THE CORDET FRAMEWORKS
Domain Analysis

Prepared by P&P Software GmbH
for the Study on Component Oriented Development Techniques
(ESA-Estec Contract 20463/06/NL/JD)

Written By: A. Pasetti
O. Rohlik
Date: 12 September 2008
Issue: 1.3
Reference: PP-FW-COR-0001

Copyright 2007 P&P Software GmbH All Rights Reserved

Copyright 2007 P&P Software GmbH All Rights Reserved

Table of Contents

1 GLOSSARY AND ACRONYMS....5
2 REFERENCES....7
3 INTRODUCTION....9
3.1 Objectives Of The CORDET Study....9
3.2 Objective Of This Document....9
3.3 Structure Of This Document....10
3.4 Status Of This Document....10
4 DH FRAMEWORK – OVERVIEW....11
4.1 Heritage....11
4.2 Domain Demarcation....11
4.3 The Framework Telecommand Concept....12
4.4 The Telecommand Manager Concept....13
4.5 The Telecommand Loader Concept....13
4.6 The Telecommand Stream Concept....14
4.7 Framework Boundaries – Telecommanding Function....14
4.8 The Framework Telemetry Packet Concept....15
4.9 The Telemetry Manager Concept....16
4.10 The Telemetry Stream Concept....17
4.11 Framework Boundaries – Telemetry Function....17
4.12 Suitability For Non-PUS Applications....18
5 DH FRAMEWORK – DOMAIN DICTIONARY....19
5.1 Domain Dictionary Entries for Telecommand Concept....19
5.2 Domain Dictionary Entries for Telemetry Concept....21
6 DH FRAMEWORK – SHARED PROPERTIES....23
6.1 Shared Properties for Telecommand Concept....23
6.1.1 Telecommand Execution....23
6.1.2 Telecommand Management....24
6.1.3 Telecommand Loading....24
6.2 Shared Properties for Telemetry Concept....25
6.2.1 Telemetry Packet Configuration....25
6.2.2 Telemetry Packets Execution....25
6.2.3 Telemetry Packet Management....26
7 DH FRAMEWORK – FACTORS OF VARIATION....27
7.1 Attributes as Factors of Variation....27
7.2 Factors of Variation for Telecommand Concept....27
7.3 Factors of Variation for Telecommand Loading Concept....29
7.4 Factors of Variation for Telecommand Stream Concept....29
7.5 Factors of Variation for Telemetry Concept....29
7.6 Factors of Variation for Telemetry Stream Concept....31
8 DH FRAMEWORK – FEATURE MODEL....32
8.1 Feature Meta-Model....32
8.2 Top-Level Features....33
8.3 Telecommand Functionality Features....34
8.4 Telecommand Features....34
8.5 Telemetry Functionality Features....35
8.6 Telemetry Packet Features....35
9 CONTROL FRAMEWORK – OVERVIEW....37
9.1 Heritage....37
9.2 Domain Demarcation....37

Copyright 2007 P&P Software GmbH All Rights Reserved

9.3 The Activity Concept....37
9.4 The Data Pool Concept....38
9.4.1 Data Pool Concept vs Localized Concept....40
9.5 The Parameter Database Concept....41
9.6 The Operational Mode Concept....41
9.7 The Activity Manager Concept....42
9.8 Framework Boundaries....43
9.9 Reference Execution Model....44
9.10 Activity Types....45
9.11 Applicability to Other On-Board Subsystems....46
9.12 Relationship to DH Framework....47
10 CONTROL FRAMEWORK – DOMAIN DICTIONARY....48
10.1 Domain Dictionary Entries for Activity Concept....48
10.2 Domain Dictionary Entries for Mode Management Concept....49
10.3 Domain Dictionary Entries for Parameter Database Concept....50
10.4 Domain Dictionary Entries for Data Pool Concept....50
11 CONTROL FRAMEWORK – SHARED PROPERTIES....52
11.1 Shared Properties for Activity Concept....52
11.1.1 Activity Initialization Properties....52
11.1.2 Activity Execution....52
11.1.3 Holding and Resuming Activities....53
11.1.4 Enabling and Disabling of Activities....53
11.2 Shared Properties for Mode Management Concept....53
11.2.1 Activity Manager Activation....53
11.2.2 Current Operational Mode Changes....54
11.3 Shared Properties for Parameter Database Concept....55
11.4 Shared Properties for Data Pool Concept....55
12 CONTROL FRAMEWORK – FACTORS OF VARIATION....56
12.1 Attributes as Factors of Variation....56
12.2 Factors of Variation for Activity Concept....56
12.3 Factors of Variation for Mode Management Concept....58
12.4 Factors of Variation for Data Pool Concept....59
13 CONTROL FRAMEWORK – FEATURE MODEL....60
13.1 Top-Level Features....60
13.2 Data Pool Features....60
13.3 Parameter Database Features....61
13.4 Activity Manager Features....61
13.5 Activity Features....62

Copyright 2007 P&P Software GmbH All Rights Reserved

1 GLOSSARY AND ACRONYMS

The table defines the most important technical terms and abbreviations used in this document.

Term	Short Definition
Abstract Interface	A definition of the signature and semantics of a set of logically related operations without any implementation details.
AOCS	The Attitude and Orbit Control Subsystem of satellites.
AOCS Framework	The old name for the Control Framework described in this document. .
Application	A software program that can be deployed and run as a single executable.
Application Instantiation	The process whereby a component-based application is constructed by configuring and linking individual components.
Component	A unit of binary reuse that exposes one or more interfaces and that is seen by its clients only in terms of these interfaces.
Component-Based Framework	A software framework that has components as its building blocks.
Computational Node	A computational resource that has memory and processing capabilities.
Control Framework	A framework covering the AOCS Subsystem (or, more in general, the control part of an on-board application).
CORBA	A widely used middleware infrastructure.
Design Pattern	A description of an abstract design solution for a common .
DH	Data Handling (one of the functional subsystems of an on-board system).
DH Framework	A framework covering the DH subsystem (one of the two frameworks presented in this document).
Domain	A short-hand for either 'family domain' or 'framework domain'.
DSL	Domain Specific Language (a language that is created to describe applications or components in a very narrow domain).
DTD	Document Type Definition. It defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements. Its purpose is similar to the one of an XML Schema, although it is not as feature rich and the syntax is different.
EMF	Eclipse Modelling Framework: a modeling framework and code generation facility for building tools and other applications based on a structured data model.
Family Domain	The set of systems whose implementation is supported by a system or product family.
FDIR	Failure Detection Isolation and Recovery.
Feature	A characteristics of a system or an application that is relevant to its users.
Feature Model	A description of a set of features and their legal combinations.
Framework Domain	The set of applications whose implementation is supported by the framework.
Framework Instantiation	The process whereby a framework is adapted to the needs of a specific application within its domain.
Functional Property	A property that can be expressed as a logical relationship among the variables that define the state of an application or system.
Generative Programming	A software engineering paradigm that promotes the automatic generation of an implementation from a set of specifications.
Generic Architecture	A set of reusable and adaptable software assets to support the instantiation of systems within a certain target domain. In the CORDET project, a generic architecture consists of a system family, to model the non-functional aspects of systems in the architecture's target domain, and a set of software frameworks, to model their functional aspects. The objective of the CORDET Project is to define a generic architecture for satellite on-board systems.
GNC	Guidance Navigation and Control (a synonym for *AOCS*).
Interface	An abstract specification of services to be provided by any concrete realisation of it.
JVM	Java Virtual Machine.
Non-Functional Property	A property other than a functional property.

Copyright 2007 P&P Software GmbH All Rights Reserved

Object Oriented Framework	A software framework that uses inheritance and object composition as its chief adaptation mechanisms.
OBS	The On-Board Software.
OBS Framework	A prototype framework for on-board systems developed by P&P Software (see [RD18]).
OtM Adaptability	Outside-the-Model Adaptability. An adaptability mechanism that is defined outside the UML2 model.
Product Family	A set of applications or systems that can be built from a pool of shared assets.
Property	Same as a 'feature' above.
Software Component	A unit of binary reuse that exposes one or more interfaces and that is seen by its clients only in terms of these interfaces.
Software Framework	A kind of product family where the shared assets are software components embedded within an architecture optimized for a certain domain and the 'product' is a software application.
System	A group of independent but interrelated hardware and software elements comprising a unified whole.
System Family	A kind of product family where the 'product' to be built using the reusable assets provided by the family is the architectural infrastructure (the 'middleware') of a complex system.
XML	Extensible Markup Language. XML documents consist (mainly) of text and tags, and the tags imply a tree structure upon the document. An XML document is said to be valid if it conforms to an XML Schema or a DTD.
XML Schema	The XML Schema language is also referred to as XML Schema Definition (XSD). They provide a means for defining the structure, contents and semantics of XML documents. XML Schemas are written in XML
WtM Adaptability	Within-the-Model Adaptability Mechanism. An adaptability mechanism that is defined within the UML2 model.

Copyright 2007 P&P Software GmbH All Rights Reserved

2 REFERENCES

RD1	Control Framework Project Web Site, control.ee.ethz.ch/~ceg/AocsFrameworkProject
RD2	RT Java Control Framework Project Web Site, control.ee.ethz.ch/~ceg/RealTimeJavaFramework
RD3	Brauer (1999) Object Oriented Languages Study. Final Report for ESA contract 12889/NL/PA
RD4	Clemens P, Northrop L (2001) *A Framework for Software Product Line Practice,* Software Engineering Institute, Carnegie Mellon University, Available from Internet Web Site: www.sei.cmu.edu/activities/plp/framework.html
RD5	Donohoe P (ed), *Software Product Lines – Experience and Research Directions,* Kluwer Academic Publisher
RD6	Fayad M, Schmidt D, R. Johnson R (eds), *Building Application Frameworks – Object Oriented Foundations of Framework Design,* Wiley Computer Publishing, 1999
RD7	Gamma E, et al, *Design Patterns – Elements of Reusable Object Oriented Software,* Addison-Wesley, Reading, Massachusetts
RD8	TimeSys Home Page, http://www.timesys.com/index.cfm
RD9	AERO Project Home Page, http://www.aero-project.org
RD10	Aicas Home Page, http://www.aicas.com
RD11	OBOSS Home Page, http://spd-web.terma.com/Projects/OBOSS/Home_Page/
RD12	Pasetti A, et al, *An Object-Oriented Component-Based Framework for On-Board Software*, Proceedings of the Data Systems In Aerospace Conference, Nice, May 2001
RD13	Pasetti A., *Software Frameworks and Embedded Control Systems*, LNCS Series, Springer-Verlag, 2001
RD14	Szyperski C, *Component Software*. Addison Wesley Longman Limited, Harrow (UK), 1998
RD15	Czarnecki, K., Eisenecker, U.: *Generative Programming – Methods, Tools, and Applications,* Addison-Wesley, 2000
RD16	Birrer I, Chevalley P, Pasetti A, Rohlik O, *An Aspect Weaver for Qualifiable Applications*, Proceedings of the Data Systems in Aerospace (DASIA) Conference, Nice 2004.
RD17	XWeaver Web Site: http://www.pnp-software.com/XWeaver
RD18	OBS Framework Web Site, http://www.pnp-software.com/ObsFramework
RD19	Introduction to Aspect Oriented Programming, http://www.pnp-software.com/AspectOrientedProgramming.html
RD20	Birrer I, Pasetti A, Rohlik O, *Implementing Adaptability in Embedded Software through Aspect Programs,* IEEE Proceedings of the Mechantronic & Robotics Conference 2004, Aachen, Germany, Sept. 2004
RD21	Automated Framework Instantiation Project Web Site, http://www.pnp-software.com/AutomatedFrameworkInstantiation
RD22	Cechticky V, Pasetti A, Rohlik O, Schaufelberger W, *XML-Based Feature Modelling,* published in: J. Bosch, C. Kueger (eds), *Software Reuse: Methods, Techniques, and Tools,* LNCS Vol 3107, Springer-Verlag, 2004

Copyright 2007 P&P Software GmbH All Rights Reserved

RD23	Cechticky V, Chevalley P, Pasetti A, Schaufelberger W, *A Generative Approach to Framework Instantiation,* published in: F. Pfenning, Y. Smaragdakis (eds), *Generative Programming and Component Engineering,* LNCS Vol 2830, Springer-Verlag, 2003
RD24	ASSERT Project – HRI pilot project, Deliverable D6.3.1-1 : *HRI System Family Definition Report*
RD25	ASSERT Project – HRI pilot project, Deliverable D6.3.2-1 : *HRI Reference Architecture Definition Report V1*
RD26	C. Moreno, G. Garcia, *Plug & Play architecture for on-board software components*, Proceedings of the DASIA 2002 conference, Nice 2002
RD27	ASSERT Project – ETH Deliverable D4.2.2-1 : *Software Building Blocks Adaptation Techniques – The FW Profile*
RD28	ASSERT Project – ETH Deliverable D4.2.4-3 : *Contribution to V2 Demonstrator*
RD29	ASSERT Project – ETH Deliverable D4.2.4-4.1 : *The ETH Demonstrator Framework – Contribution to V3 Demonstrator, Part 1 (Domain Design)*
RD30	ASSERT Project – ETH Deliverable D4.2.4-4.2 : *The ETH Demonstrator Framework – Contribution to V3 Demonstrator, Part 2 (Domain Implementation)*
RD31	ASSERT Project – UPD Deliverable TBD : Deliverable Describing the RCM Methodology
RD32	Egli M, Pasetti A, Rohlik O, Vardanega T, *A UML2 Profile for Reusable and Verifiable Real-Time Components*, in: Morisio M (ed), Reuse of Off-The-Shelf Components, LNCS Vol 4039, Springer-Verlag, 2006
RD33	ASSERT Project – ETH Deliverable D4.2.3-1 : *Product Family Meta-Model Definition – A Family Meta-Model for the XFeature Tool*
RD34	GMV, *Standard, Methods, and Tools Review for Domain Analysis and Domain Design Execution,* CORDET Deliverable GMV-CORDET-WP202-RP-01
RD35	P&P Software GmbH, *The CORDET Methodology*, CORDET Deliverable Document PP-MR-COR-001, http://www.pnp-software.com/cordet
RD36	ECSS-E70-41A, *Ground System and Operation – Telecommand and Telemetry Packet Utilization*, 30th January 2003

Copyright 2007 P&P Software GmbH All Rights Reserved

3 INTRODUCTION

This document is written as part of the ESA study on *Component Oriented Development Techniques* or CORDET. The study is done under ESA contract 20463/06/NL/JD.

3.1 Objectives Of The CORDET Study

The general objective of the CORDET study is the definition of a *generic architecture* for on-board satellite applications.

The term “generic architecture” is used to designate a set of reusable and adaptable software assets to support the instantiation of systems within a certain target domain. In the CORDET project, a generic architecture consists of a *system family*, to model the non-functional aspects of systems in the architecture's target domain, and a set of *software frameworks*, to model their functional aspects.

The terms “system family” and “software frameworks” are used to designate two kinds of *product families*. A product family is a set of applications or systems that can be built from a pool of shared assets. A system family is a kind of product family where the 'product' to be built using the reusable assets provided by the family is the architectural infrastructure (the 'middleware') of a complex system. A software framework is a kind of product family where the 'product' to be built is a software application and the shared assets are software components embedded within an architecture optimized for a certain domain.

The generic architecture to be defined in this study is called the *CORDET Generic Architecture*. The product families which constitute the CORDET Generic Architecture are called the *CORDET Product Families*.

Against this background, the more specific objectives of the CORDET study are:

- To define a methodology for the development of the CORDET Generic Architecture and, by implication, for product family-based development activities at both system- and software-level for satellite on-board applications.
- To identify and to define at the level of their functional and non-functional interfaces the product families that constitute the CORDET Generic Architecture.
- To demonstrate the proposed methodology and the proposed architecture by instantiating a subset of its product families to build an end-to-end demonstrator of an on-board system.
- To get feedback from the space community in order to reach as large an agreement as possible on the outputs of the CORDET study.

The CORDET Methodology, covering the first of the four objectives listed above, is defined in RD-35. Some familiarity with reference document RD-35 is a pre-requisite for an appreciation of the present document.

3.2 Objective Of This Document

The CORDET Methodology foresees that the CORDET Generic Architecture be split between a functional and non-functional part and that each part be developed in three phases (domain analysis, domain design, and domain implementation).

The present document covers the *domain analysis phase* for the *functional part* of the CORDET Generic Architecture.

The functional part of the CORDET Generic Architecture consists of a set of *software frameworks*, one for each functional subsystem of an on-board system. This document covers

Copyright 2007 P&P Software GmbH All Rights Reserved

the software frameworks for the Data Handling (DH) subsystem and for the Attitude and Orbit Control Subsystems (AOCS).

The term "Control Framework" is used in this document to designate the framework that covers the AOCS subsystem. Note that in earlier project documentation (and in particular in RD-35) the term "AOCS Framework" was used for the framework covering the AOCS Subsystem. The analyses performed during the preparation of the present document have, however, shown that the scope of this framework is wider than anticipated and that it can be used for generic control systems rather than just for the Attitude and Orbit Control System (AOCS). Hence, the new name "Control Framework" has been adopted for it.

The term "DH Framework" is used in this document to designate the framework that covers the DH subsystem.

In general, the functionalities implemented in an on-board application can be divided into two broad categories. The first category comprises functionalities that are essentially sporadic and event-driven (where the "driving event" can be a command originating outside the application, a request for some information to be sent outside the application, a hardware interrupt, etc). The second category of functionalities are essentially periodic and consist of "activities" that, at every cycle, process the same set of inputs to produce the same set of outputs according to the same algorithm.

The DH Framework is aimed at functionalities of the first kind. The Control Framework is aimed at functionalities of the second kind. Thus, taken together, the two frameworks cover most of the functionalities present in an on-board application.

The two frameworks are intended to be independent of each other. Note, however, that methodological requirement MR7.3-2 in RD35 requires the two frameworks to be inter-operable in the sense that the reusable assets that they provide should be designed to be deployable within the same application. The interoperability is implemented at design level and has no impact on the domain models presented in this document.

3.3 Structure Of This Document

The output of the domain analysis phase is a *domain model*. This document therefore presents the domain models for the DH and Control Frameworks of the CORDET Generic Architecture.

The domain model for the DH Framework is presented in the first part of this document coverings sections 4 to 8. The domain model for the Control Framework is presented in the second part of this document covering sections 9 to 13.

The definition of each of the two frameworks starts with an introductory section that gives an informal overview of a framework (section 4 for the DH Framework and section 9 for the Control Framework).

In accordance with the CORDET Methodology, the domain model consists of four items: the domain dictionary, the shared properties, the factors of variations, and the feature model. Each of these items is presented in a dedicated section (sections 5 to 8 for the DH Framework and sections 10 to 13 for the Control Framework).

3.4 Status Of This Document

The definition of the DH Framework and of the Control Framework is complete.

Copyright 2007 P&P Software GmbH All Rights Reserved

4 DH FRAMEWORK – OVERVIEW

This section presents an overview of the DH Framework at domain analysis level. This section is intended to place the reader in a better position to appreciate the formal definition of the domain model of the DH Framework as it is presented in the next two sections.

4.1 Heritage

The primary heritage of the DH Framework is the so-called ETH Demonstrator Framework[1] developed at ETH by the authors of this technical note in the ASSERT Project (see [RD-29] and [RD-30]).

A secondary heritage is in the part of the OBS Framework [RD-18] dealing with telecommand and telemetry management.

4.2 Domain Demarcation

The core of a DH subsystem is the handling of incoming telecommands and the generation of outgoing telemetry data. A framework approach for the DH subsystem is only possible if the format and content of the telecommands and telemetry data is, at least to some extent, standardized.

The DH Framework assumes that telecommanding and telemetry is implemented in accordance with the Packet Utilization Standard or PUS [RD-36]. Applicability to non-PUS systems is not excluded (see section 4.12) but the PUS constitutes the conceptual framework within which the DH Framework is defined.

The PUS defines the external interface of a DH application in terms of the *services* that the application must provide to other applications. The services are in turn defined in terms of *telecommand packets* that the application must be able to handle, and *telemetry packets* that the application must be able to generate.

The PUS implicitly defines the concept of an abstract telecommand packet and an abstract telemetry packet. This concept is independent of the particular service which the telecommand packet or telemetry packet supports. The definition of the abstract telecommand packets and abstract telemetry packets covers the features that are common to all PUS-compliant telecommand packets and PUS-compliant telemetry packets.

The DH Framework provides software interfaces and software components that support the implementation and manipulation at software level of abstract telecommand packets and abstract telemetry packets.

The DH Framework, in other words, transposes the PUS to the software level. The PUS standardizes the services to be provided by a DH application. The DH Framework standardizes the software interfaces through which those services are accessed at software level within a DH application.

The PUS also defines a taxonomy of specific services that may be provided by a DH application. Each kind of service is identified by a *type*. The provision of that service by the on-board application is supported by a number of telecommand and telemetry packets. Each kind of telecommand packet or telemetry packet within the service type is identified by a *subtype*.

The PUS pre-defines some service types. These pre-defined service types represent services that are commonly used in on-board systems. For these pre-defined services, the PUS defines both the physical layout and the functional interpretation of the associated telecommand and telemetry packet subtypes.

[1]

Copyright 2007 P&P Software GmbH All Rights Reserved

Application designers can use the services pre-defined by the PUS or they can defined new application-specific services.

The DH Framework does not support the implementation of the pre-defined PUS service types. Provision of this kind of support would be technically feasible and industrially desirable but it is outside the scope of the CORDET Project.

Future extensions of the DH Framework might extend the interfaces and components provided by the framework to support some or all of the PUS service types.

The DH Framework also does not support the routing of telecommand and telemetry packets between applications. The framework is aimed at the *processing* of telecommands *within* a PUS application and at the *generation* of telemetry packets from *within* a PUS application. It does not explicitly cover the transmission of packets *between* PUS applications.

The concepts provided by the framework, however, could be used to implement such a dispatching framework to link together PUS applications. In particular, the support offered by the framework for the manipulation of abstract telecommands and telemetry packets would facilitate the implementation of an application-independent infrastructure for routing telecommand and telemetry packets.

4.3 The Framework Telecommand Concept

The framework telecommand concept is one of the two key concepts of the DH Framework.

A *framework telecommand* is the abstract representation at software level of a PUS-compliant telecommand[2].

A PUS-compliant telecommand packet defines a set of *actions* that must be executed by an application over a certain time interval. The execution of the actions may be conditional upon certain *checks* being successfully passed. The checks, like the actions, are implicitly specified in the telecommand packet.

A framework telecommand is a representation of a telecommand packet in the sense that it encapsulates the checks and the actions defined by the telecommand packet. This representation is *abstract* in the sense that it is independent of the physical layout of the telecommand packet.

A telecommand packet is received by the DH application as a sequence of bytes. The framework telecommand that represents it is created by decoding this sequence of bytes.

After being created, the framework telecommand passes through four states: ACCEPTED, STARTED, IN_PROGRESS, and COMPLETED. These four states are implicitly defined by the PUS. They correspond to the stages of a telecommand execution where verification reports (PUS telemetry packets of type 1) are sent to the sender of the telecommand (see figure 4.3-1).

The telecommand states are entered in sequence as the telecommand is *executed*. The IN_PROGRESS state can be entered more than once to represent the fact that some telecommands execute actions that extend over time.

To each telecommand state one or more checks and an action may be associated. The check determines whether the state can be entered (acceptance, start, in progress, and completion checks) or exited (ready check). For instance, if the acceptance check fails, then the telecommand cannot be accepted. The action encapsulates the actions that are to be performed

[2]Note that, in this document, the term "telecommand" is overloaded. It can either designate a raw telecommand (i.e. a telecommand packet understood as a sequence of bytes) or it can designate the software entity that represents the raw telecommand within the framework. It is hoped that the context within which the term is used is sufficient to avoid ambiguities.

Copyright 2007 P&P Software GmbH All Rights Reserved

when the telecommand enters the state. For instance, the start action defines the actions to be executed when the telecommand is started.

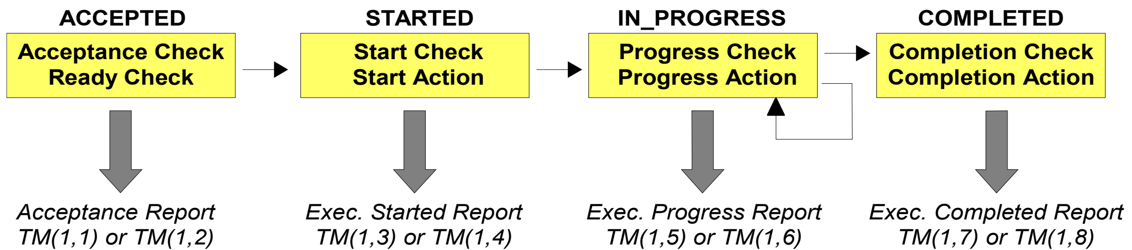

Fig. 4.3-1: Telecommand Lifecycle

Figure 4.3-1 shows the nominal life cycle of a telecommand. The DH Framework also allows telecommands to be aborted. Telecommands can be aborted either because of an external request or because they have failed one of their state checks.

The DH Framework pre-defines the logic to handle the transitions across the telecommand states. It defines, in other words, the logic to manage the execution of the telecommand checks and telecommand actions but it leaves the definition of their content open.

The contents of the telecommand checks and of the telecommand actions thus represent factors of variation for the framework. Application designers instantiate the DH Framework by specifying the content of the checks and actions to be associated to each telecommand.

4.4 The Telecommand Manager Concept

Framework telecommands are *passive* in the sense that they do not possess an own thread of execution. Some other component is responsible for controlling their execution. The DH Framework provides the *telecommand manager* as a pre-defined component to control the execution of one or more telecommands.

Telecommand managers can be *activated*. A telecommand manager holds a list of framework telecommands. When it is activated, it advances the execution of the telecommands in the list. Telecommands that have terminated their execution are unloaded from the telecommand manager.

Activation of the telecommand managers is performed by components outside the DH Framework. In a typical case, the telecommand manager would be activated by an external scheduler.

There is no restriction on the number of telecommand manager instances that may exist within the same application. The application designer is free to instantiate several telecommand managers and to allocate each of them to certain types of telecommands. For instance, one telecommand manager might be used for "fast" telecommands that have short deadlines and must execute at high priority, whereas another telecommand manager might be reserved for "slow" telecommands that have long deadlines and can run at low priority.

4.5 The Telecommand Loader Concept

Framework telecommands are abstract representations of the actions and checks defined in a telecommand packet in the sense that they are independent of the physical layout of the telecommand packet itself. The DH Framework defines a *telecommand loader* as a component that is capable of: (a) de-coding the content of the telecommand packet, (b) creating the framework telecommand that represents it, and (c) loading the framework telecommand into the telecommand manager.

Copyright 2007 P&P Software GmbH All Rights Reserved

As indicated in the previous section, there may be several telecommand managers in the same application. The telecommand loader encapsulates the logic for selecting the telecommand manager into which a newly-created framework telecommand should be loaded. Since this selection logic is obviously application-specific, its implementation constitutes one of the factors of variation of the framework.

Telecommand loaders can be *activated*. When a telecommand loader is activated, it reads the raw telecommand data and initiates the de-serialization process through which the corresponding framework telecommand is created.

The logic that controls the activation lies outside the DH Framework. The DH Framework thus is compatible with architectures where the source of raw telecommand data is sampled periodically as well as with architectures where the de-serialization process is triggered by an interrupt.

The DH Framework assumes that there is one single telecommand loader in an application.

4.6 The Telecommand Stream Concept

Telecommand packets are received by an application as streams of bytes. The physical interface through which the bytes are received is not standardized and varies across applications. In order to handle this variability, the DH Framework defines the concept of *telecommand stream* as an abstract representation of the physical interface through which the stream of bytes implementing a telecommand packet is received.

The fundamental operation that can be performed on a telecommand stream is a *read operation* whereby the next data item from the stream is acquired.

In practice, several kinds of read operations are provided by a telecommand stream, one for each type of data that is recognized by the PUS standard. Thus, for instance, there are read operations to read the packet source sequence number, its type and subtype, its acknowledge flags, etc. The presence of such dedicated read operations allows the telecommand loader to be independent of the physical layout of the incoming packets.

The DH Framework assumes that there is one single telecommand stream in an application. It assumes in other words that there is one single source of raw telecommands.

The possibility of having multiple telecommand streams (and hence multiple telecommand loaders) was considered but was eventually rejected due to the extra complexity that would have been introduced in the framework design. If there are multiple sources of raw telecommand data, then these must be hidden behind the telecommand stream interface.

4.7 Framework Boundaries – Telecommanding Function

Figure 4.7-1 sketches the conceptual boundaries between the part of an application covered by the telecommanding function of the DH Framework and the remainder of the application.

The actions and checks to be executed by the telecommands are encapsulated in framework telecommands that run under the control of a telecommand manager. The framework telecommands are created by the telecommand loader that decodes the information in the raw telecommands packets. The interface through which the raw telecommand data are received is conceptualized as a steam-like data source.

Interaction with the telecommanding function of the framework takes place through the following channels:

- through activation signals sent to the telecommand manager ;
- through activation signals sent to the telecommand loader;
- through actions performed by the telecommands on other parts of the application;

Copyright 2007 P&P Software GmbH All Rights Reserved

- through the acquisition of raw telecommand data from the telecommand stream.

A suggestive way to see the telecommanding part of the DH Framework is as an interpretation engine that is fed raw telecommand data and "interprets" them by executing the actions that they encapsulate. The telecommand interpretation mechanism is independent of the content of the telecommands and this is the reason why it can be implemented in a set of reusable and application-independent software assets.

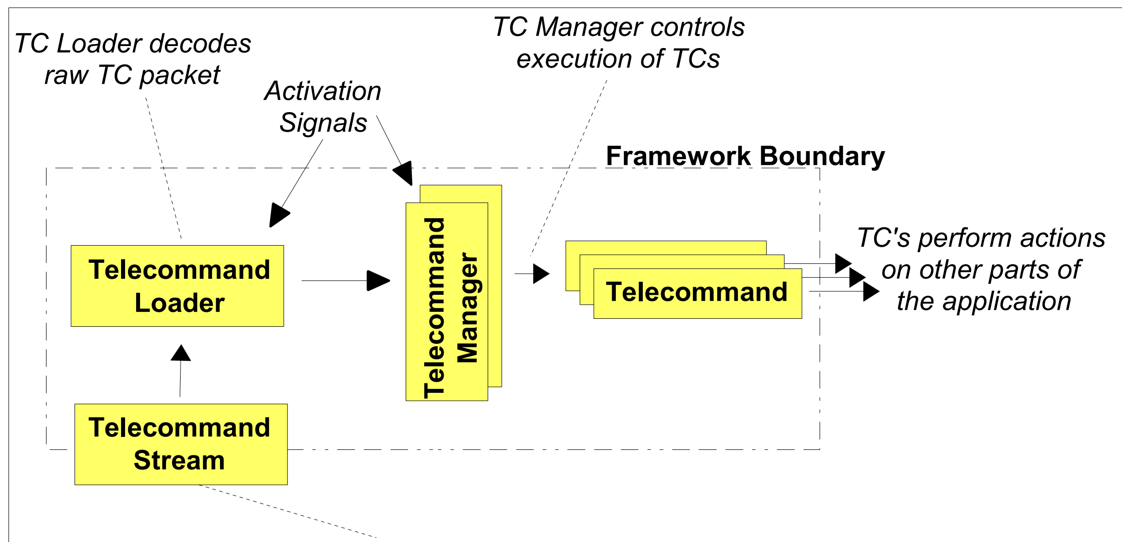

Fig. 4.7-1: Telecommanding Function Boundaries

4.8 The Framework Telemetry Packet Concept

The framework telemetry packet is the second key concept of the DH Framework.

A *framework telemetry packet* is the abstract representation at software level of a PUS-compliant telemetry packet[3].

A PUS-compliant telemetry packet defines a set of data that must be sent to another PUS application in a certain format together with the conditions under which the data must be sent.

A framework telemetry packet is a representation of a telemetry packet in the sense that it encapsulates the *data collection process* for the data in the telemetry packet together with the *conditional checks* that determine whether the packet should be sent or not. This representation is *abstract* in the sense that it is independent of the physical layout of the telemetry packet.

A telemetry packet eventually leaves the DH application as a sequence of bytes. This sequence of bytes is created by *serializing* the framework telemetry packet.

The data collection process for telemetry packets is implemented in two stages. Firstly, telemetry packets must be configured (*configuration action*). In the configuration process, the target data that a telemetry packet must collect are identified. Secondly, telemetry packets must be updated (*update action*). In the update process, a telemetry packet acquires the latest value of its target data. Configuration can only be done once when the telemetry packet is created, whereas update can be done either once (for sporadic packets) or repeatedly (for cyclical packets).

[3]Note that, in this document, the term "telemetry" is overloaded. It can either designate a raw telemetry packet (i.e. a telemetry packet understood as a sequence of bytes) or it can designate the software entity that represents the raw telemetry packet within the framework. It is hoped that the context within which the term is used is sufficient to avoid ambiguities.

Copyright 2007 P&P Software GmbH All Rights Reserved

The conditional checks determine whether the packet is disabled or enabled (*enable check*), whether it is held (*hold check*), and whether it is terminated (*termination check*).

Framework telemetry packets have a limited lifetime. They are created when some client component in the host application makes a request to generate a packet of a certain type. They remain in existence until their termination check determines that they are terminated. When telemetry packets are terminated, they execute their *termination action*.

Telemetry packets may be disabled and enabled. If a telemetry packet is disabled, then its content is not sent to its destination (i.e the telemetry packet is not serialized).

Telemetry packets may be held and resumed. If a packet is held, then its content is not sent to its destination (i.e the telemetry packet is not serialized). Thus, a packet that is held behaves like a packet that is disabled. However, packets are enabled and disabled by some external entity (typically a telecommand) and the change from enabled to disabled and vice-versa is semi-permanent and remains in force until an explicit request to change the state of the telemetry packet is received. The decision to hold a packet is instead taken by the packet itself (based on the outcome of its hold check) and depends on an assessment of the internal state of the packet itself and of its environment.

The enable/disable and hold/resume mechanisms could in principle be merged at domain analysis level since both result in the collection of telemetry data and their dispatching being suspended. The key difference between them that has led to their being kept separate is that the enable/disable mechanism is entirely implemented at framework level (there are no factors of variation associated to it – see section 7.5). This is because the enable/disable mechanism essentially consists of a toggle flag that is controlled by external components. The hold mechanism is instead more complex and more flexible and includes one factor of variation (the implementation of the hold check – see again section 7.5).

The DH Framework pre-defines the logic to handle the actions and the checks associated to the telemetry packets but it leaves the definition of their content open.

The contents of the telemetry checks and of the telemetry actions thus represent factors of variation for the framework. Application designers instantiate the DH Framework by specifying the content of the checks and actions to be associated to each telemetry packet.

4.9 The Telemetry Manager Concept

Framework telemetry packets are *passive* in the sense that they do not possess an own thread of execution. Some other component is responsible for controlling their execution (namely for initiating the execution of the actions and checks associated to them). The DH Framework provides the *telemetry manager* as a pre-defined component to control the execution of one or more telemetry packets.

Telemetry managers can be *activated*. A telemetry manager holds a list of framework telemetry packets. When it is activated, the telemetry manager advances the processing of the telemetry packets in the list. After they are configured, telemetry packets cyclically pass through the following stages: (1) they check whether they are disabled or held, (2) if they are neither disabled nor held, they update their content and (3) serialize it to the telemetry stream (see next section), and (4) they check whether they are terminated. The telemetry manager is responsible for “pushing” telemetry packets through this execution cycle. For this purpose, telemetry packets implement an *execute* operation which is called by the telemetry manager when it is activated.

Telemetry packets that have terminated their execution are unloaded from the telemetry manager.

Copyright 2007 P&P Software GmbH All Rights Reserved

Activation of the telemetry managers is performed by components outside the DH Framework. In a typical case, the telemetry managers would be activated by an external scheduler.

There is no restriction on the number of telemetry manager instances that may exist within the same application. The application designer is free to instantiate several telemetry managers and to allocate each of them to certain types of telemetry packets. For instance, one telemetry manager might be used for "fast" telemetry packets that have short deadlines and must be processed at high priority whereas another telemetry manager might be reserved for "slow" telemetry packets that have long deadlines and can be processed at low priority.

4.10 The Telemetry Stream Concept

A telemetry packet eventually leaves the source application as a sequence of bytes. This sequence of bytes is created by *serializing* the framework telemetry packet. The physical interface to which the output of the serialization is written is not standardized and varies across applications. In order to handle this variability, the DH Framework defines the concept of *telemetry stream* as an abstract representation of the physical interface through which the stream of bytes implementing a telecommand packet is sent out.

The fundamental operation that can be performed on a telecommand stream is a *write operation* whereby the next data item from the serialization process is written to the stream.

In practice, several kinds of write operations are provided by a telemetry stream, one for each type of data that is recognized by the PUS standard. Thus, for instance, there are write operations to write the packet source sequence number, its type and subtype, its destination, etc. The presence of such dedicated write operations allows the telemetry packets to be independent of the physical layout of the packets.

The DH Framework allows only one telemetry stream to be associated to the same applications. The option of having multiple telemetry streams in the same application was considered but was eventually rejected. The single-stream option was preferred for reasons of simplicity and to maintain the symmetry with the telecommanding concept (where there is only one telecommand stream for each application). The routing of raw telemetry data to the appropriate data sink must be done by the telemetry stream itself using mechanisms that are hidden from the framework part of the DH Application. The information about the destination data sink is stored in the packet header.

4.11 Framework Boundaries – Telemetry Function

Figure 4.11-1 sketches the conceptual boundaries between the part of an application covered by the telemetry function of the DH Framework and the remainder of the application.

The telemetry data and their acquisition and generation rules are encapsulated in framework telemetry packets that are controlled by one or more telemetry managers. A framework telemetry packet is created when a client component in the host application makes a request to generate a new packet. The packets serialize their content to the telemetry stream that represents the physical interface through which the raw telemetry data are pushed out of the application.

Interaction with the telemetry function of the framework takes place through the following channels:

- through activation signals sent to the telemetry manager;
- through data acquisition operations (configure and update actions) performed by the telemetry packets on other parts of the host application;
- through the sending of raw telemetry data to the telemetry stream;
- through the requests to create new telemetry packets by the client components in other parts of the host application.

Copyright 2007 P&P Software GmbH All Rights Reserved

A suggestive way to see the telemetry part of the DH Framework is as an interpretation engine that is fed telemetry packets and "interprets" them by executing the actions and checks that they encapsulate. The telemetry interpretation mechanism is independent of the content of the telemetry packets and this is the reason why it can be implemented in a set of reusable and application-independent software assets.

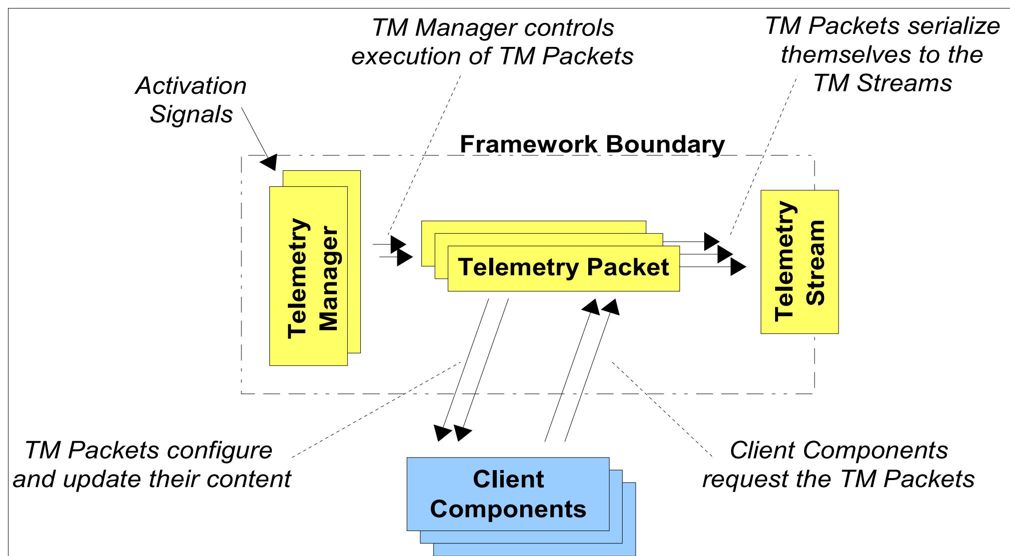

Fig. 4.11-1: Telemetry Function Concept

4.12 Suitability For Non-PUS Applications

The DH Framework is aimed at PUS-compliant applications. However, wider usage is possible subject to some restrictions.

The framework telecommands and telemetry packets are *abstract* representations of PUS-compliant telecommand and telemetry packets. They are therefore independent of the physical layout of the packets. Hence, applications that comply with the overall PUS logic but use different conventions for encoding telecommands and telemetry data can still use the DH Framework. They must, however, provide their own implementation of the telecommand loading and of the telemetry serialization processes since these are the processes where the decoding of raw telecommand packets and the encoding of raw telemetry packets is performed.

The DH Framework defines a certain logic for handling the actions and checks associated to telecommands and telemetry packets. This logic is essentially dictated by the PUS. Applications that use a reduced version of the PUS where some telecommand or telemetry actions and checks are omitted can still use the DH Framework. This is possible because the telecommand and telemetry actions and checks are left open by the DH Framework (they are factors of variation within the framework domain) and hence users can easily "switch off" some of the checks and actions and thus by-pass some of the states associated to the telecommands or the telemetry packets.

Copyright 2007 P&P Software GmbH All Rights Reserved

5 DH FRAMEWORK – DOMAIN DICTIONARY

This section presents the domain dictionary for the DH Framework. The data dictionary entries are listed in logical order (as opposed to alphabetical order). Each data dictionary entry is presented in a table with the following format:

Term	<Domain Dictionary Term>
Definition	<Domain Dictionary Definition>

As specified in the CORDET Methodology, the domain dictionary entries are formulated in natural language. However, readers will readily note that there is a common pattern to the definition of the domain dictionary entries (a sort of informal "meta-model" of the domain dictionary entries). The main items that are used to define a domain dictionary entry are: its *attributes*, its *operations*, its *actions*, and its *checks*.

An attribute designates characteristics that are entirely defined by their value. The operations, actions and checks designate executable functionalities that are associated to the entity being defined. Operations are executed *upon* the entity being defined. Actions and checks are instead executed *by* the entity being defined as a result of changes in its internal state. Checks differ from actions in that they return a value.

The domain dictionary does *not* define the semantics of the attributes, operations, actions and checks associated to its entries. It merely defines their existence. Their semantics is implied by the properties within which the domain dictionary entries appear.

5.1 Domain Dictionary Entries for Telecommand Concept

This section defines the domain dictionary entries related to the framework telecommand concept (see sections 4.3 to 4.7).

Term	**Framework Telecommand**
Definition	The abstract representation at software level of a PUS-compliant telecommand packet. A framework telecommand encapsulates the checks and the actions defined by the telecommand packet. Its representation of a telecommand packet is *abstract* in the sense that it is independent of the physical layout of the telecommand packet. A framework telecommand is characterized by a set of *attributes*, a set of *operations* that can be performed upon the telecommand, and a set of *actions* and of *checks* that the telecommand can perform upon itself or upon its environment. After being created, a framework telecommand passes through four states: ACCEPTED, STARTED, IN_PROGRESS, and COMPLETED. The transition through the states is triggered by the execution of the telecommand operations. The transition is controlled by the outcome of the telecommand checks. The actions to be performed in each state are encapsulated in the telecommand actions. The attributes associated to a framework telecommand are: • The type and subtype of the telecommand • The source sequence counter of the telecommand • The acknowledge flags of the telecommand • The current state of the telecommand

Copyright 2007 P&P Software GmbH All Rights Reserved

	The operations that can be performed upon a framework telecommand are: • The telecommand can be executed • The telecommand can be aborted The actions associated to a framework telecommand are: • The start action • The progress action • The completion action • The abort action The checks associated to a framework telecommand are: • The acceptance check • The ready check • The start check • The progress check • The completion check

The attributes associated to a telecommand are derived from the attributes defined by the PUS for the telecommand packet header and for the telecommand packet data field header. Since a framework telecommand is the representation of a PUS telecommand within the application that receives it, only PUS attributes that are relevant to the receiving application are included.

Term	**Telecommand Manager**
Definition	The encapsulation of a set of pending framework telecommands and of the logic to execute them. A telecommand manager is characterized by a set of *attributes* and by a set of *operations* that can be performed upon it. The attributes associated to a telecommand manager are: • The list of pending framework telecommands The operations that can be performed upon a telecommand manager are: • The telecommand manager can be activated • The telecommand manager can be asked to abort one of its pending telecommands • A framework telecommand can be loaded into the telecommand manager

Term	**Telecommand Loader**
Definition	The entity that is responsible for controlling the process through which raw telecommand data are de-serialized and the corresponding framework telecommands are created and loaded in a telecommand manager. A telecommand loader is characterized by a set of *attributes,* by a set of *operations* that can be performed upon it, and by a set of *actions* that the telecommand can perform upon itself or upon its environment The attributes associated to a telecommand loader are: • The telecommand managers onto which the telecommand loader can load telecommands • The telecommand stream from which the raw telecommand packets are read The operations that can be performed upon a telecommand loader are: • The telecommand loader can be activated

Copyright 2007 P&P Software GmbH All Rights Reserved

	The actions associated to a telecommand loader are: • The telecommand loading action

Term	**Telecommand Stream**
Definition	The encapsulation of the interface through which the raw telecommand packet data are received. A telecommand stream is characterized by a set of *operations* that can be performed upon it. The operations that can be performed upon a telecommand stream are: • The telecommand stream can be queried for the presence of a new telecommand packet • The raw telecommand data can be read (dedicated operations must be included to allow the header and data fields of the telecommand to be read)

The introduction of dedicated operations to read the various fields of the header and data part of telecommands is essential to ensure that framework telecommands are decoupled from the physical layout of the raw telecommand packets.

5.2 Domain Dictionary Entries for Telemetry Concept

This section defines the domain dictionary entries related to the framework telemetry concept (see sections 4.3 to 4.7).

Term	**Framework Telemetry Packet**
Definition	The abstract representation at software level of a PUS-compliant telemetry packet. A framework telemetry packet encapsulates the *data collection process* for the data in the telemetry packet together with the *conditional checks* that determine whether the packet should be sent or not. This representation is *abstract* in the sense that it is independent of the physical layout of the telemetry packet. A framework telemetry packet is characterized by a set of *attributes*, a set of *operations* that can be performed upon the telemetry packet, and a set of *actions* and of *checks* that the telemetry packet can perform upon itself or upon its environment. The attributes associated to a framework telemetry packet are: • The type and subtype of the telemetry packet • The destination of the packet • The time stamp of the packet The operations that can be performed upon a framework telemetry packet are: • The telemetry packet can be configured • The telemetry packet can be executed • The telemetry packet can be enabled and disabled The actions associated to a framework telemetry packet are: • The configuration action • The update action • The serialization action • The termination action

Copyright 2007 P&P Software GmbH All Rights Reserved

	The checks associated to a framework telecommand are: • The enable check • The hold check • The termination check

The attributes associated to a telemetry packet are derived from the attributes defined by the PUS for the telemetry packet header and for the telemetry packet data field header. Since a framework telemetry packet is the representation of a PUS telemetry packet within the application that sends it, only PUS attributes that are relevant to the sending application are included.

Term	**Telemetry Manager**
Definition	The encapsulation of a set of pending framework telemetry packets and of the logic to execute them. A telemetry manager is characterized by a set of *attributes* and by a set of *operations* that can be performed upon it. The attributes associated to a telecommand manager are: • The list of pending framework telemetry packets The operations that can be performed upon a telemetry manager are: • The telemetry manager can be activated • A telemetry packet can be loaded in the telemetry manager

Term	**Telemetry Stream**
Definition	The encapsulation of the interface through which the framework telemetry packets are serialized and transformed into raw telemetry packets. A telemetry stream is characterized by a set of *operations* that can be performed upon it. The operations that can be performed upon a telemetry stream are: • The telemetry stream can be queried for its readiness to receive the data from a new telemetry packet • The raw telemetry data can be written to the telemetry stream (dedicated write operations must be included to allow the header and data fields of the telemetry packet to be written)

The introduction of dedicated operations to write the various fields of the header and data part of telemetry packets is essential to ensure that framework telemetry packets are decoupled from the physical layout of the raw telemetry packets.

Copyright 2007 P&P Software GmbH All Rights Reserved

6 DH FRAMEWORK – SHARED PROPERTIES

This section defines the shared properties of the DH Framework. The shared properties are stated in tables with the following format:

<Property Identifier>	<Statement of the Property>

The definition of the properties is made in the following subsections. Each subsection presents the properties related to one or a small set of entries in the domain dictionary of the DH Framework.

6.1 Shared Properties for Telecommand Concept

This section defines the shared properties associated to the telecommand-related entries in the domain dictionary (see section 5.1).

6.1.1 Telecommand Execution

The lifecycle of a framework telecommand is informally sketched in figure 4.3-1. The properties defined in this section define this lifecyle more precisely.

The only operations that can be performed upon a telecommand are the *execute* and *abort* operations. Nominally, execution of a telecommand should result in its going through the sequence of states shown in figure 4.3-1 (ACCEPTED, STARTED, IN_PROGRESS, COMPLETED). An abort request may cause the sequence to be aborted. Transition from one state to the next is controlled by checks. Entry into a state is accompanied by the execution of a telecommand action.

The telecommand cannot change its internal state autonomously. Changes of its internal state can only occur as a result of an execution or abort operation being performed (by a telecommand manager) upon the telecommand.

P6.1.1-1	*A telecommand can change its internal state only in response to an execute or to an abort operation being performed upon it.*
P6.1.1-2	*When a telecommand is executed for the first time, it enters state ACCEPTED if its acceptance check is passed, otherwise it is aborted.*
P6.1.1-3	*If an ACCEPTED telecommand is executed, it performs its ready check and, if this is passed, it attempts to enter state STARTED. If it is not successful, it remains in state ACCEPTED.*
P6.1.1-4	*A telecommand can enter state INPROGRESS only if its start check is passed.*
P6.1.1-5	*If a STARTED telecommand is executed, it performs the progress check and, if this is passed, the telecommand enters state INPROGRESS, otherwise it is aborted.*
P6.1.1-6	*If an IN_PROGRESS telecommand is executed, it performs the progress check and, depending on its outcome, it either re-enters state INPROGRESS, or it attempts to enter state COMPLETED, or else it is aborted.*
P6.1.1-7	*A telecommand can enter state COMPLETED only if its completion check is passed. Otherwise it is aborted.*
P6.1.1-8	*When a telecommand becomes STARTED, it executes its start action.*
P6.1.1-9	*Every time a telecommand enters state INPROGRESS, it executes its progress action.*
P6.1.1-10	*When a telecommand becomes COMPLETED, it executes its completion*

Copyright 2007 P&P Software GmbH All Rights Reserved

	action.
P6.1.1-11	*When a telecommand is aborted, it executes its abort action.*
P6.1.1-12	*Execution of a telecommand that is aborted or COMPLETED has no effect.*

6.1.2 Telecommand Management

The telecommand manager maintains a list of pending telecommands and is responsible for executing them. The telecommand manager executes its pending telecommands when it is activated. Note that the order in which the telecommands are executed is undefined.

The telecommand manager executes the telecommands by performing the *execute* operation upon them. The telecommand manager is also responsible for checking when telecommands have reached the end of their life. Telecommands can reach the end of their life in two ways: either by reaching state COMPLETED or by being aborted. If the telecommand manager detects a telecommand that has reached the end of its life, it simply removes it from its list of pending telecommands.

Two abort operations can be performed on a telecommand manager. In one case, the abort operation is aimed at aborting one particular pending telecommand. In the other case, it is aimed at aborting all pending telecommands. In the first case, the target telecommand is aborted (i.e. its *abort* operation is executed) and then it is removed from the list of pending telecommands. In the second case, the telecommand manager is effectively reset with all its pending telecommands being aborted and the list of pending telecommands being cleared.

P6.1.2-1	*If a telecommand is loaded in a telecommand manager, then the telecommand is added to the list of pending telecommands.*
P6.1.2-2	*When a telecommand manager is activated, it executes all the telecommands in its list of pending telecommands.*
P6.1.2-3	*When a telecommand manager is activated, it removes from the list of pending telecommands the telecommands that have been aborted or that are in state COMPLETED.*
P6.1.2-4	*When a telecommand manager is asked to abort one of its pending telecommands, it aborts the telecommands and removes it from its list of pending telecommands.*

6.1.3 Telecommand Loading

The telecommand loader reads the raw telecommand data from the telecommand stream, uses them to build the corresponding framework telecommand component, and loads the newly created telecommand component onto a telecommand manager.

Since there can be several telecommand managers within the same application, the telecommand loader implements the logic that decides where each telecommand should be loaded. This logic is encapsulated in the telecommand loading action.

The telecommand loader reads the raw telecommand data when it is activated. This should not be taken to imply that a polling mechanism must be used to collect telecommands since the activation signal might be linked to the arrival of a new raw telecommand. The logic that decides when to activate the telecommand loader is outside the DH Framework. In this sense, the DH Framework neither enforces nor assumes a particular mechanism for detecting and responding to the arrival of raw telecommands.

P6.1.3-1	*When a telecommand loader is activated, it reads the raw telecommand data (if any are available) from the telecommand stream, it creates the framework telecommand component, and it executes the telecommand loading action.*

Copyright 2007 P&P Software GmbH All Rights Reserved

6.2 Shared Properties for Telemetry Concept

This section defines the shared properties associated to the telemetry-related entries in the domain dictionary (see section 5.2).

6.2.1 Telemetry Packet Configuration

Telemetry packets can only be used if they have been configured. Configuration is performed by the application component that wishes to create a telemetry packet and send it to some destination application[4].

Configuration consists in providing sufficient information to the telemetry packet to allow it to create the raw telemetry packet.

Configuration is performed in two steps. In the first step, the application component provides the configuration information to the telemetry packet. This is done in an entirely application-specific manner and therefore this first step cannot be modelled at framework level. In the second step, the *configure* operation is performed on the telemetry packet and this causes the telemetry packet to perform its *configuration action*. In the configuration action, the telemetry packet uses the information provided by the application component in the first step to internally configure itself. The content of the configuration action is obviously application specific and hence the configuration action is a factor of variation of the DH Framework.

P6.2.1-1	*When a telemetry packet is configured, it performs its configuration action.*

6.2.2 Telemetry Packets Execution

Telemetry packets are executed by their telemetry manager. Telemetry packets can only be executed if they have been configured. Normally, telemetry packets are configured by the application component that creates them. Hence, telemetry packets that have been loaded into a telemetry manager should already be configured.

Telemetry managers cyclically execute a telemetry packet (until the packet declares itself to be terminated). During an execution cycle, a telemetry packet: (1) checks whether it is enabled or held, (2) updates its content (if it is neither disabled nor held), (2) serializes its content to the telemetry stream, and (4) checks whether it is terminated.

P6.2.2-1	*Only telemetry packets that have been configured can be executed.*
P6.2.2-2	*When a telemetry packet is executed, it performs its enable check to verify whether the packet is enabled.*
P6.2.2-3	*When a telemetry packet is executed, it performs its hold check to verify whether the packet is being held.*
P6.2.2-4	*Execution of a telemetry packet that is neither disabled nor held results in the telemetry packet performing first its update action, and then its serialization action.*
P6.2.2-5	*When a telemetry packet is serialized, it writes its content to the telemetry stream associated to the telemetry manager that executes the packet.*
P6.2.2-6	*After performing the serialization action, a telemetry packet performs its termination check to verify whether it is terminated.*
P6.2.2-7	*If the termination check indicates that the packet is terminated, then the telemetry packet executes its termination action.*

[4] Note that in cases where the telemetry packet must be sent in response to a telecommand, then the application component that creates and configures the telemetry packet may be a telecommand component.

Copyright 2007 P&P Software GmbH All Rights Reserved

6.2.3 Telemetry Packet Management

The telemetry manager maintains a list of pending telemetry packets and is responsible for executing them. The telemetry manager executes its pending telemetry packets when it is activated. Note that the order in which the telemetry packets are executed is undefined.

The telemetry manager checks that telemetry packets are configured before adding them to its list of pending telemetry packets. Note that no action is defined at framework level for the case of a telemetry manager finding that an unconfigured telemetry packet is being loaded into it. At framework level, the telemetry manager simply discards the packet. It is up to the applications to specify notification mechanisms or other remedial actions to be taken in such a case[5].

The telemetry manager executes the telemetry packets by performing the *execute* operation upon them. The telemetry manager is also responsible for checking when telemetry packets have reached the end of their life. Telemetry packets have reached the end of their life when their termination check indicates that they are terminated. If the telemetry manager detects a telemetry packet that has reached the end of its life, it simply removes it from its list of pending packets.

P6.2.3-1	*If a telemetry packet is loaded in a telemetry manager and if the packet is configured, then the telemetry packet is added to the list of pending telemetry packets.*
P6.2.3-2	*When a telemetry packet is activated, it executes all the telemetry packets in its list of pending telemetry packets.*
P6.2.3-3	*When a telemetry manager is activated, it removes from the list of pending telemetry packets the packets that, after their execution, are terminated.*

[5] The attempt by an application component to load an unconfigured telemetry packet into a telemetry manager represents an example of software design fault. On-board systems are normally not designed to counter design faults.

Copyright 2007 P&P Software GmbH All Rights Reserved

7 DH FRAMEWORK – FACTORS OF VARIATION

This section defines the factors of variation for the DH Framework. The factors of variation are defined in tables with the following format:

<Identifier>	<Name of the Factor of Variation>
Description	<Description of the Factor of Variation>
Default	<Default Value of the Factor of Variation>
Range	<Legal Range of the Factor of Variation>

The description of the factor of variation includes a reference to the property to which the factor of variation applies.

The CORDET Methodology prescribes that factors of variations be defined also in terms of their mutual interactions and in particular in terms of the constraints on their legal combinations. These interactions are captured in the feature model for the DH Framework that is documented in the next section.

7.1 Attributes as Factors of Variation

All the attributes defined for the entries in the domain dictionary represent factors of variation since the framework only defines the *existence* of the attributes and the application designer is free to set their values.

Factors of variations linked to attributes are regarded as trivial and implicitly defined by the domain dictionary and are therefore not further described in this section.

7.2 Factors of Variation for Telecommand Concept

This section defines the factors of variation associated to the telecommand entry in the domain dictionary (see section 5.1).

FV7.2-1	**Telecommand Start Action**
Description	The implementation of the start action used in property P6.1.1-8 is application-specific.
Default	The default implementation of the start action returns without taking action.
Range	Unrestricted.

FV7.2-2	**Telecommand Progress Action**
Description	The implementation of the progress action used in property P6.1.1-9 is application-specific.
Default	There is no default implementation for the progress action.
Range	Unrestricted.

FV7.2-3	**Telecommand Completion Action**
Description	The implementation of the completion action used in property P6.1.1-10 is application-specific.

Copyright 2007 P&P Software GmbH All Rights Reserved

Default	The default implementation of the completion action returns without taking action.
Range	Unrestricted.

FV7.2-4	**Telecommand Abort Action**
Description	The implementation of the abort action used in property P6.1.1-11 is application-specific.
Default	The default implementation of the abort action returns without taking action.
Range	Unrestricted.

FV7.2-4	**Telecommand Acceptance Check**
Description	The implementation of the acceptance check used in property P6.1.1-2 is application-specific.
Default	The default implementation of the acceptance check returns TRUE (telecommand is successfully accepted).
Range	This check must return either TRUE (telecommand is successfully accepted) or FALSE (telecommand is not accepted).

FV7.2-4	**Telecommand Ready Check**
Description	The implementation of the ready check used in property P6.1.1-3 is application-specific.
Default	The default implementation of the ready check returns TRUE (telecommand is ready to start execution).
Range	This check must return either TRUE (telecommand is ready to start execution) or FALSE (telecommand is not yet ready to start execution).

FV7.2-4	**Telecommand Start Check**
Description	The implementation of the start check used in property P6.1.1-4 is application-specific.
Default	The default implementation of the start check returns TRUE (telecommand execution can start successfully).
Range	This check must return either TRUE (telecommand execution can start) or FALSE (telecommand execution cannot be started).

FV7.2-4	**Telecommand Progress Check**
Description	The implementation of the progress check used in properties P6.1.1-5 and P6.1.1-6 is application-specific.
Default	The default implementation of the progress check returns “telecommand has successfully completed its progress”.
Range	This check must return one of three values indicating: (1) telecommand

Copyright 2007 P&P Software GmbH All Rights Reserved

	execution is proceeding successfully but has not yet completed; (2) telecommand has successfully completed its progress; (3) telecommand cannot continue its execution.

FV7.2-4	**Telecommand Completion Check**
Description	The implementation of the completion check used in property P6.1.1-7 is application-specific.
Default	The default implementation of the completion check returns TRUE (telecommand can successfully complete execution).
Range	This check must return either TRUE (telecommand completed execution successfully) or FALSE (telecommand did not complete execution successfully).

7.3 Factors of Variation for Telecommand Loading Concept

This section defines the factors of variation associated to the telecommand loader entry in the domain dictionary (see section 5.1).

FV7.3-1	**Telecommand Loading Action**
Description	The implementation of the telecommand loading action used in property P6.1.3-1 is application-specific.
Default	There is no default implementation for the telecommand loading action.
Range	This action must result in the framework telecommand component created by the telecommand loader being loaded in at least one telecommand manager.

7.4 Factors of Variation for Telecommand Stream Concept

This section defines the factors of variation associated to the telecommand stream entry in the domain dictionary (see section 5.1). No functionality can be defined for the telecommand stream at framework level. The framework only identifies the operations to be provided by the telecommand stream but there are no functional commonalities in the implementation of these operations across different applications. Hence, the telecommand stream is considered as one single factor of variation.

FV7.4-1	**Telecommand Stream**
Description	The implementation of all the operations defined on the telecommand stream and used in property P6.1.3-1 is application-specific.
Default	There is no default implementation for the telecommand stream operations.
Range	Unrestricted.

7.5 Factors of Variation for Telemetry Concept

This section defines the factors of variation associated to the telemetry entry in the domain dictionary (see section 5.2).

FV7.5-1	**Telemetry Configuration Action**

Copyright 2007 P&P Software GmbH All Rights Reserved

Description	The implementation of the telemetry configuration action used in property P6.2.1-1 is application-specific.
Default	The default implementation of this action returns without taking any action.
Range	Unrestricted.

FV7.5-2	**Telemetry Update Action**
Description	The implementation of the telemetry update action used in property P6.2.2-4 is application-specific.
Default	The default implementation of this action returns without taking any action.
Range	Unrestricted.

FV7.5-3	**Telemetry Serialization Action**
Description	The implementation of the telemetry update action used in property P6.2.2-4 is application-specific.
Default	There is no default implementation for this action.
Range	Unrestricted.

FV7.5-4	**Telemetry Termination Action**
Description	The implementation of the telemetry termination action used in property P6.2.2-7 is application-specific.
Default	The default implementation of this action returns without taking any action.
Range	Unrestricted.

FV7.5-5	**Telemetry Hold Check**
Description	The implementation of the telemetry hold check used in property P6.2.2-3 is application-specific.
Default	The default implementation of this check returns FALSE (packet is not held).
Range	This check must return either TRUE (telemetry packet is being held) or FALSE (telemetry packet is not being held).

FV7.5-5	**Telemetry Termination Check**
Description	The implementation of the telemetry termination check used in property P6.2.2-6 is application-specific.
Default	The default implementation of this check returns TRUE (packet has terminated).
Range	This check must return either TRUE (telemetry packet has terminated) or FALSE (telemetry packet has not yet terminated).

Copyright 2007 P&P Software GmbH All Rights Reserved

Note that there the enable check (see property P6.2.2-3) is not a factor of variation. This is because the enable/disable mechanism for telemetry packets is entirely implemented at framework level.

7.6 Factors of Variation for Telemetry Stream Concept

This section defines the factors of variation associated to the telemetry stream entry in the domain dictionary (see section 5.2). No functionality can be defined for the telemetry stream at framework level. The framework only identifies the operations to be provided by the telemetry stream but there are no functional commonalities in the implementation of these operations across different applications. Hence, the telemetry stream is considered as one single factor of variation.

FV7.6-1	**Telemetry Stream**
Description	The implementation of all the operations defined on the telemetry stream and used in property P6.2.2-1 is application-specific.
Default	There is no default implementation for the telemetry stream operations.
Range	Unrestricted.

Copyright 2007 P&P Software GmbH All Rights Reserved

8 DH FRAMEWORK – FEATURE MODEL

This section describes the feature model for the DH Framework..

The feature model of the DH Framework is built with the XFeature tool[6]. This section only gives an overview of the framework feature model. The feature model itself is available in electronic form and can be downloaded from the "Domain Analysis" page of the CORDET Web Site[7].

XFeature is a meta-modelling tool. In order to be used to construct a feature model, it must first be configured with a feature meta-model and a display model (that defines how the feature model is rendered graphically). In this project, XFeature was used in the so-called "FD Configuration". This is one of the default configurations of the XFeature tools. It was defined in the ASSERT project and is documented in RD-33. For the convenience of the reader, the next subsection gives a brief overview of this configuration.

8.1 Feature Meta-Model

The FD feature meta-model allows feature models of the kind shown in figure 8.1-1 to be built. A feature model has the usual tree structure. Two main kinds of nodes can be recognized: "feature nodes" and "group nodes". Feature nodes represent a feature of the family that is being modelled. They are visually represented as rectangular boxes drawn either with solid lines or with dashed lines. Feature nodes have a cardinality expressed as a range: <m..n>. The cardinality defines the minimum and maximum number of times that the feature can be instantiated in applications within the family.

Features nodes drawn with solid lines are called "solitary features". Solitary features are mandatory features (they must appear in all applications instantiated from the family).

Feature nodes drawn with dashed lines can only enter a feature model as children of a group node. Group nodes are used as umbrellas for groups of feature nodes that are subject to a local constraint. Group nodes have a cardinality expressed as a range. A group cardinality of: <m..n> indicates that applications instantiated from the family must have at least m and no more than n features selected from among the features in the group. Group nodes are visually represented as rounded dots attached to their parent feature.

As an example, consider the feature model in the figure. This feature meta-model states that:

- feature A1 can be present as a single instance (singleton feature). A singleton cardinality (cardinality <1..1>) is the default and can also be omitted.
- feature B1 is an optional sub-feature of A1 (it belongs to a group with a cardinality of <0..1> which indicates that the feature may be present in or it may be left out of applications instantiated from the family)
- feature B2 is a mandatory sub-feature of A1 (it is a solitary feature directly attached to feature A1 which indicates that the feature must be present in all applications where feature A1 appears)
- features B3 and B4 are alternative sub-features of A1 (they belong to the same group with cardinality <1..1> which indicates that one and only one of the two features must be present in all applications instantiated from the family)

[6] The tool can be downloaded as free and open software from its home page at this address: http://www.pnp-software.com/XFeature/

[7]The CORDET Web Site is at: http://www.pnp-software.com/cordet

Copyright 2007 P&P Software GmbH All Rights Reserved

- features B5 to B7 are alternative sub-features of A1 of which up to two can be selected simultaneously (they belong to the same group with cardinality <0..2> which indicates that either none, or just one, or any two features from the group can be present in all applications instantiated from the family). All three features can be present in applications either in a single instance or in two instances (their cardinality is: <1..2>).

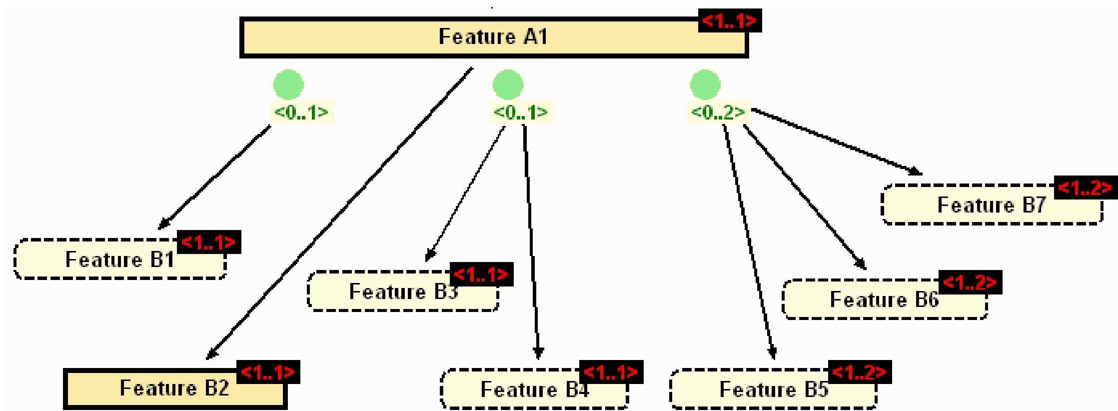

Fig. 8.1-1: Example of Feature Model created with the FD-Configuration in XFeature

The FD feature meta-model also allows to attach attributes (so-called “properties”) to each feature but this functionality is not used in the feature model of the Control Framework and hence it is not discussed further.

The XFeature tool allows global constraints to be defined over a feature model. A global constraint is a constraint that affect features that are not children of the same parent feature. Several kinds of global constraints can be defined in XFeature. The simplest ones are the “require” and the “exclude” constraints. A require constraint allows the designer to specify that the presence of a feature F in an application requires certain other features R1...Rn to be present too. An exclude constraint allows the designer to specify that the presence of a feature F in an application is incompatible with the presence of certain other features R1...Rn.

8.2 Top-Level Features

Figure 8.2-1 shows the top-level features of the DH Framework Feature Model.

The root element is called `DhApplication` since it represents a DH Application instantiated from the framework. The feature mode in the figure simply indicates that an DH Application consists of two separate functionalities: the telemetry functionality and the telecommanding functionality.

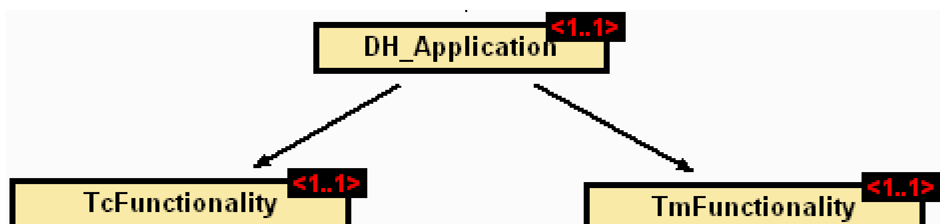

Fig. 8.2-1: Top-Level Features of DH Framework Feature Model

Copyright 2007 P&P Software GmbH All Rights Reserved

8.3 Telecommand Functionality Features

Figure 8.3-1 expands the `TcFunctionality` feature in the previous figure to show the features for the telecommanding functionality. This functionality is broken up into three subfeatures: one or more telecommands, one or more telecommand managers, and one or more telecommand loaders.

The telecommand manager feature cannot be further broken up into lower level items. This is because there are no factors of variations associated to the telecommand manager.

The telecommand loader feature can be broken up into two sub-features: the telecommand loading action and the telecommand stream. The latter, as discussed in section 7.4, is treated as a single factor of variation and cannot therefore be further broken up into lower-level features.

The structure of the telecommand feature is presented in the next subsection.

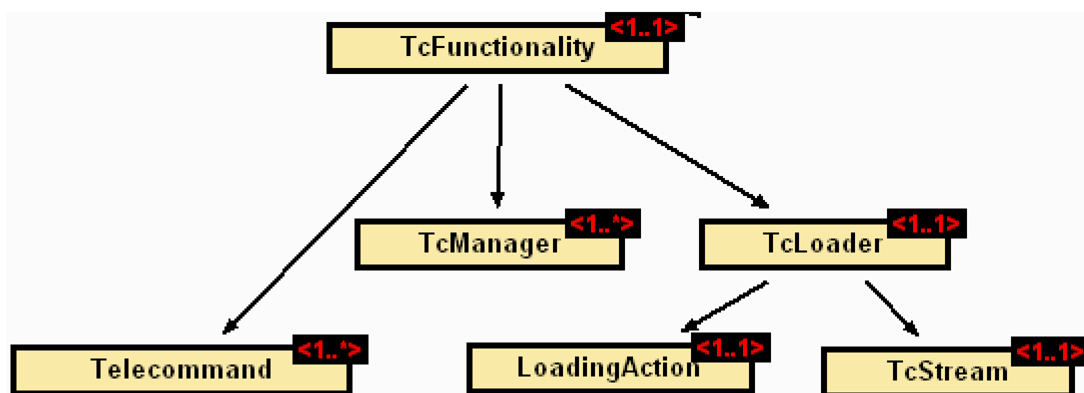

Fig. 8.3-1: Telecommanding Functionality Features of DH Framework Feature Model

8.4 Telecommand Features

Figure 8.4-1 expands the `Telecommand` feature in the previous figure to show the features for the telecommand concept.

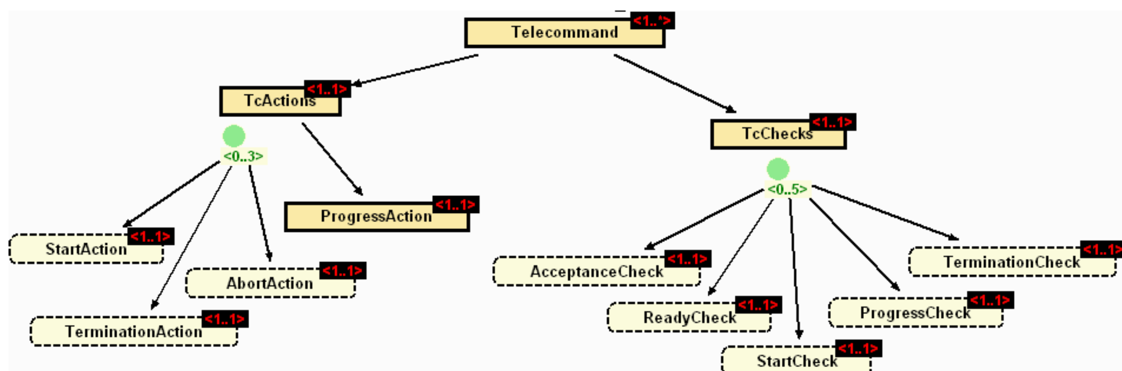

Fig. 8.4-1: Telecommand Feature of DH Framework Feature Model

The telecommand feature gives rise to two subfeatures representing the telecommand actions and the telecommand checks. Among the telecommand actions, the Progress Action is modelled as a mandatory feature. This reflects the fact that the framework does not provide any default implementation for the progress action factor of variation (FV7.2-2) and hence application developers must provide their own implementation (in other words, they do not have the option of just taking over the default implementation provided by the framework). The other three telecommand actions are instead optional and therefore grouped together in a group with cardinality 0 to 3.

Copyright 2007 P&P Software GmbH All Rights Reserved

The figure identifies five telecommand checks and all of them are optionals. They are therefore grouped together in a single group with cardinality 0 to 5.

8.5 Telemetry Functionality Features

Figure 8.5-1 expands the `TmFunctionality` feature in the top-level feature diagram. This functionality is broken up into three subfeatures: one or more telemetry packets, one or more telemetry managers, and one telemetry stream.

The telemetry manager feature is characterized by one single subfeature: the telemetry stream (as will be recalled, to each telemetry manager, one telemetry stream is associated).

The telecommand stream, as discussed in section 7.6, is treated as a single factor of variation and cannot therefore be further broken up into lower-level features.

The structure of the telemetry packet feature is presented in the next subsection.

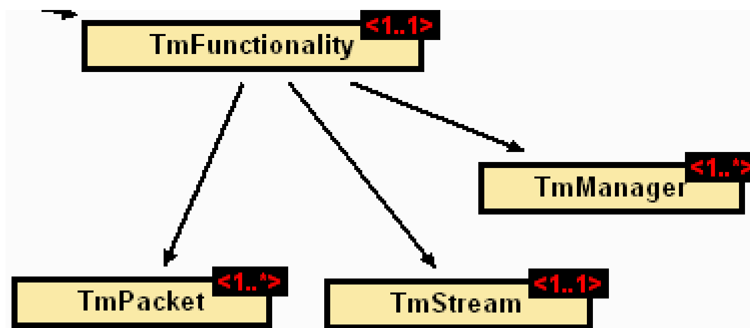

Fig. 8.5-1: Telemetry Functionality Feature of DH Framework Feature Model

8.6 Telemetry Packet Features

Figure 8.6-1 expands the `TmPacket` feature in the previous figure to show the features for the telemetry packet concept.

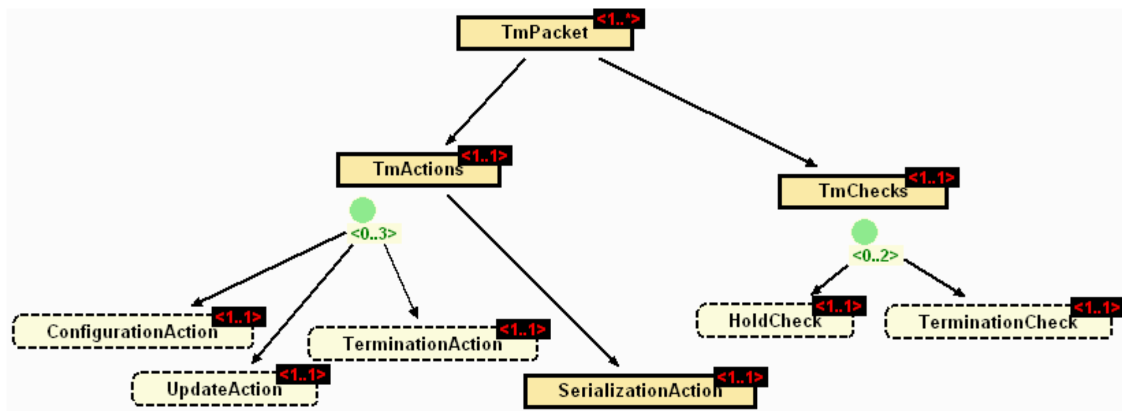

Fig. 8.6-1: Telemetry Packet Feature of DH Framework Feature Model

The telemetry packet feature gives rise to two subfeatures representing the telemetry actions and the telemetry checks. Among the telemetry actions, the Serialization Action is modelled as a mandatory feature. This reflects the fact that the framework does not provide any default implementation for the serialization action factor of variation (FV7.5-3) and hence application developers must provide their own implementation (in other words, they do not have the option of just taking over the default implementation provided by the framework). The other three telemetry actions are instead optional and therefore grouped together in a group with cardinality 0 to 3.

Copyright 2007 P&P Software GmbH All Rights Reserved

The figure identifies two telemetry checks and both are optionals. They are therefore grouped together in a single group with cardinality 0 to 2.

Copyright 2007 P&P Software GmbH All Rights Reserved

9 CONTROL FRAMEWORK – OVERVIEW

This section presents an overview of the Control Framework at domain analysis level. This section is intended to place the reader in a better position to appreciate the formal definition of the domain model of the Control Framework as it is presented in the next two sections.

9.1 Heritage

The primary heritage of the Control Framework is in the part of the OBS Framework [RD18] dealing with the implementation and management of *control blocks, data items, data pools,* and *parameter database*. More information can be found in the relevant entries of the OBS Framework domain dictionary and in the links to the associated design patterns.

9.2 Domain Demarcation

The core of an AOCS application is the implementation of transfer functions that transform measurements from a set of sensors into commands for a set of actuators. Such transfer functions are implemented as digital filters that are characterized by a set of inputs, a set of outputs, an internal state, and an algorithm to compute the next set of outputs from the latest set of inputs and the current internal state.

Peripheral functionalities that are often found in AOCS applications are:

- Management of AOCS operational modes;
- Management of the external sensors and actuators;
- Implementation of failure detection and isolation checks and recovery actions (FDIR);
- Execution of AOCS-specific telecommands;
- Generation of AOCS-specific telemetry packets.

The Control Framework directly covers the management of the AOCS operational mode through the operation mode concept (see section 9.6) and the activity manager concept (see section 9.7).

The Control Framework does not cover the management of the external sensors and actuators. This is due to the fact that the interfaces of AOCS sensors and actuators are neither standardized nor do they exhibit any significant commonalities in existing missions. Standardization of interfaces to external units (not just for the AOCS subsystem) is possible but this is done at the bus interface level, not at the functional level. Such functionalities are therefore not specific to the AOCS and are best left out of a framework targeted at the AOCS.

The FDIR functionalities are not directly included in the Control Framework in its present form. It is believed that such functionalities present sufficient commonalities to be implemented in reusable and adaptable component and they might be included in a future release of the Control Framework.

The management of the AOCS telecommands and AOCS telemetry is not directly included in the Control Framework. However, the DH Framework is interoperable with the Control Framework (see methodological requirement MR7.3-2) and hence these functionalities are supported by the two frameworks taken together.

9.3 The Activity Concept

The key concept of the Control Framework is that of *activity*. An activity encapsulates a transaction-like, passive, data processing algorithm. An activity is characterized by a set of inputs, a set of outputs, an internal state, and an algorithm to compute the next set of outputs from the latest set of inputs and the current internal state.

Copyright 2007 P&P Software GmbH All Rights Reserved

The basic operation that can be performed upon an activity is the *execution*. When an activity is executed, it perform three successive and non-overlapping steps:

- *input data read*: the activity reads its inputs from some external data source,
- *propagation*: the activity propagates its inputs and its internal state;
- *output data write*: the activity writes its outputs to some external data sink.

An activity is *transaction-like* in the sense that, once it has read its set of inputs, its operation is completely autonomous from what happens in other parts of the application within which it is embedded. The computation of its outputs depends solely on the inputs and on the internal state of the activity.

An activity is *passive* in the sense that it does not have an own thread of control. Some other component in its host application is responsible for executing them. The framework provides the activity manager as a component to execute activities (see section 9.7 below).

An activity as defined here is the natural means to host the transfer functions that AOCS's use to process sensor measurements and to compute actuator commands. Activities, however, can also be used to encapsulate other kinds of AOCS operations such as FDIR functions, or execution of sensor or actuator management functions, or computation of attitude and orbit profiles.

In addition to execution, there are two other kinds of operations that may be performed on activities. Firstly, *initialization operations* must be performed on activities during the application initialization phase. Secondly *control operations* may be performed upon it at any time. The objective of such operations is to send control signals to the activities to, for instance, enable and disable them or to hold or release them.

When an activity is *disabled* or *held*, then executing it will have no effect (i.e. its internal state and its inputs will not be propagated and its outputs will not be updated). Thus, disabling and holding an activity have the same result on its execution. Disabling an activity additionally causes its internal state to be reset to its initial value.

An activity that is disabled can be brought back to its nominal mode of operation by *enabling* it. Similarly, an activity that is held can be brought back to its nominal mode of operation by *resuming* it.

The set of activities defined in an application is statically determined and should not change at run time.

Each activity has a unique *identifier*. The identifier is the means through which components that wish to perform control operations upon an activity specify the target activity. Thus, for instance, a telecommand that is responsible for enabling and disabling an activity uses the activity identifier to specify which activity should be enabled or disabled.

9.4 The Data Pool Concept

Activities read their inputs from an external data source and write their outputs to an external data sink. In the Control Framework, this external data source and external data sink is called *data pool*. The data pool in other words is a read/write repository of data that is used by activities as a source of their inputs and as a destination for their outputs.

Given the transactional character of activities, the data pool is the simplest means through which activities can exchange data. If an activity A must produce an input for another activity B, then the link between the two activities must take place as follows:

- activity A generates an output and writes it to the data pool;
- activity B reads its input from the same location in the data pool where A wrote its output.

Copyright 2007 P&P Software GmbH All Rights Reserved

Obviously, the link between the two activities is only effective if their activation is properly synchronized, namely if activity B is made to read its input only after activity A has terminated writing its output. The implementation of such synchronization mechanisms is a non-functional issue and as such it is outside the scope of the Control Framework.

The data pool is also the only means through which the activities can exchange data with other parts of the application within which they are embedded. Thus, for instance, the fact that sensor and actuator management is not included in the Control Framework means that some other software components (not provided by the Control Framework and not derived from the Control Framework components) are responsible for placing the sensor measurements into the data pool and for collecting the actuator commands from the data pool and sending them to the external actuators.

The individual items in the data pool are called *data items*. Thus, a data item is an encapsulation of a single piece of data that cannot be further broken down into lower-level elements and that has one single value. The data item provides the means to allow this value to be read and to be set.

Each data item in a data pool has an *identifier* through which the data item is accessed.

Activities often manipulate *data structures*. A data structure consist of a set of logically related data items. Three kinds of data structure are of special important in the AOCS domain and are therefore explicitly recognized by the Control Framwork: vectors, quaternions, and matrices.

Conceptually, data items encapsulate one single elementary datum with a single value (the *current value* of the data item). The possibility was considered of building some additional logic into a data item to allow the following information to be associated to a data item:

- a *validity status* that defines whether the data item is valid or not
- a *back-up value* that becomes the value of the data item when its validity status becomes equal to “not valid”
- a *default value* that represent the value of the data item at initialization time.

In the present version of the Control Framework, only the default value attribute was retained. The validity status and the back-up value attributes might be useful to support FDIR functions but it was decided that the increase in complexity they would introduce is not justified. When validity and back-up data are required, application designers must therefore model them through dedicated data items

Activities are configured with links to the data items in the data pool from which they read their inputs and to which they write their outputs. It is natural to assume that these links are semi-permanent but the Control Framework does not enforce this assumption. Applications can in principle reconfigure the activities dynamically by re-routing their input and output links.

There is no constraint that only one data pool may be implemented in an application. The same application may have more data pools if there is a need to represents several clusters of logically related data.

The Control Framework does not dictate any *implementation* for the data pool. The concept of data pool simply calls for the definition of an *interface* that allows individual data items to be accessed through a global identifier. The data pool should thus be seen as a kind of centralized registry that controls access to all AOCS data that are shared across activities.

In particular, the data pool concept does *not* imply or require that components physically copy data in and out of the data pool. This would be a simple way to implement the data pool concept but it is not the only one. In an alternative implementation, the data pool returns pointers to the data items and activities can manipulate the shared data without ever copying

Copyright 2007 P&P Software GmbH All Rights Reserved

them in or out of the data pool. In still another implementation, the data remain physically located in the components that produce them and the data pool only hold pointers to the data.

Thus, use of the data pool concept does not necessarily imply any overheads due to copying data into and out of the data pool.

Use of the data pool concept does, however, imply that all data shared between activities become "global" since the data pool is by definition a globally accessible component. This introduces a certain implementation risk due to the possibility of data corruption. This risk however is removed if the code that links the activities (which are the only entities capable of accessing the data pool) to the data pool is generated automatically from a specification of the data that each activity requires as inputs and generates as outputs.

Note that the number of items in the data pool depends on the granularity of the activities. The data pool only contains data that are shared across activities. Hence, an application that breaks up its processing into a large number of simple activities will probably require a correspondingly large number of activities. Conversely, an application that only implements a small number of high-level activities will probably only need a data pool of small size.

9.4.1 Data Pool Concept vs Localized Concept

The data pool concept has been selected as the baseline concept for storing data that are shared by activities. Other choices would have been possible and were considered during the domain analysis activities of the CORDET Project. Given the importance of the data pool concept, it is useful to record the reasons that led to their rejection.

There is a fundamental design choice between a data pool concept where data items are accessed by name through a centralized registry, and a *localized concept* where access to a data item requires knowledge of where the data item is stored as well as of its name. The localized concept had been tried in a past project for software frameworks for AOCS[8]. In that case, the concepts of *data sinks* and *data sources* were defined to represent a generic source of input data and a generic destination for output data.

If this approach had been adopted for the Control Framework, then every activity would have to behave like a data sink and a data source. In practice, this would mean that each activity would have to implement two interfaces – the `DataSink` and `DataSource` interfaces – and, through these interfaces, it would have to be linked to the activities that generate its inputs and to the activities that consume its outputs.

The localized concept is feasible but the reason that led to the choice of a data pool concept are:

(1) With the localized concept, every activity would have to implement two additional interfaces (the `DataSink` and `DataSource` interfaces). This would increase implementation complexity.

(2) With the data pool concept, in order to access a data item, an activity only needs to know its name. With the localized concept, the activity needs to know its name and its location (the component where it is stored). The data pool thus reduces complexity because it acts like a kind of centralized registry for all AOCS data.

(3) With the localized concept, every component must be linked to (at least) two other components (the producer of its inputs and the consumer of its outputs – in most cases an activity will be linked to several other activities). With the data pool concept, components only need to be linked to the data pool.

[8]http://control.ee.ethz.ch/~ceg/RealTimeJavaFramework/doc/index.html

Copyright 2007 P&P Software GmbH All Rights Reserved

(4) The data pool makes it easier to define the boundaries of the framework since other (non-framework) components can easily write to the data pool or read from it. With the localized concept, instead, it becomes problematic to define a mechanism to allow a non-framework components to receive data from, or send data to, a framework component.

(5) The data pool makes it easy to add some behaviour to individual data items. Consider for instance the case where one wants to define a default value for every data item, or a validity status, or a back-up value. If one uses a data pool concept, the logic to implement this additional functionality is concentrated in the data pool. In the case of the localized concept, instead, this logic has to be duplicated in every component that can act as either a data sink or a data source.

9.5 The Parameter Database Concept

Activities implement data processing algorithms. The data processing algorithms are normally parameterized. In the Control Framework, the *parameter database* is the location from which the parameters of the algorithms implemented by the activities are read.

Normally, the activities read their parameters only at initialization time but they could also read them at run time as part of a reconfiguration or a reset.

Depending on the application, the values of the parameters in the database might be read from some permanent memory storage medium (a PROM or a mass memory device) or they might be loaded by telecommand.

Activities should not write to the parameter database.

The Control Framework does not enforce a constraint that there should be one single parameter database. Applications are free to instantiate multiple databases.

9.6 The Operational Mode Concept

Most AOCS systems are characterized by one or more operational modes. An operational mode is characterized by a set of sensors and actuators and by a set of algorithms to process the sensor measurements and to generate the actuator commands. Mode changes can be commanded either autonomously by the AOCS application itself or by an external telecommand.

In the Control Framework, an *operational mode* consists of one or more *activities*. The activities encapsulate the processing algorithms and other actions that must be executed in that mode.

Activities that are associated to an operational mode are *in use*. Activities that are not associated to any operational mode are *out of use*.

The same activity can be associated to more than operational mode. This is useful to model the situation where the same actions are performed in several operational modes.

The activities associated to an operational mode are organized as one or more *activity sets*. The operational mode defines a sequential ordering of the activity sets such that the activities in activity set n are executed before the activities in activity set (n+1).

Note that there is no notion of the order in which the activities within an activity set should be executed. The execution model of activities (see section 9.9 below) and their transactional character implies that the effect of executing the activities in an activity set (namely the values of the outputs they produce given a certain set of input and initial state values) is independent of the order in which they are executed.

Copyright 2007 P&P Software GmbH All Rights Reserved

The concept of activity set is introduced to capture a situation very common in AOCS applications. Such applications normally run cyclically and, within each cycle, they define actions that must be executed in sequence. In a typical case, these actions might cover:

- the acquisition of sensor measurements
- the execution of health checks on the sensor measurements
- the computation of control signals (torques, forces, etc)
- the generation of actuator commands
- the handling of telecommands
- the generation of telemetry data

Each action can typically be broken down into lower-level activities. For instance, the acquisition of sensor measurements would normally be implemented as a set of activities that are each targeted at one particular sensor; the computation of control torques might be broken up into three activities, one for each for each a spacecraft axis; etc.

The AOCS designer specifies an ordering constraint among the actions (namely it specifies that the actions listed under the bullets in the above list must be executed in a given sequence) but he does not necessarily specify any ordering constraint among the individual activities within an action (for instance, there may be no constraint on the order in which sensor measurements are processed; or they may be constraint on the order in which the control algorithms for the three spacecraft axes are computed).

The AOCS designer, in other words, only specifies *partial ordering constraints* for the activities. The notion of activity set is introduced precisely to allow application designers to specify such partial ordering constraints for the activities that are executed within the same operational mode. This ordering is *partial* because it only covers the relationship of precedence between activities in different activity sets.

Activities can be dynamically loaded and unloaded from an operational mode. This facilitates the maintenance of on-board applications since the ground can reconfigure a running application by simply changing the activities that are associated to its various operational modes. Finally, it should be noted that, since the Control Framework does not directly cover sensor and actuator management, the definition of the sensors and actuators to be used in a certain mode is implicit in the selection of the applicable activities. This is because the activities are linked to data pool locations and some of these data pool locations hold the sensor measurements or the actuator commands.

9.7 The Activity Manager Concept

The Control Framework defines an *activity manager* as a pre-defined component to manage the operational modes and to execute the activities associated to them. To each activity manager, a number of operational modes (with their activities) is associated. At any given time, one of these operational modes is designated as the *current mode*.

Activity managers can be *activated*. When an activity manager is activated, it executes all the activities that are associated to its current mode. Activation of the activity manager is performed by components outside the Control Framework. In a typical case, the activity manager would be cyclically activated by an external scheduler.

The activity manager encapsulates the mode-switching logic. The current mode can change either as a result of a request sent to the activity manager by some external entity or as a result of an autonomous decision taken by the activity manager itself. For this purpose, the activity manager can monitor the data in the data pool and, based on their values, can decide to update its current mode. This monitoring process is encapsulated in the *mode update check*.

Copyright 2007 P&P Software GmbH All Rights Reserved

A request to perform a change in current mode (originating either from the activity manager itself or from an external entity) is only executed if certain checks are satisfied. Three types of checks are defined:

- The *mode exit check* verifies that the old current mode can be exited.
- The *mode entry check* verifies that the target current mode can be entered.
- The *mode transition check* verifies that the transition from the old to the target mode can be performed (ie it verifies the legality of the transition across two modes)

The change in operational mode is only performed if all three checks are positive.

The presence of the mode update check and of the three checks on the mode transition allows application designers to implement any mode architecture and mode switching logic simply by suitably implementing the checks.

The activity manager is the only component that should execute an activity.

Note that there is no restriction on the number of activity manager instances that may exist within the same application. The application designer is free to instantiate several activity managers and to allocate each of them to a dedicated function. For instance, there might be an activity manager for the control algorithms and another activity manager for the FDIR functions.

Also, each activity manager has its own set of operational modes. Hence, multiplicity of activity managers allows the application designer to have different sets of modes for different on-board functions or to have sub-modes within modes.

Multiplicity of activity managers would also allow to have several threads for the AOCS Application where each thread is responsible for one activity manager.

9.8 Framework Boundaries

The figure below sketches the conceptual boundaries between the part of an application covered by the Control Framework and the remainder of the application.

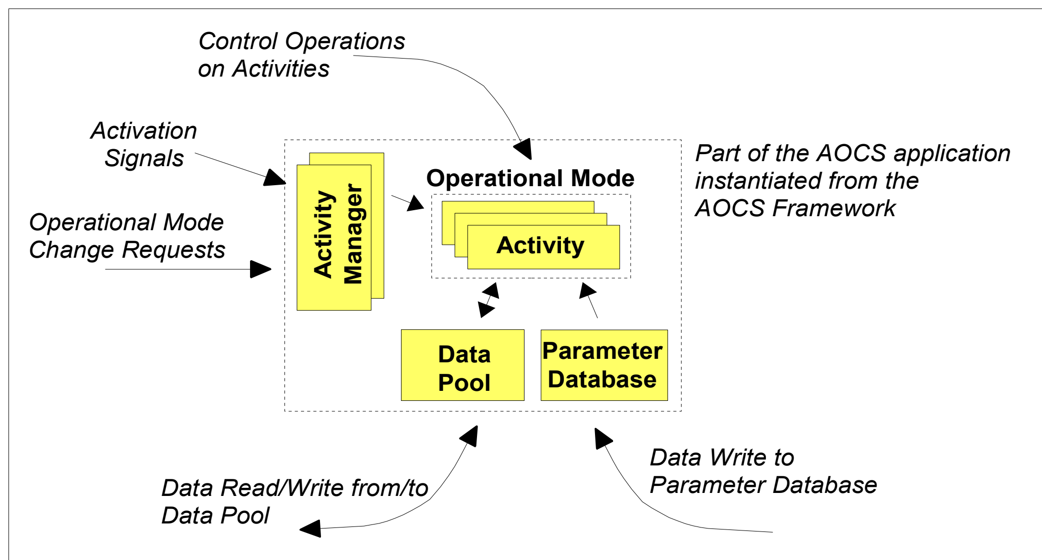

Fig. 9.8-1: Control Framework Boundaries

The core of the functionalities implemented through the framework are implemented as activities. The activities run under the control of an activity manager. The activities read their

Copyright 2007 P&P Software GmbH All Rights Reserved

configuration parameters from the parameter database and they read their data inputs from, and write their data outputs to, the data pool. Hence, interaction with the framework part of the application would typically take place through the following channels:

- through write or read operation on the data pool values;
- through updates of parameter values in the parameter database;
- through change of modes in the activity manager;
- through control operations (e.g. enable or disable) performed directly on an activity;
- through loading or unloading of activities in the current mode.

9.9 Reference Execution Model

The objective of the Control Framework is to provide reusable and adaptable software components to assist the development of AOCS applications. In keeping with the methodological rules defined in [RD35], the Control Framework only covers the functional aspects of an AOCS. The definition of a real-time architecture for the target AOCS application is therefore *not* an objective of the Control Framework.

There is, however, a *reference real-time architecture* that underlies the Control Framework in the sense that the definition and design of its components is optimized for it. Although the framework does not enforce this reference architecture and there is no preclusion to deploying its components on other types of real-time platforms, an understanding of this reference architecture is useful to give an insight into the rationale behind the specification and design choices that have been made for the Control Framework.

The reference execution model behind the Control Framework is based on a time-triggered architecture of the kind commonly used for embedded control systems. Time is divided into consecutive and non-overlapping intervals of the same lengths (*cycles*). The execution of the activity over one execution cycle takes place as follows (see also figure 9.9-1):

- the *input data read* operation is performed at the beginning of the cycle,
- the *propagation* operation is performed after the input data read operation has terminated,
- the *output data write* operation is performed after the propagation operation has terminated and must complete before the end of the cycle.

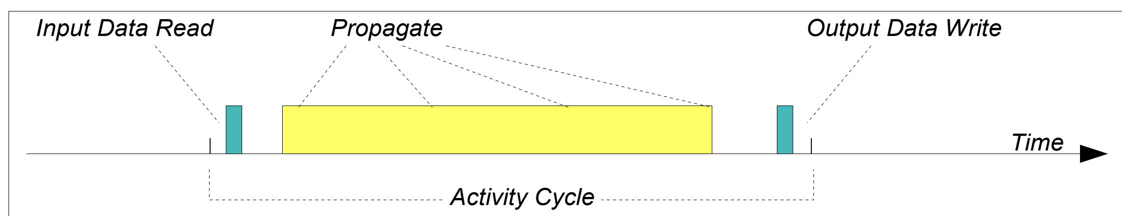

Fig. 9.9-2: Activity Execution

Figure 9.9-1 only shows the steps involved in the execution of one single activity. In the control framework, however, activities are gathered in sets that are associated to operational modes which are in turn controlled by activity managers. In each cycle, several tasks must therefore be performed that relate to:

- the execution of the activities in the current mode of the activity managers,
- the processing of mode change requests and the update of the mode of the activity managers,
- the processing of control operations on the activities (e.g. changes in the enable status of an activity),

Copyright 2007 P&P Software GmbH All Rights Reserved

- the update of data pool values by components outside the Control Framework (this is the task where, for instance, sensor measurements are deposited in the data pool),
- the collection of data values from the data pool by components outside the Control Framework (this is the task where, for instance, commands for the actuators are read from the data pool).

Figure 9.9-2 shows one possible way in which the above tasks can be performed within a tick. The various tasks are performed in dedicated and non-overlapping sub-intervals. Obviously, some changes to the ordering of the sub-intervals are possible. The important point is that the subintervals should not overlap and that the activities listed above be repeated cyclically.

A constraint (the *data integrity constraint*) must be imposed on how the activities are linked to the locations in the data pool from which they read their inputs and to which they write their outputs. This can be formulated as follows. If an activity A writes an output to location y in the data pool, then no other activity that belongs to the same activity set as A should use y as a destination for one of its output. The location y in the data pool, in other words, should be considered to be locked by activity A.

Compliance with the data integrity constraint ensures that activities under the control of the same activity manager do not overwrite each other's outputs. Note that enforcement of integrity constraints across activity managers is a non-functional issues that must be covered by the non-functional part of the application and that is therefore outside the scope of the Control Framework.

A second constraint (the *activity exclusion constraint*) must be imposed on how activities are associated to operational modes. As already mentioned, the same activity may be associated to more than one operational mode. However, in a given cycle, there cannot be more than one activity manager that executes the same activity. This constraint is necessary to avoid a situation where two activity managers (that might be running under two different thread of control) interfere with each other by attempting to control the same activity.

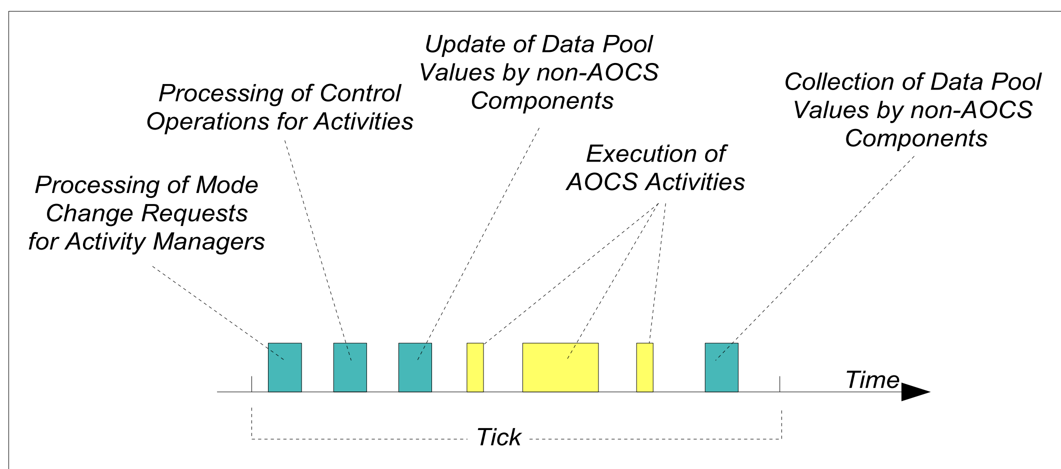

Fig. 9.9-2: Sample Internal Structure of a Tick

9.10 Activity Types

The activity is the key concept in the Control Framework. The activity concept is intended to play the same pivotal role in the definition of the Control Framework as the concepts of telecommand service and telemetry service play in the Packet Utilization Standard (PUS).

Copyright 2007 P&P Software GmbH All Rights Reserved

Adoption of the PUS allows the application designer to define his DH application in terms of the telemetry and telecommand services that it must be capable of handling. Similarly, the intention behind the introduction of the activity concept is to allow an AOCS application designer to specify his application exclusively in terms of the activities that it must support.

The PUS pre-defines a number of telemetry packet types and telecommand packet types that encapsulate the most common kinds of telecommand and telemetry services. Similarly, the Control Framework could be extended to define certain activity types encapsulating recurring kinds of AOCS functions. The definition of the activity types goes beyond the scope of the CORDET Project. By way of example, table 9.10-1 gives an overview of a few typical AOCS activity types.

Table 9.10-1: Typical AOCS Activity Types

Type	Name	Summary Description
1	Control Block	This type of activity implements a generic control algorithm.
2	Control Profile	This type of activity implements an attitude or orbit control profile, namely the computation of successive set-points for an attitude or orbit control algorithm.
3	Variable Monitoring	This type of activity implements a monitoring function where the actual behaviour of one or more variables is checked against a desired time profile that is encapsulated in the activity itself.

In the PUS, each service type is additionally divided into subtypes. A similar subdivision could be done for the AOCS activity types. Thus, for instance, the control profile activities – type 2 activities – could be subdivided into subtypes depending on whether the profile must be applied unconditionally or on whether checks must be done on the controller performance while the profile is being applied.

Similarly, subtypes for the variable monitoring activity – type 3 activity – could also be defined to represent various types of monitoring profiles: delta profile where a violation is reported if the monitored variable changes value by more than a certain threshold; out-of-limits profile where a violation is reported if the monitored variable is outside a certain interval; etc.

The DH Framework as it is defined in the CORDET Project only defines interfaces and components to represent generic telemetry and telecommand packets. The framework could be extended to specialize these interfaces and components to represent particular packet types and subtypes.

Similarly, the Control Framework as it is defined in the CORDET Project only defines interfaces and components to represent generic activities. Since there is no equivalent of the PUS for the AOCS, a first extension of this framework would be the creation of a taxonomy of activity types and subtypes. A second extension would be the specialization of the interfaces and components proposed in the CORDET Project to represent the activity types and subtypes thus defined.

9.11 Applicability to Other On-Board Subsystems

The on-board subsystem for which the Control Framework was designed is the Attitude and Orbit Control Subsystem (AOCS). However, the analyses that led to the definition of the Control Framework have shown that the characteristics of this framework are in fact not specific to the AOCS. Rather, they are specific to on-board control systems in general.

The activity concept, in particular, can be used to model any set of actions that must be executed on a cyclical basis and that have a fixed set of inputs, a fixed set of outputs, and a fixed algorithm for computing the outputs from the inputs. This type of cyclical actions are

Copyright 2007 P&P Software GmbH All Rights Reserved

found in virtually all on-board control systems and therefore the applicability of the activity concept is correspondingly wider than just the AOCS domain.

Similarly, the concept of operational mode is found in virtually all on-board applications and operational modes could, in most cases, be modelled as sets of activities. Hence, in this respect too, the concepts proposed by the Control Framework are also found in domains other than the AOCS. The same applies to the other two basic concepts of the Control Framework – the data pool and parameter database concept – which are again found in most on-board applications.

Thus, it can be concluded that the domain of applicability of the Control Framework is wider than originally anticipated and probably encompasses most on-board control applications.

9.12 Relationship to DH Framework

The DH and the Control frameworks cover two orthogonal aspects of a typical on-board application. In fact, it is expected that application designers may want to instantiate both frameworks within their application. In such a case, the application design would use the DH Framework to model and implement the telecommand processing and telemetry generation part of his application, and he would use the Control Framework to model and implement cyclical activities.

The interaction between the two frameworks would take place at application level. Note that the interaction between the two frameworks is not symmetric: the part of the application instantiated from the DH Framework might act upon the part of the application instantiated from the Control Framework but not vice-versa.

Typical forms of actions from the DH to the Control parts are telecommands that perform control operations on activities (such as enabling and disabling them, or holding and resuming them) and telemetry packets that acquire their data from the data pool or by reading the status of activities.

Activities instead are restricted to acting on the data pool and therefore cannot directly operate on the telemetry and telecommanding part of the application.

Copyright 2007 P&P Software GmbH All Rights Reserved

10 CONTROL FRAMEWORK – DOMAIN DICTIONARY

This section presents the domain dictionary for the Control Framework. The data dictionary entries are listed in logical order (as opposed to alphabetical order). Each data dictionary entry is presented in a table with the following format:

Term	<Domain Dictionary Term>
Definition	<Domain Dictionary Definition>

As specified in the CORDET Methodology, the domain dictionary entries are formulated in natural language. However, readers will readily note that there is a common pattern to the definition of the domain dictionary entries (a sort of informal "meta-model" of the domain dictionary entries). The main items that are used to define a domain dictionary entry are: its *attributes*, its *operations*, its *actions*, and its *checks*.

An attribute designates characteristics that are entirely defined by their value. The operations, actions and checks designate executable functionalities that are associated to the entity being defined. Operations are executed *upon* the entity being defined. Actions and checks are instead executed *by* the entity being defined as a result of changes in its internal state. Checks differ from actions in that they return a value.

The domain dictionary does *not* define the semantics of the attributes, operations, actions and checks associated to its entries. It merely defines their existence. Their semantics is implied by the properties within which the domain dictionary entries appear.

An example may clarify this point. The entry for "activity" in the Control Framework domain dictionary (see the next section) specifies that activities may be enabled and disabled. The properties specified in section 11.1 specify what happens to an activity when it is enabled and disabled. These properties thus give meaning to the enable and disable operations defined on activities.

10.1 Domain Dictionary Entries for Activity Concept

This section defines the domain dictionary entries related to the activity concept (see section 9.3).

Term	**Activity**
Definition	Encapsulation of a transaction-like, passive, data processing algorithm. An activity is characterized by a set of *attributes*, a set of *operations* that can be performed upon the activity, and a set of *actions* and of *checks* that the activity can perform upon itself or upon its environment. The attributes associated to an activity are: • The type and subtype of the activity • The identifier of the activity The operations that can be performed upon an activity are: • The activity can be initialized • The activity can be executed • The activity can be enabled and disabled • The activity can be put in use and out of use • The activity can be held and resumed The actions associated to an activity are:

Copyright 2007 P&P Software GmbH All Rights Reserved

	• The initialization action • The propagation action • The input read action • The output write action • The start action • The end action The checks associated to an activity are: • The initialization check • The propagation check

10.2 Domain Dictionary Entries for Mode Management Concept

This section defines the domain dictionary entries related to the operational mode and activity manager concepts (see sections 9.6 and 9.7).

Term	**Operational Mode**
Definition	A set of activities that are intended to be executed together. An operational mode is characterized by a set of *attributes,* by a set of *operations* that can be performed upon it, and by a set of *actions* and *checks* that the operational mode can perform upon itself or its environment. The attributes associated to an operational mode are: • The identifier of the operational mode • The maximum number of activity sets it can hold • The maximum number of activities in each activity set The operations that can be performed upon an activity are: • An operational mode can be queried for the list of activities it holds • An activity can be loaded in an activity set in the operational mode • An activity can be unloaded from an activity set in the operational mode The actions associated to an operational mode are: • The mode entry action • The mode exit action The checks associated to an activity manager are: • The mode entry check • The mode exit check

Term	**Activity Manager**
Definition	An activity manager encapsulates a set of operational modes together with the mechanism for selecting the current mode among them and for executing the activities in the current mode. An activity manager is characterized by a set of *attributes,* by a set of *operations* that can be performed upon it, and by a set of *actions* and *checks* that the activity manager can perform upon itself or its environment. The attributes associated to an activity manager are: • The identifier of the activity manager • The current mode of the activity manager The operations that can be performed upon an activity manager are:

Copyright 2007 P&P Software GmbH All Rights Reserved

	• The activity manager can be activated • The activity manager can be asked to perform a transition to a new current mode (mode transition request) The actions associated to an activity manager are: • The mode update action The checks associated to an activity manager are: • The update mode check • The mode transition check

10.3 Domain Dictionary Entries for Parameter Database Concept

This section defines the domain dictionary entries related to the parameter database concept (see section 10.3).

Term	**Parameter**
Definition	An encapsulation of a variable of primitive type to which activities have read-only access. A parameter is characterized by the following attributes: • Its current value • Its default value • Its identifier

Term	**Parameter Database**
Definition	An encapsulation of a set of parameters. A parameter database is characterized by a set of *attributes* and by a set of *operations* that can be performed upon it. The attributes associated to a parameter database are: • The parameters associated to it The operations that can be performed upon a parameter database are: • A parameter database can be reset • An individual parameter in the database can be reset • The current value of a parameter in the database can be read

Note that there are no operations to write or update the value of parameters in a parameter database. This does not mean that such an update is not possible. It simply means that such an operation is not offered by the Control Framework. From the point of view of the Control Framework, in other words, the parameter database is a read-only structure.

10.4 Domain Dictionary Entries for Data Pool Concept

This section defines the domain dictionary entries related to the data pool concept (see section 10.4).

Term	**Data Item**
Definition	An encapsulation of a variable of primitive type to which activities have read-write access. A parameter is characterized by a set of *attributes* and a set of *operations* that can be performed upon it.

Copyright 2007 P&P Software GmbH All Rights Reserved

	The attributes associated to a parameter database are: • Its current value • Its identifier • Its default value The operations that can be performed upon a data item are: • The current value of the data item can be read or written • The date item can be reset

Term	**Data Pool**
Definition	An encapsulation of a set of data items. A data pool is characterized by a set of *attributes* and by a set of *operations* that can be performed upon it. The attributes associated to a data pool are: • The data items associated to it The operations that can be performed upon a data pool are: • A data pool can be reset • An individual data item in the data pool can be reset • An activity can create a link to a data item in the data pool

Note that the no read or write operation is defined on the data pool. There is instead a link operation. This allows the clients of the data pool to set up an access channel to a particular data item and it is through this channel that the clients can read or write the data items in the data pool. The exact nature of the access channel is left open and is regarded as an implementation issue. In some cases, clients will have to make copies of data values whereas in other cases the link operation will give them access to a pointer through which they can directly use the data items.

Copyright 2007 P&P Software GmbH All Rights Reserved

11 CONTROL FRAMEWORK – SHARED PROPERTIES

This section defines the shared properties of the Control Framework. The shared properties are stated in tables with the following format:

<Property Identifier>	<Statement of the Property>

The definition of the properties is made in the following subsections. Each subsection presents the properties related to one or a small set of entries in the domain dictionary of the Control Framework.

11.1 Shared Properties for Activity Concept

This section defines the shared properties associated to the activity entry in the domain dictionary (see section 10.1).

11.1.1 Activity Initialization Properties

Activities should be initialized during the initialization phase of their host application. The initialization process would normally consist of setting the values of the attributes of the activity.

The initialization check encapsulates any checks that the activity may need to perform to verify that all the data required to initialize the activity have been provided and have legal values.

The initialization action can encapsulate any action that must be performed by the activity at the end of the initialization process (such as allocation of memory for internal data structures, initialization of internal data structures, etc).

P11.1.1-1	*The initialization operation on an activity is only successful if the initialization check returns TRUE. If this is not the case, then the initialization operation on the activity has no effect.*
P11.1.1-2	*If the initialization operation on an activity is successful, then the activity performs its initialization action.*
P11.1.1-3	*Unless an activity has been successfully initialized, all other operations performed upon it have no effect.*

11.1.2 Activity Execution

Activity execution consists in the reading of the inputs, one propagation cycle to compute the outputs, and the writing of the outputs. The number of propagation cycles is the period of the activity.

Normally, the execution of an activity should result in its internal state and inputs being propagated. However, propagation actually takes place only if the activity is neither disabled nor held and if its propagation check returns TRUE. The propagation check, in other words, can be used to implement a conditional propagation mechanism.

Activities can only be executed if they have first been associated to an activity manager. This is done by loading the activity into an activity set of an operational mode that is associated to that activity manager.

The execution of an activity should be transaction-like. This means that while the activity is performing its read-propagate-write cycle, any other operation that is performed upon it has no effect.

Copyright 2007 P&P Software GmbH All Rights Reserved

This in particular means that oeprations to enable/disable, hold/resume, and load or unload an activity from its operational mode are only processed if they are performed outside a propagation cycle.

P11.1.2-1	*When an activity that is neither disabled nor held is executed, it performs its propagation check and, if this returns TRUE, then the activity performs its input read operation, its propagation action(, and its output write operation.*
P11.1.2-2	*When an activity is performing read-propagate-write cycle, any other operation performed upon it has no effect.*

11.1.3 Holding and Resuming Activities

The hold-resume mechanism is intended to be used to temporarily neutralize the effect of activity execution.

P11.1.3-1	*An execute operation performed upon an activity that is held has no effect.*
P11.1.3-2	*If a resume operation is performed upon an activity that is held, then the activity is no longer held (ie it is resumed).*
P11.1.3-3	*If a hold operation is performed upon an activity that is not held, then the activity is held.*

11.1.4 Enabling and Disabling of Activities

The disabling mechanism is intended to be used to temporarily neutralize the effect of activity execution and to reset the activity. In this sense, it is similar to the hold-resume mechanism. It differs from it in that disabling an activity will also cause the internal state of the activity to be reset to its initial value and it will cause its start and end actions to be executed.

P11.1.4-1	*When an activity is disabled, it performs its end action.*
P11.1.4-2	*When an activity is enabled, it performs its start action.*
P11.1.4-3	*An execute operation performed on an activity that is disabled has no effect.*
P11.1.4-4	*If a disable operation is performed upon an activity that is enabled, then the activity becomes disabled.*
P11.1.4-5	*If an enabled operation is performed upon an activity that is disabled, then the activity becomes disabled.*

11.2 Shared Properties for Mode Management Concept

This section defines the shared properties associated to the operational mode and activity manager entries in the domain dictionary (see section 10.2).

11.2.1 Activity Manager Activation

When an activity manager is activated, it first checks whether it should update its current operational mode (mode update check), it then performs the change in current operational mode (if required), and it finally executes all the activities in the current operational mode.

The activities in an operational mode are organized in a set of activity sets. The activity manager ensures that the activity sets are processed in a fixed sequence. In other words, the activity manager ensures that activities in activity set i are executed before activities in activity set (i+1). There is instead no guarantee about the order in which the activities in an activity set are executed.

Copyright 2007 P&P Software GmbH All Rights Reserved

Note that the mode update check both determines whether a change in current operational mode is required and what the new operational mode should be.

The logic for performing the change in current operational mode is the same as when a mode change request is received from the outside (see next section).

P11.2.1-1	*When an activity manager is activated, it first performs the mode update check. It then performs a change in current operational mode (if this is required by the outcome of the mode update check), and it finally executes all the activities in the current operational mode.*
P11.2.1-2	*If an activity manager has n activity sets with n greater than 1, then it executes activities in activity set i before activities in activity set (i+1).*

11.2.2 Current Operational Mode Changes

The current operational mode of an activity manager can change either as a result of a request from some outside entity (mode transition request) or as a result of an autonomous decision of the activity manager itself (mode update check).

In both cases, the change in current mode can only take place under certain conditions. More specifically, there are three types of checks that are performed by an operational mode:

- The *mode exit check* verifies that the current operational mode can be exited.
- The *mode entry check* verifies that the target operational mode can be entered.
- The *mode transition check* verifies that the transition from the current to the target operational mode can be performed (ie it verifies the legality of the transition across two modes)

The change in operational mode is only performed if all three checks are positive. When an activity manager changes its operational mode, then it executes its mode update action.

When an operational mode is phased out from being the current mode, then the activity manager executes the end actions associated to all its activities. Similarly, when an operational mode is phased in as new current mode, the activity manager executes all its start actions.

Note that the order in which the start and end actions of the activities in an operational mode are executed is undefined because the operational mode is a *set* of activities (as opposed to an *ordered list*). The start and end actions of activities are factors of variation of the Control Framework and their content is defined by the application designer during the framework instantiation process. However, in order to preserve determinism of behaviour, they are restricted to modify only the internal state of the activity itself (see section 12.2).

Changes to the global state of the framework (namely to the content of the data pool) that should take place when an operational mode changes, must be encapsulated in the entry and exit actions of the operational mode themselves.

P11.2.2-1	*A change in current operational mode from a source to a destination operational mode is only performed if the mode exit check of the source mode, the mode entry check of the destination mode, and the mode transition check on the [source,destination] pair are successful.*
P11.2.2-2	*When an operational mode is phased out from being the current mode of an activity manager, the activity manager executes the end action of all its activities.*
P11.2.2-3	*When an operational mode is phased in as new current mode of an activity manager, the activity manager executes the start action of all its activities.*
P11.2.2-4	*When an activity manager changes its current mode, then it executes its mode update action.*

Copyright 2007 P&P Software GmbH All Rights Reserved

P11.2.2-5	*When an operational mode becomes the new current mode, it executes its mode entry action.*
P11.2.2-6	*When an operational mode is phased out from being the current mode, it executes its mode exit action.*

11.3 Shared Properties for Parameter Database Concept

This section defines the shared properties associated to the parameter and parameter database entries in the domain dictionary (see section 10.3).

P11.3-1	*If a parameter database is reset, then the current value of each of its parameters is set equal to its default value.*
P11.3-2	*If a parameter in a parameter database is reset, then its current value is set equal to its default value.*
P11.3-3	*If a parameter in a parameter database is read, then its current value is returned.*

11.4 Shared Properties for Data Pool Concept

This section defines the shared properties associated to the data pool entries in the domain dictionary (see section 10.4).

P11.4-1	*If a data pool is reset, then the current value of each of its data items is set equal to its default value.*
P11.4-2	*If a data item in a data pool is reset, then its current value is set equal to its default value.*
P11.4-3	*If an activity has created a link to a data item in a data pool, then the activity has read and write access to the data item attributes (its current value, its back-up value, and its validity status).*

Copyright 2007 P&P Software GmbH All Rights Reserved

12 CONTROL FRAMEWORK – FACTORS OF VARIATION

This section defines the factors of variation for the Control Framework. The factors of variation are defined in tables with the following format:

<Identifier>	<Name of the Factor of Variation>
Description	<Description of the Factor of Variation>
Default	<Default Value of the Factor of Variation>
Range	<Legal Range of the Factor of Variation>

The description of the factor of variation includes a reference to the property to which the factor of variation applies.

The CORDET Methodology prescribes that factors of variations be defined also in terms of their mutual interactions and in particular in terms of the constraints on their legal combinations. These interactions are captured in the feature model for the Control Framework that is documented in the next section.

12.1 Attributes as Factors of Variation

All the attributes defined for the entries in the domain dictionary represent factors of variation since the framework only defines the *existence* of the attributes and the application designer is free to set their values.

Factors of variations linked to attributes are regarded as trivial and implicitly defined by the domain dictionary and are therefore not further described in this section.

12.2 Factors of Variation for Activity Concept

This section defines the factors of variation associated to the activity entry in the domain dictionary (see section 10.1).

FV12.2-1	**Activity Initialization Check**
Description	The implementation of the initialization check used in property P11.1.1-1 is application-specific.
Default	The default implementation of the initialization check returns TRUE if all the attributes of the activity have legal values.
Range	This check must return either TRUE (activity has been successfully initialized) or FALSE (activity has not been successfully initialized).

FV12.2-2	**Activity Initialization Action**
Description	The implementation of the initialization action used in property P11.1.1-2 is application-specific.
Default	The default implementation of the initialization action does nothing.
Range	Unrestricted.

FV12.2-3	**Activity Propagation Check**

Description	The implementation of the propagation check used in property P11.1.2-1 is application-specific.
Default	The default implementation of the propagation check returns TRUE.
Range	This check must return either TRUE (activity can be propagated) or FALSE (activity cannot be propagated).

FV12.2-4	**Activity Propagation Action**
Description	The implementation of the propagation action used in property P11.1.2-1 is application-specific.
Default	There is no default implementation for the propagation action.
Range	Unrestricted.

FV12.2-5	**Activity Input Read Action**
Description	The implementation of the input read action used in property P11.1.2-1 is application-specific.
Default	There is no default implementation for the input read action.
Range	Unrestricted.

FV12.2-6	**Activity Output Write Action**
Description	The implementation of the output write action used in property P11.1.2-1 is application-specific.
Default	There is no default implementation for the output write action.
Range	Unrestricted.

FV12.2-7	**Activity Start Action**
Description	The implementation of the start action used in property P11.1.4-2 is application-specific.
Default	The default implementation of the start action does nothing.
Range	This action can only affect the internal state of the activity (ie it cannot change the value of an entry in a data pool). See discussion in section 11.2.2.

FV12.2-8	**Activity End Action**
Description	The implementation of the end action used in property P11.1.4-1 is application-specific.
Default	The default implementation of the end action does nothing.
Range	This action can only affect the internal state of the activity (ie it cannot change the value of an entry in a data pool). See discussion in section 11.2.2.

Copyright 2007 P&P Software GmbH All Rights Reserved

12.3 Factors of Variation for Mode Management Concept

This section defines the factors of variation associated to the operational mode and activity manager entries in the domain dictionary (see section 10.2).

FV12.3-1	**Mode Update Check**
Description	The implementation of the mode update check used in property P11.2.1-1 is application-specific.
Default	The default implementation of the mode update check returns: 'no mode update required'.
Range	This check must return either TRUE (mode update is required) or FALSE (no mode update is required).

FV12.3-2	**Mode Exit Check**
Description	The implementation of the mode exit check used in property P11.2.2-1 is application-specific.
Default	The default implementation of the mode exit check returns: 'mode exit allowed'.
Range	This check must return either TRUE (mode exit is allowed) or FALSE (mode exit is not allowed).

FV12.3-3	**Mode Entry Check**
Description	The implementation of the mode entry check used in property P11.2.2-1 is application-specific.
Default	The default implementation of the mode entry check returns: 'mode entry allowed'.
Range	This check must return either TRUE (mode entry is allowed) or FALSE (mode entry is not allowed).

FV12.3-4	**Mode Transition Check**
Description	The implementation of the mode transition check used in property P11.2.2-1 is application-specific.
Default	The default implementation of the mode transition check returns: 'mode transition allowed'.
Range	This check must return either TRUE (mode transition is allowed) or FALSE (mode transition is not allowed).

FV12.3-5	**Mode Update Action**
Description	The implementation of the mode update action used in property P11.2.2-4 is application-specific.
Default	The default implementation of the mode update action returns without doing anything.
Range	Unrestricted.

Copyright 2007 P&P Software GmbH All Rights Reserved

FV12.3-2	**Mode Exit Action**
Description	The implementation of the mode exit action used in property P11.2.2-6 is application-specific.
Default	The default implementation of this action returns without doing anything.
Range	Unrestricted.

FV12.3-3	**Mode Entry Action**
Description	The implementation of the mode entry action used in property P11.2.2-5 is application-specific.
Default	The default implementation of this action returns without doing anything.
Range	Unrestricted.

12.4 Factors of Variation for Data Pool Concept

This section defines the factors of variation associated to the data item and data pool entries in the domain dictionary (see section 10.4).

FV12.4-1	**Data Link Operation**
Description	The implementation of the data link operation used in property P11.4-3 is application-specific.
Default	There is no default implementation for this operation.
Range	There are two kinds of links that can be established by an activity with a data item in a data pool: a *copy link* whereby the activity accesses the data item values by copy, or a pointer link whereby the activity accesses the data item values through pointers

Copyright 2007 P&P Software GmbH All Rights Reserved

13 CONTROL FRAMEWORK – FEATURE MODEL

This section describes the feature model for the Control Framework..

The feature model of the Control Framework is built with the XFeature tool[9]. This section only gives an overview of the framework feature model. The feature model itself is available in electronic form and can be downloaded from the "Domain Analysis" page of the CORDET Web Site[10].

XFeature is a meta-modelling tool. In order to be used to construct a feature model, it must first be configured with a feature meta-model and a display model (that defines how the feature model is rendered graphically). In this project, XFeature was used in the so-called "FD Configuration". This is one of the default configurations of the XFeature tools. It was defined in the ASSERT project and is documented in RD-33. For the convenience of the reader, section 8.1 gives a brief overview of this configuration.

13.1 Top-Level Features

Figure 13.1-1 shows the top-level features of the Control Framework Feature Model.

The root element is called `AocsApplication` since it represents an AOCS Application instantiated from the framework. The feature mode in the figure simply indicates that an AOCS Application consists of:

- one or more data pools
- one or more parameter databases, and
- one or more activity managers.

The only variability captured by this high-level feature model resides in the cardinality of the various features.

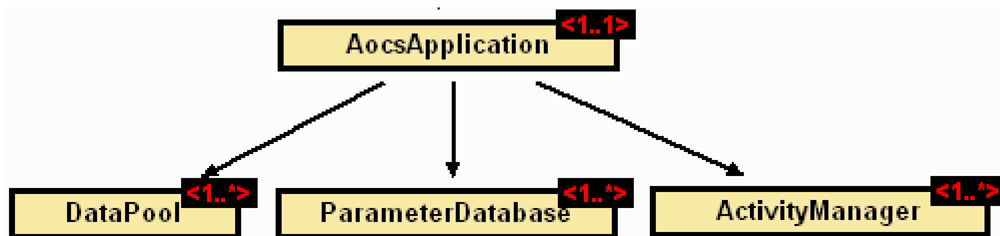

Fig. 13.1-1: Top-Level Features of Control Framework Feature Model

13.2 Data Pool Features

Figure 13.2-1 expands the `DataPool` feature in the previous figure to show the features for the Data Pool concept.

The figure indicates that a data pool consists of one or more data items and a data link mechanism. The data items are not further characterized by the feature model (there is no variability associated to them other than their cardinality). The data link mechanism can be of two kinds: it is either a "copy link" or a "pointer link". This is the same variability defined in factor of variation FV12.4-1. Note that this variability is associated to the individual data item, not to the data pool as a whole.

[9] The tool can be downloaded as free and open software from its home page at this address: http://www.pnp-software.com/XFeature/

[10] The CORDET Web Site is at: http://www.pnp-software.com/cordet

Copyright 2007 P&P Software GmbH All Rights Reserved

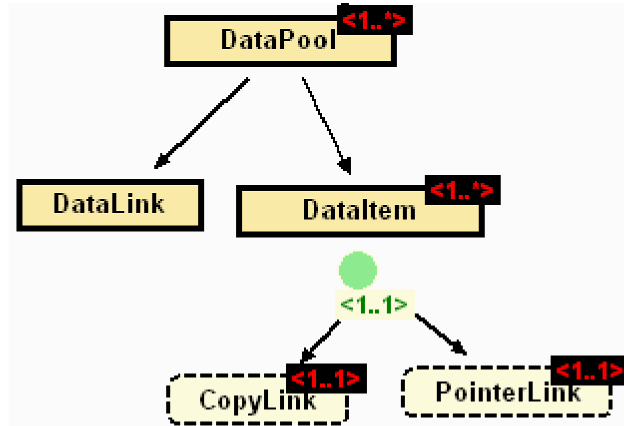

Fig. 13.2-1: Data Pool Features in Control Framework Feature Model

13.3 Parameter Database Features

Figure 13.3-1 expands the `ParameterDatabase` feature in figure 13.1-1 to show the features for the Parameter Database concept. The feature model in the figure indicates that the only variability associated to a parameter database is the number of parameters in the database itself.

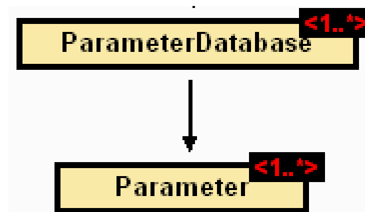

Fig. 13.3-1: Parameter Database Features in Control Framework Feature Model

13.4 Activity Manager Features

Figure 13.4-1 expands the `ParameterDatabase` feature in figure 13.1-1 to show the features for the activity manager concept.

An activity manager is characterized by: three optional features (the `ModeTransitionCheck`, the `UpdateModeCheck`, and the `UpdateModeAction`) which correspond to the factors of variation FV12.3-1, FV12.3-4, and FV12.3-5; and one or more operational modes.

An operational mode is characterized by: four optional features (the `EntryAction`, the `EntryCheck`, the `ExitAction`, and the `ExitCheck`) which correspond to the factors of variation FV12.3-2, FV12.3-3, FV12.3-5, and FV12.3-6; and one or more activity sets.

Copyright 2007 P&P Software GmbH All Rights Reserved

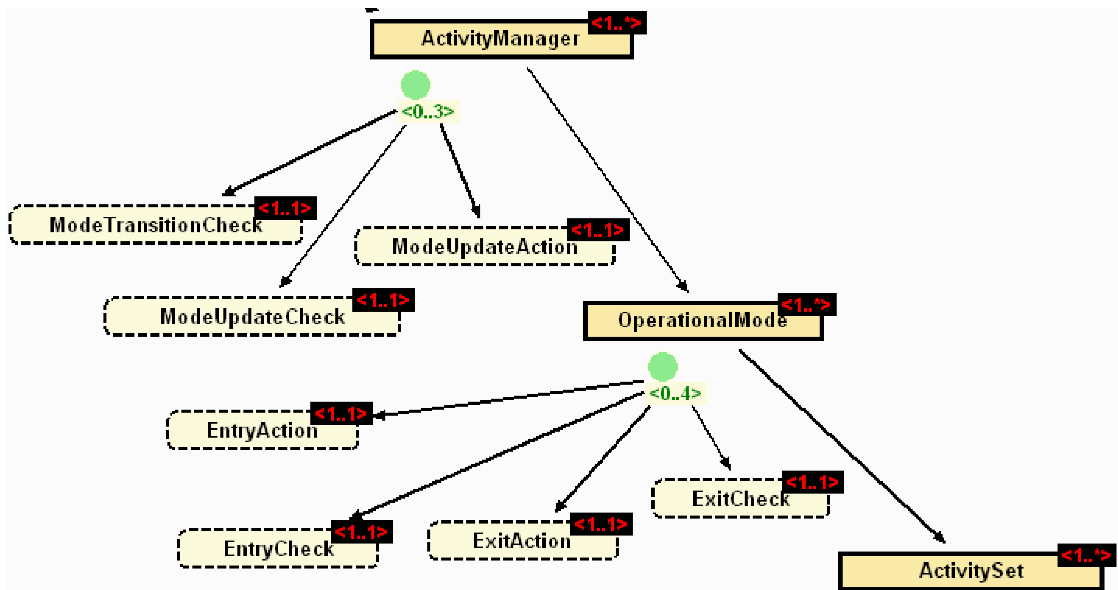

Fig. 13.4-1: Activity Manager Features in Control Framework Feature Model

13.5 Activity Features

Figure 13.5-1 expands the `ActivitySet` feature in the previous figure to show the features for the activity concept.

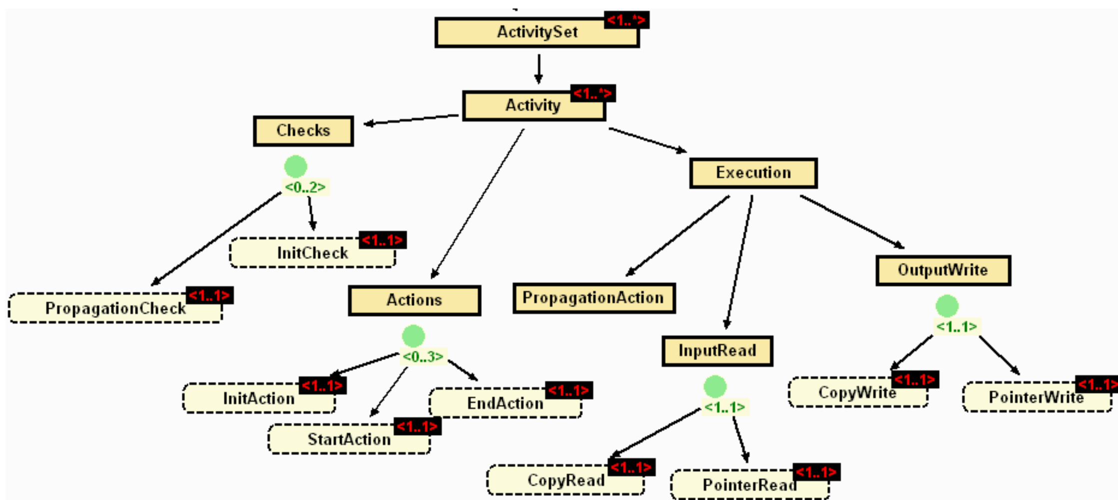

Fig. 13.5-1: Activity Set Features in Control Framework Feature Model

According to the figure, an activity set consists of one or more activities.

An activity consists of two optional checks (the initialization and the propagation check), three optional actions (the initialization action, the start action, and the end action), and a mandatory `Execution` feature.

The `Execution` feature captures the variability associated to an execution cycle of the activity. This consists of a mandatory propagation action, and mandatory `InputRead` and `OutputWrite` features.

The `InputRead` and `OutputWrite` features capture the variability related to the mechanisms through which an activity is linked to its inputs and outputs in the data pool. The data link mechanism can be of two kinds: it is either a “copy link” or a “pointer link”. This is

Copyright 2007 P&P Software GmbH All Rights Reserved

the same variability defined in factor of variation FV12.4-1 and in the data pool feature of section 13.2. There is an obvious constraint when selecting the sub-feature of the `DataPool` features and the sub-features of the `InputRead` and `OutputWrite` features. This constraint is not visible in the figures but it is expressed in the feature model as a “require global constraint”.

Copyright 2007 P&P Software GmbH All Rights Reserved