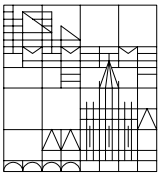


University of Constance
Dept. of Computer Science

SYSTEM 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 system management framelet. This framelet defines a component to perform system management tasks. The framelet enhances reusability because it decouples the management of the system management functions from their implementation.

Written By: A. Pasetti

Date: 30 April 2002

Issue: 2.1

Reference: SWE/99/AOCS/021

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), *Telemetry Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/003

RD4 A. Pasetti (2000), *Failure Detection Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/010

RD5 A. Pasetti (2000), *Reconfiguration Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/015

RD6 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 *system 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 a component to handle system management functions. It enhances reusability because it decouples the *management of the system management functions* from their *implementation*.

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 section dealing with system management.

The architecture proposed here follows the general concept outlined 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.

– In the C++ version of the framework, reconfigurable objects notify the system manager every time their configuration changes by calling method `configurationStateChange` exposed by class `SystemManager`. In the Java version of the framework the notification of configuration changes is done through events. Reconfigurable objects act as sources of reconfiguration events. The system manager is a listener to reconfiguration events. The system manager must register with reconfigurable

objects during the initialization phase and will then be automatically notified whenever there are configuration changes.

3.3 Notation

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

UML class diagrams were obtained with the Together tool (version 4.0).

4 FRAMELET CONSTRUCTS

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

SYTEM MANAGEMENT FRAMELET
Framelet Design Patterns
System Management Pattern : design pattern to systematically perform the same operations on a target set of objects *Memento Pattern* : design pattern to preserve configuration information across system resets. This is a standard design pattern taken from RD1.
Framelet Interfaces
`Resettable` : interface to declare object reset services `Configurable` : interface to declare object configuration services
Framelet Core Components
`SystemManager` : system management component `RootObject` : base class for all objects in the framework `AocsObject` : base class for all non-trivial objects in the framework

5 SYSTEM MANAGEMENT FUNCTIONS

A *system management function* is a function that is performed systematically on all AOCS objects present in the AOCS software at a given time. An *AOCS object* is a component that is derived from the basic class `AocsObject`. The *system manager* is the component responsible for performing system management functions.

Two system management functions are foreseen by the AOCS prototype:

- *System Reset*

  A system reset causes the internal state of all AOCS objects to be reset to a default state. Note that, with this definition, a system reset does not entail a system reboot.

- *System Configuration Check*

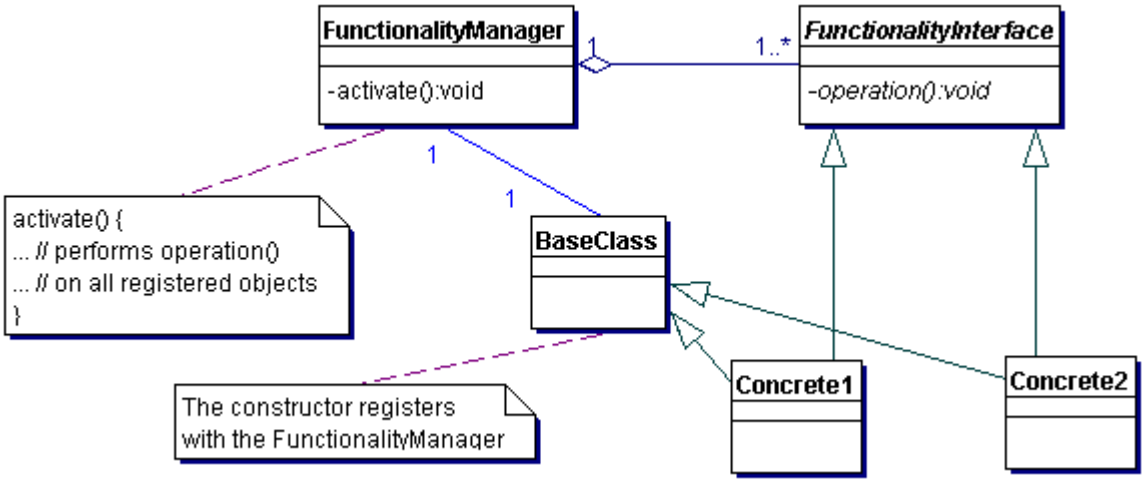
  A configuration check causes the configuration of all AOCS objects to be checked. The system configuration check function returns `true` if all AOCS objects are configured and returns `false` otherwise.

These two functions are described in greater detail in sections 5.3 and 5.4.

5.1 The System Manager Design Pattern

The system manager design pattern is introduced to address the problem of systematically performing the same set of operations on a target set of objects. The system manager design pattern is obtained by instantiating the manager meta-pattern.

The structure of the system manager design pattern is shown in the following UML diagram:



The functionality interface encapsulates the operations to be performed on the target set of objects. This interface is to be implemented by all objects in the target set. Additionally, all objects in the target set are to be derived from a single base class whose constructor registers with the functionality manager. Thus, the functionality manager is provided with a list of all the objects in the target set. Deregistration is not required because objects in the target set are assumed never to be destroyed.

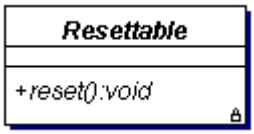
5.2 Instantiation of System Manager Pattern

In the AOCS framework, the system manager design pattern is instantiated twice, once for the system reset function and once for the system configuration check function, as follows:

- The functionality manager is the `SystemManager` component
- The base class is `AocsObject`. This means that the set of objects upon which the system manager acts is the set of AOCS objects.
- The abstract interfaces associated to the system reset and system configuration checks functions are `Resettable` and `Configurable`, respectively
- The functionality interfaces `Resettable` and `Configurable` are implemented directly by `AocsObject`. This ensures that they are implemented by all objects upon which the system manager is required to act (ie by all AOCS objects).

5.3 The System Reset Function

The `Resettable` interface associated to the system reset function is defined as follows:



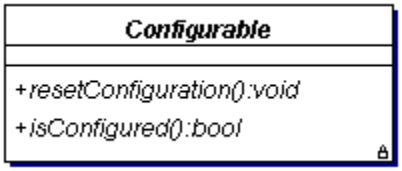
This interface is inherited by all AOCS objects that must therefore provide an implementation for method `reset`(). A call to method `reset` causes the state of the object to be reset to its default value. By *state* here it is meant the set of internal attributes that are regularly updated as a result of the object performing its task.

Calling the system reset function causes the reset method of all existing AOCS objects to be called in sequence. The order in which the `reset` method is called is not defined.

The system reset function is intended primarily for failure recovery. It is milder than a system reboot because it does not destroy the configuration of the AOCS software.

5.4 The System Configuration Check Function

The `Configurable` interface associated to the configuration check function is defined as follows:



This interface is inherited by all AOCS objects that must therefore provide an implementation for methods `resetConfiguration`() and `isConfigured`().

A call to `resetConfiguration` causes the configuration information of the object to be destroyed. The configuration data include those attributes of an object that are normally set during the application initialization to configure the object. These attributes are used to tune the object's behaviour and are modified during the object's lifetime on an occasional basis only.

Method `isConfigured` returns true if the object is configured, ie. if all its configuration attributes have been defined and have non-null values.

Calling the system configuration reset function causes the `isConfigured` method to be called in sequence on all existing AOCS objects to be called in sequence. The order in which the method is called is not defined. The system configuration check function returns true if all the `isConfigured` methods returned true.

The system configuration check is provided to be called at the end of the AOCS software initialization phase to verify that the system is properly configured.

6 BASIC CLASSES

The system manager was introduced to perform operations systematically on all AOCS objects in the AOCS software. Uniform treatment of a large class of objects is best done when they are all derived from a common base class. The basic object classes were introduced for this purpose.

A *basic class* is a class that serves as a base class to a large number of framework classes. Two basic classes are present in the AOCS framework:

- `RootObject`

  This is the base class for all objects in the AOCS framework. It defines the object identifiers.

- `AocsObject`

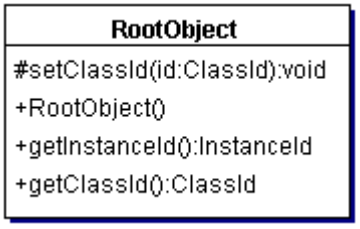
  This is the base class for all non-trivial objects in the AOCS framework. It adds to `RootObject` services for object reset and configuration and for telemetry management and for error reporting.

The two basic class are described in greater detail in the next subsections.

The system manager operates on the `AocsObject` class.

6.1 The `RootObject` Class

Class RootObject defines two read-only attributes: the *instance identifier* and the *class identifiers*. Its class diagram is:

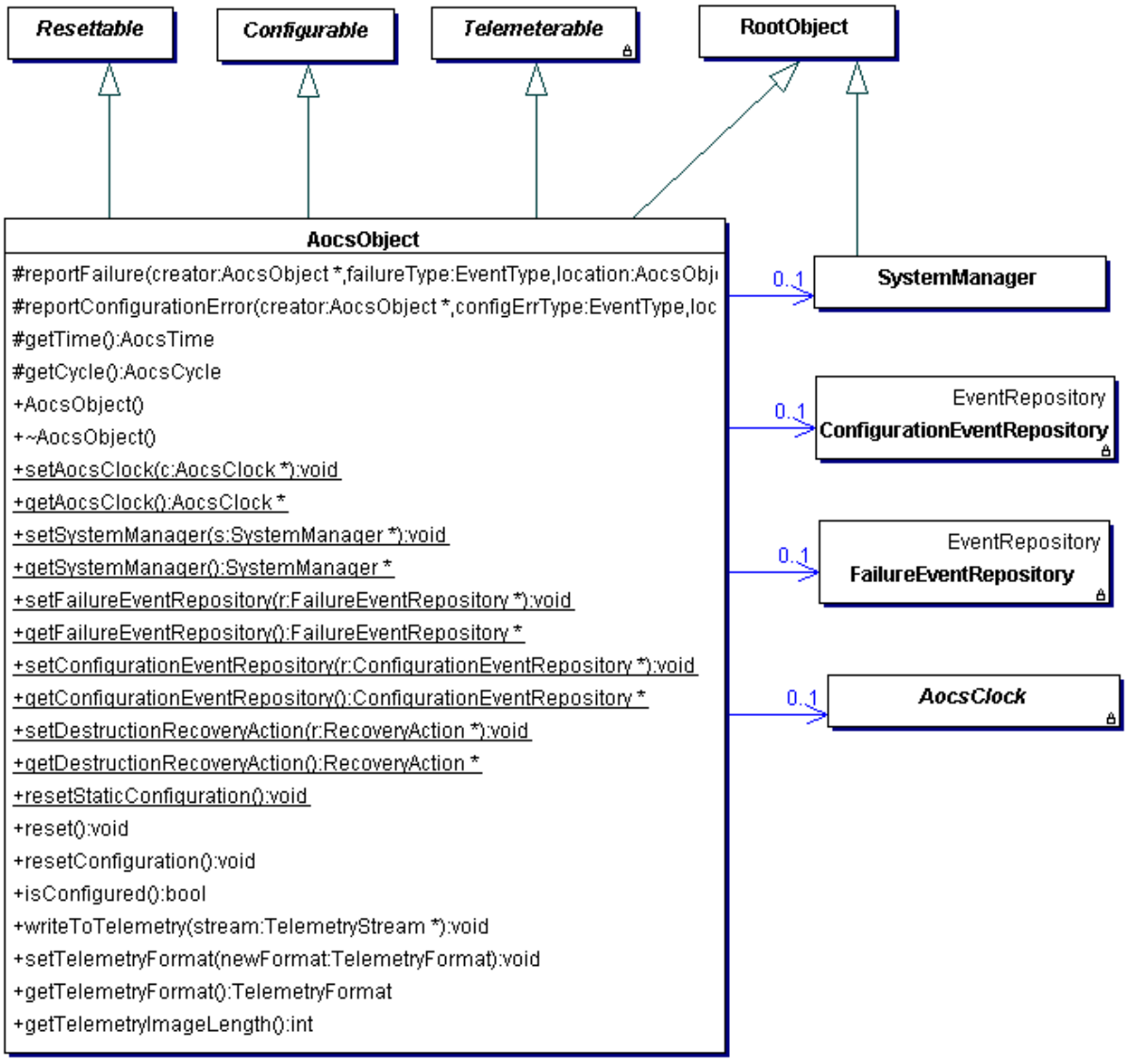


The class defines two getter methods for the instance and class identifiers. The instance and class identifiers are set by the constructor and cannot be changed.

Class `RootObject` allows uniform treatment of all objects in the AOCS software and ensures that all objects and all classes can be uniquely identified.

6.2 The `AocsObject` Class

Class `AocsObject` is directly derived from `RootObject`. It acts as a base class for all non-trivial objects in the AOCS software. Its definition is:



Objects derived from class `AocsObject` are called *AOCS Objects.*

Class `AocsObject` is designed to gather together services that apply to all non-trivial objects in the framework and to make them available through a simple interface. Some such services are *internal* in the sense that they are encapsulated in protected methods that can be called only by the children objects. Internal services are implemented by plug-in components.

Other services are *external* in the sense that they are encapsulated in public interfaces that are implemented by class `AocsObject`. The external services can be called by any object.

The following internal services are offered:

- *Time Recovery*

  `AocsObject` holds a reference to the `AocsClock` interface that defines the AOCS clock. Through it, it can recover the time and make it available to its children object through protected method `getTime`.

- *Failure Reporting*

  `AocsObject` offers a protected method, `reportFailure`, through which failures can be reported. Failure reports are transformed into failure events that are stored in the failure event repository to which `AocsObject` has a reference. Failure events are described in RD4.

- *Configuration Error Reporting*

  `AocsObject` offers a protected method, `reportConfigurationError`, through which configuration errors can be reported. Configuration error reports are transformed into configuration events that are stored in the configuration event repository to which `AocsObject` has a reference.

The external services are encapsulated in the interfaces that are implemented by `AocsObject` and inherited by all its children:

- *Interface* `Telemeterable`

  Implementation of this interface ensures that all AOCS objects can potentially be included in telemetry. Interface `Telemeterable` is described in RD3.

- *Interface* `Resettable`

  Implementation of this interface ensures that all AOCS objects are able to perform a state reset operation that brings their internal state to a default initial value. Interface `Resettable` is described in section 5.3.

- *Interface* `Configurable`

  Implementation of this interface ensures that all AOCS objects can perform a configuration reset and can check whether they have been configured. Interface `Configurable` is described in section 5.4.

AOCS objects are intended to be created during initialization and never to be destroyed. To ensure that this is the case, the `AocsObject` destructor generates an error if it is called. Note that this means that AOCS objects cannot be created on the stack: they can only be created statically.

Since AOCS objects are created statically and never destroyed, it is safe to use pointers to them. A design rule in the AOCS framework dictates that pointers can only be used on objects of type `AocsObject`.

6.3 Default Interface Implementations

Class `AocsObject` is concrete and provides simple default implementations for the methods declared by its interfaces.

Method `writeToTelemetry` from `AocsObject` writes the following data to the telemetry stream as a function of the telemetry format:

TM Format	TM Data
Short	instance and class identifiers
Normal	same as short TM
Long	same as normal TM
Debug	long TM + instance identifiers of AOCS clock and failure and reconfiguration event repositories

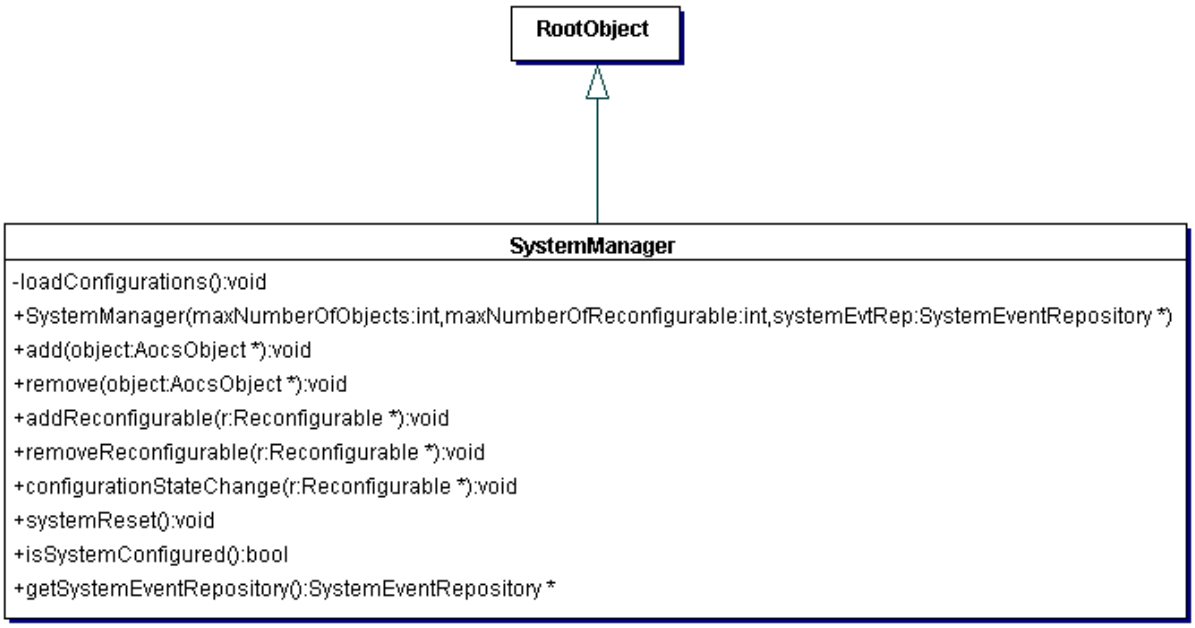
Method `reset` returns without taking any action

Method `resetConfiguration` resets the telemetry format to `normalTm`.

Method `isConfigured` returns true is all the plug-in components for class `AocsObject` have been loaded.

7 THE SYSTEM MANAGER

The system manager is the component responsible for performing the system management functions. Its class diagram is:



Note that the system manager is derived from `RootObject` rather than from `AocsObject` as is the case for all other non-trivial objects in the framework. This is because the primary function of the system manager is to perform system management operations on AOCS objects and therefore the system manager should not itself be an AOCS object.

The semantics of the public methods exposed by the system manager class is:

`SystemManager(n,m,rep)`
Constructor that sets the maximum number n of AOCS objects that can be present in the AOCS application, the maximum number m of reconfiguration managers that can be present in the AOCS application, and the system event repository rep.
`add(), remove()`
The system manager maintains an internal list of all AOCS objects created in the AOCS application. These methods allows an item to be added and removed from this list. See section **Error! Reference source not found.**.

`addReconfigurable(), removeReconfigurable()`
The system manager maintains an internal list of all reconfiguration managers created in the AOCS application. These methods allows an item to be added and removed from this list. See section 7.1.
`configurationStateChange()`
The system manager maintains a record of the current configuration of each reconfiguration manager in the AOCS application. This is a call-back method that is invoked by a reconfiguration manager whenever it updates the configuration of the reconfiguration group it manages. See section 7.1.
`systemReset()`
This method implements the system reset function. A call to this method causes all AOCS objects in the AOCS application to be reset. See section 5.3
`isSystemConfigured()`
This method implements the system configuration check function. See section 5.4
`getSystemEventRepository()`
Getter method for the system event repository.

In order to implement the system reset function (see section 5.3) and the configuration check function (see section 5.4), the system manager internally maintains a list of all AOCS objects created in the AOCS application. The system manager pattern guarantees that this list is completed. The pattern requires that class `AocsObject` holds a reference to the system manager and that its constructor call the `add` method on the system manager:

```
AocsObject::AocsObject()
{
     . . .
     if (systemManager!=NULL)
          systemManager->add(this);
     . . .
}
```

Since the `AocsObject` constructor is implicitly called by the constructor of all AOCS objects in an AOCS application, this guarantees that the all AOCS objects are automatically registered with the system manager.

The system manager also maintains a list of all reconfiguration managers (see RD5) created in the AOCS application. The same design pattern is used to ensure the completeness of this list by having the constructor of each reconfiguration manager call method `addReconfigurable` on the system manager:

```
ConcreteReconfigurationManager:: ConcreteReconfigurationManager(. . .)
{
     . . .
     AocsObject::getSystemManager()->addReconfigurable(this);
     . . .
}
```

7.1 Implementation of the System Reset Function

When the system reset method is called, the system manager goes through the list of registered AOCS objects and calls method `reset` on each.

There is some information that should be preserved across resets. This in particular concerns configuration information. Unit reconfigurations are managed by dedicated objects called *reconfiguration managers* (see RD5). Syntactically, reconfiguration managers are objects of type `Reconfigurable`.

Reconfiguration managers can return an object of type `ConfigurationState`. The *memento design pattern* from RD1 is used to store the configuration of their reconfiguration group.

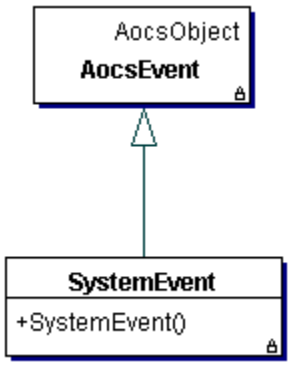
The system manager therefore holds a list of all reconfiguration managers present in the AOCS application and maintains an associated list of their current configuration. After a system reset has been performed, the configuration of all reconfiguration managers is restored to the value it had prior to the reset action.

In order to ensure that the system manager is kept informed of changes in the configuration of the reconfiguration managers, a call-back method `configurationStateChange` is provided that reconfiguration managers should call whenever they change the configuration of their reconfiguration.

In some implementations, configuration information could be stored in non-volatile memory to be preserved across system reboots. It is for this reason that a call-back method is used to keep the configuration record in the system manager up-to-date. In an alternative implementation, the reset method itself, prior to commanding the reset, collects the configuration information from all registered reconfiguration managers. This obviously, would not ensure that the system manager is up-to-date in case of a system reboot.

8 SYSTEM EVENTS

When the system manager encounters error situations or performs a system reset, it records the fact as a *system event*. System events are instances of class `SystemEvent`:



Thus, class `SystemEvent` adds neither attributes nor methods to its base class. It is introduced uniquely as a placeholder to characterize from a syntactic point of view system events.

9 FRAMELET HOT-SPOTS

This section classifies the framelet hot-spots defined in the previous sections of this document. The classification is as described in RD6.

9.1 Resettable Hot-Spot

Name: Resettable Hot-Spot
Visibility Level: framework –level
Adaptation Time: compile-time
Adaptation Method: implementation of interface `Resettable`
Pre-defined Options: default implementation provided by class `AocsObject` (see section 6.3)
Related Hot-Spots: none
Description AOCS objects must provide an implementation of method `reset` to reset their internal state.

9.2 Configurable Hot-Spot

Name: Configurable Hot-Spot
Visibility Level: framework –level
Adaptation Time: compile-time
Adaptation Method: implementation of interface `Configurable`
Pre-defined Options: default implementation provided by class `AocsObject` (see section 6.3)
Related Hot-Spots: none
Description AOCS objects must provide an implementation of method `resetConfiguration` to destroy their internal configuration and of method `isConfigured` to check that they have been configured.