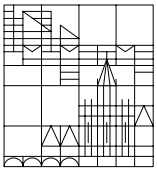

University of Constance
Department of Computer Science

AOCS FRAMEWORK - PROTOTYPE DEFINITION

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 was developed in full at the architectural design level but only a representative subset of it will be implemented at the prototype level. This document defines the part of the framework that will be implemented in the framework prototype.

Written By: A. Pasetti

Date: 30 April 2002

Issue: 2.1

Reference: SWE/99/AOCS/019

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 Deleted

RD4 A. Pasetti (2000), *Methodological Issues*, AOCS Framework Document ref. SWE/99/AOCS/018

RD5 A. Pasetti (2000), *Inter-Component Communication Framelet* AOCS Framework Document ref. SWE/99/AOCS/005

RD6 A. Pasetti (2000), *Object Monitoring Framelet*, AOCS Framework Document ref. SWE/99/AOCS/008

RD7 A. Pasetti (2000), *Data Processing Framelet*, AOCS Framework Document ref. SWE/99/AOCS/006

RD8 A. Pasetti (2000), *AOCS Unit Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/017

RD9 A. Pasetti (2000), *Reconfiguration Management Framelet* AOCS Framework Document ref. SWE/99/AOCS/015

RD10 A. Pasetti (2000), *Operational Mode Management Framelet* AOCS Framework Document ref. SWE/99/AOCS/009

RD11 T. Brown, A. Pasetti (2000), *Manoeuvre Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/012

RD12 A. Pasetti (2000), *Failure Detection Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/010

RD13 A. Pasetti (2000), *System Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/021

RD14 A. Pasetti (2000), *Failure Recovery Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/011

RD15 A. Pasetti (2000), *Telemetry Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/003

RD16 A. Pasetti (2000), *Telecommand Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/014

RD17 A. Pasetti, T. Brown (2000), *Controller Management Framelet*, AOCS Framework Document ref. SWE/99/AOCS/016

RD18 A. Pasetti (2000), *AOCS Prototype Definition*, AOCS Framework Document ref. SWE/99/AOCS/020

RD19 *MACS Bus Handbook*

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
MIMO	Multi-Input-Multi-Output
NM	Normal Mode
NTT	Non-Time-Tagged
OBDH	On-Board Data Handling system (aka as OBDS)
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
SISO	Single-Input-Sinle-Output
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 *prototype AOCS framework*. The prototype AOCS framework is a partial implementation of the AOCS framework. The prototype framework implements a representative subset of the constructs defined by the full framework. Its purpose is to serve as a proof-of-concept demonstrator for the full framework.

3.1 Context

The context for the definition of the prototype framework is the architectural design of the full framework. This is presented at system concept definition level in RD2 and at framelet concept and at framelet architectural definition level in RD5 to RD17.

The prototype framework was intended for use at the end of the study to implement a *prototype AOCS*. The decision as to which elements of the framework to include in the prototype framework was made with a view to the implementation of the prototype AOCS. The expected features of the prototype AOCS are described in section 5.

The intention at the beginning of the study was to provide a prototype implementation of the framework that would implement a subset of the framelets in full. The concept that is adopted here is different being based on a prototype framework that offers a partial implementation of all framelets.

3.2 Document Structure

The AOCS framework was designed as a collection of framelets. The definition of the prototype framework is also made at the framelet level. Framelets are defined in terms of the architectural constructs they export. They can export three types of constructs: design patterns, interfaces and components. In some cases interfaces are implemented as abstract classes (ie. they include some basic implementation). Exported components may be default implementations of the interfaces or abstract classes.

For each framelet, this document provides a list of the architectural constructs exported by the framelet and of those which will be implemented for the prototype framework.

A justification of the selection of which features to include and which to leave out is also included.

4 HARDWARE INTERFACES

As mentioned above, the components included in the prototype framework are developed with a view to their use in the assembly of the AOCS prototype. Hence the definition of some of these components requires a precise definition of the external interfaces of the prototype AOCS. This is notably the case for the following items:

- definition of the external unit interfaces
- definition of telemetry interface
- definition of telecommand interface

The assumption concerning these interfaces are described in the following subsection.

4.1 External Unit Interface

AOCS unit proxy objects delegate interaction with the unit interface to a lower level component characterized by the implementation of the `AocsUnitHardware` interface. The prototype AOCS is MACS-based: all communications between the AOCS computer and the external AOCS unit take place over the MACS bus. The prototype framework accordingly provides a component implementing the `AocsUnitHardware` to interface with a MACS controller using the MACS-TC protocol (see RD19).

The assumed interface with the MACS controller is as follows. Communication with the MACS controller is through three 16-bit registers:

- MACS_CS: control / status register
- MACS_INSTR: instruction register
- MACS_DATA: data register

All three registers are read / write memory-mapped registers that are accessed by the processor like any other word-length memory location. The actual memory location is determined by external hardware. The address of the three registers are passed as constructor parameters to the MACS interface component.

The layout of the MACS_CS register is as follows:

- bit 0: writing 1 to this bit causes the instruction currently in the MACS_INSTR register to be sent to the bus
- bit 1: this bit is set to 1 when processing of the last instruction by the MACS controller has been completed. This includes emission on the bus of any data words associated to the

instruction or latching into MACS_DATA of any data word received as a result of the instruction being emitted.

- bit 2: this bit is set to 1 if no acknowledge was received for the last instruction or for any data words associated to it.
- bit 3 : this bit is set to 1 if a parity error was detected by the MACS controller while processing the last instruction or any data words associated to it.
- bit 4 : this bit is set to 1 if the error bit was set by the MACS controller while processing the last instruction or any data words associated to it.
- bit 5 : writing 1 to this bit causes the MACS controller to be reset aborting any on-going transaction.

Register MACS_INSTR contains the instruction to be emitted on the bus with the following layout:

- bits 0-2 : extension
- bits 3-7 : destination address
- bits 8-12 : destination sub-address
- bits 13-15 : instruction code

Register MACS_DATA contains the data word that is associated to the instruction in MACS_INSTR.

The MACS interface component supplied by the prototype framework assumes that no interrupts are associated to the operation of the MACS controller. Checking that a bus transaction has been successfully completed must therefore be done by polling the MACS_CS register.

4.2 Telemetry Interface

The telemetry interface assumptions have an impact on the implementation of the `TelemetryStream` interface in the telemetry management framelet. The framework prototype offers a component – `DmaTelemetryStream` - implementing this interface. This component assumes a DMA-based telemetry interface. This means that the telemetry data are assumed to be forwarded to the central on-board computer by a dedicated hardware interface that collects them from a pre-defined memory area in the AOCS computer. This pre-defined memory area is called the *DMA buffer*. It is defined by its start address and by its length. Both start address and buffer length are settable parameters of class `DmaTelemetryStream`.

No synchronization mechanism is assumed between the AOCS software and the hardware telemetry interface. When the latter is triggered, it simply collects whatever happens to be in the DMA buffer.

4.3 Telecommand Interface

The telecommand interface is based on the following assumptions:

- Telecommands are deposited by a DMA mechanism operating independently of the AOCS processor to a predefined memory area in the address space of the AOCS processor.
- Arrival of a new telecommand is signaled by an interrupt called the *telecommand interrupt*.
- The telecommand interrupt deposits the telecommands in sequence in a memory area called the *telecommand buffer*. Its size is TC_BUFFER_SIZE bytes and its start address can be retrieved from `DmaTelecommandLoader`.
- The DMA telecommand loader is activated periodically to process the telecommands in the telecommand buffer and load them as instances of class `Telecommand` into the telecommand manager.

The prototype framework does not supply the telecommand interrupt servicing routine. This is because the ERC32 simulator where the prototype AOCS is tested cannot simulate the presence of interrupts. It only supplies the DMA telecommand loader as an instance of class `DmaTelecommandLoader`.

5 THE AOCS PROTOTYPE

The prototype framework was used to implement a *prototype AOCS*. The AOCS prototype is a simplified AOCS software which is implemented using the constructs offered by the AOCS prototype framework. The AOCS prototype thus serves as a test bed for the AOCS prototype framework.

The AOCS prototype is not intended to be representative of any real AOCS. Its interest lies simply in the extent to which it allows the functionalities of the AOCS prototype framework to be exercised and the constructs exported by it to be utilized.

The prototype AOCS is described in RD18.

The features implemented by the prototype framework are, to some extent, dictated by the need to construct the AOCS prototype.

6 FRAMEWORK IMPLEMENTATION STATUS

The AOCS framework is divided into 14 framelets. Each framelets is characterized by the constructs it exports. Exported constructs are listed in a table at the beginning of each framelet description document. Framelet constructs can be of three types:

- design pattern
- abstract interfaces and base classes
- core and default components

The prototype framework does not implement all the constructs defined by the framelets. This section describes, for each framelet, the implementation status of its constructs.

7 SYSTEM MANAGEMENT FRAMELET

The system management framelet defines the following constructs:

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

The system management components are provided in full. Their implementation makes use of both framelet design patterns.

The framelet interfaces are fully implemented by all AOCS objects provided by the prototype framework.

8 OBJECT MONITORING FRAMELET

The object monitoring framelet defines the following constructs:

OBJECT MONITORING FRAMELET
Design Patterns
Property Definition Pattern : pattern to define properties in objects and the methods to access them *Additional Properties Pattern :* pattern to add new properties to a component that is already packaged as a binary unit *Direct Monitoring Pattern :* pattern to directly monitor an object's property *Monitoring through Change Notification Pattern :* pattern to implement a notification mechanism when a property changes in a specified manner.
Framelet Interfaces
`ChangeObject` : interface for object encapsulating a type of property change
Framelet Core Components
`Property` : encapsulation of property objects
Framelet Default Components
`SimpleChange` : implementation of interface `ChangeObject` encapsulating a simple change in a property value `OutOfRangeChange` : implementation of interface `ChangeObject` encapsulating an out-of-range change in a property value `DeltaChange` : implementation of interface `ChangeObject` encapsulating a delta change in a property value `SpikeFilteredDeltaChange` : implementation of interface `ChangeObject` encapsulating a delta change in a property value with spike filtering

The prototype framework implements the following attributes of its predefined components as properties:

- the operational mode indicators of mode manager components
- the data items of AOCS data

These properties are built into their host components and hence the pattern to implement additional properties is not used.

These properties can be monitored using both mechanisms offered by the object monitoring framelet. Monitoring with change notification is used in follower mode managers. Direct monitoring is used in monitoring check objects.

Monitoring with property objects and with change notification requires the availability of change objects. The prototype framework therefore pre-defines components implementing some basic kinds of change objects.

The next table summarizes the implementation status of the framelet constructs in the prototype framework:

OBJECT MONITORING FRAMELET
Implemented Design Patterns
Property Definition Pattern : implemented *Additional Properties Pattern :* not implemented *Direct Monitoring Pattern :* implemented *Monitoring through Change Notification Pattern :* implemented
Implemented Framelet Interfaces
`ChangeObject` : implemented in change object components listed below
Framelet Core Components
`Property` : implemented
Implemented Framelet Components
`SimpleChange` : implemented `OutOfRangeChange` : implemented `DeltaChange` : implemented

`SpikeFilteredDeltaChange` : not implemented

9 INTER-COMPONENT COMMUNICATION FRAMELET

The inter-component framelet defines the following interfaces and components:

INTER-COMPONENT COMMUNICATION FRAMELET
Design Patterns
Shared Data Pattern : pattern to exchange data among components using shared data areas *Shared Event Pattern :* pattern to exchange events among components using shared data areas
Framelet Interfaces and Abstract Base Classes
`AocsEvent` : abstract base class for AOCS events `EventRepository` : abstract base class for event repositories `AocsData` : abstract base class for all AOCS data `DataPool` : abstract base class for AOCS data pools
Framelet Core Components
`TelecommandEvent` : telecommand event `ModeEvent` : mode change event `RecoveryEvent` : failure recovery event `FailureEvent` : failure event `ManoeuvreEvent` : manoeuvre event `ChangeEvent` : property change event `ConfigurationEvent` : configuration error event `SystemEvent` : system event `ReconfigurationEvent` : reconfiguration event `TelecommandEventRepository` : telecommand events repository `ModeEventRepository` : mode change events repository `RecoveryEventRepository` : failure recovery events repository `FailureEventRepository` : failure events repository `ManoeuvreEventRepository` : manoeuvre events repository `ChangeEventRepository` : property change events repository `ConfigurationEventRepository` : configuration error events repository `SystemEventRepository` : system events repository

`ReconfigurationEventRepository` : reconfiguration events repository
`Scalar` : scalar data `TwoEulerAngles` : set of two Euler angles `ThreeEulerAngles` : set of three Euler angles Nvector : set of n elements treated as an n-vector
`AttitudeDataPool` : data pool for attitude data
`DataItemRead` : component encapsulating a read-only access to a data item
`DataItemWrite` : component encapsulating a read/write access to a data item

All the constructs exported by the framelet and listed in the above table are implemented in full in the prototype framework.

The two design patterns are used to construct the data pools and the event repositories.

The abstract base classes and interfaces are implemented by the core components.

10 SEQUENTIAL DATA PROCESSING FRAMELET

The data processing framelet defines the following interfaces and components:

DATA PROCESSING FRAMELET
Framelet Interfaces and Abstract Base Classes
`AbstractControlChannel` : interface for control channels
`ControlChannelBlock` : abstract class encapsulating control channel block
`XmathUcbBlock` : abstract class offering an interface to Xmath autocode
Framelet Core Components
`ControlChannelSuperBlock` : container component for a control channel super-block
Framelet Default Components
`P_Block` : control block implementing a proportional transfer function
`I_Block` : control block implementing an integral transfer function
`D_Block` : control block implementing a derivative transfer function
`AdderBlock` : control block to add two inputs
`DifferenceBlock` : control block to take the difference of its two inputs
`LimitBlock` : control block to saturate an input
`PassThruBlock` : control block with unitary transfer function
`SplitterBlock` : control block to split a single input into several identical outputs
`TwoByTwoMatrixBlock` : control block implementing a 2x2 matrix multiplication transfer function
`XmathUcbBlock` : control block embedding a generic UCB routine from the Xmath autocode
`XmathUcbPidBlock` : control block implementing a PID controller (from Xmath autocode)

All the constructs listed in the table are implemented in full in the AOCS framework prototype.

The integral and derivative blocks (I_Block and D_Block) use very simplified algorithms to implement integration and derivation, respectively, and should only be used for testing purposes.

11 AOCS UNIT FRAMELET

The AOCS unit framelet defines the following architectural constructs:

AOCS UNIT FRAMELET
Design Patterns
Fictitious Unit Pattern : pattern to make objects that process unit data look like units
Framelet Interfaces and Abstract Base Classes
`AocsUnitHardware` : interface for objects managing low level exchanges with external units. `UnitInstruction` : interface structure defining a generic protocol for data exchanges with external units `AocsUnitFunctional` : interface for objects representing the functional exchanges between the AOCS software and an AOCS unit (either real or fictitious) `AocsUnitHousekeeping` : interface for objects representing the housekeeping exchanges between the AOCS software and a real (ie. non fictitious) AOCS unit `AocsUnit` : abstract class serving as base class for all objects representing external unit proxies in the AOCS software `TriggerList` : interface for trigger list objects, namely list of units due to be triggered at the same time in the AOCS cycle
Framelet Core Components
`PollingTrigger` : trigger object to perform full data transfer (transaction + refresh cycle) with polling on registered units `UnitTrigger` : trigger object to perform full data transfer (transaction + refresh cycle) without polling on registered units `RefreshTrigger` : trigger object to perform refresh operations on registered units `TrasactionTrigger` : trigger object to perform transaction operations on registered units
Framelet Default Components
`FullTriggerList` : full implementation of interface `TriggerList` `FunctionalTriggerList` : partial implementation of interface `TriggerList` covering only

functional units

`MacsTcController` : implementation of interface `AocsUnitHardware` for a MACS telecom controller

`FssPrototype` : two-axis fine sun sensor AOCS unit for the AOCS prototype

`GyrPrototype` : single-axis gyro AOCS unit for the AOCS prototype

`RwPrototype` : reaction wheel AOCS unit for the AOCS prototype

`SapPrototype` : solar acquisition and propulsion electronics AOCS unit for the AOCS prototype

`TorquingThrusters` : fictitious AOCS unit to command a set of thrusters directly with the torque requests around spacecraft axes

The prototype framework provides a limited number of simplified implementations for unit components. The implementations assume a MACS-based AOCS[1].

The prototype framework provides implementations for the units used in the AOCS prototype (see RD18). The implementation is very simple and does not match the characteristics of any real AOCS units. It is only provided for testing purposes. Note that this implementation requires an exact definition of the hardware interfaces to the external units (see section 4).

Unit triggering in the AOCS prototype is through normal trigger objects (instances of class `UnitTrigger`). Accordingly, this is the only type of trigger objects implemented by the prototype framework.

Only the functional data are modeled in the prototype AOCS and hence `FunctionalTriggerList` is the only trigger list used in the prototype AOCS.

Fictitious units are defined to manage the unit reconfiguration (see also section 12 on the reconfiguration management framelet) and to provide a high-level interface to the thrusters (class `TorquingThrusters`) .

The next table summarizes the implementation status of the framelet constructs in the prototype framework:

[1] MACS is the name of a data bus that is widely used on ESA science satellites.

AOCS UNIT FRAMELET
Implemented Design Patterns
Fictitious Unit Pattern : implemented in reconfiguration managers and TorquingThruste component
Implemented Framelet Interfaces and Abstract Base Classes
`AocsUnitHardware` : implemented in MACS TC hardware unit object `UnitInstruction` : implemented to define MACS bus protocol `AocsUnitFunctional` : implemented in AOCS unit objects listed below and in reconfiguration manager components of section 12 `AocsUnitHousekeeping` : implemented in AOCS unit objects listed below `AocsUnit` : implemented in AOCS unit objects listed below `TriggerList` : implemented in trigger lists components listed below
Implemented Framelet Components
`PollingTrigger` : trigger object to perform full data transfer (transaction + refresh cycle) with polling on registered units `UnitTrigger` : trigger object to perform full data transfer (transaction + refresh cycle) without polling on registered units `RefreshTrigger` : trigger object to perform refresh operations on registered units `TrasactionTrigger` : trigger object to perform transaction operations on registered units `FullTriggerList` : full implementation of interface `TriggerList` `FunctionalTriggerList` : partial implementation of interface `TriggerList` covering only functional units `MacsTcController` : implementation of interface `AocsUnitHardware` for a MACS telecom controller FssPrototype : two-axis fine sun sensor AOCS unit for the AOCS prototype `GyrPrototype` : single-axis gyro AOCS unit for the AOCS prototype `RwPrototype` : reaction wheel AOCS unit for the AOCS prototype `SapPrototype` : solar acquisition and propulsion electronics AOCS unit for the AOCS prototype `TorquingThrusters` : fictitious AOCS unit to command a set of thrusters directly with the torque

requests around spacecraft axes

12 RECONFIGURATION MANAGEMENT FRAMELET

The reconfiguration management framelet defines the following architectural constructs:

RECONFIGURATION MANAGEMENT FRAMELET
Design Patterns
Reconfiguration Design Pattern : pattern to make handling of reconfigurable objects independent of their reconfigurability
Framelet Interfaces
`Reconfigurable` : interface to be implemented by all reconfiguration managers.
Framelet Core Components
`ConfigurationState` : encapsulation of the state of a reconfiguration group
Framelet Default Components
`ReconfigurerHelper` : helper object to handle the management of a reconfiguration group `BasicUnitReconfigurer` : reconfiguration manager for a group of identical objects `RwSet` : reconfiguration manager for a set of 4 identical reaction wheels

All the constructs listed below are implemented in the AOCS prototype framework.

The prototype AOCS assumes all sensors and actuators to be redundant. In order to facilitate implementation of the prototype AOCS, the prototype framework provides a basic reconfiguration manager to manage reconfigurations across a total of N identical units and a reconfiguration manager to manage reconfigurations across sets of 4 reaction wheels.

Cold redundancy is assumed by all reconfiguration managers predefined in the prototype framework.

Configuration state objects are provided for simple reconfigurations and for 1-out-4 redundancy management.

The reconfiguration design pattern is implemented by the reconfiguration managers provided by the framework prototype.

13 OPERATIONAL MODE MANAGEMENT FRAMELET

The operational mode management framelet defines the following architectural constructs:

OPERATIONAL MODE MANAGEMENT FRAMELET
Framelet Design Pattern
Mode Management Pattern : pattern to endow components with mode-dependent behaviour
Framelet Core Components
`AocsMissionModeManager` : AOCS mission mode manager `ModeManager` : core mode manager component `ModeChangeAction` : encapsulation of a mode change action
Framelet Default Components
`CyclingModeManager` : cycling mode manager component `FollowerModeManager` : follower mode manager component `NullModeChangeAction` : default mode change action that does nothing

All the constructs listed above are implemented in the prototype AOCS framework.

Mode dependent behaviour is implemented in the following components in the prototype framework:

- failure detection manager
- failure recovery manager
- telemetry manager
- unit triggers
- attitude controller

Pre-defined mode managers are provided for each of the above components. Except for the telemetry manager that uses a cycling mode manager, all other mode-dependent components use follower mode managers.

Only hard-coded mode change actions are used in the prototype AOCS. Mode change action objects are not used (except for the default `NullModeChangeAction` that is used to configure mode managers).

14 MANOEUVRE MANAGEMENT FRAMELET

The manoeuvre management framelet defines the following interfaces and components:

MANOEUVRE MANAGEMENT FRAMELET
Framelet Interfaces
`Manoeuvre` : abstract class serving as base class for all manoeuvre classes `ManoeuvreMonitor` : interface to be implemented by objects that need to be notified of changes in manoeuvre status
Framelet Core Components
`ManoeuvreManager` : manoeuvre manager component

All the constructs listed above are implemented in the prototype AOCS framework.

TBW

15 FAILURE DETECTION FRAMELET

The failure detection management framelet defines the following interfaces and components:

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.

All the constructs listed above are implemented in the prototype AOCS framework.

Interface `ConsistencyCheckable` is implemented by the AOCS data objects and by event repositories.

16 FAILURE RECOVERY MANAGEMENT FRAMELET

The failure recovery management framelet defines the following interfaces and components:

FAILURE RECOVERY MANAGEMENT FRAMELET
Design Pattern
Failure Recovery Pattern: design pattern to separate the management of failure recovery from the implementation of failure recovery strategies.
Framelet Interfaces and Base Abstract Classes
`RecoveryAction` : abstract base class for objects encapsulating recovery actions `RecoveryStrategy`: abstract base class for objects encapsulating failure handling strategies `FailureRecoveryModeManager` : interface for the operational mode manager for the failure detection manager.
Framelet Core Components
`FailureRecoveryManager` : failure recovery manager component (including mode manager)
Framelet Components
`SystemReset` : recovery action component encapsulating a system reset `SystemReboot` : recovery action component encapsulating a system reboot `ObjectReset` : recovery action component encapsulating a reset on a specific object `Reconfiguration` : recovery action component encapsulating a reconfiguration action `ModeChange` : recovery action component encapsulating a mode change action `NullRecoveryAction` : null recovery action `SystemResetOnTooManyFailures` : failure recovery strategy to command a system reset if too many failures are found in the failure recovery repository `LocalRecoveryActions` : failure recovery strategy to perform the recovery actions associated to each failure found in the failure recovery repository `FollowerFailureRecoveryModeManager` : failure recovery mode manager based on based on follower mechanism

The prototype framework includes a full failure recovery manager and the failure recovery action and failure strategy components that are expected to be needed for the prototype AOCS.

The next table summarizes the implementation status of the framelet constructs in the prototype framework:

FAILURE RECOVERY MANAGEMENT FRAMELET
Design Pattern Implementation
Failure Recovery Pattern: implemented in failure recovery manager component.
Framelet Interfaces and Base Abstract Classes
`RecoveryAction` : implemented in failure recovery action components listed below `RecoveryStrategy`: implemented in failure recovery action components listed below `FailureRecoveryModeManager` : implemented in default failure recovery mode manager listed below
Framelet Core Components
`FailureRecoveryManager` : implemented
Framelet Components
`SystemReset` : implemented `SystemReboot` : not implemented `ObjectReset` : implemented `Reconfiguration` : implemented `ModeChange` : implemented `NullRecoveryAction` : implemented `SystemResetOnTooManyFailures` : implemented `LocalRecoveryActions` : implemented `FollowerFailureRecoveryModeManager` : implemented

17 TELEMETRY MANAGEMENT FRAMELET

The telemetry management framelet defines the following interfaces and components:

TELEMETRY MANAGEMENT FRAMELET
Abstract Interfaces and Abstract Base Classes
`TelemetryStream` : abstract base class for telemetry streams `Telemeterable` : interface for objects that can write their own state to telemetry `TelemetryModeManager` : interface for the operational mode manager for the failure detection manager.
Core Components
`TelemetryManager` : component encapsulating a telemetry manager (including mode management)
Default Components
`DmaTelemetryStream` : implementation of `TelemetryStream` interface representing a DMA-based telemetry stream `CyclingTelemetryModeManager` : default mode manager for the telemetry manager component implementing a cycling mode management mechanism. `MemorySection` : component encapsulating a range of contiguous memory addresses that are to be copied to the telemetry stream. `TestTelemetryStream` : component simulating a telemetry stream (the telemetry data are sent to a data file).
Design Patterns
Telemetry Management Pattern : design pattern to make an object a telemeterable object

All the constructs listed above are implemented in the prototype AOCS framework.

The prototype AOCS is based on a telemetry interface where raw telemetry data are collected in DMA mode by dedicated hardware (see section 4.2). The prototype framework offers pre-defined components that are intended to facilitate implementation of this type of telemetry

management. In particular, it offers a telemetry stream component that writes raw telemetry data to a fixed memory buffer.

The `Telemeterable` interface foresees several telemetry formats. The prototype framework implements only the *normal* and *short* format.

The `Telemeterable` interface is inherited by most objects in the AOCS software through `AocsObject`. In the prototype framework, however, not-trivial implementations are only provided for the following objects:

- AOCS data (instances of class `AocsData` and its subclasses)
- AOCS events (instance of class `AocsEvent` and its subclasses)
- Event repositories (instances of class `EventRepository` its subclasses)

The telemetry manager component is provided in full.

18 TELECOMMAND MANAGEMENT FRAMELET

The telecommand management framelet defines the following interfaces and components:

TELECOMMAND MANAGEMENT FRAMELET
Framelet Design Patterns
Telecommand Transaction : design pattern to handle sequences of telecommands as a single entity
Framelet Interfaces
`TelecommandLoader` : interface for the telecommand loader
Framelet Core Components
`Telecommand` : base class for telecommands `TelecommandTransaction` : base class for transaction telecommands `TelecommandManager` : telecommand manager component
Framelet Default Components
`ModeChangeTelecommand` : simple telecommand to change the mode of a mode manager `ModeChangeTransactionTelecommand` : transaction telecommand to change the mode of a mode manager `TelemetryFormatTelecommand` : simple telecommand to change the format of a telemeterable object `ManoeuvreTelecommand` : simple telecommand to load a parameterless manoeuvre in the manoeuvre manager `AttitudeSlewTelecommand` : simple telecommand to configure and load an attitude slew manoeuvre `TelemetryFormatTransactionTelecommand` : transaction telecommand to change the format of a telemeterable object `ReconfigureTelecommand` : simple telecommand to command a reconfiguration to a reconfiguration manager `ReconfigureTransactionTelecommand` : transaction telecommand to command a

reconfiguration to a reconfiguration manager

`VsDmaTelecommandLoader` : DMA-based telecommand loader for the Visual Studio environment.

`Erc32DmaTelecommandLoader` : DMA-based telecommand loader for the ERC32 environment with the GNU compiler

All the constructs listed above are implemented in the prototype AOCS framework.

The telecommand manager is implemented in full.

The implementation of the telecommand loader assumes a DMA-based telecommand interface. See section 4.3 for more details.

The loader assumes that the code for the telecommands is already present in the AOCS software memory space and that only telecommand data are loaded.

19 CONTROLLER MANAGEMENT FRAMELET

The controller management framelet defines the following interfaces and components:

CONTROLLER MANAGEMENT FRAMELET
Framelet Interfaces
`Controllable` : interface for controllers `MimoControllable` : interface for MIMO controllers
Framelet Components
`ControllerComponent` : component encapsulating a controller (including mode management) `ControllerManager` : component encapsulating a controller manager

The prototype framework only supports SISO controllers. This restriction is justified by the absence of MIMO controllers in most current AOCS systems and in particular in the prototype AOCS.

The controller components and the controller manager components are implemented in full.

TBW