

DOMAIN ENGINEERING METHODOLOGIES SURVEY

CORDET

Internal Code: GMVSA 20580/07
Versión: Issue 1
Date: 26/07/2007

THIS PAGE IS INTENTIONALLY LEFT IN BLANK

Prepared by: Elena Alaña
Ana Isabel Rodríguez

Approved by: Ana Isabel Rodríguez
Project Manager

Authorized by: Ana Isabel Rodríguez
Project Manager

DOCUMENT STATUS SHEET

Version	Date	Pages	Changes
Issue 1 Draft A	29/05/2007	29	First draft version
Issue 1 Draft B	13/06/2007	29	Second draft version
Issue 1 Draft C	21/06/2007	30	Third draft version.
Issue 1 Draft D	03/07/2007	30	Fourth draft version.
Issue 1	26/07/2007	38	First issue of the report. Updated according to MoM "CORDET-MoM-DKMR-06072007", [RD.64].

TABLE OF CONTENTS

1. INTRODUCTION	7
1.1. PURPOSE AND SCOPE	7
1.2. APPLICABLE DOCUMENTS	7
1.3. REFERENCE DOCUMENTS	7
2. METHODOLOGIES	11
2.1. DOMAIN ENGINEERING PROCESS	11
2.1.1. DOMAIN ANALYSIS	11
2.1.2. DOMAIN DESIGN	12
2.1.3. DOMAIN IMPLEMENTATION	12
2.2. DOMAIN ENGINEERING METHODS	12
2.2.1. DOMAIN ENGINEERING METHODS BASED ON THE ANALYSIS OF THE DOMAIN	12
2.2.2. DOMAIN ENGINEERING METHODS BASED ON THE PRODUCT LINE	17
2.2.3. DOMAIN ENGINEERING AND OBJECT-ORIENTED ANALYSIS AND DESIGN METHODS	18
2.2.4. SODA (STRATEGIC OPTIONS DESIGN AND ASSESSMENT)	19
2.3. ARCHITECTURAL ANALYSIS METHODS: TRANSITION FROM DOMAIN MODELING TO DOMAIN ARCHITECTURE DEFINITION	20
2.4. DOMAIN ENGINEERING NOTATIONS AND TOOLS	20
2.4.1. NOTATION AND TOOL SUPPORT FOR DOMAIN ANALYSIS	20
2.4.2. NOTATION AND TOOL SUPPORT FOR DOMAIN DESIGN	21
2.4.3. NOTATION AND TOOL SUPPORT FOR DOMAIN IMPLEMENTATION	22
3. CONCLUSIONS	24
3.1. OVERVIEW OF DOMAIN ENGINEERING METHODOLOGIES	24
3.2. DOMAIN ENGINEERING PHASE	26
3.2.1. DOMAIN ANALYSIS	26
3.2.2. DOMAIN DESIGN	29
3.2.3. DOMAIN IMPLEMENTATION	30
3.3. SUMMARY AND RECOMMENDATIONS	31
ANNEX A. MAP OF GMV APPROACH AND ISO 12207 DOMAIN ENGINEERING REQUIREMENTS	33
ANNEX B. GMV, INTECS AND P&P NOTATION AND TOOLS COMPARATIVE	36

LIST OF TABLES AND FIGURES

Table 1-1- Applicable Documents	7
Table 1-2- Reference Documents.....	10
Table 3-1- Summary of domain engineering methodologies	25
Table 3-2- Summary of Models, Notations and tools	31
Table A-1. Map of GMV approach and ISO 12207 requirements	35

1. INTRODUCTION

1.1. PURPOSE AND SCOPE

This report is written as part of the ESA study on Component Oriented Development Techniques (CORDET), AO/1-5237/06/NL/JD ([AD.1] and [AD.2]).

The document is the output of the task WP-202 "DOMENG Inputs" with the major constituent of the identification of methodologies for the Domain Engineering Process ([AD.3]).

As stated in the Statement of Work [AD.1], the scope of the task "Methodologies selection" is to identify the methodologies for all the domain engineering process activities and to identify and select the languages to be utilized to create generic models needed to describe model architecture for space systems.

This report supports the selection of the domain analysis and domain design methodologies collecting the Domain Engineering methodology survey and identifying standards, methods, technologies and tools.

The report summarizes the different models, notations and tools needed to cover the whole domain engineering process and also provides an analysis of the compliance of the approach proposed by GMV and the ISO 12207 [AD.4].

1.2. APPLICABLE DOCUMENTS

The following documents, of the exact issue shown, form part of this document to the extent specified herein. Applicable documents are those referenced in the Contract or approved by the Approval Authority. They are referenced in this document in the form [AD.X]:

Ref.	Title	Code / Version
[AD.1]	Component Oriented Development Techniques: Statement of work	EME-012 – issue 1.1 – 28 July 2006
[AD.2]	Invitation to tender number AO/1-5237/06/NL/JB – Component Oriented Development Techniques	ESA /IPC (2001) 11 – Item no. 01.1EM.06
[AD.3]	Component Oriented Development Techniques. Invitation To Tender AO/1-5237/06/NL/JB: Part 1 Technical Proposal Part 2 Management Proposal Part 3 Financial and Contractual Proposal	ASP-06-EL/PE/S-157, Issue 01, October 16 th , 2006.
[AD.4]	ISO AMD 1 and AMD 2 2004, Corrigenda –Information of ISO/IEC 12207:1995 Software Life cycle Processes. ISO/IEC 12207 – limited to the domain engineering process	ISO/IEC 12207 :1995

Table 1-1- Applicable Documents

1.3. REFERENCE DOCUMENTS

The following documents, although not part of this document, amplify or clarify its contents. Reference documents are those not applicable and referenced within this document. They are referenced in this document in the form [RD.X]:

Ref.	Title	Code / Version
[RD.1]	ECSS-E-40 Part 1B – Space engineering – Software – Part 1: Principles and requirements	ECSS-E-40 Part 1B – 28 November 2003
[RD.2]	ECSS-E-40 Part 2B – Space engineering – Software – Part 2: Document requirements definitions (DRDs)	ECSS-E-40 Part 2B –31 March 2005
[RD.3]	ECSS-Q-80 B, Space Product Assurance – Software Product Assurance	ECSS-Q-80 – 10 October 2003
[RD.4]	Avionics Architectural Description Language (AADL) http://www.aadl.info/	-

Ref.	Title	Code / Version
[RD.5]	Domain Engineering, Software Engineering Institute. Carnegie Mellon http://www.sei.cmu.edu/domain-engineering/index.html	-
[RD.6]	Feature Oriented Domain Analysis (FODA) http://www.sei.cmu.edu/domain-engineering/FODA.html	-
[RD.7]	Unified Modelling Language (UML) http://www.omg.org/uml and http://www.uml.org	
[RD.8]	IEEE Standard for Information Technology – Software Reuse – Data Model for Reuse Library Interoperability: Basic Interoperability Data Model (BIDM)	IEEE Std 1420.1-1995(R2002)
[RD.9]	IEEE 1420.1a Supplement to IEEE Standard for Information Technology – Software Reuse –Data Model for Reuse Library Interoperability: Asset Certification Framework.	IEEE 1420.1a-1996(R2002)
[RD.10]	IEEE 1420.1b IEEE Trial-Use Supplement to IEEE Standard for Information Technology –Software Reuse – Data Model for Reuse Library Interoperability: Intellectual Property Rights Framework	IEEE 1420.1b-1999(R2002)
[RD.11]	Hard Real Time-UML (HRT-UML) http://bssc.esa.int/design.htm	
[RD.12]	Model driven Architecture (MDA) http://www.omg.org/mda/	
[RD.13]	Feature-Oriented Domain Analysis (FODA) Feasibility Study. CMU/SEI-90-TR-021	
[RD.14]	Domain Analysis and Framework-based Software Development. Software Production Engineering Lab, Department of Electrical and Computer Engineering, and Thera S.p.A.	
[RD.15]	ECLIPSE http://www.eclipse.org	
[RD.16]	IDEAL http://www.sei.cmu.edu/ideal/	
[RD.17]	OCA Mapping a Domain Model and Architecture to a Generic Design. Technical Report. Software Engineering Institute. CMU/SEI-94-TR-8 ESC-TR-94-008	
[RD.18]	Informal Technical Report for Software Technology for adaptable, reliable systems. Organization Domain Modeling (ODM) guidebook. Version 2.0.	
[RD.19]	FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Pohang University of Science and Technology.	
[RD.20]	Integrating Feature Modeling with the RSEB. Proceedings of Fifth International Conference on Software Reuse. Victoria, B.C.	
[RD.21]	The software architecture process. University of Houston-Clear Lake.	
[RD.22]	DARE: A Domain Analysis and Reuse Environment. Phase I Final Report. Reuse, Inc.	
[RD.23]	FAST product-line architecture process. Tampere University of Technology.	
[RD.24]	PuLSE: A Methodology to Develop Software Product Lines. Fraunhofer Institute for Experimental Software Engineering.	
[RD.25]	Working with objects. The OOram Software Engineering Method. Trygve Reenskaug.	
[RD.26]	Software Reuse: Issues and Experiences. Rubén Priet-Días. Reuse, Inc.	
[RD.27]	SysML (System Modeling Language) http://www.sysml.org	
[RD.28]	HRT-UML toolset http://www.ellidiss.com/hrtuml.shtml	
[RD.29]	Eclipse UML2 http://www.eclipse.org/modeling/mdt/?project=uml2	
[RD.30]	XFeature http://www.pnp-software.com/XFeature	

Ref.	Title	Code / Version
[RD.31]	TOPCASED tool http://www.topcased.org	
[RD.32]	REQUILINE tool- http://www-lufqi3.informatik.rwth-aachen.de/TOOLS/requiline/index.php	
[RD.33]	MDA Guide. Version 1.0.1 http://www.omg.org/docs/omg/03-06-01.pdf	
[RD.34]	STOOD http://www.ellidiss.com/stood_users.shtml	
[RD.35]	CORBA http://www.omg.org/technology/documents/spec_catalog.htm and http://www.corba.org/	
[RD.36]	XML Metadata Interchange http://www.omg.org/technology/documents/formal/xmi.htm and http://xml.coverpages.org/xmi.html	
[RD.37]	EUREKA SI! 2023 – ITEA 04006 project Model-Based Approach for Real-Time Embedded Systems development (MARTES) – Specification of the Model Driven Engineering Process	ITEA 04006 V 1.0 22/09/2006
[RD.38]	Borland Together – Visual Modeling for Software Architecture Design http://www.borland.com/us/products/together/index.html	
[RD.39]	Object Management Group (OMG), Software Process Engineering Metamodel Specification, http://www.omg.org/docs/formal/05-01-06.pdf	SPEM v1.1- January 2005
[RD.40]	CHEDDAR – Open Source– University of Brest, advanced scheduling analysis toolset http://beru.univ-brest.fr/~singhoff/cheddar/	
[RD.41]	Open Source AADL Tool Environment (OSATE) http://www.sei.cmu.edu/tools-methods/system.html	
[RD.42]	SADT-IDEF0 (Structured Analysis and Design Technique) http://www.idef.com/idef0.html	
[RD.43]	AADL (Architecture Analysis and Design Language) http://www.axlog.fr/aadl/aadl_en.html	
[RD.44]	Borland Together http://www.eclipseplugincentral.com/Web_Links-index-req-viewlink-cid-50.html	
[RD.45]	CORDET Report: Domain Engineering Methodologies, Languages and Tools	COReT/R02 Issue: 10 Date: 2007-05-28
[RD.46]	Methodology and approach for information gathering across the network – CORDET - SWC Technical Note 1	CORDET/R1 Is. 0.4 Date: 2007-06-21
[RD.47]	MDG Technology (Sparx Systems) http://www.sparxsystems.com.au/ea.htm	
[RD.48]	Rhapsody (Telelogic) http://modeling.telelogic.com/	
[RD.49]	Tau G2 (Telelogic) http://www.telelogic.com/corp/products/tau/q2/index.cfm	
[RD.50]	SysML Toolkit (EmbeddedPlus) http://www.telelogic.com/corp/products/tau/q2/index.cfm	
[RD.51]	MagicDraw http://www.magicdraw.com/	
[RD.52]	Open Architecture ware Eclipse plug-in http://eclipse-plugins.2y.net/eclipse/plugin_details.jsp;jsessionid=2495DE80D44BA3138879EC625800683E?id=782	
[RD.53]	UMT (UML Model Transformation tool) http://umt-qvt.sourceforge.net/	

Ref.	Title	Code / Version
[RD.54]	AndroMDA http://www.andromda.org/	
[RD.55]	Motion Modeling http://motionmodeling.sourceforge.net/	
[RD.56]	MTL Engine http://modelware.inria.fr/	
[RD.57]	ModFact http://modfact.lip6.fr/	
[RD.58]	UML 2 http://sparxsystems.com.au/resources/uml2_tutorial/index.html	
[RD.59]	Strategic Option Design and Assessment (SODA) ftp://ftp.estec.esa.nl/pub/wm/wme/cordet/Ionita.ppt	2005
[RD.60]	Global Business Network http://www.gbn.com/	
[RD.61]	AUTOSAR http://www.autosar.org/find02_ns6.php	
[RD.62]	ARINC Specification 653-2 Avionics Application Software Standard Interface. Part 1- Required Services.	Date: 2005-12-01
[RD.63]	EGOS http://www.egos.esa.int/portal/egos-web/others/Events/Workshop/egos-2005.html	
[RD.64]	Minutes of meeting "Domain Knowledge and Methodology Review"	CORDET-MoM-DKMR-06072007 Version2

Table 1-2- Reference Documents

2. METHODOLOGIES

This section presents an overview of several methodologies candidates to implement the whole Domain Engineering Process: domain analysis, domain design and domain implementation.

The reference documents in section 1.3, amplify or clarify its contents.

2.1. DOMAIN ENGINEERING PROCESS

Domain engineering [RD.5] is the process of analysis, specification, and implementation of software assets in a domain which is used in the development of multiple software products and as such the requirements and recommendations of ISO guides [AD.4] its execution providing a rigours and structured process to organize, correlate and conduct the activities.

The Domain Engineering Process is divided into three phases: Domain Analysis, Domain Design and Domain Implementation. This report is focused on the first two phases in accordance to the work to be carried out as defined in the SOW [AD.1]:

- Perform a Domain Engineering Analysis with the purpose to identify and establish commonalities, through a variation analysis of the space systems. Investigation on the current system and software families has to be carried out in order to identify a common application domain framework, as the basis for domain engineering design activities for future spacecrafts (platform and payload). The activity shall be performed using a proposed domain engineering methodology.
- Perform a Domain Engineering Design with the purpose to propose generic software architecture (software framework) to be reused for design and development of future spacecrafts. This shall be carried out with an appropriate methodology.

2.1.1. DOMAIN ANALYSIS

Domain Analysis is the activity that discovers and formally describes the commonalities and variability within a domain. The domain engineer captures and organizes this information in a set of domain models with the end of making it reusable when creating new systems. The output of domain analysis is a domain model: an explicit representation of knowledge about the domain.

The Domain model will consists of:

- Domain dictionary (domain lexicon)
- Context models (using e.g. diagrams, formalisms,) and
- Feature models

The Domain dictionary provides and defines the terms concerning the domain. Its purpose is to make communication among developers and other stakeholders easier and more precise.

The Context models specifies the boundaries of the domain. The model considers both the commonalities and variabilities of the application in the domain. The analysis will include:

- Description of the domain and its relation to other domains and a record of important decisions and alternatives.
- Commonalities: A structured list of assumptions that are true for every member of the family
- Variabilities: A structured list of assumptions about how family members differ
- Parameters of variation: A list of parameters that refine the variabilities.

The Feature model is a hierarchical decomposition of features. Feature models that also tell which combinations of features are meaningful can depict features. Feature models provide notations for different kinds of features such as the FODA-like features.

2.1.2. DOMAIN DESIGN

The Domain Design takes a Domain Model as input and applies a Partitioning Strategy from Architecture as a control model to produce a Generic Design. According to the domain models, it should also be selected which components or items (such as requirements) are provided in the core architecture and which items are implemented as variations in individual applications.

The partitioning strategy defines the elements (e.g. subsystems, objects, data types etc.) and how the domain features are allocated to them. Selection of a strategy in part depends on the major factors of change identified in the domain models.

2.1.3. DOMAIN IMPLEMENTATION

The Domain Implementation takes as inputs the design models and the generic architectures designed to identify and create reusable assets. The main outputs are these reusable assets and also application generators and domain languages.

2.2. DOMAIN ENGINEERING METHODS

Several methods have been developed with the end of reusing software. They mainly vary in how they identify the domain effectively, and make use of available domain, architecture and systems expertise. This section describes some of the most important software reuse processes developed up to day. These processes are divided into three groups.

- Methods based on the goal of analyzing and modeling the domain to achieve reuse from similarities in a domain. These methods differ regard to the level of formality in the method and products, or the information capture techniques. Some of these methods are ODM, FODA, FORM, RSEB and DSSA.
- Product Line Processes. Product line comprises concrete products or applications to be developed. These domain engineering methods have been extended or connected to other methods to cover the whole product-line engineering process. Some methods using this model are FAST or PuLSE.
- Domain engineering and OOA/D methods. OOA/D methods supporting domain engineering are OOram or JODA.

It is also important to consider that a great number of approaches to domain engineering only specify what the domain model should include, but they do not specify any processes for analyzing and designing process.

2.2.1. DOMAIN ENGINEERING METHODS BASED ON THE ANALYSIS OF THE DOMAIN

Methods based on domain analysis and design; denote set of systems or functional areas, which exhibit similar functionality. Some of these methods are described in this document: ODM, FODA, FORM, RSEB and DSSA.

2.2.1.1. ODM (Organization Domain Modeling)

Organization Domain Modeling [RD.18] has been developed to provide an overall framework for a domain engineering life-cycle. Its main aim is the transformations of artifacts from multiple inherit systems into assets, which may be used in multiple applications. ODM is configurable, and it may be integrated with other software engineering technologies.

ODM divides the domain engineering process into three different phases:

- Plan Domain. Firstly, it is necessary to discover and validate an optimum set of objectives taking into account the interest of project stakeholders. The stakeholders' goals are reconsidered as critical points throughout the domain modeling life cycle. According to these goals and the characteristics of the domains of interest, a domain has to be selected for the target project. This domain is called "domain of focus". Finally, the domain is defined and it is also specified what is in and out of the domain, this is bound the domain.

- **Model Domain.** This phase is concerned with gathering and documenting relevant domain information. Once the information is integrated and the most relevant system features have been identified, the domain has to be described. A domain model is produced based on the domain definition produced during the plan domain phase. This model represents the commonality and variability within the domain and tries to formalize the space of possible alternatives for individual systems in the domain. This model includes feature profiles that map specific features or feature combinations onto the domain settings to which they may be applied. As well as the domain model, a domain dossier, domain lexicon, and a set of concept and feature models are generated.
- **Engineer Asset Base.** The objective of this phase is to scope, architect and implement an asset base that addresses the needs of multiple systems.

The scope sub-phase is in charge of correlating features with customers and selecting which will be implemented. Architect sub-phase determines the external and internal architecture constraints and the definition of architecture. Finally, the implementation sub-phase plans the implementation and implements the assets and infrastructure.

This method is useful for a wide range of organizations and domains and it may be integrated with a variety of software engineering processes, methods and implementation technologies. However, it does not have support for creating domain-specific languages and application generators. ODM is most successful when it is used to support domain engineering projects for domains, which are mature, reasonably stable and economically viable.

2.2.1.2. FODA (Feature Oriented Domain Analysis)

FODA ([RD.6] and [RD.13]) has been developed at Software Engineering Institute. It is based on the identification, analysis and documentation of the main features of software systems in a domain, not only the common aspects but also the differences with other domains. The final result will be generic domain products widely applicable within a domain. These products are based on abstraction and refinements with parameterization.

- Abstractions are used to create a domain product from a determinate application in a specific domain. This process is done applying aggregation and generalization of features with the end of obtaining the common assumptions of the application.
- Refinements are used to adapt domain products through some factors that specify the abstraction. They represent the differences among applications. The refinements can use parameters to specify the context.

The domain analysis process is divided into three phases:

- **Context analysis.** The objective is to establish the bounds of the domain, the relation with other domains, and the scope of the analysis. It is important to identify the features that make each application unique and those that can be abstracted away. So, a requirement capturing process is needed to collect information for various activities gathered from various sources.
 - The final results of this analysis are documented with two different kinds of diagrams, which define the boundary of the domain: structure and data-flows diagrams. The structure one is an informal block diagram in which the domain is placed relative to higher, lower, and peer-level domains. The domains that interface with the domain are also identified. The data-flow diagram shows the communication between the target domain and other domains and entities. The variability of the domain boundary must be indicated in the diagram.
- **Domain modeling.** The context model generated previously is analyzed generating domain models. These models include:
 - **Information model.** It consists of the domain's entities and the relations between them. There are two basic relationships: "is-a"/"is-set-of" and "consists-of". With both kind of relationships the content of a domain is described and organized.
 - **Feature model.** It captures the common features and differences of the applications in a domain and their relationships. The feature model contains four elements: feature diagram, feature definitions, composition rules and rationale for features.

The feature diagram divides hierarchically the features into sub-features, identifying each one as mandatory (every system that delivers the feature above must include that feature), alternative (every system must have one out of multiple optional features) or optional (a system may have it or not).

Composition rules belong to one of these two groups: one feature requires another feature or one feature is mutually exclusive of another.

- Operational model. It describes the control and data flow in the application domain, the relationships among objects in the information model and the feature model. They also include definitions of software functions.
- Generation of a domain terminology dictionary, which contains all the standard vocabulary of domain experts.
- Architecture Modeling. Domain models are used to create a high-level architecture model. The architecture is defined at various levels of abstraction in order to improve the reuse. It establishes the structure of implementations of software in the domain. This architecture model can be instantiated to develop individual applications.

The features are represented hierarchically which makes easy the identification and understanding of features. Nevertheless, this representation does not encourage component-based development, but it can be achieved using OCA [RD.17]. OCA provides an easy way to map the features of the feature models to objects, which integrate the different subsystems and systems of the architecture.

FODA does not have any specific process for requirements specification, verification and management.

2.2.1.3. FORM (Feature-Oriented Reuse Method)

FORM [RD.19] is a systematic method which extends FODA to design and implement phases. This extension is needed since features, which characterize each variant product in a domain and the code that implements them should be packaged, managed and reused as software modules. FORM provides user observable features and references software architectures of the domain that is being considered.

FORM domain engineering is divided into three phases: context analysis, feature modeling and architecture or component modeling.

During the context analysis phase the scope of the domain is identified, as well as the interaction with other domains. Then, FORM analyzes the commonality and variability among applications in that domain, representing the mandatory and alternative features that could be selected for different domain applications. This phase is called 'feature model' and identifies and analyzes aspects from four different perspectives:

- Capabilities. Features related to the performance that an application might possess.
- Operating environment. Features link to the environment in which the application is used.
- Domain technologies.
- Implementation techniques.

These relations among features are represented with a feature hierarchical diagram. The features may be mandatory, optional or alternative and the relationships among features may be 'composed-of', 'generalization/specialization' and 'implemented-by'. In addition to the feature diagram, composition rules and justifications for feature selection are added.

The feature model is used during the architecture modeling to define a set of reference architectures, which are organized in three hierarchical levels:

- Subsystem Model. It defines the system structure by grouping functions. Subsystems communicate via a coupled message queue (non-blocking) or via a tightly coupled message/reply mechanism (blocking).
- Process Model. It represents the dynamic behavior of each subsystem. The processes are tagged with the next categories:
 - Depending on the duration of activation: resident or transient.
 - Depending on whether they can be forked off: multiple or single.

- Module Model. Each module represents a set of relevant features and these modules also contain an abstract specification.

These modules are the bases of the generation of reusable components, which can be integrated during the development of new applications. It will be necessary to establish a mapping between the feature model and the architectural model.

2.2.1.4. FeatureRSEB (Feature Reuse-Driven Software Engineering Business)

Before the existence of FeatureRSEB, RSEB was developed. RSEB is a use-case driven systematic reuse process based on Unified Modeling Language (UML) notation. UML is used to specify applications systems, reusable component systems, and layered architectures. Variability between systems is expressed with variation points and attached variants. The domain engineering activities are separated into two groups:

- Application Family Engineering. Higher-level of abstraction to develop a conceptual model and a common layered architecture for the entire domain.
- Component System Engineering. Lower-level of abstraction to develop functional building blocks for the layered product platform.

This method is based on modeling variability but it includes neither domain analysis techniques nor a description of a systematic way to perform the asset development. Due to these drawbacks FeatureRSEB was developed.

FeatureRSEB [RD.20] combines FODA and ODM concepts but the feature models generated are simpler than those in original FODA, focusing on the feature-oriented parts for domain-engineering purposes. The architecture and reusable subsystems are described using use-case diagrams, and lately they are transformed into object models, which can be mapped from the use-case diagram.

FeatureRSEB includes domain analysis solving the limitations of RSEB. Domain analysis starts with a domain scoping and feature modeling. It consists on a division into component's systems and the generation of a high level use case model, which describes the architecture of the system and the context.

The next step consists of the identification of the commonality and variability of the elements. The domain entities and the interaction among them are represented in a use case and object model. The dynamic relations among the domain entities are represented in the sequence and interaction diagrams.

FODA and RSEB are compatible with each other and they have much in common. Both of them are model-driven methods providing several models, which correspond to different points of view of the domain.

2.2.1.5. DSSA (Domain-Specific Software Architectures)

DSSA approach was developed under the Advanced Research Project Agency's (ARPA) DSSA program. The program central role is the software architecture. Multiple variations of a DSSA approach were developed.

Domain Specific Software Architecture (DSSA) is architecture for a specific domain based on the commonalities and differences. It should be general enough to support several applications of the domain. It focuses on the process, which specifies how to provide the common and variable features and how to derive the final architecture for each application. It does not specify any formalism for capturing information.

DSSA domain analysis is comprised of:

- Domain analysis is used to capture and identify components and operations in a class of similar systems in a particular domain. Once the elements are identified, the relationships among them (inheritance, aggregations, etc) are defined and the dataflow and control flow among elements are captured. The result is a requirements document.
- Later, it is needed to identify constraints and software, hardware and performance requirements on the implementation.

Finally, the architecture is developed. The components are identified and the decision for mapping variabilities to changes is captured. For each component, the next parameters must be specified:

- Interface
- Behavior

- Entry/Exit conditions
- Constraints
- Configuration parameters

Finally, based on the architecture developed and all the information obtained, the reusable components are developed.

2.2.1.6. Sandwich method

This method is also known as "STARS Reuse Library Process Mode SRLPMI" [RD.26]. Sandwich Method specifies components that can be implemented and put into a library for future reuse. The domain models are produced in the form of a generic architecture or standard designs. The reuse is guaranteed because low level components act as building blocks for composing the architecture.

Domain analysis phase prepares domain information, classifies the domain entities, derives domain models and finally, expands and verifies models and classification. The whole phase generates a model, which includes the following information:

- Concepts to enable the specification of systems in the domain
- Plans describing how to map specifications into code.
- Rationales for the specification concepts, their relations, and the relations to the implementation plans.

Sandwich is based on two procedures with match between high level generic models and low level components using domain models as skeleton guides:

- Bottom-up activities for classification. Identification and classification of low level common functions and standardization of their interfaces. The products of this analysis are associated with the structures derived by top-down.
- Top-down activities for system analysis. The outcomes are generic architectures.

Several applications of Sandwich method have been done. Prieto-Díaz, the author of this method, usually mentions three important experiences in the following domains: command and control systems, flight simulators, and avionics displays. However, it has drawbacks like: there is no much information related to the whole process or it does not support the development of languages.

2.2.1.7. DARE (Domain Analysis and Reuse Environment)

DARE [RD.22] provides support environment for partially automating the activities of domain analysis. It focuses on knowledge acquisition and knowledge structuring activities.

DARE users produce a Domain Analysis Book that contains all domain analysis inputs, outputs and supporting material.

DARE supposes that the domain is already defined. Taking into account that the domain is defined, DARE process consist of four activities:

- Acquire domain knowledge. All the domain information is stored in a database. Users interact with the database through an interface to extract and analyze data about domains. Knowledge is acquired from existing systems, domain related documents and information of domain expert.
Existing systems, for instance code sources, will be selected manually and re-structure using reengineering tools.
Domain related documents are processed using sophisticated information retrieval procedures.
Knowledge from experts will be converted semi-automatically to text format through questionnaires. The main goal of the expert support is to help the analyst to postulate initial domain architecture.
The outcomes are domain analysis products such as system architectures, standard designs or reusable code components.

- Structure domain knowledge for commonality analysis. The structuring activities will be done automatically. Some structures are faceted classification, domain vocabulary, high level functional description; design rationale charts or systems code structures.
- Identify commonalities. The identification is based on clustering techniques, reverse engineering and code similarity detection tools.
- Generate domain models.

These four activities are iterative to refine domain models.

2.2.2. DOMAIN ENGINEERING METHODS BASED ON THE PRODUCT LINE

Domain engineering methods based on Software Product Line focuses on sharing features. This section focuses on methods linked to product lines and software families, this is group of products that share common features and meet the needs of a particular market area. There are few methodologies available to develop and deploy product lines beyond existing domain engineering approaches. This section presents two of these methodologies.

2.2.2.1. FAST (Family Oriented Abstraction, Specification and Translation)

FAST [RD.23] defines the Domain Engineering and the Application Engineering processes. Hence, it covers the whole product-line engineering process. David Weiss at Lucent Technologies Bell Laboratories developed it. At this laboratory FAST has been applied to over 25 domains reducing the development time and cost by 60 to 70% for new family members.

FAST is not applicable to every domain. It is only applicable to domains, which satisfy the next requirements:

- The domain is mature. This is, domains where expert knowledge already exists.
- The domain must be stable. To consider a domain stable it has to maintain the structure and basic concepts a long the time.
- Existence and availability of domain experts.

During the domain engineering, the software family is defined and an environment for producing family members is developed. The domain process is based on sharing common features, and the process is reduced to two phases: domain analysis and domain implementation. Some aspects related to domain design are considered in the domain analysis phase.

- Domain analysis, also called "Commonality Analysis". It begins with collecting and documenting the knowledge of a particular domain. It includes activities needed to understand the domain and what constitutes solutions in the domain. The end of this analysis process is to obtain a documented product family requirement whose structure provides a high-level view of the domain, and their content captures the domain expertise. This document is created through a series of moderated meetings with domain experts.

Later, a decision model is defined. It captures the set of requirements and engineering decisions that an application must resolve to describe and construct a product.

Finally, an Application Modeling Language (AML) is designed. Application engineers to specify new product family members will use it. All the decisions identified by the decision model must be expressible in the AML. In order to generate working applications either a compositional or a compiler may be used for the AML. On the one hand, when a compiler is used, a design is created where only the abstract modules and some parameters of variation are specified in the design. On the other hand, a compositional approach creates the architecture. Therefore, the output is an application modeling language design and a domain design.

- Domain implementation. It includes the development and refinement of the environment that meets the specification given by the domain mode. Generation and analysis tools may be created.

FAST does not recommend any specific technique to carry out this task. Once the domain is specified, the translation into individual products is carried out using a domain specification language, this way the translation can be done automatically.

2.2.2.2. PuLSE (Product Line Software Engineering)

PuLSE [RD.24] is a customized product line practice. It has three basis elements:

- **Deployment Phases.** Logical steps to describe the activities needed to define a family of products. These activities are classified as initialization, infrastructure construction, infrastructure usage, and evolution and management phase.

During the initialization, PuLSE is customized to fit the particular application. PuLSE infrastructure considers the construction of the infrastructure including the scope, model and architecture of the product line. The product line infrastructure is used to create individual products during the PuLSE infrastructure usage. Finally, the evolution and management activity evolve the infrastructure over the time and manage it.

- **Technical Components.** Technical knowledge required carrying out PuLSE activities. It is used during deployment phases.

For instance, PuLSE-DDSA develops a domain specific architecture based on the product-line model, PuLSE-ECO concentrates on economic scoping or PuLSE-EM concentrates on evolution and management.

- **Support Components.** Guidelines to solve any non-technical issues. Using these packages of information enables better adaptation, evolution, and deployment of the product line.

PuLSE does not recommend any tool or technique for each one of the activities.

2.2.3. DOMAIN ENGINEERING AND OBJECT-ORIENTED ANALYSIS AND DESIGN METHODS

This method combines object-oriented analysis and design (OOA/D) with domain engineering. Although OOA/D methods do not support reuse, there exist several applications, which try to solve this problem. This section presents some of these approaches.

Some methods described previously, such as FeatureRSEB, also follow an Object-Oriented analysis.

2.2.3.1. OORam

OORam [RD.25] is a generic method that provides a framework for creating a variety of methodologies. The whole development cycle is focused on interactions. This way improves reuse, traceability and the ability to cope with complexity.

The basic idea is that different methodologies are needed for different purposes. So, OORam method is generic. It is a frame of reference for a family of object-oriented methodologies. OORam defines a "role model" which collects objects together according to their common goal. To achieve this goal there exist three processes:

- **Model creation process.** Creation of the model.
- **System development process.** It covers the typical software life cycle: system user model, system requirement model, system design model and system implementation model.
- **Reusable asset building process.** Production of several closely related systems by configuring proven components.

The work processes on all three levels are iterative and the deliverables evolutionary.

2.2.3.2. JODA (Joint Integrated Avionics Working Group Object- Oriented Domain Analysis)

JODA uses OOA/D instead of functional methods to cover the domain analysis phase. Object-oriented analysis techniques are used to define the structure and capture requirements.

This domain analysis phase identifies what is reusable, how it can be structured, and how it can be reuse. It consists of three phases:

- Domain preparation. This activity is in charge of identifying and collecting source material, references and software artifacts linked to the domain. The result is a domain source material and support from domain experts; this is help from experienced people.
- Domain definition. It bounds the domain and the analysts use this domain to clarify what can be reusable. This phase generates a top-level subject diagrams, top-level whole-part diagrams, top-level generalization-specialization diagram, domain services, domain dependencies, domain glossary and textual description. Finally, the context for potential re-users is identified.
- Domain modeling. This last phase is the one which extends from OOA and it follows the next steps:
 - Definition of attributes and services. Identification of objects and their relationships.
 - Identification, definition and simulation of the domain scenarios.
 - Objects are abstracted and grouped to enable reuse.

These three steps are iterated until the final domain model is obtained. It must be accurate enough to define the domain.

This method has already been used by the JIAWG Reuse Subcommittee's domain analysis group to analyze an avionics domain (stores management).

2.2.3.3. SHERLOCK

Sherlock [RD.14] is a product line practice, which uses OOA for analysis. It uses case diagrams and class diagrams for modeling. Entities corresponding to the common requirements are represented using solid lines and those corresponding to the variable requirements are represented with dotted lines.

The domain framework is developed using five basic steps.

- Domain Definition.
- Domain Characterization.
- Domain Scoping.
- Domain Modeling.
- Domain Framework.

Sherlock provides tool support for managing each activity but there is no specific technique for requirements specification, verification and traceability.

2.2.4. SODA (STRATEGIC OPTIONS DESIGN AND ASSESSMENT)

SODA method [RD.59] provides an approach to design long-lived system architectures. This goal is achieved analyzing different strategic scenarios before developing final architectures. This way, SODA tries to anticipate to future requirements or technology changes.

SODA method starts proposing several possible architectural options. These options are flexible architectures which also cover future needs. Then, the feasibility of these options is evaluated. The final result should be a flexible architecture which can be adapted to different changes over time.

The whole process is divided into four activities:

- Develop Strategic Scenarios. This activity is in charge of identifying strategic scenarios and evaluating their characteristics like current and future features, possible systems to build, type of customers, etc. These scenarios are developed using GBN model (Global Business Network) ([RD.59] and [RD.60]). These strategic scenarios are useful to identify future changes.
- Propose Business Strategies. The end of this activity is to anticipate to future actions carried out by the organizations.
- Design Architectural Scenarios. To represent the architectures different models and views can be used. Additionally, variations (possible design choices) are represented using variation models. The final result is a set of proposed architectures which are described using architecture scenarios. These

architecture scenarios describe the final architecture views with no variations, this is choices in the variation models have been made.

- Assess Scenario Feasibility. The proposed architectures are evaluated. There has been developed a method within SODA to perform this task: SQUASH (Systematic Quantitative Analysis of Scenario Heuristics) [RD.59].

2.3. ARCHITECTURAL ANALYSIS METHODS: TRANSITION FROM DOMAIN MODELING TO DOMAIN ARCHITECTURE DEFINITION

During architectural analysis, the domain engineer selects an appropriate design approach for building a generic design. The design approach must accommodate the required features and it also has to be flexible enough to accommodate differences among individual applications to be built from a generic design.

Many of the methods mentioned previously only provide a feature model with a high-level architecture design. But it is needed a transition from the domain model to the final architectural design. For instance, SEI has build DSSA mapping the products of FODA onto an architectural style called OCA.

2.3.1.1. OCA (Object Connection Architecture)

OCA [RD.17] is the architectural model used to structure a generic design. This model is typically used with FODA. However, usable results may be possible when OCA is combined with other domain analysis methods.

It is needed a process to map domain information captured in FODA models into a generic design for software domain (OCA). This mapping process takes as inputs the domain model and the architecture information and the result is a generic design, which is used in an application development process to produce application code.

The mapping process is divided into the following steps:

- Analysis of the domain model identifying the domain objects and the groups of related features that describe the subsystems.
- Process of mapping the object and subsystems onto code templates.

The organization of the architecture is based on a partitioning strategy and a coordination model. The portioning strategy means building blocks to decompose large software problems. The coordination model binds separate activities into an ensemble. It is achieved using rules and templates, which determine how the building blocks interact. OCA has three architectural elements:

- Objects. An object maintains state information about the behavior of a real-world or virtual entity.
- Controllers. A controller aggregates objects to form a cohesive subsystem. A controller is a specific activity or task with which the group of objects is charged.
- Executives. The executive provides the operating environment for the subsystems within the application and, in most cases, is the arbitrator over conflicts between processes competing for time and access to shared resources. The executive monitors and controls time for subsystems and monitors interfaces to external entities such as hardware devices, other computers, and humans.

2.4. DOMAIN ENGINEERING NOTATIONS AND TOOLS

2.4.1. NOTATION AND TOOL SUPPORT FOR DOMAIN ANALYSIS

During the domain analysis the bounds of the domain are identified. There exist several notations to represent these bounds, the commonalities and the relations with other domains and entities. Some notations for domain analysis representation linked to features are:

- SADT (Structured Analysis and Design Technique) [RD.42] is a Software Engineering technique for describing systems as a hierarchy of functions. It is available as a commercial CASE tool under the name "IDEFO".

SADT provides two different kinds of diagrams: activity and data models. They are created following a top-down strategy. Diagram notation is based on data and activity boxes, which are interconnected through different types of arrows. Diagram decomposition is achieved by using a refinement process.

- UML (Unified Modeling Language) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems. It is a semi-formal object-oriented modeling language and supports graphical notation based on 9 types of diagrams.

→ Tools that support UML are Rational Rose (from Rational Software Corporation), Ms Visio, or Platinum Paradigm Plus.

- SysML (System Modeling Language) [RD.27] is based on UML 2.0 but the notation is more general, this way, it can be applied to every system or application.

SysML is a domain-specific modeling language for system engineering applications. It supports the specification, analysis, design, verification and validation of a wide range of systems. It also supports use-case diagrams, class diagrams, parametric diagrams, activity diagrams, sequence diagrams and state machine diagrams. It uses OMG XML Metadata Interchange (XMI) to exchange modeling data between tools. UML also uses XMI with this end.

- The reusable assets generated from feature modeling after normalization, expansion and constraint checking will be output into XML (eXtensible Markup Language) files. UML and XML are OMG's established standards.

The OMG standard XML Metadata Interchange (XMI) [RD.36] is an interchange format for metadata that is defined in terms of the Meta Object Family MOF. The objective is to apply XML to abstract systems such as UML or other OO data models. This way, a neutral-format is obtained which can be stored and exchanged between UML tools. Nowadays, XMI does not specify how to record graphical information.

Many domain analysis methodologies are based on features. Therefore, tools which can model features are also needed, such as:

- XFeature [RD.30]. This tool supports feature modeling. Users can define their own meta-models that describes a set of applications, which belong to the same domain. It uses standard technology such as XML and Eclipse. XFeature is free and open software under a General Public License.
- RequiLine [RD.32]. It is a requirements engineering tool for management of software product lines. It supports requirements modeling in term of features and natural-language requirements. It is based on FORM notation. This tool is still under development
- Pure: variants. It is a commercial tool. It supports feature modeling and configuration.

2.4.2. NOTATION AND TOOL SUPPORT FOR DOMAIN DESIGN

There are several notations to represent the domain design, such as:

- SysML (System Modeling Language).
- UML2 (Unified Modeling Language 2.0) [RD.58]. UML2 supports Model Driven Architecture and model-driven development. It provides a usable implementation of UML meta-models and a common XMI schema. UML2 facilitates the definition of domain-specific languages based on UML and this version also improves the language organization and precision.

It provides 13 different kinds of diagrams. Some of these new diagrams are interaction overview and UML Timing diagrams.

- AADL (Architecture Analysis and Design Language)([RD.4] and [RD.43]). AADL is a language standardized by SAE. It provides features for modeling a software system's conceptual architecture, distinguished from the system's implementation. It must support the building blocks of an architectural description:
 - Components.
 - Connectors.
 - Configurations.

AADL provides a consistent notation (textual and graphical) to develop real-time, critical and complex systems. It can describe functional interfaces to components and non-functional aspects of components. It can be used to model and analyze systems already in use and design and integrate new systems.

Candidate tools to develop the domain design:

- CORBA to integrate tool objects.
- ECLIPSE Framework [RD.29].
- Ms Visio.
- TOPCASED. It supports UML, AADL, SysML.
- Open Source AADL Tool Environment (OSATE) [RD.41], which includes multiple analysis plug-ins, TOPCASED integration.

2.4.3. NOTATION AND TOOL SUPPORT FOR DOMAIN IMPLEMENTATION

There are several notations, which support the domain implementation, such as:

- MDA (Model-Driven Architecture) ([RD.12] and [RD.33]) provides a framework to MDD. The basic function of MDA is the generation of applications from a set of procedures (UML model). This generation is independent of the platform and the language used. MDA can achieve this end with several transformations. Therefore, MDA provides a mechanism to transform the feature model instances into an executable application automatically or semi-automatically.

When a platform is chosen for a particular system, then the system specification is transformed into a particular platform. So the specification of the system functionality can be separated from the implementation of that functionality.

MDA defines system functionality using three kinds of models:

- PIM (Platform-independent model). PIM describes how the system will be implemented without specifying any technology platform. The specifications are represented using UML.
- CIM (Computational-independent model). CIM describes the requirements and the context where the application is going to be used.
- PSM (Platform-specific model). PSM describes the implementation in a specific platform.

PIM is translated into one or more specific models for the platform chosen (PSM) by mapping PIM to some implementation language or platform using formal rules. An example of model transformation standard is QVT (Query-View Transformation), which can use ATL language.

Several different approaches and standards exist for specifying the models of an application in development such as UML, Meta-Object Facility (MOF), XML Metadata Interchange (XMI), Enterprise Distributed Object Computing (EDOC) or Software Process Engineering Metamodel (SPEM).

- HRT-UML (Hard-Real Time UML Models) [RD.11] defines an extension profile of UML. So, concepts and techniques of the HRT-HOOD method can be expressed in standard UML.

It is used to model generic architectures and it is especially useful for modeling Hard-Real-Time systems. The design can considerer functional and timing requirements. It also analyses the schedulability of the system according to the theory and the design can be mapped into the implementation.

The traditional space design methods are based on HOOD standard (Hierarchical Object Oriented Design): HOOD4 (HOOD Reference Manual Release 4) and HRT-HOOD3, where HOOD4 is used on ground and (HRT-) HOOD3 on board.

HRT-UML is an adaptation of UML aiming at keeping the design engineering qualities of HOOD. HRT-UML is evolving towards a profile of UML2.0.

HOOD is a method of hierarchical decomposition of the design into software units based on identification of objects, classes and operations reflecting problem domain entities or more abstract objects related to digital programming entities. It is intended for the Architectural Design, Detailed

Design and coding for software to be developed in programming languages such as Ada, C, or FORTRAN, as well as in object oriented languages such as C++, Ada95 or Eiffel.

The HOOD method comprises textual and associated diagrammatic representations allowing formal refinement, automated checking, user customizable documentation generation and target language source code generation.

- HRT-UML/RCM (Hard-Real Time UML Models/Ravenscar Computational Model). RCM Methodology is followed in ASSERT project to design the non-functional part. Functional part follows FW Methodology. UML2 is suitable to model functional requirements, but meta-models are better suited to represent concurrency and timing semantics. RCM is a domain specific meta-model. This meta-model provides the possibility of designing high abstract level hard real-time systems. It includes three different views:
 - Functional view (Platform independent). It represents the sequential behaviour of the model.
 - Timing view (Platform dependent). It represents the concurrent architecture.
 - Interface view (Platform independent). It represents a high-level abstraction of the whole model.

The basic RCM entity is the RCM containers at application level (AP-level) and system level (VM-level).

Candidate tools to develop the domain design are:

- CORBA to integrate tool objects [RD.35].
- ECLIPSE Platform [RD.15].
- Ms Visio.
- HRT-UML toolset [RD.28].
- STOOD [RD.34].
- CHEDDAR [RD.40]. Cheddar is a real time scheduling tool developed by LISYC Team. Cheddar framework provides analysis tools and extensible simulation services. It can be used to perform AADL performance analysis.

Currently GMV is involved in the evaluation of a framework, developed by GMV for ITEA Program [RD.37], a model-based approach for Real-Time Embedded Systems including the specification of the Model Driven Engineering Process.

MARTES framework is based on plug-ins for a standard UML tool used in GMV (Together Architect). Together [RD.38] is a visual Modeling for Software Architecture Design, which Create UML 2 and business process models.

The Initial models have been applied to a Satellite Packet Data Archiving System (PDS) for Galileo Program. The framework for MARTES that suits GMV's needs (C++ development, Galileo SW standards).

MARTES MDD Process Framework is influenced by OMG's SPEM standard [RD.39] the Software Process Engineering Meta-Model that defines concepts for modeling software development processes.

3. CONCLUSIONS

3.1. OVERVIEW OF DOMAIN ENGINEERING METHODOLOGIES

Table 3-1 presents an overview of domain engineering methodologies analyzed in this document. The following information is included:

- **Analysis technique** Technique used to model the domain: feature (functional data decomposition), product-line features (functional data decomposition), Object-Oriented.
- **Domain Engineering Phases** Not all the methodologies cover the whole domain engineering process. The table indicates the specific phases that each method implements.
- **Requirements specification, verification and management.** Techniques can specify some processes or techniques for these stages.
 - Requirement specification. Requirements are documented using specific formats or languages such as XML.
 - Requirement verification. This stage is needed to check the precision, relevance and suitability of the requirements chosen to specify the domain.
 - Requirement management. Products must be properly documented and each activity well organized.

Domain Engineering Methodology	Analysis Technique	Domain Engineering Phases	Requirements, Specification, Verification and Management
ODM	Features	Plan domain Model domain Engineer asset base	No
FODA	Features	Context analysis Domain modeling Architecture modeling	No
FORM	Features	Context analysis Feature modeling Architecture modeling	Very little
FeatureRSEB	Features and Object-Oriented	Domain analysis Model Design Architecture definition	Yes
DSSA	Features	Requirements analysis Architecture modeling	Yes
Sandwich	Features	Domain analysis Domain model	Very little
DARE	Features	Analysis and design phase Architecture designs	Little
FAST	Product-line features	Domain analysis Domain implementation	Yes

Domain Engineering Methodology	Analysis Technique	Domain Engineering Phases	Requirements, Specification, Verification and Management
PuLSE	Product-line features	Initialization Infrastructure construction and usage Evolution Management	Very little
OOram	Object-Oriented	Model creation System development Building reusable asset	No
JODA	Object-Oriented	Domain preparation Domain definition Domain modeling	No
Sherlock	Object-Oriented	Domain definition Domain characterization Domain scoping Domain modeling Domain framework.	No

Table 3-1- Summary of domain engineering methodologies

Taking into account all domain-engineering methods described briefly previously, we have concluded that the most suitable approach is the feature-oriented domain analysis model, this is FODA ([RD.6] and [RD.13]). FODA abstracts the application until no difference exists. This way, specific applications can be created from these abstractions through refinement processes.

This choice has been done based on the following advantages:

- FODA represents domain knowledge using several complementary models.
- It is oriented to analyze commonalities and variabilities within a domain.
- Easy understanding of feature models from end users' and designers' point of view.
- Generic method. There exists an extended FODA, which complements this general approach. Extended FODA includes architecture design and object-oriented component development.
- There exists a tight relationship between models generated using FODA and those found in the majority of the Object Oriented Analysis and Design.
- Although FODA does not cover the Application Engineering process, it specifies all the Domain Analysis process until the architecture design.
- This method has been applied to several industry application domains with great results.

Following FODA method as outline, next section presents the different models, notations and tools chosen to cover the domain engineering life cycle.

Neither any specific notation nor tools are specified in FODA. Therefore, several possibilities have been analyzed [2.4] and these are the final notations and tools chosen for each one of the models.

3.2. DOMAIN ENGINEERING PHASE

3.2.1. DOMAIN ANALYSIS

Domain Analysis phase defines the scope of the domain providing a set of models which represent formally all the knowledge obtained about the domain.

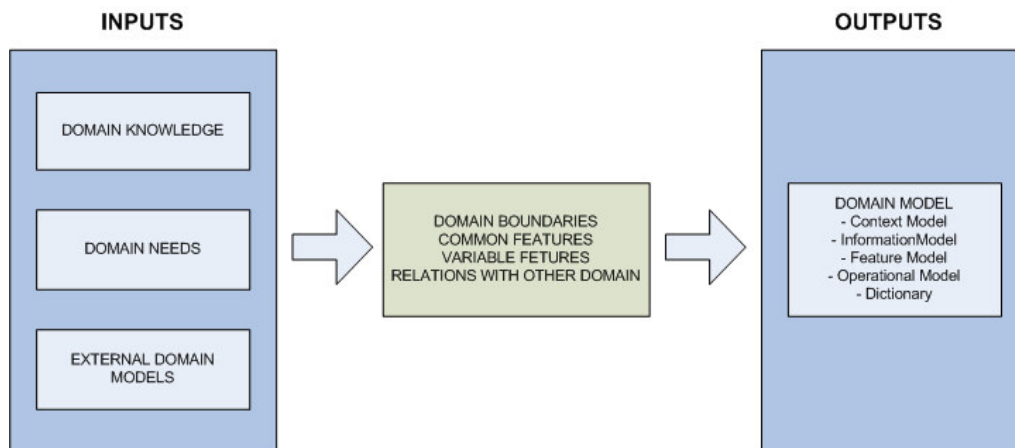


Figure 1. Domain Analysis Process

The domain model is based on the knowledge and needs of the target domain. From these information is extracted the requirements and features which are going to be modeled and also the information needed to specify the domain boundaries. In addition, existing domain models are taking as input. They provide additional information to determine the most suitable models for a specific target domain. As well, previous experiences can help us to identify areas to model, identification of features, etc. Indeed, these previous external domain models can also be an important point to avoid drawbacks obtained in previous models and take advantages of the main goals achieved.

Some examples of external domains to consider are AUTOSAR [RD.61] in the automotive domain, ARINC 653 [RD.62] in the aeronautical domain or EGOS [RD.63] in the space domain. They are developing standardized and general architectures in order to reduce costs, improve the development-cycle-process, enable the reuse/update of software, and so on.

Domain Analysis process has been divided into three different parts to get the final domain model:

- Requirements analysis.
- Context analysis.
- Domain modeling.

3.2.1.1. Requirements Analysis

First of all, functional and non-functional system requirements are collected. These requirements, also known as software requirements, are identified by stakeholders' and end users' point of view.

Requirements mainly differ from features with regard to the level of abstraction: features capture system functionalities and stakeholders' needs in a high abstraction level, whereas requirements are more specific.

The requirements and features gathering is a complex task. Stakeholders and end users needs are captured through interviews and questionnaires, [RD.46]. Domain Engineering developers has to understand each particular need with the end of transforming it into a proper requirement or feature.

Once, all the requirements are identified they have to be modeled to generate a requirements model.

- Inputs: List of domain requirements and needs.
- Outputs: Requirements model.

- Notation: SysML (UML does not provide any kind of diagram to model requirements).
- Tool: Topcased.

Requirements model can be completed with domain use-case diagrams to express the functional requirements. This diagram shows the interactions between end-users and the systems. As non-functional requirements are not represented in use-case diagrams, they can be attached in a separate document.

- Inputs: List of domain requirements and needs.
- Outputs: Use-case diagram.
- Notation: SysML/UML.
- Tool: Topcased.

3.2.1.2. Context Analysis

Context analysis tries to establish the bounds of the domain, the relationships with other domains (inputs and outputs), stored data requirements for the domain, and the scope of the analysis.

- Inputs: standards, application framework (information gathering in order to identify features).
- Outputs: A context or data-flow diagram where the domain is placed relative to higher, lower, and peer level domain. Variabilities are identified using several diagrams. In each one of the possible diagrams, the context is identified. This diagram also shows the data-flows between the target domain and other abstractions.
- Notations: To implement data-flow diagram two different notations have been considered: SysML Block diagrams (Block Definition Diagrams and Internal Block Diagrams) and UML Class diagram and Composite Structure diagram.
- Tools: Topcased.

3.2.1.3. Domain modeling

During the domain modeling phase, the context model is analyzed to generate domain models which involves:

- Information model.
- Feature model.
- Operational model.
- Dictionary.

These are the notations and tools chosen to represent each of them.

3.2.1.3.1. Information model

This model consists of the domain's entities or abstractions, and the relations between them. The information model is used by end-users to identify the capabilities of the domain.

- Inputs: Features, context model.
- Outputs: A more refinement block diagram than the one designed during the context analysis phase. Use-case diagrams to specify the information exchanged among end-users and the abstracted application.
- Notation: SysML use-case/block diagrams or UML use-case/class/structure diagrams.
- Tool: Topcased

3.2.1.3.2. Feature model

FODA defines features as a user-visible aspect or a characteristic of a software system. The feature model captures these common features and differences of the applications in a domain and their relationships.

- Inputs: Knowledge gathered among experts, end-users, etc.
- Outputs: Feature Diagram, feature definition, composition rules and rationale for rules.
- Notation: There is no specific standard to represent feature models. The most used (FODA notation) is the following. Features are represented in a tree-like structure. Each node represents a feature and its children nodes are the sub-features. There exist three different types of features:
 - Mandatory.
 - Alternative.
 - Optional

A possibility is to represent the hierarchical model as a UML static structure diagram or SysML block diagrams.

- Tools: XFeature [RD.30].

3.2.1.3.3. Operational model

The operational model describes the control and data-flow in the application domain, the relationships among objects in the information model and the feature model. It is useful for the analyst to define the functionality of the applications and the requirements needed. It is also used to identify existing functional and behavioral variations between various applications of a domain.

- Inputs: Context model, information model, feature model, technology of the domain.
- Outputs: interaction diagram, state diagrams. To express the variations these models have to be parameterized.
- Notation: SysML or UML. As SysML already provides parametric diagrams, this will be the best notation.
- Tools: Topcased

3.2.1.3.4. Dictionary

Generation of a domain terminology dictionary.

- Inputs: Experts/stakeholders vocabulary.
- Outputs: Text files considering all the standard vocabulary of domain experts.
- Notation: -
- Tools: Text editor. For instance, Microsoft Word.

At this moment, the requirements model can be refined basing on the information and design models generated.

Every model created in the analysis phase has to take into account features variabilities. The variabilities can be represented through different mechanisms, such as:

- Provide a set of different diagrams.
- Extensions.
- Inheritance.

The diagrams developed need to be manipulated by programs. This end can be achieved through the use of Metamodel Interchange XMI. It provides a platform independent representation of UML and SysML designs.

3.2.2. DOMAIN DESIGN

Domain Design is used to specify the application architecture within a domain.

- The domain architecture is a generic architecture based on the domain model. This architecture has to be documented and evaluated.
- The software architecture represents the structured used during the modeling process. The architecture may be described at different level of abstraction showing components used to build the system, their relations, interfaces, etc.

A generic architecture is platform and code language independent. It's like PIM element in MDA approach.

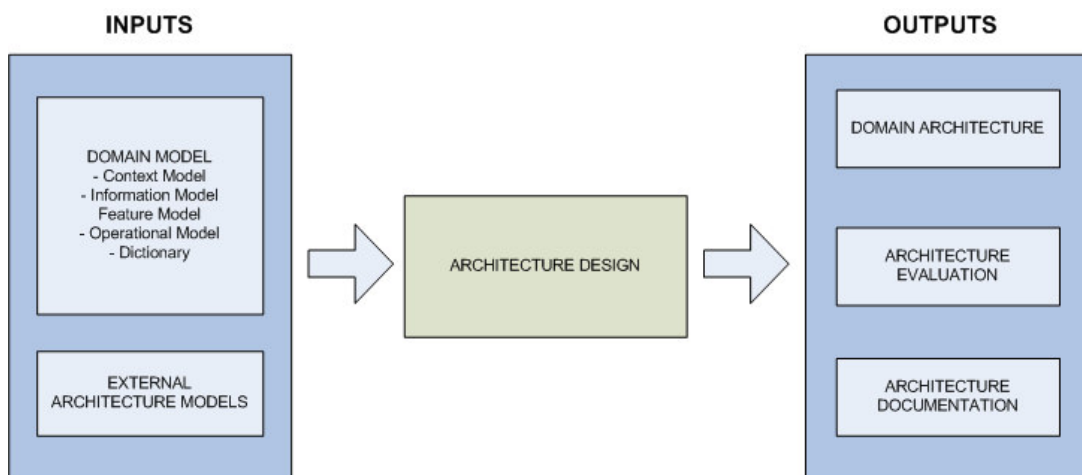


Figure 2. Domain Design Process

- Inputs: Domain model (Context model, Information model, Feature model, Operational model and dictionary) and other external architecture models. The external architecture models provide a vision of generic architectures already developed: types of architecture (centralized/distributed, layered,...), interfaces, communication mechanism, fault isolation mechanisms, etc. So, as mentioned in the domain analysis phase, several external standards have to be considered and the analysis of their architecture models will provide another important input in order to make decisions about the target architecture design.
- Outputs: Different architecture diagrams from different perspectives and level of abstraction.
 - Functional view: Class structure diagram, which represents the hierarchy of classes in a static view with interfaces and interconnections, and state charts diagrams that show the internal behavior of each particular component.
 - Non-Functional view: Timing, interaction overview and activity diagrams. As MDA promotes non-functional requirements have to be integrated in UML models. Examples of non-functional requirement are the performance or the schedule.
 - Optionally, sequence may be used. They complement the code generation from these models.
- Notations: UML2 [RD.58].
- Tools: Topcased

3.2.3. DOMAIN IMPLEMENTATION

Domain Implementation starts with the identification of reusable assets. This identification is made based on the domain model and the general architecture obtained in the previous phases. Using compilers and code standards the assets are implemented. Finally, reusable assets are created and saved in an assets' library.

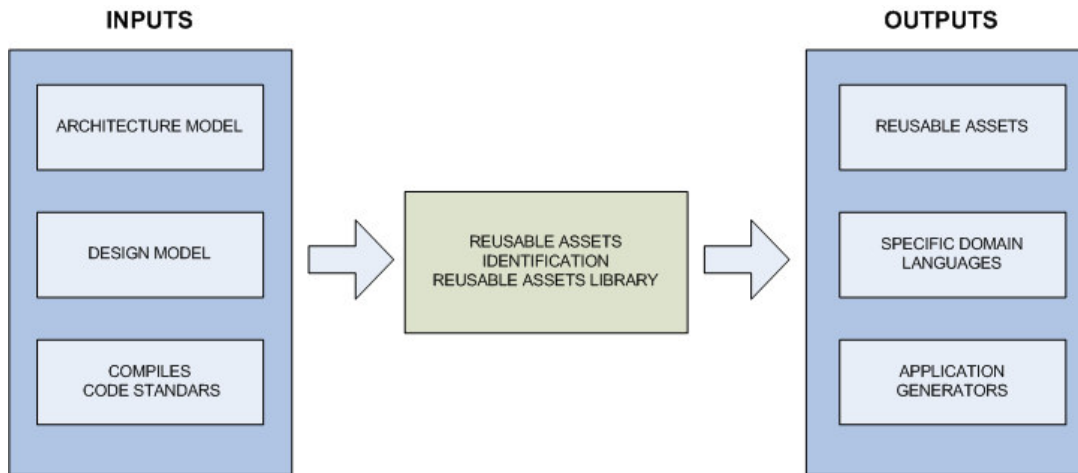


Figure 3. Domain Implementation Process

Code generation approach should be as generic as possible, so that each component could be transformed into a specific language and platform. This transformation process can be done manually, automatically or semi-automatically.

The tool selected to perform this task is Together [RD.38], which is a product line from Borland. It provides UML and an integrated development environment. There exist synchronization between design and implementation. It can be integrated in Eclipse platform through Eclipse plug-ins.

3.3. SUMMARY AND RECOMMENDATIONS

The next table summarizes the different models, notations and tools needed to cover the whole domain engineering process.

Domain Engineering Phase	Model	Inputs	Outputs	Notation	Tool
Domain Analysis	Requirements Model	Knowledge acquisition process and CORDET State of the art Reports	Requirements Model	SysML Requirement Diagram	Topcased
			Use-Case Diagram	SysML Use-Case Diagram	Topcased
	Context Model	Standards Application framework	Context Model	SysML Block diagrams	Topcased
	Information Model	Features Context Model	Information Model	SysML Use-Case and Block diagrams	Topcased
	Feature Model	Domain knowledge	Feature Model Feature definition Composition rules Rationales about rules	-	XFeature
	Operational Model	Context, Feature and Information models.	Parametric Model	SysML Parametric diagrams	Topcased
	Dictionary	Experts knowledge	Dictionary	-	Word or dictionary tool
Domain Design	Domain Architecture	Context, Information, Feature and Operational models.	Architecture models	UML2	Topcased or Together
	Domain Documentation	Context, Information, Feature, Operational and Architecture models	Architecture documentation	-	Word
	Architecture Evaluation	Architecture models	Architecture evaluation	-	Word
Domain Implementation	-	Architecture models	Physical Model	Reusable assets	Together

Table 3-2- Summary of Models, Notations and tools

Some considerations taken are the next ones:

- Some models can be represented using UML[RD.7] or SysML[RD.27]. The final choice has been SysML due to several reasons:
 - The context model can be represented through SysML block diagrams or UML class and composite structure diagrams. In this case, SysML provides a useful approach for system analysis because the abstraction level is higher. A block can represent any level of the system hierarchy and these diagrams also represent communications and relations among block. Therefore, SysML block diagrams are suitable candidates to represent the context of the domain.
 - The operational model can be represented with UML iteration and state diagrams. But, these diagrams do not represent variations. To represent variations, they have to be parameterized. SysML provides directly parametric diagrams in order to express the performance and constraints.
 - In some other cases both UML and SysML notation can be used indistinctly, because they provide the same features. In this case, SysML is used as well in order to use always the same notation.

SysML is just an extension to UML2 providing new stereotypes and diagrams. As well, SysML models can be exchanged among tools, and data generated in external tools can be included in a SysML model.

- There exist several SysML modeling tools, such as:
 - MDG Technology (Sparx Systems) [RD.47].
 - Rhapsody[RD.48] and Tau G2 [RD.49] (Telelogic).
 - SysML Toolkit (EmbeddedPlus) [RD.50].
 - Topcased [RD.31].
 - MagicDraw [RD.51].

Rhapsody and TAU G2 does not support SysML1.0 yet. MDG Technology and SysML Toolkit licenses need a royalty.

The final tool selected to represent SysML diagrams has been Topcased [RD.31]. It is free and open-source software tool and TOPCASED-UML2 can be built as an Eclipse plug-in. It can be download directly from its web page

- XFeature tool [RD.30] selected to develop the feature model can be built, as an Eclipse plug-in. XFeature is a free and open source tool.
- Domain implementation can be performed through Borland Together tool ([RD.38] and [RD.44]). Apart from Together, there exist several free tools to cover the code generation, such as:
 - Open Architecture ware Eclipse plug-in [RD.52].
 - UMT (UML Model Transformation Tool) [RD.53].
 - AndroMDA [RD.54].
 - Motion Modeling [RD.55].
 - MTL Engine [RD.56].
 - ModFact [RD.57].

ANNEX A. MAP OF GMV APPROACH AND ISO 12207 DOMAIN ENGINEERING REQUIREMENTS

The annex is intended to analyze the compliance of the approach proposed by GMV and the ISO 12207 [AD.4].

Annex G of ISO 12207/Amd.1 includes new processes to support Domain Engineering. Clause 7.7 defines the Domain engineering process, which is focus on Domain Engineering Activities and Tasks. The process covers “the development and maintenance of the domain models, domain architecture, and other assets for this domain”.

The Domain engineering process also considers periodical reviews to check if there exists any modification of reusable assets. This process is linked to “7.7.5 Asset maintenance” clause, which is not applicable to the phases defined in this technical note.

ISO 12207 notations followed is the next one:

- Subclause x.x denotes a process.
- Subclause x.x.x denotes an activity.
- Subclause x.x.x.x denotes a task.

Next table shows the status of compliance of the Domain Engineering phases considered by GMV (section 2.1) versus ISO 12207 activities and tasks [AD.4]. The status of compliance can be:

- FC – Fully Compliance.
- PC – Partially Compliance.
- NC – Non-compliance.
- NA – Non-applicable. Clauses out of scope of WP-202 are considered as NA.

ISO/IEC 12207:1995/Amd.1: 2002 [AD.4]			
Clause	Definition	GMV Approach	Status of Compliance
Activity 7.7.1	Process Implementation. Create, document and execute a domain engineering plan, including standards, methods, tools, activities, assignments, and responsibilities for performing domain engineering. Select the representation forms to be used for the domain models and architectures.	This document covers the domain engineering plan. It selects the most suitable standards, notations, models and tools.	FC
Activity 7.7.2	Domain Analysis. It is the activity that discovers and formally describes the commonalities and variabilities within a domain. The domain engineer captures this information in a set of domain models.	Domain Analysis	
Task 7.7.2.1	The domain engineer will define the boundaries of the domain and the relationships between this domain and other domains.	Context Model	FC
Task 7.7.2.2	The domain engineer will identify the current and anticipated needs of developers of software products within this domain.	Requirements Model	FC
Task 7.7.2.3	The domain engineer will build the domain models using the representation forms selected in the process implementation activity for this process.	Feature Model Information Model Operational Model	FC

ISO/IEC 12207:1995/Amd.1: 2002 [AD.4]			
Clause	Definition	GMV Approach	Status of Compliance
Task 7.7.2.4	The domain engineer will construct a vocabulary that provides the terminology to describe the important domain concepts and the relationships among similar or common assets of the domain.	Dictionary	FC
Task 7.7.2.5	The domain engineer will classify and document the domain models.	Feature, Information and Operational Models documentation.	FC
Task 7.7.2.6	The domain engineer will evaluate the domain models and domain vocabulary in accordance with the provisions of the modeling technique selected and in accordance with the organization's asset acceptance and certification procedures. The results of the evaluation shall be documented.	Feature, Information and Operational Model evaluation.	PC (No procedure proposed to evaluate domain models)
Task 7.7.2.7	The domain engineer will conduct domain analysis joint review(s) in accordance with the Joint Review Process specified in subclause 6.6. Software developers, asset managers, domain experts, and users shall be included in the reviews.	N/A	N/A
Task 7.7.2.8	The domain engineer will submit domain models to the asset manager.	N/A	N/A
Activity 7.7.3	Domain Design. It defines the domain architecture and specifies the assets that can be used to build software products.	Domain Design	
Task 7.7.3.1	The domain engineer will create and document the domain architecture, consistent with the domain model and following organization standards.	Domain Architecture	FC
Task 7.7.3.2	The domain architecture will be evaluated in accordance with the provisions of the architecture design technique selected and the organization's asset acceptance and certification procedures. The result of the evaluation shall be documented.	Architecture Evaluation	PC (No procedure proposed to evaluate domain architectures)
Task 7.7.3.3	For each entity selected to be designed for reuse, the domain engineer will develop and document an asset specification.	Domain Documentation	FC
Task 7.7.3.4	For each asset specified, the specification will be evaluated in accordance with the provisions of subclause 5.3.6.7, and in accordance with the organization's asset acceptance and certification procedures. The results of the evaluation shall be documented.	N/A	N/A
Task 7.7.3.5	The domain engineer will conduct domain design joint review(s) in accordance with the Joint Review Process specified in subclause 6.6. Software developers, domain experts, and asset managers shall be included in the reviews.	N/A	N/A
Task 7.7.3.6	The domain engineer will submit the domain architecture to the asset manager.	N/A	N/A

ISO/IEC 12207:1995/Amd.1: 2002 [AD.4]			
Clause	Definition	GMV Approach	Status of Compliance
Activity 7.7.4	Asset provision. The asset provision activity develops or acquires assets that can be used to assemble software products.	Domain Implementation	
Task 7.7.4.1	The domain engineer will develop the asset: Executing the Acquisition Process to cause a contract for the asset to be put in place if the asset is to be acquired, or executing the Development Process if the asset is to be developed internally.	Implementation of reusable assets	FC
Task 7.7.4.2	The asset will be documented and classified.	Reusable assets documentation	PC (No procedure proposed to classify reusable assets)
Task 7.7.4.3	The domain engineer will evaluate the asset in accordance with the organization's asset acceptance and certification procedures. The results of the evaluation will be documented.	Reusable assets evaluation.	PC (No procedure proposed to evaluate reusable assets)
Task 7.7.4.4	The domain engineer will conduct asset joint review(s) in accordance with the Joint Review Process specified in subclause 6.6. Software developers and asset managers shall be included in the reviews.	N/A	N/A
Task 7.7.4.5	The domain engineer will submit the asset to the asset manager.	N/A	N/A
Activity 7.7.5	Asset maintenance contains the tasks for modifying assets, including domain models and domain architectures.	N/A	N/A

Table A-1. Map of GMV approach and ISO 12207 requirements

ANNEX B. GMV, INTECS AND P&P NOTATION AND TOOLS COMPARATIVE

CORDET	GMV Issue 1			INTECS Issue 1.0		P&P Issue 1.0		Compatibility
	Model	Notation	Tool	Notation	Tool	Notation	Tool	
Domain Analysis	Requirements Model	SysML Requirement diagrams.	Topcased	Requirements diagrams. SysML Requirements diagrams	Topcased or COTS solution	<ul style="list-style-type: none"> - Domain Analysis Phase should not use any formalism with the exception of the feature model. It should be analogous to a conventional requirements definition phase where natural language is used. - XFeature tool in "FD Configuration" is chosen to build the feature model. "FD Configuration" is defined in Assert project. - Construction of a domain dictionary used to define terms that can be used to formulate shared properties. 		
		SysML Use-Case diagrams.	Topcased					
	Context Model	Context and Data-flow diagrams. SysML Block (Block definition and internal block) diagrams.	Topcased	Context diagrams. SysML block diagrams	Topcased or COTS solution			
				Use case diagrams. SysML use case diagrams.				
	Feature Model	UML static structure diagram or SysML block diagrams.	XFeature (Eclipse plug-in)	Feature model –specific notation (UML notation would be possible)	XFeature			
	Dictionary	-	Word or dictionary tool	Text	Word or other documentation tool			
	Information Model	SysML Use-Case and Block diagrams	Topcased	-	-			
	Operational Model	Interaction and State diagrams. SysML Parametric diagrams	Topcased	-	-			
High-level generic architecture	-	-	Top-Level partitioning (into applications). SysML Block diagrams	Topcased or COTS solution				

CORDET	GMV Issue 1			INTECS Issue 1.0		P&P Issue 1.0		Compatibility
	Model	Notation	Tool	Notation	Tool	Notation	Tool	
				<u>Applications use cases.</u> SysML Use-case diagrams. <u>Application Behaviour.</u> SysML interaction diagrams. <u>Specification of functions.</u> SysML Activity diagrams <u>Specific of dynamic behaviour-</u> SysML State Machine diagrams. Parametric diagrams -SysML Parametric diagram				
Domain Design	Domain Architecture	UML2 <u>Functional view:</u> Class structure and state chart diagrams. <u>Non-Functional view:</u> Timing, interaction overview and activity diagrams.	Topcased/ Eclipse(Together plug-in)	UML2 diagrams/ASSERT Methodology	Standard UML2 TOPCASED or COTS solution and ASSERT HRT-UML2 tool	UML2 models based on FW Profile for CORDET Software Framework. UML2 meta-model based on the RCM for CORDET System Family.	Topcased customized with the FW Profile Eclipse Plug-in. Topcased customized with RCM metadata model.	GMV, INTECS and P&P consider UML2 to design the architecture and Topcased is chosen as design tool.
	Transformation Rules	-	Together (Eclipse)	-	Eclipse Transformation Tools	-	-	
	Domain Documentation and Architecture Evaluation	-		Word	-	-	-	

CORDET	GMV Issue 1			INTECS Issue 1.0		P&P Issue 1.0		Compatibility
	Model	Notation	Tool	Notation	Tool	Notation	Tool	
	Platform Architecture	UML2	Topcased/ Eclipse(Togeth er plug-in)	Middleware/Component Model- UML diagrams/ASSERT Methodology/DisCo Space- Oriented Middleware	-	-	-	