UTILIZATION OF FEATURE MODELING IN AXIOMATIC DESIGN

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ORHAN ÜÇTEPE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

APRIL 2008

Approval of the thesis:

UTILIZATION OF FEATURE MODELING IN AXIOMATIC DESIGN

submitted by **ORHAN ÜÇTEPE** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen ______________________
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Volkan Atalay ______________________
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Ali H. Doğru ______________________
Supervisor, Computer Engineering Dept., METU

Examining Committee Members:

Asst. Prof. Dr. Pınar Şenkul ______________________
Computer Engineering Dept., METU

Assoc. Prof. Dr. Ali H. Doğru ______________________
Computer Engineering Dept., METU

Dr. Meltem Turhan Yöndem ______________________
Computer Engineering Dept., METU

Asst. Prof. Dr. Aysu Betin Can ______________________
Informatics Institute, METU

Dr. Ali Öztürk ______________________
Senior Engineer, Havelsan

Date: 07. 04.2008

ABSTRACT

UTILIZATION OF FEATURE MODELING IN AXIOMATIC DESIGN

ÜÇTEPE, ORHAN

M.S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Ali Hikmet Doğru

April 2008, 86 pages

This thesis provides an approach to use feature modeling with a set of guidelines for requirements definition and decomposition activities of the axiomatic design methodology. A tool that supports the development of feature models and modeling of the Axiomatic Design activities is implemented to be utilized for guiding the designer. Axiomatic Design suggested four domains of information in the transformation of the problem definition to the solution, and provided mechanisms for supporting the mapping among some of those domains. The approach suggested in this thesis fills an important gap, which is the transition from the customer needs to functional requirements, in axiomatic design. A case study is carried out in order to analyze advantages and disadvantages of the proposed approach.

Keywords: Feature Modeling, Axiomatic Design, Functional Requirements, Reuse in Software.

ÖZ

AKSİYOMATİK TASARIMDA YETENEK MODELLEMESİ KULLANIMI

ÜÇTEPE, ORHAN

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Ali Hikmet Doğru

Nisan 2008, 86 sayfa

Bu tez, aksiyomatik tasarım metodunda gerenksinimlerin tanımlanması ve hiyerarşik çözümlenmesi sırasında yetenek modellerinden faydalanılmasını önermektedir ve bazı yönergeler sunmaktadır. Tasarımcıyı yönlendirmek için yetenek modellemesinin ve aksiyomatik tasarim aktivitelerinin gerçekleştirildiği bir araç geliştirilmiştir. Aksiyomatik tasarım, problem tanımlarından çözüme gitmede dört bilgi alanı sunmuş ve bunlardan bazıları arasında geçiş mekanizması sağlamıştır. Bu tezde sunulan yaklaşım, aksiyomatik tasarımın ilk iki bilgi alanı olan müşteri ihtiyaçlarından fonksiyonel gereksinimlere geçişte önemli bir eksikliği doldurmaktadır. Sunulan yaklaşımın olumlu ve olumsuz yönleri örnek bir sistemin gereksinimleri üzerinde çalışılmıştır.

Anahtar Kelimeler: Yetenek Modellemesi, Aksiyomatik Tasarım, Fonksiyonel Gereksinimler, Yazılımda Yeniden Kullanım.

ACKNOWLEDGEMENTS

The author wishes to express his deepest gratitude to his supervisor Assoc.Prof. Dr. Ali Doğru for his guidance, advice, criticism, encouragements and insight throughout the research.

The author would also like to thank Cengiz Togay for his suggestions and comments during research activity.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

As the granularity of software systems started to increase; it became harder to compete with the increasing complexity of software systems in terms of quality, budget and time. Therefore, new software production methodologies have been proposed in order to increase the quality and decrease the cost and time of software production. Many research studies and industrial experiences have shown that reuse of previously produced assets and well-defined system decomposition plays an important role in producing high quality software systems in time and budget. Component oriented software engineering [3] and axiomatic design [5] are two approaches for decomposition strategies of software systems design. Although they can be used independent from each other, a combined approach [6, 7, 8] that adapts axiomatic design as an activity of component oriented software engineering has resulted with more efficient decomposition strategy. Domain analysis [32] activity and within it, feature modeling have proven to be a successful strategy for reuse mechanism in software development. Feature modeling activity is similar to axiomatic design in a way that it can be adapted to other methodologies as an extension and has been used to extend software methodologies.

Reuse can be applied to all phases of software development activity. Code reuse is the latest and very important part of a reuse strategy leading to a high quality software product. However, implementation phase lies at the latest stages of software development activity and it is supported by the artifacts created at the previous phases, like analysis and design. Therefore, besides code reuse, reuse of other artifacts is important in today's software development.

A well-defined reuse strategy starts at the beginning of a software development activity and continues to the latest phases. At the beginning of a software development activity, reusing previously gained knowledge, functionality and requirements leads to easier understanding of the problem domain and provides high quality artifacts for the remaining phases.

Methodologies supported reuse in different phases of their software production activity. Object oriented methodology supported code reuse within a single application development activity. Component based/oriented approaches introduced sharing common components among similar applications while developing a software system. Domain Engineering introduced reuse of previously created assets for similar applications at the very early stages of software production activity with a clear guidance.

Object oriented methodology tries to overcome the complexity problem by modeling and programming the software in terms of interacting objects similar to real world objects. Considering the world as a collection of interacting objects, object oriented methodology considers the software systems as collections of interacting objects having attributes, operations, and behaviors. Three main phases: object oriented analysis, object oriented design, and object oriented programming constitute major phases of the object oriented methodology.

Object oriented analysis tries to define what the system is supposed to do with sets of models constituting a whole conceptual model by considering all the inputs from all interested parties, like requirements, interviews… etc. Object oriented design takes the output models of analysis phase and tries to define how the system implements the required functionalities by also considering system specific constraints like environment, hardware, network, programming language etc. Design phase outputs are sets of models which map the conceptual models of analysis phase to implementation elements. At the end object oriented programming is used to implement the system using models created in the previous phases.

Object oriented methodology includes several concepts like class, inheritance, encapsulation, polymorphism to provide easier understanding and implementation of the required systems for better quality. Complex systems have been reduced to less complex systems by applying object oriented concepts and principles. Reuse of existing code provided less complex element sets and aided the creation of system parts in a less amount of time [1]. On the other hand, object oriented methodology has been insufficient to support a complete reuse mechanism. Although reusing existing code provided a partial reuse mechanism to the methodology, reuse of assets created at the end of each phase have not been well-defined in object oriented methodology. Methodology does not guide a specific reuse process for similar systems sharing common knowledge and functionality. Adapting reuse strategies to object oriented methodology have been studied in order to extend the reuse to other phases of the object oriented methodology [1, 2].

Production based engineering principles affected software engineering leading to the definition of component based methodologies. Component based methodology defines a process of functional and logical decomposition of a system using components providing a set of interfaces for interaction. Component based methodology led to the introduction of component oriented development and component oriented software engineering. Component oriented software engineering defines software production activity as decomposing the problem domain into abstract design elements (components, interfaces, functional abstractions…etc) and their interaction in a recursive way leading the decomposition of design elements to lower level design elements. Decomposed hierarchy is analyzed to reach the existing components and system is built by integrating existing components [3].

Component based/oriented methodologies define a reuse strategy based on high-level design elements: namely components. Reuse strategy starts with the analysis of system design and then reaching the existing components. When an existing component is found for a design element then further decomposition is not needed and design of that component can be used at the design level. At the implementation phase, existing component is used as it is and integrated into the current system without any implementation effort for the functionality that component provides.

Object oriented methodology and component oriented methodology do not have any well-defined process and guidance for a reuse strategy at higher levels than design level. Higher level reuse mechanisms and their importance have been a main research topic for years and current domain engineering concepts complete the missing phases of reuse activity for software production methodologies.

Today’s most software production activities start with the analysis of the current problem domain and then continue until a conceptual model is created to guide the related stakeholders to the next level of creating a specific solution model for the problem. At this phase, decomposition of the problem into smaller sub-problems is one of the most important steps. While object oriented methodology tries to solve the problem by creating a conceptual model composed of sets of objects similar to real world objects; component oriented methodology decomposes the system into functional and/or logical subsystems and components. At its domain analysis phase, domain engineering provides a similar approach to component oriented methodology by decomposing the problem domain into smaller sub-problems based on the functionality and/or logical definition of the sub-problems and then creating conceptual solution architecture. Since decomposition starts at the very early stages of software development activity; a well-defined and correct decomposition will lead to high quality results at the subsequent stages of the software development. The lack of guidance for decomposition in each methodology might lead to incorrect and inefficient decomposition of the problem domain. In this respect, extending the methodologies with efficient strategies for decomposition will create more effective methodologies.

Axiomatic design methodology, "a process to improve the quality and performance of complex system and product development" [5], provides a set of well-defined guidelines and principles for product development with a clear decomposition strategy. Therefore adapting axiomatic design into existing methodologies will provide more efficient methodologies to create software systems. Different research studies have been published [5,6,7,8,9,10,11,12,13,14,15,16] on this topic and some of them will be explained in Section 2.3.

During this thesis study, a collaborative research activity took place as part of the PhD research studies of Cengiz Togay. In his research studies, he introduced a new approach, called Axiomatic Design for Component Orientation (ADCO), to Component Oriented Software Engineering (COSE) methodology [3]. In ADCO, a decomposition strategy guided by the axiomatic design is provided to decompose the system into functional requirements and design elements. Guidelines and tools to search for existing components and development of unavailable components according to the design are provided for the production of higher quality systems using the COSE methodology.

In this thesis, utilization of feature models and a set of guidelines are provided for functional requirement definition and decomposition in axiomatic design. As axiomatic design does not have any process for reuse at the requirement level, reuse of existing knowledge and/or experience in feature models are utilized. This approach will also provide well-defined problem decomposition at the domain-level for axiomatic design activity. An axiomatic design tool and a feature modeling tool were developed as part of this thesis study to provide tool support for the proposed approach.

ADCO approach supports the process of decomposing the requirements and mapping them to design parameters. Whereas Axiomatic Design does not provide techniques or guidance for a mapping between the Customer Needs and Functional Requirements. This thesis actually supports Axiomatic Design through complementing it with such missing capabilities.

This thesis document is divided into five chapters. Second chapter will provide necessary knowledge in order to understand the concepts discussed in this thesis study. Domain analysis, feature modeling and axiomatic design concepts will be provided in detail. Third chapter is about the utilization of the feature modeling in axiomatic design. Rationale for this thesis study and proposed approach are provided. Fourth chapter contains a case study in order to analyze benefits and drawback of the proposed approach. Last chapter provides an analysis of the proposed approach and concludes the document.

CHAPTER 2

BACKGROUND

As this thesis study is about utilization of feature modeling in axiomatic design activity; three main concepts (domain analysis, feature modeling and axiomatic design) will be introduced, including research studies and previous experiences in the field, in order to provide a clear understanding of the contents discussed in the following chapters.

Feature modeling and axiomatic design are the main concepts used in this thesis; however a clear understanding of domain analysis is needed in order to understand the exact purpose and advantages of feature modeling. Therefore this chapter will start with the information about domain analysis, then feature modeling is explained in detail and the last section will be about axiomatic design.

2.1 Domain Analysis

In this section; first the definition of the domain is analyzed through historical research studies, and then a brief introduction to domain engineering is provided with domain engineering phases. Remaining sections will provide detailed information about domain analysis.

2.1.1 Domain

Domain term has been used to define different concepts for different research areas. In WordNet database of Princeton University, domain has a definition as "… the content of a particular field of knowledge" [17]. In American Heritage Dictionary it is defined as "A sphere of activity, concern, or function; a field: *the domain of history*" [18]. Those two definitions basically define the domain as a set of information content related to a specific area/activity. Some of the definitions for domain in software development area are similar to the definitions above; however most definitions focus on the reuse aspect of the domain.

In SEI-CMU domain engineering web site, domain is defined as:

"The term domain is used to denote or group a set of systems or functional areas, within systems, that exhibit similar functionality" [19].

In this definition, systems creating the domain have a set of similar characteristics and in this way contain reusable elements. Similar definitions have been published. In FODA (Feature Oriented Domain Analysis) proposed by SEI-CMU; domain has a similar definition of "A set of current and future applications which share a set of common capabilities and data" [20].

These and similar definitions focus on the commonality in terms of functional and data characteristics. Based on previous common definitions of the domain as a field of focus, below definition will provide a different focus to domain:

"A software engineering domain - a field of study that defines a set of common requirements, terminology, and functionality for any software program constructed to solve a problem in that field". [21] In this definition, domain is not defined as a set of systems; but it is defined as a field studying a set of systems for commonality in a specific area. Notice that this definition also provides requirements as a reusable common characteristic of a software system.

In his PhD thesis, K.Czarnecki analyzes different definitions of the term domain and provides his own definition of the domain as:

"Domain: An area of knowledge

- scoped to maximize the satisfaction of the requirements of its stakeholders,
- including a set of concepts and terminology understood by practitioners in that area, and
- including knowledge of how to build software systems (or parts of software systems) in that area." [22]

Czarnecki groups domain scope into two categories: Horizontal and Vertical Scope. In general, complete systems constitute vertical domains; on the other hand horizontal domains are composed of the parts of the systems. For example; domain of hospital information systems represents a well-defined vertical domain; domain of containers can be an example of a horizontal domain. [22]

Relations between domains provide three types of domain sets: sub-domains, support domains and analogy domains. [22] Similar to set relation in mathematics, when content of domain B is fully contained in the content of domain A; then domain B is a sub-domain of domain A. Domain of text formatted military messages is a sub-domain of domain of command control (C2) systems as command control (C2) domain also includes the formatted messaging.

When content of domain A uses the content of domain C; then domain C is a support domain of domain A. Domain of geographical information systems is used in command control (C2) systems to define many different concepts. In this respect, domain of geographical information systems is a support domain of command control (C2) systems.

When one domain is similar to the other domain in a way that information content in one domain helps understanding the other domain; then those domains are analogous to each other. Domain of mission planning systems used in the army and domain of mission planning systems used in the navy are similar to each other from many aspects and knowledge of one of these domains will help understanding the other one; therefore these domains are analogous to each other.

2.1.2 Domain Engineering

Many years of software development research showed that software systems with the same or similar target domains share a lot of common characteristics. Starting from the problem specifications to the requirements, conceptual model elements and concrete model elements; there are a lot of common development elements for software systems in a domain. Presence of reusable characteristics and principles taken from other engineering disciplines (like electronics and mechanical engineering) led the researchers to study principles to maximize reuse and create a software production methodology similar to other engineering principles. Result of these studies brought new and efficient concepts to the software world like domain engineering and product line engineering.

A simple definition of the domain engineering would be the process of analyzing a domain in order to create a set of artifacts to provide efficient and reuse based production of new software systems in that domain.

In glossary of software reuse terms, domain engineering is defined as:

"the process of

- defining the scope (i.e., domain definition)
- analyzing the domain (i.e., domain analysis)
- specifying the structure (i.e., domain architecture development)
- building the components (e.g., requirements, designs, software code, documentation)

for a class of subsystems that will support reuse" [4].

Czarnecki defines the domain engineering as "… the activity of collecting, organizing, and storing past experience in building systems or arts of systems in a particular domain in the form of reusable assets (i.e. reusable work products), as well as providing an adequate means for reusing these assets (i.e. retrieval, qualification, dissemination, adaptation assembly, etc.) when building new systems." [22]

There are similar definitions of the domain engineering in the literature. Domain engineering definitions and phases might have small changes from one research study to another research study; however the main focus of the domain engineering has been reuse of previously gained knowledge and creation of assets for creation of similar software systems [23].

Domain engineering activities can be grouped into three main phases; domain analysis, domain design and domain implementation [22].

Domain analysis is the process of gathering domain information and creating a set of models to represent the domain information. The domain model created in the domain analysis phase is used in domain design phase to create a conceptual model of the domain. At the latest phase, common conceptual model is implemented to produce a reuse infrastructure with reusable components, domain specific languages etc. A domain engineering study creates outputs for application engineering process, which results with new software products. Three main phases of domain engineering combined with application engineering can be seen in Figure 1 [22].

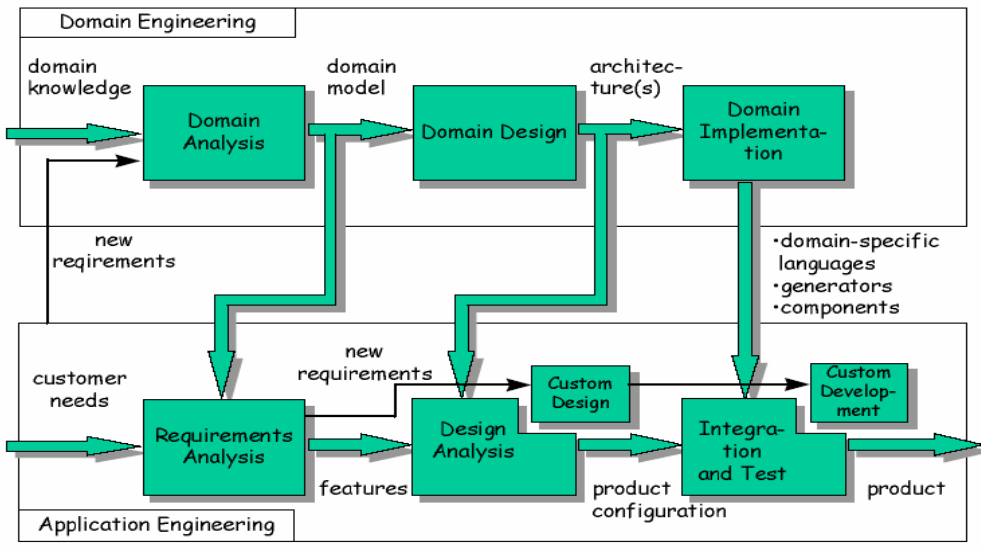


Figure 1 Domain Engineering steps combined with Application Engineering steps [22]

This thesis study focuses on domain analysis phase of domain engineering. Therefore domain engineering phases will not be further explained in detail. Domain analysis will be explained and analyzed in more detail in the following section.

2.1.3 Domain Analysis

2.1.3.1 Definition

Domain analysis is the first activity of domain engineering in order to collect relevant information within a domain and then create well organized set of models. Latest research studies provided domain analysis approaches similar to the domain engineering discussed in section 2.1.2; however this thesis study focuses on the definition provided by Czarnecki [22] which separates the domain architecture creation activity from domain analysis process.

Prieto-Díaz defines the domain analysis as "a process by which information used in developing software systems is identified, captured, and organized with the purpose of making it reusable when creating new systems"[24]

This definition provides three important activities of domain analysis:

- Identifying information content related to the domain at focus.
- Capturing identified knowledge as a set of models understandable by different stakeholders of the domain analysis.
- Organizing captured knowledge in a way that can be used for the subsequent phases after the domain analysis.

The most important outcome and aim of the domain analysis is defined to be the creation of a domain model leading to a reusable asset base for production of new software systems.

In FODA feasibility study, domain analysis has an extended definition providing a set of activities:

“The process of identifying, collecting, organizing, and representing the relevant information in a domain based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain”[20] This definition also provides a set of methods to be applied during domain analysis process. Mentioned methods will be explained when domain analysis activities are explained in detail.

Similarly, Czarnecki defines two main purposes of domain analysis as defining the domain of focus and then collecting information and representing that information content in a domain model [22].

2.1.3.2 Domain Model

The resulting output of the domain analysis is the domain model consisting of information required to understand and evaluate the systems within that domain. Domain model is "the formal representation of the knowledge necessary to support specific operational goals" [28]. Some of the operational goals supported by the domain model are listed below [28]:

- Requirements & Specifications: System specifications and software requirements can be identified, formalized and verified using a well-defined domain model.
- Automated code generation: Using previously created component repository and transformation languages, an automated code generation can be used to create required components from a given system specification with the support of the domain model.
- Reverse Engineering, Decision Modeling: Underlying rationale using and implementing for a set of previously created components and usage context of those components can be understood using a well-defined domain model.
- Training & Education: A common viewpoint of the software systems in the domain for different stakeholders (customer, analyst, user … etc) can be provided with a well-defined domain model and can be used by different stakeholders of the system.

Czarnecki defines domain model as an "explicit representation of the common and the variable properties of the systems in a domain and the dependencies between the variable properties" [22]. The actual aim of the information content in a domain model is well-defined in this definition. Domain model contains commonality and variability information about software systems within a domain. Dependencies among variable elements are also included within a domain model. Contents of a resulting domain model are provided as [22]:

- **Domain definition**: The domain of interest is defined and scoped to only interested information content. In a domain model, domain definition is provided with examples of the systems within and outside the domain in order to provide an easily understandable definition and scope of the domain.

- **Domain Lexicon**: Vocabulary of the concepts related to the domain of interest is provided inside a domain model. Understanding the meanings of the concepts in a domain will provide easier understanding of the systems and information content within that domain. For example, domain of command control (C2) systems includes lots of concepts like mission, situational awareness, data links … etc. Understanding these concepts will provide understandability of the command control (C2) systems for each phase of software production activity to the stakeholders related to that phase.

- **Concept Models**: A concept model is a well-defined representation of a concept in the domain using specific models. Concept models can be object diagrams, interaction diagrams, state diagrams, entity-relation diagrams and data flow diagrams. Concept models provide understandability of the domain concepts from a functional view and provide relations among concepts.

- **Feature Models**: Feature model defines common and variable characteristics of the systems within a domain. It also includes relations among features and helps defining valid feature sets for a specific system in the domain. Feature models are the most important part of a domain model and most important output of the domain analysis process for reusability.

Domain model is also defined as "a definition of the functions, objects, data, and relationships in a domain" [20]. In this definition, domain model is defined using its content. Mentioned content provides the contents of the domain similar to the ones provided by Czarnecki [22]. Content of the domain model is provided in FODA process as below:

"…

- features of software in the domain
- standard vocabulary of domain experts
- documentation of the entities embodied in software
- generic software requirements via control flow, data flow, and other specification techniques" [20].

It can be seen from above explanations that content of the domain model after a domain analysis activity is similar in different research studies. Each of these assets constituting the domain model will lead to better understandability of the domain and help an efficient reuse strategy while creating specific instances of applications in the domain. “An ideal domain model and architecture would be applicable throughout the life cycle from requirements analysis through maintenance.” [20].

2.1.3.2.1 Feature Models

A feature is a property or characteristic of a concept that is valuable from the point of relevant stakeholders’ view. For example zoom functionality is a feature for domain of geographical information systems domain for the map concept.

In domain analysis, common and variable characteristics of the systems in the domain are represented using feature models. A feature model consists of features and their relations. Some of the high-level features are also named as concepts.

Features are grouped into two categories; mandatory features and optional features. Based on the definition and constraints of a feature, a group of sub-features can be included in a set of alternative-features or or-features. In an alternative-feature set, only one feature can be selected at a time, meaning each feature in an alternative set is mutually exclusive with all other ones in the set. In an or-feature set, one or more of the features can be selected at a time. Mutually exclusive and require are two rules which can be used in feature models to represent relations among features [20, 22]. An example feature model from [20] is provided below.

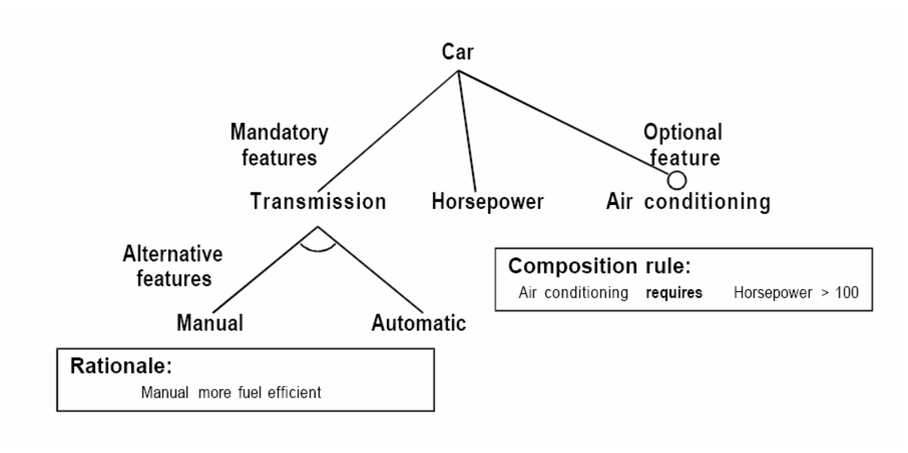


Figure 2 An example feature model [20]

In this feature model, the main concept (car) has two mandatory features (transmission and horsepower) and one optional feature (air conditioning). Transmission feature has an alternative set of features consisting of mandatory features (manual and automatic) as sub-features. The model also defines a rationale for selecting manual transmission and requires constraint for air conditioning.

Feature modeling activity of the domain analysis process can be utilized as plug-in to other software development methodologies in order to extend that methodology with a more efficient reuse capability. Research studies have been carried out to adopt feature modeling into software development methodologies and it has been experienced that feature modeling increases the rate of reuse. More detailed analysis of feature modeling will be provided in Section 2.2

2.1.3.3 Domain Analysis Methods

Domain analysis term has been used in research studies since 80s. Some of the first the research projects [25, 26, 27, 29] named their methodology as domain analysis without giving a proper definition of the process. By the time, new research studies proposed similar approaches for the definition of the domain analysis process and identification of the activities within it.

The published reports for earlier research projects [25, 26, 27, 29] have used some of the domain analysis principles. However main focus of those projects was the resulting outcome, not the process itself [24]. Clear identification of the domain analysis process has been proposed in many research publications. In the remaining parts of this section, ODM (Organizational Domain Modeling) [31] is introduced briefly. Then, FODA [20], a widely referenced domain analysis methodology, and FORM [33], an extension of FODA, will be provided in detail. For more detailed information on proposed domain analysis methods see [32] and related publications.

2.1.3.3.1 ODM (Organizational Domain Modeling)

ODM is a domain engineering methodology created and improved as a result of a set of academic and industrial research studies [22, 31]. ODM defines three main phases for domain engineering; domain planning, domain modeling and engineering asset base.

In domain planning phase, a set of stakeholders for the domain of focus is determined. Objectives of each stakeholder are defined as candidate objectives and then a set of final objectives are selected from the candidate objective set. Resulting objectives are used to scope the domain and define the boundaries of the domain. Features of the systems in the domain are identified; then, scoped domain is analyzed in order to define relations with other domains.

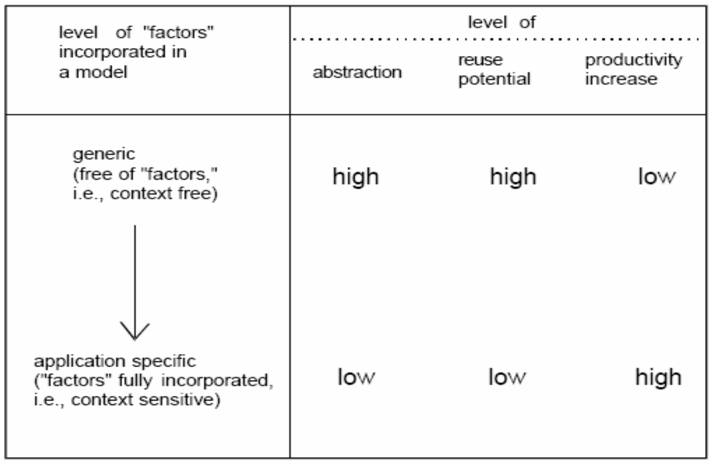
In domain modeling phase; a set of information sources is analyzed and a domain dictionary is created. Domain concepts with their associated features are discovered. Commonality and variability analysis are applied and results are combined in a domain model to be used in the engineering asset base activity.

In engineering asset base activity, concepts and features are prioritized and correlated to define an asset base in the domain. Each asset base is analyzed and an architecture model is created; then, designed architecture is implemented based on the created architecture model.

The domain analysis concept in ODM is similar to the one defined by Czarnecki in [22], where domain design is separated from domain analysis activities and handled in a separate phase.

2.1.3.3.2 FODA (Feature Oriented Domain Analysis)

The purpose of the FODA method is to identify "prominent or distinctive *features*" [20] of software systems in a specific domain. At the beginning of domain modeling activity, common characteristics are included and variable characteristics are excluded in the model; then variable characteristics are added to define variability. In this way a generic model to be used in all applications are developed. Higher level abstractions in an abstraction hierarchy are defined to be more reusable than the lower levels. For example transmission property of a car concept is common to all cars in the domain. When we decompose the transmission feature to lower levels, we create two new sub-features; automatic and manual. At this level, reusability is decreased. Although reusability is decreased, lower level features provide refinement of the model and creation of specific instances from generic model and increases productivity. Figure 3 [20] provides an insight to this approach.



level of "factors" incorporated in a model	level of		
	abstraction	reuse potential	productivity increase
generic (free of "factors," i.e., context free)	high	high	low
↓			
application specific ("factors" fully incorporated, i.e., context sensitive)	low	low	high

Figure 3 Characteristics of element levels in a model in FODA [20]

FODA defines four groups of information sources for domain analysis activity:

- **Textbooks:** Today it is possible to find lots of textbooks, published reports, and papers about almost all fields. Information content of the domain from theory to application can be found in these sources. On the other hand, source of these books are different authors and therefore information provided in these sources might have been effected by the author's specific view of the domain.
- **Standards:** Standardization has been an important activity for almost all production engineering principles for most of the domains. For example, military standards, procedures and doctrines have been defined and can be used to understand the domain. NATO has been studying military standards for years and most of the companies working on military software, which is connected to NATO operations, use defined standards in order to comply with the domain. The only disadvantage of the standards is that they may not be up-to-date.
- **Existing Systems:** As domain analysis tries to define common and variable characteristics of the systems in the domain; analyzing existing systems in the domain is the most important source of domain knowledge. Assets created for an existing system can be used to determine features of the systems in the domain. Architectural decomposition of the existing systems can help to define a well decomposed domain model. Analysis of existing systems is a time consuming effort and might be complex if implementation level analysis is required. Therefore, a filtering strategy should be applied while selecting existing systems in order to decrease number of the systems to be analyzed and to increase the quality of the analysis results.
- **Domain Experts:** Experience is an important factor to collect domain information. Domain experts can give important details about a procedure or a functionality that cannot be found elsewhere. For example; an F-16 flight management system analysis will not be complete unless some F-16 pilots provide their experience and knowledge for the domain. Domain experts are hidden treasures for software development methodologies. Absence of a domain

expert will cause more effort and more cost at all stages of the software development.

Domain analysis in FODA consists of three main phases, Context Analysis, Domain Modeling and Architecture Modeling. In Figure 4 [20] three phases of the FODA can be seen with outputs of each phase.

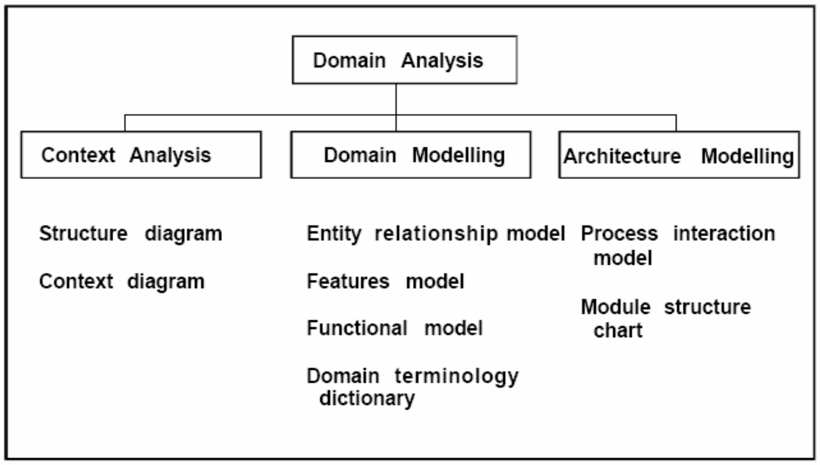


Figure 4 Domain Analysis activities in FODA [20]

2.1.3.3.2.1 Context Analysis

In context analysis, domain boundaries are analyzed through domain scoping. Domain is scoped to the focus area and external factors are analyzed to determine relations, commonality, and constraints of the selected domain with those external factors. Operating environment, standards, project constraints, other related domains and sub-domains are analyzed and a context model is created to identify the different contexts of the domain.

Context model is a representation of the domain scope and is used in domain modeling phase to solve the problem pertained to the domain within its defined scope. Context model consists of structure and data-flow diagrams (context diagrams). A context diagram can be defined as a representation of different situations, operations or functions and relations among elements during that situation, operation or function. Context diagram also provides the scope of the domain and a set of external and internal interfaces and data flow for the domain. An example structure diagram and context diagram for Army Movement Control Domain can be seen in [30].

2.1.3.3.2.2 Domain Modeling

In the domain modeling phase, context model created in context analysis phase is used to understand the scope of the domain. Applications in the domain are analyzed in detail to create a set of commonality and variability results. Feature model, entity-relationship diagrams, functional models, and domain dictionary are important outputs of this phase. Domain modeling phase can be divided into three activities; feature analysis, entity-relationship modeling, and functional analysis. [20]

Feature analysis is the core activity of the domain modeling phase in FODA. During feature analysis user visible characteristics (features) of the applications in the domain are discovered. Definition and context of each feature are defined and relations among features are identified. Resulting feature models provide common and variable characteristics of the systems in the domain in a way that customers can understand and select a set of features. Created feature models become input to other phases of domain analysis for generalization and parameterization of other models [20].

A feature model provides hierarchical representation of features in the domain. Each non-leaf feature contains a set of sub-features to provide more specific functionality. A simple introduction for feature modeling is provided in section 2.1.3.2.1 and more detailed analysis is provided in section 2.2. Therefore feature modeling information will be left to those sections.

In entity-relationship modeling activity, domain knowledge is modeled with a set of entities and relations among them. Most of the domain knowledge resides within source code and therefore it is not always possible to reuse that domain knowledge during analysis phase. Resulting model of the entity-relationship modeling activity provides a reusable set of models for domain knowledge hidden in the source code. Entity-relationship diagrams can be used to define domain elements and relations during functional analysis and architecture modeling.

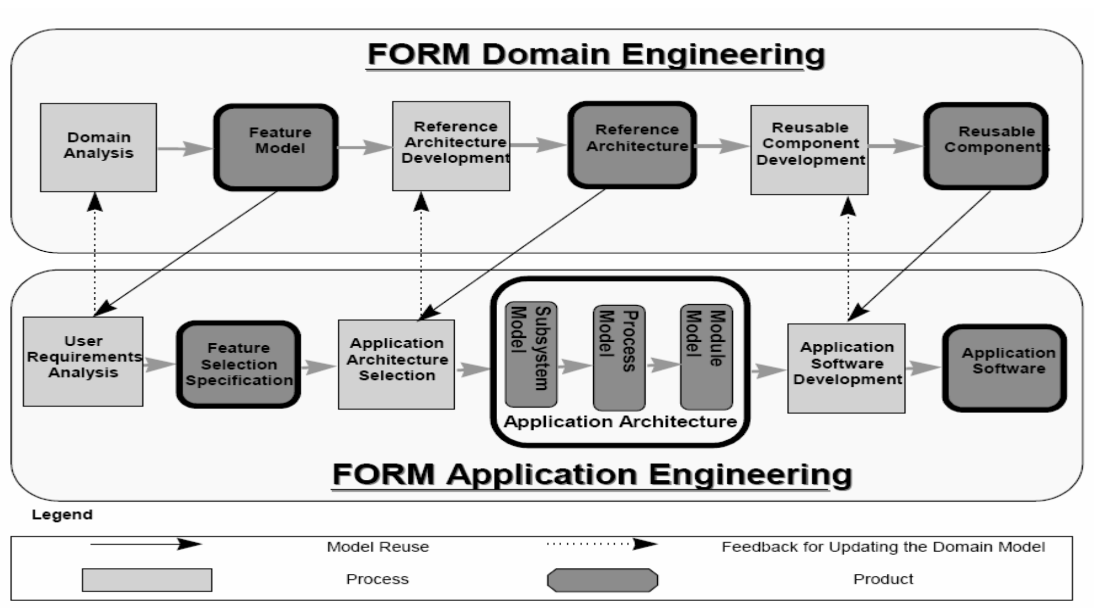
In functional analysis activity, operational details of the processes in the domain are analyzed to detail common and variable characteristics of the applications in the domain. Feature models and entity-relation diagrams are used to determine required elements of the domain for a specific functionality and relations among domain elements during the processing of that functionality. State diagrams and activity diagrams are widely used diagrams to constitute a complete functional model during functional analysis activity. Functional models can be reused during architecture modeling activity in FODA and application design of software development process.

2.1.3.3.2.3 Architecture Modeling

Architecture modeling phase is aimed to create an abstract design model to provide a reusable architectural design for the development of new applications in the domain. Architecture model is defined to be a solution to the problems defined in the domain model [20]. Architecture modeling tries to create a high-level design in order to provide more reusability. Output models of the previous activities are used, common and variable characteristics of the domain applications are analyzed, and an efficient layered structure is created at the end of the activity.

2.1.3.3.3 FORM (Feature Oriented Reuse Modeling)

FORM is an extension to the FODA methodology and has the same purpose of analyzing commonality and variability in a domain in order to develop domain architectures and components [33] to be used during application development. Similar to generative programming [22], FORM provides two levels for software product development; domain engineering and application engineering (Figure 5 [33]).



FORM Domain Engineering
Domain Analysis
Feature Model
Reference Architecture Development
Reference Architecture
Reusable Component Development
Reusable Components
User Requirements Analysis
Feature Selection Specification
Application Architecture Selection
Subsystem Model
Process Model
Module Model
Application Architecture
Application Software Development
Application Software
FORM Application Engineering
Legend
Model Reuse
Feedback for Updating the Domain Model
Process
Product

Figure 5 Software product development process in FORM [33]

In domain engineering phase, common and variable characteristics of the applications in the domain are modeled using feature models; then a set of reference architecture models are created. Common components are identified and described according to their roles and characteristics in each reference architecture model. As in FODA approach [22], FORM also includes context analysis, domain modeling and architecture modeling for domain engineering.

The core activity of the FORM is feature modeling where feature modeling “defines a decision space for application development” [33]. Features are categorized into four main groups; capability, operating environment, domain technology and implementation technique. The detailed definition of these feature groups will be provided in section 2.2.

In FORM, feature model is used during the architecture modeling activity. Generally functional features are used to define architectural components and non-functional features are used to define structural decomposition of components and connectors among components.

FORM defines three models for architecture modeling phase. Subsystem model defines the grouping of system functionalities into subsystems. Process model provides dynamic behavior for each subsystem defined in the subsystem model. Module model provides specification and abstractions of components in the domain. Each of these models is created using and analyzing feature models created in domain modeling phase.

During application engineering, user requirements are analyzed and a set of features are selected from the feature model. Resulting feature set is used to get the reference architecture model for the aimed application. Then, reusable components are identified and used in the development of the application.

2.2 Features and Feature Modeling

2.2.1 Definition

In section 2.1.3.2.1 feature is defined as a property or characteristic of a concept that is valuable from the point of relevant stakeholders' view. In this section, definition of the feature concept will be provided together with feature modeling activity.

In FODA, feature is defined as "a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems" [20]. Czarnecki redefines the feature as "an important property of a concept" where a concept can be any element or structure in the domain [22].

This definition of feature includes the expressiveness characteristics of a feature for the variable and common characteristics of concept instances in the domain. In section 2.1.3.2.1 base feature types (mandatory, optional, alternative, or) were introduced; more detailed explanation on these feature types will be provided while feature diagrams are explained.

Feature modeling can be defined as the activity of analyzing variable and common features of the systems in the domain and then creating a model based on this analysis. The resulting model is called a feature model consisting of detailed identification of features and their relations. In FORM feature model is defined as:

"A feature model, including feature definitions and composition rules, describes a domain theory. It not only includes the standard terms/concepts and their definitions, it also describes how they are related structurally and compositionally." [33].

2.2.2 Feature Diagrams and Feature Representations

Feature diagrams constitute an important part of a feature model. Feature diagrams are the place where common and variable features and their relations are represented graphically. In the remaining paragraphs of this section a commonly accepted definition and structure of the feature diagrams will be provided. [20, 22]

A feature diagram consists of nodes for features and directed edges for relations among features. Each node can contain one or more sub-nodes. A high-level node may represent a concept or feature in the domain. Edge types depend on the type of the feature and there are four types defined for edge types representing features; mandatory features optional features, alternative features and or features.

2.2.2.1 Mandatory Features

A mandatory feature provides an obligatory feature for the definition of its parent. Existence of a parent requires existence of its mandatory features and vice versa: existence of a mandatory feature requires existence of its parent, too. In feature diagrams, mandatory features are represented by mandatory edges connection to the feature node from its parent. A mandatory edge is a simple edge with a filled circle on its connection point to the mandatory feature.

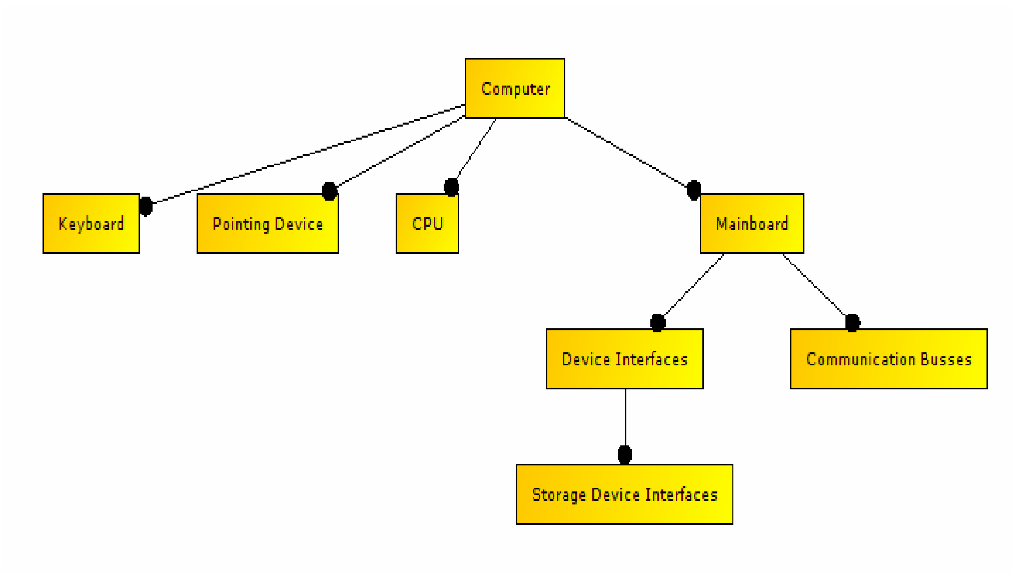


Figure 6 Example feature diagram for mandatory features

In Figure 6, a set of examples for mandatory features are provided. In this example, a partial feature diagram for a computer is provided. A functional definition of computer requires existence of a keyboard and a pointing device for user inputs. A mainboard and a CPU are required in order to process input and create outputs for the users. From the bottom up approach, device interfaces for a computer requires a mainboard to utilize communication among different devices and a mainboard is functional only if it is provided in a computer.

2.2.2.2 Optional Features

An optional feature provides a variable feature for the definition of its parent. Existence of a parent does not require existence of its optional features. Existence of an optional feature requires existence of its parent. In feature diagrams, optional features are represented by optional edges, connecting the feature node to its parent. An optional edge is a simple edge with an empty circle on its connection point to the optional feature.

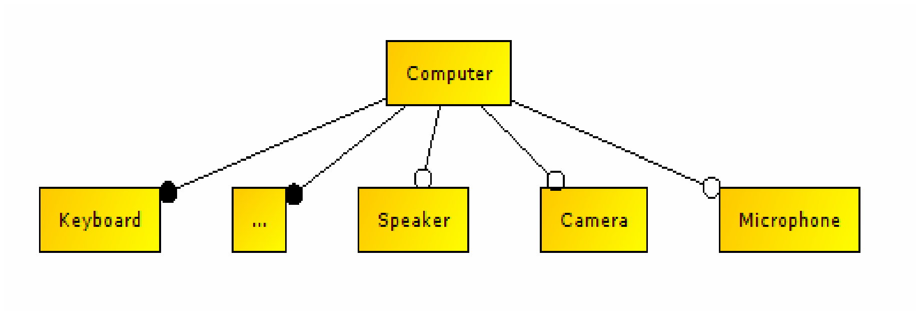


Figure 7 Example feature diagram for optional features

In Figure 7, a set of examples for optional features are provided for a computer. A functional definition of computer may include existence of a speaker, camera or a microphone. Although those features extend the capabilities of a computer, they are not required for the computer to provide its common requested operations. In this way, these features provide a set of variable instances of a computer. Each optional feature provides a different instance of the parent feature. In this example there are $2^3 = 8$ variable instances of a computer system.

2.2.2.3 Alternative Features

An alternative feature defines a variability point for its parent feature. It exists with a mutually exclusive relation with its siblings in the same alternative set. Existence of more than one feature in an alternative set is not allowed in an instance of an application in the domain. Therefore, only one feature from an alternative set can be selected. Existence of an alternative feature also requires existence of its parent feature. An alternative set is provided by an empty arc connecting edge of each alternative feature.

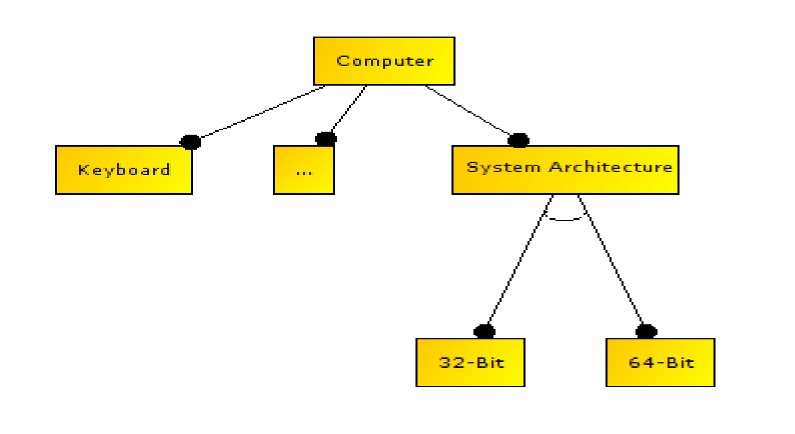


Figure 8 Example feature diagram for alternative features

In Figure 8, an alternative set example is provided for a computer system. Underlying architecture of a computer can be either a 32-bit architecture or a 64-bit architecture. It is not allowed for a computer to have both of the architectures in its hardware level for the assumed domain. Notice that the context of the system architecture feature in this diagram refers to the underlying hardware type the computer has, not the supported system types. Each alternative feature provides a variable instance for its parent feature. In this example system architecture for a computer system can be either 32-bit or 64-bit, providing two variable instances for the system architecture.

2.2.2.4 Or Features

An or feature is another variability point for its parent feature. In a set of or features, one or more of each feature can be selected at the same time. Different combinations of features in an or set provides variable instances of the parent feature. Existence of an or feature also requires existence of its parent feature. An or set is provided by a filled arc covering the edges of all the features that are in the "or" relation.

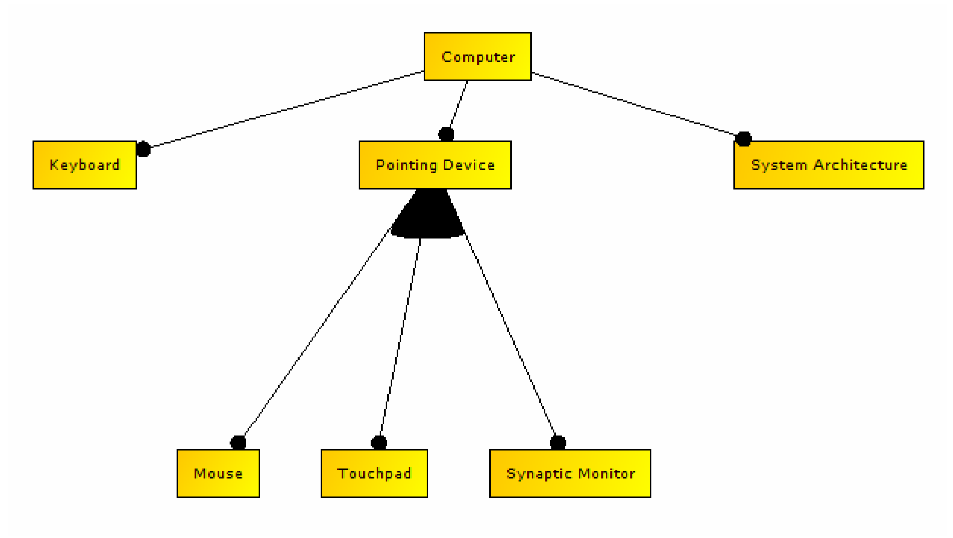


Figure 9 Example feature diagram for or features

In Figure 9, an or feature set example is provided for the pointing device feature of a computer system. Pointing devices in a computer system can be provided by any combination of mouse, touchpad or a synaptic monitor. Each of these alternatives provides required functionality from a synaptic device. Each combination of or feature provides a variable instance for its parent feature. In this example, pointing device functionality can be provided by $(2^3 – 1) = 7$ combinations of the or features it has.

2.2.2.5 Commonality, Variability and Constraints

Commonality in a domain is represented by common mandatory features in the feature diagram. A common mandatory feature is a mandatory feature that exists in all instances of applications created in the domain which means there is a path of mandatory features connecting the common mandatory feature to the root node. In Figure 6, all the features in the diagram are common mandatory features. Variability in a domain is represented by optional, alternative and or features. Each of these feature types defines variable instances for their parent feature.

In addition to constraints provided by feature types, there are two defined constraints among features. Requires constraint can be used in a way that is similar to mandatory features. The difference is that mandatory constraint can be applied to the features that have hierarchical decomposition semantics with its parent feature. When a hierarchical decomposition cannot be applied between two features and existence of one requires another then “requires” constraint is used. Requires constraint can also be used to define parameterized constraints in the domain (See Figure 2 in section 2.1.3.2.1). In the same manner, “mutually exclusive” constraint is used when an alternative feature cannot be used in order to prevent existence of two features in an instance at the same time.

2.2.2.6 Feature Categorization

In addition to feature types, FORM [33] defines a grouping strategy among features based on the “types of features they represent” [33, 34]. This grouping also defines a set of interested stakeholders for each group. There are four feature groups in FORM; capability features, operational environment features, domain technology features and implementation technique features. Below each of these groups are explained in detail.

- Capability Features: Capability features represent services, operations and functions provided by the system and non-functional characteristics of the system. Capability features can be divided into two categories; functional and non-functional, where non-functional features describe performance, quality or usage constraints. [33, 34]
- Operating Environment Features: This type of features is used to identify the characteristics of the underlying operational environment of the system. Underlying hardware, operating systems, database management systems, communication systems, file systems etc. constitute a whole operational environment and their attributes are represented by operating environment features. [33, 34].

- Domain Technology Features: This type of features is considered to be implementation level features. Any technology specific to the domain of focus is provided via domain technology features. An example might be frequency management algorithms for communication domain [33, 34].
- Implementation Technique Features: Similar to domain technology features, this type is also considered to be an implementation level feature type. However, in contrast to the domain technology features, this type of a feature is a common approach to the implementation technologies and can be generalized to other domains [33, 34].

An example feature diagram with these four layers is provided in [33].

2.3 Axiomatic Design

Design is one of the most important inputs for most high quality industrial products in different industrial fields, like automotive, electronics, software. An efficient design will lead to a high quality product and will increase profits of the company. On the other hand, it has always been a challenging activity to create an efficient and high quality design. Industrial experiences and research studies resulted with highly applicable and efficient strategies for product design. Axiomatic design is one of those proposed methods trying to increase the design quality via a well-defined methodology and a set of design principles. It has been applied to different industrial products and adapted to different fields as a plug-in to existing methodologies [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 36]. It should be noted that axiomatic design is not an overall complete methodology providing a complete set of well-defined activities for all phases of product development; it is a support methodology to aid design for development of new products by decreasing risks, cost and time for production.

Axiomatic design methodology is “a process to improve the quality and performance of complex system and product development” [5]. The main focus of the axiomatic design is to create the most efficient and highest quality design for a set of alternative design decisions and choices.

An important characteristic of the axiomatic design is the activity of traversing through output artifacts of the design process in both ways, forward and backward. In this way, efficiency of the decomposition and alternative decisions can be evaluated at each phase.

“There are four main concepts in axiomatic design:

- Domains
- Hierarchies
- Zigzagging
- Design axioms.” [5]

In axiomatic design, four domains are defined; customer domain, functional domain, physical domain, process domain. Each of these domains is a representation of a design activity and consists of customer needs, functional requirements, design parameters, and process variables, respectively. Customer needs provide a set of customer requirements. Functional requirements are the result of functional analysis based on customer needs and provide functional characteristics of intended design solution. Design parameters provide conceptual design elements of the product design and process variables define the production methodology for design elements [5, 36]. Below in Figure 10, four domains of axiomatic design are provided with the connection between these domains.

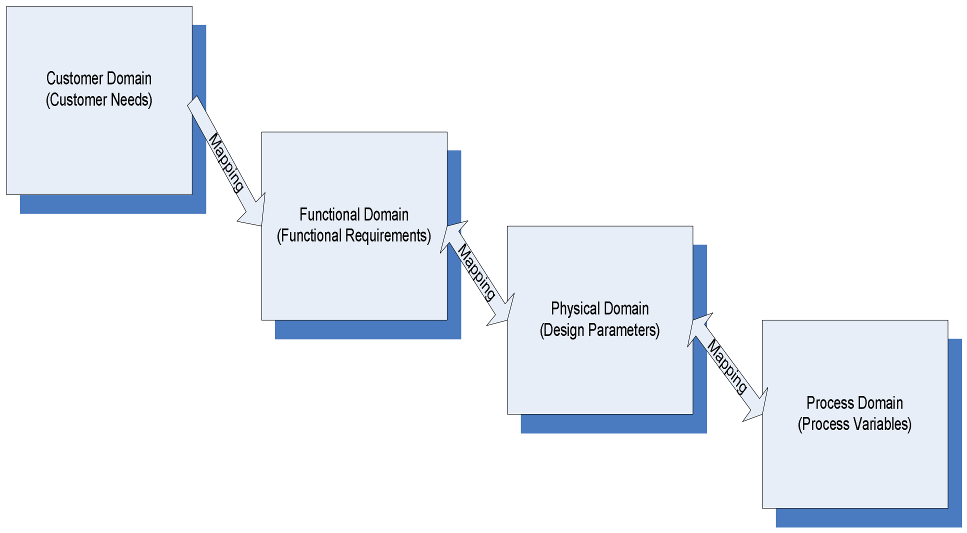


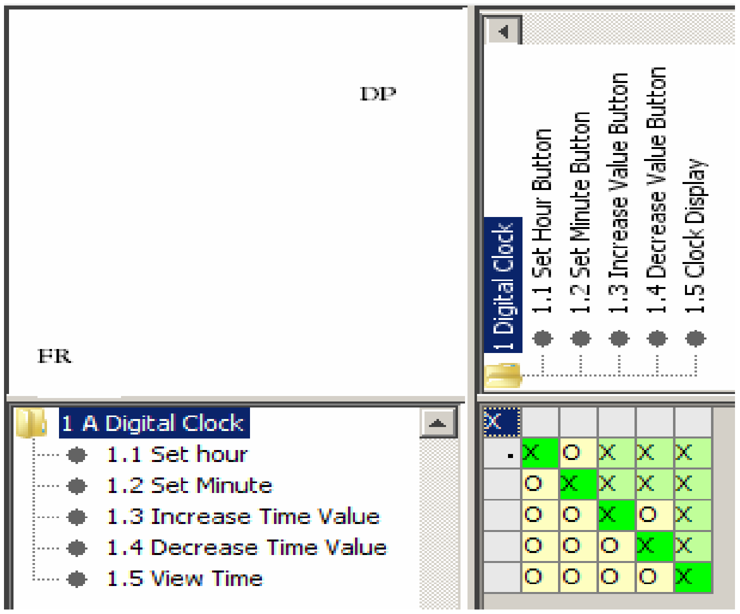
Figure 10 Axiomatic Design domains

It can be observed from the above figure that there is a connection between each adjacent domain. Each domain pair represents a set of partial problem-solution definition. For each pair the domain on the left defines the answer to the question of “what to achieve?” in this way defines a problem, the domain on the right defines the answer to the question of “how to achieve?” providing a solution to the problem [5, 36].

The process starts with the identification of the customer needs. After a clear identification of the customer needs, process starts with the definition and decomposition of high-level functional requirements. As soon as a functional requirement is defined, a design element is defined and mapped to that functional requirement in order to provide a solution for the problem of satisfying corresponding functional requirement. As the decomposition goes on, the designer traverses through each domain in both ways, which is called zigzagging. Decomposition for a functional requirement continues until the functional requirement has a well-defined design element to satisfy that functional requirement.

This thesis study focuses on two domains of axiomatic design; functional requirements and design parameters. Therefore following paragraphs of this section will focus on these two domains and will not detail the remaining domains of axiomatic design.

In axiomatic design, mapping information between domains is kept in a matrix structure called the design matrix. While functional requirements define the row elements, design elements constitute the corresponding column elements in the design matrix. This design matrix represents the relations among functional requirements and design elements. Below in Figure 11, a simple FR (Functional Requirements) – DP (Design Parameters) design matrix is provided as an example.



X					
	X	O	X	X	X
	O	X	X	X	X
	O	O	X	O	X
	O	O	O	X	X
	O	O	O	O	X

Figure 11 Example functional requirements-design parameters matrix

In this example, client needs a digital clock in order to see the time and set the time as needed. The whole system is decomposed into five functional requirements and each functional requirement is associated with a design parameter to satisfy that requirement. The design parameter for the whole system is defined to be the “Digital Clock”. “Set Hour” and “Set Minute” functional requirements are provided to define the functionality of setting hour and minute of the time and two design parameters are defined for those functional requirements as “Set Hour Button” and “Set Minute Button” respectively. Increase and decrease operations on numerical values of hour and minute values of the time are also provided with functional requirements and associated design parameters. “View Time” functional requirement is associated with the design parameter “Clock Display”. Each cell of the matrix are assigned to the value of “X” or “O” where “X” means that functional requirement defined on the row of the current cell is provided by the design parameter defined on the column of the current cell. For example; in order to provide the functional requirement “Set Hour”, system needs the design parameters “Set Hour Button”, “Increase Value Button”, “Decrease Value Button” and “Clock Display”. The matrix also represents dependencies among design parameters as well. In this example, “Set Hour Button” design parameter also depends on “Increase Value Button”, “Decrease Value Button” and “Clock Display” design parameters in order to fulfill its proposed functionality.

The latest concept of axiomatic design is design axioms. Design axioms in axiomatic design provide simple guidance for the definition and decomposition of the functional requirements and design elements. There are two design axioms in axiomatic design; independence axiom and information axiom.

Independence axiom aims to decrease the dependency among functional requirements. It states that each design parameter should be defined in a way that associated “functional requirement can be satisfied without effecting other functional requirements” [35]. Independence axiom also effects the dependency relations among design parameters. A coupled design does not satisfy independence axiom. Although uncoupled designs are the main purpose of the independence axiom, it is hard and sometimes impossible to create an uncoupled design. Therefore most of the designs created according to independence axiom are decoupled.

The importance of independence axiom and its importance can be clearly understood with an everyday example. A water faucet can be designed to control the temperature and flow of the water [5]. Consider the design of a water faucet with two handles. One handle sets the flow of the hot water; other one sets the flow of the cold water. In this way water temperature can be set by setting flow of the hot and cold water. In this design, each of the handles affects both temperature and flow of the water and therefore this design is coupled. Any change on the design of one of the functional requirements will affect the other one. Another design might be designing two handles according to two functional requirements. One handle is used to set the flow rate and another one is used to set the temperature. In this way, design parameters satisfying one functional requirement will not affect the other functional requirement. Any change on one of the design parameters will not affect other functional requirements.

The second axiom, information axiom, focuses on the information content of the design. It states that among alternative design decisions the best design is the one with minimum information content.

Integrating axiomatic design to other software methodologies has been studied and different approaches are proposed [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. Using axiomatic design as part of the object oriented methodology [10, 11] and integrating it to the component oriented software engineering [6, 7, 8] are two important approaches trying to adopt axiomatic design to software design processes.

ADo-oSS (Axiomatic Design of Object Oriented Software Systems) combines both the axiomatic design and object oriented programming infrastructure in order to provide a methodology for the design of object oriented systems. Process starts with the top-down approach of axiomatic design. A design matrix of functional requirements and design parameters are created by decomposing high-level functional requirements into lower level ones and defining design parameters for each functional requirement. The resulting design matrix is used to explore both high-level and detailed definitions of the modules of the system in a bottom-up manner. The process maps high-level functional requirements to high-level design elements (package, class, interface …etc) of object oriented technology. Leaf-level functional requirements are used to determine the behaviors (methods) of the design elements. Leaf-level design elements are mapped to data structures (fields) associated with the high-level design elements [11].

Axiomatic design has also been integrated into component oriented software engineering methods [6, 7, 8]. In these studies, axiomatic design is used to decompose the system into a set of functional requirements and design parameters based on the result of analysis results of customer needs and domain knowledge. During design parameter identification, current available components are reused and provided as a design parameter. If a design parameter cannot be associated with any available component; then, it is developed at the implementation phase. Creation of the design matrix also includes definition of the design elements (packages, interfaces, modules) of component oriented system design. After the design matrix is created, missing components are developed and system is created by integrating the components.

Adapting axiomatic design into existing methodologies provides an efficient decomposition mechanism for the design process and helps to determine the errors in the design at the early stages by analyzing coupling for each level.

CHAPTER 3

UTILIZATION OF FEATURE MODELING IN AXIOMATIC DESIGN

In this chapter, feature modeling is utilized in axiomatic design with a set of guidelines. First, rationale for this thesis study is provided. Afterwards, proposed approach is introduced and detailed as a process starting from domain analysis activity to the definition and decomposition of requirements.

3.1 Rationale

An ongoing research study, ADCO [6, 7, 8, 9], provides a methodology for component oriented software engineering. Proposed methodology uses axiomatic design activity as the base process for component oriented design. During research studies, it is observed that axiomatic design activity has a drawback. Customer needs does not have enough information regarding to the design. Therefore transition from customer needs to functional requirements is not efficient and might result with invalid system design. In order to overcome this drawback, in this thesis study, requirements definition and decomposition activities in axiomatic design are extended by utilization of feature models.

In ADCO, process starts by definition of customer needs. As soon as customer needs are identified, a corresponding mature domain is selected for the aimed application. Mature domain in ADCO refers to a field that has matured enough to have a set of components and component libraries to be used for the production of similar applications in that field. After a mature domain is selected for the application, axiomatic design activity starts [8].

In axiomatic design phase, functional requirements and their corresponding design parameters are identified and decomposed. During functional requirement definition, available components in the mature domain are analyzed and if a component in the mature domain satisfies the requirement; then the component is selected as a design parameter for that functional requirement. Decomposition of a functional requirement is bound to the availability of a component to satisfy that requirement. If a component is found, then the decomposition activity ends at that point. Otherwise the decomposition activity continues until all the child functional requirements are mapped to a component in the mature domain or decomposed enough to provide the design of the component to satisfy that requirement [8]. Process diagram for the ADCO is provided below in Figure 12 (modified from [8]).

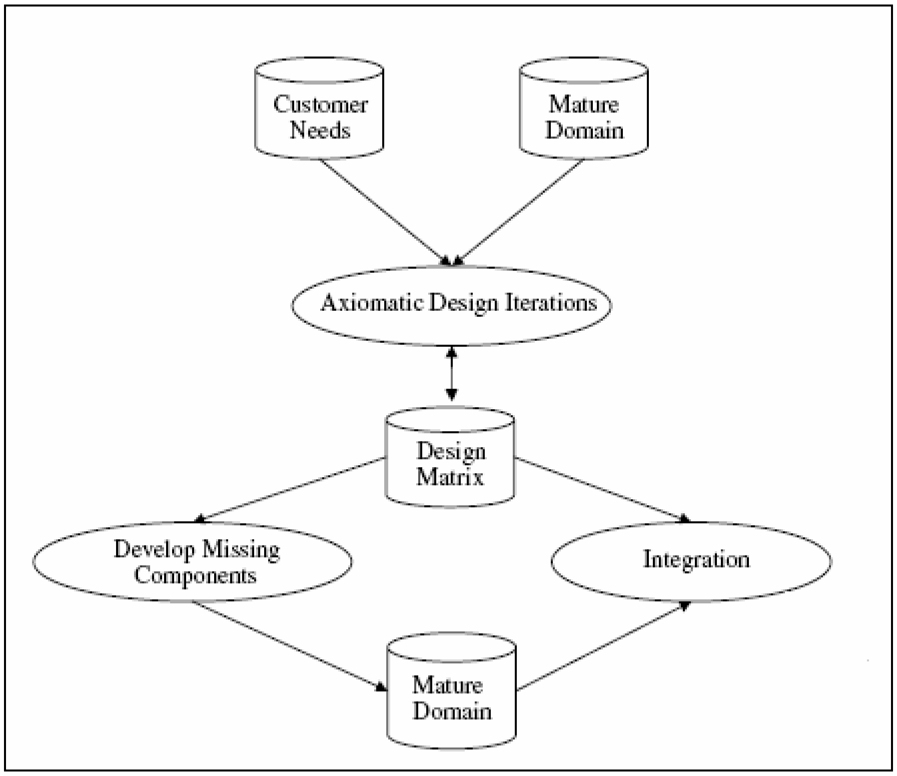


Figure 12 ADCO Process (modified from [8]).

Absence of well-defined principles or guidance during requirements definition and decomposition phase is a shortcoming for the axiomatic design. ADCO reuses predefined components or component libraries during the axiomatic design phase; however this reuse strategy does not support requirements identification directly. Since customer needs provide customer's view of the application, they are not sufficient enough to be used for the design process.

This thesis study aims to extend the requirement analysis phase in axiomatic design in order to:

- Reuse existing knowledge gained in domain analysis activity.
- Guide the designer to define requirements through the features identified in feature models.
- Guide the designer to decompose the requirements and in this way design parameters using the decomposition strategy applied to the feature models.
- Validate requirements and decomposition strategy through feature models.

Since ADCO process assumes the existence of a mature domain, this thesis study assumes existence of a domain analysis activity and its assets. Therefore a feature model for the domain of focus is assumed to exist. Proposed approach is provided below in Figure 13.

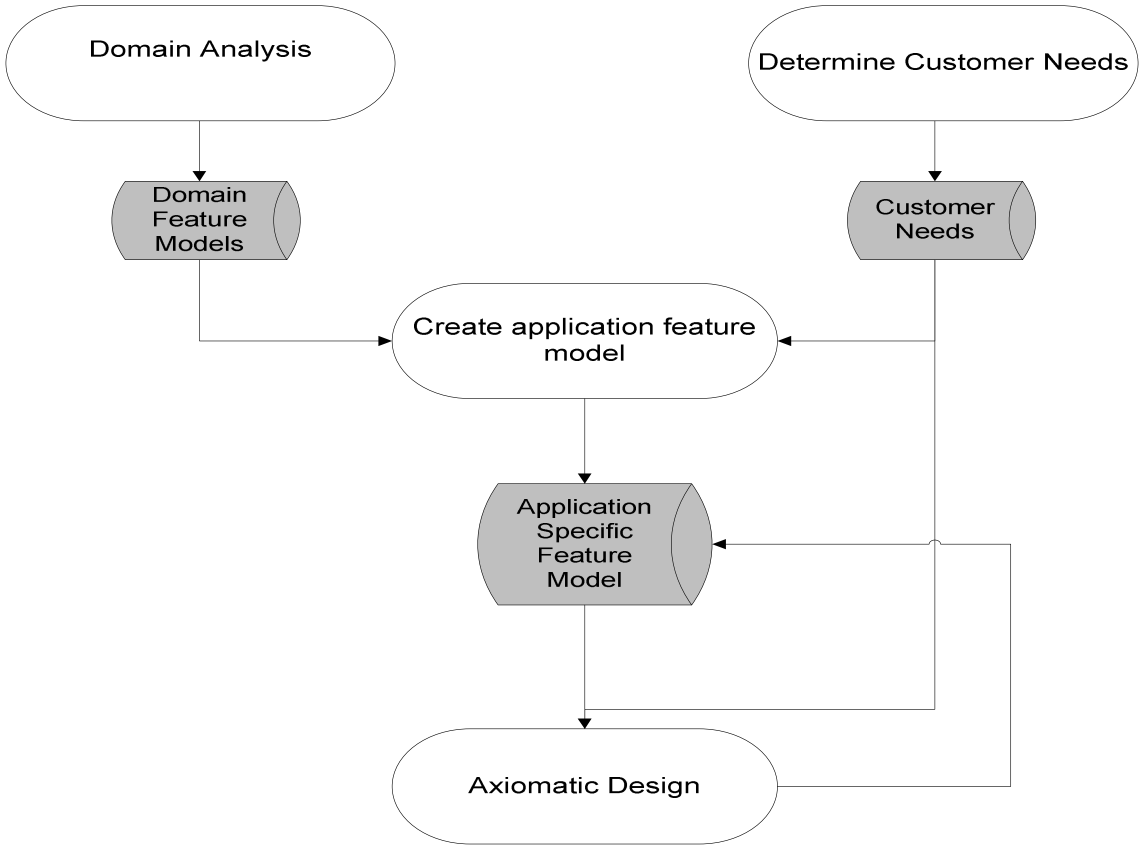


Figure 13 Proposed approach for requirements analysis

The first activity is definition of customer needs. Since customer needs provide the definition of the system from customer's point of view, it can be used to determine the domain of interest. As soon as the domain of interest is determined; domain-level feature models are used to create a feature model or a set of feature models specific to the aimed application. Identification of required features is guided using customer needs. After this phase, axiomatic design activity starts. At this point, the designer uses application specific features to define high-level functional requirements. It is possible that the designer defines high-level requirements independent from feature models and then validates the design decisions using feature models.

Decomposition strategy behind functional requirement hierarchy is either guided by feature hierarchy or validated using the feature model when appropriate. A list of guidelines constituting the approach is provided below:

- 1: Use customer needs for the determination of domain of interest and for creating application specific feature models.
- 2: Use feature models as well as customer needs during definition of new functional requirements.
- 3: During decomposition of functional requirements use similar hierarchy in feature models when appropriate.
- 4: Use feature model to validate defined functional requirements and decomposition of a functional requirement when appropriate.

A more detailed explanation for each activity and guidelines are provided in the next sections.

3.2 Functional Requirements Definition and Decomposition

3.2.1 Identifying Customer Needs

Identification of customer needs is the starting activity for axiomatic design. This activity can be performed using the principles provided for similar phases of existing methodologies. In general, meetings and interviews are performed together with the customer and other related stakeholders in order to define the characteristics of the system from user's point of view. Customer needs do not have enough information for the design process and only include customer's requirements for the application.

Customer needs will be used as an input for determining domain of interest and creating application specific feature models. Remaining sections will provide examples in order to increase understandability of the analyzed concepts and provided guidelines. To provide integrity, a partial functionality, radar visibility of a command control (C2) system, REMISSYS (Rescue Mission Support System) will be used as an example. The following paragraph will provide introductory information and customer needs in textual format for the radar visibility functionality.

Radars are the necessary hardware for all airplanes in order to provide visibility for the pilot and the crew. One of the shortcomings of a radar system in an airplane is that geographical obstacles (like mountains, hills … etc) prevent the visibility of the radar. This shortcoming is important when a military mission is planned for an aircraft. Therefore it is important to determine the invisible sectors of the overall geographic area in order to create efficient mission plans. Customer’s point of view for this functionality might be represented as:

“Calculate and view radar visibility for a geographical area.”

3.2.2 Feature Modeling

After customer needs are determined, the domain of interest is identified using customer needs. The next step will be analysis of domain feature models and then creation of application specific feature models. In this section; first, feature modeling tool, which is developed as part of this thesis study, is introduced and then creation of application specific feature model is provided.

3.2.2.1 Feat - A Feature Modeling Tool

Feat is a feature modeling and instance feature model creation tool to guide the users throughout the approach provided in this thesis study. The tool can also be used as a standard feature modeling tool. The symbols defined in [22] are used as a baseline and some modifications are applied for feature diagrams.

Using Feat, a complete feature diagram can be created to represent the features and relations among features. Detailed description and rationale for the selection of the features are also provided with the tool leading to the creation of a feature model either for a single system or a set of systems in the domain. Representation of feature model elements, their associated information and relations among them are provided in Table 1.

Table 1 Feature Model Element Representations in Feat

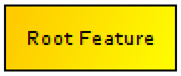
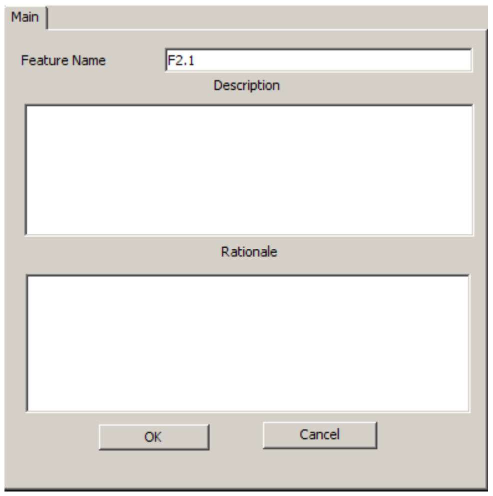
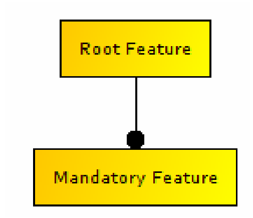
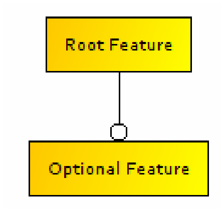
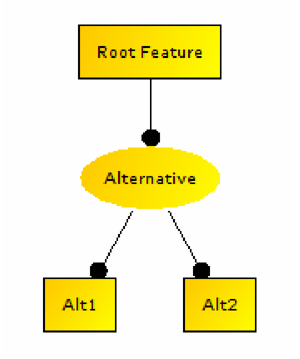
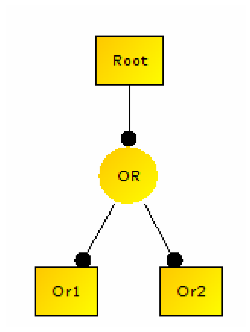
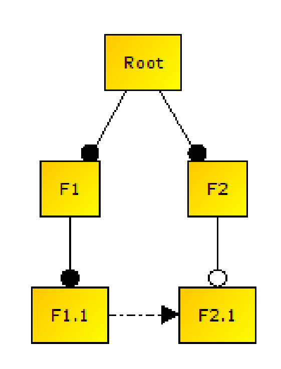
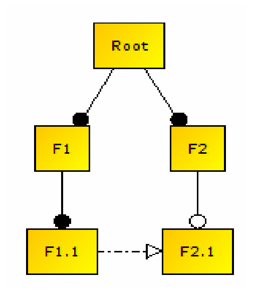
Feature Modeling Information	Symbol Used	Explanation
Features		Each feature is represented as a rectangular area. At the center of the rectangle resides the title of the feature.
Detailed Description and Rationale	 Main Feature Name F2.1 Description Rationale OK Cancel	Detailed description of each feature and rationale for the selection of each feature are provided with an editor dialog to the user.
Mandatory Features		Mandatory sub-features are represented by their edges. The edge connection point on the sub-feature has a filled circle to represent mandatory property.
Optional Features		Optional sub-features are represented by their edges. The edge connection point on the sub-feature has an empty circle to represent optional property.

Table 1 (continued)

Alternative Features		Alternative features are grouped into an alternative set. Alternative sets are represented by ellipses and each alternative feature is added as a mandatory sub-feature of the alternative set.
Or Features		Or features are grouped into an or set. Or sets are represented by circles and each or feature is added as a mandatory sub-feature of the or set.
Required Constraint		Required constraints between features are represented by dashed arrows with filled heads. The edge is directed to the target edge. In the example in the left cell, the existence of F1.1 requires the existence of F2.1.
Mutually Exclusive Constraint		Mutually exclusive constraints between features are represented by dashed arrows with hollow heads. The edge is directed to the target edge. In the example in the left cell, F1.1 can not exist together with F2.1.

Feat tool provides the users to create feature model instances for new applications in the domain (Figure 14). Feature model instance creation activity in Feat is supported by automated rule enforcement mechanism that checks the relations and constraints among features and enforces them during the activity.

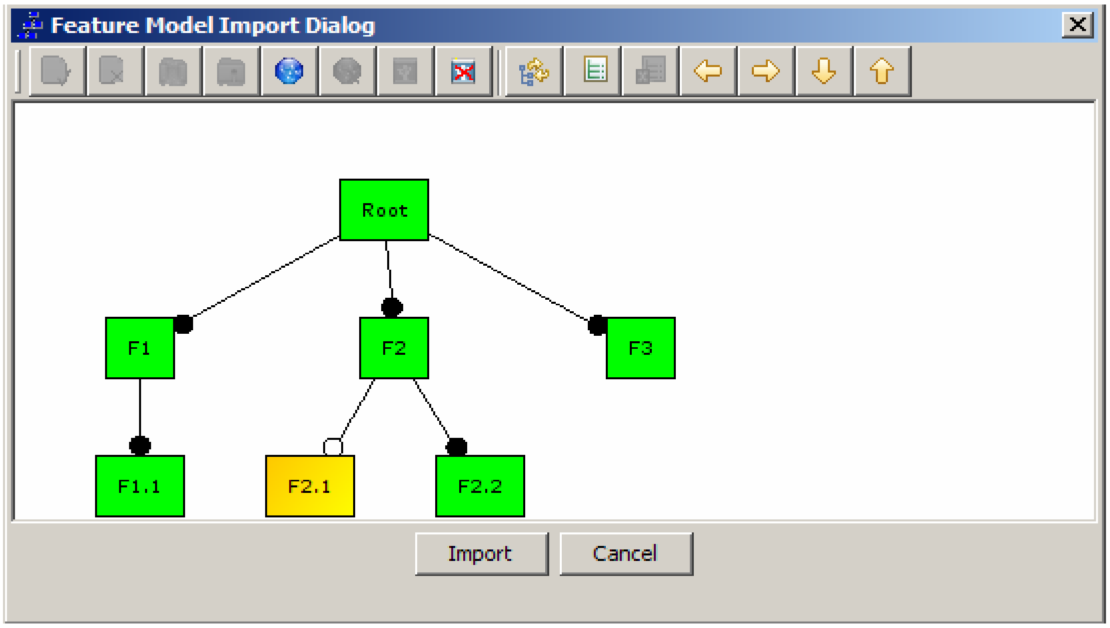


Figure 14 Application specific feature model creation in Feat

3.2.2.2 Creating Application Specific Feature Models

As soon as domain of interest is determined, domain-level feature models and customer needs are used in order to create application specific feature models. At this phase domain-level features are analyzed and selected based on the required functionalities defined in the customer needs. During this activity, the constraints and relations among features are taken into consideration. Common functionality represented by mandatory features are always included in the feature model instances; however alternative functionalities are selected based on the required functionalities of the aimed application.

Each customer need might be represented by a set of features and each feature might represent a set of customer needs. It is important to determine the scope of the feature model based on functionalities required for the application. Selected features should provide all the functionalities defined in the customer needs and redundant features should not be selected.

A partial feature model of GIS applications in command control systems domain is provided in Figure 15. The focus of this partial feature model is visibility feature for a command control system. As it can be seen from the figure, visibility feature requires existence of geometric layer support. Other related constraints are also satisfied in the feature model by mandatory features.

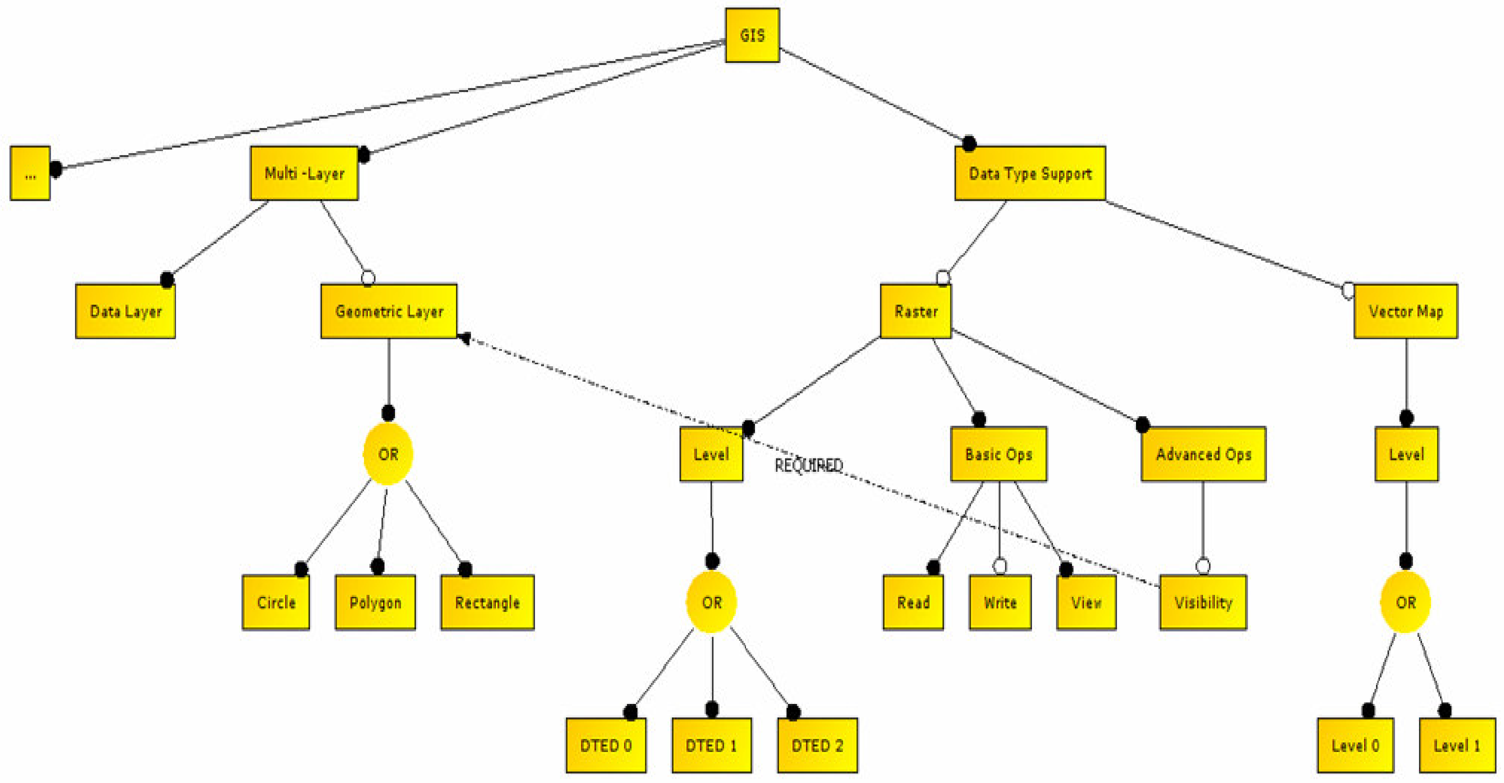


Figure 15 Partial feature model of GIS applications in the Command Control Systems domain

During application specific feature model creation for REMISSYS, visibility feature in the domain feature model is selected based on the customer need provided in section 3.2.1. It is trivial to determine that visibility feature represents the required functionality provided in textual customer need. Partial application specific feature model for the REMISSYS is provided in Figure 16.

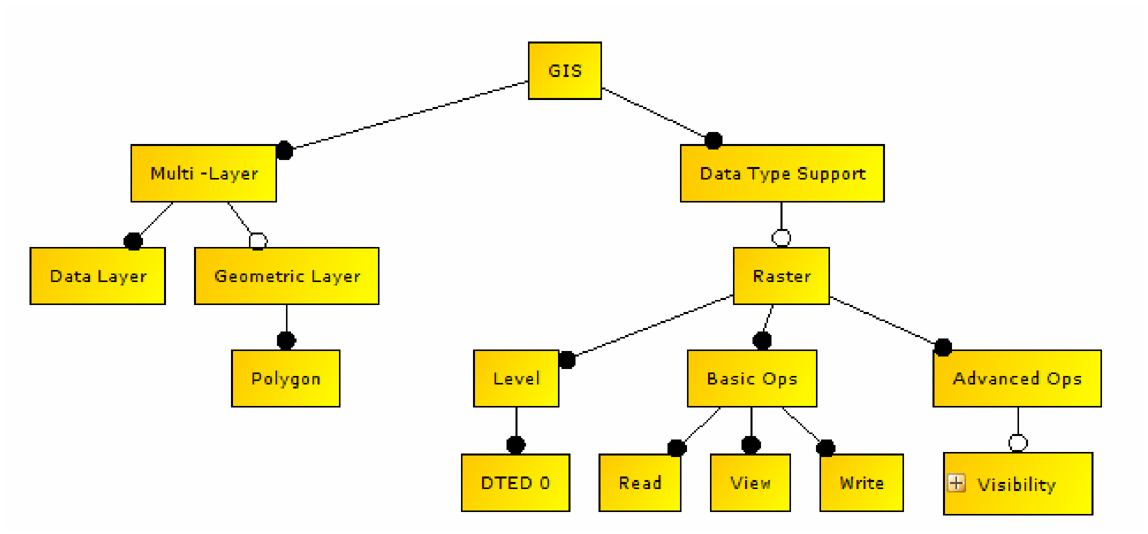


Figure 16 Partial feature model for REMISSYS application

Selection of the visibility feature resulted in the selection of polygonal geometric layer feature for the GIS application. Notice that geometric layer support was an optional feature; however since visibility feature requires geometric layer support (see Figure 15), existence of geometric layer feature is required for the visibility feature.

It can be seen that applying feature modeling for the application provides more detailed information regarding to the design decisions. In this way, more efficient utilization of customer needs for definition of functional requirements at the design phase is provided via feature modeling.

3.2.2.3 Functional Requirements

Identification of functional requirements is important in axiomatic design in order to create efficient and valid designs. Therefore it is of utmost importance for a designer to be able to analyze customer needs and create corresponding functional requirements; however customer needs lack the information regarding the design. At this point, feature models are utilized. Identification of functional requirements is supported by feature models which are created in consistence with customer needs and have more information regarding the system design. Feature models support the functional requirement identification activity in four ways, regarding four guidelines provided in this thesis study:

1. Identification of a new functional requirement
2. Decomposition of a functional requirement
3. Validation of a functional requirement definition
4. Validation of a functional requirement decomposition

These four activities will be supported by feature models at specific steps of an axiomatic design activity and explained in the next sections.

3.2.2.3.1 Level of Detail in Features and Requirements

In feature modeling, features are the basic elements to define commonality and variability among applications in the domain. Higher level features define the common characteristics of the applications. Decomposing high-level features into lower level features reveals variable characteristics of the common features in different applications. There are no well-defined boundaries or rules for the depth of the decomposed feature model and the level of the detail in feature modeling, therefore each user requirement might not be represented properly in the feature model. Level of detail has not been discussed in feature modeling in a well-defined way, representing the requirements with detailed constraints on the functionality might cause the feature model to grow to an unmanageable size. In this way, complexity of the diagram increases and the model loses its purpose. Therefore, level of detail in feature models might be different than the level of detail provided in requirements.

An example can be more appropriate to understand this. Visibility functionality of a GIS system in the command control domain can be represented similar to the one in Figure 15 or the one in Figure 17. Both feature models include visibility functionality as a feature; however the second one decomposes the feature into more sub-features in order to provide more details about the feature. When it comes to the requirements, we can have different requirements to define functionality within the system. In our case, alternative requirements with different levels of detail might exist for different systems in order to define required visibility functionality as provided below:

1. "The system should provide visibility analysis for a given geographical area and aircraft altitude. Results should be viewed as a polygonal raster layer."
2. "The system should provide visibility analysis for a given geographical area, aircraft altitude and target altitude. Results should be viewed as a polygonal raster layer."
3. "The system should provide visibility analysis for a given center point, radius, aircraft altitude and target altitude. Results should be viewed as a circular raster layer."

Each of these requirements may be used in a system's requirements document. The first requirement defines only two parameters for the required functionality. A previously defined geographical area and an aircraft altitude is input to the system and the system is supposed to calculate the visibility and then view the results as a polygonal raster layer. This is the most basic and common functionality of a GIS application for radar visibility analysis. However, similar systems might require different parameter sets for the same functionality. The second requirement adds another parameter for the visibility functionality where a target's visibility is analyzed. In the third requirement, previously defined generic area is replaced with radius and center point parameters defining a circular area and the resulting raster layer is required to be circular.

In the feature model provided in Figure 15, visibility feature is not decomposed into lower level features. When visibility functionality is needed, selection of this feature will not be clear enough to represent the requirements provided above. In its current state, this feature does not represent any of the requirements provided above with the level of detail provided in the requirement.

In Figure 17, visibility feature is decomposed to provide more detail about the feature. Notice that the feature model provided in this figure is able to represent all the requirements provided above. A design will be able to define and decompose the requirements for visibility functionality more efficiently by analyzing this feature model.

Above examples are provided to point to the variability of the level of detail in requirements and feature models. During specific application production activities, it is important to define requirements within the scope of the system and providing enough detail about the required functionality. Features might not be detailed enough for this purpose. Therefore, requirements identification activity should be performed with detailed analysis of feature models and customer needs.

3.2.2.3.2 Identification and Validation of Functional Requirement Definitions

At the beginning of an axiomatic design activity, designer defines high-level requirements. Definition of these high-level requirements also defines high-level decomposition of the system in terms of functional requirements. At some point during design activity, new requirements are defined as a result of decomposing higher level requirements. Requirement identification in both levels can be guided using feature models.

At each level, if feature model provides enough information regarding the requirement; designer can define high quality functional requirements by analyzing features in the feature model. Different conditions may appear during this activity.

1. A feature can lead to the definition of a new requirement: The feature itself can be directly used to define a new requirement. After an analysis on feature model of REMISSYS application in Figure 16, below requirements can be defined considering Multiple-Layer support feature as below;

“System will provide multiple-layer support for presentation of different data groups.”

“System will show radar specific data, planning data, and operator specific data in different layers.”

2. A set of features can lead to the definition of a new requirement: Considering Figure 16, DTED0 feature and raster read feature can be combined in a requirement as:

   “System will be able to read DTED0 data.”

3. A feature can lead to the definition of a new set of requirements: Consider the data layer feature in Figure 16. This feature might be provided as it is. In that case, designer might want to define different requirements regarding the data layer features for each data type for the system as provided below:

   “An operator can create a new VMAP layer for vector maps.”

   “An operator can create a new raster layer.”

   “DAFIF data will be viewed by a specific layer.”

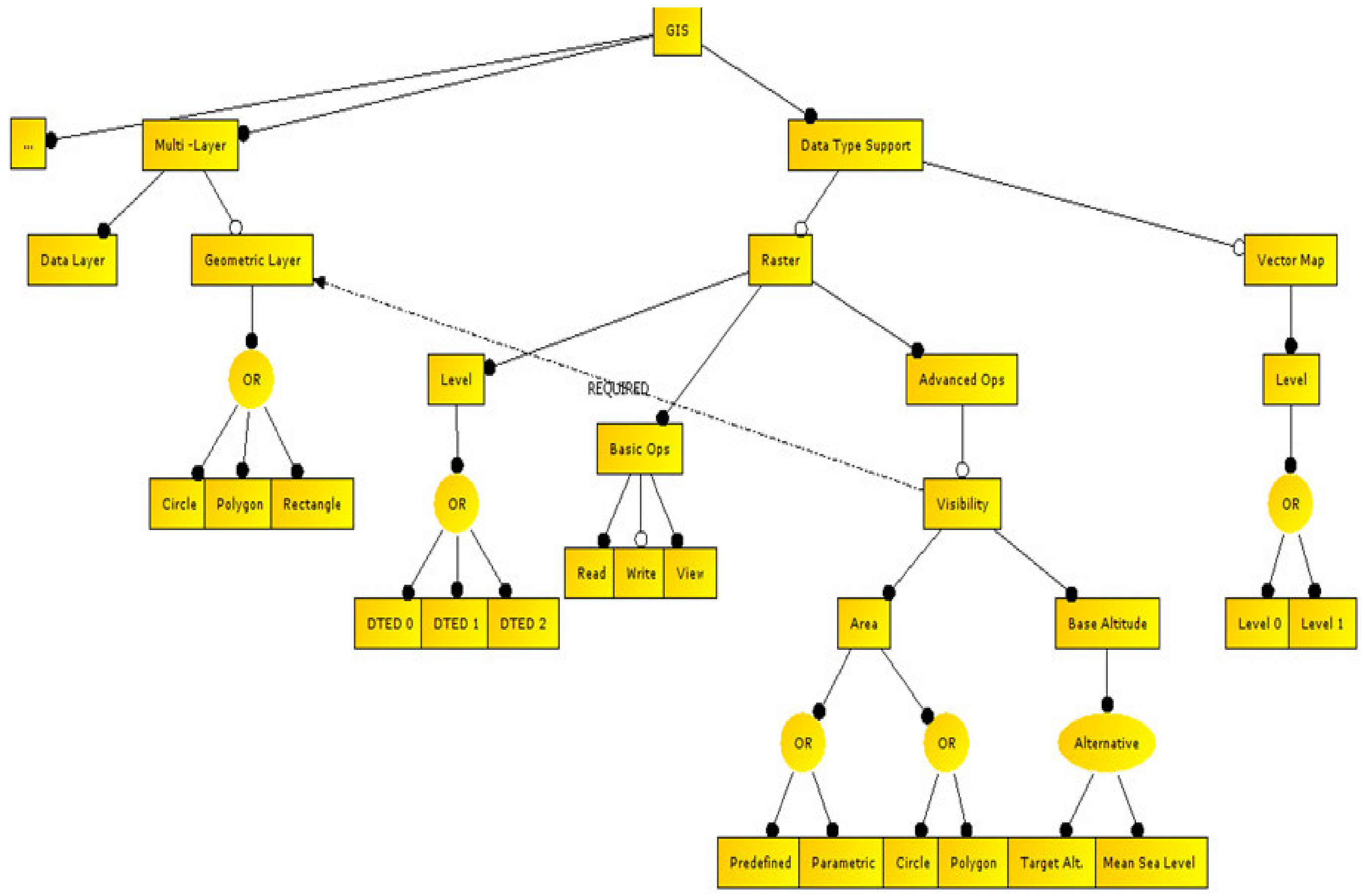


Figure 17 Partial feature model of GIS domain - Visibility feature detailed

Three conditions provided above might appear at specific time of the design activity based on the content provided by feature models and customer needs. Detailed analysis on both feature models and customer needs will guide the designer to define valid and efficient definitions of functional requirements.

An important property of feature models is that they provide constraint definitions among features. During requirements identification, these rules and constraints can help the designer to select efficient design decisions. For example; in the feature model in Figure 15, visibility feature requires geometric layer support feature. Therefore, when feature model specific to the REMISSYS application is created (Figure 16), geometric layer support feature is selected as a mandatory feature. During requirements identification, design will be guided to select geometric layer support as it is provided as a mandatory feature. Otherwise, required selection of the geometric layer feature might remain unnoticed leading to invalid and incomplete designs at earlier stages.

During design activity, some of the functional requirements can be defined as a result of design element mapping. Those kinds of definitions can be validated using feature models. Definition of a new requirement might be modified or might lead to the modifications or creation of new functional requirements due to relations and constraints in the feature model. For example, the design element A is defined for the functional requirement X and that A requires B to function properly. Therefore design element B is also added to the design matrix. Addition of design element B requires definition of a functional requirement Y. At this phase, validation of definition for functional requirement Y can be done by analyzing the feature model and customer needs.

3.2.2.3.3 Decomposition and Validation of Decomposition

Decomposition is an important step as its main purpose is to reduce the complexity of the system. Reduced complexity leads to the development of high quality systems. During axiomatic design designer decomposes high-level functional requirements into lower level functional requirements in synchronization with the decomposition of design elements. Functional requirement decompositions can be guided by feature models.

Feature models contain domain-level logical decomposition of the overall system. Similar functionalities are grouped under the same parent feature. In this way, a designer can use decomposition at the feature model as a starting point for all functional requirement decomposition activities. There are two conditions during this activity:

1. Decomposition in feature model is valid and efficient for the design activity: If the designer finds the decomposition in the feature model efficient, then he uses the similar decomposition in the functional requirements. Children features will be used to define new functional requirements. For example in Figure 18 a feature model for REMISSYS application is provided. In this feature model visibility feature is decomposed to provide more detail about the feature. The similar hierarchy can be used to decompose the requirements as in Figure 19.

2. Decomposition in feature model is not efficient for the design activity: If decomposition in the feature model is not appropriate to be used for the design, then the designer chooses another alternative decomposition for the design or does not decompose the functional requirement at all. For example, assume an existing component satisfies the visibility functionality with target altitude as a parameter and polygonal area as a result. Then, there is no need to further decompose corresponding functional requirement and the designer might define the functional requirement as provided below:

   "System should provide visibility calculations for a geographical area based on a target altitude. Results should be provided as a polygonal area."

   Notice that the above functional requirement contains details (polygonal result, target altitude) about the functionality in its own description. However, in the feature model the details are provided as children features of the visibility feature.

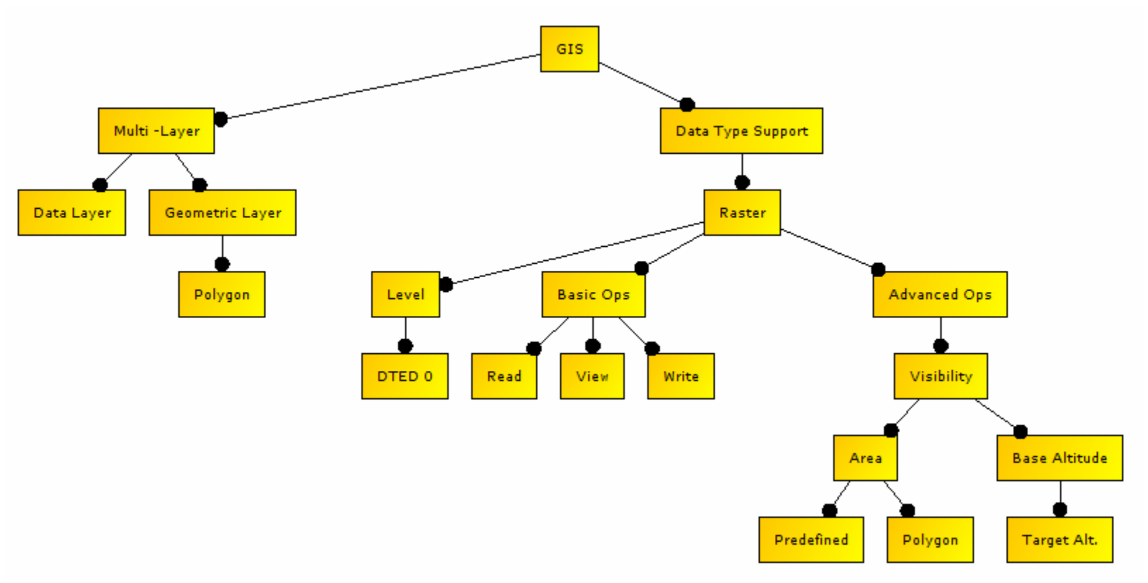
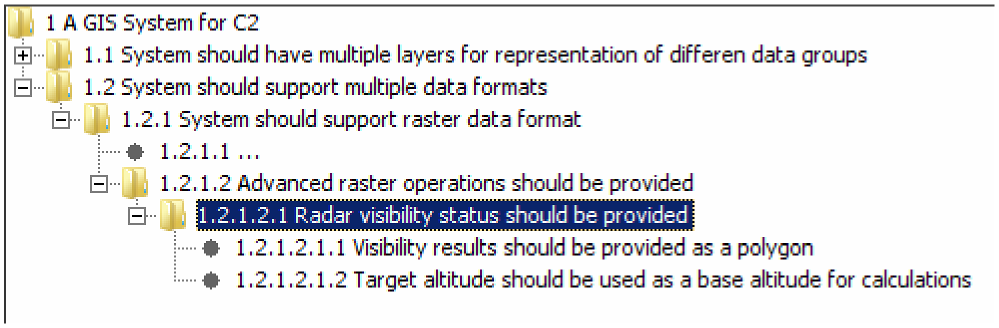


Figure 18 Partial feature model for REMISSYS application (Visibility detailed)



1 A GIS System for C2
1.1 System should have multiple layers for representation of differen data groups
1.2 System should support multiple data formats
1.2.1 System should support raster data format
1.2.1.1 ...
1.2.1.2 Advanced raster operations should be provided
1.2.1.2.1 Radar visibility status should be provided
1.2.1.2.1.1 Visibility results should be provided as a polygon
1.2.1.2.1.2 Target altitude should be used as a base altitude for calculations

Figure 19 Partial functional requirement decomposition for visibility in REMISSYS

During design activity, some of the functional requirements can be decomposed as a result of design element mapping during zigzagging in axiomatic design. Those kinds of decompositions can be validated using feature models when appropriate. For example, a design element A is defined for a functional requirement X and A is decomposed to design elements A1 and A2 as a design choice. At this phase, functional requirement X should be decomposed to functional requirements X1 and X2. Efficiency of this design decision can be validated by analyzing decomposition in the feature related to functional requirement X and modifications can be applied using similar decomposition in the feature models.

CHAPTER 4

CASE STUDY: REMISSYS APPLICATION

In order to analyze benefits and drawbacks of the provided guidelines, an example process will be provided for a basic rescue mission support system for aircrafts, REMISSYS (Rescue Mission Support System) application. A set of customer needs will be provided. Then, an application specific feature model will be created for REMISSYS. Customer needs and application specific feature model will be used during axiomatic design activity. Results will be discussed in the last chapter.

4.1 Customer Needs

Customer needs for the REMISSYS application are provided below in Table 2.

Table 2 Customer Needs for REMISSYS

Customer Need ID	Text
CN 0	System will be used for aircraft mission planning for rescue operations.
CN 1	System should provide visualization of geographic information.
CN 1.1	Visualization of political boundaries.
CN 1.2	Visualization of geographical formations, like rivers, mountains …etc.
CN 1.3	Visualization of elevation for different world representations (spheroids).
CN 2	System should provide visualization of civilian tactical assets for a geographical area.

Table 2 (continued)

CN 2.1	Visualization of structures, like hospitals, schools …etc.
CN 2.2	Visualization of transportation assets, like airports, roads, highways …etc.
CN 3	System should provide visualization of military assets or entities for a geographical area.
CN 3.1	Visualization of structures, like military hospitals, headquarters, barracks …etc
CN 3.2	Visualization of defensive assets; like SAM (Surface to Air Missile) sites, naval weaponry, military aircrafts …etc.
CN 4	User should be able to define new tactical assets for further use by modifying existing data.
CN 5	User should be able to perform airspace control planning by analyzing existing civilian aircraft routes and by defining tactical routes for the rescue operation.
CN 6	User should be able to manage information content regarding to mission planning by creating tactical points, areas and borders and assigning functional properties to them.
CN 7	User should be able to analyze the availability of the civilian and military assets for a rescue operation.
CN 8	System should be able to provide analysis of transportation alternatives for a rescue mission.
CN 9	System should be able to parse xml formatted messages supporting rescue missions.
CN 9.1	Support for parsing existing NATO messages.
CN 9.2	Support for introducing new xml formatted national messages.
CN 9.3	Support for ICAO and METAR messages
CN 10	System should provide visibility analysis for a geographical area.

4.2 Application Specific Feature Model

Customer needs of REMISSYS contain information related to military domain and geographical information systems domain. Since domain modeling is a detailed and time consuming activity; partial domain analysis is applied for these two domains. Scope of the domain analysis is limited to the subjects related to REMISSYS and corresponding partial feature models are created for these two domains.

4.2.1 Partial Feature Model for Command Control (C2) Systems

Figures 20, 21 and 22 provide different parts of the created feature model of C2 systems. C2 system features are divided to two main groups, tactical elements and mission planning. Tactical elements represent the logical grouping of the data related to most command control systems. Identification of each tactical element is required for command control systems. There are two types of tactical elements, structures and weaponry. Variable types of these tactical elements are provided in Figure 20.

Mission planning feature is decomposed into six sub-features in Figure 21. Basic operations feature is provided to represent simple operations for mission planning. A command control system might provide mission planning activity for different types of missions. Each mission might be performed on one or more of the ground, airborne or naval platforms. Airspace management functionality is decomposed of route and corridor management. As airspace management is not needed for grounds based or naval based missions, it is provided as an optional feature. Route management is provided by definition of the flight routes for the aircrafts joining the mission. Flight route analysis provides the consistency of the defined routes by analyzing existing civil and military aircraft routes. This feature might not be needed for all mission planning activities and therefore left as an optional feature. Air corridor management defines capability of visualization and modification of specific air corridors which are bound by international laws. The number of assets joining the mission and relations among them brings necessity for communication planning. Communication planning is divided into two sub-features as in Figure 21. Management of communication radio terminals are separated from management of link network terminals since these two groups of terminals are related to two different concepts in the command control domain.

Each mission planning activity might be supported or scoped by different sources of information. Different parties might provide information enclosed in messages as provided in Figure 21. Messages might be provided by civilian or military sources. Most nations create their own message formats for military operations. Nations participated to NATO also supports NATO specific message formats. Two types of messages from civilian agencies are provided in this feature model, ICAO and METAR. ICAO provides route information for civil aircrafts and airports. METAR provides weather information for civil airports. Message might be provided in different formats. Since xml has been popular for data communications, most of the messages are either in xml format or can be converted to xml format. Some messages might be in binary or plain text format. Some of the current systems also support IRIS format which is the format used in IRIS message management software.

Message processing strategy is divided into three features. In direct processing, processing details are provided at system implementation phase. Therefore modifications and extensions cannot be applied at runtime by the user. In order to add new message types, new implementations should be performed and system should be build again. Extensible processing strategy provides run time configurability for message processing. In this strategy, most of the configurations are applicable at run time. Although it supports to configure existing message processing at run time, it needs implementation for new message types. Declarative message processing strategy provides full extensibility for the message processing functionality by reflection technology introduced recently. In this strategy, a common infrastructure is provided for processing messages in the same format. Extension with new messages and modification of existing message processing functionality can be performed at run time by user defined configuration files.

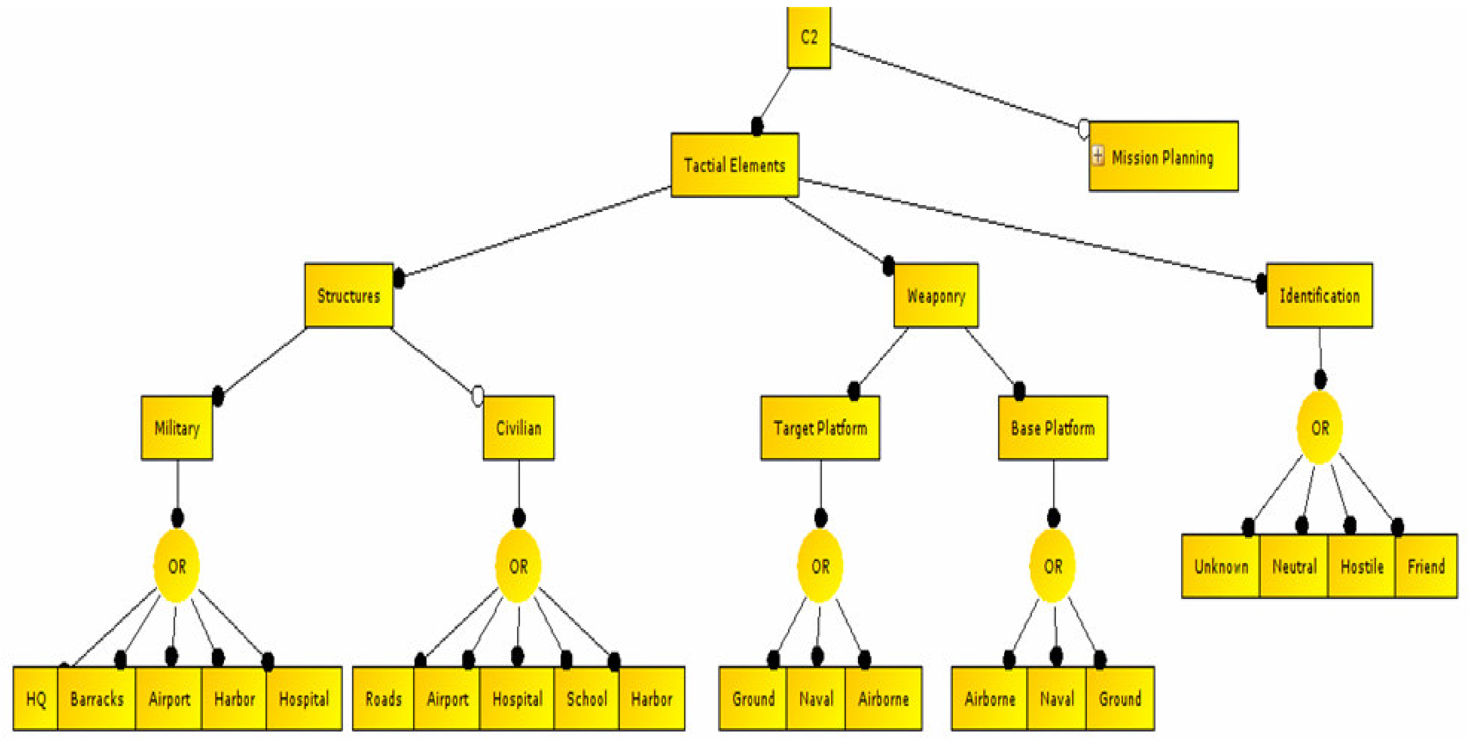


Figure 20 Partial Feature Model of Command Control Systems - Part 1

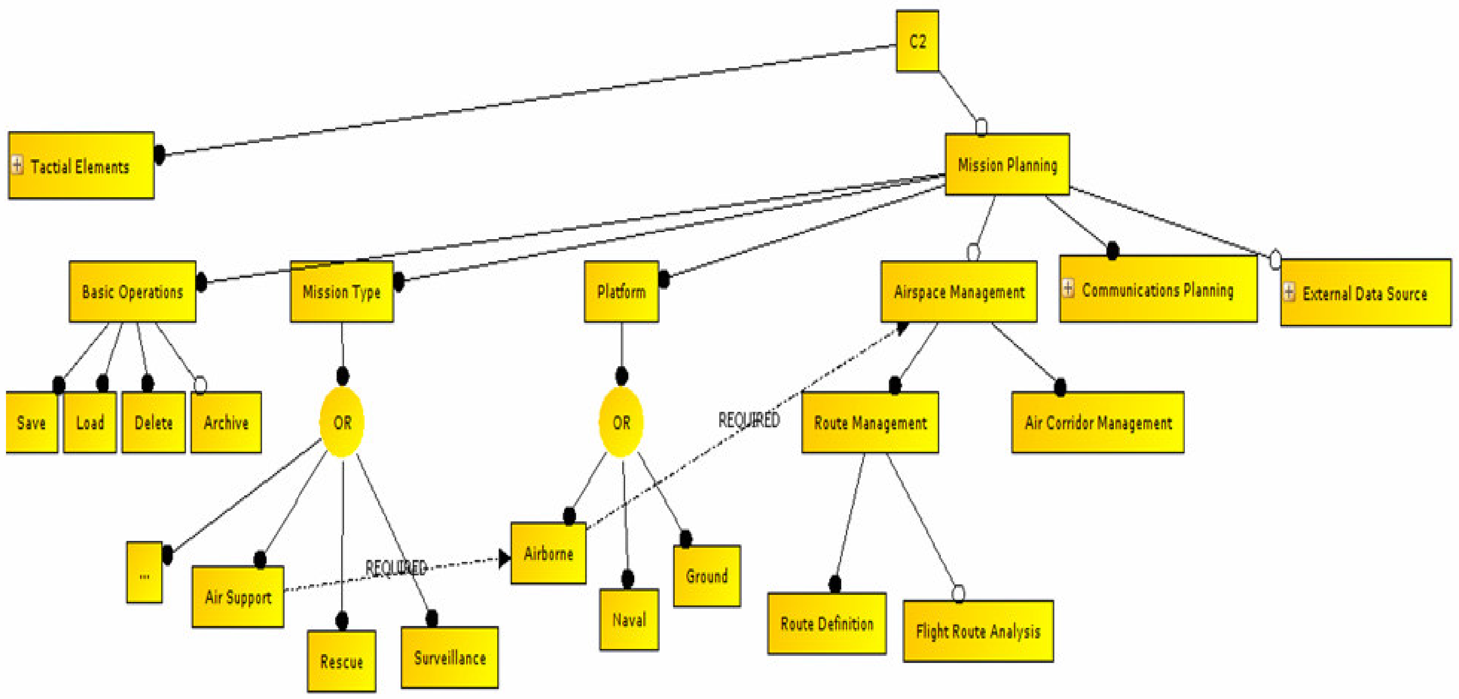


Figure 21 Partial Feature Model of Command Control Systems - Part 2

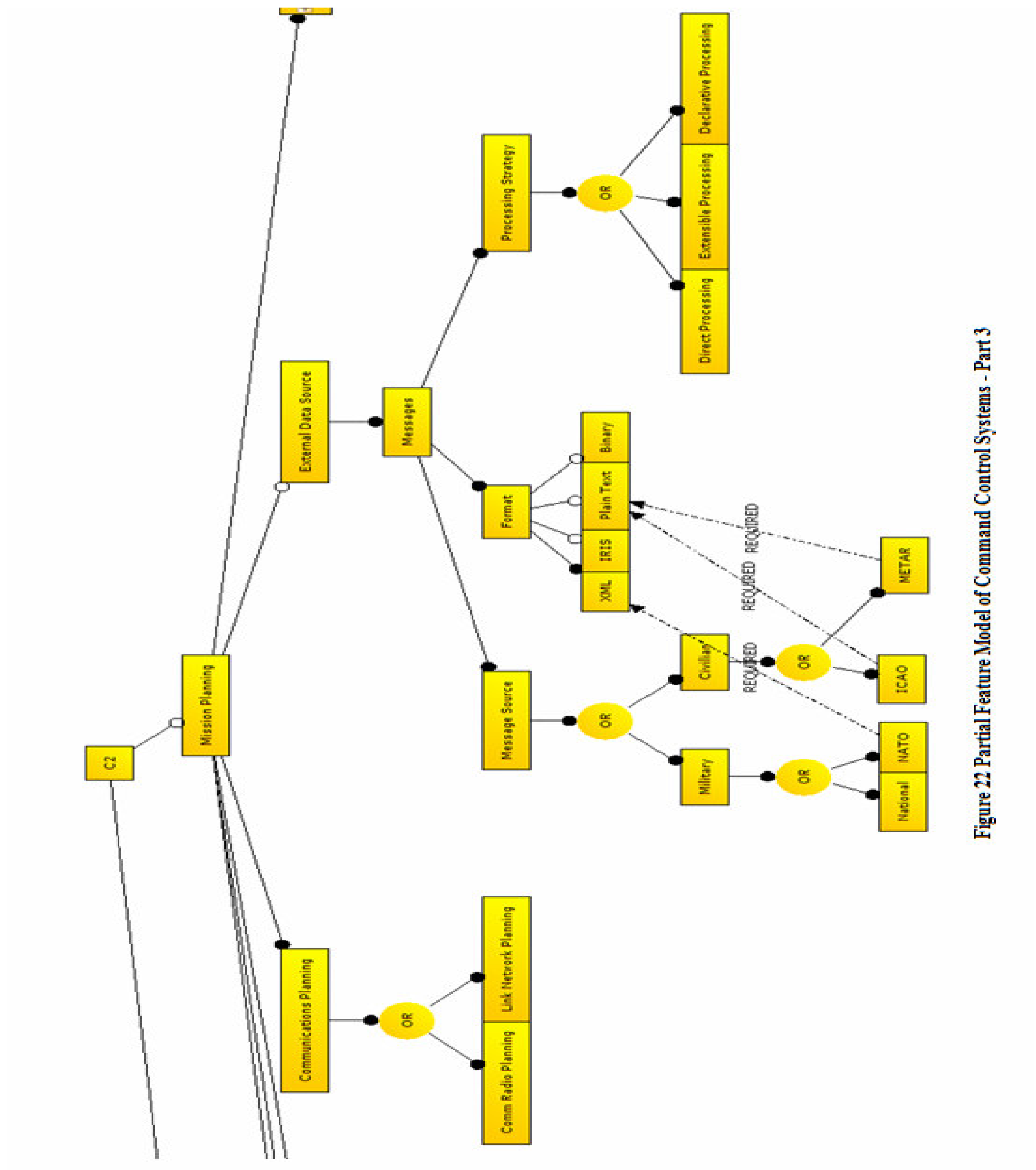


Figure 22 Partial Feature Model of Command Control Systems - Part 3

4.2.2 Partial Feature Model for GIS Applications

Figure 23, 24 and 25 provide created feature model for GIS applications. Data type support is an important characteristic of a GIS application since there are multiple different data structures used for geographic visualization. In Figure 23, three base data structures are provided as vector, triangulated irregular network (TIN) and raster. Vector data models represent geographic information with points, lines and polygons. Raster data provides geographical information as a table of cells and TIN visualizes the earth as a network of connected set of triangles. Basic operations for vector data should be provided in a GIS applications. Some GIS applications also provide advanced functionalities for vector data. Query functionality provides querying for geographical information; routing provides routing algorithms and dynamic editing provides an extensibility mechanism by modification and addition of geographical data to existing vector data sources. In common, vector data is static and does not change; however some systems require modification of the vector data and therefore these kinds of systems require dynamic structure for the vector data. There are multiple vector data file types and some of them are provided in Figure 23, generally GIS systems support most of these file formats.

In Figure 24, raster data format is detailed. Data file formats and basic operations are provided. Most GIS applications also support advanced operations on raster data. In this feature model visibility, projection and transformation is provided. Visibility functionality calculates the visible areas for a geographical area from a reference altitude. Projection capability provides valid visualization of raster data on different geodetic representation alternatives and transformation provides conversion of raster data to other data formats.

Figure 25 provides multiple layer support and map management features. Map management is provided in all GIS applications and it provides simple operations for geographical maps. In some GIS applications, merge functionality is provided for the creation of new maps by merging data from different data structures. Most GIS applications provide data structure support by providing multiple layers for visualizations. Geometric layering is commonly used for visualization of user defined geographical data and provided by advances GIS applications.

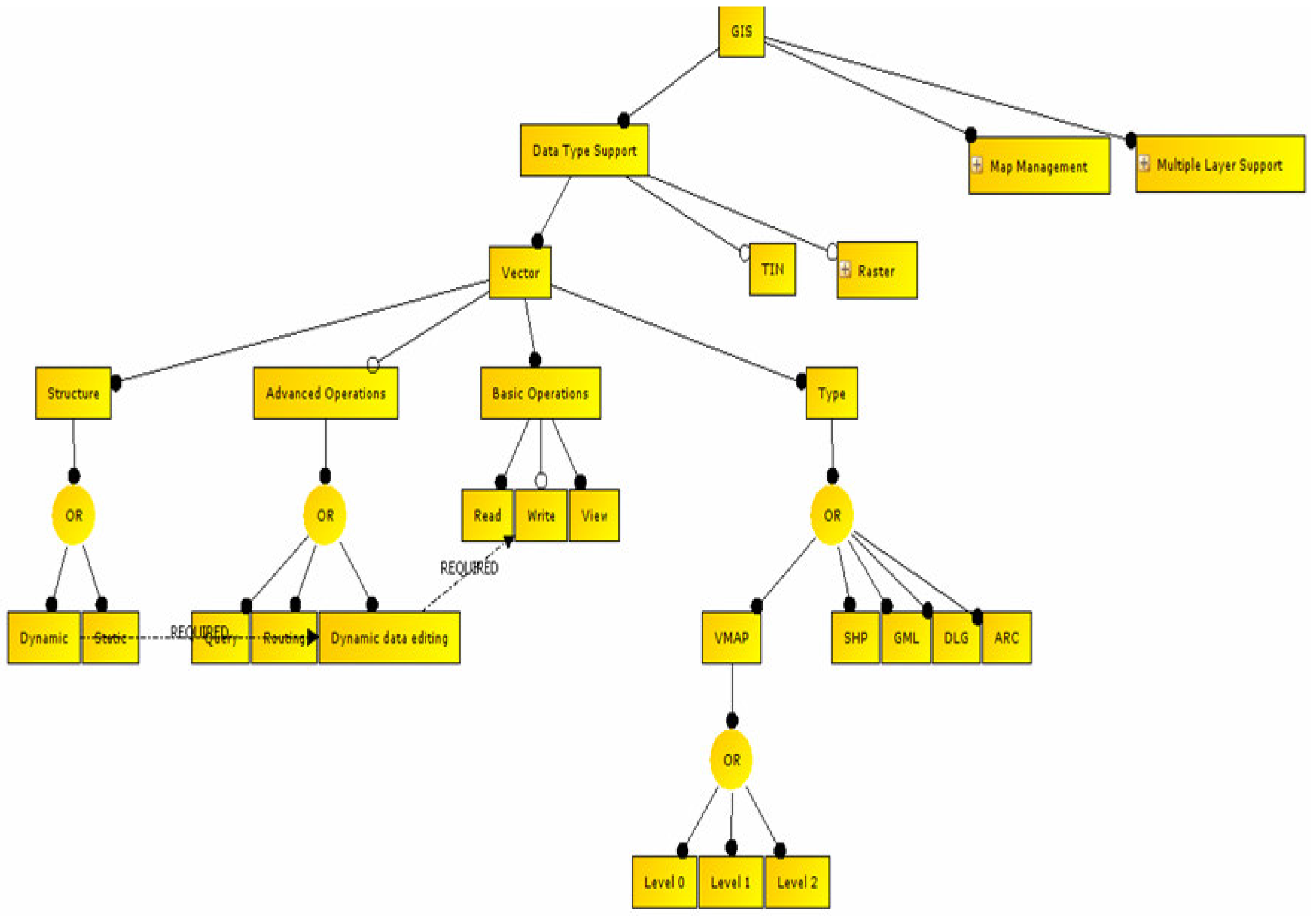


Figure 23 Partial Feature Model of GIS Domain - Part 1

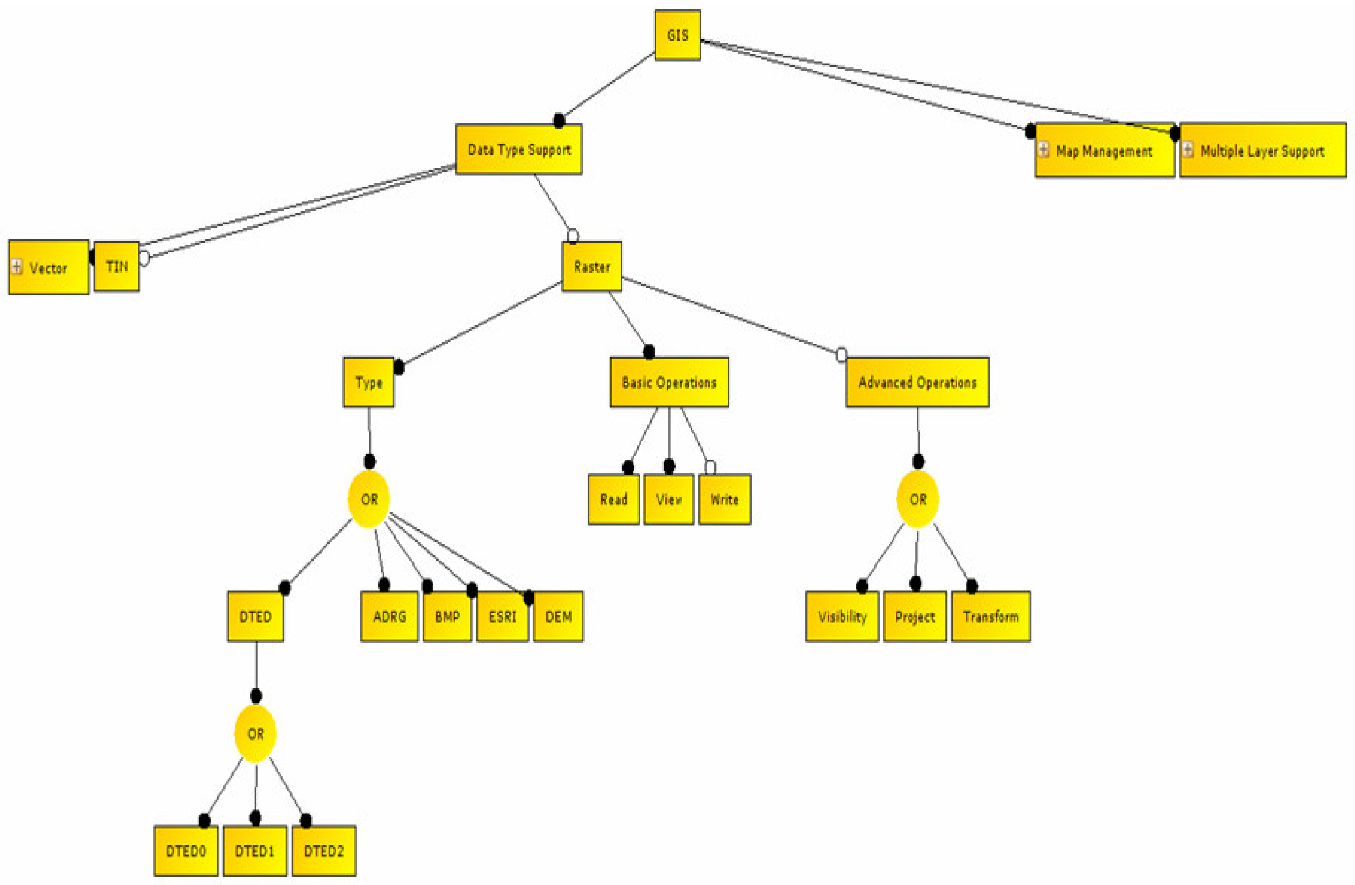


Figure 24 Partial Feature Model of GIS Domain - Part 2

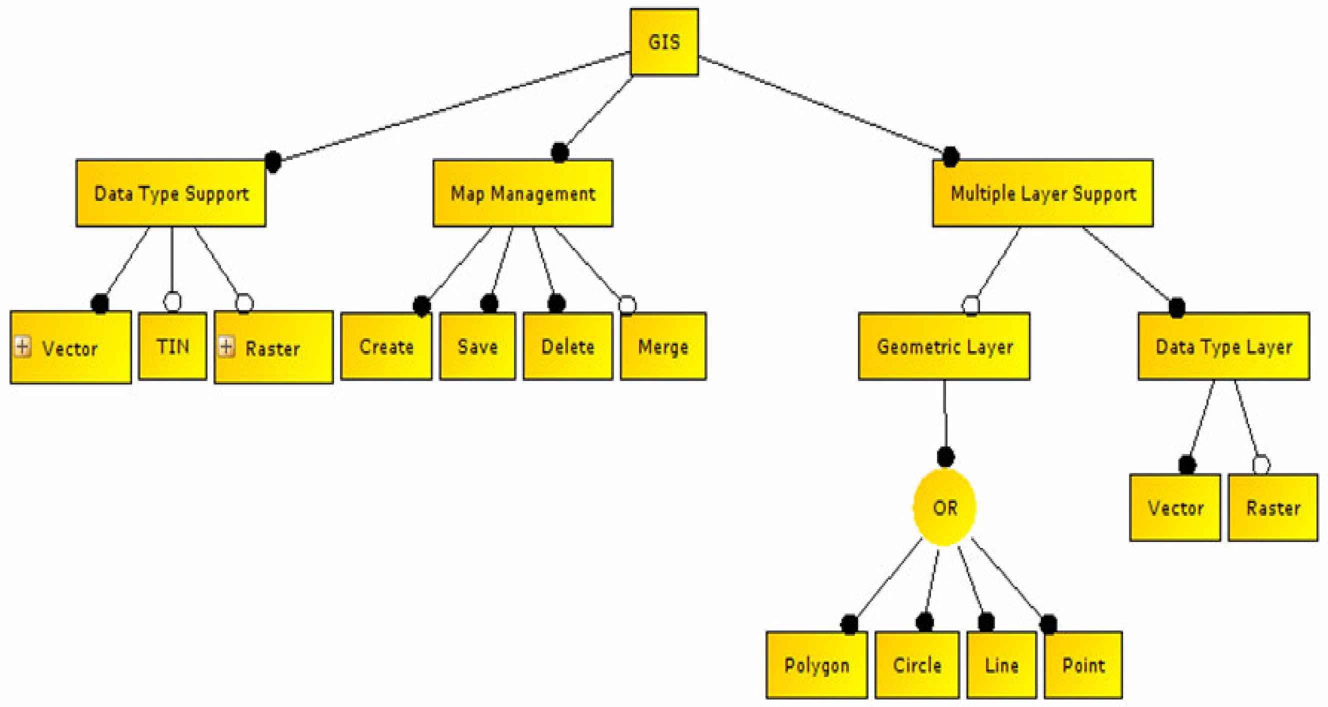


Figure 25 Partial Feature Model of GIS Domain - Part 3

4.2.3 Application Specific Feature Model for REMISSYS Application

Detailed analysis of customer needs and feature models of the C2 and GIS domains resulted with the creation of application specific feature model for REMISSYS as provided in Figures 26, 27, 28 and 29. Mandatory features have been selected by default. Variable features which are optional or exist within "*alternative*" or "*or*" set have been selected according to customer needs. Variable features and related customer needs are provided for clarification in Table 3.

Table 3 Customer needs for selected variable features

Customer Needs	Feature
CN 0	Rescue mission, airborne platform, communications and link network planning.
CN 1, CN 1.1, CN 1.2, CN 2, CN 3	Vector data types, map merge
CN 1, CN 1.3, CN 10	Raster data types, advanced raster operations (Project, Transform), raster data layer
CN 4, CN 6	Dynamic vector data structure, dynamic vector data editing, write (vector), geometric layers
CN 5, CN 7, CN 8	Vector data query, routing
CN 2	Civilian structures
CN 3, CN 3.1	Military structures, tactical element identification types
CN 3, CN 3.2	Weaponry for all platforms, tactical element identification types

Table 3 (continued)

CN 5, CN 6, CN 7, CN 8	Airspace, route and corridor management features.
CN 9	External data source, messages
CN 9.1, CN 9.2	Military message formats, XML message format, declarative programming for message processing.
CN 9.3	Civilian message formats, plain text messages

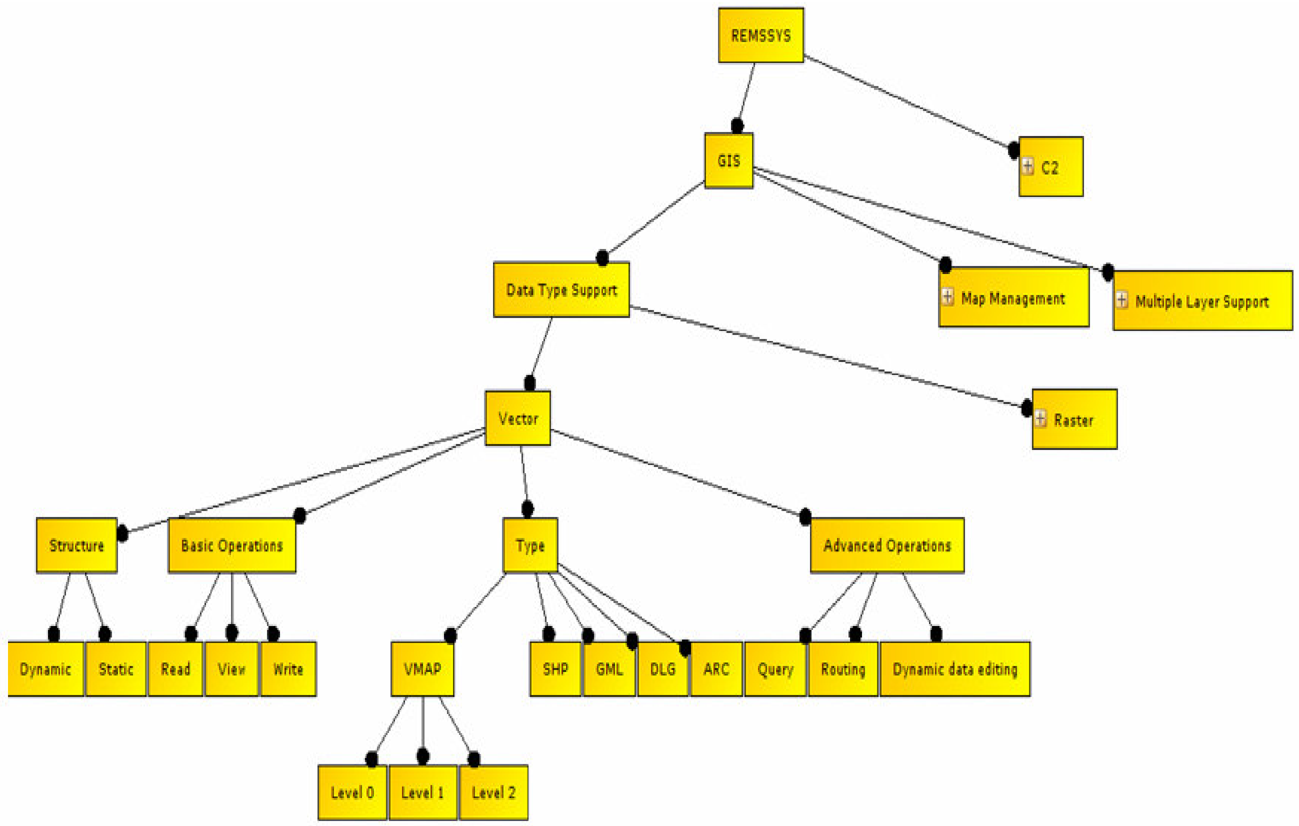


Figure 26 Feature Model of REMISSYS Application - Part 1

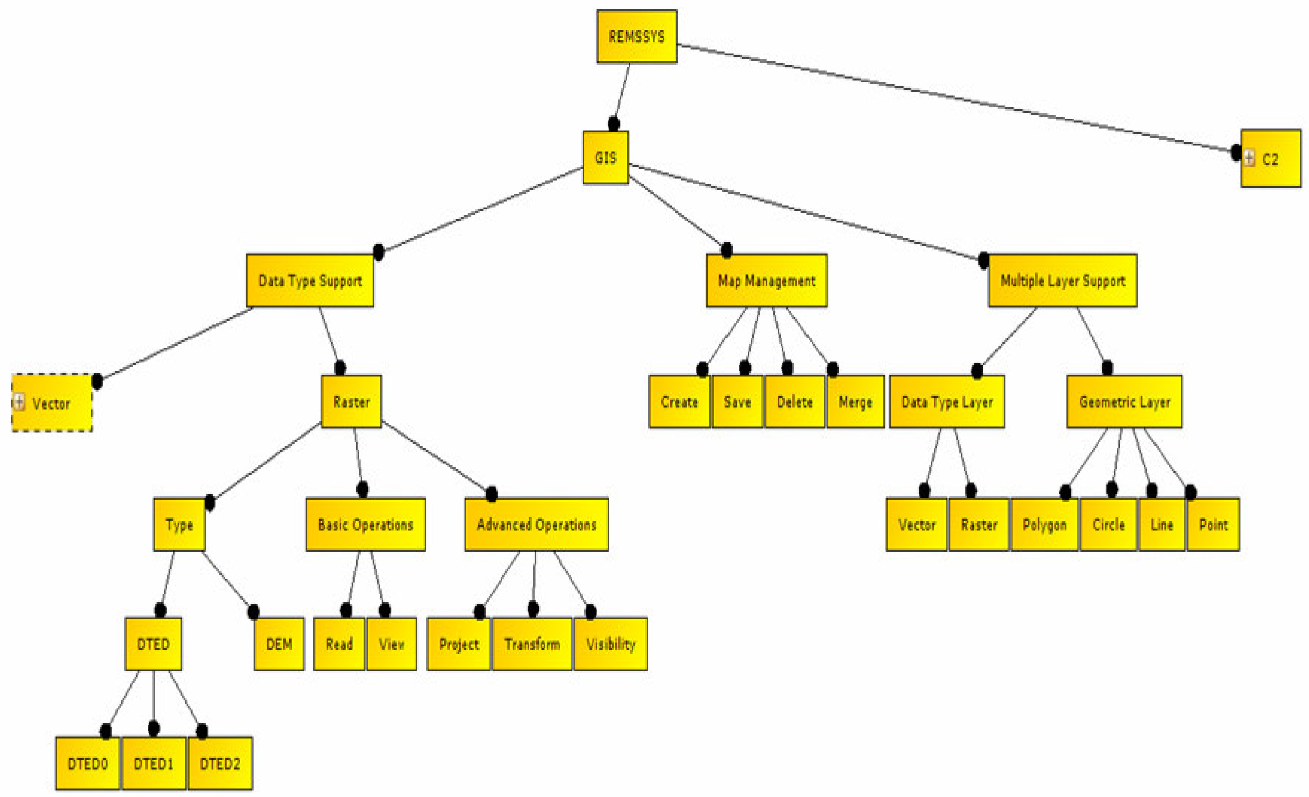


Figure 27 Feature Model of REMISSYS Application - Part 2

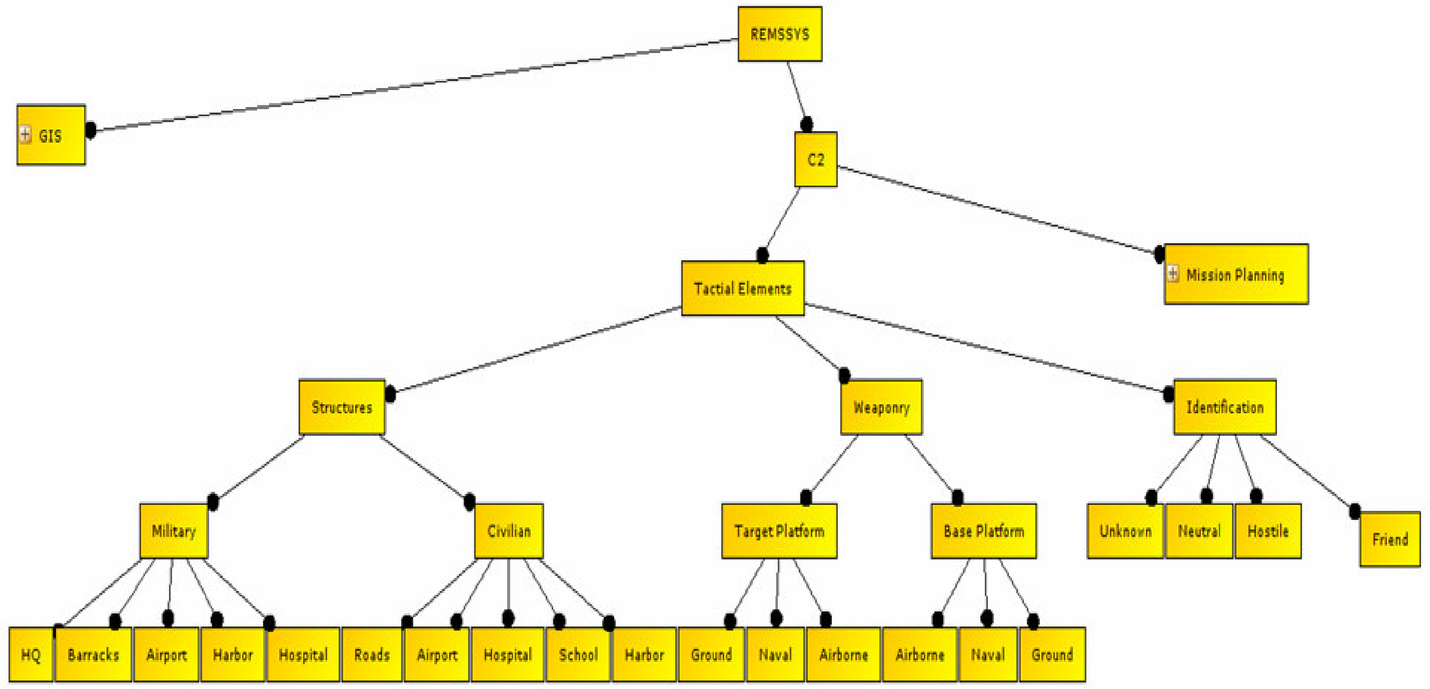


Figure 28 Feature Model of REMISSYS Application - Part 3

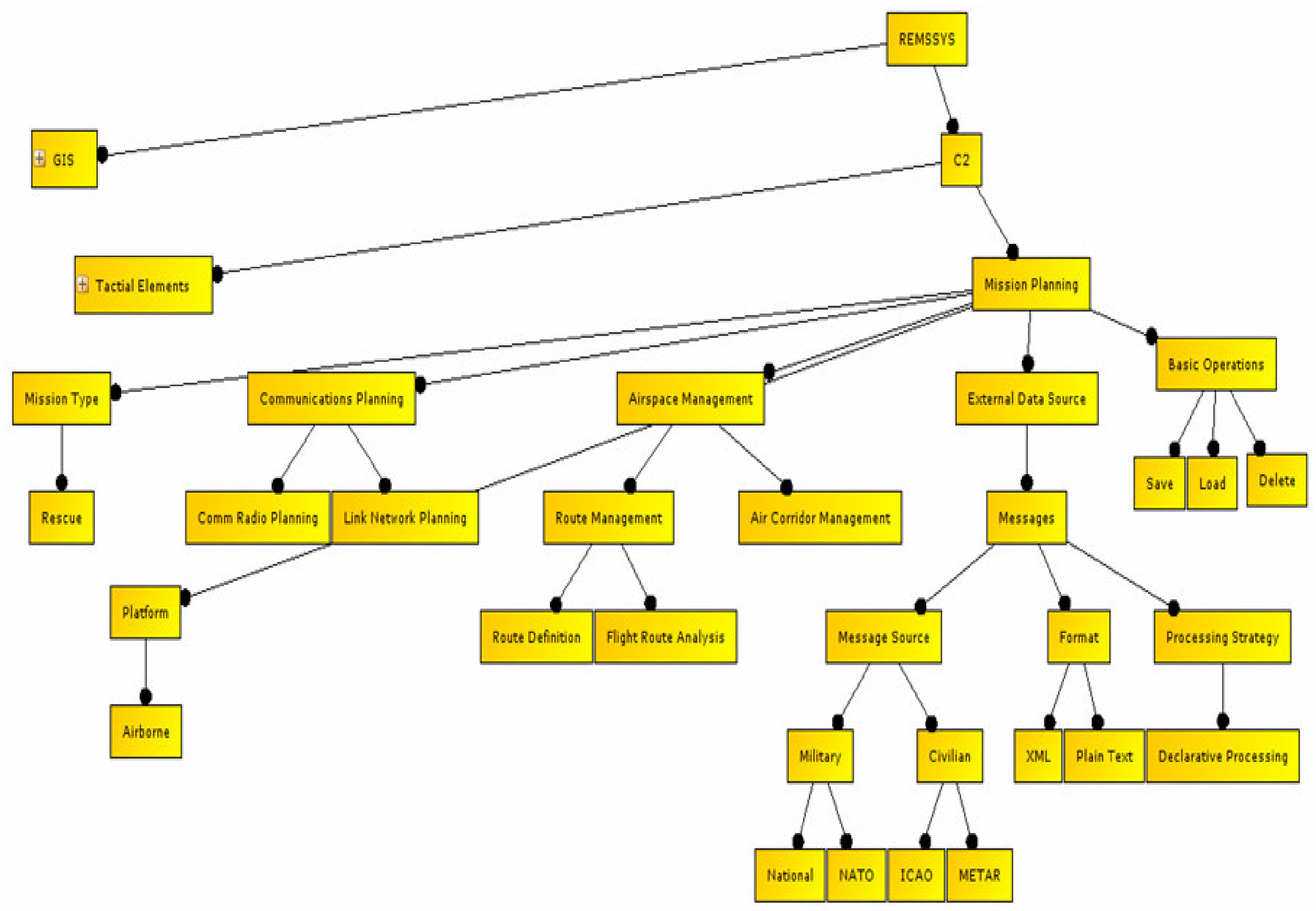


Figure 29 Feature Model of REMISSYS Application - Part 4

4.3 Requirements for REMISSYS Application

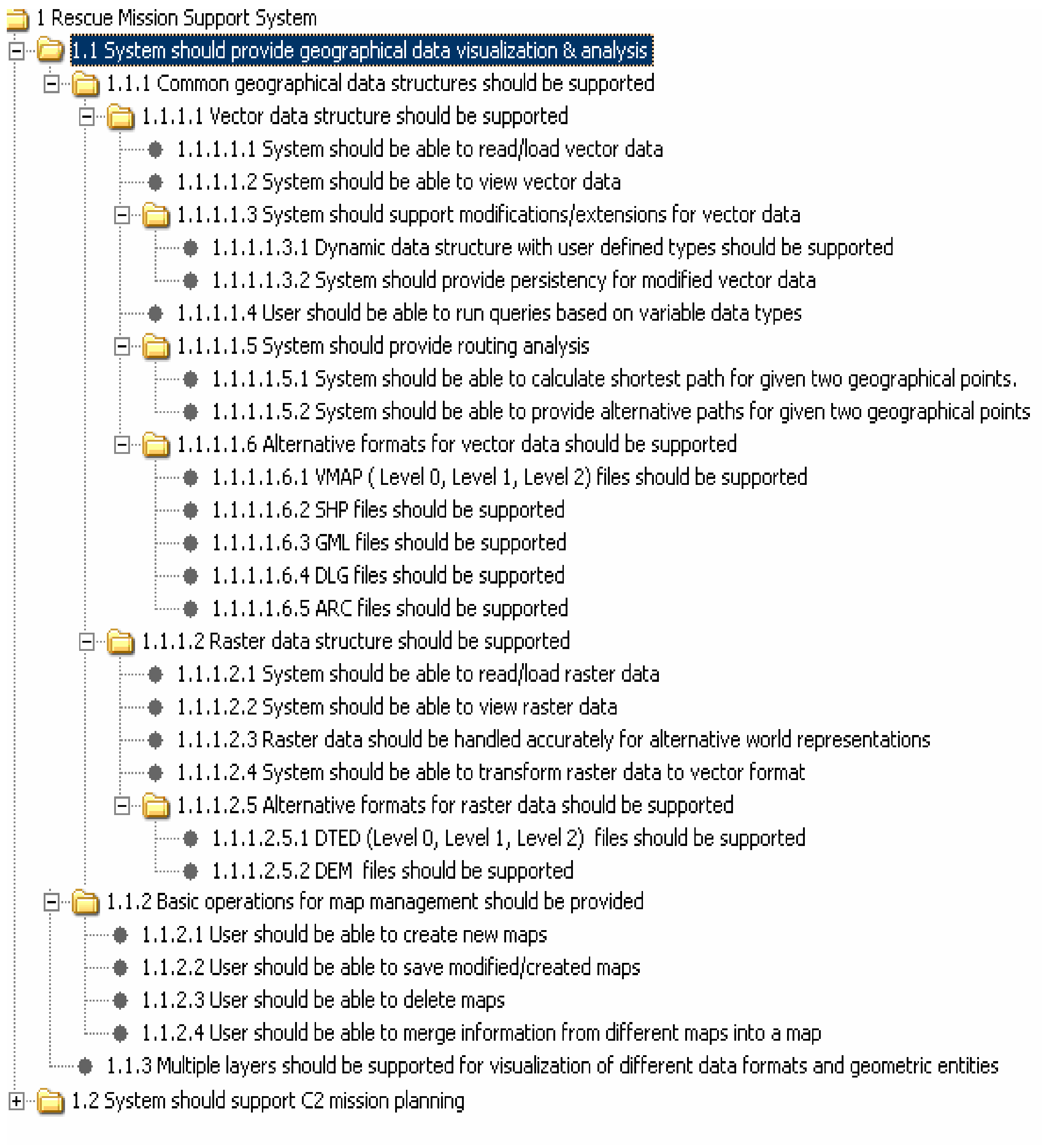
After application specific feature model is created for REMISSYS application, functional requirements are defined and decomposed using both a feature model and customer needs. A complete list of functional requirements is provided in Figures 30 and 31. This functional requirement definition and decomposition activity is performed by analyzing REMISSYS customer needs and the feature model. In this section REMISSYS functional requirements will be referenced by their numbers. For example functional requirement at the top will be referenced as FR 1.

First level decomposition was performed similar to the decomposition in the feature model. Two new functional requirements (FR 1.1 and FR 1.2) are defined for GIS related functionalities and C2 mission planning functionalities. Notice that C2 and mission planning features in the feature model are combined to define FR 1.2. Although tactical element feature exists under C2 feature, it is used to define functional requirements in different levels (FR 1.2.2 and FR 1.2.5.2.4).

FR 1.1 is decomposed same as it is in feature model. FR 1.1.1, FR 1.1.2 and FR 1.1.3 are defined according to their relevant features in the feature model. FR 1.1.1 is decomposed according to the data structure types supported.

Based on vector data structure feature and associated customer needs, FR 1.1.1.1 is defined. Decomposition of the FR 1.1.1.1 is performed after a detailed analysis on vector feature in feature model and customer needs. First, functional requirements for read and view operations (FR 1.1.1.1.1 and FR 1.1.1.1.2) are defined. Then dynamic structure, write operation for the vector data and advanced dynamic data editing features are combined into one functional requirement (FR 1.1.1.1.3) based on customer needs 4 and 6. In the same way, FR 1.1.1.1.3 is decomposed into two functional requirements (FR 1.1.1.1.3.1 and FR 1.1.1.1.3.2) one for user defined data types and one for persistency of the new/modified data. Vector data querying and advanced routing analysis features are used to define FR 1.1.1.1.4 and FR 1.1.1.1.5. FR 1.1.1.1.5 is decomposed into more functional requirements after a detailed analysis on customer need 8. Based on alternative vector data types, FR 1.1.1.1.6 is defined and decomposed.

Similarly, functional requirements related to raster data structure format and basic map management features are defined and composed. Notice that each basic and advanced operation features for raster data structure guide the definition of related functional requirements (FR 1.1.1.2.1, FR 1.1.1.2.2, FR 1.1.1.2.3 and FR 1.1.1.2.4). FR 1.1.1.2.4 is defined to represent projection feature; however definition of the functional requirement is guided by the customer need 1.3. At the end, data layer support functional requirement (FR 1.1.3) is defined by combining multiple layer support feature and its child features.



- 1 Rescue Mission Support System
  - 1.1 System should provide geographical data visualization & analysis
    - 1.1.1 Common geographical data structures should be supported
      - 1.1.1.1 Vector data structure should be supported
        - 1.1.1.1.1 System should be able to read/load vector data
        - 1.1.1.1.2 System should be able to view vector data
        - 1.1.1.1.3 System should support modifications/extensions for vector data
          - 1.1.1.1.3.1 Dynamic data structure with user defined types should be supported
          - 1.1.1.1.3.2 System should provide persistency for modified vector data
        - 1.1.1.1.4 User should be able to run queries based on variable data types
        - 1.1.1.1.5 System should provide routing analysis
          - 1.1.1.1.5.1 System should be able to calculate shortest path for given two geographical points.
          - 1.1.1.1.5.2 System should be able to provide alternative paths for given two geographical points
        - 1.1.1.1.6 Alternative formats for vector data should be supported
          - 1.1.1.1.6.1 VMAP (Level 0, Level 1, Level 2) files should be supported
          - 1.1.1.1.6.2 SHP files should be supported
          - 1.1.1.1.6.3 GML files should be supported
          - 1.1.1.1.6.4 DLG files should be supported
          - 1.1.1.1.6.5 ARC files should be supported
      - 1.1.1.2 Raster data structure should be supported
        - 1.1.1.2.1 System should be able to read/load raster data
        - 1.1.1.2.2 System should be able to view raster data
        - 1.1.1.2.3 Raster data should be handled accurately for alternative world representations
        - 1.1.1.2.4 System should be able to transform raster data to vector format
        - 1.1.1.2.5 Alternative formats for raster data should be supported
          - 1.1.1.2.5.1 DTED (Level 0, Level 1, Level 2) files should be supported
          - 1.1.1.2.5.2 DEM files should be supported
    - 1.1.2 Basic operations for map management should be provided
      - 1.1.2.1 User should be able to create new maps
      - 1.1.2.2 User should be able to save modified/created maps
      - 1.1.2.3 User should be able to delete maps
      - 1.1.2.4 User should be able to merge information from different maps into a map
    - 1.1.3 Multiple layers should be supported for visualization of different data formats and geometric entities
  - 1.2 System should support C2 mission planning

Figure 30 REMISSYS Functional Requirements - Part 1

FR 1.2 decomposition is performed a little different than feature model. Functional requirements for basic mission planning operations (FR 1.2.1, FR 1.2.1.2, FR 1.2.1.3 and FR 1.2.1.4) are defined and decomposed similar to how they would be done in the feature model. Additionally FR 1.2.1.1 is added.

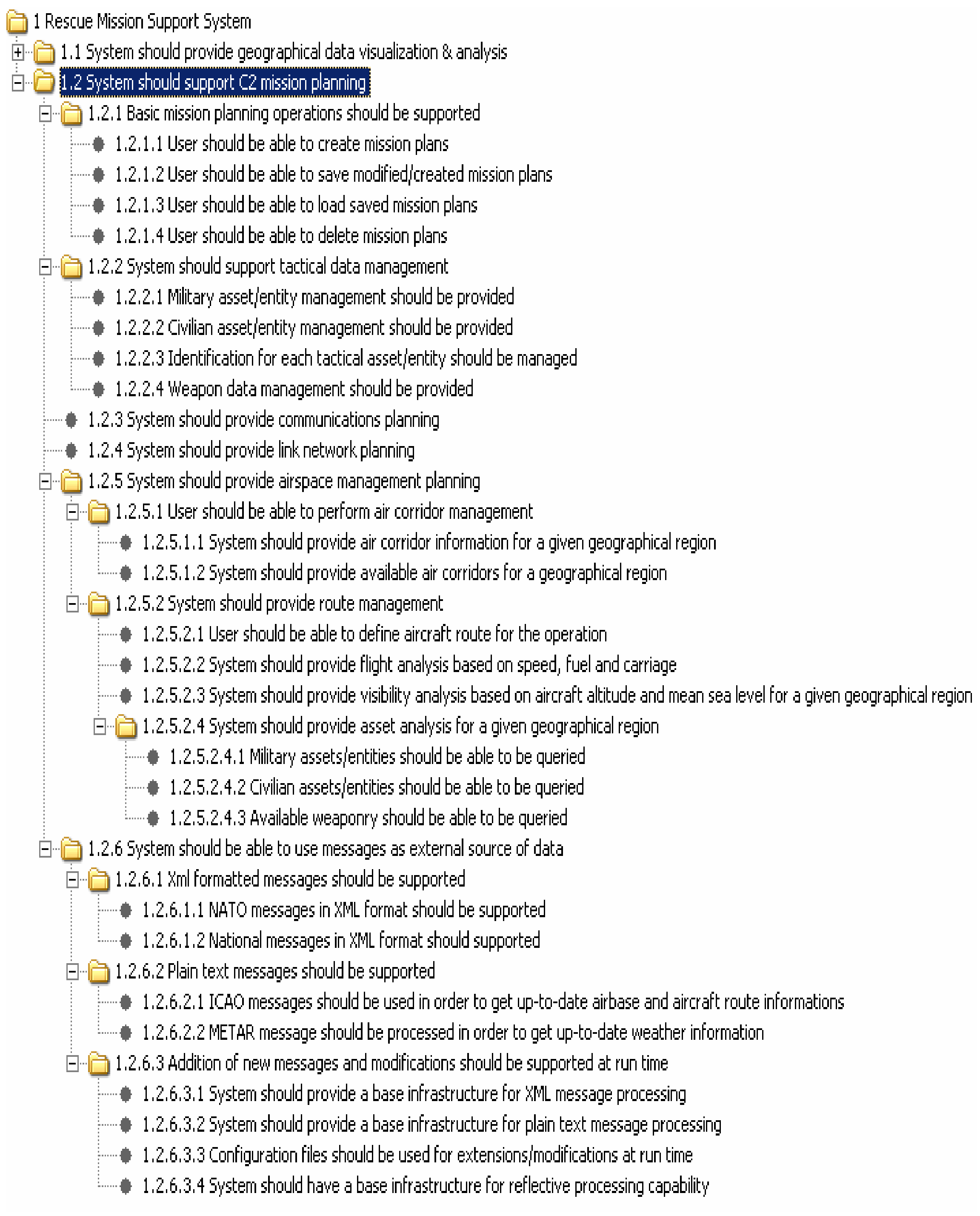
Although tactical element feature is at the same hierarchical level with the mission planning feature in the feature model, tactical element related functional requirements (FR 1.2.2) are handled as a child functional requirement of mission planning functional requirement (FR 1.2). Feature model defines the hierarchical relations according to overall domain where tactical elements are common to all C2 systems; like simulation applications, and mission planning applications. Therefore it is normal for the tactical element feature to be at the same level of hierarchy with "mission planning" feature in the feature model. However; in REMISSYS, tactical element functionalities are a part of the mission planning functionality. Therefore we represent tactical element functionalities as a child functionality of mission planning. Tactical element functionality is defined in FR 1.2.2 and decomposed based on military, civilian, weapon, and identification features. Notice that decomposition is not same as it is in the feature model; however it is guided by the feature model.

Communications radio planning and link network planning features are added as separate functional requirements (FR 1.2.3 and FR 1.2.4). Notice that although customer needs do not provide any information related to these functional requirements, feature model for the C2 domain provided the necessity for these functionalities of the system.

Airspace management feature is used to define FR 1.2.5. Route definition feature provided the definition of FR 1.2.5.1. Flight route analysis feature, tactical elements feature and customer needs 3, 5, 7, 8 and 10 guided definition and decomposition for FR 1.2.5.2, FR 1.2.5.2.1, FR 1.2.5.2.2, FR 1.2.5.2.3, FR 1.2.5.2.4, FR 1.2.5.2.4.1, FR 1.2.5.2.4.2, and FR 1.2.5.2.4.3.

External data source and message features are combined to define FR 1.2.6. Decomposition of this functional requirement is based on message data format types and processing strategy. FR 1.2.6.1 is defined based on XML message formats and decomposed according to military message formats; because, NATO messages require XML format in the domain feature model and customer need 9.2 provides a requirement for XML formatted national military messages. FR 1.2.6.2 is defined based on plain text message format and decomposed according to civil message formats; because, ICAO and METAR messages require plain text format in the domain feature model. FR 1.2.6.3 is defined according to declarative processing feature model. Decomposition of FR 1.2.6.3 is decided according to the existing knowledge of the similar message processing systems and customer need 9.2.

Utilization of feature models for functional requirement definition and decomposition process is provided in an example in this section. In the conclusion chapter, benefits and drawbacks off this approach are discussed in detail.



- 1 Rescue Mission Support System
  - 1.1 System should provide geographical data visualization & analysis
  - 1.2 System should support C2 mission planning
    - 1.2.1 Basic mission planning operations should be supported
      - 1.2.1.1 User should be able to create mission plans
      - 1.2.1.2 User should be able to save modified/created mission plans
      - 1.2.1.3 User should be able to load saved mission plans
      - 1.2.1.4 User should be able to delete mission plans
    - 1.2.2 System should support tactical data management
      - 1.2.2.1 Military asset/entity management should be provided
      - 1.2.2.2 Civilian asset/entity management should be provided
      - 1.2.2.3 Identification for each tactical asset/entity should be managed
      - 1.2.2.4 Weapon data management should be provided
    - 1.2.3 System should provide communications planning
    - 1.2.4 System should provide link network planning
    - 1.2.5 System should provide airspace management planning
      - 1.2.5.1 User should be able to perform air corridor management
        - 1.2.5.1.1 System should provide air corridor information for a given geographical region
        - 1.2.5.1.2 System should provide available air corridors for a geographical region
      - 1.2.5.2 System should provide route management
        - 1.2.5.2.1 User should be able to define aircraft route for the operation
        - 1.2.5.2.2 System should provide flight analysis based on speed, fuel and carriage
        - 1.2.5.2.3 System should provide visibility analysis based on aircraft altitude and mean sea level for a given geographical region
        - 1.2.5.2.4 System should provide asset analysis for a given geographical region
          - 1.2.5.2.4.1 Military assets/entities should be able to be queried
          - 1.2.5.2.4.2 Civilian assets/entities should be able to be queried
          - 1.2.5.2.4.3 Available weaponry should be able to be queried
    - 1.2.6 System should be able to use messages as external source of data
      - 1.2.6.1 Xml formatted messages should be supported
        - 1.2.6.1.1 NATO messages in XML format should be supported
        - 1.2.6.1.2 National messages in XML format should supported
      - 1.2.6.2 Plain text messages should be supported
        - 1.2.6.2.1 ICAO messages should be used in order to get up-to-date airbase and aircraft route informations
        - 1.2.6.2.2 METAR message should be processed in order to get up-to-date weather information
      - 1.2.6.3 Addition of new messages and modifications should be supported at run time
        - 1.2.6.3.1 System should provide a base infrastructure for XML message processing
        - 1.2.6.3.2 System should provide a base infrastructure for plain text message processing
        - 1.2.6.3.3 Configuration files should be used for extensions/modifications at run time
        - 1.2.6.3.4 System should have a base infrastructure for reflective processing capability

Figure 31 REMISSYS Functional Requirements - Part 2

CHAPTER 5

CONCLUSION

In this thesis study, utilization of feature modeling with a set of guidelines for requirements definition and decomposition activities of the Axiomatic Design methodology is provided. A tool that supports the development of feature models and modeling of the axiomatic design activities is implemented to be utilized for guiding the designer. The approach suggested in this thesis fills an important gap that is the transition from the customer needs to functional requirements, being the first two domains of Axiomatic Design. This research is complementing other work carried out after the introduction of Axiomatic Design, which supported various phases in the complete spectrum. A case study is carried out in order to analyze advantages and disadvantages of the proposed approach

During design activities for the provided case study, it is observed that proposed approach improved the axiomatic design activity in many ways; however its dependency on designer experience and knowledge remains as a drawback.

It is observed that feature model utilization saves a lot of effort spent for functional requirement definition and decomposition activity. It provides efficient design decisions at the early stages of the design activity. Most of the detailed analysis for system parts is already done and modeled in feature models. Therefore, feature models provide efficient domain-level definitions and decompositions for system functionalities. In this way, a template is already provided to the designer. Relations and constraints in feature models guide the design activity and prevent incomplete and invalid design decisions. Utilization of feature modeling provides reuse of existing knowledge gained in domain analysis activity. It guides the designer to define and decompose requirements through features identified in feature models. It is possible that some of the functional requirements are defined or decomposed as a result of zigzagging activity in axiomatic design. In that case feature models can be used to validate requirements and decomposition strategy when applicable.

The main drawback of this approach is its dependency on the designer's experience, knowledge and understanding of the system and related domains. It can be observed from the provided case study that, although functional requirements are defined and decomposed based on features and customer needs; at some point during the design, designer has to decide based on his experience and knowledge. Definition of some functional requirements might be identified more efficiently by combining one or more functionalities in the feature model. Although feature models provide more information than customer needs regarding the system design, level of detail might not be sufficiently provided enough to create a complete and efficient design. Therefore designer will have to provide the required details. Some decomposition strategies in the feature model might not be the most efficient choice for the aimed application. In these cases, designer needs to analyze the alternatives and select the most efficient design decision.

To sum up; in this thesis study, efficiency and quality of the axiomatic design activity is improved by utilization of feature models during functional requirement definition and decomposition activities. Customer needs are utilized more efficiently together with feature models. Although dependency on designer's experience and knowledge remains as a drawback, invalid and incomplete design decisions are prevented and most of the effort is saved for the design activity as a result of reusing existing knowledge provided in feature models and domain-level constraints are applied automatically in design decisions.

For future, this work can be enhanced by a detailed methodology in the transition from the customer needs to functional requirements. For this purpose, future models were already utilized in this thesis; however, further guidance can be obtained by supporting the feature model with ontology. Also the guidance could be more comprehensive by investigating the implementation related issues during every requirements decision.

REFERENCES

[1] Johnson, M. S. 1995. A survey of object-oriented reuse. In Proceedings of the 1995 Conference of the Centre For Advanced Studies on Collaborative Research (Toronto, Ontario, Canada, November 07 - 09, 1995). K. Bennet, M. Gentleman, H. Johnson, and E. Kidd, Eds. IBM Centre for Advanced Studies Conference. IBM Press, 35.

[2] Srinivasan, S. and Vergo, J. 1998. Object oriented reuse: experience in developing a framework for speech recognition applications. In Proceedings of the 20th international Conference on Software Engineering (Kyoto, Japan, April 19 - 25, 1998). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 322-330.

[3] Dogru, A. H. and Tanik, M. M. 2003. A Process Model for Component-Oriented Software Engineering. IEEE Softw. 20, 2 (Jan. 2003), 34-41.

[4] Katz, S., C. Dabrowski, K. Miles and M. Law (1994), "Glossary of Software Reuse Terms," NIST Special Publication 500-222, Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD.

[5] Axiomatic Design Solutions Inc. Technology Overview. 2006. Axiomatic Design Solutions Inc. http://www.axiomaticdesign.com/technology/axiomatic.asp, 22 April 2007.

[6] Togay, C., Dogru, A.H., "Component Oriented Design Based on Axiomatic Design Theory and COSEML", Lecture Notes in Computer Science, 4263/2006: 1072-1079, 2006

[7] Togay, C., Dogru, A.H., Tanik, U.J., Grimes, G.J., "Component Oriented Simulation Development with Axiomatic Design", The Ninth World Conference on Integrated Design and Process Technology, San Diego, California, June 25-30, 2006

[8] Togay, C., Dogru, A.H., Tanik, J.U.," Systematic Component-Oriented Development with Axiomatic Design", The Journal of Systems & Software, DOI information: http://dx.doi.org/10.1016/j.jss.2007.12.746, April 2008.

[9] Togay, C., Bicer, V., Dogru, A.H., “Deadlock Detection in High-level Architecture Federations using Axiomatic Design Theory”, EUROSIM07, Slovenia, September, 2007

[10] Do, S. H. and Suh, N. P., June 2000, "Object Oriented Software Design with Axiomatic Design," Proceedings of ICAD2000 First International Conference on Axiomatic Design.

[11] Clapis, P. J. and Hintersteiner, J. D., 2000, "Enhancing Object Oriented Software Development Through Axiomatic Design," First International Conference on Axiomatic Design, Cambridge, MA.

[12] Hintersteiner, J. D. and Nain, A., "Integrating Software into Systems: An Axiomatic Design Approach" Proceedings of the 3rd International Conference on Engineering Design and Automation, Vancouver, B. C. Canada. August 1-4, 1999.

[13] Steward D., Tate D., “Integration of Axiomatic Design and Project Planning”, First International Conference on Axiomatic Design (ICAD2000), (edited by D. Tate), Cambridge, MA, pp. 285-289, June 21-23, 2000.

[14] The New York State Center for Engineering Design and Industrial Innovation. 2008. NYSCEDII Administrator. <http://www.dsmweb.org/content/view/21/26/>.22 May 2007.

[15] Browning, T. R. "Applying the design structure matrix to system decomposition and integration problems: a review and new directions." Engineering Management, IEEE Transactions on 48 (2001): 292-306.

[16] Suh, N.P., 1998. Axiomatic design theory for systems. Research in Engineering Design 10 (4), 189–209.

[17] Cognitive Research Laboratory. Word.Net. 2006. Princeton University. <http://wordnet.princeton.edu/perl/webwn?s=domain>. 19 Nov. 2007.

[18] “Domain”. The American Heritage Dictionary of the English Language. 4th ed. Boston: Houghton, 2000..

[19] Software Engineering Institute. Domain Engineering: A Model-Based Approach. 2007. Carnegie Mellon University. 19 Nov. 2007 <http://www.sei.cmu.edu/domain-engineering/>.

[20] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, November 1990

[21] Wikimedia Foundation, Inc. Domain. 2008. Wikimedia Foundation, Inc. http://en.wikipedia.org/wiki/Domain, 19 Nov. 2007.

[22] K. Czarnecki. Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models. Ph.D. thesis, Technische Universität Ilmenau, Germany, 1998.

[23] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., and DeBaud, J. 1999. PuLSE: a methodology to develop software product lines. In Proceedings of the 1999 Symposium on Software Reusability (Los Angeles, California, United States, May 21 - 23, 1999). SSR '99. ACM, New York, NY, 122-131

[24] Prieto-Díaz, R. 1990. Domain analysis: an introduction. SIGSOFT Softw. Eng. Notes 15, 2 (Apr. 1990), 47-54

[25] J. Neighbors. Software construction using components. Ph.D. dissertation, (Tech. Rep. TR-160), Department of Information and Computer Science, University of California, Irvine, 1980

[26] J. Neighbors. The Draco Approach to Construction Software from Reusable Components. In IEEE Transactions on Software Engineering, vol. SE-10, no. 5, September 1984, pp. 564-573

[27] Neighbors, J. M. 1989. Draco: a method for engineering reusable software systems. In Software Reusability: Vol. 1, Concepts and Models, T. J. Biggerstaff and A. J. Perlis, Eds. ACM, New York, NY, 295-319.

[28] Iscoe, N. 1993. Domain modeling—overview & ongoing research at EDS. In Proceedings of the 15th international Conference on Software Engineering (Baltimore, Maryland, United States, May 17 - 21, 1993). International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, 198-200.

[29] CAMP, Common Ada Missile Packages, Final Technical Report, Vols. 1, 2, and 3. AD-B-102 654, 655, 656. Air Force Armament Laboratory, AFATL/FXG, Elgin AFB, FL, 1987.

[30] S. Peterson and S. Cohen. A Context Analysis of the Movement Control Domain for the Army Tactical Command and Control System. Technical Report, CMU/SEI-91-SR-3, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1991

[31] M. Simos, D. Creps, C. Klinger, L. Levine, and D. Allemang. Organization Domain Modeling (ODM) Guidebook, Version 2.0. Informal Technical Report for STARS, STARS-VC-A025/001/00, June 14, 1996.

[32] G. Arrango. Domain Analysis Methods. In Software Reusability, Schäfer, R. Prieto-Díaz, and M. Matsumoto (Eds.), Ellis Horwood, New York, 1994, pp. 17-49

[33] Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E., and Huh, M. 1998. FORM: A feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng. 5 (Jan. 1998).

[34] M. Riebisch, Towards a more precise definition of feature models. Position Paper, in: M. Riebisch, J.O. Coplien, D, Streitferdt (Eds.), Modelling Variability for Object-Oriented Product Lines, 2003.

[35] Six Sigma and Beyond: Statistics and Probability, by D. H. STAMATIS, Boca Raton, FL: CRC Press, 2003, ISBN 1-57444-312-7. pp. 542-548.

[36] Axiomatic Design - Advances and Applications by Nam Pyo Suh, Massachusetts Institute of Technology,Oxford University Press, New York.Oxford, 2001, Chapter 5, pp. 239-298.