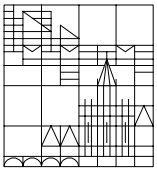


University of Constance
Dept. of Computer Science

FAILURE DETECTION MANAGEMENT FRAMELET

Concept And Architecture Description

Abstract

This document was written as part of the study "Design and Prototyping of a Software Framework for the AOCS" done under contract Estec/13776/99/NL/MV for ESA-Estec. The purpose of the study is the development of a software framework for the Attitude and Orbit Control Subsystem (AOCS) of a satellite. The framework will be built as a collection of framelets. This document describes the failure detection management framelet. This framelet defines an architecture to handle failure detection tasks. The framelet enhances reusability because it decouples the task of managing the failure detection function from the task of carrying out failure detection tests.

Written By: A. Pasetti (University of Constance / SWE)

Date: 30 April 2002

Issue: 2.2

Reference: SWE / 99 / AOCS / 010

TABLE OF CONTENTS

1 REFERENCES

RD1 E. Gamma *et al.* (1995), *Design Patterns – Elements of Reusable Object Oriented Software*, Reading, Massachusetts: Addison-Wesley

RD2 A. Pasetti (2000), *AOCS Framework – Concept Level Description*, AOCS Framework Document ref. SWE/99/AOCS/004

RD3 A. Pasetti (2000), *Operational Mode Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/009

RD4 A. Pasetti (2001), *Software Frameworks and Embedded Control Systems*, LNCS Series, Springer-Verlag, To appear in Dec. 2001

2 ACRONYMS

AAD	Attitude Anomaly Detection
AOCS	Attitude and Orbit Control Subsystem
AST	Autonomous Star Tracker
CSS	Coarse Sun Sensor
ES	Earth Sensor
FDIR	Failure Detection, Isolation and Recovery
FPM	Fine Pointing Mode
FSS	Fine Sun Sensor
GYR	Gyroscope
KF	Kalman Filter
IAM	Initial Acquisition Mode
OBDH	On-Board Data Handling system (aka as OBDS)
NM	Normal Mode
NTT	Non-Time-Tagged
OCM	Orbit Control Mode
OO	Object-Oriented
PD	Proportional-Derivative controller
PI	Proportional-Integral controller
PID	Proportional-Integral-Derivative controller
RRM	Rate Reaction Mode
RTOS	Real-Time Operating System
RW	Reaction Wheel
SAS	Sun Attitude Sensor
SBM	Stand-By Mode
SPS	Sun Presence Sensor
STR	Star Tracker
SLM	Slewing Mode
SM	Safe Mode
TC	Telecommand
THU	Thruster
TM	Telemetry
TT	Time-Tagged

3 INTRODUCTION

This document describes the *failure detection management framelet* for the AOCS framework. The framelet is described at both the framelet concept level and at the framelet architectural level.

This framelet defines an architecture to handle failure detection tasks.

The framelet enhances reusability because it decouples the task of *managing the failure detection function* from the task of *carrying out failure detection tests*.

3.1 Context

The context for the design of the framelet is described in RD2. The present document assumes that the reader is familiar with RD2 and in particular with the sections dealing with failure detection management and with the overall FDIR approach.

The architecture proposed here follows the concept outlined in RD2.

In comparing the present document with RD2, readers should bear in mind that the class definitions presented in the latter document are not necessarily entirely consistent with the class definitions presented here. This is because the main purpose of RD2 was to introduce an architectural *concept* whereas the main purpose of the present document is to describe an architecture. The design presented here therefore should be regarded as an evolution of the design presented in RD2.

3.2 Applicability to Java Version

The AOCS Framework was first implemented in C++ and then ported to Java. This document was originally written for the C++ version and is only partially applicable to the Java version. Generally speaking, the description of the framelet at design level – in particular its design patterns – is language-independent and is equally applicable to both the C++ and Java versions whereas the architectural-level description is more tied to the C++ version. For a detailed description of the architecture of the Java framework, readers should refer to the JavaDoc documentation generated from it.

The porting of the AOCS Framework to Java was done in the "Real Time Java Project". The issues that should be borne in mind when using this document for the Java version of the AOCS framework are presented in the project web site currently located at the following address: www.aut.ee.ethz.ch/~pasetti/RealTimeJavaFramework/index.html. Some specific points to note are:

– Events in the Java framework are implemented using the Java event mechanism.

– Property monitoring (see section 5.3) is done in a slightly different way. The Java framework does not have property objects. Monitorable components (components that expose properties that can be subjected to monitoring) are characterized by implementation of interface `Monitorable`. Monitoring check objects therefore hold a reference to a `Monitorable` component rather than to a `Property` object.

– The change event repository hot-spot (section 8.3) is not applicable to the Java framework. Event repositories are event listeners and can be linked to the mode manager through the associated `addListener` methods.

3.3 Notation

The pseudo-code examples in this document use a C++ notation.

The class diagrams use UML notation generated with the reverse engineering tool of the *Together* tool.

4 FRAMELET CONSTRUCTS

The architectural constructs exported by this framelet are listed in the following table:

FAILURE DETECTION MANAGEMENT FRAMELET
Design Pattern
Failure Detection Pattern: design pattern to separate the management of failure detection tests from their implementation.
Framelet Interfaces
`ConsistencyCheckable` : interface for objects that can perform consistency checks on their internal state. `FailureDetectionModeManager` : interface for the operational mode manager for the failure detection manager.
Framelet Core Components
`MonitoringCheck` : component encapsulating a monitoring check action `FailureDetectionManager` : component encapsulating a failure detection manager
Framelet Default Components
`FollowerFailureDetectionModeManager` : default mode manager for the failure detection manager based on the follower mechanism.

The components listed above are those envisaged for the prototype version of the AOCS framework. Later versions may offer a richer set of default implementations of the framelet interfaces. In particular, more default mode manager components may be supplied implementing alternative mode switching logics and interface `ConsistencyCheckable` should be implemented more fully.

5 FAILURE DETECTION MODEL

The failure detection model is as outlined in the failure management section of RD2. As explained in this reference, three types of failure detection mechanisms are possible:

- autonomous failure checks
- consistency checks
- property monitoring

The above types of failure detection mechanisms are discussed in greater detail in the next three subsections.

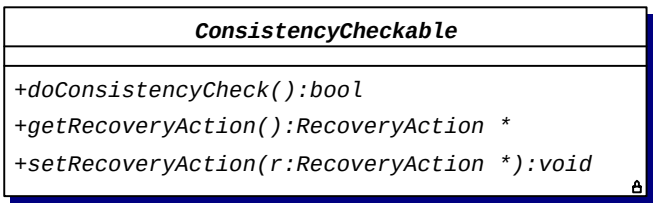
5.1 Autonomous Failure Checks

The failure detection manager performs systematic failure detection checks. However, all other AOCS objects may also perform failure checks as part of their normal operation. These checks are called autonomous failure checks.

AOCS objects inherit from their base class `AocsObject` an operation – `reportFailure` – that they can use to report a failure. This operation creates a failure event and stores it in the failure event repository.

5.2 Consistency Checks

Consistency checks are performed on objects implementing the `ConsistencyCheckable` interface. This interface is defined as follows:



The consistency check is performed by calling method `doConsistencyCheck`. The consistency check test can either `true` (no consistency failure) or `false` (consistency check failure). In case of detection of a consistency check failure, the failure detection manager creates an event of type `FailureEvent` as described in the next section.

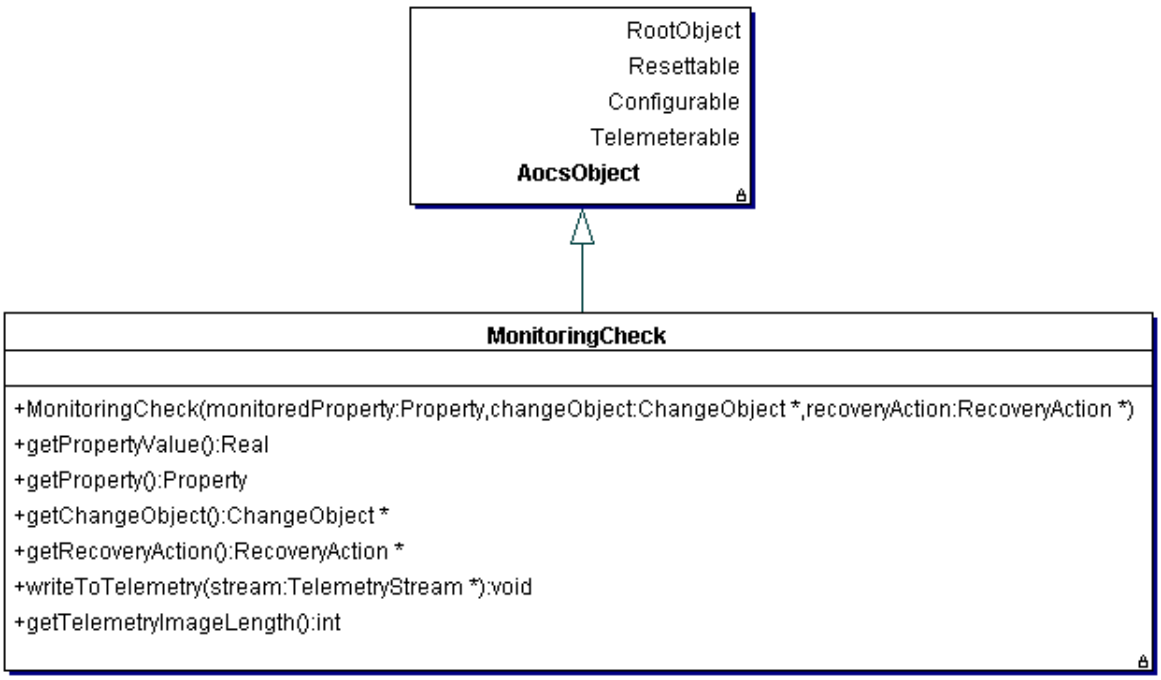
The AOCS framework prescribes that a recovery action be associated to each consistency check. The getter and setter methods allow the recovery action objects to be retrieved or defined.

5.3 Property Monitoring

Failures are detected by monitoring the properties of some objects for specific changes. Property monitoring can be performed in several manners but the failure detection manager performs it exclusively through the direct monitoring mechanism.

A property monitoring is encapsulated in a *monitoring check* object which packages in a single object the property to be checked, the change object defining the type of change and the recovery action to be performed in case the change is found to have occurred.

A monitoring check object has the following structure:



The public methods specific to this class (ie. not inherited from base classes) are described in the table:

`MonitoringCheck(p, c, r)`
Constructor that builds a monitoring object for object p and change object c and associate recovery action r to the monitoring action (ie. r is the recovery action associated to the failure event generated when the value of property p is found to have undergone the change specified by c).
`getPropertyValue`

Return the value of the property encapsulated in the monitoring object.
`getProperty`
Return property object encapsulated in the monitoring object.
`getChangeObject`
Return the change object encapsulated in the monitoring object.
`getRecoveryAction`
Return the recovery action encapsulated in the monitoring object.

The monitoring action consists in passing the property value – obtained from the `Property` object – through the change filter – as encapsulated in the `ChangeObject` object. If the change in the property value is found to have occurred, a failure event is created and stored in the failure event repository.

5.4 Consistency Checks and Property Monitoring

The two types of failure detection tests performed directly by the failure detection manager – consistency checks and property monitoring checks – are not orthogonal. Indeed, a consistency check ultimately reduces to a comparison of the value of a certain variable against a certain threshold. Hence, consistency checks could be subsumed under the more general category of property monitoring checks.

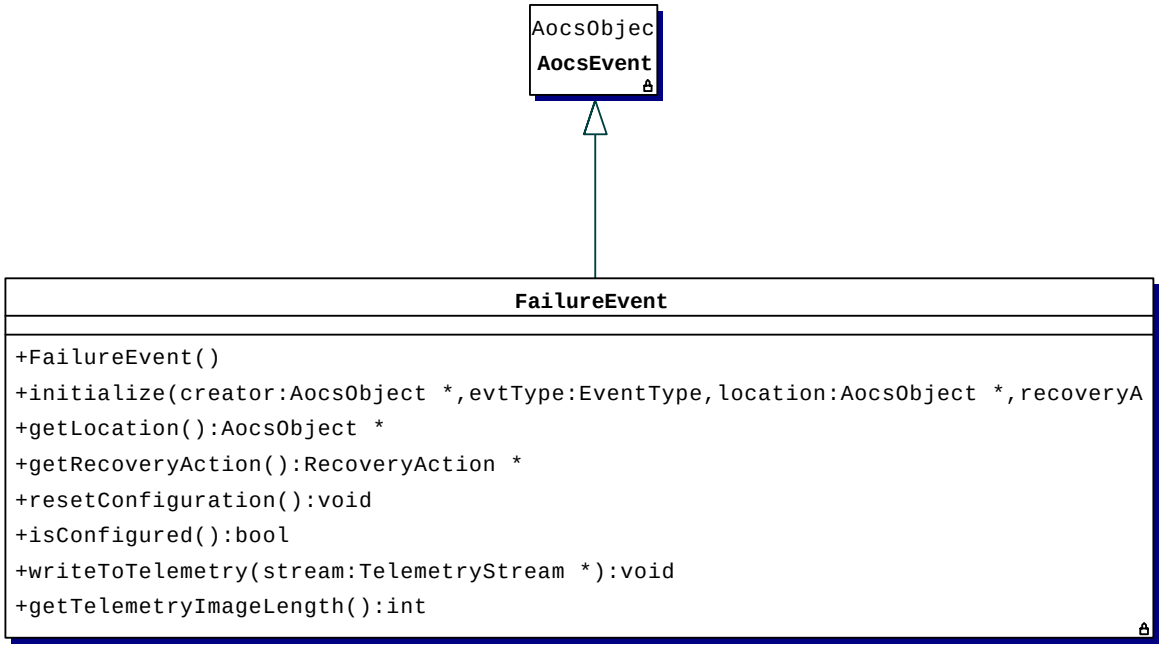
The two types of tests are, however, conceptually separate. The criterion for separating them lies in where the knowledge required to carry out the check is located. In the case of a consistency check, the knowledge to carry out the test is located *inside* the object to be tested. In the case of monitoring checks, this knowledge is located *outside.*

6 FAILURE EVENTS

Detections of failures are reported as failure events in the failure event repository. Failure events are implemented as instances of class `FailureEvent` which is itself derived from the generic `AocsEvent` class.

Failure events are stored in dedicated a event repository instantiated from class `FailureEventRepository`.

The `AocsEvent` class is defined as follows:



Thus, failure events add the following attributes to those defined by the base class `AocsEvent`:

- `location` : the object where the failure was detected.
- `recoveryAction`: the recovery action associated to the failure

Creation of failure events is usually done indirectly by calling method `reportFailure` that class `AocsObject` offers to all its derived classes. This operation is provided to allow all objects – not just the failure detection manager – to report failures (see also section 5.1).

6.1 The Telemetry Interface

Failure events are telemetry objects because they (indirectly, through `AocsEvent`) inherit from `AocsData` the `telemeterable` interface.

The data sent to the telemetry stream by a failure event in each telemetry mode are summarized in the table:

TM Format	TM Data
Short	none
Normal	none
Long	instance identifier of failure location and recovery action
Debug	same as long TM

6.2 The Reset and Configurable Interface

Failure event objects inherit from `AocsObject` the `Resettable` and `Configurable` interfaces and must therefore implement the corresponding method.

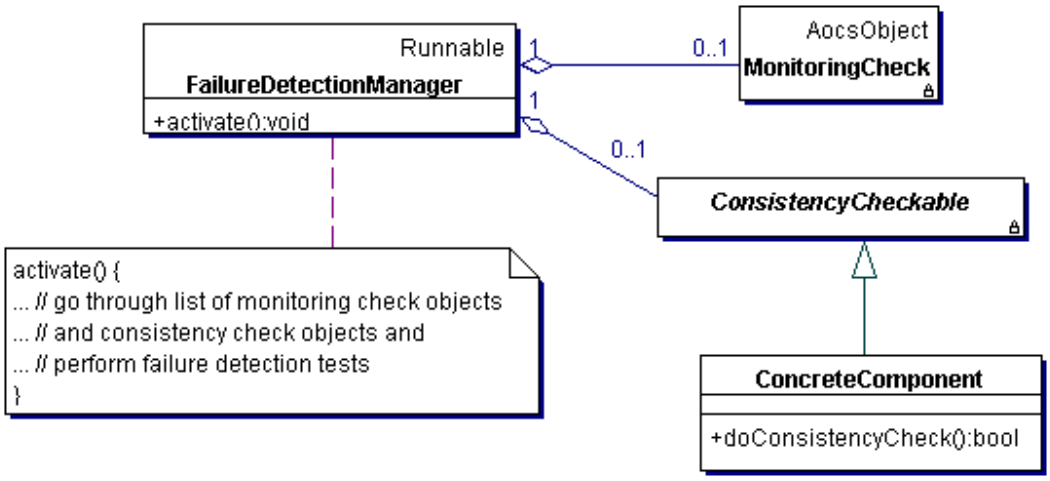
Failure events have no dynamic state associated to them and therefore they do not define a class-specific `reset` method.

Failure events define a class-specific `resetConfiguration` method that resets all event attributes to zero. Method `isConfigured` returns true if the reference to the failure location is non-NULL.

7 THE FAILURE DETECTION DESIGN PATTERN

This design pattern is introduced to address the problem of separating the management of failure detection tests from their implementations. It is based on the manager meta-pattern of RD2.

The pattern is illustrated in the following class diagram:



The failure detection manager maintains lists of monitoring check objects and of consistency checkable objects. When it is activated, it goes through the list and performs the failure detection tests on each item in the lists.

The pseudo-code below shows a basic implementation for the activation cycle of the failure detection manager where the failure checks are performed:

```
ObjectListTemplate<ConsistencyCheckable>* consistencyCheckableList;
ObjectListTemplate<MonitoringCheck>* monitoringCheckList;

. . .

// Perform the consistency checks
for ( all objects ‘obj’ in ‘consistencyCheckableList’ ) do
     if ( !obj->doConsistencyCheck() )
          . . . // check failed, failure has been detected!

// Perform the monitoring checks
for ( all object ‘obj’ in ‘monitoringCheckList’ ) do
     if ( obj->getChangeObject()->checkValue(obj->getPropertyValue()) )
```

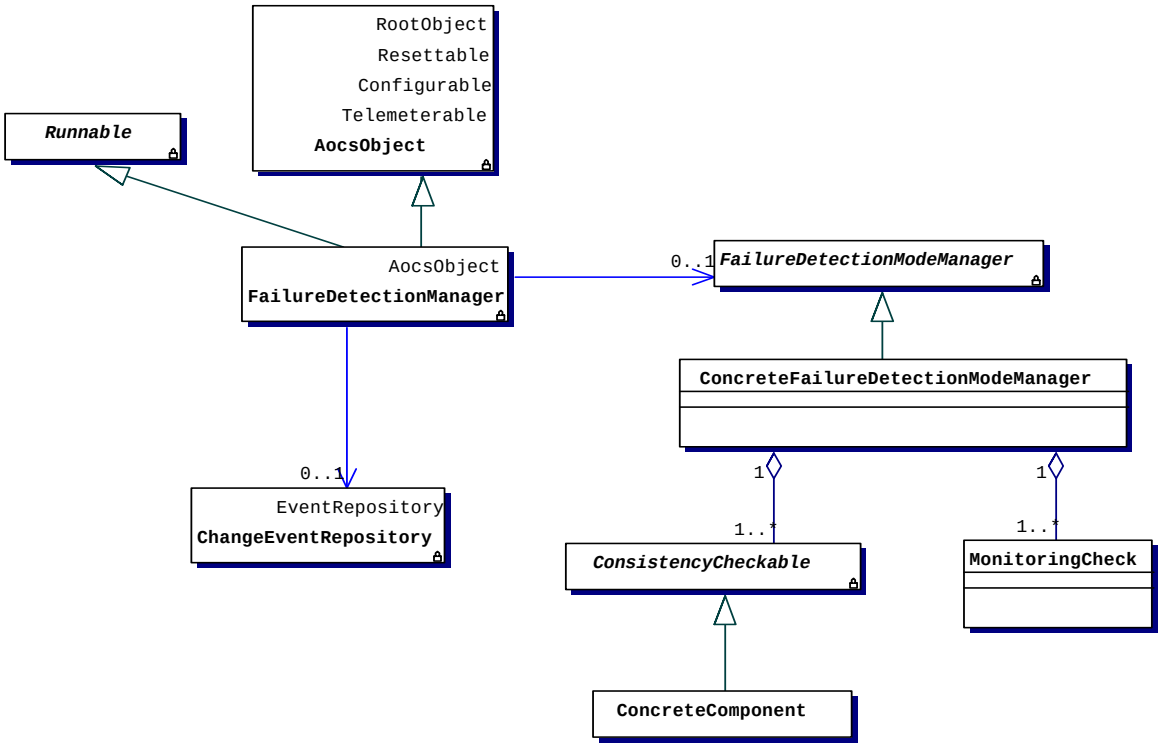
```
{    . . . // check failed, failure has been detected!
}
```

7.1 Instantiation of Failure Detection Pattern

The failure detection pattern is instantiated as follows for the framework:

- the failure detection manager is an active object and its `activate` method is the `run` method declared by interface `Runnable`.
- If failures are found, they are reported as failure events in the failure event repository.
- If a monitoring check fails, then a property change event is created.
- In most cases, the list of components to be subjected to consistency checks and the list of monitoring check objects depends on operational conditions. This is taken into account by making the failure detection manager mode-dependent. The failure detection mode manager then manages two strategies corresponding to the list of consistency checkable objects and to the list of monitoring check objects.

The class diagram for the instantiated failure detection pattern is:



The mode manager is characterized by an abstract interface as discussed in greater detail in section 7.3.

The failure detection manager does not have an explicit link to the failure event repository because it can use the failure reporting service offered by its `AocsObject` base class.

The pseudo-code for the failure detection tests now becomes:

```
class FailureDetectionManager : public AocsObject, public Runnable {

  FailureDetectionModeManager* modeManager;
  ObjectListTemplate<ConsistencyCheckable>* consistencyCheckableList;
  ObjectListTemplate<MonitoringCheck>* monitoringCheckList;

public :

  void run(AocsTime t) {

     // Retrieve the list of objects to be checked in this cycle
     consistencyCheckableList=modeManager->get consistencyCheckableList();
     monitoringCheckList=modeManager->get monitoringCheckList();

     // Perform the consistency checks
     for ( all objects obj in the consistencyCheckableList ) do
          if ( !obj->doConsistencyCheck() )
               . . . // check failed, create failure event

     // Perform the monitoring checks
     for ( all object obj in the monitoringCheckList ) do
          if ( obj->getChangeObject()->checkValue(obj->getPropertyValue()) )
          {    . . . // check failed, create failure event
               . . . // create a property change event
          }

     }
}
```

7.2 The Failure Detection Manager

The failure detection manager class is defined as follows:

The public methods specific to this class (ie. not inherited from base classes) are described in the table:

`setFailureDetectionModeManager, getFailureDetectionModeManager`
Setter and getter methods for the failure detection mode manager.
`setChangeEventRepository, getChangeEventRepository`

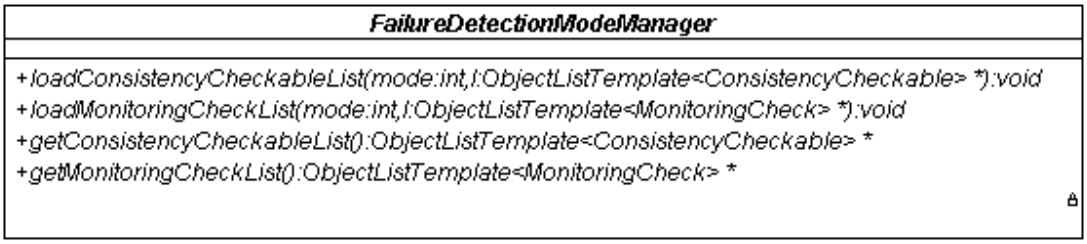
Setter and getter methods for the change event repository.

7.3 The Failure Detection Mode Manager

In general, the type of checks to be performed at a given type depends on the operational conditions of the rest of the AOCS software. This dependency is modelled by endowing the failure detection manager with operational mode.

The mode manager is constructed by instantiating the mode management pattern prescribed in RD3 as shown in the last class diagram of section 7.1.

The failure detection mode manager must be able to supply to the failure detection manager the lists of consistency checkable objects and monitoring checks. Its interface is accordingly defined as follows:



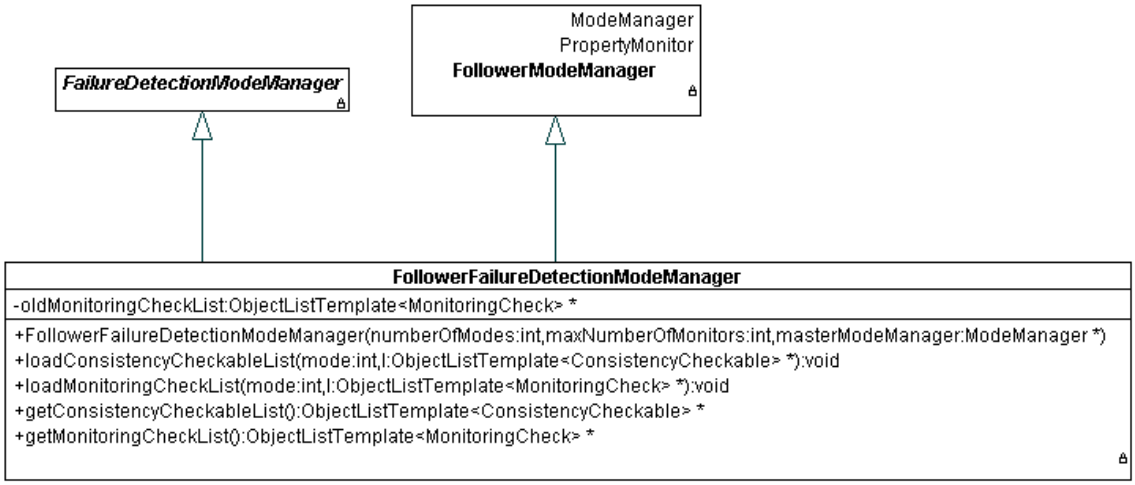
The semantics of the operations defined by this interface are summarized in the following table:

`getConsistencyCheckableList()`
This method is called by the failure detection manager to retrieve the list of objects to be subjected to a consistency check in the current activation cycle.
`getMonitoringCheckList()`
This method is called by the failure detection manager to retrieve the list of monitoring check objects to be used in the current activation cycle.
`loadConsistencyCheckableList (int i, ObjectListTemplate<ConsistencyCheckable>* ccList)`
This method is used to configure the failure detection mode manager. It associates the consistency checkable list `ccList` to operational mode i.
`loadMonitoringCheckList (int i, ObjectListTemplate<MonitoringCheck>*`

`mcList)`
This method is used to configure the failure detection mode manager. It associates the monitoring check list `mcList` to operational mode i.

Concrete failure detection mode managers are defined by the mechanism that they use to decide which particular consistency checkable and monitoring check lists should be returned at any given point in time.

The prototype framework provides a default failure detection mode manager that is based on the follower mode manager. This default failure detection mode manager is instantiated from the following class `FollowerFailureDetectionModeManager`:



Thus, the default failure detection mode manager uses the services offered by the generic follower mode manager component exported by the operational mode framelet.

In a typical configuration, this mode manager would be slaved to the AOCS mission mode manager.

This failure detection mode manager also performs hard-coded mode change actions: when there a mode transition, all the change objects in the monitoring check list that becomes active are reset. This is advisable because change objects normally have an internal state.

7.4 Failure Detection Manager Telemetry Interface

The telemetry manager is itself a telemetry object because it inherits (indirectly, through `AocsObject`) the `telemeterable` interface.

The data sent to the telemetry stream by a core mode manager in each telemetry mode are summarized in the table:

TM Format	TM Data
Short	none
Normal	instance ID of current consistency checkable and monitoring check lists
Long	same as normal TM
Debug	long TM + instance ID of failure mode manager

7.5 The Failure Detection Manager Reset and Configurable Interfaces

The failure detection manager inherits from `AocsObject` the `Resettable` and `Configurable` interfaces and must therefore implement the corresponding method.

Failure detection managers have no internal state and therefore they do not provided a class-specific implementation of method `reset`.

A call to method `resetConfiguration` unloads the following plug-in components: the failure detection mode manager, the change event repository.

Method `isConfigured` returns true if both the failure detection mode manager and the change event repository have been loaded.

8 FRAMELET HOT-SPOTS

This section classifies the framelet hot-spots defined in the previous sections of this document. The classification is as described in RD4.

8.1 Failure Detection Mode Manager Plug-In

Name: Failure Detection Mode Manager **Plug-In**
Visibility Level: framework-level
Adaptation Time: run-time
Adaptation Method: plug-in component in `FailureDetectionManager` class (method `setFailureDetectionModeManager`)
Pre-defined Options: `FollowerFailureDetectionModeManager` component exported by this framelet.
Related Hot-Spots: none
Description Failure detection managers need a mode manager to supply them with the list of consistency checkable and monitoring check objects. This hot-spot allows the failure detection mode manager to be loaded in the failure detection manager.

8.2 Consistency Checkable Hot-Spot

Name: Consistency Checkable Hot-Spot
Visibility Level: framelet-level
Adaptation Time: compile-time
Adaptation Method: virtual method in `ConsistencyCheckable` interface
Pre-defined Options: none
Related Hot-Spots: none
Description

The implementation of interface `ConsistencyCheckable` defines how consistency checks are performed on selected classes of AOCS objects.

8.3 Change Event Repository Plug-In

Name: Change Event Repository **Plug-In**
Visibility Level: framelet-level
Adaptation Time: run-time
Adaptation Method: plug-in component in `FailureDetectionManager` class (method `setChangeEventRepository`)
Pre-defined Options: default `ChangeEventRepository` component exported by the inter-component communication framelet.
Related Hot-Spots: none
Description The failure detection manager performs monitoring checks where the value of a property is checked for a certain type of change. The occurrence of a change in a property has to be reported both as a failure (it is a failure of a monitoring check) and as a change event (a property has undergone a change). Hence, the failure detection manager needs a reference to the change event repository to create and store the change event.

8.4 Monitoring Check Hot-Spot

Name: Monitoring Check Hot-Spot
Visibility Level: framework-level
Adaptation Time: compile-time
Adaptation Method: instantiation and definition of monitoring check objects (instances of class `MonitoringCheck`).
Pre-defined Options: none
Related Hot-Spots: none

Description The failure detection manager performs systematic monitoring of the values of properties. A property monitoring check is encapsulated in an object of class `MonitoringCheck`. For each property that the application developer wishes to be at least potentially monitorable, a corresponding `MonitoringCheck` object must be defined and initialized.

8.5 Consistency Checkable List Plug-In

Name: Consistency Checkable List Plug-In
Visibility Level: framework-level
Adaptation Time: run-time
Adaptation Method: plug-in component in `FailureDetectionModeManager` class (method `setConsistencyCheckableList`).
Pre-defined Options: none
Related Hot-Spots: Monitoring Check List Plug-In
Description The failure detection mode manager maintains a list of consistency checkable objects. This hot-spot defines the point where a new list is loaded into a failure detection mode manager.

8.6 Monitoring Check List Plug-In

Name: Monitoring Check List Plug-In
Visibility Level: framework-level
Adaptation Time: run-time
Adaptation Method: plug-in component in `FailureDetectionModeManager` class (method `setMonitoringCheckList`).
Pre-defined Options: none
Related Hot-Spots: Consistency Checkable List Plug-In

Description

The failure detection mode manager maintains a list of monitoring check objects. This hot-spot defines the point where a new list is loaded into a failure detection mode manager.