THE CORDET METHODOLOGY

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-MR-COR-0001

Table of Contents

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.
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.
CORBA	A widely used middleware infrastructure.
Design Pattern	A description of an abstract design solution for a common
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.
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.
Object Oriented Framework	A software framework that uses inheritance and object composition as its chief adaptation mechanisms.
OBS	The On-Board Software.
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.

2 REFERENCES

RD1	AOCS Framework Project Web Site, control.ee.ethz.ch/~ceg/AocsFrameworkProject
RD2	RT Java AOCS 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
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	Panunzio M, Vardanega T, *An Approach to the Timing Analysis of Hierarchical Systems*, Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, August 2007
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, Domain Engineering Methodologies Survey*,* CORDET Deliverable GMV-CORDET-WP202-RP-01
RD35	ASSERT Project – ETH Deliverable D4.2.4-4.3 : *The ETH Demonstrator Framework – Contribution to V3 Demonstrator, Part 3 (Design Contracts)*
RD36	Panunzio M, Vardanega T, *A Metamodel-Driven Process Featuring Advanced Model-Based Timing Analysis*, Proceedings of the 12th Conference on Reliable Software Technologies – Ada-Europe'07, Geneva, Switzerland, Springer Verlag. June 2007.
RD37	Bordin M, Vardanega T, *Correctness by Construction for High-Integrity Real-Time Systems: A Metamodel-Driven Approach*, Proceedings of the 12th Conference on Reliable Software Technologies – Ada-Europe'07, Geneva, Switzerland, Springer Verlag. June 2007

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 software 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.

3.2 Objectives of This Document

This document fulfills the first of the four specific objectives identified in the previous section. The document defines the *CORDET Methodology*, namely the methodology to be followed to develop the CORDET Generic Architecture. Additionally, this document also defines the *CORDET Tool Chain,* namely the set of support tools to be used in the CORDET Project to implement the CORDET Methodology.

The CORDET Methodology implements a subset of the domain engineering part of the ISO/IEC 12207 process. Broadly speaking, the CORDET Methodology can be divided into two parts.

The first part of the CORDET Methodology defines the activities to be performed during the development of the Generic Architecture in terms of their sub-activities, their objectives, and their outputs. It defines, in other words, *what* should be done to build the Generic Architecture.

The second part of the CORDET Methodology defines a set of rules that guide and constrain the development of the CORDET Generic Architecture and that help the designer to achieve the objectives laid down in the first part of the methodology. This part of the CORDET Methodology, in other words, defines *how* the Generic Architecture should be built.

The second part of the CORDET Methodology depends on a particular choice of software technologies or on a particular way of using a certain software technology. The first part of the methodology, by contrast, is intended to be more abstract and to capture the essential - non-technology related - aspects of the development of a generic architecture.

A software methodology is only useful if tools exist to support its practical application. This document accordingly identifies the support tools proposed for the CORDET Methodology. The support tools identified in this document are those proposed for use in the CORDET Project. Obviously, whereas the methodology is intended to be general, the choice of tools is contingent and may evolve over time.

The specific aim of the CORDET Methodology is to support the creation of the CORDET Generic Architecture. More generally, however, the CORDET methodology can be used to support the creation of generic architectures for systems that satisfy the basic assumptions defined in section 5. The most important assumption is that it must be possible to decompose the software of the target system into two layers of which one – the middleware layer – implements the system's non-functional requirements whereas the other – the application layer – implements its functional requirements.

3.3 Methodology Survey

The methodology proposed in this document is eclectic and does not conform to any pre-existent domain engineering methodology. It does, however, have two main sources of inspirations in the FODA Methodology for the domain analysis phase, and in the FW and RCM Methodologies from the ASSERT Project for the domain design phase.

The approach proposed for the domain analysis phase is similar to the classical FODA approach in two major respects. Firstly, like FODA, the CORDET Methodology sees the primary objective of the domain analysis phase in the identification of commonalities and variabilities within a family of related applications. Secondly, like FODA, the CORDET Methodology proposes feature models as one of the means to describe the domain model of a product family. It should, however, be stressed that, in the CORDET Methodology, feature models play a less prominent role than in FODA. In the latter approach, the feature model is the primary formalism used to define the domain model. In the CORDET Methodology, instead, feature models are used to describe variability within the target family whereas commonality is captured by so-called shared properties.

The approach proposed for the domain design phase draws heavily on the work done in the ASSERT Project. In particular, the CORDET Methodology takes over unchanged the FW Methodology for the design of the functional part of the generic architecture and it proposes a modified version of the RCM Methodology for the non-functional part of the generic architecture.

No attempt is made in this document to present a general survey of domain engineering standards, methods and tools since this can be found in a companion CORDET Deliverable written by GMV (see reference RD-34).

The GMV deliverable, in addition to presenting a methodological survey, also makes methodological recommendations which are based on the work done by GMV in other related projects (in particular in the DOMENG Project). The GMV methodological recommendations are made from a more general standpoint than those presented in this document. It is noteworthy that they are broadly in line with the CORDET Methodology as it is defined in this document. The main similarities are the reliance on FODA as an umbrella methodology for the domain analysis phase, and the selection of UML2-based approaches for the domain design phase. The main difference is perhaps the emphasis GMV places on the use of sysML as a modelling language for the domain analysis phase whereas the CORDET Methodology advocates the use of natural language as a means to express the domain model. This difference is due to the narrower scope of the CORDET Project and to its more limited resources (see also discussion in section 7.1).

3.4 Structure of This Document

The next section discusses the basic concepts and terminology that are used in this document. Its intention is to put the reader in a position to understand the material presented in the remainder of the document and to avoid misunderstandings due to different terminological conventions found in the domain engineering literature.

The methodology defined in this document has a rather narrow focus. It is specifically aimed at the development of the CORDET Generic Architecture. It could also be used for the development of generic architectures for systems with similar characteristics. In order to clarify the range of systems to which the proposed methodology would be applicable, section 5 discusses and makes explicit the assumptions that lie behind the CORDET Methodology.

Sections 6 and 7 present, respectively, the first and second part of the CORDET Methodology. The methodology is defined as a set of requirements. Where appropriate, discussions that clarify and justify the requirements are interleaved with the requirements themselves.

Section 8 describes the tools that are proposed for use in the CORDET study to support the application of the CORDET Methodology.

Finally, section 9 presents a summary in table format of the methodological and tool selections made in this document.

3.5 Status of This Document

This document is complete and incorporates all comments agreed at the review held jointly with the first CORDET Progress Meeting on July 6th 2007.

4 BASIC CONCEPTS AND TERMINOLOGY

This section discusses the basic concepts and terminology that are used in this document. The intention of this section is to put the reader in a position to understand the material presented in the remainder of the document and to avoid misunderstandings due to different terminological conventions found in the product engineering literature.

The CORDET study relies to a significant extent on the results of the ASSERT Project. Several ASSERT deliverables are used by or are applicable to the CORDET study. In order to facilitate the use of these inputs from the ASSERT project, wherever possible, the same terminology is used as in ASSERT.

Note that a glossary with a brief definition of the technical terms and acronyms used in this study is available in section 1 at the beginning of this document.

4.1 Application and Domain Engineering

Traditionally, software and system engineering have been application-oriented: processes, methods, and tools were geared towards the development of a specific application in response to requirements formulated by an end user. Reuse was always an option but it remained incidental and tended to be limited to individual and disjoint components.

The introduction, in the mid-nineties, of the *domain engineering paradigm* shifted the focus away from single applications and towards clusters of related applications. The key idea is to direct the efforts of developers towards the development of reusable assets from which applications can be built.

The key concept in domain engineering is that of *product family* [RD-4, 5, 15]. Indeed, the two terms – domain engineering and product family – are often used interchangeably.

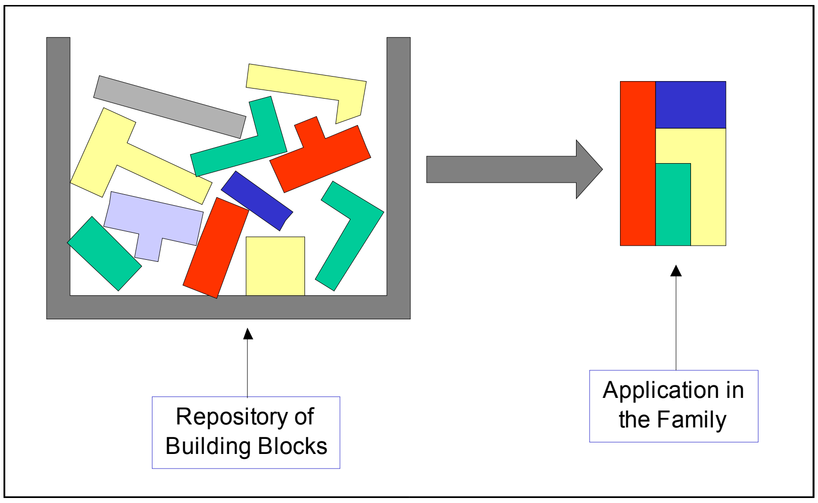

Fig. 4.1-1: Software Product Families

A *product family* is a set of applications that can be constructed from a pool of shared assets[1]. The shared assets can be seen as generic *building blocks* from which applications in the family can be built. Usually, a product family is aimed at facilitating the instantiation of applications within a narrow domain. Figure 4.1-1 illustrates the concept of product family. On the left hand-side, the building blocks offered by the product family are shown. These building blocks

[1]In this section, the term 'application' is used in a generic sense to designate the 'product' that is built with the help of the product family.

are used during the *family instantiation process* to construct a particular application within the family domain.

Product families are characterized by two distinct development processes (see 4.1-2). In the *family creation process*, the family's reusable assets are designed and developed. In the *family instantiation process*, the reusable assets offered by the family are used to construct a specific application within the family domain.

The family creation process is in turn divided into three phases. In the *domain analysis phase*, the set of applications that must be covered by the family is identified and characterized. The output of this phase is a domain model. In the *domain design phase*, the reusable assets that are to support the instantiation of applications within the family are designed. The output of this phase is one or more models of the family assets. The models express various aspects of the domain design (e.g. there may be functional models, timing models, etc). In the *domain implementation phase*, the family assets are implemented as concrete building blocks that can be used towards the construction of family applications.

Often, the implementation of the family assets is done automatically by processing the models defined in the domain design phase.

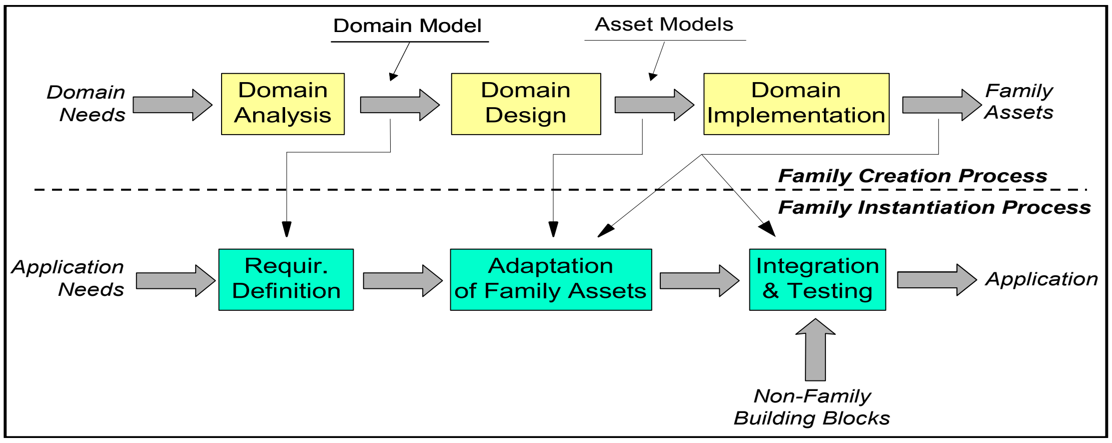

Fig. 4.1-2: Development Process for Software Product Families

Three matching phases can be identified in the family instantiation process (bottom half of figure 4.1-2). In the *requirement definition phase*, the family domain model is used to verify whether the target application falls within the family domain. This decides whether the family assets can be used to help build the application. If this is the case, a sizeable proportion of the application requirements can be expressed in terms of the domain model, for instance by identifying the features in the family domain that are needed by the target application. In the *adaptation phase*, the software assets required for the target application are selected from among those offered by the family. They are then adapted to match its needs. Depending on how the family assets are organized and implemented, adaptation can be done either at the level of the asset models, or at the level of the implementation. Finally, in the *integration and testing phase*, the target application is constructed by assembling the adapted building blocks offered by the framework. Usually, some integration with building blocks that are external to the family is also required in this phase.

The effectiveness of the product family approach derives from the fact that the level of design abstraction is raised from that of individual applications to that of domains of related applications. This allows investment in the design and development of software assets to be reused across applications. Current research attempts to further extend the effectiveness of product families by automating their instantiation process. The objective is to arrive at a generative environment of the kind shown in figure 4.1-3. The environment automatically translates a specification of an application in the family domain into a configuration of the family assets that implements it.

An environment of the type shown in 4.1-3 would, in a sense, represent a synthesis of the model-driven and reuse-driven approaches because it would allow the target application to be constructed automatically from its specification while at the same time taking advantage of the existence of predefined building blocks that implement part of the application functionality. It is for this reason that the product family approach – although a *reuse-based approach* – should not be seen as an alternative to, but as rather complementary to, the *model-driven approach*.

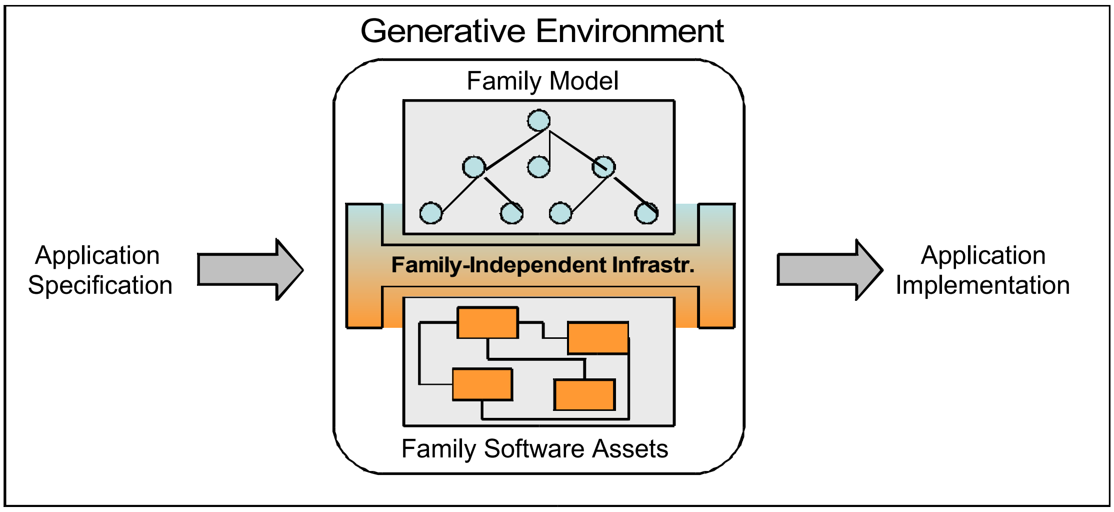
Generative Environment
Family Model
Application Specification
Family-Independent Infrastr.
Application Implementation
Family Software Assets

Fig. 4.1-3 : Automated Instantiation Environment for Software Product Families

4.2 System Families and Software Frameworks

The product family concept is very general and does not imply any assumption about the nature of the building blocks (are they components? Procedures? Code fragments? Design models?), or about their mutual relationships (can they be used independently of each other? Are they embedded within a higher-level structure?), or even about the type of products it is aimed at (software applications? Hardware-based systems? Hybrid systems?).

This document is concerned with two specific kinds of product family for which the terms "system family" and "software framework" are used.

More specifically, in this document, the term *system family* designates a product family where the 'product' that is to be built using the reusable assets provided by the family is the architectural infrastructure (the 'middleware') of a complex system. The term *software framework* instead designates a product family where the target 'products' are the software applications that run on top of the architectural infrastructure covered by the system family. Section 5.1 further particularizes these terms to the case of the CORDET study.

Both software frameworks and system families are *component-based* in the sense that their reusable building blocks consist of software components. The term *component* is used to designate a software entity with the following characteristics:

- It can be deployed as a stand-alone unit (hence it owns a clear specification of its *required interface*)
- It provides an implementation for one or more interfaces (hence it owns a clear specification of its *provided interface*)

- It interacts with other components exclusively through these (required and provided) interfaces

Both software frameworks and system families are *architecture-centric* in the sense that the components they provided are not intended to be used in isolation form each other but are instead intended to be embedded within an architecture that is optimized for their target domain and that is defined by the software framework or system family itself.

Thus, software frameworks and system families are particular ways of organizing the shared assets of a product family in the sense that they define the type of building blocks that can be provided by the product family and they define an architecture within which these building blocks are to be used.

Figure 4.2-1 illustrates the concept of an architecture-centric product family such as a software framework. The figure should be contrasted with the previous figure 4.1-1 to highlight the fact that the family reusable assets (the building blocks in the repository) are now organized as a set of interacting entities embedded within an architecture that is itself reusable. The framework approach, in other words, allows the reuse not only of the individual items but also of their mutual interconnections (the latter being an important and often neglected added value).

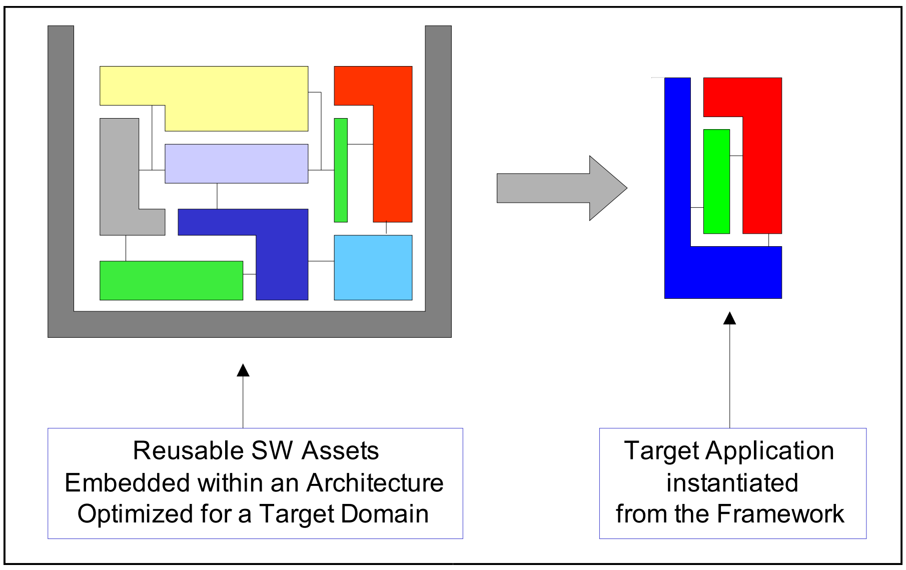

Fig. 4.2-1: The Software Framework Concept

As already mentioned, both system families and software frameworks are component-based but their architecture-centric character means that for both it is possible to recognize coarser grained building blocks.

Like all product families, software frameworks and system families are concerned with fostering reuse. One important question when considering them is what is the *unit of reuse*. At its most basic, the unit of reuse of a component-based family is a component. This follows from the fact that one of the distinctive features of a component is that it is deployable as a stand-alone unit. The components provided by a software framework or a system family, however, are embedded within an architecture (which is also defined by the framework or system family). Hence, users of the framework or system family are likely to focus their attention not on individual components but on groups of cooperating components that, taken together, support the implementation of some functionality that is important within the framework or system family domain. In fact, well-designed frameworks or system families encourage this higher granularity of reuse by being organized as a bundle of functionalities that users can choose to include in their applications. Inclusion of such a functionality implies that a whole set of cooperating components and interfaces is imported into the application. The right unit of reuse – and hence, according to this view, the right building block – is precisely such a set of components and interfaces.

An example may help to clarify the above concept. Consider a software framework for satellite on-board systems. One typical functionality that is often found in such systems is the storage of key data on a mass-memory device. Accordingly, the software framework would implement default mechanisms for managing such devices. This would probably be done through a set of cooperating components and interfaces. Application developers who need the mass-memory functionality for their target application and who decide to implement it with the help of the assets provided by the framework will import the entire set of components and interfaces. Use of components or interfaces individually is unlikely to make sense because the components and interfaces are specifically designed to work together within a certain architecture. The building block in this case is the set of components and interfaces that support the implementation of the mass-memory functionality.

The building blocks provided by a product family are, by definition, intended to be reusable. To reuse a software asset (a component, a fragment of code, a design model, etc) means to use it in different operational contexts. In practice, varying operational contexts will always impose differing requirements on the reusable assets. Hence, effective reuse requires that the reusable assets be *adaptable* to different requirements. In this sense, adaptability is the key to reusability and the availability of software adaptability techniques is the necessary pre-condition for software reusability.

The product family representation shown in the previous section should therefore be modified as in figure 4.2-2. The items that are selected from the repository are passed through an adaptation stage before being integrated to build the target application. In the adaptation stage, the characteristics of the reusable assets are modified to make them match the requirements of the target application.

Software reuse is perhaps the oldest means to reduce software costs and has often been tried in the past. Past attempts, however, had only mixed success primarily because they either ignored the adaptation phase shown in figure 4.2-2, or because the state-of-the-practice adaptation techniques available at the time were not sufficiently powerful to model the extent of variability in the target domain.

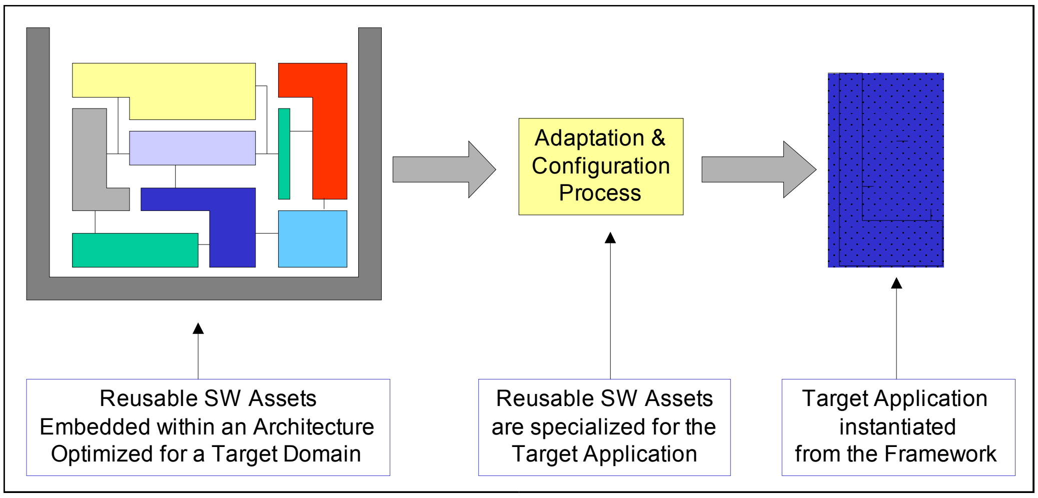
Adaptation & Configuration Process
Reusable SW Assets Embedded within an Architecture Optimized for a Target Domain
Reusable SW Assets are specialized for the Target Application
Target Application instantiated from the Framework

Fig. 4.2-2: Product Families and Adaptability

The quality of a product family largely depends on the ease with which the artifacts it offers – its building blocks – can be adapted to the requirements of its users. In this respect, software frameworks and system families differ from each other. Software frameworks tend to be object-oriented in the sense that they use *inheritance* and *object composition through abstract coupling* as their chief adaptation techniques. System families instead tend to be object-based and to use simpler adaptation mechanisms such as type genericity or use of adjustable configuration parameters.

The reason for this difference is that software frameworks are concerned with the application layer of a system and therefore normally privilege adaptation with respect to functional behaviour whereas system families are concerned with the middleware layer of a system and therefore normally privilege adaptation with respect to non-functional behaviour. The range of adaptation techniques currently available for functional behaviour is much wider than that available for non-functional behaviour.

In summary, system families and software frameworks are special kinds of product families that differ from each other in their target domains (middleware infrastructure vs software applications) and in their adaptation mechanisms (object-orientation vs paramter configuration) but resemble each other in being component-based and architecture-centric.

5 BASIC ASSUMPTIONS

The CORDET study is about applying a product family approach to satellite on-board systems. The product family concept has considerable intuitive appeal and nowadays few would dispute that it is desirable to identify commonalities and variations among related systems in order to build a pool of reusable assets from which individual systems can be constructed.

Translating these generalities into practice is, however, extraordinarily difficult. This is because the product family concept, by itself, is very vague. This concept does not say anything about what exactly the reusable assets should be (model-level entities? Source-level components? Fragments of code? Mini-applications? Etc), and neither does it say what exactly reusability means (use without changes? Use after configuration? Use after aspect-like adaptation? Etc), or what exactly the commonalities and variations are (requirements? Code? Features? Etc). In the case of complex domains, such as that of satellite on-board systems, it is also unclear whether one should aim at one single product family or whether multiple product families covering subsets of the target system should be built.

It is tempting to argue that these and other similar questions should be addressed and answered in the domain analysis phase of the product family development life-cycle. This, however, is naïve. The idea that, simply by looking at several systems in the target domain, one may, somehow and almost magically, find answers to the questions raised above is entirely misguided. This misconception probably accounts for the high failure rate of product family development projects.

The intention here is not to deny that the analysis of existing systems is an important step in the definition of a product family. The intention is rather to argue that such an analysis can only be effective if it is done in a structured manner and with a clear understanding of what kind of product family one is looking for. The domain analyst, in other words, should not simply *look* at existing systems. He should also know *what* he is looking for.

This section defines the assumptions that underlie the CORDET Methodology. These assumptions answer some of the general questions raised above. They define the conceptual framework within which the CORDET study will be carried out. It is important to stress that these assumptions are regarded as an *input* to the study whose adequacy is demonstrated by past work done outside the CORDET study itself [RD-24 to 31].

This section is divided into three subsections. The first subsection defines the high-level structure of the systems targeted in the study. The following two subsections operationalize the twin concepts of family commonality and variability.

5.1 Structure of CORDET Target Systems

The first basic assumption for the CORDET study concerns whether one single product family should be defined in the study covering the entire satellite on-board domain, or whether several interlocking families covering various subsets of this domain are needed.

In this study, the second approach is selected. There are three reasons that justify this choice. The first one is the sheer complexity of on-board systems that cover functions as diverse as attitude control, payload management, provision of OS-like services, etc. This diversity would make it hard to define one single all-encompassing product family.

The second reason is based on typical procurement policies adopted in the European space industry where it may happen that different parts of the same system are sub-contracted to different companies. This parcelization would become very difficult if the whole system were instantiated from the same product family.

The third reason is derived from the experience of AAS, P&P and UPD in their past projects and in particular in the ASSERT project. This experience demonstrates that it is indeed feasible to define multiple product families addressing different concerns of on-board systems.

A decision to develop multiple product families implies a need to define the high-level structure of the systems targeted by the product families. This is necessary in order to identify the boundaries of the various product families.

The high-level system structure assumed in this study is the same as used in the ETH Demonstrator Framework proposed in the ASSERT project and used for the ASSERT HRI demonstrator [RD28, 29, 30]. This high-level structure is sketched in 5.1-1 below. The figure shows that a system is decomposed into two layers of which the top one is in turn divided into two sub-layers.

The bottom layer is the *middleware*. This layer implements the *non-functional* requirements of the system. These non-functional requirements consist of: timing-related requirements, reliability-related requirements, and distribution-related requirements. The middleware layer thus defines the system-level architecture of an on-board system.

The top layer implements the functional part of an on-board system. It implements the functional requirements of the system. This layer can be split into vertical blocks corresponding to the various subsystems of an on-board system (AOCS, DH, thermal, payload, etc).

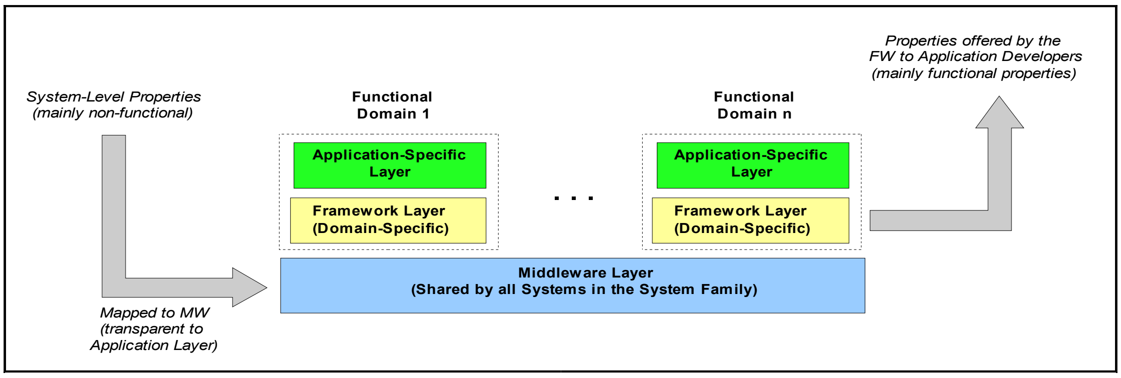
System-Level Properties (mainly non-functional)
Functional Domain 1
Functional Domain n
Properties offered by the FW to Application Developers (mainly functional properties)
Application-Specific Layer
Framework Layer (Domain-Specific)
Application-Specific Layer
Framework Layer (Domain-Specific)
. . .
Middleware Layer (Shared by all Systems in the System Family)
Mapped to MW (transparent to Application Layer)

Fig. 5.1-1: High Level Structure of Target Systems

Each subsystem identifies a functional domain. Within each functional domain, commonalities are present. It therefore makes sense to decompose the top layer into two sub-layers: one gathers the components implementing the functionalities that are invariant within each functional domain whereas the other implements functionalities that are entirely application-specific.

As suggested by the system structure shown in figure 5.1-1, two types of product families must be defined: *a system family* to cover the middleware layer and a set of *software frameworks* to cover the functional domains in the application layer.

As explained in section 4.2, a system family is a kind of product family. In this study, this term will be used to designate the product family that covers the middleware layer of figure 5.1-1. This term is chosen for consistency with the terminology used in ASSERT and because, in the approach proposed in this study, the middleware is responsible for providing the mechanisms to enforce system-level concerns (i.e. concerns which cannot be allocated to any specific functional subsystem). Such concerns are typically non-functional in character and are therefore naturally implemented in the middleware layer.

As explained again in section 4.2, a software framework is another kind of product family that is aimed at the development of software applications and whose reusable building blocks consist of software components. In the CORDET approach, software frameworks must be defined for each on-board functional subsystem. The software frameworks implement the functionalities that are common to all instances of each subsystem. Its components will be designed to be adapted and extended to match the requirements of a specific subsystem application.

Thus, the generic architecture to be designed in the CORDET study will consist of a system family and a set of software frameworks.

The approach outlined above is premised on two key assumptions. The first one is that one single system family (namely, one single middleware) can cover the entire, or at least a large segment of, the on-board system domain. This assumption had initially been rejected by the ASSERT project which started with the objective of defining several system families adapted to various types of missions. This attempt, however, failed and the evidence from the ASSERT project is that one single family can be sufficient to cover a wide range of very different systems.

Further evidence in support of the single-family approach also comes from AAS experience in developing generic architectures for on-board systems [RD26]. The results of these internal activities again indicate that one single architecture can be sufficiently flexible to cover a large number of commonly used satellite architectures.

The second assumption behind the approach proposed for this study is the feasibility of separating functional from non-functional aspects in the design of a system: the layered architecture of figure 5.1-1 assumes that non-functional requirements are entirely implemented at the level of the middleware whereas functional requirements are entirely implemented at application level. This split greatly simplifies the task of defining a generic architecture because it means that non-functional concerns can be covered in the system family whereas functional concerns can be covered in the software frameworks.

This assumption has been verified in the work done in ASSERT by ETH and UPD. Indeed, the feasibility of separating functional from non-functional aspects is arguably the single most important theoretical result to have emerged from the ASSERT project. The technique proposed to achieve this separation has been published in a number of ASSERT technical reports [RD27, 30] and in a peer-reviewed paper [RD31] and it has been demonstrated in the ASSERT demonstrator built with the HRI pilot project [RD28, 29, 30].

The assumptions proposed and justified in this section are summarized in table 5.1-1. These assumptions, together with the assumptions presented in the next two sub-sections, underlie the CORDET Methodology defined in the remainder of this document.

Table 5.1-1: Assumptions About the Structure of the CORDET Target Systems

ID	Assumption
A5.1-1	The systems to be covered by the CORDET Generic Architecture are assumed to have a layered structure as shown in figure 5.1-1.
A5.1-2	It is assumed that, in the systems targeted by the CORDET Generic Architecture, it is possible to separate the design and implementation of functional and non-functional aspects.
A5.1-3	The CORDET Generic Architecture shall consist of one single System Family to cover middleware-level non-functional aspects of satellite on-board systems and a set of Software Frameworks to cover their functional aspects.

5.2 The Commonalities of CORDET Product Families

The concept of *commonality* plays a central role in product family engineering. The search for commonalities is the key activity of the domain analysis phase and the implementation of the commonalities in the reusable assets is the key activity in the domain design phase. An essential pre-requisite for these activities is an understanding of what kind of commonalities are sought.

In line with the assumptions made in the previous subsection, the CORDET study assumes that systems are endowed with functional and non-functional *properties*. *Functional properties* are defined at the level of software applications and consist of logical relationships among the variables that define the state of the application. *Non-functional properties* are defined at system level and are implemented at the level of the middleware. Typical examples of non-functional properties are reliability and schedulability.

In the CORDET study, the search for commonalities among related systems is understood as a search for *shared properties*. The objective of the domain analysis phase then becomes the identification of the functional and non-functional properties that are shared by all systems in a certain domain. Similarly, the objective of the domain design phase becomes the definition of components and architectures that guarantee that shared properties are satisfied.

The previous section postulated a split between functional and non-functional issues with a generic architecture made up of a system family that covers non-functional concerns and a set of software frameworks covering functional subsystems.

The system family then becomes responsible for enforcing the non-functional properties of a set of systems. More specifically, the domain of the system family consists of a set of systems that share the same non-functional properties and the system family itself consists of a middleware that guarantees that all systems built on top of it will be endowed with those non-functional properties.

Similarly, the software frameworks become responsible for enforcing the functional properties of a set of applications. More specifically, the domain of a software framework consists of a set of software applications that share the same functional properties and the software framework itself consists of a set of components that guarantee that all applications built using those components will be endowed with those functional properties.

The background to and justification for the property-based approach proposed for this study comes from the ASSERT project. ASSERT has taken a property-based approach from the beginning and this has proved to be a successful choice. For this reason, this same approach is retained in this study. More details can be found in the ASSERT deliverables [RD-27 to 30].

The assumptions proposed and justified in this section are summarized in table 5.2-1. These assumptions, together with the assumptions presented in the previous and the next sub-section, underlie the CORDET Methodology defined in the remainder of this document.

Table 5.2-1: Assumptions About the Commonalities of CORDET Product Families

ID	Assumption
A5.2-1	The systems targeted by the CORDET Generic Architecture shall be assumed to be endowed with functional and non-functional properties.
A5.2-2	The search for commonalities among related systems shall be operationalized as a search for shared functional and non-functional properties.

5.3 The Variability within CORDET Product Families

The concept of *variation* or *variability* is the twin concept of commonality in product family engineering. In this case, too, an operationalization of this concept is essential for a successful execution of the domain analysis and domain engineering phases.

In this study, the concept of variation is regarded as related to the concept of *reusability*. The assets provided by a product family are intended to be *reusable*. To reuse a software-based artefact means to use the same artefact in different operational contexts. However, experience teaches that different operational contexts always impose different requirements. Hence, a software artefact can be reused only if it can be *adapted* to match these different requirements.

Thus, in this study, the statement that certain artefacts must be reusable will be treated as equivalent to the statement that those artefacts must be adaptable. In the specific context of product family engineering, however, adaptability must be constrained. This is because the reusable (and hence adaptable) assets provided by a product family must support certain fixed properties which are invariant within the family domain (see discussion in the previous section).

The objective of the domain analysis phase thus becomes the identification of factors and ranges of variations within the shared properties that characterize the systems in the target domain. Similarly, the objective of the domain design phase becomes the definition of mechanisms that allow the assets provided by the product family to be adapted without violating the properties that define them.

The functional and non-functional properties, in other words, are the defining characteristics of a family. They are invariant within the family. Adaptability is necessary in order to have reusability but it must be constrained not to violate the invariant properties.

The background to and justification for the adaptability and variability concepts proposed for this study comes from the work done in ASSERT where property-preserving adaptability mechanisms have been successfully defined for component-based, object-oriented software frameworks (see in particular the ASSERT deliverable [RD27]).

The assumptions proposed and justified in this section are summarized in table 5.3-1. These assumptions, together with the assumptions presented in the previous and the next sub-section, underlie the CORDET Methodology defined in the remainder of this document.

Table 5.3-1: Assumptions About the Variability within the CORDET Product Families

ID	Assumption
A5.3-1	The reusable assets provided by the CORDET product families shall be endowed with property-preserving adaptability mechanisms.
A5.3-2	The search for variability within the CORDET product families shall be operationalized as a search for the parameters with respect to which adaptability must be provided together with the their range of variation.

6 THE CORDET METHODOLOGY – PART I

The CORDET Methodology is the methodology that is used to build the CORDET Generic Architecture. This section defines the CORDET Methodology in terms of its activities and subactivities, their objectives, and their outputs. For clarity, figure 6-1 sketches the overall process that is behind the CORDET Methodology. Note that this is the classical domain engineering process as it is defined in, for instance, the ISO/IEC 12207 standard.

Note that the CORDET Methodology only covers the *creation* of the generic architecture. The *instantiation* of the generic architecture is not covered by the methodology in its present form.

The CORDET Methodology is defined through a set of requirements. Requirements are formulated at the point in the document where the discussion justifying them is presented. They are stated in boxes with the following format:

Ref.	***Requirement***
MRx-y	*<Formulation of the requirement>*

The first column contains an identifier of the requirement. The identifier is formed by the letters 'MR' followed by the number 'x' of the section where the requirement is formulated, and by a sequential number 'y' that identifies the requirement within a certain section. Thus, for instance, requirement MR4.2-3 is the third requirement formulated in section 4.2. The second column in the table gives a concise statement of the requirement.

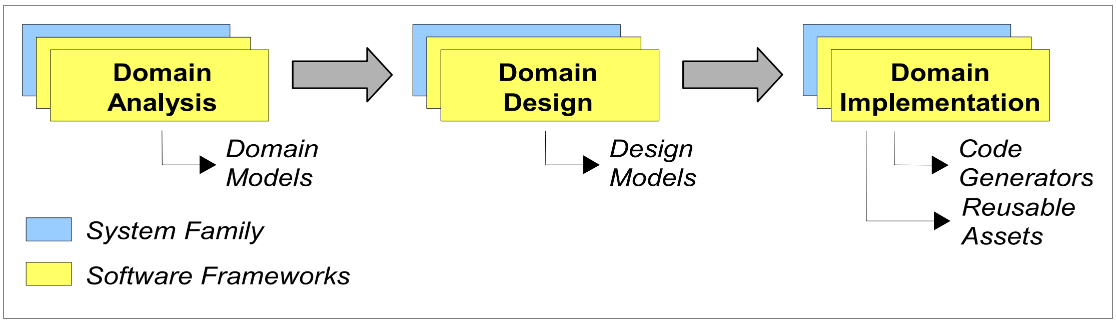

Fig. 6-1: Family Development Process

6.1 General

This section defines the overall objectives of the CORDET Methodology and it defines how it splits the family development process into three activities. The three activities are the classical ones of product family engineering: domain analysis, domain design, and domain implementation.

At this general level, the main peculiarity of the CORDET Methodology is the fact that several parallel development flows are proposed: one aimed at covering the functional part of the target systems and the others aimed at covering their non-functional parts. This approach reflects the drive behind the CORDET Project to separate the handling of non-functional aspects (which are located in the system family) from functional aspects (which are located in the software frameworks).

MR6.1-1	*The objective of the CORDET Methodology shall be the creation of a* generic architecture *for systems within a certain domain (the* target domain *of the architecture).*

MR6.1-2	*The generic architecture shall provide* reusable *and* adaptable *software assets to support the instantiation of systems within its target domain.*
MR6.1-3	*The generic architecture shall consist of a* system family *and one or more* software frameworks.
MR6.1-4	*The system family shall provide reusable and adaptable software assets to support the instantiation of the middleware layer of the systems in the architecture domain.*
MR6.1-5	*A software framework shall provide reusable and adaptable software assets to support the instantiation of one of the functional applications of the systems in the architecture domain.*
MR6.1-6	*The CORDET Methodology shall be implemented as three sequential activities: the* domain analysis activity*, the* domain design activity*, and the* domain implementation activity*.*
MR6.1-7	*Each of the activities identified in the previous requirement shall be performed separately for the system family and for each software framework.*

Requirement MR6.1-6 asks for the three activities – domain analysis, domain design, and domain implementation – to be implemented sequentially. This simply implies that an activity should only be started when the outputs of the previous activity are available but it does not preclude the possibility of an iterative life-cycle where the sequence of three activities is performed more than once.

6.2 The Domain Analysis Activity

The domain analysis activity is the first of three activities in which the CORDET Methodology is divided. This section defines its objectives and its outputs.

MR6.2-1	*The objective of the domain analysis activity shall be the definition of a* domain model *for the system family or the software framework.*
MR6.2-2	*The domain model shall describe the* commonalities *and the* factors of variation *within the target domain of the system family or software framework.*
MR6.2-3	*The commonalities of the domain model for the system family shall consist of the non-functional properties that are shared by all systems in the family's domain and that must be enforced by the reusable assets provided by the family.*
MR6.2-4	*The commonalities of the domain model for a software frameworks shall consist of the functional properties that are shared by all systems in the framework's domain and that must be enforced by the reusable assets provided by the framework.*
MR6.2-5	*The factors of variation of the domain model shall be those with respect to which the (adaptable) assets provided by the system family or software framework must provide adaptation.*

It is recalled that the domain analysis activity must be performed separately for the system family and for the software frameworks that make up the CORDET Generic Architecture. Thus, the output of the domain analysis phase for the generic architecture as a whole will be a set of domain models, one for the system family and one for each software framework.

The last three requirements essentially state that the domain model is the *specification* of the system family or software framework.

A system family or software framework provides software assets that are intended to help developers build systems or applications within a certain domain. The reusable assets fulfill this purpose by implementing the invariant features (the "common properties") of the systems or applications in the target domain and by providing adaptation mechanisms that model the variability across systems or applications in the domain. Thus, the identification of the common features and of the dimensions of variability to be covered by the family or framework assets constitutes a specification for these assets.

Note that the objective of the domain analysis activity is not to identify *all* commonalities or all dimensions of variability within the family or framework domain. The objective is instead to identify those that are *relevant* to the family or framework development process, namely those that will be supported by the family or framework assets (see requirements MR6.2-4 to 6.2-6).

The requirements stated above assume that the domain of the family or framework – namely the set of systems or applications whose instantiation must be supported by the family or framework assets – is known. It might be argued that the definition of this domain ought to be one of the outputs of the domain analysis activity. In practice, however, the definition of the domain is a futile activity since it is simply not possibly to exactly demarcate the set of systems or applications that can be built using a a set of reusable assets. It will always be a question of engineering judgement whether a certain system or application can be efficiently and economically built using certain existing assets.

It is for this reason that the CORDET Methodology opts for an identification of common properties and their factors of variation rather than for an identification of target systems. The target systems are identified implicitly by the definition of the common properties and their factors of variations.

6.3 The Domain Design Activity

The domain design activity is the second of three activities in which the CORDET Methodology is divided. This section defines its objectives and its outputs.

MR6.3-1	*The objective of the domain design activity shall be the definition of a* design model *for the system family or the software framework.*
MR6.3-2	*The design model shall define at design level the reusable assets provided by the system family or software framework.*
MR6.3-3	*The design model shall implement the* shared properties *identified in the domain model of the system family or software framework.*
MR6.3-4	*The design model shall identify the properties whose implementation by the model must be verified through some formal means.*
MR6.3-5	*The design model shall define the* adaptation mechanisms *that allow the factors of variability identified in the domain model of the system family or software framework to be implemented.*
MR6.3-6	*The design model shall identify the conditions under which the application of the adaptation mechanisms preserves the shared properties associated to the design model itself.*

It is recalled that the domain design activity must be performed separately for the system family and for the software frameworks that make up the CORDET Generic Architecture. Thus, the output of the domain analysis phase for the generic architecture as a whole will be a set of design models, one for the system family and one for each software framework.

In general, the purpose of the design models is to show how the shared properties and adaptation mechanisms identified in the domain analysis phase can be implemented.

It would of course be desirable that the design *provably* implements the shared properties of the product family. This will often be possible but it seems unreasonable to ask that this be the case for all shared properties. As a compromise, requirement MR6.3-4 requires that the design models identifies which properties are formally verifiable at design level. Note that the carrying out of the formal verification is not part of the CORDET Methodology. This is regarded as a future extension of the process.

Similarly, it would be desirable to ensure that the adaptation mechanisms proposed by the design model be fully *property preserving*. The reusable assets provided by the CORDET product families will both be endowed with certain properties and will be adaptable. The obvious question that arises in this connection is: do the properties only hold on the reusable assets before they are adapted or are they also guaranteed to hold after the adaptation process?

The adaptation process is property preserving if the properties are guarantee to hold even after the adaptation process. This is obviously desirable but may be impossible to realize in all practical cases. As a compromise, requirement MR6.3-6 requires that the conditions under which property preservation can be guaranteed be identified.

There is a second and subtler implication to requirements MR6.3-4 and MR6.3-5. The CORDET Methodology allocates formal verification activities to the design phase, and not to the domain analysis phase (there are no analogue of requirements MR6.3-4 and MR6.3-5 for the domain analysis activity).

In principle, it would have been possible to mandate that formal models be built at requirements level (namely at domain analysis level) and that some formal verification be performed already at this level. The CORDET Methodology instead took a different approach where properties defined during the domain analysis phase may be formulated using natural language. The level of abstraction and formalization of the domain analysis phase, in other words, is that which is usual for the user requirement phase in single application development processes. There is of course no preclusion to using some kind of formalism to express the shared properties but there is no requirement from the CORDET Methodology that such a formalism be compatible with formal verification techniques.

Formal verifiability in the CORDET Methodology is required only at domain design level. The rationale for this choice is that, in practice, the construction of a formal model requires in any case the preliminary definition of an informal model. Hence, if the domain model (the output of the domain analysis phase) were required to be formally verifiable, then it would be necessary to introduce a new phase before the domain analysis phase to define an informal description of the target product family.

6.3.1 Typology of Design Models

The term “design model” in the previous section (and in most of this document) is used in a very broad sense to designate a repository of design-level information about the reusable assets of a product family. In practice, the structure of this repository may be very complex because the family encompasses a potentially large set of individual systems or applications.

This multiplicity of representation is achieved by building into the family assets adaptation mechanisms that allow the same software asset to match the potential needs of a large number of individual systems or applications. Hence, the structure of the design model depends crucially on the expressive power and flexibility of the adaptation mechanisms.

During the family instantiation process, the family-level adaptation mechanisms are applied to the family-level design model to transform it into the design model of the target system or application. Nowadays, the latter will invariably take the form of one single UML2 model.

It is therefore natural to assume that the family-level model, too, must somehow be based on one or more UML2 models. The family-level model, however, cannot always be reduced to a single UML2 model. This is due to the fact that, although UML2 formalism is well-suited to capturing invariant properties, it may not be sufficiently expressive to capture variability. Hence, although a family-level design model may have a UML2 core, it will in general consist of more than just a plain UML2 model.

The basic ways in which the family-level design model can be structured around its UML2 core are:

- *Single-Model Case*: the family-level design model consists of one single UML2 model. This is possible when the adaptation mechanisms can be modeled within the UML2 model itself. During the family instantiation process, the adaptation mechanisms are used to transform the family-level UML2 model into the UML2 model of the target system or target application.

  Object-oriented software frameworks offer a typical example of this case because they use object-orientation as their chief adaptation mechanism and because object-oriented adaptation mechanisms can be represented within a UML2 model.

- *Multiple-Model Case*: the family-level design model consists of several UML2 models where each represents one variant of target application or target system. In an extreme case, the family-level design model might simply be implemented as the collection of the UML2 models of all the systems or applications in the family domain.

  In this case, the adaptation process consists in selecting one out of all the available UML2 models.

- *Generative-Model Case*: the family-level design model consists of a model generator that can automatically generate the UML2 model of a target system or target application. The input to the generator would typically be an instance of the domain model. Thus, the adaptation of the assets to the requirements of a particular target system or target application is performed on the domain model by selecting the features that are of interest for a particular instance of the family.

- *Transformative-Model Case*: the family-level design model consists of a single UML2 model together with a transformation program (a model-level aspect weaver) that can automatically transform it into another UML2 model. The transformation program implements the adaptation mechanisms and the output of the transformation is the UML2 model of the target system or target application.

- *Meta-Model Case*: the family-level design model consists of a single UML2 meta-model. The instances of this meta-model (namely the UML2 models that conform to the meta-model) are the models of the systems and applications in the family domain. The meta-model thus captures the design constraints to which the models of the family instances must obey.

The cases listed above are “pure” cases but, obviously, in practice mixed solutions with elements of several of the above cases are possible.

The selection of the applicable case for a particular product family depends on the range of diversity of the systems or applications in the family domain and on the expressive power of the available adaptation mechanisms.

For the case of the CORDET Generic Architecture, the single-model case will be used for the software frameworks and the meta-model case will be used for the system family. A more detailed discussion of this choice and the applicable requirements can be found in sections 7.3.3 and 7.3.4.

6.4 The Domain Implementation Activity

The domain implementation activity is the second of three activities in which the CORDET Methodology is divided. This section defines its objectives and its outputs.

MR6.4-1	*The objective of the domain implementation activity shall be the development of* a code generator *for the system family or the software framework.*
MR6.4-2	*The code generator shall be capable of transforming the design model of the reusable assets of the system family or software framework into their source code implementation.*
MR6.4-3	*It shall be possible for the same code generator to process design models from more than one product families (system family or software framework) and to generate a joint set of reusable assets.*
MR6.4-4	*It shall be possible to apply the adaptation mechanisms defined in the design model to the source-level reusable assets generated by the code generator without manually changing their source code.*

According to requirement MR6.4-1, the primary output of the domain implementation activity is a code generator. One important implication of this requirement is that the code generator is seen as being domain-specific. Many software development environments available on the market at present offer code generating facilities but their code generators are normally intended to be “generic”. Some customization options are offered but the assumption is that the same code generator can be used for all kinds of applications.

This approach is regarded as inadequate for embedded applications for two reasons. Firstly, embedded applications normally must undergo certification processes that, among other things, may impose certain coding standards and coding rules. Secondly, minimization of memory and execution overheads will often require that the code be tailored to the peculiarities of the underlying operating system or even of the underlying processor.

For these reasons, the CORDET Methodology assumes that different domains will have different coding needs and it therefore makes the code generator an output of the product family development process.

Requirement MR6.4-2 states that the code generator must be capable of processing the family design models in their entirety and that it must be capable of generating their complete source code. The situation is, in other words, excluded where, in addition to the code generation process, some manual coding must be performed.

In general, it is not realistic to assume that a complete embedded application can be entirely generated from its models. No modelling language is sufficiently expressive to capture all aspects of a complex embedded application. However, the assumption behind requirement MR6.4-2 is that, at least at family level, it is possible to capture all relevant aspects of the reusable family assets in their models. This is expected to be possible because the family assets will normally only capture the more abstract aspects of the target systems and applications and these are precisely those aspects that can be expressed well using available modelling languages.

The final output of the domain implementation activity are the reusable assets of the product families. According to MR6.4-2, they must be available as source code. It would in theory be possible to have them only as binary code but, given the critical nature of the target applications, a source-level implementation is mandated since it is expected that some kind of code inspection will be part of the certification process both for the product family as a whole and for the systems instantiated from it.

The CORDET Methodology foresees several parallel flows of activities: one for the system family and one for each software framework. These activity flows are completely separate at domain analysis and domain design level but requirement MR6.4-3 allows a partial merging to be done at domain implementation level. The requirement foresees the possibility that the same code generator may be defined to process design models across product family boundaries. This option is introduced to allow optimization of generated code.

The generated code must implement the adaptation mechanisms defined in the design models. Requirement MR6.4-4 asks that adaptation take place without manual changes to the source code of the family reusable assets. The generation of the application-specific building blocks from the family-level building blocks, in other words, must take place without the need to perform any manual changes to the source code of the latter.

This requirement is important because it allows some form of certification to be performed at family level and to be reused at application level. If the family-level reusable assets had to be manually modified during the family instantiation process, then their code would have to be re-certified since manual modifications would undermine the certification results performed at family level.

Note, finally, that requirement MR6.4-4 only bans *manual* modifications. *Automatic* modifications – for instance through some kind of aspect weaving – are allowed since certification of the code transformation tool would still allow certification results obtained at family level to be re-used at application level.

7 THE CORDET METHODOLOGY – PART II

This section defines the methodological requirements for the second part of the CORDET Methodology. These requirements are intended to support the execution of the activities defined in section 6.

The next section defines the constraints that informed the definition of the CORDET methodological requirements specified in this section. Sections 7.2 to 7.4 present the methodological requirements for each of the three activities of the CORDET Methodology (domain analysis activity, domain design activity, and domain implementation activity).

7.1 Methodological Constraints

The requirements presented in the section 6 defined the activities to be performed to develop the CORDET Generic Architecture. The requirements specified in the present section instead define the methodological means to be used to implement these activities. In principle, other methodological choices would have been possible. The specific choices that are made in this section are driven by the particular needs of the CORDET Project and by the industrial and technological heritage of the CORDET project team.

More specifically, the following constraints have driven the selection of the methodological rules presented in this section:

1. The CORDET activities must be performed within comparatively modest temporal and financial resources (target project duration of one year and total budget of 250 kEUR).
2. The results of the CORDET Project must be applicable (at least in principle) to the satellite missions studied in the State Of The Art Survey task[2].
3. The focus of functional design activities must be on the AOCS and DH subsystems.
4. There must be continuity with the results of the ASSERT Project.

The first constraint imposes an emphasis on simplicity and the use of well-proven techniques since these are the best means to minimize budget and schedule risks.

The second constraint arises from a desire to ensure that the CORDET Generic Architecture be relevant to concrete missions as they are done at present.

The third constraint derives from a choice made at the beginning of the CORDET Project. It was clear that it would not be possible to cover all functional subsystems of on-board satellites and hence a decision was taken to limit the project to the AOCS and DH domains which are the most complex and the most representative of satellite on- board functionalities.

The fourth constraint stems from the fact that virtually all CORDET key personnel have been active members of the ASSERT Project and they quite naturally see the CORDET Project as a means to verify the industrial viability of some of the concepts and ideas that were proposed in ASSERT.

Note that the last constraint is related to the first one since use of ideas and concepts first explored in ASSERT is a means to reduce budget and schedule risks.

7.2 Methodological Requirements – Domain Analysis

This section presents the methodological requirements for the domain analysis activity. The methodological approach proposed for the domain analysis phase is inspired by the well-known FODA Methodology as is recommended in RD-34.

[2]The State of The Art Survey task is one of the tasks of the CORDET Projects. The results of this survey are documented in a dedicated CORDET deliverable.

7.2.1 Identification of Functional Domains

According to assumption A5.1-3, the CORDET Generic Architecture consists of a system family and one or more software frameworks covering the functional domains of the target systems. It follows from this assumption that one of the basic inputs to the architecture development process must be the identification of the functional domains for which software frameworks must be developed.

MR7.2.1-1	*One of the inputs to the domain analysis of the CORDET Generic Architecture shall be the identification of the functional domains for which software frameworks must be provided by the Generic Architecture.*

It is important to stress that the identification of the functional domains is an *input* to the domain analysis phase. This was already implied by the first part of the CORDET Methodology.

One could of course imagine a different process and a different methodology where domain analysis is performed at two levels: at the top level, the identification of the functional domains is performed and, at a lower level, each selected functional domain is characterized.

The choice of one single level of domain analysis in CORDET is dictated by three factors. Firstly, there is broad agreement within the European space community about what are the main functional domains and there is therefore not much point in debating this issue. Secondly, the limited resources available to the project (constraint 1 in section 7.1) advise against a two-level domain analysis. Thirdly, the focus on the AOCS and DH subsystems (constraint 3 in section 7.1) effectively preempts the decision about the target functional domains for the CORDET Generic Architecture.

7.2.2 Bottom-Up vs Top-Down Approach

In general, there are two basic ways to perform the domain analysis for a product family. The first one takes a top-down approach. In this case, the family designer defines a set of requirements for the family in a manner that is independent of existing systems and building blocks and that is aimed at optimizing the family architecture with respect to certain predefined design parameters.

The second way to performing domain analysis has a bottom-up flavour. In this case, the family designer considers a set of existing systems in the domain of interest and tries to identify their commonalities and variabilities. The objective of this second approach is to arrive at an architecture that maximizes the reuse of existing design concepts and building blocks.

The first approach is more ambitious in that it holds the promise of arriving at an architecture that is truly optimal with respect to the selected design criteria. This approach, however, is also riskier because, in its strive for optimality, it is more likely to deviate sharply from existing solutions. The second approach is more conservative and less risky because it simply aims to identify the “best practice” within the domain and to capture it in the Generic Architecture.

In view of constraints 1 and 2 in section 7.1, the bottom-up approach is selected for the CORDET Generic Architecture. The set of systems that will serve as an input for the domain analysis will be those studied in the State-of-the-Art Survey task of the CORDET Project.

MR7.2.2-1	*The primary input for the domain analysis phase of the CORDET Generic Architecture shall be the results of the CORDET State-of-the-Art Survey.*
MR7.2.2-2	*The shared properties to be supported by the CORDET Generic Architecture shall be derived from an analysis of the functional and non-functional*

	properties shared by the satellite missions considered in the CORDET State-Of-The-Art Survey.
MR7.2.2-3	*The factors of variations to be supported by the CORDET Generic Architecture shall be derived from an analysis of the variability found in the satellite missions studied in the CORDET State-Of-The-Art Survey.*

Note that, as implied by the methodological requirements MR7.2.2-2 and MR7.2.2-3, the intention is not to create a Generic Architecture that supports the instantiation of *all* the systems studied in the State-of-the-Art Survey. Rather, the intention is to use the missions studied in the State-Of-The-Art Survey to identify the most significant properties in the on-board domain together with their ranges of variations with the objective of supporting the instantiation of "typical" systems.

No specific methodological requirements are given on how the significant properties may be identified among all the properties exhibited by the missions analyzed in the State-Of-The-Art Survey. Such a choice can only be based on engineering judgment.

Note also that, as a consequence of requirements MR7.2.2-1 to MR7.2.2-3, the selection of the missions to be analyzed in the State-Of-The-Art Survey is *outside of and prior to* the domain analysis phase. This selection is driven by engineering considerations (what kind of missions are amenable to standardization?) and by business considerations (what kind of missions is it desirable to standardize?).

The fundamental reason why the State-Of-The-Art Survey is kept outside the formal CORDET Methodology is that its value lies precisely in its being informal. There are some obvious activities that may need to be performed in a state-of-the-art survey (interviews with domain experts, interviews with project engineers, analysis of project documents, etc) but there is no point in trying to prescribe the order in which these activities should be performed or in trying to break them up into lower-level steps. This would simply create administrative overheads without adding anything to the value of the survey.

In general, an activity that is part of a formal process must always be preceded by some informal work where inputs are prepared and where a first iteration of the target activity is performed without being subject to the constraints of a formal process or methodology. Hence, the attempt to pull the State-Of-The-Art Survey into the CORDET Methodology would simply result in a new activity having to be (more or less implicitly) defined that is prior to the survey itself and that is outside the CORDET Methodology.

The value of such an activity is dubious at best and its execution is in any case not compatible with the tight resources available to the CORDET Project (constraint 1 in section 7.1).

7.2.3 Domain Model – Shared Properties

According to the process requirements defined in section 6.2, the domain model must describe the commonalities (shared properties) of the CORDET target systems and applications and their factors of variations. The process requirements, however, do not say anything about the exact form of the domain model and about how the commonalities and their factors of variations should be expressed.

In general, in the CORDET Methodology, formal modeling is only expected at the domain design phase (see discussion of process requirements MR7.2.3-4 and MR7.2.3-5 in section 6.3). A formal model must be expressed using a formal language. In order to be understandable and usable, such a model must be accompanied by a description in natural language. In fact, prior to building a formal model for some design artefacts, it is normally necessary to describe them informally in natural language. The CORDET Methodology allocates the formal modelling activities to the design phase and it accordingly proposes that the models to be developed in the domain analysis phase be expressed in natural language.

The shared properties will thus be expressed in natural language. As discussed in section 7.3.1, they will be formalized in the domain design phase. Unique identifiers will be attached to them to allow traceability to the formal properties defined at design level.

Although no formal language is proposed to express the shared properties, it is still helpful to define a set of terms that can be used to formulate the shared properties. For this purpose, the CORDET Methodology mandates the construction of a *domain dictionary*.

The domain dictionary gathers together the terms that are necessary to express the shared family properties. In general, the terms in the domain dictionary can designate either *abstractions* that exist in many applications or systems in the domain but take a different form in each, or they can designate *domain-invariant functionalities* that share the same implementation in all applications or systems in the domain.

The use of a domain dictionary is proposed by several methodologies for domain analysis. The domain dictionary is usually complemented by a *logical model* or *concept model* that should describe how the items defined in the domain dictionary relate to each other. The idea is that the domain dictionary defines the key abstractions and concepts of the target domain whereas the logical model describes their mutual relationships.

In the CORDET Methodology, no such logical model is required since this is already implied by the set of shared properties. The shared properties, in other words, are intended to capture the relationships among the items defined in the domain dictionary.

Figure 7.3.1-1 in section 7.3.1 gives an overview of how shared properties are handled at various levels by the CORDET Methodology.

MR7.2.3-1	*The shared properties of the CORDET Product Families shall be expressed in natural language.*
MR7.2.3-2	*A unique identifier shall be attached to each shared property defined in the domain analysis phase.*
MR7.2.3-3	*A* domain dictionary *shall be built to define the key abstractions and invariant functionalities in the target domain.*
MR7.2.3-4	*The shared properties shall be expressed in terms of the entries of the domain dictionary.*

7.2.4 Domain Model – Factors of Variation

The definition of the factors of variations for the shared properties is the second element of the domain model.

The definition of the factors of variation must encompass, at a minimum, the following elements:

1. It must identify the shared property or other feature to which the variation applies.
2. It must identify the range of the variation (the set of legal values for the factor of variation).
3. It must identify any default values for the factor of variation.
4. Interactions among the various factors of variation must be described. These interactions will typically take the form of constraints on the legal combinations of the values of the factors of variations.

Each factor of variation must be clearly identified. This is important to allow traceability to the domain design level where it must be demonstrated how the variability behind the factor of variation is captured and implemented in the reusable and adaptable asset offered by a product family.

Like the shared properties and for the same reasons, the factors of variation will be defined in natural language.

MR7.2.4-1	*The factors of variation identified in the domain analysis phase shall be described in natural language.*
MR7.2.4-2	*For each factor of variation identified in the domain analysis phase, the shared property or other feature to which the variation applies shall be identified.*
MR7.2.4-3	*For each factor of variation identified in the domain analysis phase, its range of variation shall be defined.*
MR7.2.4-4	*For each factor of variation identified in the domain analysis phase, applicable default values shall be identified.*
MR7.2.4-5	*Interactions and mutual dependencies among the factors of variation shall be defined.*
MR7.2.4-6	*A unique identifier shall be attached to each factor of variation defined in the domain analysis phase.*
MR7.2.4-7	*A feature model shall be built to describe the variability in the domain model.*

7.2.5 Feature Models

In principle, the domain dictionary, the shared properties, and the factors of variations are sufficient to describe the domain model. However, in addition to these three elements expressed in natural language, it is useful to provide a *feature model* to capture at least a part of the domain model in a single formal model.

The purpose of the feature model is to present a concise overview of the domain model. Since feature models are especially adept at describing variability, they can in particular be used to model the relationships among the factors of variation and the constraints on their legal combination.

In addition to serving as an economical way to describe variability within families, feature models may be useful as a way of either building or customizing a design model. This point is especially important and its importance straddles the domain analysis and the domain design phase. It is discussed in section 7.3.2 below.

Consistent use of feature models in the context of the CORDET Methodology requires that the key concepts of the methodology – domain dictionary entries, shared properties, and factors of variation – be mapped to features in a feature model.

Both the entries in the domain dictionary and the shared properties can be seen as “features” since both describe characteristics of a system that are relevant to its end users (and this is the standard definition of “feature”). Similarly, the factors of variations can be seen as a description of a feature that can take different values within the family.

Given that the emphasis of the feature model is on the description of variability, the latter kind of features must be included in the feature model. There is instead no need to map all domain dictionary entries and shared properties to features in a feature model. This only needs to be done for those entities and properties that are affected by variability and that enter in the definition of the factors of variation.

It must finally be stressed again that, in a difference with other feature-based approaches to domain modelling, the feature model as proposed by the CORDET Methodology should not be seen as *equivalent to* the domain model. The feature model is *part of* the domain model: it is an auxiliary tool to present an overview of the domain model with a special emphasis on the representation of variability within the domain.

MR7.2.5-1	*A feature model shall be built to describe the variability in the domain model.*

7.2.6 Content of Domain Model

By way of summary of the previous three sections, a domain model according to the CORDET Terminology will consist of the following items:

- A *domain dictionary* to define (in natural language) the terms required to express the shared properties
- A set of *shared properties* to define (in natural language) the invariants within the target domain
- A set of *factors of variations* to define (in natural language) the variability within the target domain
- A *feature model* to present an overview of the domain model and in particular of its variability

Figure 7.2.6-1 below illustrates the structure of the domain model proposed by the CORDET Methodology.

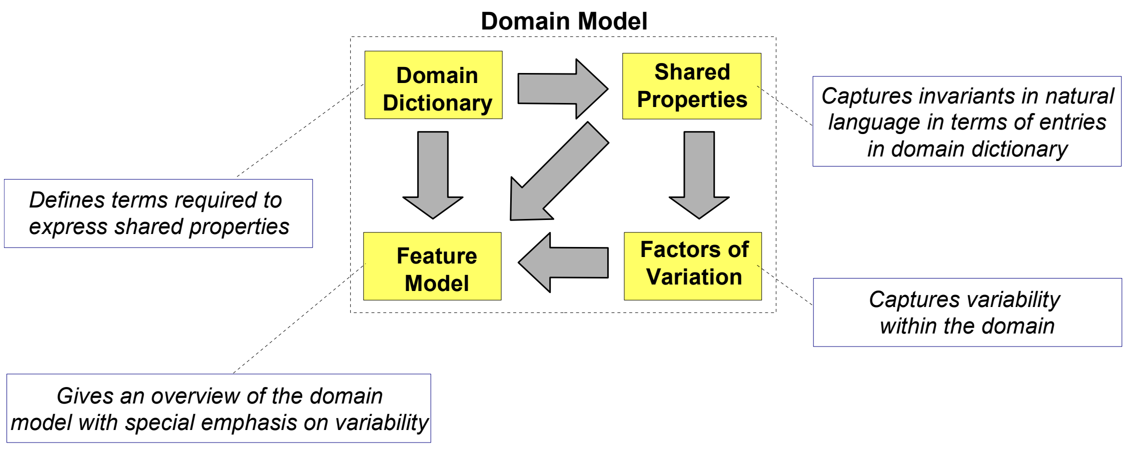
Domain Model
Domain Dictionary
Shared Properties
Feature Model
Factors of Variation
Captures invariants in natural language in terms of entries in domain dictionary
Defines terms required to express shared properties
Captures variability within the domain
Gives an overview of the domain model with special emphasis on variability

Fig. 7.2.6-1: Structure of Domain ModelIterative Definition of Domain Model

7.2.7 Iterative Definition of Domain Models

As already mentioned in section 6.1, iteration is possible in the execution of the CORDET methodology and its activities.

Some iteration in the definition of the domain model is obviously desirable. In view of the limited time available for the project, only two iteration cycles are foreseen for the CORDET Domain Model. The end of the first iteration cycle should be marked by a review where the project stakeholders can submit their concerns and their suggestions for improvements.

MR7.2.7-1	*The definition of the domain model shall be done in two iterations.*
MR7.2.7-2	*At the end of the first iteration, a review shall be performed where selected CORDET stakeholders shall be asked to provide comments.*

7.3 Methodological Requirements – Domain Design

This section presents the methodological requirements for the domain design activity.

The objective of the domain design phase is the definition of a design model. The design model must represent at design level the shared properties and the factors of variation identified at domain analysis level.

The CORDET Methodology specifies that distinct design models are to be generated for the CORDET System Family and for the CORDET Software Frameworks. The former captures non-functional system-level aspects of the CORDET Generic Architecture, the latter captures functional software-level aspects of the CORDET Generic Architecture. The CORDET Methodology proposes two distinct methodologies for these two levels of design. They are discussed in sub-sections 7.3.3 and 7.3.4.

In section 6.3.1, five basic types of family-level design models were identified. The CORDET Methodology chooses the *single model type* for the software frameworks and the *meta-model type* for the system family. This choice is dictated by the different kinds of variability that is behind the software frameworks (functional variability) and system families (non-functional variability). This issue is discussed in greater detail in section 7.3.2.

Thus, the design model of a CORDET Software Framework will consist of a single UML2 model that will describe both the shared properties and the factors of variation associated to the framework. Use of a single UML2 model is possible because the CORDET Software Frameworks are object-oriented and object-oriented adaptation mechanisms are both sufficiently powerful to capture the functional variability behind the framework, and amenable to representation with UML2 formalism.

The design model of the CORDET System Family will instead consist of a UML2 meta-model. The meta-model can be seen as a generative device that captures and enforces the domain-wide design constraints to which the CORDET system architecture must obey. These constraints are defined to ensure that the system-level properties defined in the domain analysis phase are guaranteed to be satisfied.

As discussed in section 7.2.1, two software frameworks will be developed for the CORDET Generic Architecture targeting, respectively, the AOCS and the DH subsystems of satellites. In general, a generic architecture for on-board systems should provide several software frameworks, one for each on-board subsystem. The software frameworks should obviously be independent of each other since they are intended to be used to develop separate applications.

The contours of on-board subsystems, however, vary from mission to mission. It is therefore important that the software frameworks be inter-operable in the sense that it should be possible to instantiate them within the same application. This will allow their use in missions where some or all on-board subsystems are merged in the same application.

MR7.3-1	*The design model of a CORDET Software Framework shall consist of a single UML2 design model.*
MR7.3-2	*The design model of the CORDET System Family shall consist of a single UML2 design meta-model.*
MR7.3-2	*The CORDET Software Frameworks shall be designed to be inter-operable.*

7.3.1 Mapping Shared Properties to the Design Level

One important function of the design model is to show how the design of the reusable family assets implements the shared properties identified at domain analysis level.

The methodological requirements to be followed when implementing the shared properties are presented separately for the case of functional and non-functional properties in sections 7.3.3

and 7.3.4. In general, the important point is that each property identified at domain analysis level should be translated into one or more design-level properties. The objective is to demonstrate that the proposed design covers all the properties defined at domain analysis level.

Process requirement MR6.3-4 prescribes that, for each design-level property, it must be stated whether or not the property should be verified formally. From a methodological point of view, the only sensible translation of this requirement is to ask that, for each property, an indication be given of how that property could potentially be verified formally. The verification itself, however, remains outside the scope of the CORDET Methodology.

Thus, shared properties in the CORDET Methodology exist at two levels. At domain analysis level, they encapsulate the developer's knowledge of the family domain and are expressed in natural language. At this level, the shared properties can be seen as (part of) the specification of the reusable assets to be offered by the product family.

At domain design level, the shared properties are expressed more formally and they reflect potentially provable characteristics of the proposed design. At this level, the shared properties can be seen as a description of what the family reusable assets will offer to the product developers.

A link must be established between the two levels at which the shared properties are defined to show how the domain level properties are mapped to the design level properties. The objective is to show how the specification of the family (the domain level properties) are implemented in the design of the family of reusable assets.

As discussed in section 7.2.3, the shared properties are expressed in terms of the entries in the domain dictionary. The mapping from shared properties at domain analysis level to their implementation in the design model must therefore cover both the entries in the domain dictionary and the properties themselves.

The precise nature of the design elements to which the shared properties and the dictionary entries are mapped is different for the system family and for the software framework and is therefore defined separately for each in sections 7.3.3 and 7.3.4.

It is finally possible to conceive of a third level at which the shared properties exist. If it is desired to formally verify that the shared properties hold, then it may be necessary to set up one or more verification models that represent the relevant parts of the design model expressed in a suitable verification language. In this case, the properties will have to be re-stated within this verification-oriented formalism.

As already indicated in section 6.3, however, formal verification activities are currently not included in the CORDET Methodology. This potential third representation of the shared properties of a family will therefore not be considered further.

The next figure illustrates the three levels at which shared properties may exist and their mutual links.

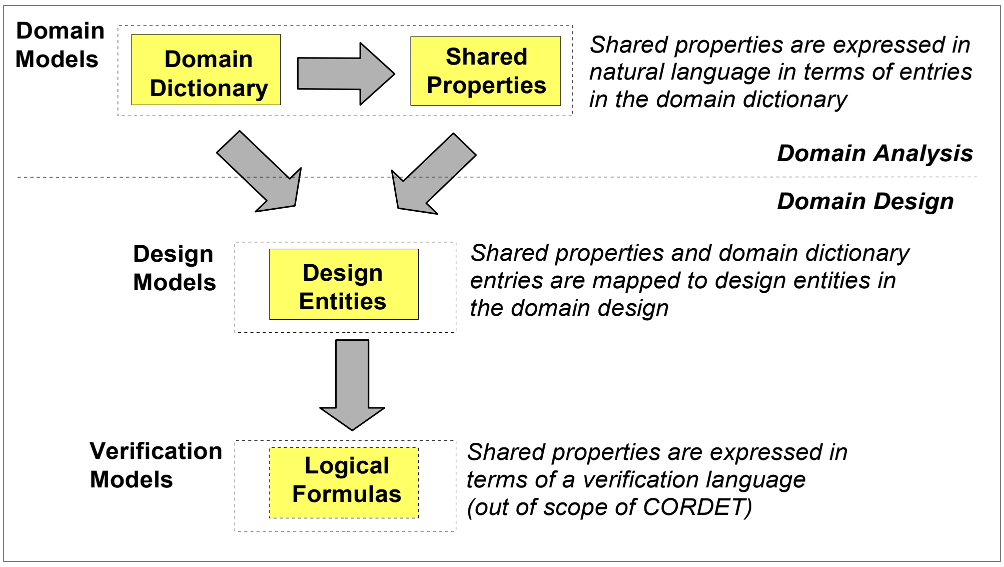
Domain Models
Domain Dictionary
Shared Properties
Shared properties are expressed in natural language in terms of entries in the domain dictionary
Domain Analysis
Domain Design
Design Models
Design Entities
Shared properties and domain dictionary entries are mapped to design entities in the domain design
Verification Models
Logical Formulas
Shared properties are expressed in terms of a verification language (out of scope of CORDET)

Fig. 7.3.1-1: Shared Properties at Domain Analysis and Domain Design Level

MR7.3.1-1	*The shared properties and the domain dictionary entries defined at domain analysis level shall be transposed to the design level and shall be re-formulated in terms of the abstractions defined by the design model.*

7.3.2 Mapping Domain-Level Variability to Design-Level Adaptability

The description of the variability within the family domain is one of the two key elements of the domain model built in the domain analysis phase. As discussed in section 6.3, during the domain design activity, this variability must be mapped to adaptation mechanisms in the design model (see requirement MR6.3-5).

When UML2 models are used as the basic element of a design model, then there are two ways in which this domain-level variability can be mapped to design-level adaptability: *Within-the-Model Adaptability* (or WtM Adaptability) and *Outside-the-Model Adaptability* (or OtM for short).

In the WtM Adaptability case, the design model consists of one single UML2 model and the variability represented by a certain factor of variation is modelled *within* this model using a design-level adaptation mechanism such as object-orientation, templates, or configuration parameters. In this case, the product designer who instantiates the product family only needs to use the one single UML2 model that contains all the information about the family shared properties and their variability.

The OtM case instead arises when the adaptation mechanisms available at design level are not sufficiently powerful to cover the variability expressed by a certain factor of variation. In this case, the presence of the factor of variation must be modelled *outside* the UML2 model. In practice, this can be done either by providing several UML2 models (one for each potential value of the factor of variation), or by providing a way to customize the UML2 model, or by providing a way to generate the UML2 model. The latter could for instance be done by generating the UML2 model from a description of the factor of variation.

In the OtM case, the design model may consist of a set of UML2 model (one for each variation option), or of a model generator (to generate the UML2 model from a description of the factors of variation), or of a UML2 model and a program that can automatically modify it (to customize it to take account of the factors of variation).

Note that both the WtM and OtM forms of adaptability may be used in the same product family (but for different factors of variation). In general, WtM Adaptability is preferable because it is simpler. OtM Adaptability should only be used for factors of variation expressing variability that cannot be handled within the UML2 model.

In order to clarify the distinction between WtM and OtM Adaptability, an example may be useful. Consider a domain model where a factor of variation V is present that, within the target family, can take two possible values: V_1 and V_2.. This means that a product (either a system or an application) instantiated from the family will be characterized either by the value V_1 or by the value V_2.. When the design model for the target family is built, it is accordingly necessary to define an adaptation mechanism that allows the reusable assets provided by the family to be adapted to implemented either the V_1 option or the V_2 option for the factor of variation.

The WtM case arises when it is possible to build a UML2 model where the possibility of selecting either the V_1 option or the V_2 option is described within the model itself. For instance, the designer might have an abstract method in one of the classes in the UML2 model and he might say that option V_1 corresponds to one particular way of overriding this abstract method whereas option V_2 corresponds to a second way of overriding this same abstract method.

Or the designer might include a boolean parameter in his UML2 model and might specify that option V_1 corresponds to a situation where the boolean parameter is initialized with the value TRUE whereas initialization with value FALSE corresponds to option V_2.

In both previous cases, the variability implied by the selection between the two options V_1 and V_1 is modelled within the same UML2 model and this UML2 model is the design model that must be generated in the domain design phase.

Consider instead a second case where the switch from the V_1 to the V_2 option requires some major reshuffling of the classes in the UML2 model and of their mutual relationships. This would represent a typical case of OtM Adaptability. In such a case, the only reasonable option would be to have two UML2 models corresponding to the two values of the factor of variation.

In a less extreme case of OtM Adaptability, the switch from the V_1 to the V_2 option may require a systematic change to be made to the UML2 model (for instance, adding a particular method to a certain category of classes, or changing the number of parameters in certain methods, etc). In such a case, one possible solution would be to build a model generator that reads a description of the factor of variation and generates the UML2 model accordingly. An alternative solution is to treat the factor of variation as an *aspect* (in the technical sense of the Aspect Oriented Programming paradigm) that must be woven onto the design model.

The important point to stress is that in all the previous cases the application of the adaptation mechanism will involve either the creation of a new UML2 model or the customization of a UML2 model. The design model then may include a model generator or a model aspect weaver since the model generator or the model weaver are the repository of the information about how adaptability to the factors of variations identified at domain level is implemented.

In the OtM case, the feature model built in the domain analysis phase is the obvious candidate for serving as an input to the model generator or the model weaver. More precisely, the model generator or the model weaver should be driven by an *instance* of the feature model.

Section 6.3.1 discussed the structure of the family-level design model and identified five basic structures for it. The main discriminant among these five structural types is the way in which domain variability is represented in the design model. It therefore becomes possible to establish a correlation between the type of design model and the type of adaptation technique. This correlation is presented in table 7.3.2-1.

As already noted, there is a general preference for WtM Adaptability but this preference cannot be always accommodated. In particular, most of the adaptation mechanisms offered by mainstream software technology are aimed at functional adaptation. Hence, in the case of the CORDET Generic Architecture, it is anticipated that adaptability with respect to functional variation can be implemented as WtM Adaptability. Methodological requirement MR7.3-1 accordingly specified that the design model of the CORDET Software Framework must be of the single-model type. As indicated in table 7.3.2-1, this type of design model relies exclusively on WtM Adaptability.

By contrast, adaptability with respect to non-functional variation must be implemented as OtM Adaptability. Table 7.3.2-1 shows that there are several kinds of design models that are primarily based on OtM Adaptability. Requirement MR7.3-2 selects the meta-model type for the CORDET System Family because of the heritage from the ASSERT project where non-functional architectural constraints were captured through a meta-model.

Table 7.3.2-1: Correlation Between Design Model Type and Adaptability Mechanism Type

Design Model Type	Type of Adaptability Mechanism
Single-Model Case	WtM Adaptability (by definition of WtM Adaptability) .
Multiple-Model Case	OtM Adaptability to select among models. WtM Adaptability possible within each model.
Generative-Model Case	OtM Adaptability to define the generator. WtM Adaptability possible within the generated model.
Transformative-Model Case	OtM Adaptability to define the transformation program. WtM Adaptability possible within the transformed model.
Meta-Model Case	OtM Adaptability.

MR7.3.2-1	*When mapping domain-level variability to design-level adaptability, preference shall be given to the use of WtM Adaptability mechanisms.*
MR7.3.2-2	*Factors of functional variation (factors of variations applicable to the CORDET Software Frameworks) shall be mapped using WtM Adaptability Mechanisms.*

7.3.3 Design Methodology – Software Frameworks

The FW Methodology [RD27] is a design methodology that was specifically developed to guide the design of software frameworks[3]. The FW Methodology uses the term "software framework" in the same sense as the CORDET Methodology. The CORDET Methodology therefore takes over the FW Methodology as it stands and it mandates its applicability to the design of the software framework part of the CORDET Generic Architecture.

The FW Methodology only covers the development of a design model of a software framework. The CORDET Methodology also requires that a link be established between the domain analysis phase and the domain design phase. In particular, the shared properties defined in the domain model must be transposed to the design level (see requirement MR7.3.1-1) and the domain-level variability must be mapped to design-level adaptability (see requirement MR6.3-5). These mappings from the domain model to the design model are intended to ensure that the specifications of the framework are correctly implemented in its design.

The mapping rules specified by the CORDET Methodology for the shared properties of software frameworks are as follows:

[3] A web site has been set up giving access to all the documentation and software related to the FW Methodology. The current address is: http://control.ee.ethz.ch/~ceg/assert/ch.ethz.fwprofile/

- Each entry in the domain dictionary of a software framework must be mapped to an element in the UML2 design model of the framework (a class, a method, an attribute, etc). The UML2 element implements at design level the abstraction described by the domain dictionary entry.
- Each shared property formulated in the domain model of a software framework must be mapped to an assume-guarantee contract defined on a functional component or cluster of functional components in the design model of the software framework. The assume-guarantee contract represents the translation of the property expressed in natural language at domain analysis level into a more formal property defined on the UML2 design model.

The term "assume-guarantee contract" is used in the following sense. The "assume part" of the contract specifies the way in which the component or group of components to which the property is associated is intended to be used. The "guarantee part" specifies the behaviour that the component or group of components must guarantee. A concrete example of how this kind of assume-guarantee contracts is used in framework design can be found in [RD-35].

Figure 7.3.1-1 describing the levels of representation of shared properties can thus be recast as in figure 7.3.3-1 for the specific case of software frameworks.

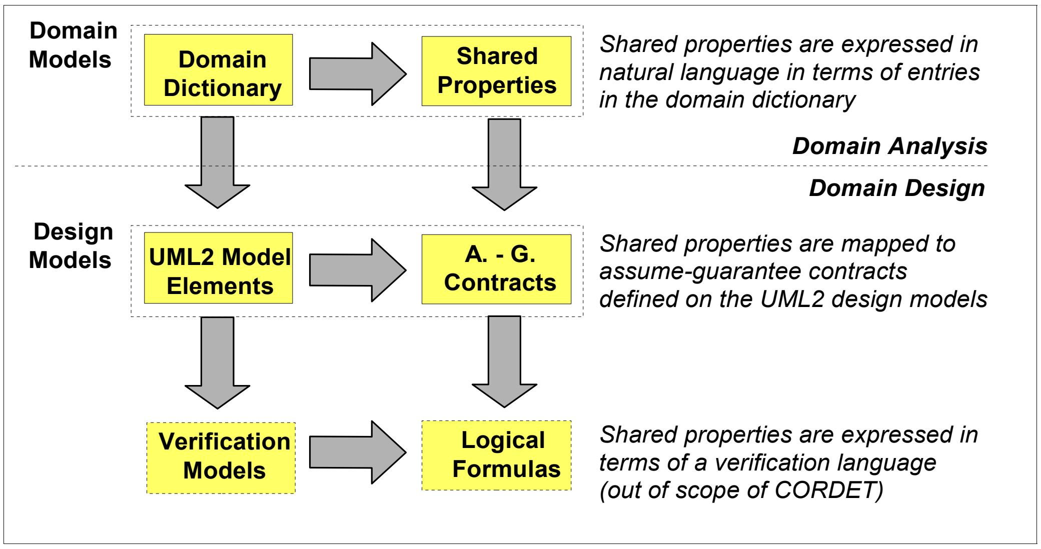
Domain Models
Domain Dictionary
Shared Properties
Shared properties are expressed in natural language in terms of entries in the domain dictionary
Domain Analysis
Domain Design
Design Models
UML2 Model Elements
A. - G. Contracts
Shared properties are mapped to assume-guarantee contracts defined on the UML2 design models
Verification Models
Logical Formulas
Shared properties are expressed in terms of a verification language (out of scope of CORDET)

Fig. 7.3.3-1: Mapping of Shared Properties for Software Frameworks

The factors of variations, like the shared properties, must be mapped from the domain model to the design level. The FW Methodology that is mandated for the CORDET Software Frameworks (see section 7.3.3) mandates the explicit identification of *points of adaptation* in the framework design. These are the natural target for the factors of variation identified in the domain model.

The framework points of adaptation implement an object-oriented form of adaptation. Although this is the primary form of adaptation baselined for the CORDET Frameworks, other forms of adaptation are also acceptable. For this reason, it is not possible to state that every domain-level factor of variation must be mapped to a design-level point of adaptation.

MR7.3.3-1	*The CORDET Software Framework shall be designed in accordance with the FW Methodology.*
MR7.3.3-2	*Each domain dictionary entry of a software framework domain model shall be*

	mapped to the element in the UML2 design model of the framework that implements it at design level.
MR7.3.3-3	*Each shared property of a software framework domain model shall be translated into an assume-guarantee contract on one or a set of functional components defined in the UML2 design model of the framework.*

7.3.4 Design Methodology – System Family

The basic methodology for the design of the non-functional part of the CORDET Generic Architecture is the RCM Methodology [RD31, RD36 and RD37]. The UML2 meta-model that must be the output of the design process for the CORDET System Family (see requirement MR7.3-2) will therefore be defined as a modification of the existing RCM meta-model.

The RCM Methodology was defined with a purpose more general than the design of the reusable software assets of a product family. It can in particular be applied to the design of single applications as well as to the design of reusable building blocks. One implication of this greater generality of applicability of the RCM Methodology is that some of its aspects are not relevant to the design of the CORDET System Family.

The modifications to the RCM meta-model to be undertaken in the CORDET Project will therefore be aimed at adapting the meta-model to the specific needs of the CORDET Project. The exact kind of modifications will be decided during the design phase, however, it is may be worth mentioning that there are three potential simplifications to the current RCM Methodology that may be considered for CORDET[4].

The first simplification concerns the effective elimination of the transformation from AP- to VM-Level containers[5]. This is a model-to-model transformation that transforms a representation of the target system expressed in terms of AP-Level containers into a semantically equivalent representation expressed in terms of VM-Level containers. The objective of this transformation is to identify non-terminal entities among the AP-Level Containers and to break them up into their composing terminal VM-Level Container entities.

The decomposition from non-terminal to terminal entities makes sense when performing single-application design. In that case, the initial expression of the design in terms of high-level AP-Level Containers is helpful as a means to master complexity. A reuse-based approach, however, necessarily takes a bottom-up approach where a system is built by combining pre-defined building blocks. These pre-defined building blocks are monolithic and cannot be further decomposed.

It should also be stressed that the RCM composition principles are based on relationships of containment between components. The design approach taken in the CORDET Project is instead object-oriented (this is a consequence of the use of the FW Methodology) where the basic relationship among design entities are based on inheritance hierarchies. Complexity in this approach is mastered by introducing high-level abstract interfaces rather than by defining high-level containers. It follows that the use of high-level AP-Level Containers as a way of mastering complexity is no longer needed (and, in fact, becomes an obstacle to the definition of inheritance hierarchies).

Such a bottom-up object-oriented approach is of course compatible with the RCM Methodology only if the pre-defined building blocks are VM-Level containers. In this case, no

[4] The discussion in the remainder of this section assumes the reader to be familiar with the RCM Methodology.

[5] More precisely, the transformation is still applied (since it is an essential part of the RCM Methodology) but it becomes trivial since the AP-Level Containers are identical to VM-Level Containers (recall that a VM-Level Container is also an AP-Level Container).

transformation from AP-Level to VM-Level containers is required since the design is already expressed in terms of terminal VM-Level Containers.

In this respect, it is noteworthy that the design entities built using the FW Methodology can indeed be wrapped in VM-Level Containers.

From the a methodological point of view, the requirement that ensues from the previous discussion is that the design of the generic architecture should be expressed in terms of VM-Level Containers.

Note that this requirement only applies to design activities performed *at family level*. It is clear that, when instantiating the generic architecture to build a specific system, there is no preclusion to using a top-down design approach where decomposition from AP-Level to VM-Level containers plays a crucial role. The problem of how the (terminal) building blocks provided by the family can be mixed with the (non-terminal) building blocks defined at application level concerns the family instantiation process and goes beyond the scope of the CORDET Project.

The second simplification proposed for the RCM Methodology is a consequence of the first one. In the RCM Methodology, operations must be characterized by the state which they may access. This information is used when partitioning a non-terminal component into lower-level terminal components.

As discussed above, this type of decomposition is unnecessary in the case of the components defined in the CORDET design model. Hence, there is no need to characterize operations with state information.

In the RCM Methodology, an AP-Level Container is characterized by the interfaces it requires as well as by those it implements. In the case of an object-oriented application (and the use of the FW Methodology implies an object-oriented design for the CORDET Generic Architecture), the information about the required interfaces is already implied in the description of the interfaces that the component implements. A required interface must be an interface that is passed to a component as an argument of one of its operations (typically a setter operation). Thus, one can derive the set of required interfaces of a component simply by inspecting its provided interfaces.

Hence, in the third and last simplification proposed for the application of the RCM Methodology to the CORDET Project, the explicit definition of the required interfaces for the framework components is dropped.

The key element of the RCM Methodology is the RCM Meta-Model. This meta-model captures and enforces the design constraints specified by the RCM Methodology. In the CORDET Project, a modified version of this meta-model will be defined that captures and enforces a modified RCM Methodology. Conceptually, this modified version of the RCM meta-model will constitute the design model of the CORDET System Family (see also requirement MR7.3-2). In practice, the simplified meta-model may also be implemented as a UML2 profile or as a restriction on the existing RCM meta-model.

MR7.3.4-1	*The CORDET System Family shall be designed in accordance with a modified version of the RCM Methodology.*
MR7.3.4-2	*The UML2 meta-model that will constitute the output of the design activity for the CORDET System Family shall be obtained as a modification of the existing RCM Meta-Model.*

7.3.5 Merging the Functional and Non-Functional Designs

The CORDET Methodology is based on a split between the handling of non-functional concerns (design of the system family) and the handling of functional concerns (design of the software frameworks). However, it is clear that the two halves of the design of the CORDET Generic Architecture must eventually be merged when the architecture is instantiated. This is done during the application development process which is outside the scope of the CORDET Project.

Although the merging of the functional and non-functional halves of the generic architecture is not done at family level, the family level activities must be constrained to ensure that this merge is possible when the generic architecture is instantiated.

In this respect, it should be noted that the FW and RCM Methodologies are designed to be *independent but compatible*. This means that they can be used independently of each other when designing the functional and non-functional parts of the generic architecture while at the same guaranteeing that the design items that result from their application are compatible with each other. More information on the compatibility of the two methodologies can be found in [RD27].

In practice, the merge of the functional and non-functional design is done by embedding the functional components obtained by instantiating the software frameworks into the RCM containers. Thus, the requirements that the two halves – functional and non-functional – of the CORDET Generic Architecture can be translated into a requirement that *embedding rules* be stated that define how the functional components may be embedded within the non-functional containers during the architecture instantiation process.

The term *functional component* is used to designate a component obtained as an instantiation of a framework class, or as an instantiation of a class derived from a framework class, or as an instantiation of a class built to implement one or more framework interfaces.

The embedding rules must specify, for each functional component:

1. Either the kind of VM-Level Containers (sporadic, cyclic, protected, or passive) within which the component can be embedded, or
2. How a VM-Level Container can be built that can host the functional component.

Case (1) arises when the structure of the functional component is such that it can be directly embedded in a VM-Level Container. Note that, in view of requirement MR7.3.4-2, only embedding within VM-Level Containers is considered and not within the more general AP-Level Containers.

For software frameworks designed in accordance with the FW Methodology, case (1) applies to all components instantiated from framework classes that do not define any nominal operations or that only implement framework classes that do not define any nominal operations. In general, such components can be embedded without changes within either protected or passive RCM containers.

Case (2) may instead arise for functional components that must be embedded within cyclic or sporadic containers. The constraints that apply to the kind of operations that such containers may define will often make it impossible for them to host a functional component. As discussed in the previous section, decomposition of the component into lower-level components is not an option since the functional components are reusable entities that cannot be further split.

Compatibility with the RCM containers must instead be achieved by defining additional components that mediate the interaction between the RCM container and the functional component. The embedding rules should specify how these additional components can be built.

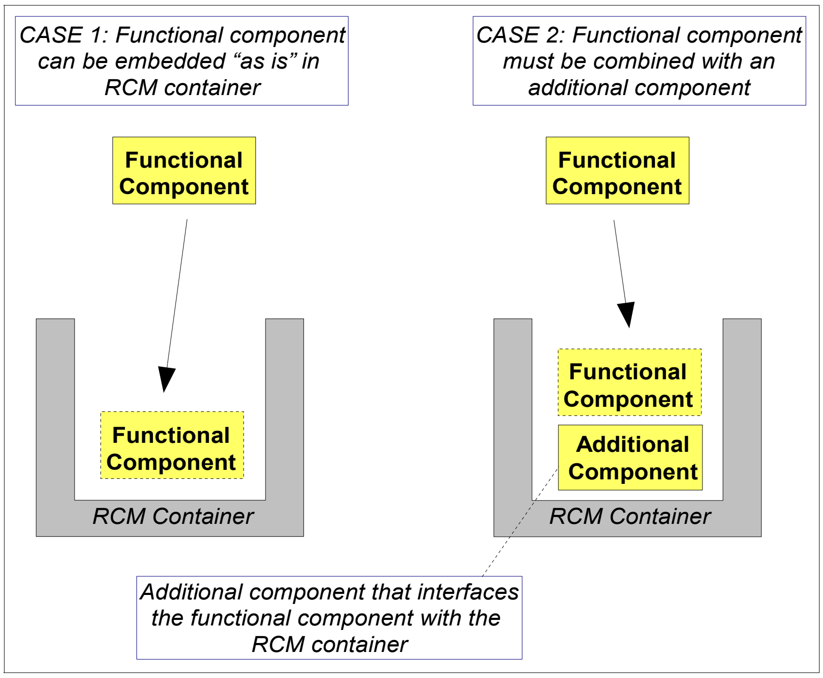
CASE 1: Functional component can be embedded "as is" in RCM container
CASE 2: Functional component must be combined with an additional component
Functional Component
Functional Component
Functional Component
RCM Container
Functional Component
Additional Component
RCM Container
Additional component that interfaces the functional component with the RCM container

Fig. 7.3.5-1: Embedding Rules for Functional Containers

MR7.3.5-1	*A CORDET Software Framework shall define the* embedding rules *for each functional component that may be instantiated from the framework.*

7.3.6 Iterative Definition of Design Model

As already mentioned in section 6.1, iteration is possible in the execution of the CORDET Methodology and its activities.

Some iteration in the definition of the design model is obviously desirable. In view of the limited time available for the project, only two iteration cycles are foreseen for the CORDET Domain Model. The end of the first iteration cycle should be marked by a review where the project stakeholders can submit their concerns and their suggestions for improvements.

MR7.3.6-1	*The definition of the design model shall be done in two iterations.*
MR7.3.6-2	*At the end of the first iteration, a review shall be performed where selected CORDET stakeholders shall be asked to provide comments.*

7.4 Methodological Requirements – Domain Implementation

This section presents the methodological requirements for the domain implementation activity.

The only methodological issue for the domain implementation phase is whether the separation between functional and non-functional aspects should be maintained down to code level (as is currently done in the ASSERT demonstrator), or whether instead the two aspects should be merged at code level. The latter approach would allow a higher optimization of execution and memory footprint performance but it is not really compatible with the use of the Ada language.

The first approach is selected for CORDET both because of the SSERT heritage and to keep the door open to the use of the Ada language.

MR7.4-1	*In the domain implementation phase, two generators will be developed to process, respectively, the functional and non-functional models of the Generic Architecture.*

8 THE CORDET TOOL CHAIN

This section defines the tools that are baselined for use in the CORDET Methodology. Section 8.1 discusses the main tool selection criteria. The following three sections define the tools for each of the three activities of the CORDET Methodology. The last section defines the tools baselined for other tasks not specifically related to any of the three CORDET activities.

8.1 Tool Selection Criteria

Two main criteria have informed the choice of tools for the CORDET Project.

The first selection criteria is the strong preference for tools that are publicly available under *free and open software licences*. This restriction is intended to avoid high licence costs and dependence on tool vendors, and to allow freedom to modify the tools to make them better suited to the needs of the CORDET Methodology.

The second selection criteria is a restriction to tools that are usable at the time the selection is made (namely at the time this document is written). In other words, the situation must be avoided where a tool is chosen based not on what it can offer and do at the time the selection is made but based on *expectations* of what it will be able to do and offer at the time it will be needed. This restriction is intended to minimize the risk of selecting a tool that will not be ready when it is not needed.

8.2 Tool Selection – Domain Analysis Phase

The domain model is entirely built using natural language with exception of the feature model. Thus, the only tool required for the domain analysis phase is a feature modelling tool. The selected tool is XFeature[6]. This tool has the following advantages:

- It is available under a free and open software licence,
- It has been successfully used in ASSERT and has proven itself adequate to its task,
- It was partly developed by P&P Software GmbH (under ESA contract 18499/04/NL/LvH) and it is therefore well known to one of the CORDET partners. This will ease maintenance.

The 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, an application meta-model generator, a family display model, and an application display model generator. In th CORDET Project, it is proposed to use XFeature in the so-called "FD Configuration". This configuration was defined in the ASSERT project to domain analysis tasks and it is regarded as adequate for the needs of CORDET. This configuration is documented in [RD33].

8.3 Tool Selection – Domain Design Phase

Two basic tools are required for the domain design phase to support, respectively, the FW Methodology (for the functional design model, see section 7.3.3) and the simplified RCM methodology (for the non-functional design model, see section 7.3.4).

The definition of the functional design model in accordance to the FW Methodology will be done using the TOPCASED Tool[7]. This is a generic UML2 design tool. In order to support the FW Profile (which in turns enforces the FW Methodology), the tool will be customized with the FW Profile Eclipse Plug-In developed at ETH in the ASSERT Project (see [RD27]).

[6] The XFeature tool can be downloaded from: http://www.pnp-software.com/XFeature/

[7] The TOPCASED tool suite can be downloaded from: http://www.topcased.org/

The definition of the non-functional design model in accordance to the RCM Methodology will also be done using the TOPCASED Tool but the tool will be customized with the RCM-specific meta-model defined in the system family domain design phase.

8.4 Tool Selection – Domain Implementation Phase

Since the design models will be implemented as UML2 models, the simplest choice for the code generators is to use MOF Scripts[8]. This approach has already been successfully applied in the ASSERT Project.

8.5 Other Tools

The following tools will also be used to support the CORDET Methodology as a whole:

- The Mantis Bug Tracking Tool[9] will be used to collect and keep track of comments to the generic architecture design and implementation.
- The SVN Repository Tool[10] repository will be used to build and maintain a repository for the design artefacts of the CORDET Generic Architecture.

All the above tools are well-known, well-proven, and available under free and open software licences.

[8] MOFScript is an Eclipse project and its home page is at: http://www.eclipse.org/gmt/mofscript/

[9] The Mantis tool can be downloaded from: http://www.mantisbt.org/

[10] The SVN Eclipse plug-in can be downloaded from: http://subversion.tigris.org/

9 SUMMARY TABLE

The table below summarizes the modelling approach and the support tools that have been proposed in this technical note and that constitute the CORDET Methodology

Phase	Modelling Approach	Support Tool
Domain Analysis	Domain Model broadly follows FODA and consists of: (a) Domain Dictionary in natural language (b) Shared Properties in natural language (c) Factors of Variation in natural language (d) Feature Model to describe variability	No special tools required for (a), (b), and (c). XFeature with "FD Configuration" for (d)
Domain Design	Design Model for functional part (application layer) consists of UML2 models compliant with FW Profile Design Model for non-functional part (middleware layer) consists of UML2 meta-model derived from RCM Meta-Model.	Topcased with FW Prodile plug-in for functional part. Topcased with modified RCM meta-model for non-functional part.
Domain Implementation	Code Generator to transform design models into source code.	MOF Scripts