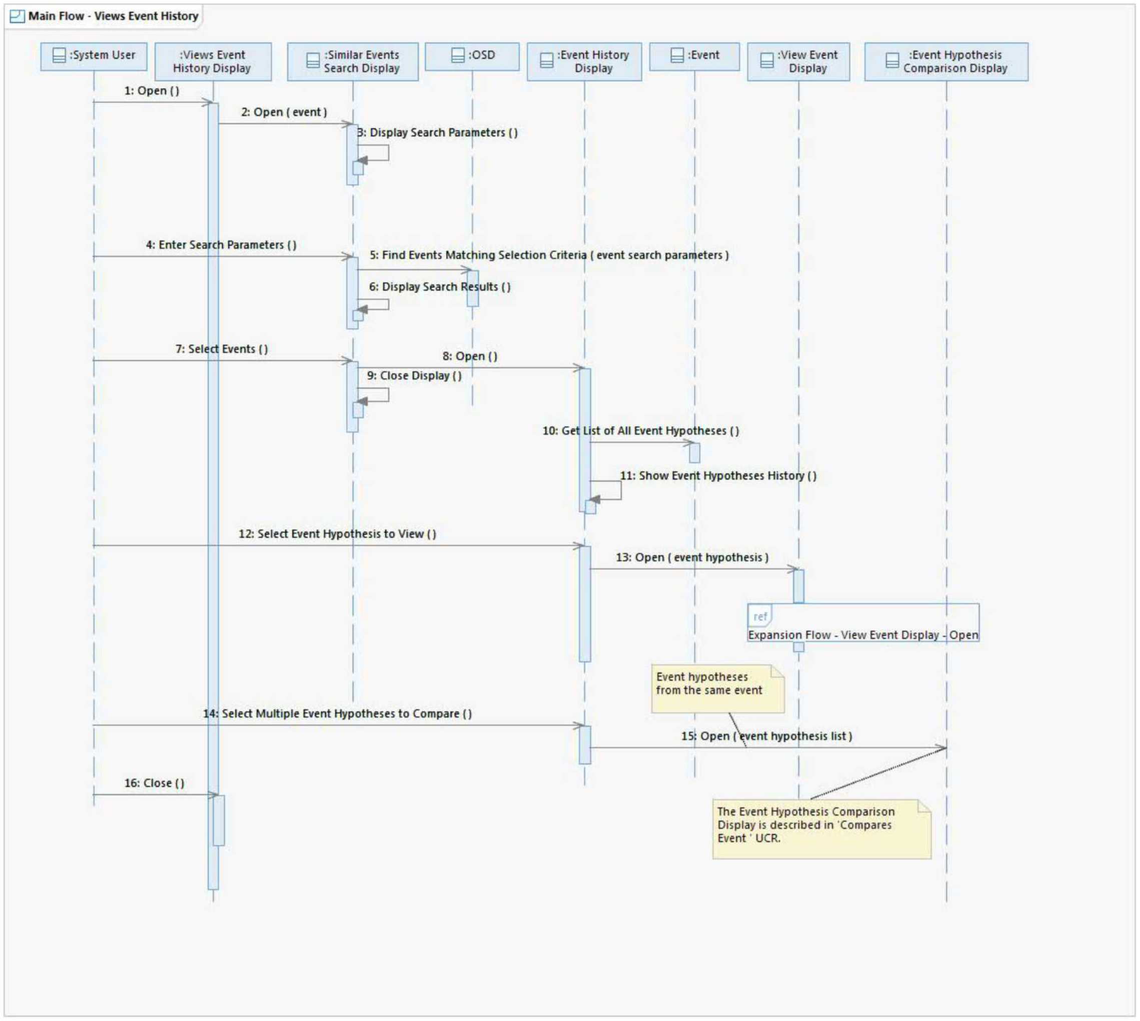
***<<mechanism>> OSD***
Represents the Object Storage and Distribution mechanism for storing and distributing data objects internally within the system.

## SEQUENCE DIAGRAMS

### Flow Overview


Flow Overview
ref
Main Flow - Views Event History
ref
Expansion Flow - View Event Display - Open

## Main Flow - Views Event History


Main Flow - Views Event History
:System User
:Views Event History Display
:Similar Events Search Display
:OSD
:Event History Display
:Event
:View Event Display
:Event Hypothesis Comparison Display
1: Open ()
2: Open ( event )
3: Display Search Parameters ()
4: Enter Search Parameters ()
5: Find Events Matching Selection Criteria ( event search parameters )
6: Display Search Results ()
7: Select Events ()
8: Open ()
9: Close Display ()
10: Get List of All Event Hypotheses ()
11: Show Event Hypotheses History ()
12: Select Event Hypothesis to View ()
13: Open ( event hypothesis )
ref
Expansion Flow - View Event Display - Open
Event hypotheses from the same event
14: Select Multiple Event Hypotheses to Compare ()
15: Open ( event hypothesis list )
16: Close ()
The Event Hypothesis Comparison Display is described in 'Compares Event ' UCR.

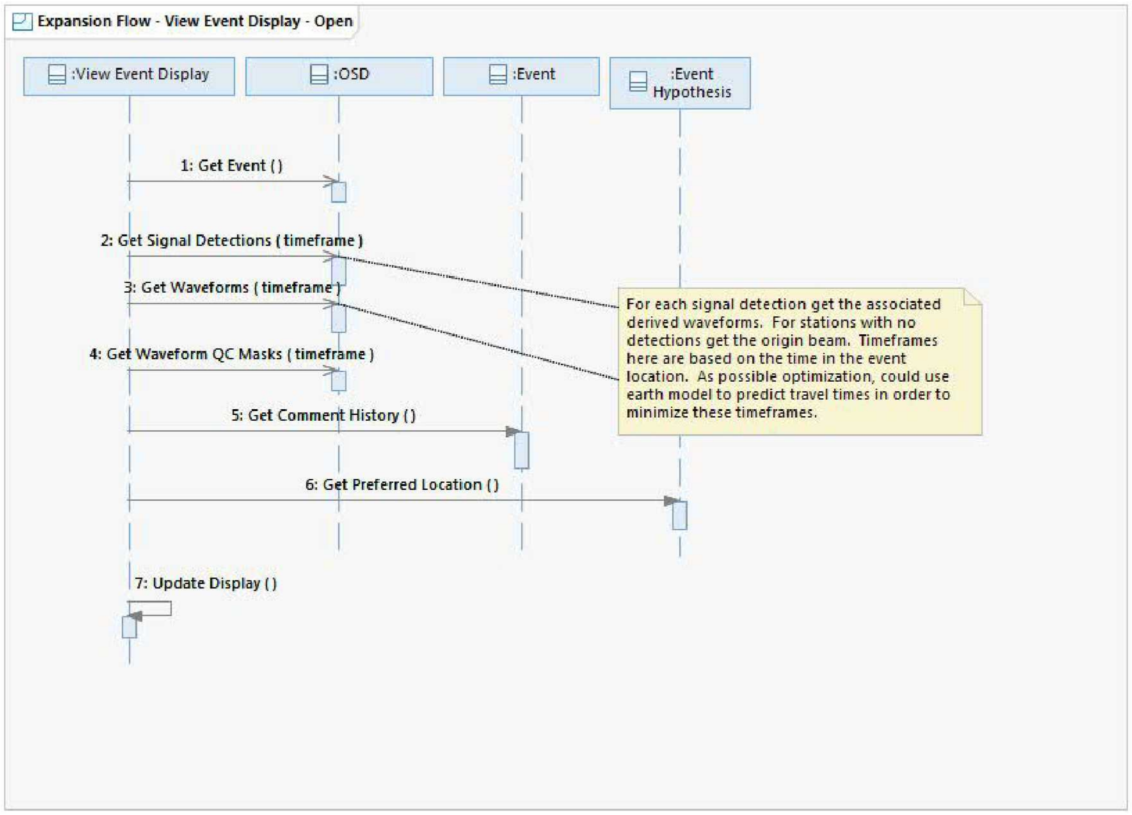
This flow shows how the System User selects an Event and views the event history. Optionally, the System User may view a read-only copy of a selected Event Hypothesis or compare multiple Event Hypotheses.

### *Operation Descriptions*

***Operation: Event::Get List of All Event Hypotheses()***

Return a list of all the Event Hypothesis for the given event, including summary information such as the processing stage and which hypotheses have been designated as preferred.

## Expansion Flow - View Event Display - Open


Expansion Flow - View Event Display - Open
:View Event Display
:OSD
:Event
:Event Hypothesis
1: Get Event ( )
2: Get Signal Detections ( timeframe )
3: Get Waveforms ( timeframe )
4: Get Waveform QC Masks ( timeframe )
5: Get Comment History ( )
6: Get Preferred Location ( )
7: Update Display ( )
For each signal detection get the associated derived waveforms. For stations with no detections get the origin beam. Timeframes here are based on the time in the event location. As possible optimization, could use earth model to predict travel times in order to minimize these timeframes.

This flow shows how the View Event Display is created and the classes contributing information for the display.

*Operation Descriptions*

***Operation: Event::Get Comment History()***

Return all Analyst-entered comments associated with the Event.

## STATE MACHINE DIAGRAMS

None

## SSD MAPPINGS

## NOTES

1. View Event Display is a read-only display of an Event Hypothesis and related information. The event hypothesis information is generated in 'System Detects Event' and 'Refines Event' UCs and includes station quality metrics, event hypothesis quality metrics, map displays including geographic regions and geospatial data, and both unassociated and associated signal detections. Displays will be similar to Analyst displays for 'Refines Event' UC.

2. Event Hypothesis Comparison Display shows QC Masks for associated waveforms per S-1292.

## OPEN ISSUES

None

## DISTRIBUTION

| | | | |
|---|---|---|---|
| 1 | MS0401 | Rudy Sandoval | 5563 (electronic copy) |
| 1 | MS0899 | Technical Library | 9536 (electronic copy) |

Sandia National Laboratories