SYSTEMATIC COMPONENT-ORIENTED DEVELOPMENT WITH AXIOMATIC DESIGN

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

BY

CENGİZ TOĞAY

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

JULY 2008

Approval of the thesis:

SYSTEMATIC COMPONENT-ORIENTED DEVELOPMENT WITH AXIOMATIC DESIGN

Submitted by **CENGİZ TOĞAY** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy 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

Prof. Dr. Muslim Bozyiğit _____
Computer Engineering Dept., METU

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

Prof. Dr. Mehmet R. Tolun _____
Computer Engineering Dept., Çankaya University

Assoc. Prof. Dr. Halit Oğuztüzün _____
Computer Engineering Dept., METU

Assist. Prof. Dr. Bülent Gümüş _____
Industrial Engineering Dept., TOBB Univ. of Econ. and Tech.

Date: 16/07/2008

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Cengiz Toğay

Signature

ABSTRACT

SYSTEMATIC COMPONENT-ORIENTED DEVELOPMENT WITH AXIOMATIC DESIGN

Toğay, Cengiz

Ph.D., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Ali Hikmet Doğru

July 2008, 131 pages

In this research, component oriented development is supported with design guidance by extending the Axiomatic Design Theory for component orientation, and utilizing domain engineering and ontology mechanisms. Guidance is offered in the form of suggesting missing components and discovering incompatibilities among the candidate elements of software development, corresponding to different phases such as requirement analysis, design, and implementation. A mature domain concept is developed suggesting the availability of reference models for customer needs, software system requirements, software design, and also a rich set of implemented components. As the system is being defined starting with the customer needs and progressing towards components, at every step the developer is presented what is available in the domain and what becomes unavailable. This guidance is based on the selections made so far, utilizing ontology based constraint checking. Feature Models are incorporated for modeling customer needs. Case studies are presented for demonstration purposes.

Keywords: Component Orientation, Axiomatic Design Theory, Feature Model, COSEML, Ontology.

ÖZ

AKSİYOMATİK TASARIM İLE SİSTEMATİK BİLEŞENE YÖNELİK GELİŞTİRME

Toğay, Cengiz
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Ali H. Doğru

Temmuz 2008, 131 sayfa

Aksiyomatik tasarım teorisi, bileşen yönelimli geliştirmeyi desteklemek üzere genişletilmiştir. Ayrıca, bileşen yönelimli geliştirmeyi desteklemek için alan mühendisliği ve ontoloji mekanizmaları yardımı ile bir tasarım rehberliği oluşturulmuştur. Rehberlik, eksik bileşenlerin ve geliştirme ögeleri adayları arasındaki uymusuzlukların gereksinimler, tasarım, ve uygulama gibi değişik safhalara yönelik olarak önerilmesi şeklindedir. Olgun alan kavramı, müşteri ihtiyaçları, yazılım sistem gereksinimleri, yazılım tasarımı ve çok sayıda geliştirilmiş bileşenler için referans modellerinin mevcut olmasına bağlı olarak geliştirildi. Sistem geliştirme süreci, müşteri ihtiyaçlarından başlayarak bileşenlere ulaşma yönünde devam ederken her daımda geliştiriciye alanda neyin uygun olduğu ve neyin uygunsuzlaştığı bildirilir. Bu rehberlik, yapılmış seçimler ışığında ontolojıye dayanarak kısıtların kontrol edilmesi yolu ile gerçekleştirilir. Müşteri gereksinimlerinin modellenmesi için yetenek modeli kullanılmaktadır. Örnek uygulamalar ile yöntem anlatılmıştır.

Anahtar Kelimeleri: Bileşen yönelimi, Aksiyomatik Tasarım Teorisi, y Modeli, COSEML, Ontoloji.

DEDICATION

To My Wife Sine

ACKNOWLEDGMENTS

I would like to express my deepest gratitude and appreciation to my supervisor, Assoc. Prof. Dr. Ali H. Doğru, for his guidance, advice, criticism, encouragements, insight and patience throughout the research.

I would also like to thank my committee members Prof. Dr. Mehmet R. Tolun, Prof. Dr. Muslim Bozyiğit, Assoc. Prof. Dr. Halit Oğuztüzün, and Assist. Prof. Dr. Bülent Gümüş for their invaluable suggestions and comments.

Prof. Dr. Murat Tanik's guidance and suggestions during my visit at University of Alabama (UAB) have presented crucial steering which I appreciated deeply.

I would also like to thank Prof. Dr. Muslim Bozyiğit, Burak Ulutoprak, Dr. Okan Topçu for their technical support for developing simulations, Orhan Üçtepe for his technical support while developing ADCO tool, my friends Alper Kılıç, Alev Mutlu, Eren Koçak Akbıyık, Gayathri Sundar, Gülşah Tümüklü, Levent Bayındır, Selma Süloğlu, Oral Dalay, Dr. Özgür Aktunç, Özgür Kaya, Dr. Urcun Tanik, and all my friends not named for their support during my Ph.D, and the staff members at CENG department, especially Sultan Arslan, Perihan İlgun and Güldane Öcal for their indefinite help and cooperation.

Last but not least, I would like to give special thanks to my wife and my parents for their continual and indefinite support and prayers.

This study was supported by the State Planning Organization (DPT) Grant No: BAP-08-11-DPT2002K120510.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ADCO	Axiomatic Design with Component Orientation
ADT	Axiomatic Design Theory
CN	Customer Need
CORBA	Common Object Request Broker Architecture
COSE	Component Oriented Software Engineering
COSEML	Component Oriented Software Engineering Modeling Language
COTS	Commercial Off-The Shelf
CSP	Communicating Sequential Process
DCOM	Distributed Component Object Model
DM	Design Matrix
DP	Design Parameter
DSM	Design Structure Matrix
EJB	Enterprise JavaBeans
FR	Functional Requirement
HLA	High Level Architecture
IDL	Interface Definition Language
OMT	Object Model Template
OO	Object Oriented
PV	Process Variable
UML	Unified Modeling Language
OWL	Web Ontology Language

CHAPTER 1

INTRODUCTION

The Axiomatic Design Theory (ADT) has been proposed by Suh as a scientific approach [100-103]. ADT encapsulates the design process from customer needs to product, utilizing two fundamental axioms and corollaries to obtain "good design" in terms of complexity, maintenance, and testing concepts. ADT has been applied to software engineering for structured analysis and development of software designs [32], object oriented software design [19, 33, 34, 91, 101], requirements management [46], and project planning [98]. In this study, we applied ADT to Component-Oriented Software Engineering (COSE) [113, 114, 116, 118, 119] and named the new approach as Axiomatic Design with Component-Orientation (ADCO). We also added some enhancements to ADT such as collaboration diagrams [114] that are proposed to identify dependencies in the ADT's design matrix, deadlock detection [112, 120] to test solutions against requirements, component interface representations in terms of COSEML [114], component congruity measurement in terms of information content [111], and Feature Models [64] utilized to identify customer needs.

Using component technologies is one of the cost-effective ways of constructing systems. In Component-Based software engineering approaches, system design and component usage are not drastically different from Object-Oriented software development [37]. However, in COSE, components, identified based on customer requirements and their composition, are represented, hence avoiding code development. In one of the COSE approaches namely COSEML [36, 37], requirements are evaluated and systems are created through structural decomposition. COSEML approach is based on the available components and developer experience. Success of system development with COSE is dependent on

availability of mature domains. Mature domains including various components still carry some problems in terms of integration. In this dissertation, we are proposing another COSE approach based on ADT that is Axiomatic Design with Component Orientation (ADCO) for mature domains. One advantage of integrating ADT and COSEML [36, 37] is to support component interfaces with more information. Well defined interfaces and constraints help to locate and integrate components [21]. If information about the components is not adequately presented to the developers, the developers will not optimally benefit from the reuse potential of the components.

In our approach, we assume that the services published by components are implemented to solve functional requirements. Incorporating ADT, supplements component interfaces with a design matrix that stores all relationships among functional requirements and interface items (methods, attributes, and events). Also, any component internal dependencies as well as external dependencies are possible to represent.

There are two opposite approaches for system design. In the first approach, same set of functional requirements can be solved with the same solution set. This approach increases reliability, decreases design cost because most of the design problems have been identified and solved before. The gained expertise is transferred to the mature domain for future use. Also mature domains increase chance of utilization success for components. But at the same time, this approach prevents finding more effective solutions. The second approach proposes to focus on functionalities without considering available designs [45]. Since in software world, reuse is a primary goal in the software development because of well known reasons, ADCO is closer to the first approach.

In a mature domain, a generic system is defined that is utilized in the instantiation of specific products. Mature domains include experiences of the designer. Experiences are reflected to the mature domain with various constraints. Constraints which relate to components, features, requirements, design parameters, and dependencies should be evaluated and utilized to guide designers. Expert requirements analysts can

decide which functional requirements can be defined in a new system through interpretation of the customers' needs. Customer needs can be identified with various tools such as brainstorming, interviews, observation of work patterns, reverse engineering, technical documentation review etc. [45]. We have utilized features [64] to identify customer needs. Feature Models as a domain analysis tool provides a communication environment between customers and other stakeholders such as designers. Since features can include requirements, implementation level information, constraints, etc., they can be useful in mature domains. Therefore, one of the contributions in this study is to construct a bridge between customers and the solution, based on available solution alternatives in mature domains. Another advantage of Feature Models is guidance for customers to express themselves with available materials (features). Generally, customers do not express themselves because of cultural/educational differences between customers and designers. Customers define their needs using the domains Feature Model through selection of features. Designers benefit from the selected features to specify Functional Requirements for specific systems originating from the mature domain. We define a mapping approach based on ontology among the features and ADT domains corresponding to requirements, design, and implementation domains.

Beyond this first chapter, the dissertation is organized as follows: In chapter 2, required background and our related studies are described. In Chapter 3, our ontology creation method is described. In Chapter 4, ADCO approach is explained and applied to two different domains namely Conference Management System (web service based application) and High Level Architecture [54] based simulations. A brief conclusion for this dissertation and further work that can be performed based on this work is represented in the last chapter.

CHAPTER 2

BACKGROUND

2.1. Axiomatic Design Theory

Axiomatic Design Theory (ADT) is a systematic methodology to decompose requirements and solution in a top-down fashion that assists designers to structure design problems [101, 103, 104]. Ultimate goal of ADT is to reach the "best" or "good" design as other Decision Based Design (DBD) methods and methodologies [82]. ADT is an interdisciplinary approach which is applied to various engineering domains such as mechanical [80, 130], GRID engineering [107], and software engineering such as structured analysis and development of software designs [32], object oriented software design [19, 33, 34, 91, 101], requirement management [46], project planning [98], and Component-Oriented software design [113, 114, 116, 118, 119]. Also, research has been conducted for incorporating collaboration diagrams [114], deadlock detection [112, 120], COSEML [114], component congruity [111], test concepts [46] in the ADT. Concepts of ADT are introduced in section 2.1.1 and a systematic approach for Object Oriented design is explained briefly in section 2.2.2.

2.1.1. Concepts of Axiomatic Design Theory

Essentially, axiomatic design concentrates on four concepts: *domains*, *hierarchies*, *zigzagging*, and *axioms* [101]. The following subsections explain those concepts.

2.1.1.1.Domains

The domains are divided into four inter-related parts: (1) customer domain, (2) functional domain, (3) physical domain, and the (4) process domain, providing

respectively, Customer Needs (CNs), Functional Requirements (FRs), Design Parameters (DPs), and Process Variables (PVs), as shown in Figure 2.1.

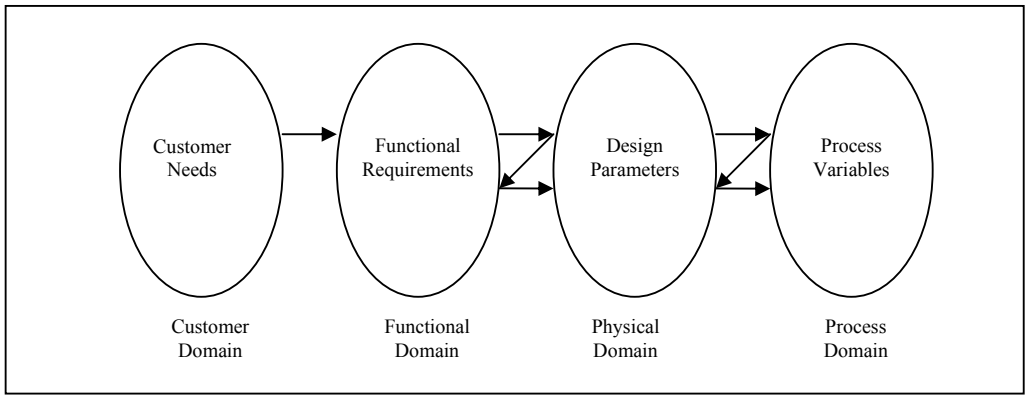

Figure 2.1 Domains and their elements (adapted from [101])

The relation between the domains is expressed as "What" and "How" questions (e.g. what the customer wants (CN) is addressed by how it is accomplished (FR)). FRs are defined as a minimum set of independent requirements that completely characterize the functional needs. The FRs represent system requirements in a hierarchy that specifies CNs and should describe the expectations from the products, and how such expectations should succeed is not of concern. DPs are defined as the key physical variables such as methods, services, components, and some abstractions in terms of software that characterizes the design that satisfies the FRs. The PVs satisfy the DPs with implemented items such as implemented components. Design domains can be interpreted differently by various disciplines as listed in Table 2.1. As represented in Table 2.1, in this dissertation, we have characterized customer attributes as features [64] in the customer domain. Features are mapped to FRs and there are many to many relationships among FRs, DPs, PVs, and features. This is one of our contributions to ADT. How relationship between FRs and features are defined is explained in Chapter 3.

Table 2.1 Meaning of the design domains in various disciplines (adapted from [101])

		Customer Domain	Functional Domain	Physical Domain	Process Domain
Manufacturing		Customer attributes	FRs specified for product	DPs that can satisfy FRs	PVs that control DPs
Materials		Desired performance	Required properties	Microstructure	Process
Organizations		Customer satisfaction	Functions of the organization	Programs, offices, activities	People and other resources to support programs
Systems		Attributes desired of the overall system	FRs of the system	Machines, components, subcomponents	Resources
Business		ROI	Business goals	Business structure	Human and financial resources
Software	**General**	Attributes desired in the software	Output specification of program codes	Input variables, algorithms, modules, program codes	Subroutines, machine codes, compilers, modules
	Object Oriented	Customer attributes	Objects	Data	Subroutines, machine codes
	Component Oriented	Attributes desired in the software (features)	FRs specified for products	Processes, methods, abstractions (component, interface, package)	Real components and web services

Constraints are used to define the boundaries of the acceptable solutions and they have to be consistent with each other [101]. There are two kinds of constraints namely input and system constraints [101]. Input constraints are defined in the beginning of design activity and affect the whole design decisions for instance price, time, industrial standards, environment constraints, etc. System constrains

such higher- level design decisions are specified during design process. In this study, we have used ontology definitions as explained in Chapter 3 to define some input constraints.

2.1.1.2. Hierarchies

The second concept of axiomatic design that is consistent with Simon [96] is hierarchical decomposition process for all ADT domains. According to Simon, an important approach to solving complex problems is to divide the problems into simpler parts, solve them, and integrate into the solution [96]. Thus, axiomatic design decomposes the problems (functional requirements) into simpler parts since it has a top-down approach (hierarchies), while introducing the concept of Zigzagging in all axiomatic design domains.

2.1.1.3. Zigzagging

The third concept is Zigzagging. Instead of decomposing only the FR domain thoroughly, independent of any other domain, zigzagging allows a parallel decomposition of all four domains. Process starts with specifying the customer needs. FRs are specified by answering the question, "What must this design do to satisfy our customer' needs?" We used Feature Models to define customer's needs. Hence, the decomposition of a complex problem starts with determination of the most general FR from the customer needs. Then, the designer "zigs" to the DP domain and determines an appropriate DP to fulfill that particular FR. Once a DP is chosen to satisfy FR and the DP is not implementable, then the designer "zags" to the FR domain to a level below the former FR, thus creating a dependency of that particular FR to the previous DP choice. This process of zigzagging continues until all the leaves of the DPs are satisfied with implementable PVs, as shown in Figure 2.1 and Figure 2.2.

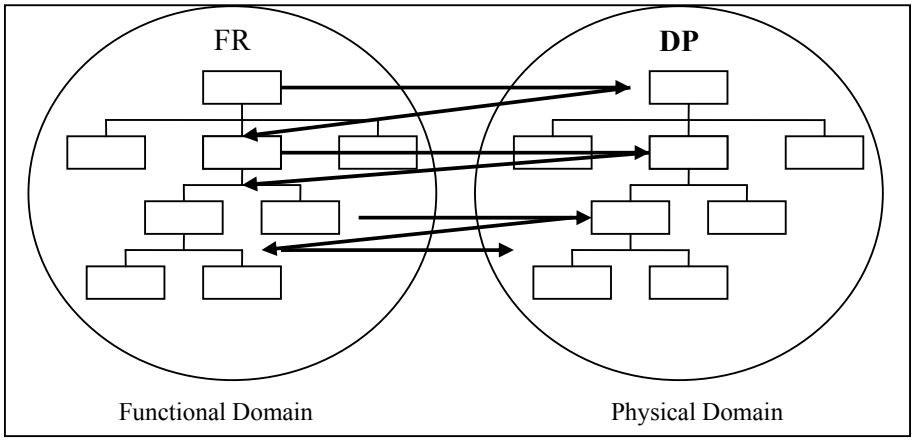

Figure 2.2 Zigzagging (adapted from [101])

2.1.1.4. Axioms

The fourth concept establishes two axioms to select the best design among candidates and also accelerate the design in the right direction without much trial and error [69]. Axioms defined based on the investigation of various system designs in different engineering disciplines are listed below [101]:

> Axiom 1 (Independence): "Maintain the independence of the functional requirements": The independence here corresponds to the functional requirements set so that a requirement is understood easily without having to refer to the others extensively. The axiom leads to keeping the design simple.
>
> Axiom 2 (Information): "Minimize the information content of the design": Information content is measured based on the design range specified by the designer and system range provided by DP. This axiom also prevents the design from getting unnecessarily complex.

Mappings between domains are represented using a square design matrix implying the expectation that one FR should correspond to one and only one DP. Square design matrices are recommended to provide independence axiom but design may be started with non-square matrix. There are two kinds of design matrices in ADT

namely FR-DP and DP-PV matrices called *Process Matrix*. Mappings between FR and DP vectors can be represented in equation 2.1 where |A| is called the FR-DP design matrix.

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{vmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{vmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_2 \end{Bmatrix} \tag{2.1}$$

Elements of the equation can be written as defined in equation 2.2.

$$\begin{aligned} FR_1 &= A_{11}DP_1 + A_{12}DP_2 + A_{13}DP_3 \\ FR_2 &= A_{21}DP_1 + A_{22}DP_2 + A_{23}DP_3 \\ FR_3 &= A_{31}DP_1 + A_{32}DP_2 + A_{33}DP_3 \end{aligned} \tag{2.2}$$

Similarly, DP-PV matrix is represented in equation 2.3.

$$\begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix} = \begin{vmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{vmatrix} \begin{Bmatrix} PV_1 \\ PV_2 \\ PV_2 \end{Bmatrix} \tag{2.3}$$

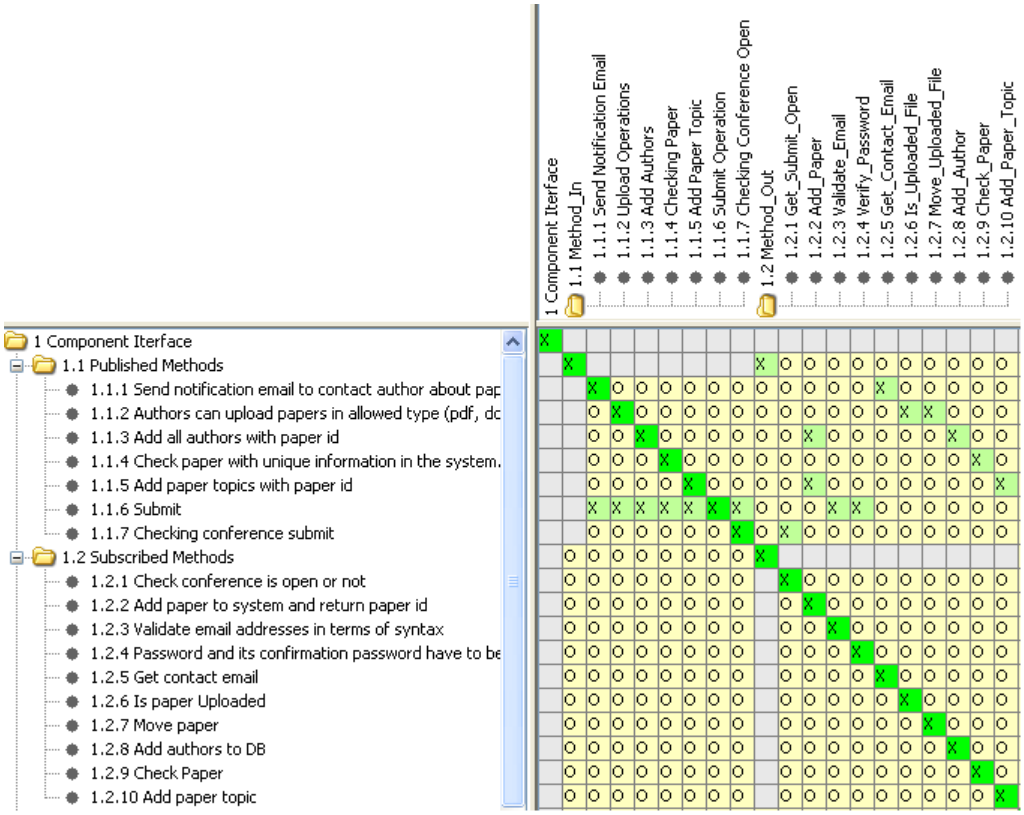

Figure 2.3 Partial Design Matrix of Submit Component Interface

The mappings between domains can be described with "X" symbols in related cells of the matrix. So if Aij is "X" then FRi is related to DPj for the FR-DP matrix. Otherwise, the case of no relationship is indicated by "0" as depicted in Figure 2.3. In the design matrix, FRs are represented in rows and DPs are represented in columns. As an example, FR1.1.1 is satisfied by DP1.1.1 and DP1.2.5. The main DP to satisfy FR1.1.1 is the DP1.1.1. The DP1.2.5 is the supporter DP required by DP1.1.1.

2.1.1.4.1. Independence Axiom

The Independence Axiom is used to identify whether the design is coupled, decoupled, or uncoupled by utilizing design matrices. Independence axiom is used during decomposition to reduce couplings. Coupling corresponding to the dependencies among FRs increases the complexity of a system. The three cases involved in such fulfillments are as follows:

- **Uncoupled Design**: This kind of design is the ideal case, but rarely occurs in the real world. Each FR is satisfied by one DP so that a diagonal design matrix is produced. Diagonal matrix is formed when A_{ij} =0 except those where i=j.

- **Decoupled Design**: This arrangement occurs most often in the design world. The design matrix must be triangular, meaning that all the relationships indicated by "X" must be placed at only one of the sides of the diagonal in the design matrix.

- **Coupled Design**: The relationships "X"s are everywhere in the design matrix, indicating a highly interdependent design.

Independence axiom should be applied after partitioning algorithms [97, 99]. Although some designs can be seen coupled, after the application of partitioning algorithms they can be converted to decoupled designs. A design can satisfy the independence axiom if it is uncoupled or decoupled. There may be a unique solution that satisfies the FRs in a coupled design but such a design produces various problems. For instance, if one of the FRs is changed all the design matrix is effected from this change. If the system is uncoupled it provides advantages such as FRs and related DPs can be changed or modified dynamically without affecting others. This property is important for Component Oriented Approaches. Another advantage of the uncoupled design occurs during decomposition. Uncoupled parts can be decomposed separately from others. Any modification of the higher-level FRs is local in uncoupled designs.

The coupling degree of a design matrix can be a quality factor for design. Coupled designs can cause unintended consequences [34] such as deadlocks [120]. An approach has been proposed to measure strength of a coupled design [99]. Another method is proposed by Suh [102] named imaginary complexity and calculated as equation 2.4:

$$C_I = - \log_2 (z/m!) \qquad (2.4)$$

where z is the number of acceptable sequences and m is the design tasks. Therefore, fuzzy values can be obtained to evaluate the coupling degree of a design. In another study, we combined ADT and Design Structure Matrix [97] to identify some design conflicts [44, 120].

2.1.1.4.2. Information Axiom

A problem can be solved by different designers through different functionally equivalent and acceptable set of solutions. The main goal of the information axiom is to define the best solution among alternatives providing a quantitative measurement to determine the complexity of a design [101].

Probability of success P_i is defined as the overlapping of the System Range (SR) that is provided by the DP and the Design Range (DR) that is provided by the design. In terms of probability of success (P_i) of FR_i and the information content (I_i), information content of the system (I_{sys}) for the case where all FRs are independent is represented in equation 2.5.

$$I_{sys} = -\sum_{i=1}^{n} I_i = -\sum_{i=1}^{n} \log_2 P_i \tag{2.5}$$

When an FR_i is not statistically independent, conditional probability of success $P_i|\{j\}$ is calculated with all other correlated FR_j where $j = 1,\ldots,m$. The information axiom states that the minimum information content I_{sys} is the best. The overlapping area between the design range and the system range is called the common range as shown in Figure 2.4. P_i is represented with proportion of them as given in equation 2.6.

$$P_i = \frac{CR}{SR} \tag{2.6}$$

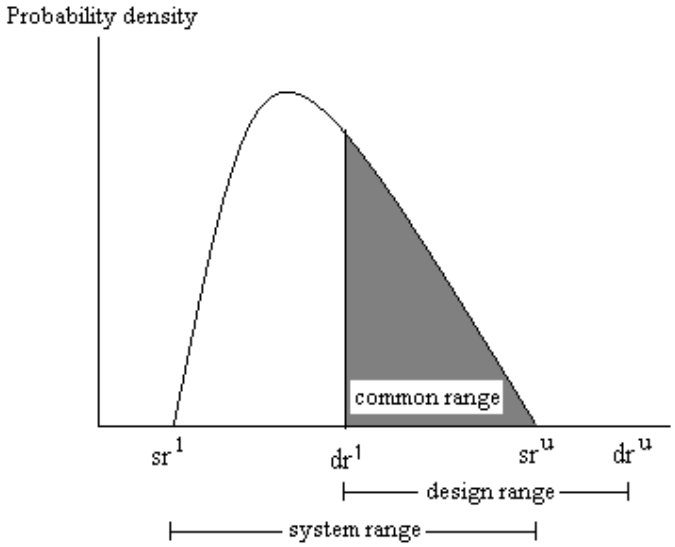

Figure 2.4 Design range, system range, common range, and system probability density (adapted from [101]).

Although, designers should satisfy the independence axiom first for acceptable designs, there is a relationship between independence and information axioms. Coupled designs cause more information content than decoupled designs. ADT states that there can be decoupled or uncoupled designs which have less information than coupled design.

A direct relationship between information measure and Taguchi's Quality Loss Function and a new measure of quality named Signal to Noise (S/N) ratio is identified in [69]. Depending on the study, when a DP has low information content, it has also low quality loss and high S/N ratio.

It is possible that semantically correct and deadlock free components may not be composed because of their constraints [111, 116]. Information axiom can be used for accordance measurements of component to design or component to component. For applying the information axiom, all components and applications must be designed based on our axiomatic design approach introduced in Chapter 4. Although it is not mandatory to define the design range of components or

applications, some of them can have this definition. For example, we can assume that there are two components. One of them is a *wind turbine* that produces *electricity* from *wind speeds* in the range of 5 to 25 km/hr and the other is the *environment* component that produces a range of wind speeds that is 50 to 100 km/hr. These two components are suitable to compose in terms of their interfaces but their design ranges are not. That means if we compose these components, the *wind turbine* component will not do anything. Also application developer can define design ranges such as a *wind tribune* application that can run with winds of 1 to 10 km/hr strength. We have utilized the information axiom to calculate congruity among methods of components [111, 116]. Regular information content calculation is based on the FR and DP ranges. The client server relationship between FRs and DPs can also be among methods since methods can call other methods. A designer only concentrates on the satisfaction of the FRs with DPs. DPs such as methods need other methods which are not considered in design. When a method is called, the method can call other methods of components if all required components which are required are in the system. We have identified three cases to utilize information content between methods:

1. This is regular form of calculation of information content. An FR satisfied by a DP is shown in Figure 2.5.

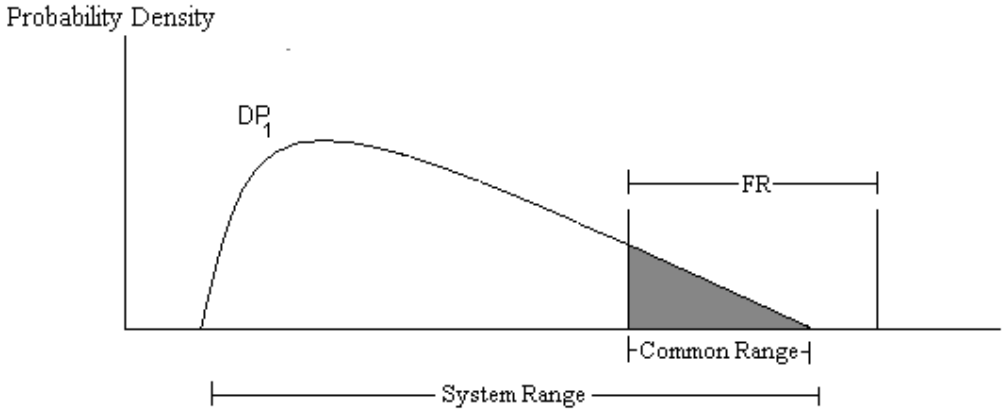

Figure 2.5 Probability of DP_1 success with specified FR (adapted from [111, 116]).

2. This case can occur when the range of FR is not defined and a DP calls another DP. In this situation, we can calculate method-to-method information content. If DP_1 is called by DP_2 then the design range of DP_1 can be accepted as the system range as shown in Figure 2.6.

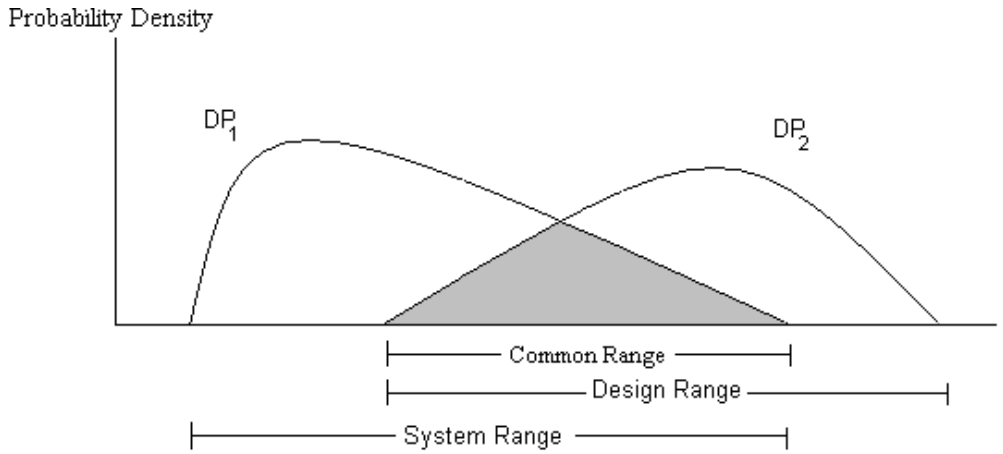

Figure 2.6 Probability of success graph of composed two components (adapted from [111, 116]).

3. For the calculation of information content of composed methods, intersection area of DPs can be accepted as system range as shown in Figure 2.7. This depicts the composed DPs harmony within the given design range that correspond to a FR. Common area of DPs' ranges forms the system range. Information content calculated based on the common area of the intersection of FR's design range and the system range.

Applications are formed from components and components share methods. We have applied this method to evaluate congruity of components and to guide designer about detecting conflicts among components.

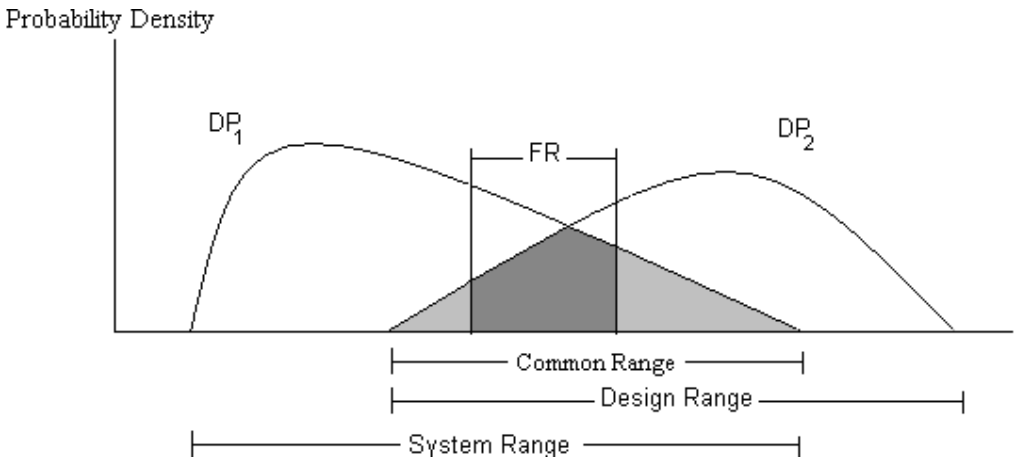

Figure 2.7 Probability of success for a composition of two components with specified FRs (adapted from [111, 116]).

Information content represents the FRs' satisfaction by DPs. Similarly sum of the information contents of methods in a component represent the satisfaction degree of the component depending on other components. Value of information content will increase based on the number of methods, and ranges. Evaluation of the value of information content is left up to designer. It should be noted that zero value represents the best congruity and infinity represents the incongruity. In the incongruity case, designer has to decide which components will be changed or modified. Designer can decide to change or modify the component that has infinity value or its neighbors cause the infinity value.

In our case study [111], we generated seven components ($C_{0...6}$) and their related eleven DPs with design ranges that are as listed in Table 2.2. Publish-subscribe information is obtained from Figure 2.8. For instance, C_0 publishes DP_1**,** DP_{10} and subscribes to DP_0**,** DP_3**,** and DP_9. As an example, C_0 subscribes DP_0 with ranges between 50 and 150, and publishes to DP_1 with ranges between 30 and 100 as shown in Table 2.2.

Table 2.2 Design Ranges of Components (adapted from [111, 116])

	DP_0	DP_1	DP_2	DP_3	DP_4	DP_5	DP_6	DP_7	DP_8	DP_9	DP_{10}
C_0	50, 150	30, 100		60, 70						10, 50	100, 200
C_1		5, 45	300, 400		1000, 3000	50, 200				20, 30	
C_2			100, 350	0, 100					30, 150		50, 150
C_3					500, 1000						
C_4						0, 150	10, 150	100, 200			
C_5								0, 300			
C_6									0, 5000		

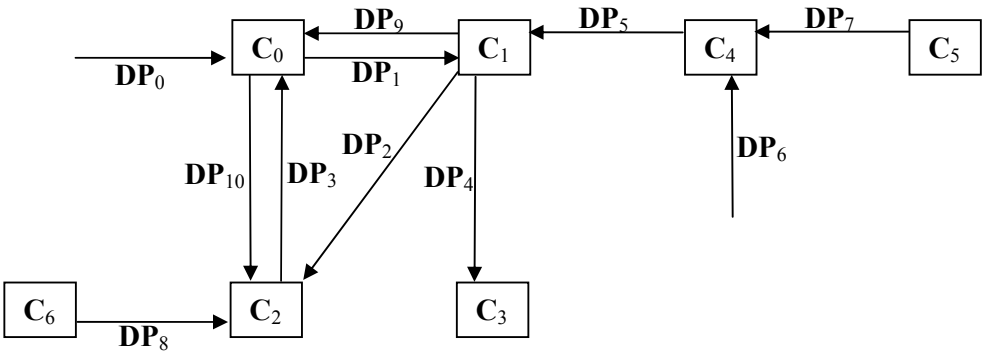

Figure 2.8 Mature domain (adapted from [111, 116])

We assume the FR design ranges are specified by application designer as listed in Table 2.3. We can obtain the probability of success for FRs as listed in Table 2.4 and total information contents are listed in Table 2.5. There are only three

components that relate with FR's design ranges as listed in Table 2.4. We know that smaller information content means better fit for composition. If information content of component is not infinity then these components can be accepted for composition. However, if there are similar components then the component which yields less information content is selected.

Table 2.3 FR design ranges (adapted from [111, 116])

	DP_1	**DP_3**	**DP_9**
Design Ranges	10 - 40	50-150	0-100

Table 2.4 Probability of success for FRs (adapted from [111, 116])

Components	**DP_1**	**DP_3**	**DP_9**
C_0	0.142	1.0	1.0
C_1	0.75		1.0
C_2		0.5	

Table 2.5 Information content of components corresponding to the related design ranges (adapted from [111, 116])

Components	**Information content**
C_0	2.807
C_1	0.415
C_2	1
C_3	0
C_4	0
C_5	0
C_6	0

When our third approach is applied for calculating the information content (utilizing intersections), results in Table 2.6 are obtained. Information content of C_3 is calculated as infinity. We know that infinity value depicts the incongruity between components. Therefore, C_3 or connected component (C_1) has to be modified or replaced with another one in the domain or a new component should be developed.

Table 2.6 Information content of application components

Components	Information content
$\mathbf{C_0}$	0
$\mathbf{C_1}$	1.16
$\mathbf{C_2}$	7.38
$\mathbf{C_3}$	infinity
$\mathbf{C_4}$	1.5
$\mathbf{C_5}$	0
$\mathbf{C_6}$	0

2.1.2. Axiomatic Design of Object-Oriented Software Systems

One of the applications of ADT is Object-Oriented (OO) software systems [33, 34, 101]. A systematic OO programming methodology is represented in Figure 2.9. OO development with respect to ADT starts with considering the customer needs and continues towards completing a design matrix obtained and then the design matrix is mapped to class diagrams as depicted in Figure 2.10. The created class diagrams are implemented and integrated. In this methodology, FRs are accepted as an object which has behavior. The DPs represent the DATA which are used by behaviors. FR-DP dependencies are represented in a matrix and mapped to class diagrams as methods. We have extended this approach for component oriented software engineering, as described in Chapter 4. In another OO software engineering

approach based on ADT [33, 34, 91, 101], functional requirements are represented with use case diagrams.

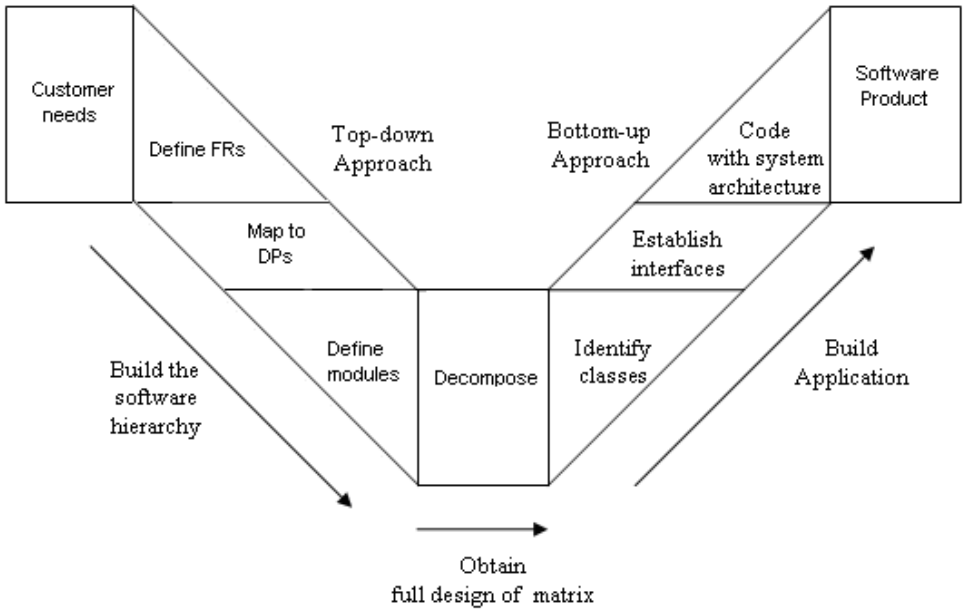

Figure 2.9 Systematic OO Programming with Axiomatic Design (adapted from [34])

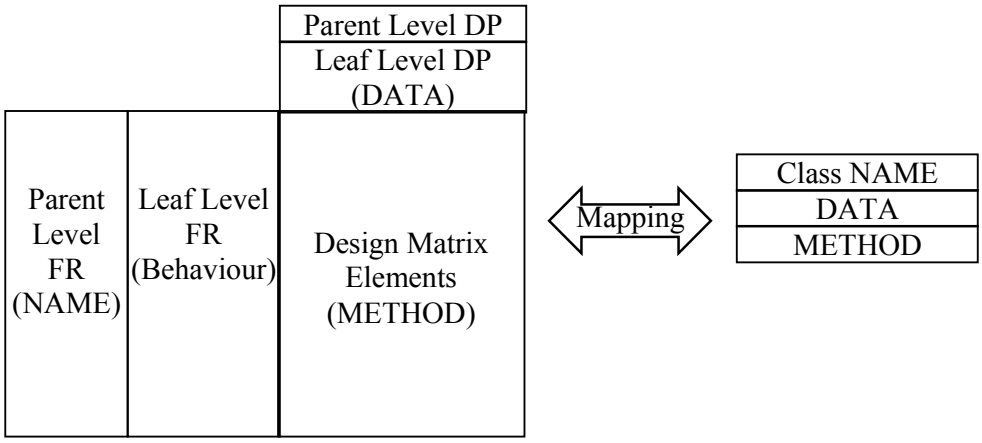

Figure 2.10 Mapping between design matrix and OO class diagrams (adapted from [101])

2.2. High Level Architecture and Object Model Template

Both High Level Architecture (HLA) [54-56] and its ancestor Distributed Interactive Simulation (DIS) [51, 52] were developed by the US Department of Defense to provide a common architecture for simulations. HLA provides re-usability and interoperability among federates. The first version of HLA was released in August 1996, and the final version was released in March 1998. The responsibility for HLA evolution was moved to IEEE's Simulation Interoperability Standards Committee in 1997 [81]. The committee has released three standards to define the core specifications [109]:

The Framework and Rules [54] (IEEE Std. 1516) specify the federation and federate responsibilities through ten defined rules. HLA Federate Interface Specification [55] (IEEE Std. 1516.1) defines the standard services of an interface to runtime infrastructure (RTI). The RTI is a simulation-oriented middleware that provides services. The RTI describes the interface between federates and the RTI in six classes of services, namely federation management, declaration management, object management, ownership management, time management, and data distribution management. For using these services, HLA application programming interfaces are prepared for various programming languages (C++, ADA 95, Java, and WSDL). Therefore, federates implemented in different platforms can communicate with others in federations.

Figure 2.11 depicts the structure of communication between federates. All communication must be provided by the RTI. Publish/subscribe methods are used for the communication of federates in HLA simulations. A federate that wants to announce a variable or an interaction declares to the simulation environment that it will publish. Federates, that are interested in the variable or interaction that is published, subscribe to it [56]. Federates do not require to have information about the location of each other in the environment. For executable simulation, at least one publisher has to publish an attribute or interaction for other federates to subscribe to it. RTI controls the data and procedure flows among concurrently running federates like an operating system.

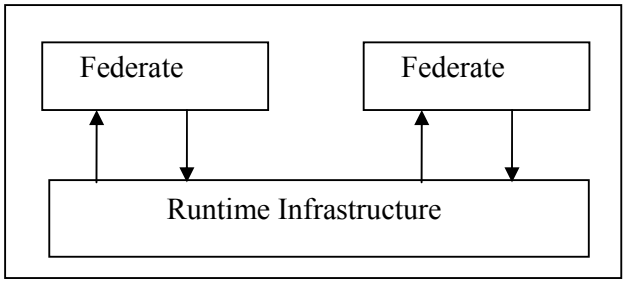

Figure 2.11 High Level Architecture based components communication (adapted from [118])

HLA Object Model Template (OMT) [56] is used to specify the interface of federates (known as Simulation Object Model (SOM)) that describes which items (objects, attributes, interactions (methods), parameters) will be exchanged with other federates. Also OMT is used to specify simulation objects which will be shared in a federation (known as a Federation Object Model (FOM)). A FOM file is used during the beginning of a federation; however, a SOM file is used only to describe the federate. Federates do not communicate with one another directly; therefore, each federate defines objects which will be shared (published) with others and required (subscribed) from others in the SOM file. Objects are defined in fourteen tables (object model identification, object class structure, interaction class, attribute, parameter, dimension, time representation, user-supplied tag, synchronization, transportation, switches, data type, notes, and FOM/SOM lexicon) of an OMT. These tables define all the information about the federate or the federation.

Federation Development and Execution Process (FEDEP) [57] introduce a process to create federations, promoting interoperability among federates. Our component oriented approach is applied to FEDEP process as defined in Section 4.2.2.

In 2005, the HLA Evolved Program Development Groups (EPDG) was established to evaluate proposals for improving IEEE standards in the areas of WSDL API for

IEEE 1516.1, fault tolerance, dynamic link compatible HLA API for IEEE 1516.1, XML schema for IEEE 1516.2, conformance specification, and additional flexibility in update rate [81]. In terms of component oriented approaches, a federate is called an "HLA component," and a federation that consists of more than one component is called a "distributed system" [84, 87]. An HLA component can be a computer simulation, a manned simulator, a simulation utility (data collector, passive viewer, etc.), or a simulation interface to live players [27, 29]. An HLA component can consist of one or more software components.

Interoperability has been considered from two perspectives: technical interoperability and substantive interoperability. Resolving technical interoperability issues ensures that the federation will run, but says nothing about the adequacy of the federation to accomplish its mission. Technical interoperability includes composition anomalies defined in [28, 47, 108, 109]. These anomalies should be detected and solved during composition of federates. Substantive interoperability is driven by the needs of the federation and has to be addressed by each federation in a federation specific way [8].

2.3. Component Oriented Software Engineering

All industries attempt to reduce cost and time required to develop increasingly sophisticated products without sacrificing reliability. Reuse is the primary goal of the components. There are various and similar definitions of components; one of them is [50] "software components are (binary) units of independent production, acquisition, and deployment that interact to form a functioning system." Definitions of components are not enough to define all properties of them. We have identified some common properties of component definitions as following:

- As defined by Szyperski [105], component reuse will be cost effective easier than redeveloping it. We can expand the easier term as cheaper, time effective etc. Their weight is dependent on the stakeholders. For example, important issue for some stakeholders is time, for another, cost.

- Components can be developed in different programming languages and then compiled. Therefore, binary form of components is language-neutral. For example, a component can be developed through a C++ environment, and it can be composed by EJB components.

- Components should provide enough information about themselves. This information should be what components are publishing as services and what they need from other components. Five level categories are introduced to represent the information in [13, 77] :

    - Syntactic Level (Basic Contract): Signature of the component (such as Interface Description Languages defined by CORBA, COM) which includes published operations, input output parameters, and possible exceptions. Most of the component interfaces include this level of information. However, signature of the component does not provide required information to compose components [127]. Since internal mechanisms are not known factors proposed, they cannot be considered during component quantification of third–party components [Bro96].

    - Behavioral Level: Semantic descriptions are represented. Since third party component users do not have access to component implementations, they can only expect the component to do the functionality what method names imply. To provide more information about components, Boolean assertions and pre and post conditions can be used. In COSEML [37], required relations among components are represented in a graphical form.

    - Synchronization Level: Concurrency issues are represented. One approach is synchronization policies attached to components. Another approach proposed by Yelling and Strom [127] introduce collaboration specification that consist of set of sequencing constraints that defines the legal ordering of messages.

    - Quality-of-Service Level: Nonfunctional requirements are represented such as maximum delay, average response, quality of response, etc. Quality attributes for components are introduced with more detail in [12, 60].

    - Non-Technical Level: Business oriented information such as submitted by, resource url, category, language, marketing type, version number contact address, price, etc. [11].

- Components can be developed or implemented independently.

- Components reduce maintenance costs. Modifications on components are local because of the inherent encapsulation of components.

- Components reduce test costs. Although, components are self tested, they also should be tested in the application based on such as a black box technique.

- Components increase the reliability of applications. Components are shared among various applications; therefore they are tested in different application sets. Component usage provides more reliable systems than newly coded systems.

- Components can be classified as visual (controls and containers) and non-visual components (command packages, interacts with visual objects such as spelling checker, library, business, and framework) [121].

- Components can offer many interfaces [37, 59].

We refer to components for two separate purposes: 1- as software components 2- as federates of HLA (HLA components) [54-56].

Inter-connection technologies .NET, CORBA and J2EE provide environments for components to create runtime instances of components, discover other components,

and communicate components. These technologies provide mainstream software buses for components to connection. Ideally, each component should be able to connect with components developed anywhere; bridges among technologies provide capability to connect components wherever there are problems due to different technologies.

In software engineering world, objects and components are confused most commonly. Although, components and objects have commonalities such as encapsulation, well defined interfaces etc., an object is not a component [105]. Object exists at runtime, while components are binaries. Components can be developed through highly collaborative objects (classes), but when they are compiled they form components. Users of the component do not need to know how a component is internally represented such as class diagrams etc. Therefore another difference is the granularity between objects and components. One of the important concepts in object orientated software engineering is inheritance. Instead of inheritance, composition is proposed in Component Oriented Software Engineering (COSE).

Other confused terminologies are Component-Based and Component-Oriented. In component-based approaches component development and integration is essential but Component-Oriented approaches are based on integration of already available components [37].

Some challenges exist that can be generally attributed to following:

- Multiple competing standards such as .NET, CORBA and J2EE [62].
- Lack of standards (such as documentation), as is often the case with separately designed components [62].
- Component interfaces present insufficient information about its capabilities and functionalities [13, 62, 70, 77].

- Component finding through internet is defined as a cumbersome effort in 2002 [121], and still continues. Some web sites for component searching are listed in [121].

- Lack of security policies. Especially during component selection security requirements should be identified and defined by components [76].

- Functional congruency: when composing with other components some incompatibilities can occur such as:

    - Type problem such as a method can get a parameter as integer and other component can try to call the method with real. To solve this problem interface mapping is required [127].

    - Protocol (synchronization or control) problems. [127]. To solve this problem, Yelling and Strom [127] introduce collaboration specification.

There are two approach to solving complex problems namely top-down and bottom-up. In bottom-up approach, components are iteratively composed into higher level components until a system that satisfies the customer expectations is emerged [10]. In this approach, components are being composed without considering the system. There is no idea about the whole system when composition is started. In top-down approach, most abstract requirement based on customer needs is specified and decomposition is started. Decomposition progress toward concrete components. In top-down approach, complex problems are solved by the "divide and conquer" approach, which seeks to divide the problem into simpler parts, solve them, and integrate into a viable solution [18, 96, 106]. Problem of the top-down approach is requirements analysis. Such as designer in the beginning can define abstractions wrongly because of missing requirements, inconsistent requirements etc. [10]. Practical experiences indicate hybrid approaches being the best for component based-oriented systems [10]. A kind of approach is introduced by Dogru and Tanik named COSE Modeling Language (COSEML) [37]. COSEML stresses that while top-down decomposition is continuing, existing components should be kept in mind.

2.4. COSE Modelling Language

Component-Oriented development environments generally assume the existence of already developed components as an integral part of a successful Component-Oriented development process [37, 59]. COSEML is a graphical modeling language utilizing a single hierarchy diagram supports this. Modeling starts with a top-down decomposition of a system while defining its modules in such a way that those modules can be matched by available components. Therefore, system development is reduced to a decompose-find-integrate operation instead of define-develop from scratch. Component-Oriented development environments generally assume the existence of already developed components as a requirement for a successful Component-Oriented development process [37]. To be effective, available components should be considered while decomposing the system. COSE separates the parts of the system (components) from its abstract specification. A COSEML specification consists of two parts: abstractions and components as shown in Figure 2.12. Decomposition starts with a package and a package can include more than one component or sub packages. Each component has zero or more interfaces to represent its *properties*, *methods in*, *methods out*, *events in*, and *events out* elements. The *methods in* and *events in* represent the published services of a component and the *methods out* and *events out* represent the subscribed services of other components. COSEML representation forms the static view of the system structure. There is a need for a dynamic model to verify compatibility of static structure with requirements. Therefore, *logical and run- time* collaboration diagrams are used [35, 123]. If a given set of abstractions are not enough to provide the required functionality then decomposition must be reconsidered.

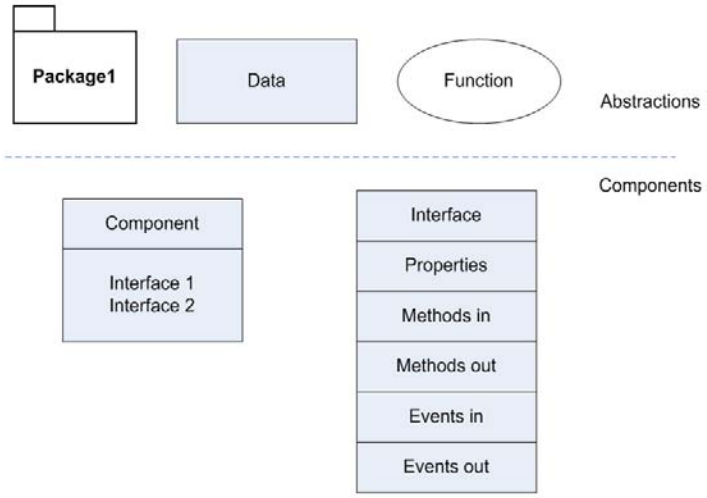

Figure 2.12 COSEML symbols used in this study (adapted from [114])

Component oriented approaches have not incorporated the relationships among requirements (FRs) and solutions (Design Parameters (DPs)). Although interface concepts are important, interfaces do not include enough information to describe components. Therefore, we presented an approach [114, 116, 118] that composes the axiomatic design and COSE concepts. This approach provides an environment to design and develop components and applications in a mature domain. A mature domain is formed from components that are designed using this approach. This approach contains the design matrix that is used to keep relationships between FRs and DPs. COSEML and design matrix depict the different views of systems. While COSEML presents the system view with components and their integration, design matrix presents the relationships among FRs and DPs (*property*, *method*, and *event* names) as shown in Figure 2.13. Creation of design matrices suffers from missing of a method to specify the dependency relationships among FRs and DPs. It is completely left to the developer. In [114], we proposed usage of the collaboration diagrams to find these relationships. After specifying components, they can be verified depending on specific scenarios. Scenarios are prepared using run-time collaboration diagrams. Assuming that all components are designed with Axiomatic Design, giving two advantages:

1. We can see the relations among DPs (*attributes*, *methods*, *events*) in one component,

2. Design matrix carries the information on why a specific DP is used or which functionalities are satisfied.

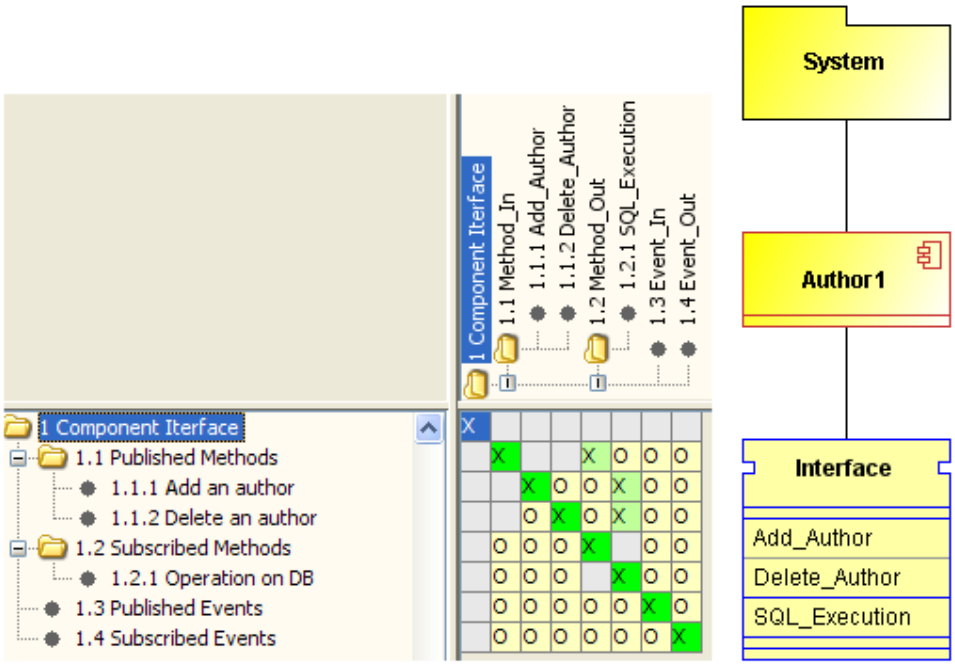

Figure 2.13 Axiomatic design representation (left side) and COSEML representation (right side)

The method includes the construction of the COSEML model and the design matrix of ADT corresponding to the system. Table 2.7 lists the steps to perform our suggested development method in [114].

Table 2.7 Development Process (adapted from [114])

Step	Description
1	Construct the COSEML decomposition and compatible design matrix for abstract levels.
2	Prepare the logical collaboration diagrams for abstract functions of COSEML diagram.
3	Specify available and missing components
4	Add FRs and DPs of components, gathering from their design matrices, to the system design matrix. If there is no candidate component then define components with interfaces and specify corresponding FRs and DPs in the system design matrix.
5	Prepare the run-time collaboration diagrams for each specific scenario.
6	Verify the components' functionality with run-time collaboration diagrams
7	Develop missing components depending on run-time collaboration diagrams and design matrices.
8	Integrate components.

2.5. Design Structure Matrix

The Design Structure Matrix or Dependency Structure Matrix (DSM) was developed by Steward [97] for representing and analyzing task dependencies. DSM also provides a visual representation to detect requirement of compositions and decompositions [15]. Categorized DSM applications are given in [15] in a form such as component-based representation that represents component relationships, team-based representation that represents team relations; activity-based

representation that represents information flows, and parameter-based representation that represent physical design parameters relationships. Since DSM uses a square matrix called N^2 diagram. We utilized DSM for component-based applications where the elements listed are one-to-many comparisons of four components and component-component interactions as shown in Figure 2.14 and Figure 2.15. Some applications of DSM's cells include more than one information such as spatial, energy, information and materials [92]. For the simplicity, relations represented with "X" as in design matrix of ADT are depicted in Figure 2.14. In the matrix in Figure 2.14, the publisher components are listed in the columns and the subscriber components are listed in the rows, such as component B, provides some information to component C and component D. Also it can be inferred that component D is a subscriber component publishing nothing to other components.

	A	**B**	**C**	**D**
A	**A**			
B	**X**	**B**	**X**	
C	**X**	**X**	**C**	
D		**X**	**X**	**D**

Figure 2.14 Example Design Structure Matrix (adapted from [120])

Relations are used to calculate coupling degrees of components similar to ADT. Coupling degrees of components show their run time communication way; uncoupled components can be executed concurrently, decoupled components can be executed sequentially since one component influences the behavior of another element in a uni-directional fashion, and coupled components may not be executed together. Coupled components assembly can result in architectural mismatch when trying to integrate components with incompatible interaction behavior resulting in deadlocks, live-locks, or failing to satisfy some desired functional properties of the system [58]. Complexity of DSM matrix is reduced using partitioning algorithm [97, 99] that rearranges the DSM matrix.

In one of our studies [120], we utilized DSM to detect coupled components. However, not all coupled components can be detected by DSM which cause deadlock. Component dependency relation in DSM matrix represents high-level relationships among components. Low-level (wiring level) relations have to be considered. We proposed to use Axiomatic Design Theory and design matrix to identify the method and attribute interactions of the components and discover deadlock situations among the coupled components [120].

DSM can be used to detect this kind of interaction that could lead to coupling. We create a DSM shown in Figure 2.15 by considering the transition property represented in Figure 2.8, as indicated by the conclusion that $C_0 \rightarrow C_3$ (C_0 publishes method(s) to C_3), since $C_0 \rightarrow C_1$ (C_0 publishes method(s) to C_1) and $C_1 \rightarrow C_3$ (C_1 publishes method(s) to C_3). The values in the cells of the DSM show distance values; e.g. In Figure 2.8, the distance between the C_1 and C_3 is shorter than the distance between C_0 and C_3. One value in the cell represents the direct connection between components. Such as, C_5 and C_4 components are directly connected as shown in Figure 2.15.

	C_5	**C_6**	**C_4**	**C_0**	**C_1**	**C_2**	**C_3**
C_5	**C_5**						
C_6		**C_6**					
C_4	1		**C_4**				
C_0	4	2	3	**C_0**	**2**	**1**	
C_1	2	3	1	**1**	**C_1**	**2**	
C_2	3	1	2	**2**	**1**	**C_2**	
C_3	3	4	2	2	1	1	**C_3**

Figure 2.15 DSM table for the components in the example (adapted from [120])

The DSM table as depicted in Figure 2.15 shows that C_0, C_1 and C_2 are tightly coupled. Coupled components can cause deadlock but not always. Our method to

confirm whether the coupled components can cause deadlock or not, is by developing the components using Axiomatic Design principles and by mapping their DPs and FRs using a design matrix. For instance the design matrices for each of the coupled components, C_0, C_1 and C_2 could be developed based on same FRs with different DPs. The design matrices for three components are shown in Figure 2.16 and Figure 2.17. Components can be used in composition only if their relations do not occasion the components to deadlock as depicted in Figure 2.16. If the coupled components do not cause a deadlock, they can be used to form a super-component in the mature domain. Coupling test is a very difficult task for complex systems which includes various components. To handle this problem, we have proposed an approach [112] to utilize communicating sequential process (CSP) [48] as defined in section 2.6.

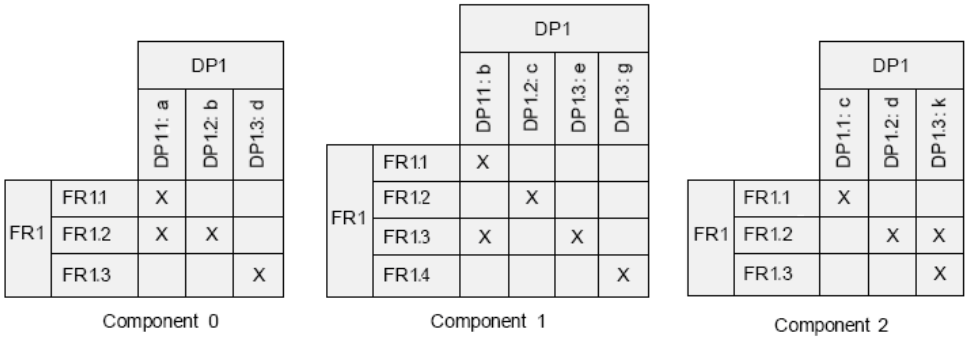

Figure 2.16 Axiomatic design diagrams of components 0-2 (deadlock free) (adapted from [120])

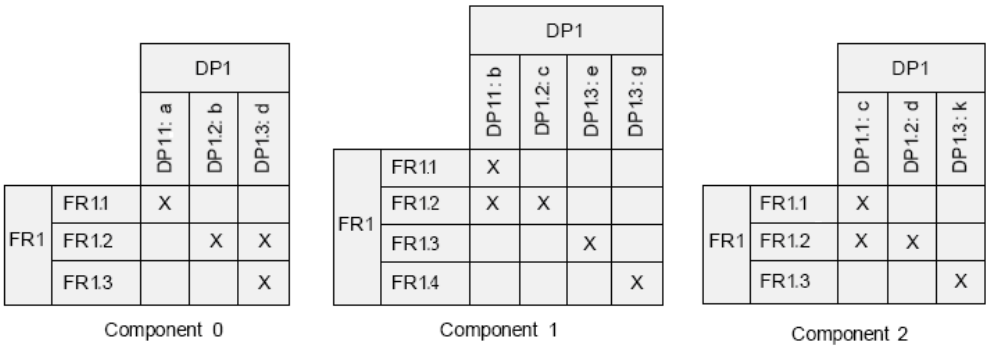

Figure 2.17 Axiomatic design diagrams of components 0-2 (deadlock) (adapted from [120])

2.6. Communicating Sequential Process

Communicating Sequential Processes (CSP) is a process algebra introduced by Hoare [48]. CSP is a language and is supported by the tools: Failures-Divergence-Refinement (FDR2) [40] for model checking and Process Behavior Explorer (ProBE) [41] for state machine based models. Wright [2, 4] is an architecture description language that uses a CSP like notation to describe components' ports and roles. For instance, HLA Runtime Infrastructure (RTI) [55] is formalized using Wright to detect deadlocks and race conditions [3]. It should be noted that developed tools translate the Wright representation to CSP for utilizing the FDR2 tool. CSP can also be used for modeling complex service choreography for checking for deadlock among integrated services [128, 129].

Table 2.8 CSP expressions used in this article

CSP Expression	**Explanation**
P[| A |] Q	P and Q processes are partial interleaved parallel composition. A is the set of the events. If A is empty then composition of P and Q behaves interleaved parallel.
P ||| Q	P and Q are interleaved parallel

Table 2.8 (Cont'd)

e-> P	Event e performed first and then Process P is executed after an external trigger occurred.
SKIP	Successfully termination
STOP	Deadlock
Datatype x = a | b | c	Defines x datatype with a set of alternatives
Channel e	Defines event e
Channel e:x	Defines event e with x datatype
e ? a	Defines input on event e of an item defined during channel definition. As defined in datatype, instead of an item, b or c items can be used.
e ! a	Defines output on event e of an item. After this expression is performed, e?a expression in another process in waiting situation can be performed. Input and output expressions are used to provide synchronization.
Union	Unions the sets.

In CSP, processes defined statically include a set of events. Events are atomic and provide synchronization among processes. They are used to define the behavior of processes. More than one process can be executed at a time in concurrent systems. This causes well known problems such as deadlocks. CSP theory and FDR2 are used for checking defined processes in terms of traces, stable failures, and failure-divergence models. In this section, we will concentrate on the traces to check for the deadlock situation in the composed system using the FDR2 tool. CSP expressions that will be used in this article are listed in Table 2.8.

In one of our studies [112], we have utilized CSP to detect deadlocks of a system constructed from components. Deadlock also represents the availability of missing components. We have defined a method to translate design matrices of components to CSP language in order to detect deadlocks in federations and compatibilities

among federates in terms of system requirements. Component interfaces are represented in COSEML notation. A process concept in CSP corresponds to a partial or a whole component. Methods and component events are defined as events in terms of CSP and they are represented in a process. Input and output definitions in COSEML notation are represented in CSP as listed in Table 2.9.

Table 2.9 COSEML and CSP representations

COSEML representation	**CSP representation**
Component	Process
Published method *a*	Output event (e ! a)
Subscribed method *a*	Input event (e ? a)
Published event *a*	Output event (e ! a)
Subscribed event *a*	Input event (e ? a)

Systems can be defined in an application design matrix which includes components composed to satisfy functional requirements. The application design matrix is represented with the CSP language to utilize the FDR2 tool. The required mapping mechanism is listed in the following rules [112]:

- Input and output definitions are specified based on dependence relationships of published methods or events. For instance, *Method 1* requires *Event 1* is represented as "Event1? e1 -> Method1! m1"

- A Component is represented as a process that consists of one or more sub processes as shown in Table 2.10.

- An application is also represented as a process and it is formed from one or more component processes.

- Processes are composed based on shared methods or events among processes. If there is no shared item(s) than the "|||" term is used to connect processes. If there are, then "[| |]" is used.

- Shared items among processes are looked up from the design matrix. If there are events defined as input events (subscribed) and required output events (published) from other components, they must be considered during the forming of the application process.

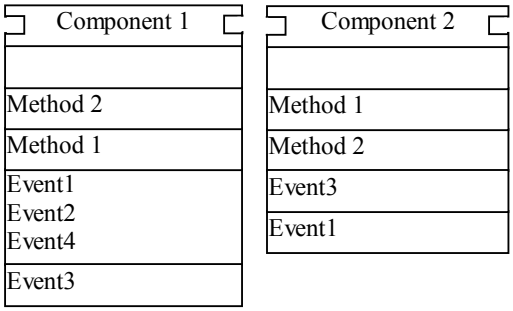

Figure 2.18 COSEML representation of Component 1 and Component 2 (adapted from [112])

Based on the mapping mechanism and design matrices of components as depicted in Figure 2.19 and Figure 2.20, we can create CSP language representation of the application design matrix as depicted in Table 2.10. *Component 1* has four published items therefore there are four sub processes which are *C1_SUB1*, *C1_SUB2*, *C1_SUB3*, and *C1_SUB4*. Only *Method 2* is shared between *C1_SUB1* and *C1_SUB2*. Processes *C1_SUB3* and *C1_SUB4* can be executed concurrently since there is no shared item between them. *Component 2* has two published items therefore there are two sub processes namely *C2_SUB1* and *C2_SUB2*. Only *Method 1* is shared between *C2_SUB1* and *C2_SUB2*. Composition of *Component 1* and *Component 2* is represented in Table 2.10 as one process namely

APPLICATION. The process composes components with their shared items namely *Method 1*, *Method 2*, *Event 1*, and *Event 3*.

We tested the executable CSP codes in Table 2.10 and obtained a deadlock free application. Although, coupling is available between *Component 1* and *Component 2* as shown in the design matrix in terms of DSM, we can conclude that components which are sharing methods and events are not forming cycles. In this application, all components are satisfied in terms of their required interface items. Otherwise, FDR2 tool warn us about deadlock which means some of the processes require other process (es) to produce required items.

Table 2.10 CSP representations of an Application that compose Component 1, Component 2 (adapted from [112])

```
datatype D_i1= i1, D_i2= i2, D_e1= e1, D_e2= e2, D_e3= e3, D_e4= e4
channel Method1:D_i1, Method2:D_i2, Event1:D_e1, Event2:D_e2,
Event3:D_e3, Event4:D_e4
--------------------------------------Component 1----------------------------------------
C1_SUB1 = Event1?e1 -> Method1?i1 -> Method2!i2 -> C1_SUB1
C1_SUB2 = Method2?i2 -> Event3?e3 -> Event4!e4 -> C1_SUB2
C1_SUB3 = Event1!e1 -> C1_SUB3
C1_SUB4 = Event2!e2 -> C1_SUB4
C1 = (C1_SUB1 [|{|Method2|}|] C1_SUB2 ) || C1_SUB3 ||| C1_SUB4
--------------------------------------Component 2----------------------------------------
C2_SUB1 = Event1?e1 -> Method1!i1 ->C2_SUB1
C2_SUB2 = Method1?i1 -> Method2?i2 -> Event3!e3 -> C2_SUB2
C2 = (C2_SUB1 [|{|Method1|}|] C2_SUB2 )
--------------------------------------Application----------------------------------------
APPLICATION=(C1[| union( union( union ({|Event1|}, {|Method1|}),
{|Method2|}), {|Event3|})|] C2)
```

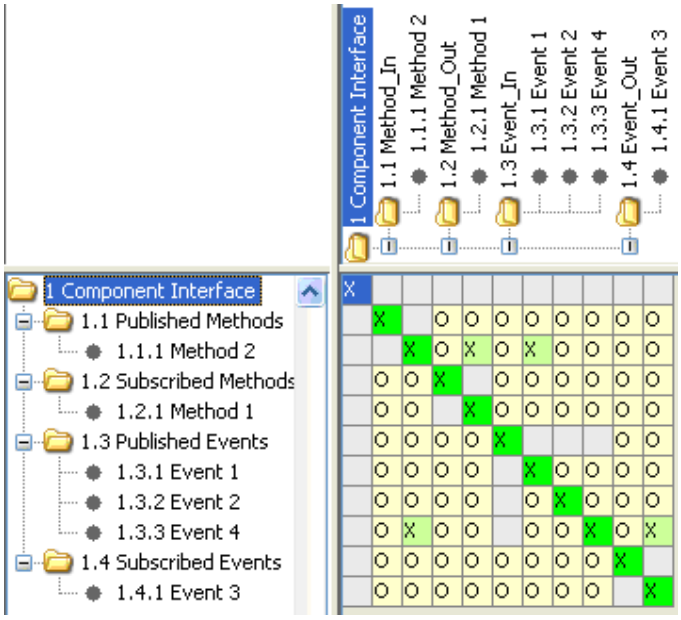

Figure 2.19 Design matrix of Component 1 (adapted from [112])

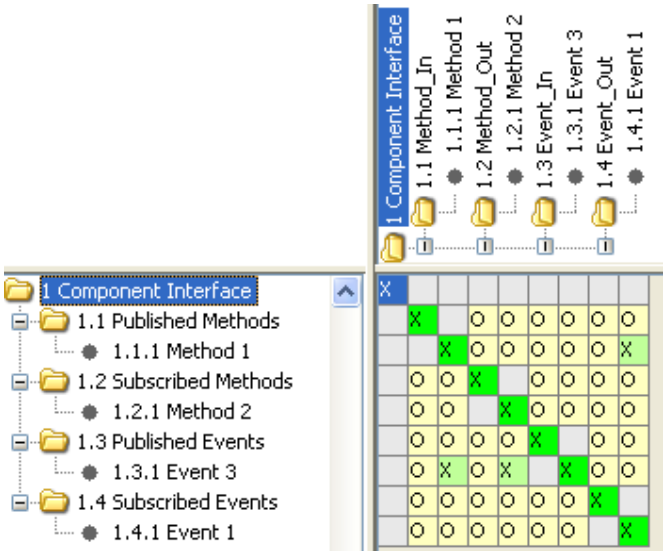

Figure 2.20 Design matrix of Component 2 (adapted from [112])

2.7. Feature Model

Customer needs are mapped to functional requirements but there is no defined straightforward method in ADT for narrowing this gap. Customers concentrate on the system features and designers focus on solutions [124]. Feature concept is introduced by Kang [64] to define information about the domain in the Feature Oriented Design Analysis (FODA) [64] and its enhanced version: Feature Oriented Reuse Modeling (FORM) [65]. There are various definitions of a feature [30, 53, 75, 124] but all definitions have a common point that features are stakeholder (user, customer, developer, domain expert, etc.) visible aspects and they represent the commonality and variability of products in terms of aspect, quality or characteristic. It is essentially an abstract or product characteristic that both costumer and developer understand [67]. They have also the capability to represent customer needs in a domain. Therefore, Feature Modeling can be defined as a domain modeling technique [25]. There are some example domains where Feature Model is applied such as the bulletin board system domain [65], the private branch exchange domain [66], web services domain [94], elevator control systems [72], bank account and transaction systems [75].

Features should be well-known by both customers and designers and can be functional (services or operations) and nonfunctional (capacity, usage, cost, and other quality attributes) [67]. Main source to be used in the identification of features are books, user manuals, experiences of experts, customers' domain knowledge, terminology, etc. [64]. Different stakeholders have different interests about features therefore features are classified in terms of capabilities (services and non-functional characteristics), domain-technologies (way of implementing services), implementation techniques (synchronization mechanisms), and operating environments [64]. Features are organized in a graphical model called Feature Model that represents distinctiveness and commonalities among features in a hierarchical view [64, 65, 67]. Feature Models can be represented in both graphical [23, 64] and textual forms such as Feature Definition Language [31] , feature diagram algebra [31], textual specification language [65], semantic model [61] and

XML based [17]. Features are organized in multiple levels of increasing detail in the Feature Model [25]. Feature Model is utilized by various methods, architectures such as reuse driven software engineering business [43], aspect-oriented programming [73], generative programming [22], product line software engineering [20, 67, 68], reengineering [85, 86], object oriented software engineering [66, 72, 90], component based systems [63, 65, 88, 110, 115, 117], feature oriented programming [6].

The core feature diagram, presented in [64], has been expanded with the introduction of the new extensions and variations in the recent years [23, 24] and still there is no consensus on notation of Feature Models [9]. Set of features interact to define purpose of the product [39]. Five main types of feature interactions namely intentional interaction, resource-usage interaction, environment induced interaction, usage dependency, and excluded dependency are identified [39]. There are three types of relationships between parent and children features: composed-of, generalization/specialization, and implemented-by. Features are represented in a diagram formed as a tree and connected to their parents in the diagram through mandatory, optional, and group relationships as represented in Figure 2.21. To avoid redundancy, a feature can be child of more than one parent feature, however in this situation tree form is broken. To handle this problem Czarnecki et al. [22] proposed the sub-models for reusing and connecting a feature to various parent features [16]. Mandatory features are common features among all products and they have to be selected in all products. On the contrary, optional features may take place in some products and selection of optional features is left to the customers hence letting the customer define a product. There are two kinds of grouping among features namely OR and alternative (XOR). In OR grouping, one or more children features can be selected however in alternative grouping only one of the children features can be selected. Also features can specify constraints that define exclude and require relationships with other features. When a feature is selected, required other features have to be selected and excluded other features have to be un-selected.

The level of standardization in a field can perhaps indicate the maturity of engineering in the field [64]. For example, the car domain is matured and therefore no one designs all parts from the beginning. In the designing of a new car, probably it will not be necessary to design a new transmission. Perhaps it will be sufficient to select its feature: automatic or manual as depicted in Figure 2.21. In this diagram, Air Conditioning is an optional feature and in order to be selected, the car is required to have 100 horse-power of engine capacity. Transmission can be automatic or manual.

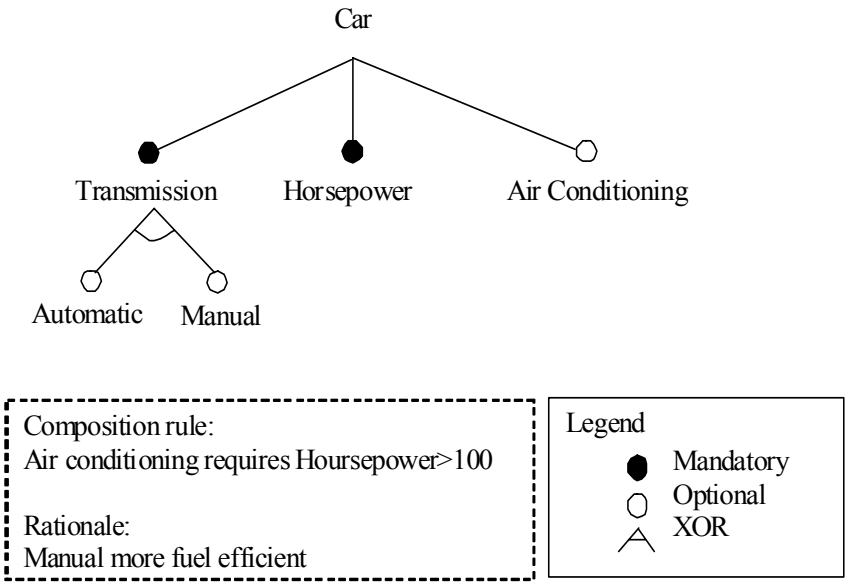

Figure 2.21 Feature Model (adapted from [64])

Amount of features and constrains are important factors, of complexity of Feature Models. For instance, as one of the mature domains the automotive industry has Feature Models consisting of up to 10.000 features [8]. Relationships among features are defined through rules. It should be noted that independently defined relations can be inconsistent with other relations [95]. Therefore, we have identified two methods to handle complexity in multi-level feature trees [93] and checking the

consistency of the Feature Model. Checking method is a challenging problem [9, 14, 89, 125, 126]. Descriptive power of ontologies is applied to Feature Models [14, 25, 126]. Feature Models are represented in Web Ontology Language (OWL) for tool support utilizing query and constraint mechanisms [25]. Consistency can be required in a Feature Model, or instance Feature Model which is created through feature selections in the Feature Model. We have utilized number 7 in the following list, as defined in number 8, for this work. There are eight approaches to handle this problem:

1. Approach [9] is utilizing Constraint Satisfaction Problem [9] and Java constraint solvers through Feature Model with cardinalities translating into Constraint Satisfaction Problem.

2. In approach [7], logic truth maintenance systems [7] and SAT solver [38] are utilized to debug by confirming compatible and incompatible feature sets.

3. In approach [95]., feature computation tree model is proposed to consistency checks of requirements

4. Approach [26] proposing the Object Constraint Language (OCL) [83] and SAT solver [38] to verify feature configurations.

5. In our approach [14], we represented the Feature Model with utilizing OWL and Semantic Web Rule Language (SWRL) [49, 79] based on feature notations [23, 24]. As depicted in Figure 2.22, we proposed a three-layer approach for representing the Feature Models in OWL. Meta ontology as depicted in Figure 2.22 is a base to create Feature Models. During feature configuration, the user creating instances of features have to obey constraints. As well as parent child relationships among features, the most common constraints in the Feature Model, the “requires” and “excludes” are required to obey. These rules are defined by using SWRL and checked by the supporting rule engine. Previous approaches are applied after Feature

Model is created, but our approach directly involves the user with consistency.

6. Approach [89] is similar to our approach [14] and both papers are simultaneously published. They are different in terms of representation of Feature Models in ontology.

7. Approach [126] is about utilizing OWL [78] and reasoning engine Fast Classification of Terminologies (FACT++) [122]. Instance of the Feature Model have to be consistent with the core Feature Model and its constraints. Constraint violating features can be detected through executing a reasoning engine.

8. We have expanded the seventh approach with our axiomatic design ontology and ADCO tool. Since feature diagram view of ADCO tool saving all information about parent child, required, excluded etc. relations, we have omitted the hierarchical representation of features as depicted in [14] and represented only features and their dependencies through setting constraints. Reasoning engines provide the information about consistency of an instance of the domain Feature Model as defined in [126]. Detailed information is given in Chapter 3.

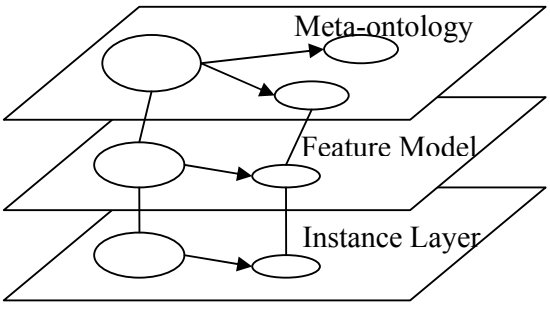

Figure 2.22 Three layers of the Feature Models representation in OWL (adapted from [14])

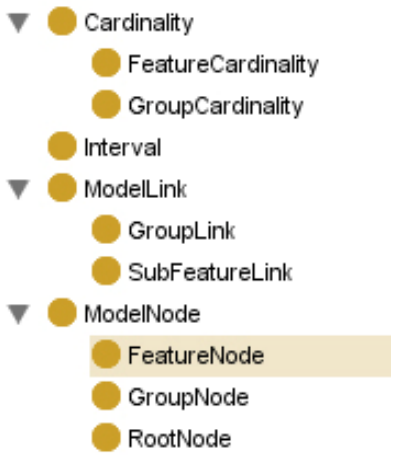

Figure 2.23 Meta-ontology classes (adapted from [14])

Axiomatic design theory considers the customer and functional domains separately. Domain analysis artifacts are not employed to specify functional requirements but they are used as an input to functional analysis process [64]. The system capabilities are specified by customer needs represented in a Feature Model. Features can be not only requirements but also implementation level information. Features are different from the functional requirements and a mapping is required. Therefore, we utilized the Feature Model in the customer domain of ADT to create common understanding between designers and customers.

Feature Models are used by different approaches in different ways:

- Feature Models are used to represent variations and communalities among products or components [20]. Similar approach proposed for Service Oriented Architectures with utilizing OWL. Therefore there is a direct connection between features and components such as defined in [63, 65]. In [63], many-to-many relationship is identified between features and components thus a feature-based component selection is targeted.

- Features are mapped to reference architecture including subsystem, process, and module models and the reference architecture is used to obtain components [65].

- Features are mapped to behaviors that are described by scenarios. Requirements or goals are achieved by scenarios [71].

In one of our studies [117], we have identified relationships between features and Object Model Template (OMT) [56] items in High Level Architecture (HLA) [54] based simulations. As illustrated in Figure 2.24, maneuver feature is connected to *Turn_Right*, *Get_Wind_Speed*, and *Coordinate_XYZ* OMT items. Relationship between features and OMT is hidden from end-users. When user selects maneuver feature, federates (components) that are publishing or subscribing to related OMT items are searched.

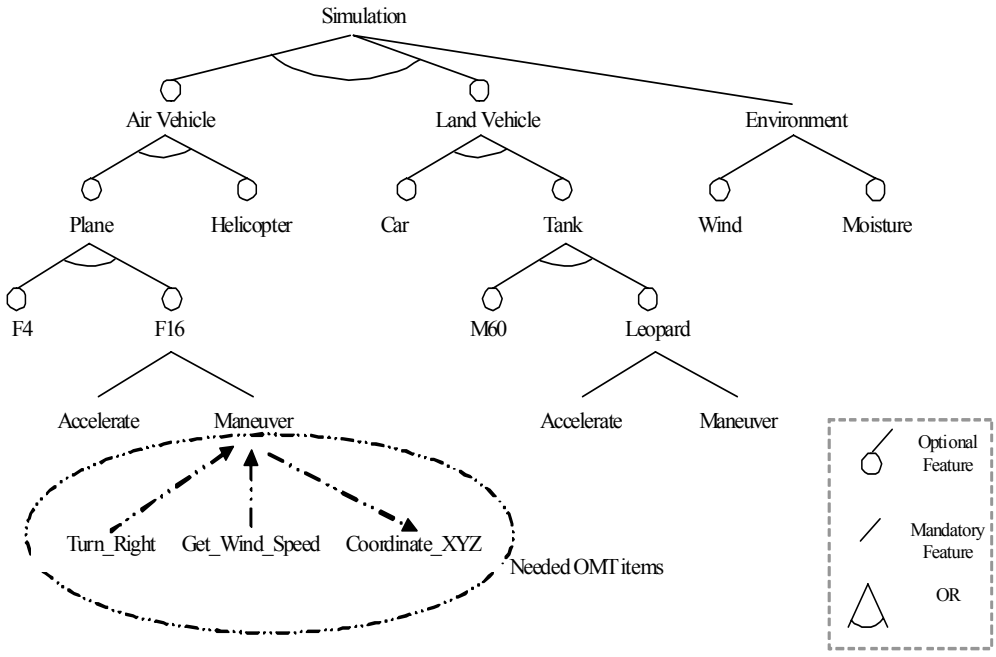

Figure 2.24 An example Feature Model for military vehicles(adapted from [117])

We have identified some problems:

- There is no representation of dependency relationship between OMT items. As noted in [65], interaction problems can occurr and ordering relations should be defined for functional features [95].

- Most of the time a feature is not capable to define requirements as defined in [64, 65].

Therefore, in this dissertation, we are using Feature Models to capture customer understanding, and mapping them to FRs. Even there is no direct mapping between features and functional requirements. Some combinations of features, FRs, and DPs only causes the activation of an FR. FRs are defined by the designer and then combinatorial dependency is set between the FR and features. Components and their methods satisfy the FRs, therefore there is no direct connection from features to components.

2.8. Knowledge-Base

Knowledge-bases include the symbols of the computational model in form of statements about the domain and use them to perform reasoning [42]. With utilizing knowledge-base, applications can make their decisions based on domain-relevant questions [42]. Concepts and relationships among them are represented in semantic networks [42]. For instance, *Professor* and *Course* concepts are connected with *instructorof* relationships and it can be represented as depicted in Figure 2.25.

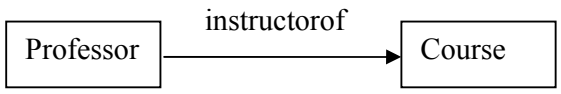

Figure 2.25 Semantic web represention of knowledge

Another form of expressing knowledge is rules [42]. Such as relationship between *Professor* and *Course* concepts are represented in the equation 2.7.

$$\Pr offesor(x?) \wedge instructorOf(x?, y?) \longrightarrow Course(y?) \tag{2.7}$$

Both semantic networks and rules can be represented with logic languages [42]. Such as, rule in equation 2.7 can be represented in logic as depicted in equation 2.8.

$$\begin{gathered} \forall x, y : (instructorOf(x, y) \longrightarrow \Pr ofessor(x) \wedge Course(y) \\ \forall x : \exists y : (Course(x) \longrightarrow \Pr ofessor(y) \wedge instructerOf(y, x) \end{gathered} \tag{2.8}$$

Knowledge is processed by reasoning engines through deriving new statements [42]. Such as, based on "student is a human", "Ahmet is a student" statements, reasoning engine such as (Fast Classification of Terminologies (FACT++) [122]) can derive "Ahmet is a human" statement. Description logic (DL) [5] is one of formalism to represent rules. In abstract notation, we use the letter *A* for atomic concepts, the letter *r* for atomic roles, and the letters *C* and *D* for concept descriptions. Table 2.11 lists some DL constructs used in this dissertation.

Table 2.11 Summary of constructors to form different description logics (adapted from [5])

Name	**Syntax**	**Description**
Atomic Concept	A	Class
Atomic negation	¬A	Complement of class
Concept conjunction	C ⊓ D	Intersection of classes
Concept disjunction	C ⊔ D	Union of classes

Table 2.11 (Cont'd)

Value restriction	$\forall r.C$	All range of values of r in C
Limited existential restriction	$\exists r.C$	some range of values of r in C
Concept equivalence	$C \equiv D$	Equivalence of concepts
Inclusion axiom	$C \subseteq D$	Specialization

A knowledge-base in the basic DL has two parts: the TBox and the ABox. TBox introduces the terminology, i.e., the vocabulary of the domain of discourse, while the ABox contains assertions about named individuals in terms of this vocabulary. TBox axioms can be concept inclusions of the form $C \subseteq D$ or concept equivalences of the form $C \equiv D$ (i.e. $C \subseteq D$ and $D \subseteq C$). The equation 2.8 can be represented as utilizing DL as following:

$$Course \subseteq \exists instructorOf.\Pr ofessor \tag{2.9}$$

An important ontology language is Web Ontology Language (OWL) [78] that provides an expressive ontology model for Semantic Web. It has three subsets with different power of expressiveness - OWL Lite, OWL-DL and OWL Full. In this dissertation, we use OWL-DL as it provides direct support for (classical) negation, disjunction, cardinality restrictions, enumerations, and value restrictions compared to OWL Lite. Protégé is one of the tools to represent OWLs and plugins such as FACT++ [122] reasoning engines can be utilized for reasoning.

2.9. Mature Domain

From the early days on when the module concept was introduced, reuse has been a very important topic because of well-known considerations such as cost and time to develop. It is beneficial to satisfy same functional requirement items using the same set of existing design items. Therefore, in a mature domain, a generic system is

defined that is utilized in the instantiation of specific products. Mature domain concept is also investigated by Kang et al. [65]. In terms of Kang et al., maturity is indicated by the existence of utilization of standards, documented standard terminology, availability of experts, etc. COSE approaches assume that there are some mature domains that include components which are suitable for integration. Table 2.12 lists the assets that should be included in a mature domain for the proposed approach.

Table 2.12 Mature Domain Concepts

Item	**Description**
A Feature Model	Mature Domains must be satisfying the common understanding. All features in a mature domain are represented in a feature diagram. An instance of the domain Feature Model identifies customer needs for a new application. Customer selects the features considering dependencies among features.
A dictionary	Features are read by all stakeholders therefore stakeholders should have the same understanding about them. Also, standard method names are implemented by components.
A design Matrix	A mature domain has one FR-DP design matrix. This matrix is used later to create the applications FR-DP design matrix. Therefore, a new system's design matrix is only a subset of this matrix.
Components	There should be sufficient number of components in mature domain to implement designs.

Table 2.12 (Cont'd)

Design matrices of components	Since components are designed through ADCO as described in chapter 4, there are design matrices defining functionalities and dependencies.
Ontology	• Features • FRs, DPs (methods), PVs (components) • Components and their design matrices • Relationships based on rules among features, FRs, DPs, and components • Object Model Template Classes for HLA based simulations
Collaboration diagrams	Collaboration diagrams can be helpful to define FR-DP dependencies and verify application in terms of functionality.

To be effective, mature domains are allowed to be populated with new components as time progresses. A mature domain expert can add or delete components to a domain. In mature domains, applications can be created as instances of the domain. Different FR subsets of the domain are utilized to satisfy customer needs. Also, mature domains can provide DP alternatives that have similar capabilities to satisfy an FR. Number of alternative DPs increase the flexibility of a mature domain. At the same time it increases the complexity of the design process. This complexity is due to the fact that a selected DP should be consistent with the rest of the mature domain.

CHAPTER 3

ONTOLOGY MODELING

The gap between customer needs and Functional Requirements (FRs) is reduced through utilizing Feature Models, description logic and ontologies which are used to provide formal mechanisms for representing system requirements and component specifications. We assume that a mature domain includes a number of elements such as a Feature Model, FRs, Design Parameters (DPs), and Process Variables (PVs). In our methodology, each of these elements is represented by the corresponding ontology in order to be reused in development processes with the representation and reasoning capabilities of the Description Logic (DL). These ontologies, as a whole, constitute the knowledge-base (i.e. TBox) of the domain.

Mature domain concepts namely features, FRs, DPs, PVs and their dependencies are represented in an ontology that is expanding Wang's method [126]. Each concept may have one or more concept relationships derived from a base concept. Dependencies are represented using the *Linkedtoaconceptname* role. A concept is represented with an equivalence constraint for reasoning:

$$\text{Concept}_i \equiv \exists\ \text{LinkedtoConcept}_i.\text{Concept}_i$$

Relationships among concepts are represented using the subsumption constraint (ex. Concept_i requires Concept_m) as represented below.

$$\text{Concept}_i \subseteq \exists\ \text{LinkedtoConcept}_m.\text{Concept}_m$$

Constraints can be combinations of more than one constraint (union, intersection or complement). We can write the restriction related with $Concept_i$ such as: $Concept_i$ requires $Concept_m$ and complement of $Concept_n$.

$$Concept_i \subseteq \exists\ LinkedtoConcept_m.Concept_m \sqcap \neg\exists LinkedtoConcept_n.Concept_n$$

Definition of any element (feature, FR, DP, or PV) in our mature domain ontology is determined by the base concepts, namely *Feature*, *FunctionalRequirement*, *DesignParameter*, and *ProcessVariable*. In the text, we refer to this ontology as the mature domain core ontology.

Definition 1: (Mature Domain) Given a set of features, $F=\{f1,...,fn\}$, functional requirements, $FR=\{fr_1,\ldots, fr_m\}$, design parameters, $DP=\{ dp_1,\ldots, dp_s\}$, and $PV=\{pv_1,\ldots,pv_t\}$, a mature domain is defined as a terminology including the following basic axioms:

- $f_i \equiv \exists$ Linkedtof$_i$.f$_{i,}$ for $1\leq i \leq n$

  $f_i \subseteq \neg \exists$ Linkedtof$_i$.f$_{i,}$ for $1\leq i \leq n$
- $fr_j \equiv \exists$ Linkedtofr$_j$.fr$_j$, for $1\leq j \leq m$

  $fr_j \subseteq \neg \exists$ Linkedtofr$_j$.fr$_{j,}$, for $1\leq j \leq m$
- $dp_k \equiv \exists$ Linkedtodp$_k$.dp$_k$, for $1\leq k \leq s$

  $dp_k \subseteq \neg \exists$ Linkedtodp$_k$.dp$_k$, for $1\leq k \leq s$
- $pv_x \equiv \exists$ Linkedtopv$_x$.pv$_{x,}$ for $1\leq x \leq t$

  $pv_x \subseteq \neg \exists$ Linkedtopv$_x$.pv$_{x,}$ for $1\leq x \leq t$

In the early stages of design, FRs and DPs do not have complemented subsumption restrictions because applications which are instances of the domain have to have these FRs and DPs. Therefore feature and PV concepts violate the constraints in the core ontology. The core ontology is utilized by applications which are instances of the domain. Our implementing tool allows application designer to select only

features and components. Therefore, applications only reflect selections to the core ontology with deleting the following restrictions:

$$f_i \subseteq \neg \exists \text{ Linkedtof}_i.f_i, \text{ for } 1 \leq i \leq n$$

$$pv_x \subseteq \neg \exists \text{ Linkedto } pv_x.pv_x, \text{ for } 1 \leq x \leq t$$

When a reasoning engine such as FACT++ [122] executes on the ontology, elements which are not selected and all dependent concepts will be in an unusable state.

In the following sections, we introduce ontology modeling for each base concept. There are many-to-many relationships among concepts. For instance, a FR can depend on a combination of features, FRs, DPs, and PVs.

3.1. Modeling Features

Features are an important part of the mature domain as they are the means to allow customers to specify their needs. As stated in Definition 1, each feature is represented as a sub-class of the base *Feature* concept in our mature domain. Although this is necessary, it is not sufficient to define the full semantics of a feature within the ontology. The actual semantics can only be revealed by considering the relationships of a feature with other features as described in Section 2.7. The parent/child relations in a Feature Model, represented as a complete tree have not been incorporated in the ontology presented in this article. Such work has been included in [14] and summarized in section 2.7. In this section, we will explain our Feature Modeling technique and explain it on a conference management system and an aircraft simulation example depicted in Figure 3.1. The representations of relations in the Feature Model such as mandatory and optional are presented below.

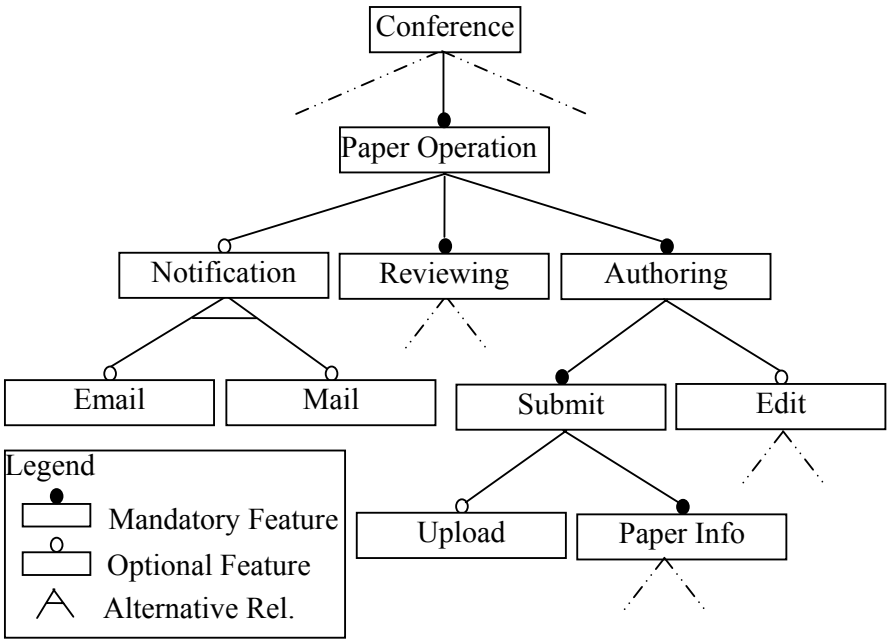

Figure 3.1 Partial Feature Model of the Conference Domain

Proposition 1: Given a parent feature f_p and its mandatory subfeature f_s, a mandatory feature relationship between them can be defined by further restriction such as:

$$f_p \subseteq \exists \text{ Linkedtof}_s.\ f_s$$

For *Submit* feature in our example Feature Model in Figure 3.1, we can specify a mandatory *PaperInfo* subfeature as:

$$\text{Submit} \subseteq \exists \text{ Linkedto PaperInfo.PaperInfo}$$

In a similar fashion, we can define other feature relationships.

Proposition 2: Given a parent feature f_p and its optional subfeature f_s, there is no need to define any restriction.

Proposition 3: If a set of subfeatures $F_s=\{ f_{s1},\ldots, f_{sn}\}$ is related to a parent feature f_p over an *Or* relationship, concept f_p is restricted with the following restriction:

$$f_p \subseteq \exists \text{ Linkedtof}_{s1}.f_{s1} \sqcup \ldots \sqcup \exists \text{ Linkedtof}_{sn}.f_{sn}$$

Proposition 4: *Alternative* relationship is also similar to Or relationship, but this time we can select one and only one subfeature. If a set of disjoint subfeatures $F_s=\{f_{s1},\ldots,f_{sn}\}$ is related to a parent feature f_p over an *Alternative* relationship, parent concept f_p is restricted with the following restriction:

$$f_p \subseteq (\exists \text{ Linkedtof}_{s1}.f_{s1} \sqcap \neg \exists \text{ Linkedtof}_{s2}.f_{s2} \sqcap \ldots \sqcap \neg \exists \text{ Linkedtof}_{sn}.f_{sn}\ )$$
$$\sqcup \ldots \sqcup (\exists \text{ Linkedtof}_{sn}.f_{sn} \sqcap \neg \exists \text{ Linkedtof}_{s1}.f_{s1} \sqcap \ldots \sqcap \neg \exists \text{ Linkedtof}_{sn-1}.f_{sn-1}\ )$$

For instance, *Notification feature* in the example includes a number of notification means represented by features within an *Alternative* relationship. Therefore, we include such a restriction to show this in the ontology:

$$\text{Notification} \subseteq (\exists \text{ LinkedtoEmail.Email} \sqcap \neg \exists \text{ LinkedtoMail.Mail}) \sqcup$$
$$(\neg \exists \text{ LinkedtoEmail.Email} \sqcap \exists \text{ LinkedtoMail.Mail})$$

It should be noted that constraints which include *complementof* is not observable unlikely others. This constraint yields the *Root* as unusable (represented in the following statement) but not the related concept. For instance, notification feature will be consistent when *Email* and *Mail* features are selected. However, root concept which includes all concepts as represented in the following statement will be unusable. To detect which statement makes *Root* concept unusable, debugging is required. Our implemented tool includes a Feature Model preparation and debugging capability.

$$\text{Root} \subseteq \exists \text{ LinkedtoEmail.Email} \sqcap \exists \text{ LinkedtoMail.Mail} \sqcap \exists$$
$$\text{LinkedtoNotification.Notification} \sqcap \ldots$$

3.2. Modelling Functional Requirements

As depicted in Figure 3.2, FRs are represented in a tree structure. In this study, we have identified two properties of the parent child relationship between FRs, corresponding to mandatory and optional specifications. Parent FR can be considered as decomposed if some set of the child FRs are consistent with the ontology. This kind of relationship is similar to mandatory relationship in the Feature Model. Other relationship is similar to optional relationship in the Feature Model meaning that implementation of a set of child FRs do not affect the parent FR. Each FR has to be satisfied by at least one DP. For *FR1.1* in Figure 3.2, we can specify *FR1.1.4* and *FR1.1.5* as mandatory FRs since without them *FR1.1* is not capable to realize *Submit* process.

Other children FRs are not added as restrictions as defined in Proposition 2. FRs are satisfied by DPs. Therefore, restriction for *FR1.1* is defined as following.

$$\text{FR1\_1} \subseteq (\exists\ \text{LinkedtoFR1\_1\_4}.\ \text{FR1\_1\_4}) \sqcap (\exists\ \text{LinkedtoFR1\_1\_5}.\ \text{FR1\_1\_5}) \sqcap (\exists\ \text{LinkedtoDP1\_1}.\ \text{DP1\_1})$$

In some situations, FRs can be satisfied by one among the alternative DPs. Designer has to select one of them. This restriction is similar to restrictions of alternative features and is defined as in the following statement.

$$fr_p \subseteq (\exists \text{Linkedtodp}_{k1}.dp_{k1} \sqcap \neg\ \exists \text{Linkedtodp}_{k2}.dp_{k2} \sqcap \ldots \sqcap \neg\ \exists \text{Linkedtodp}_{kn}.dp_{kn}) \sqcup \ldots \sqcup (\exists \text{Linkedtodp}_{kn}.dp_{kn} \sqcap \neg\ \exists \text{Linkedtodp}_{k1}.dp_{k1} \sqcap \ldots \sqcap \neg\ \exists \text{Linkedtodp}_{kn-1}.dp_{kn-1})$$

FRs can be dependent on features. For instance, *FR1.1.1.1* is dependent on Email feature and corresponding restriction is defined as following:

$$\text{FR1\_1\_1\_1} \subseteq (\exists\ \text{LinkedtoEmail}.\ \text{Email}) \sqcap (\exists\ \text{LinkedtoDP1\_1\_1\_1}.\ \text{DP1\_1\_1\_1})$$

FRs can be dependent on the complements of features, DPs, or FRs.

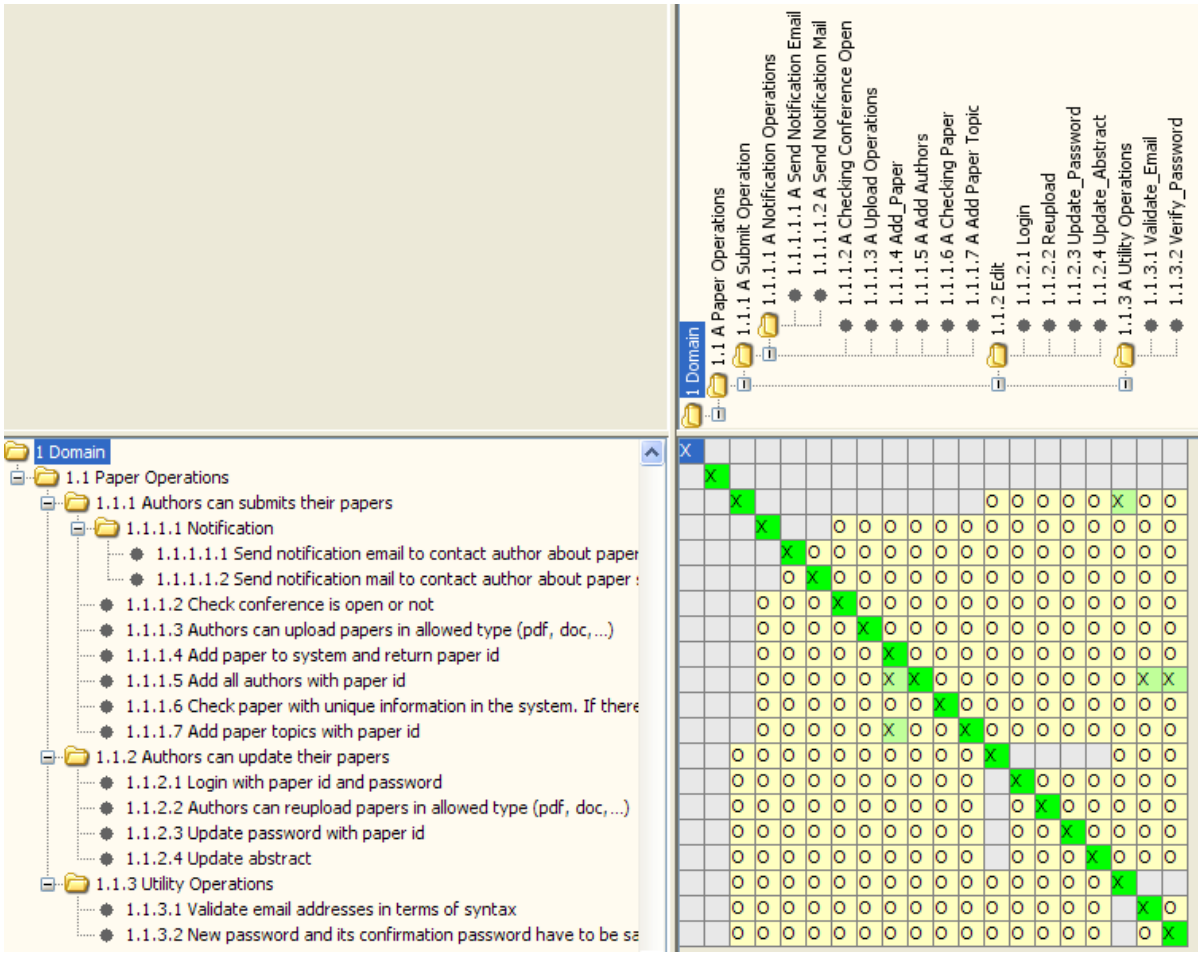

Figure 3.2 Partial FR-DP Design Matrix of the Conference Domain

3.3. Modelling Design Parameters

Design parameters satisfy the FRs and they can be in the form of a process (method) in terms of software terminology. Design parameters can be dependent on other DPs, features, and components. DPs are published by components and DP-component relationship is defined in the DP concepts. For instance, *DP1.1.4* in Figure 3.2 is published by *PV_1 (*such as *PaperOperations* component*)*. Therefore without *PV_1, DP1.1.4* is not a valid DP. This restriction can be represented as follows:

$$DP1\_1\_4 \subseteq (\exists\ LinkedtoPV\_1.\ PV\_1)$$

If more than one component is publishing a DP then this restriction is defined as in the following statement.

$$dp_k \subseteq \exists\ Linkedtopv_0.pv_0 \sqcup \ldots \sqcup \exists\ Linkedtopv_s.pv_s$$

DPs can be dependent on features. For instance different implementations of DPs can have different capabilities with different PVs. This kind of restrictions is presented for dp_k in pv_x as in the following statement:

$$dp_k \subseteq (\exists\ Linkedtopv_x.pv_x \sqcap \exists\ Linkedtof_i.f_i),\ \text{for } 1 \leq i \leq n$$

DPs can be dependent on other DPs and this restriction is represented for dp_i in pv_x as following:

$$dp_i \subseteq (\exists\ Linkedtopv_x.pv_x \sqcap \exists\ Linkedtodp_j.dp_j)\ \text{, for } 1 \leq j \leq n$$

DPs can be dependent on complements of features, DPs, FRs, or PVs.

3.4. Modelling Process Variables

Components are represented as PVs. There can be relationships between PVs and features and other PVs. Some examples of dependencies (e.g. pv_j requires pv_1) are represented as following:

$$pv_j \subseteq (\exists \text{ Linkedtopv}_1.\ pv_1)$$

pv_j can be dependent on features.

$$pv_j \subseteq (\exists \text{ Linkedtof}_m.\ f_m)\text{ , for } 1 \leq m \leq n$$

PVs can be dependent on complemented forms of features or PVs.

CHAPTER 4

PROPOSED APPROACH

4.1. Development Processes

In this study, we propose a systematic component oriented system development framework. We have utilized complementary tools such as Feature Model, Axiomatic Design Theory, COSEML, and Ontology which are explained in Chapter 2. Since axiomatic design's process model does not yet address component level architecture issues, we have outlined a new method to combine Component-Oriented (CO) approach with the Axiomatic Design Theory (ADT) process model, offering design guidance. This approach is based on the Axiomatic Design with Component-Orientation (ADCO) approach [116, 118]. ADT provides some advantages when applied to Component-Orientation:

1. Documentation: Software industries tend to develop systems with limited documentation because of cost and time constraints. However, in ADT, design artifacts (requirements, customer needs, design matrices, etc.) are part of the design - without them design cannot be completed. Therefore, documentation is mandatory in ADT. Additional to standard artifacts of the ADT, we have proposed using collaboration diagrams and Feature Models. Since design matrices, Feature Models, and collaboration diagrams are products of the design process, documentation is produced without extra effort.

2. Component Interface: COSEML based interfaces are represented in design matrices. Dependency information among methods (published or subscribed) is represented in design matrices. Functional requirements represent why a method is defined or implemented. Since design matrices are a product of the design process, component interfaces are enhanced without extra effort.

3. Scientific bases: ADT is an assurance towards "better design". In terms of software, it means less maintenance costs, design failures etc.

4. Measurement quality of design: Some derived complexity formulas are used to measure complexity of system in terms of coupling of system. When coupling is increased, various problems can be occurred such as maintenance, debugging etc. costs.

5. Independent designs: One of the axioms independence axiom advice uncoupled designs (ideally). There are various advantages of modular designs.

    - Customer requirements can be changed in different phases of development. Changes in customer requirements can easily be reflected to the uncoupled designs.

    - We propose to design components based on ADT. Therefore, as addition to interface enrichment, components can be decomposed easily to sub-components if required.

    - Interdependent modules (methods, components, etc.) can be tested separately.

ADCO is based on the mature domain concept. There are two ways of creating mature domains for compatibility with our approach: 1) utilizing ADT and 2) other software development as depicted in Figure 4.1. Although, currently available projects are prepared with different software design and development methods, they may be used to create mature domains. In the following sections, we have assumed that projects are designed and developed with utilizing axiomatic design approach.

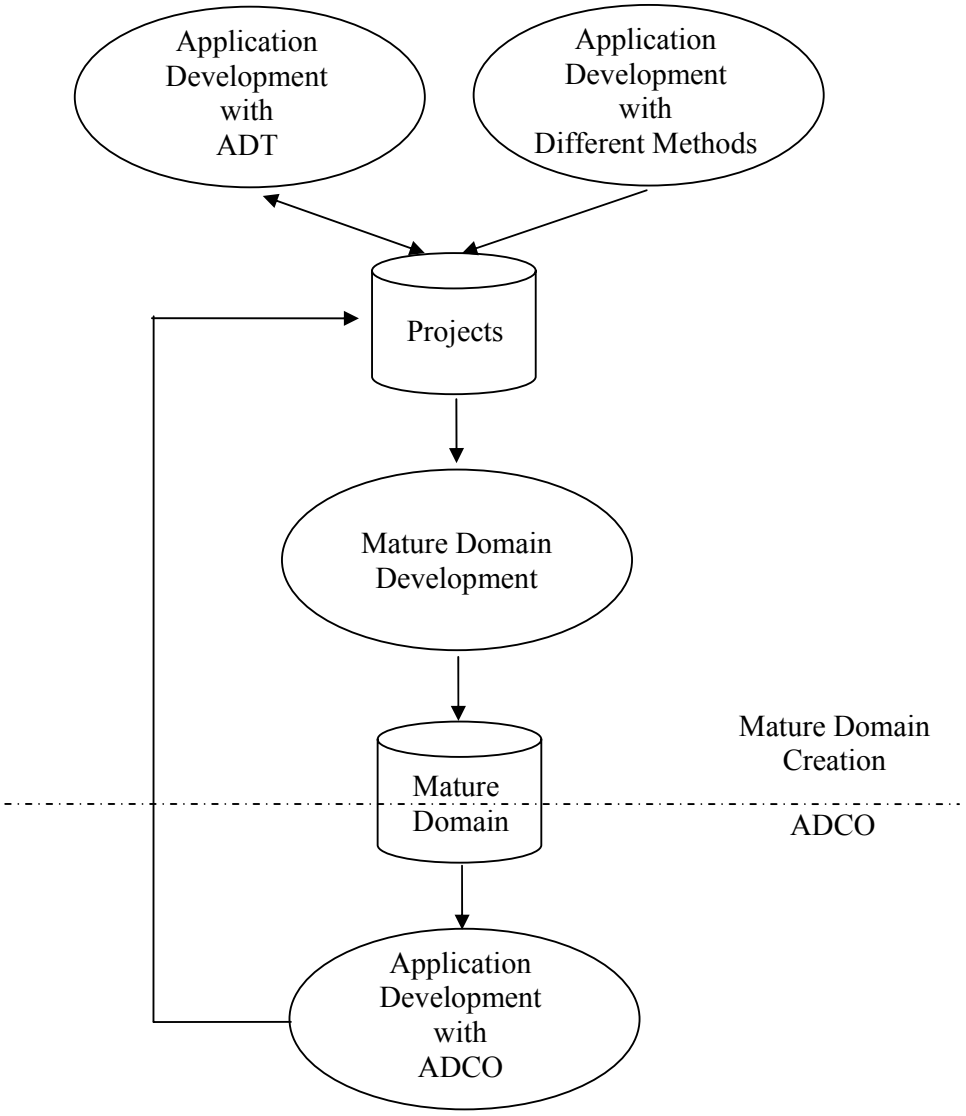

Figure 4.1 Development Processes

4.1.1. Mature Domain Creation

Component-Oriented Software Engineering (COSE) approaches assume that there are mature domains that include components that are suitable for integration [37]. This is also a fundamental assumption in our approach. When number of projects in a domain is increased, mature domains can be created. One way of create mature domains is utilizing existing projects developed based on the axiomatic design concepts and similar projects. Reusing of the similar projects increases the chance of creation mature domains since mature domains are formed from similar projects which share lots of commonality especially in the design matrices.

4.1.1.1. Application Development

We expanded Do and Suh's specific process model called the *V-Model* [34] as depicted in Figure 4.2, which serves as an *axiomatic* methodology for Object-Oriented (OO) software development as defined section 2.1.2. The parts of our approach that depart from the original OO version are shown in white boxes. The development process looks very similar to that of the original OO version. However, there are key differences due to the COSE approach that assumes the existence of components in a domain. A general an *11-step* method is adapted from [118] for the proposed process as listed in Table 4.1.

The process starts with identification of customer needs. The developer then identifies the domains which include similar projects based on the identified customer needs. Then a top-down design utilizing projects' design matrices in accordance to AD principles is applied. Therefore, software modules are effectively identified in a top-down fashion. Modules represent the services which are satisfied by components. If there are projects utilized during decomposition, their components may help to satisfy required services. If components are still required, first their design matrices are created based on the application design matrix. Application design matrix includes the required services and other dependent services. Then components are developed and added to the repository. Components

are integrated and designer solves integration problems. Application development with ADT process [116] is outlined in Table 4.1.

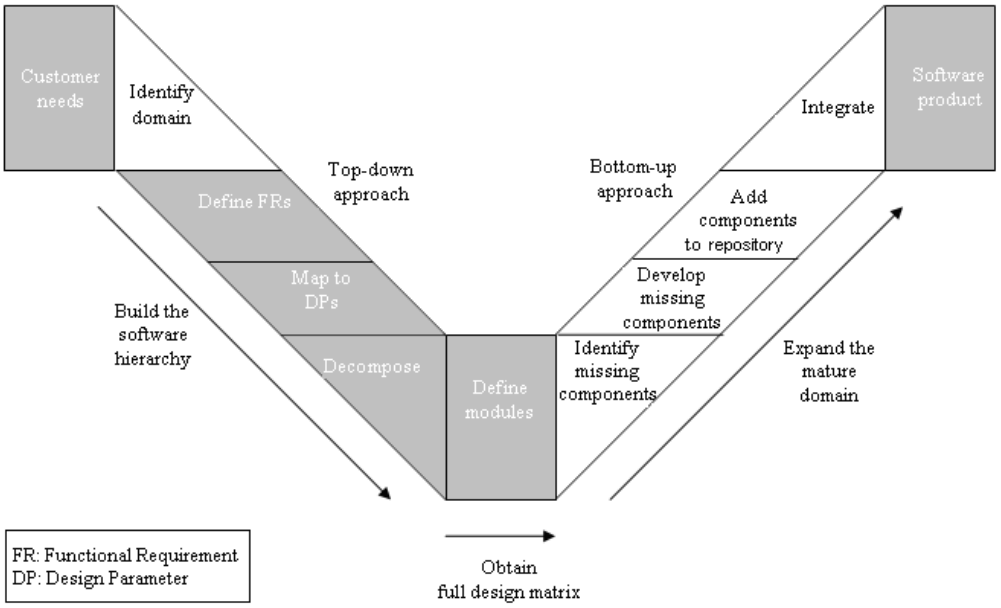

Figure 4.2 Axiomatic design process for CO software system: white boxes represent additions to V-Model (adapted from [116])

Table 4.1 Application development process without mature domain (adapted from [116] and [114])

Step	**Description**
Step 1	Customer Needs: The first step in designing a software application is to determine the customer needs (CNs) or attributes in the customer domain that the software systems must satisfy. One way to solve communication problem between customers and designers is utilizing Feature Models.

Table 4.1 (Cont'd)

Step 2	Identify Domain: The next step is to find similar projects related with the customer needs. As defined by Suh [101], if an Functional Requirement (FR) and its decomposition is already included in a project, then it should be reused. More than one project can be utilized, in an application. As we define later, these projects will be composed to create a mature domain.
Step 3	Define Functional Requirements (FRs): FRs are defined by the developer to satisfy the customer needs. Lower-level FRs guide the developer in selecting a specific attribute or a method. Design matrices corresponding to mature domain components help in determining lower-level FRs. It must be kept in mind that FRs are defined without considering Design Parameters (DPs).
Step 4	Define Design Parameters (DPs): FRs are mapped to DPs. We are using COSEML notations to define DPs. Therefore a DP can be a package name, abstractions, component, interface, method, property, or event name. Abstract representations are preferred especially for higher-level DPs. If function or data abstraction is used, then logical collaboration diagrams can be prepared [114]. Since projects design matrices and their components are shared in various projects, standard name pools (such as, OMT tables in HLA) are created. All DPs do not have to correspond to existing component interface items but all physical DPs must be extracted from the standard name pool or they should be added to the pool if required. Abstract DPs, however, can be in COSEML's abstract representation. It should be noted that the independence axiom must be applied to the design.
Step 5	Decomposition: Decomposition is continued until all FRs are mapped to physical DPs (component, method, property, or event). Therefore steps 2-5 must be conducted recursively.

Table 4.1 (Cont'd)

Step 6	Define Modules: Leaf-level DPs are specified as system modules that can be components, methods, properties, or events. These modules define what is required and should be satisfied by components.
Step 7	Identify Missing Components: Since DPs are chosen from the name pool, identified DPs in the previous step can be used to reach existing components. When all components are identified, missing DPs required by other components or defined in the application matrix but not satisfied by identified components are ascertained. More than one component can be used to solve the application problem. We are proposing two methods to select components among alternatives: • The Information Axiom is applied to pick correct components in terms of their information content and also for inter-component congruity detection as explained in section 2.1.1.4.2. • Communicating Sequential Processes (CSP) [48] and Failures-Divergence-Refinement (FDR2) [40] tool utilizing our method explained in section 2.6 can be used. This method is used where number of components is huge and relationships are complex.
Step 8	Develop Missing Components: The FRs of the application and the related components provide the development reason for the required DPs. At this point, the design phase is already completed for the components because the FRs and DPs and their dependencies in the design matrix and collaboration diagrams are known. Components are implemented depending on the design matrix and optionally collaboration diagrams.
Step 9	Add Components to Repository: For the purpose of reuse, all newly developed components are added to the component repository.

Table 4.1 (Cont'd)

Step 10	Integration: Components are integrated during the execution. Mismatch problems [62] and composition anomalies defined in [28, 47, 108, 109] may need to be solved.
Step 11	Software Product: Execute the application.

4.1.1.2. Mature Domain Development

When number of related projects increase in a field, a mature domain is created by a domain expert. Feature Model is one of the fundamental tools in domain creation. Feature Model is used to define customer needs. These features are mapped to functional requirements considering design parameters. Also these features are used to define capabilities of components. All mappings are realized through our mapping approach based on ontology among the features and ADT domains corresponding to requirements, design, and implementation domains as defined in Chapter 3. We have identified five steps to create a mature domain and listed in Table 4.2. At the end of this process, a mature domain with a domain design matrix and an ontology file that includes design constraints are created.

Table 4.2 Mature domain development process

Step	**Description**
Step 1	Domain Identification: Mature domain concept is required where similar projects are created again and again with minor differences. Therefore, selection of similar projects is very important for identification of a domain.
Step 2	Domain Analysis: Domain expert prepares a reference Feature Model for a domain. This Feature Model will be used to define customer needs and

Table 4.2 (Cont'd)

	to create rules on FRs, DPs and PVs. Customers domain view will help in the definition of relations among the features, FRs, DPs, and PVs should be located in the Feature Model. In other words, if a feature is not utilized in a rule, it should not be defined. Otherwise, lots of features are located in Feature Models and they increase the complexity of Feature Models. Also, unused features can confuse customer since features defines the expectation of customers. Feature Model is created by investigating available domain projects.
Step 3	Functional Requirement and Design Parameter specification: Intersection of the domain projects' design matrices creates domain (reference) design matrix.
Step 4	Design is satisfied by project's components. However still there can be request to design and develop components as defined in step 8 in Table 4.1. Some components can be created newly, some components can be composed, some of them decomposed to create new components, and some of them modified.
Step 5	Create ontology: Relationships among features, FRs, DPs and components are defined and added to ontology as explained in Chapter 3. • Set Rules for features: Relationships among features are added to the ontology. • Set Rules for FRs: To activate an FR, dependent FRs, DPs, and features are specified and added to the ontology. In this step, new features can be added or unused features can be deleted. • Set Rules for DPs: DPs' publishing and subscribing relationship with PVs and feature relationships are added to the ontology. We assume that all DPs are published by at least one PV.

Table 4.2 (Cont'd)

	• Set Rules for DPs: DPs' publishing and subscribing relationship with PVs (components) and feature relationships are added to the ontology. We assume that all DPs are published by at least one PV. • Set Rules for PVs: Relationships among PVs (such as require and mutually exclusive), relationships among features and PVs are added to the ontology as defined in Chapter 3

4.1.2. ADCO Process

In this section, we are proposing the application development process in mature domains. We have identified seven steps to create applications in mature domain and listed in Table 4.3. Developer first seeks to find a mature domain (a collection of interrelated components and their AD artifacts). Application designers benefit from features selected by customers to specify functional requirements for specific systems originating from the mature domain. All constraints among features, FRs, DPs, and PVs are used to define consistency conditions of FRs which are set during mature domain creation process.

Table 4.3 Application development process with mature domain

Step	**Description**
Step 1	Identify mature domain: The most related domain is selected. Since domains can be flexible, if expansions are required, they have to be consistent with available features, FRs, DPs, and PVs in the mature domain and should be conducted by domain experts.
Step 2	Identify customer needs: Customer needs are identified through mature

Table 4.3 (Cont'd)

	domain Feature Model. Therefore, application Feature Model is a sub-Feature Model of the mature domain. Customers select or unselect the features to specify their needs. A Feature Model guides the customer about which features can be selectable.
Step 3	Select Components: Application designer can decide to extract some components from the solution space. After that, a reasoning engine is executed on the ontology.
Step 4	Design the system: Application design matrix is created from the domain design matrix. Since reasoning over the ontology reveals usable and unusable concepts (features, FRs, DPs, and PVs), designer can be warned about unusable FRs. If root FR concept is consistent, there is an application including at least one component. If inconsistency starts with the root FR, in this situation no application will be implemented, because selected features and components cause inconsistency. Designer can debug the causes of the inconsistency using the ontology.
Step 5	Search components: Consistent PVs (components) in the ontology represent the usable components. Alternative components can be available in solution space. Designer selects the components among the alternatives utilizing the information axiom (section 2.1.1.4.2) and CSP (section 2.6).
Step 6	During the integration, mismatch problems [62, 118] and composition anomalies defined in [28, 47, 108, 109] may need to be resolved (e.g. by type casting, synchronization, etc.).
Step 7	Execute system: If unexpected results occur in terms of customer expectation, design is modified. It should be noted that modifications will be local; that is an advantage of this approach if design is uncoupled.

4.2. Guidance

We have identified that design guidance can be provided where mature domains and various parameters (constraints) in the mature domains are available. Our aim is helping designer to consider all parameters (customer needs, functional requirements, design parameters, components etc.) during his decisions. To handle this requirement, we are proposing a guidance mechanism that considers all ingredients of design artifacts and guides application designer in the mature domains. Designer is warned about functional requirements which should be implemented utilizing related customer needs and the environment which includes available components. We have developed an ADCO tool as depicted in Figure 4.3 to implement our framework. The ADCO tool has a capability to create designs for mature domains, components, and applications based on the mature domain. There are six views in the ADCO tool:

1. FR-DP design matrix
2. Component lists: component designer/domain expert can decide intuitively which components are not congruent to execute together because of conflicts such as performance, price, security, etc. with a specific component.
3. COSEML harmonized with the design matrix view represents components in structural view.
4. Feature Model: Feature Model view is used to define customer needs.
5. Rule List: Constraints are created and represented as rules.
6. Information: Represents the mappings with ontology concepts and design items (features, FRs, DPs, and PVs) and some debugging information.

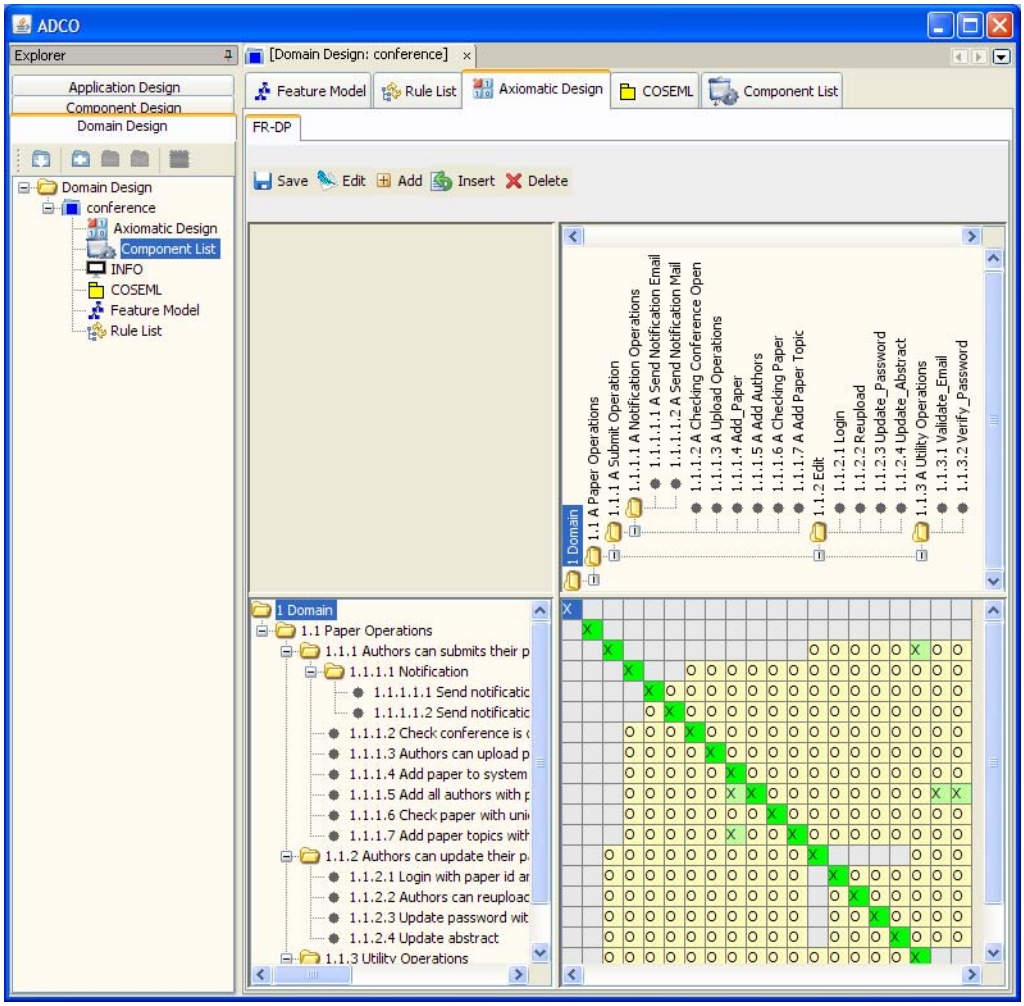

Figure 4.3 Domain view of ADCO Tool

These views become different depending on where they are used: COSEML view represents all components used in *Application Design* but represents a component related with its design matrix in *Component Design* as depicted in Figure 2.13.

Although our framework is more general and applicable to other domains, we applied it to the Conference Management System domain and the HLA based Aircraft Simulations domain as detailed in the following sections.

4.2.1. Conference Management System

We will represent our guidance on a Conference Management System example. This example is adapted from Open Conference [131] which is a conference management system based on the PHP technology. Some functionalities of this PHP based system is converted to a component based system. To represent guidance we concentrate on the *submit* functionality and we also identified nine core components (web services) as depicted in Figure 4.4 and their design matrices are depicted in Appendix A. We have also represented function call order of Submit method with BPMN representation as depicted in Figure A.8 which can be used to detect dependencies similar to collaboration diagrams. Some of these components can also be used to satisfy other functionalities in the mature domain such as editing etc. The FR-DP design matrix belonging to the conference management system domain is shown in Figure 4.5.

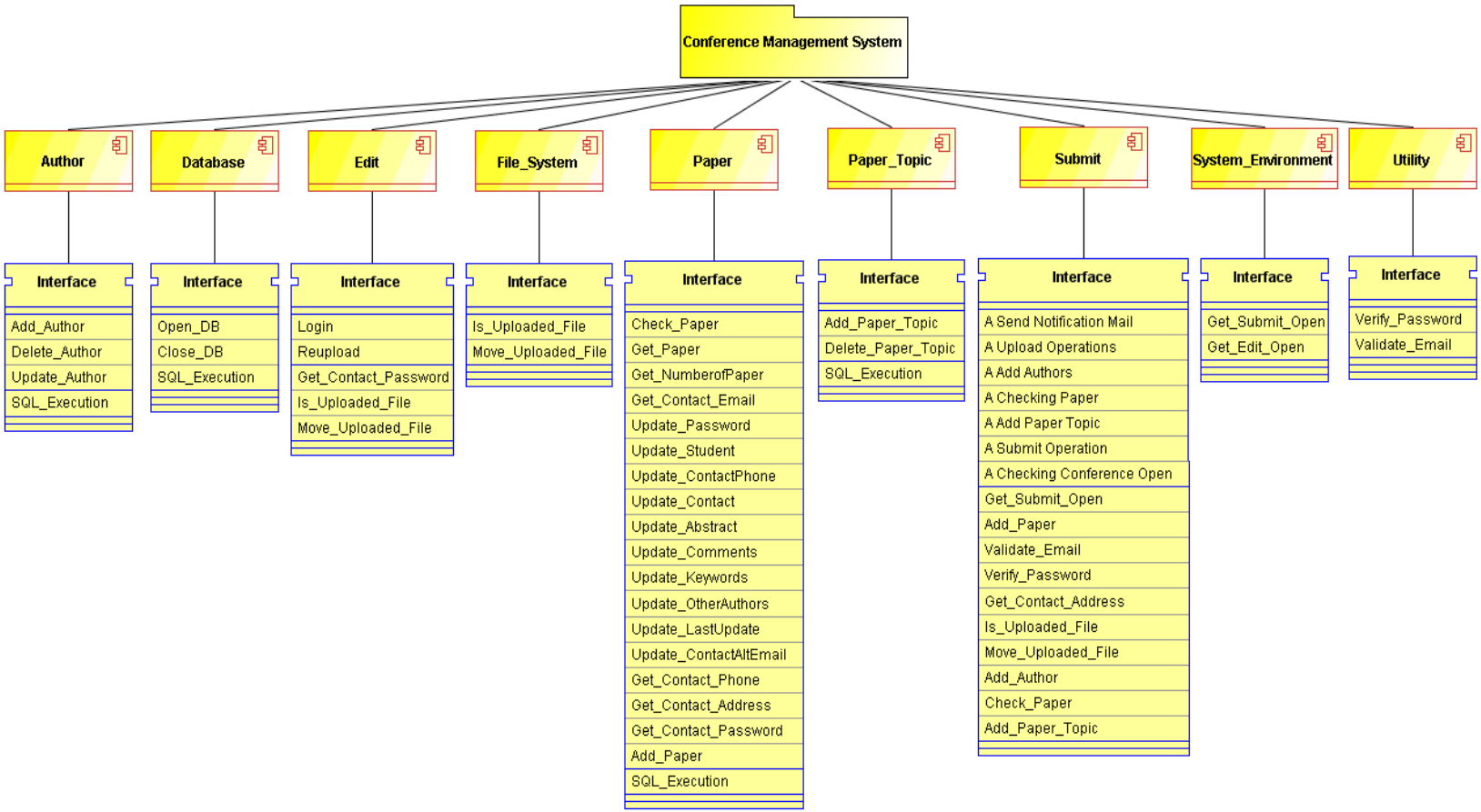

Figure 4.4 Conference Management Components

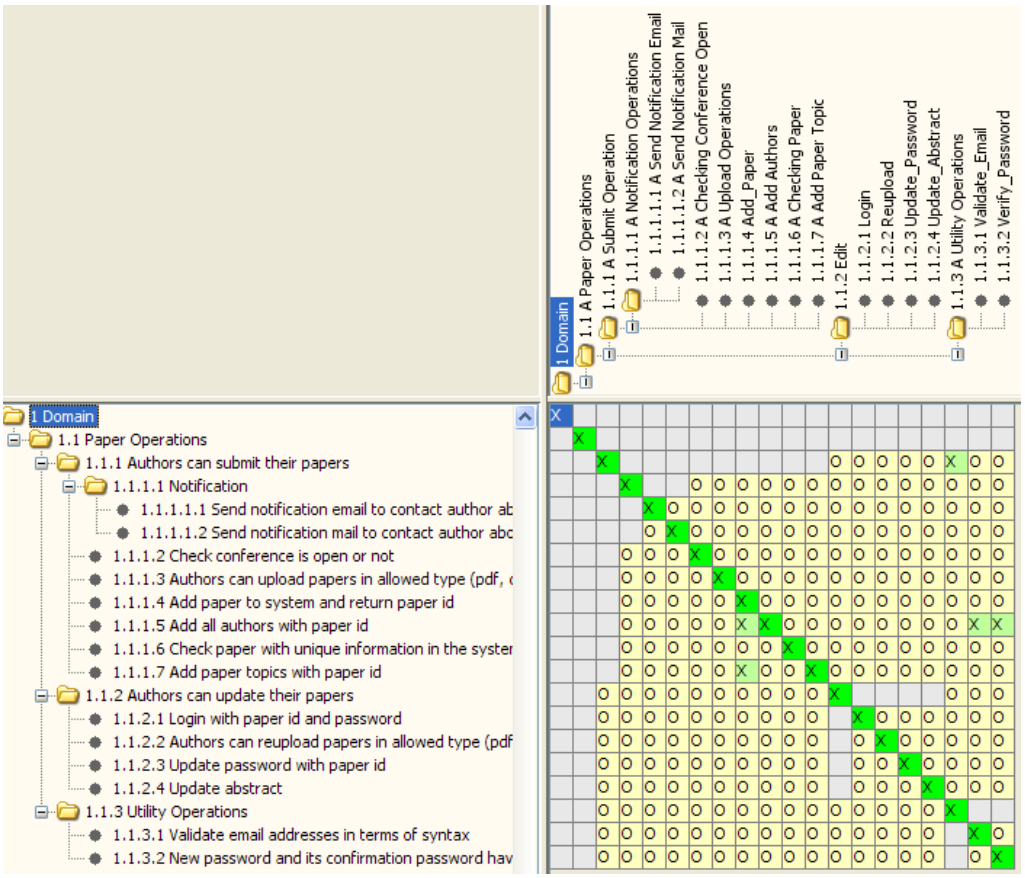

Figure 4.5 Partial FR-DP design matrix

ADCO tool automatically produces some constraints as explained in Chapter 3, such as dependencies among FRs and DPs. We have defined some rules in addition to the automatically generated rules for the Conference Management System as listed in Table 4.4. Normally concepts are defined with unique identifiers in the ontology; for simplicity we used abbreviations such as the names for FRs start with "FR", DPs start with "DP", features start with "F" and components start with "PV" in the Tables 4.4 and 4.5. Some rules are defined for FR-DP design matrix represented in Figure 4.5 by the designer and represented in Table 4.4. For instance, "FR1.1.1: Authors can submit their papers" can be consistent, if "FR1.1.1.4: Add paper to system and return paper id" and "FR1.1.1.5: Add all authors with paper id"

are consistent. This constraint defines *FR1.1.1.4* and *FR1.1.1.5* as mandatory FR for *FR1.1.1.* Another example, "FR.1.1.1.1.1: Send notification email to contact author about submission" can be consistent if *Email, Notify_Author*, and *Alternate_Email* features are selected as represented in Figure 4.6.

Table 4.4 Constraints for Conference Management System

No	**Constraints**
1	FR1⊆ ∃ LinkedtoFR1_1. FR1_1
2	FR1_1⊆ ∃ LinkedtoFR1_1_1. FR1_1_1
3	FR1_1_1⊆ ∃LinkedtoFSubmit.FSubmit
4	FR1_1_1⊆ (∃ LinkedtoFR1_1_1_4. FR1_1_1_4 ⊓ ∃ LinkedtoFR1_1_1_5. FR1_1_1_5)
5	FR1_1_2⊆ ∃LinkedtoFEdit.FEdit
6	FR1_1_1_1⊆ ∃ LinkedtoFR1_1_1_1_1. FR1_1_1_1_1 ⊔ ∃ LinkedtoFR1_1_1_1_2. FR1_1_1_1_2
7	FR1_1_1_3⊆ ∃LinkedtoFUpload.FUpload
8	FR1_1_1_5⊆ ∃LinkedtoFName.FName
9	FR1_1_2_2⊆ ∃LinkedtoFReupload.FReupload
10	FR1_1_1_1_1⊆∃ LinkedtoFEmail.FEmail ⊓ LinkedtoFNotify_Author. FNotify_Author ⊓ LinkedtoFAlternate_Email
11	FR1_1_1_1_2⊆∃ LinkedtoFMail.FMail ⊓ LinkedtoFNotify_Author. FNotify_Author ⊓ LinkedtoFAddress

Based on the automatically generated constraints and constraints defined by domain experts, reasoning engine provides information about inconsistencies. We are interested in figuring out which FRs should be implemented. If *FR1* (root FR) is specified as an unusable concept then the application designer can decide that no application can be implemented in this circumstance. Which constraints cause this circumstance can be found through debugging.

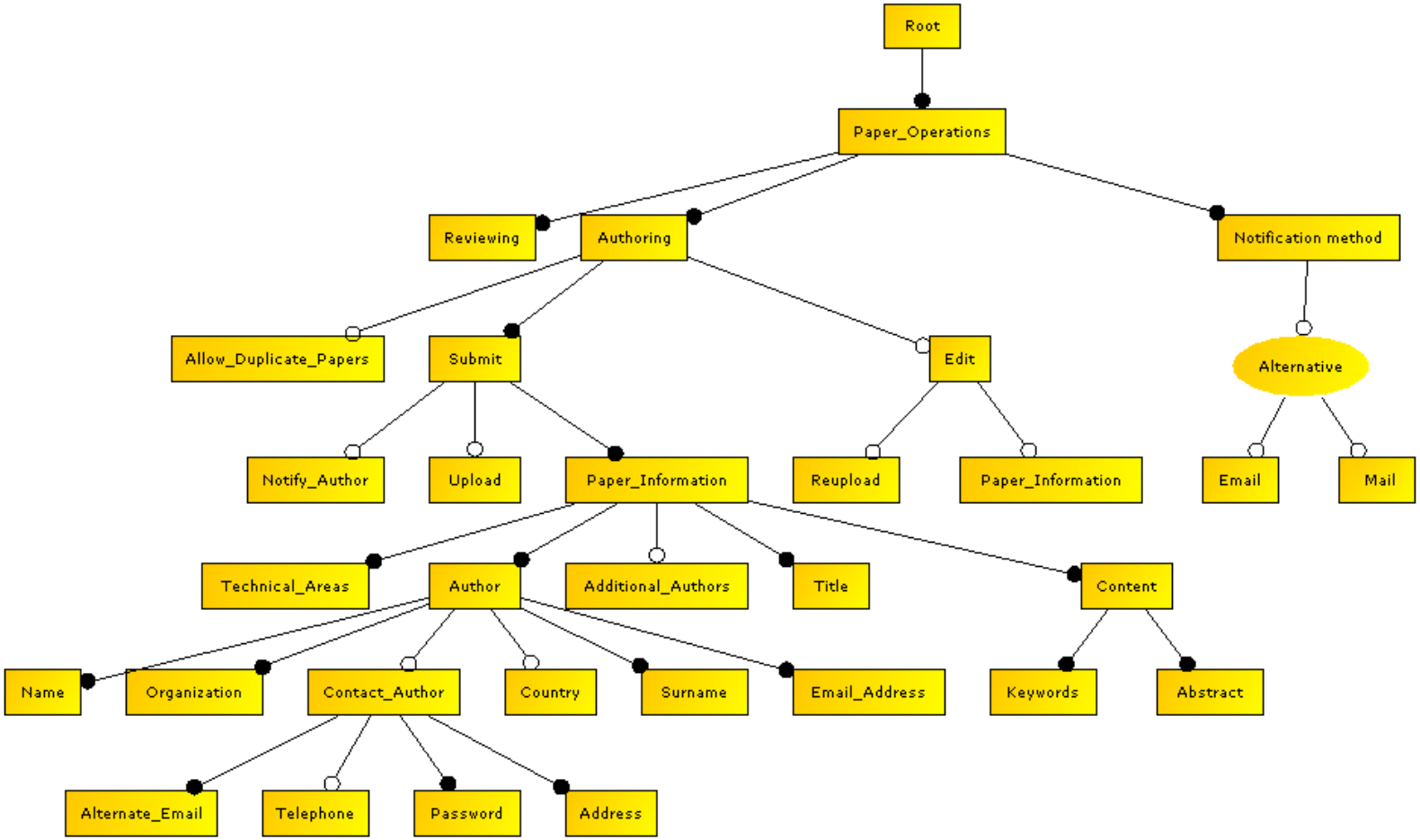

Figure 4.6 Partial Feature Model of the conference management system

In Table 4.5, we have represented all constraints related with *FR1_1_1_5*. First four rules are the same as those in Table 4.4. The following constraints are automatically generated from the design matrices of domain and design matrices of the *Author* and *Database* components. If a dependency chain is broken somewhere the reasoning engine will detect the inconsistency and ADCO tool will warn the application designer. For instance if there is no *Author* component in the environment, concepts will be unusable; *DPAdd_Author* concept because of constraint six, *FR1_1_1_5* concept because of constraint five, *FR1_1_1* concept because of constraint three, FR1_1 concept because of constraint two, and *FR1* concept because of constraint one will be unsatisfied. ADCO tool represents all these inconsistencies in the design matrix as depicted in Figure 4.7. When the *Mail* feature is not selected from the Feature Model, *FR1_1_1_1_2* will be unusable because of violation constraint eleven in Table 4.4 as represented in Figure 4.7.

Table 4.5 Constraints for *FR1_1_1_5*

No	**Constrains**
1	FR1⊆ ∃ LinkedtoFR1_1. FR1_1
2	FR1_1⊆ ∃ LinkedtoFR1_1_1. FR1_1_1
3	FR1_1_1⊆ (∃ LinkedtoFR1_1_1_4. FR1_1_1_4 ⊓ ∃ LinkedtoFR1_1_1_5. FR1_1_1_5)
4	FR1_1_1_5⊆ ∃LinkedtoFName.FName
5	FR1_1_1_5⊆ ∃LinkedtoDPAdd_Author. DPAdd_Author
6	DPAdd_Author⊆ ∃LinkedtoDPSQL_Execution. DPSQL_Execution ⊓ ∃LinkedtoPVAuthor. PVAuthor
7	DPSQL_Execution ⊆ ∃LinkedtoPVDatabase. PVDatabase

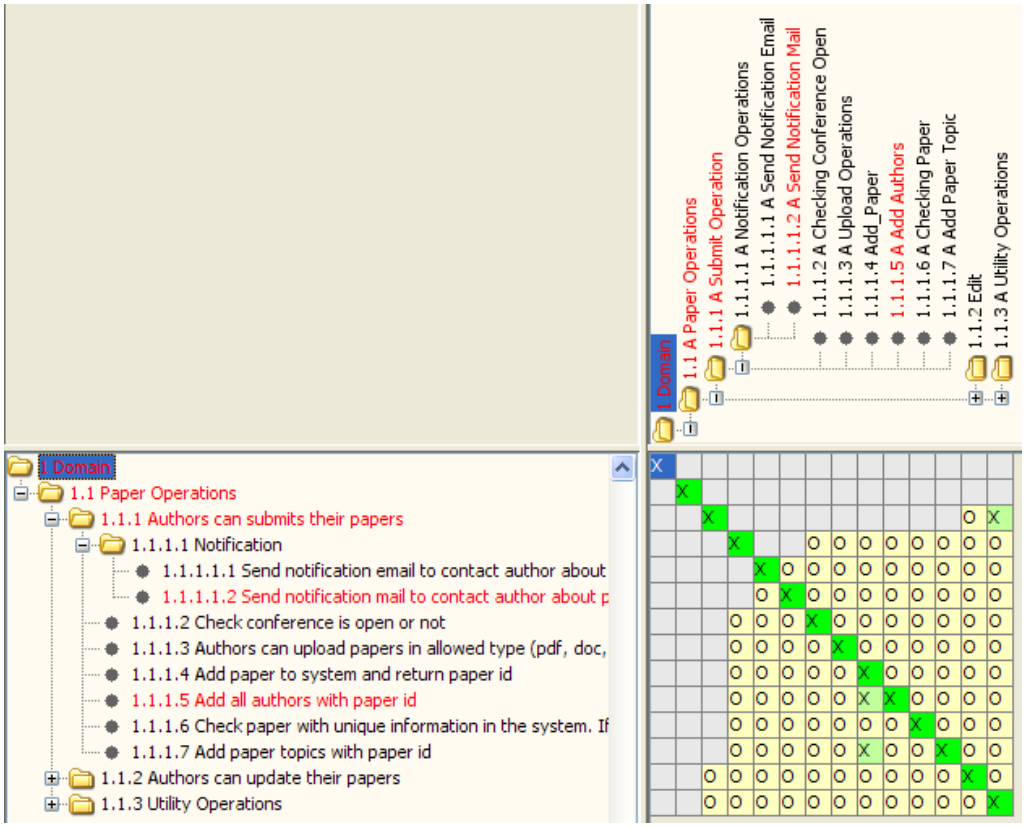

Figure 4.7 FR-DP design matrix representing inconsistencies

4.2.2. Aircraft Simulations

We have adapted ADT to FEDEP [118]. The steps of this process are lısted in Table 4.6. This approach is similar to application development process without mature domain as presented in Table 4.1.

Table 4.6 Adapted FEDEP process with ADT (adapted from [118])

Step	**Description**
Step 1	Define federation objectives: Costumer requirements and features of the expected software systems are determined.

Table 4.6 (Cont'd)

Step 2	Perform conceptual analysis: The next step is to find mature domains related to the customer needs. Customer needs to point to a general idea about which domain (s) can include the components.
Step 3	Design federation: Problem is decomposed to the parts utilizing ADT. Available federates are identified utilizing a design matrix for the problem and design matrices of available federates. Required actions are listed below: ▪ Define Functional Requirements (FRs) ▪ Define Design Parameters (DPs) ▪ Define dependencies among FRs and DPs ▪ Check axioms ▪ Identify missing components
Step 4	Develop federation: Missing federate development and/or available federate modifications are realized utilizing design matrices in this step.
Step 5	Plan, integrate, and test federation.
Step 6	Execute federation and prepare outputs.
Step 7	Analyze data and evaluate results.

ADCO process is applied to simulations based on High Level Architecture [118] and explained in this case study. Since HLA can be used to develop agents for games [74], and military simulations are so popular, we developed a war-vehicle domain. There are two kinds of components: software components and federates formed by the software components as represented in Figure 4.8. We have applied

ADCO to HLA based simulations in two levels. As it can be seen in Figure 4.8, software components create federates and federates forms federation. Federates communicate with federates through RTI and software components communicate in software components. In this section, we will represent three aircraft (F16, F18, Su25) federates and software components formed these federates. One of federates F18 with center view is represented in Figure 4.9. In this simulation, only OMT attributes as listed in Table 4.7 and saved as FED file are shared among federates. There is no OMT interaction classes in this case study.

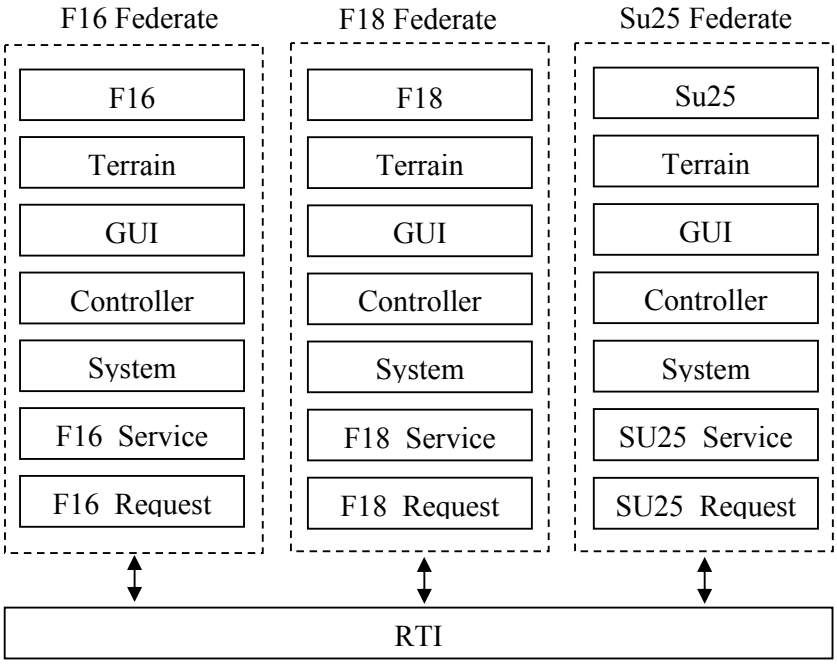

Figure 4.8 All components in Aircraft simulation

In the simulation environment, there are four software components common in federates as represented in Figure 4.8 and Figure 4.10. In our example, all federates can be implemented thorough these components because of their similar capabilities. Recalling the ADCO processes, mature domains are created where some components and functionalities become common. *Controller* component is used to control federates. Its COSEML representation is depicted in Figure 4.10 and

FR-DP design matrix is depicted in Figure 4.11. In the *Terrain* component, aircrafts and effects are represented as depicted in Figure 4.10 and design matrix of *Terrain* component is depicted in Figure 4.12. In the *GUI* component, actual values of OMT attributes are represented as depicted in Figure 4.9 and design matrix is depicted in Figure 4.13. *System* component produce events which are obtained from operating system such as keyboard events as shown in Figure 4.10 and Figure 4.14.

Table 4.7 OMT classes of simulation (adapted from [118])

Class	Attribute	DataType	Description
Vehicles (PublishSubscribe)	Longitude	Integer	Actual longitude within terrain
	Altitude	Integer	Actual altitude within terrain
	Latitude	Integer	Actual latitude within terrain
	Roll_Angle	Integer	Roll angle of vehicle to represent vehicle on terrain
	Pitch_Angle	Integer	Pitch angle of vehicle to represent vehicle on terrain
	Yaw_Angle	Integer	Yaw angle of vehicle to represent vehicle on terrain
	Vehicle_Type	String	Vehicle type: F16, F18, etc.
	Crashed	Boolean	Value is set to “true” if vehicle has crashed
	Height	Integer	Height of vehicle to represent vehicle on terrain and calculating crashes
	Width	Integer	Width of vehicle vehicle to represent vehicle on terrain and calculating crashes
	Speed	Integer	Actual speed value of vehicle

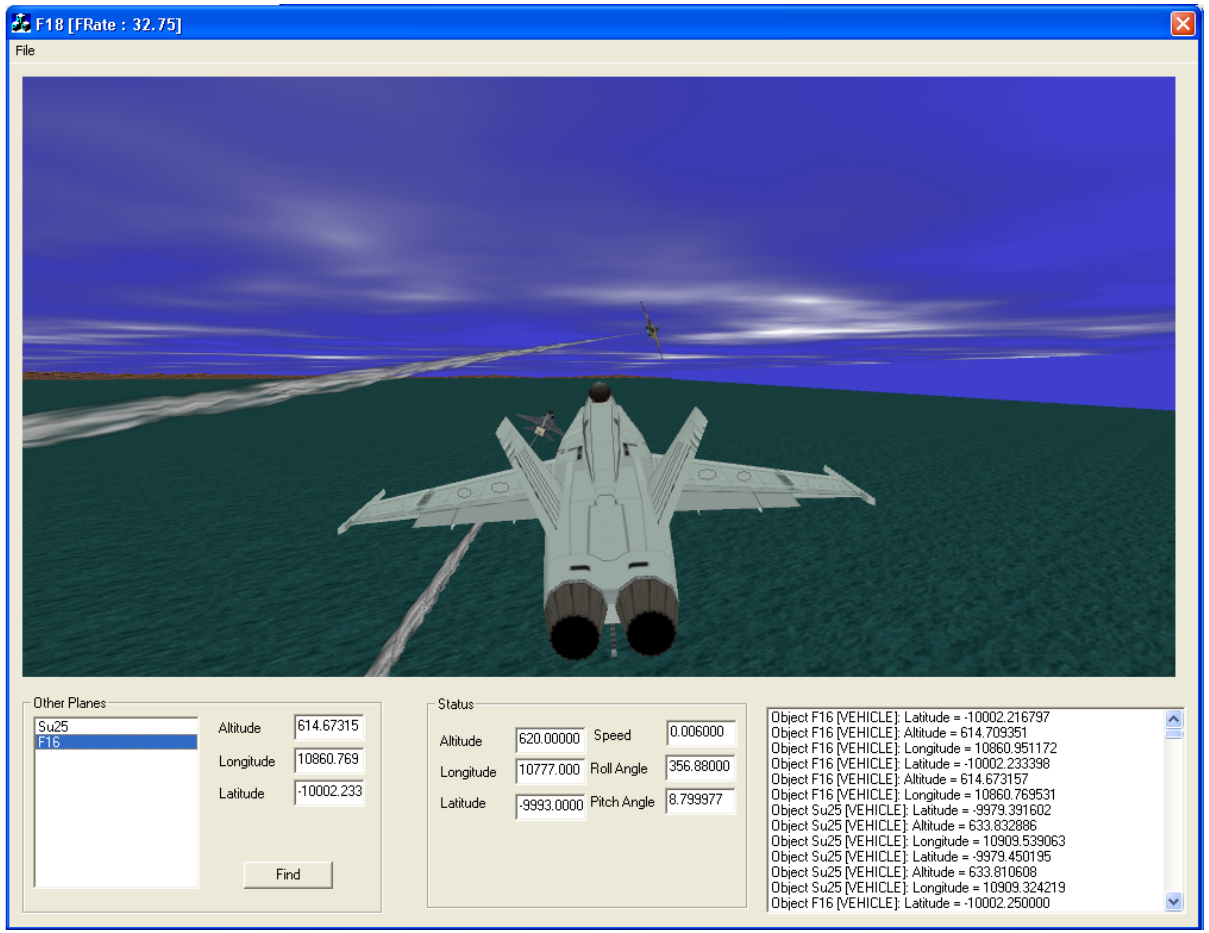

Figure 4.9 Screenshot from F18 federate includes F16, F18, and Su25 federates

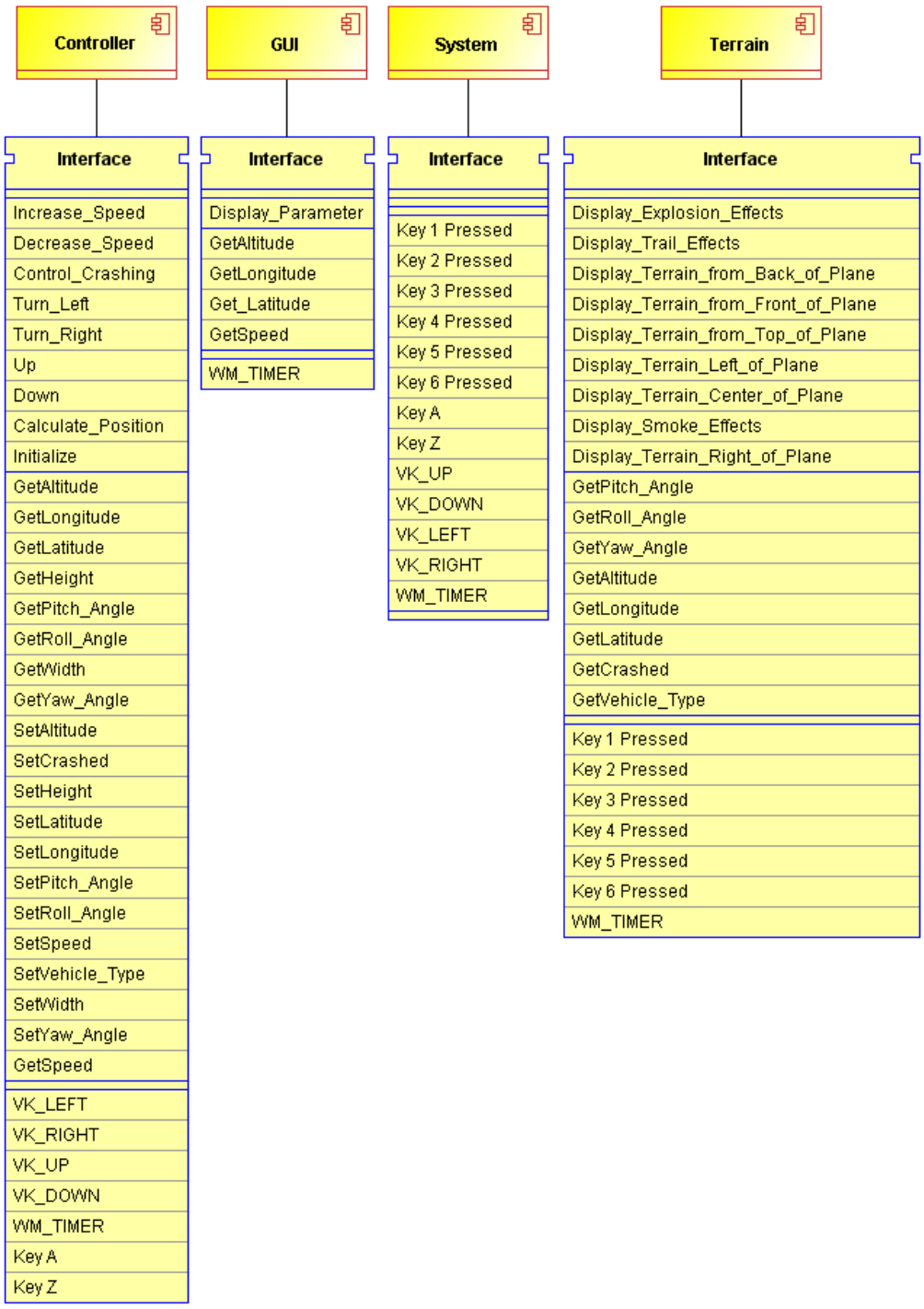

Figure 4.10 Software components

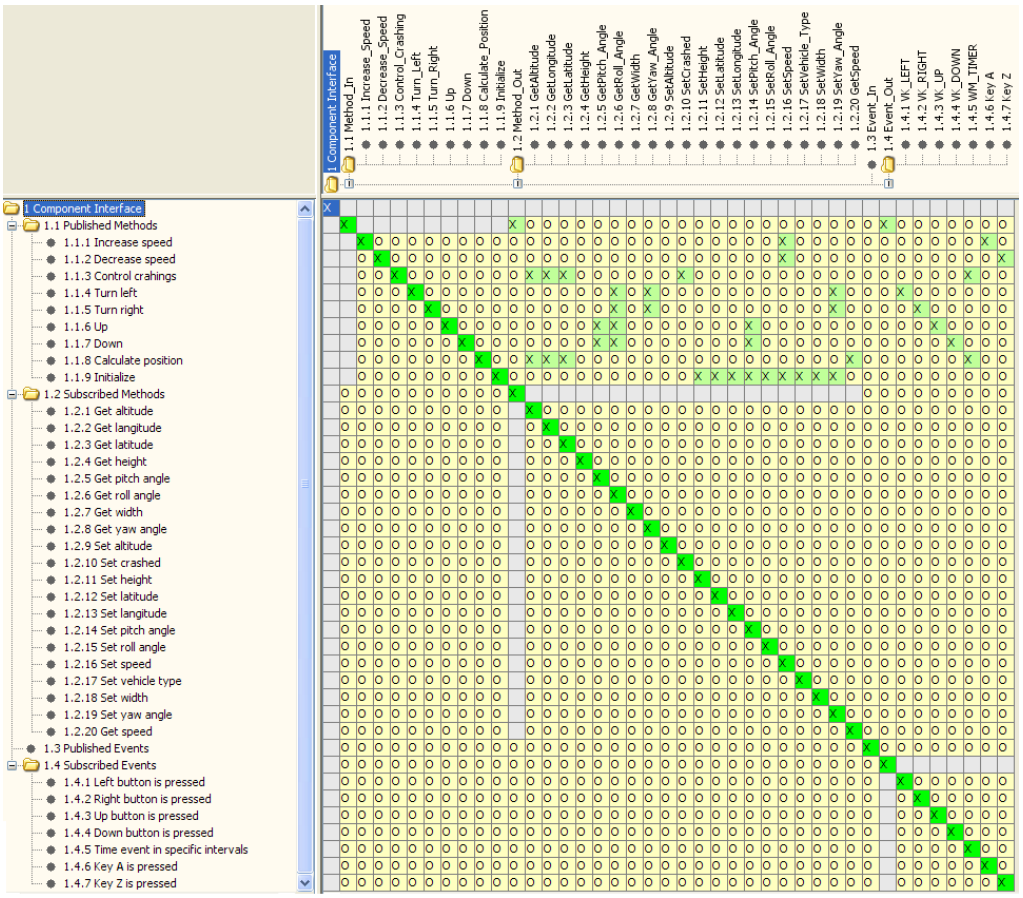

Figure 4.11 FR-DP design matrix of Control component

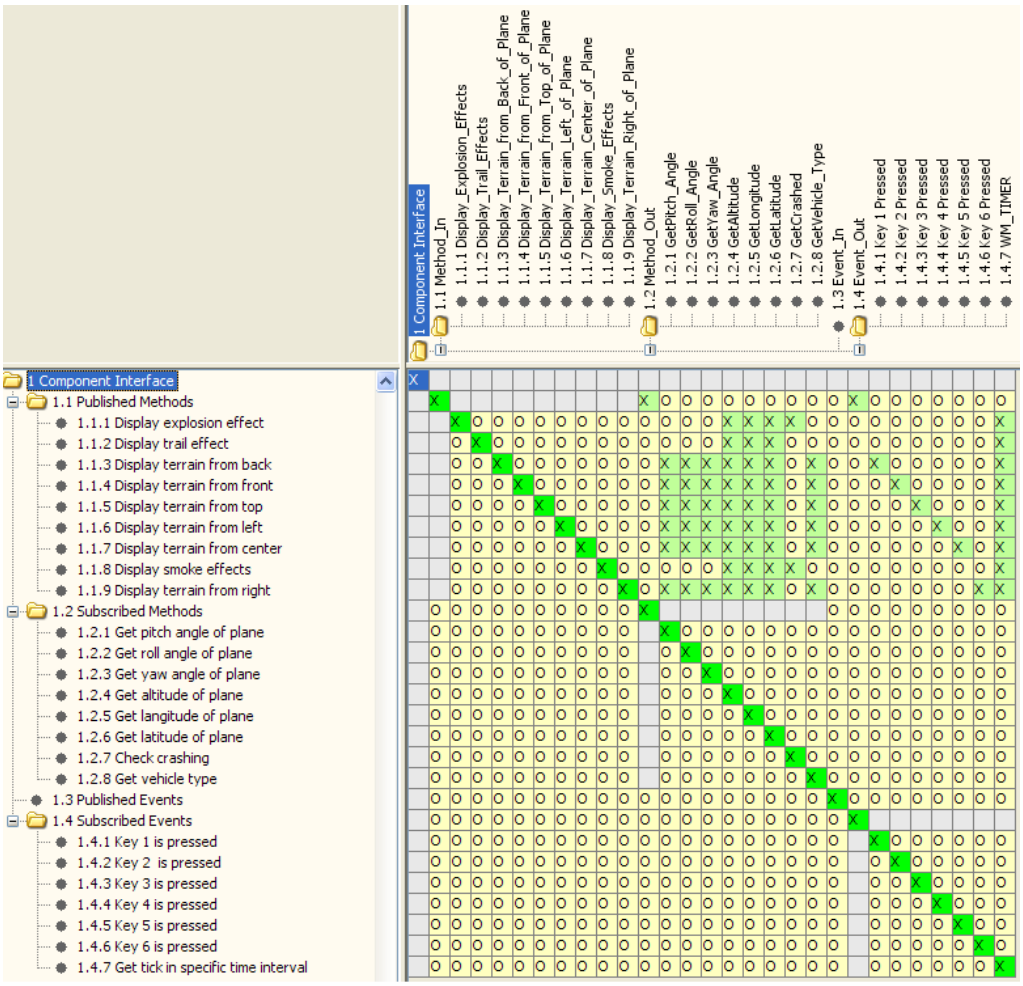

Figure 4.12 FR-DP design matrix of Terrain component

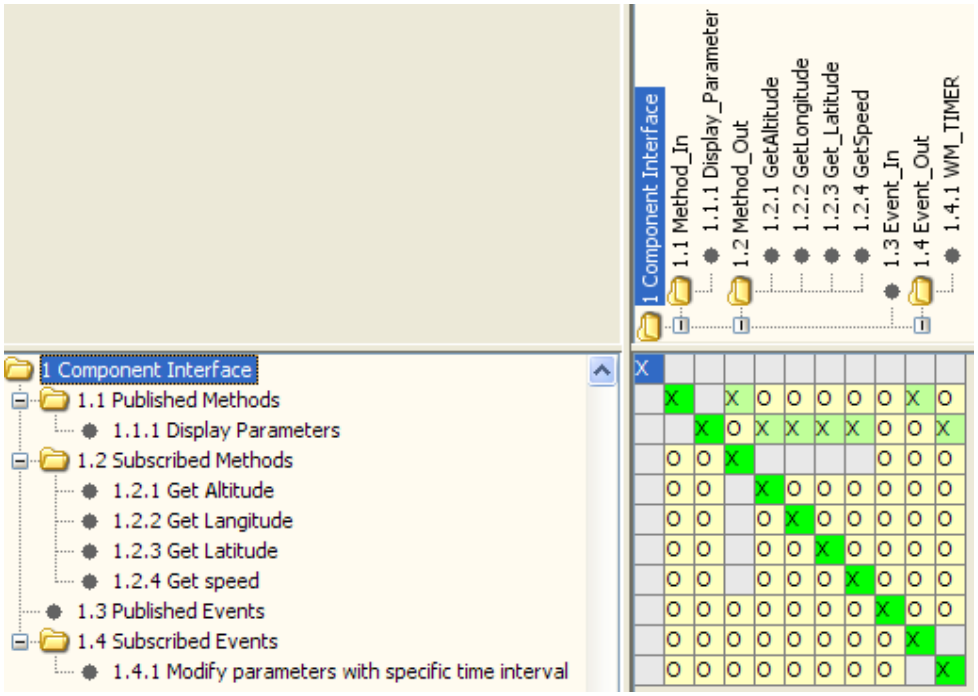

Figure 4.13 FR-DP design matrix of GUI component

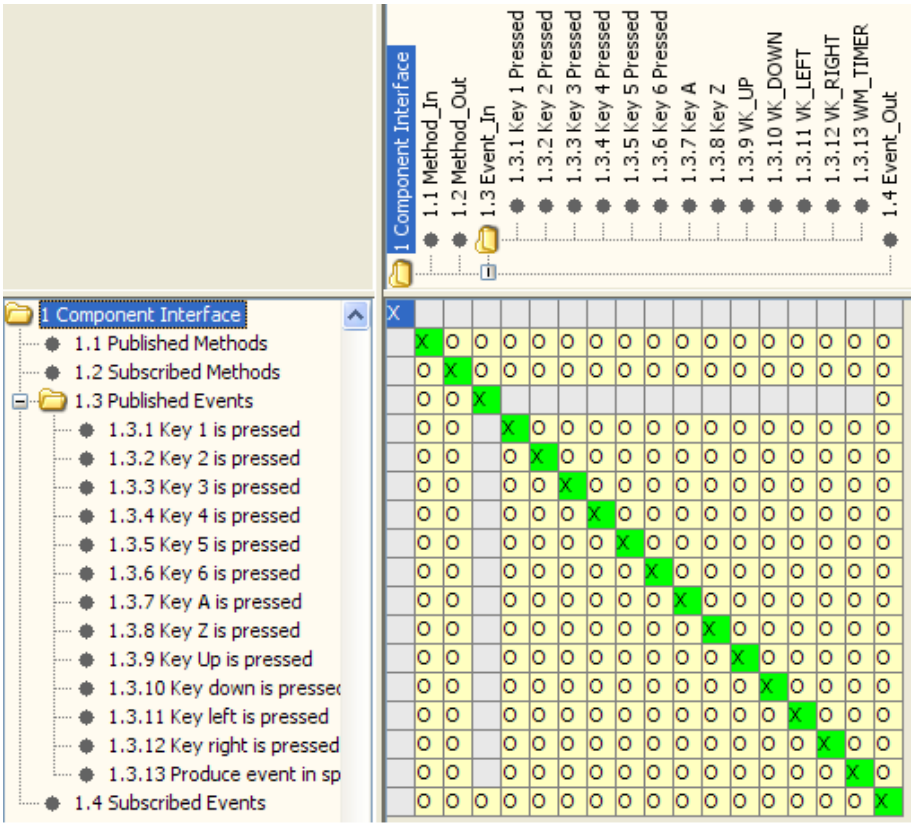

Figure 4.14 FR-DP design matrix of System component

Federates publish some functionalities such as for F16 is represented in Figure 4.15. Federates are represented by communication components *Request* and *Service* used during communication with other federates. Such component representations for F16 federate are depicted in Figure 4.15. The *F16_Request* component accepts subscribed OMT attributes published by other federates through RTI. The OMT attributes are evaluated as events in the components. Values of OMT attributes are used through published methods of the *F16_Request* component by software components. Such as for F16 federate, dependencies between methods and OMT attributes are depicted in Figure 4.16. The *F16_Service* component publishes OMT attributes and they can be reachable by software components through *get* and *set* methods as represented in Figure 4.17.

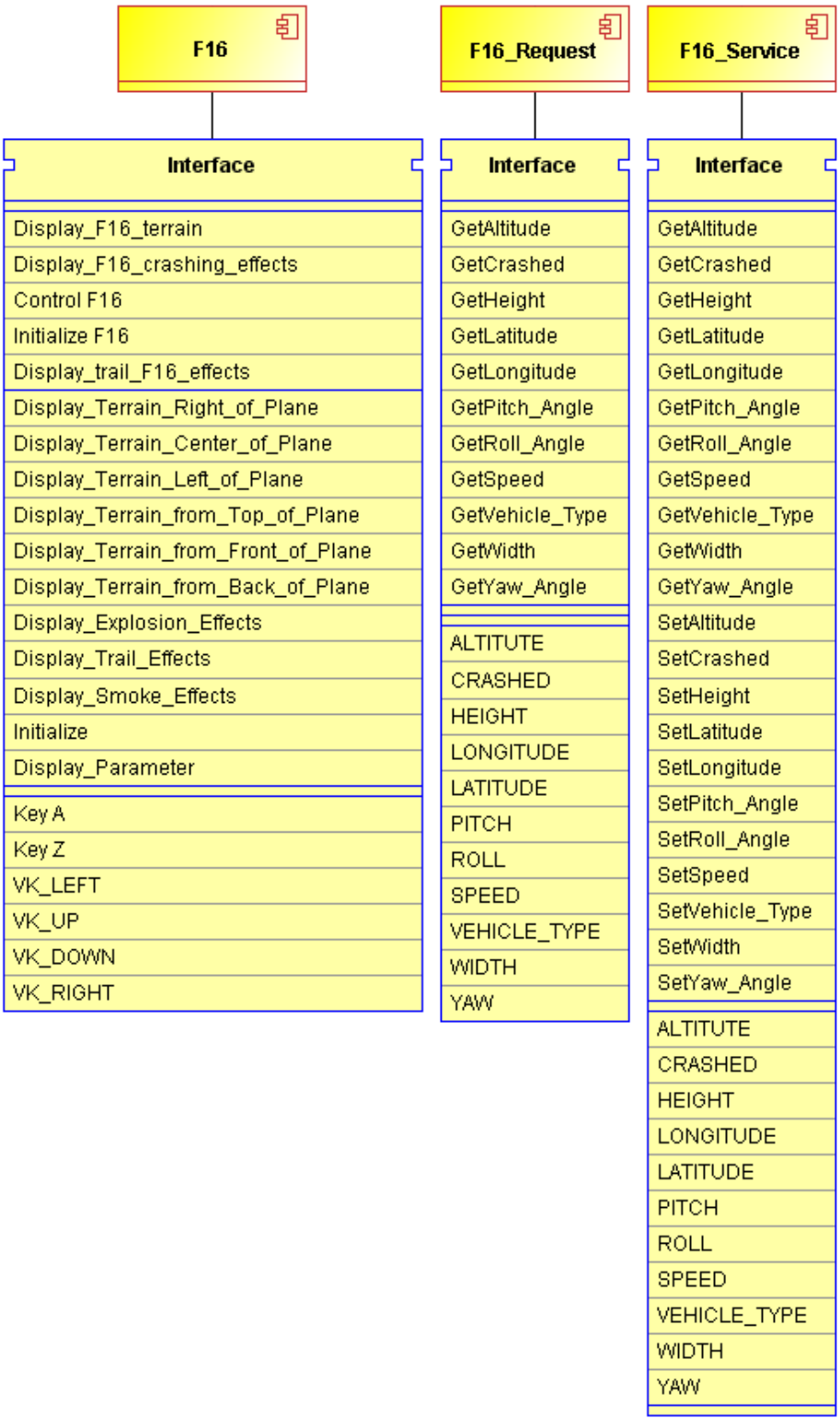

Figure 4.15 F16 component and interfaces

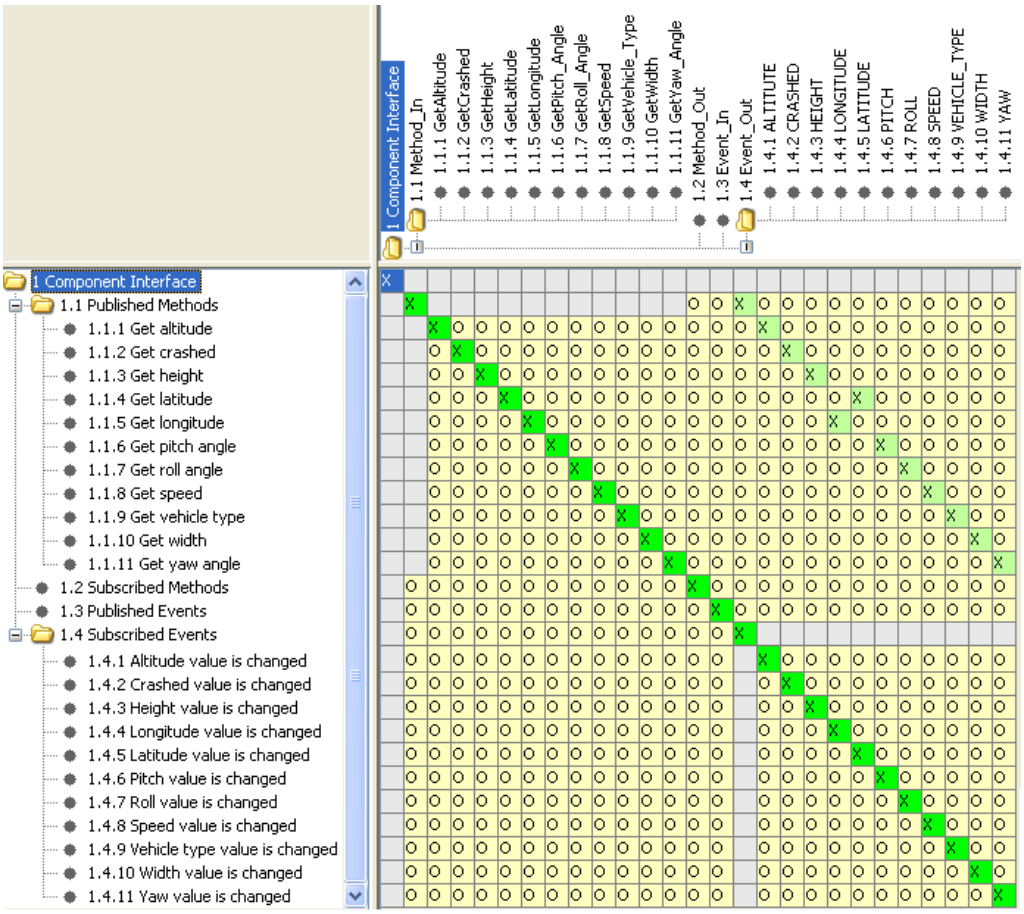

Figure 4.16 F16_Request component

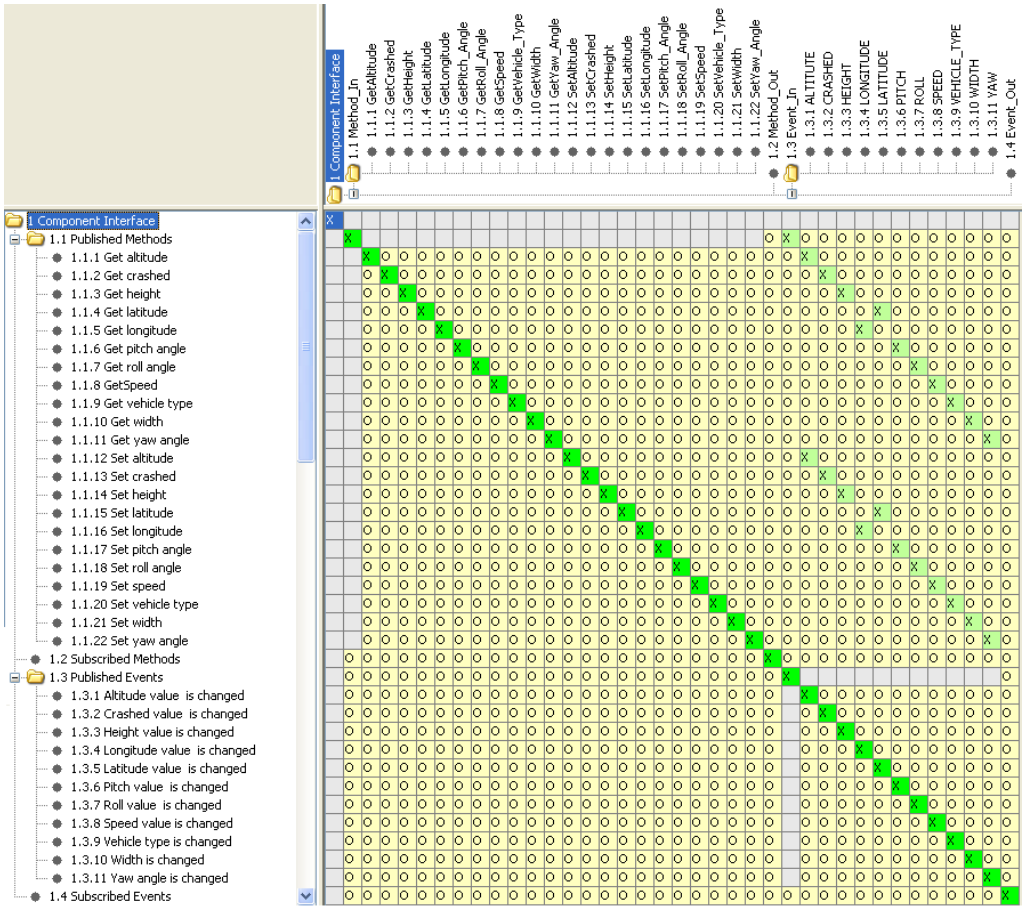

Figure 4.17 F16_Service component

Mature domain includes three federates and their software components as represented in Figure 4.8. Mature domain Feature Model as represented in Figure 4.19 includes features to create federations. Design matrix of the domain as depicted in Figure 4.20 includes functionalities of federations. ADCO guides designer utilizing constraints listed in Table 4.8 and automatically generated constraints from design matrices of components and domain. If *F18* component is extracted from a solution and features are selected as depicted in Figure 4.21, then the design matrix as depicted in Figure 4.22 is obtained. As it can be seen in Figure 4.21, feature *Su25* and *Trail* features are not selected. In this situation, designer is interested in a federation that includes *F16 federate* without trail effect. For this case, in the *F16*

component, *Display_trail_F16_effects* should be omitted and a new *F16* component should be generated. Also related components in *F16* federation with this method can be required modifications. We can extract this knowledge from design matrices and ontology. “FR1.5.1:F16 trail effect” in the application design matrix depicted in Figure 4.22, requires “DP1.5.1:Display_trail_F16_effects” which is published by *F16* component as represented in Figure 4.18. *Display_trail_F16_effects* method requires *Display_trail_effects* published by the *Terrain* component. Since *Display_trail_effects* is not required, then *Terrain* component also can be required to be modified.

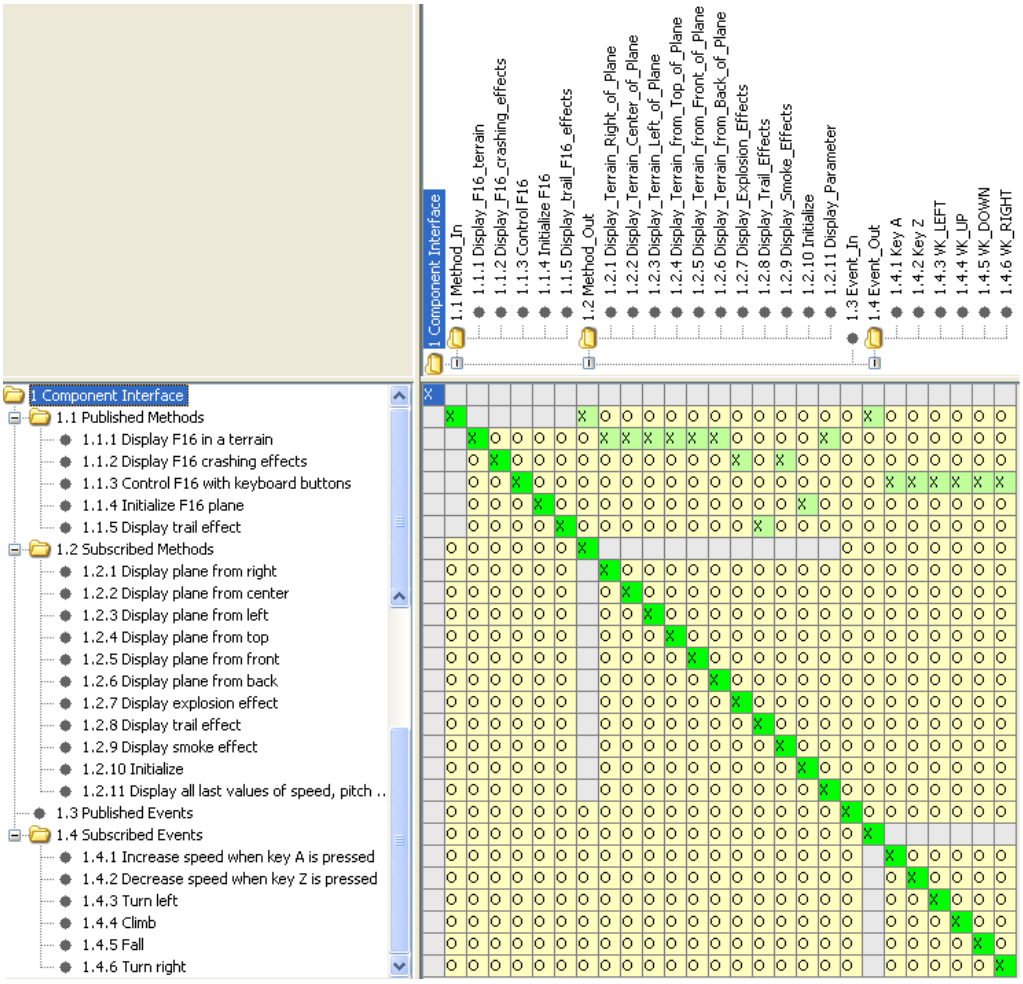

Figure 4.18 FR-DP design matrix of F16 component

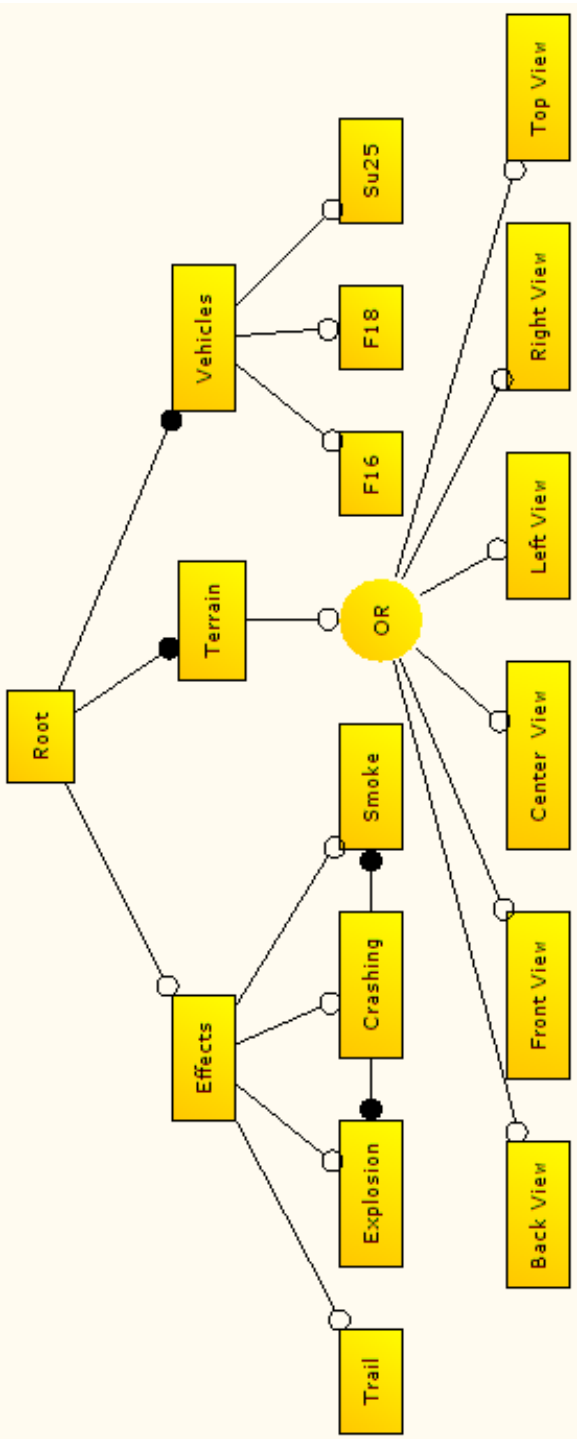

Figure 4.19 Feature Model of simulation domain

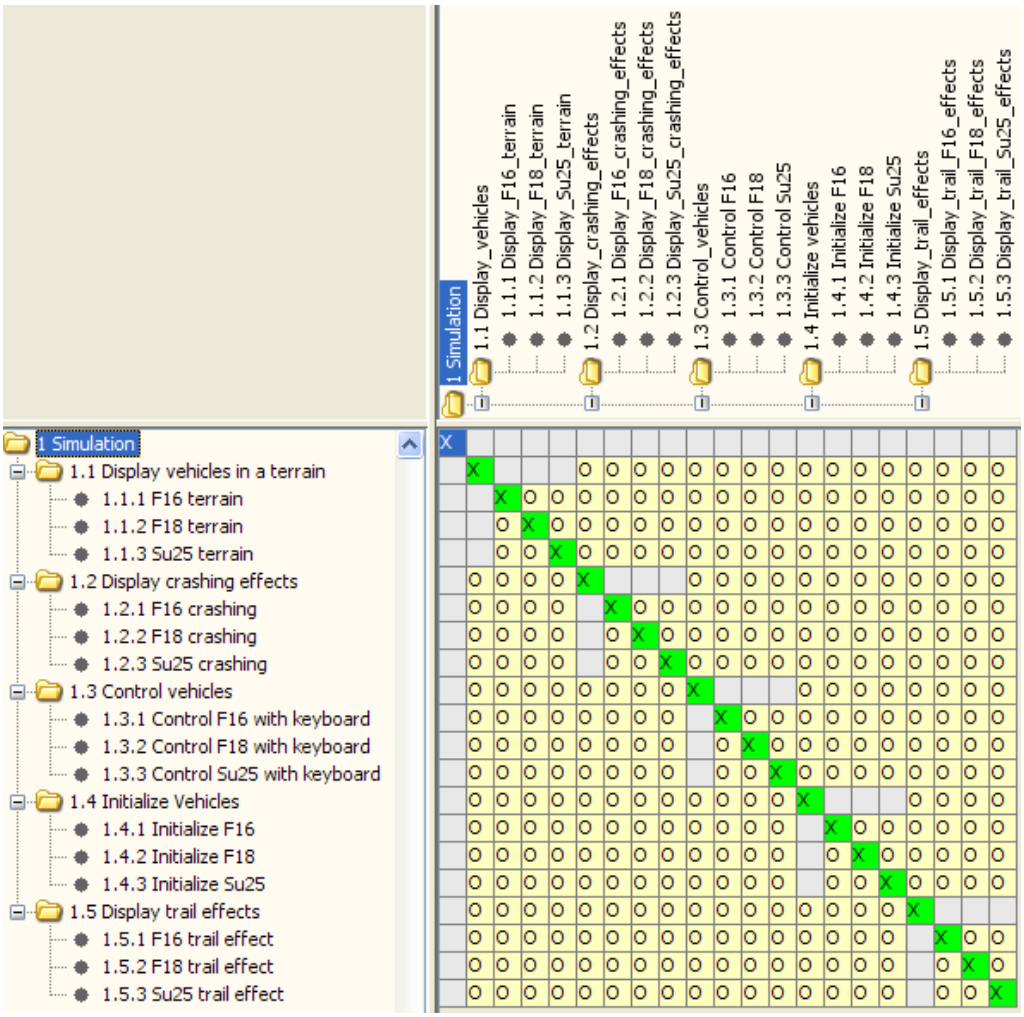

Figure 4.20 Domain FR-DP design matrix of domain and application

Table 4.8 Constraints for simulation domain

No	Constrains
1	FR1⊆ ∃ LinkedtoFR1_1. FR1_1 ⊔ ∃ LinkedtoFR1_3. FR1_3
2	FR1_1⊆ ∃ LinkedtoFR1_1_1. FR1_1_1 ⊔ ∃ LinkedtoFR1_1_2. FR1_1_2 ⊔ ∃ LinkedtoFR1_1_3. FR1_1_3
3	FR1_2⊆ ∃ LinkedtoFR1_2_1. FR1_2_1 ⊔ ∃ LinkedtoFR1_2_2. FR1_2_2 ⊔ ∃ LinkedtoFR1_2_3. FR1_2_3
4	FR1_3⊆ ∃ LinkedtoFR1_3_1. FR1_3_1 ⊔ ∃ LinkedtoFR1_3_2. FR1_3_2 ⊔ ∃ LinkedtoFR1_3_3. FR1_3_3
5	FR1_4⊆ ∃ LinkedtoFR1_4_1. FR1_4_1 ⊔ ∃ LinkedtoFR1_4_2. FR1_4_2 ⊔ ∃ LinkedtoFR1_4_3. FR1_4_3
6	FR1_5⊆ ∃ LinkedtoFR1_5_1. FR1_5_1 ⊔ ∃ LinkedtoFR1_5_2. FR1_5_2 ⊔ ∃ LinkedtoFR1_5_3. FR1_5_3
7	FR1_1_1⊆ ∃ LinkedtoFF16. FF16
8	FR1_1_2⊆ ∃ LinkedtoFF18. FF18
9	FR1_1_3⊆ ∃ LinkedtoFSu25. FSu25
10	FR1_2_1⊆ ∃ LinkedtoFF16. FF16 ⊓ ∃ LinkedtoFCrashing.FCrashing
11	FR1_2_2⊆ ∃ LinkedtoFF18. FF18 ⊓ ∃ LinkedtoFCrashing.FCrashing
12	FR1_2_3⊆ ∃ LinkedtoFSu25. FSu25 ⊓ ∃ LinkedtoFCrashing.FCrashing
13	FR1_3_1⊆ ∃ LinkedtoFF16. FF16
14	FR1_3_2⊆ ∃ LinkedtoFF18. FF18

Table 4.8 (Cont'd)

15	FR1_3_3⊆ ∃ LinkedtoFSu25. FSu25
16	FR1_4_1⊆ ∃ LinkedtoFF16. FF16
17	FR1_4_2⊆ ∃ LinkedtoFF18. FF18
18	FR1_4_3⊆ ∃ LinkedtoFSu25. FSu25
19	FR1_5_1⊆ ∃ LinkedtoFF16. FF16 ⊓ ∃ LinkedtoFTrail.FTrail
20	FR1_5_2⊆ ∃ LinkedtoFF18. FF18 ⊓ ∃ LinkedtoFTrail.FTrail
21	FR1_5_2⊆ ∃ LinkedtoFSu25. FSu25 ⊓ ∃ LinkedtoFTrail.FTrail

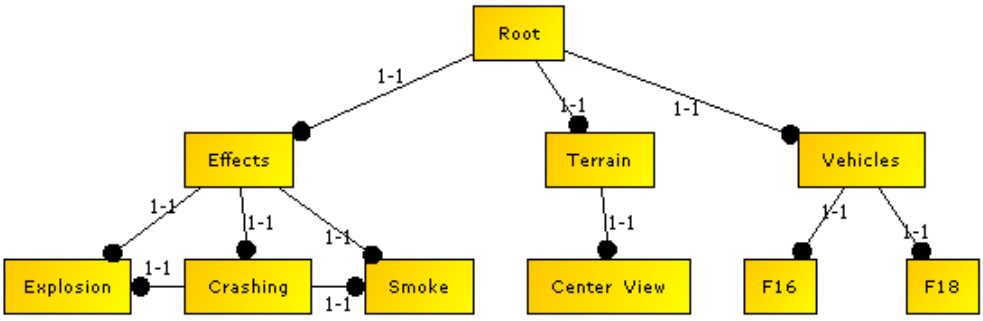

Figure 4.21 Selected features

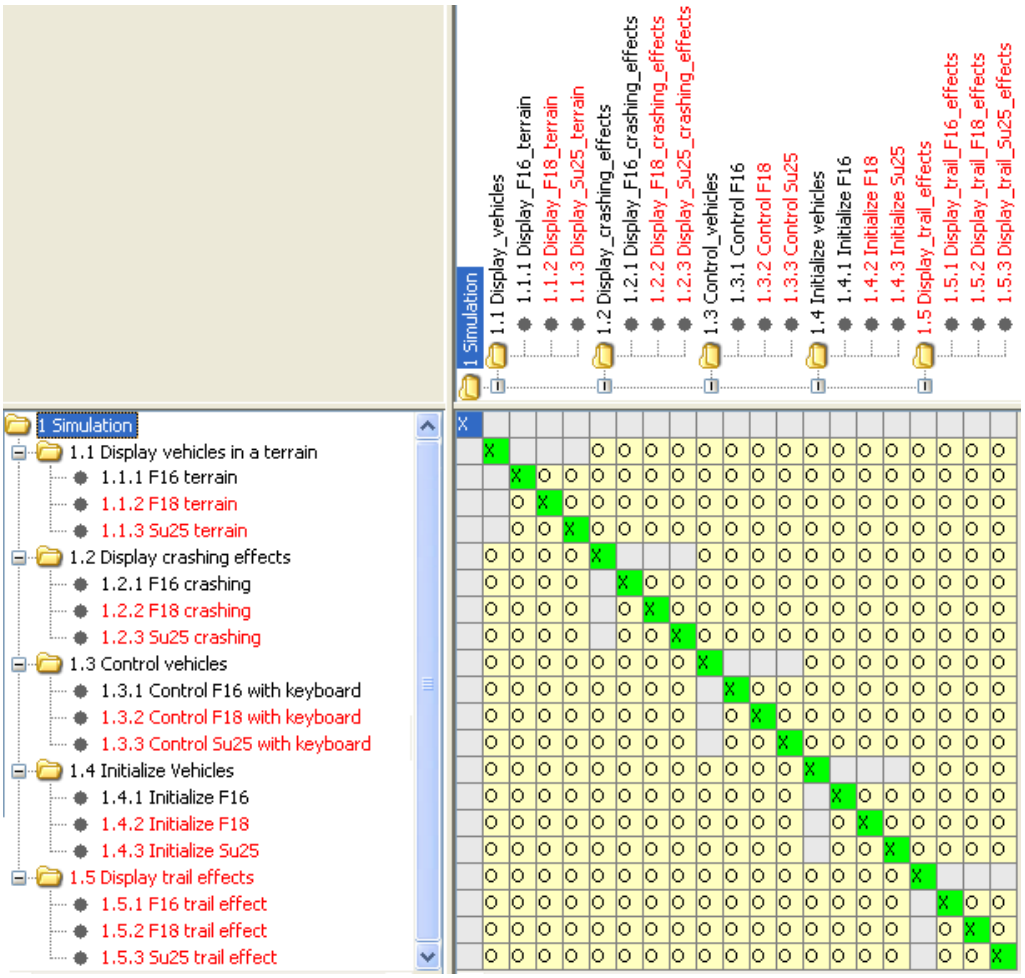

Figure 4.22 Mature Domain FR-DP design matrix of application after constraint evaluation

CHAPTER 5

CONCLUSION

This chapter includes the final remarks about this dissertation. The first section includes the evaluation and critique of the work performed and the following subsection discusses future work and items open for improvement.

5.1. Conducted Work

This research proposed techniques for domain oriented software development, through component facilities. Since component oriented software engineering is based on integration of available components, component utilization can be effective when a set of components that satisfy stakeholders' expectations are located. Therefore, two requirements are identified: 1- definition of stakeholders' expectations and 2- locating components based on the expectations. Customer needs can include various attributes which can range from functional needs to implementation-level details. Customer needs should be understandable by all stakeholders. In order to locate components, they should include enough information. Not only method names and some dependencies but also functional reasons, why a method is defined, should be available. Success of component location and integration is based on the domains' maturity. Without mature domains, component composition and integration suffers from implementation-level problems. Also semantic matching between customer needs and component services can suffer from such problems.

Mature domains can be formed from similar applications (projects). Our goal has been to guide the designers based on their on-going decisions (constraints), available components, and customer needs in mature domains. In order to achieve

this goal, we have proposed Axiomatic Design with Component-Orientation (ADCO). ADCO integrates Axiomatic Design Theory (ADT) and Component-Oriented Software Engineering approaches. ADT with scientific bases is an assurance towards "good design". Customer needs are evaluated and Functional Requirements (FRs) are identified. The FRs are mapped to Design Parameters (DPs). At the end of the design, design parameters are used to locate components which are already developed based on ADT. In ADCO, Feature Models are utilized to capture customer needs and they are mapped to FRs, DPs, and Process Variables (PVs). Experiments/decisions of the design experts are saved in ontology through features, FRs, and DPs. We have implemented an ADCO tool that utilizes reasoning engines operating on the ontology to guide designers in mature domains.

ADCO is based on ADT and we have some contributions to it as listed below:

- Mappings between the ADT domains are represented in an ontology representation for providing guidance through reasoning.
- Constraints are represented in the ontology.
- Feature models are proposed to specify customer needs.
- Collaboration models and process models are incorporated to identify the dependencies between domains.
- A Component-Oriented measurement method based on information content is proposed.

During the research, a deadlock checking mechanism was initially developed and incorporated the Communicating Sequential Processes (CSP) notation and supporting tools. Later it was found that a more general capability was attained by conducting various constraint checking over the ontology. Therefore the final version of the ADCO Tool does not include the CSP based operations.

In conclusion, we have developed the ADCO approach to guide designers in mature domains in an effort to more successfully apply Component-Oriented software development.

5.2. Evaluation

Our tool provides an environment to experiment with the proposed approach. Case studies have operationally demonstrated the functionality of the mechanisms within the specified perspective. It has been observed that for a mediocre design, manual maintenance of the dependencies, constraints and the compatibility of various ADT domains can easily grow to enormous complexities – Applying ADCO the process becomes manageable. Also this highly specialized domain based experience otherwise would be only recorded in the minds of the experts hence yielding the valuable knowledge, to be volatile. Representing this knowledge in the mature domain helps keeping the expertise partially in the organization, ready for duplicated usage.

The involved approaches are all new. Unfortunately there is no compatible method to be used in a comparative evaluation study on our method. Therefore the feasible selection limits us to those involved approaches that were expanded in this study anyway. One of those is Component Oriented Software Engineering (COSE) and the other one is ADT. However, ADT is not component oriented whereas our approach is. Table 5.1 presents a comparison of the proposed ADCO approach with COSE and ADT approaches.

Table 5.1 A comparison of ADCO with COSEML and ADT

Comparison parameter	**ADCO**	**COSEML**	**ADT**
Design Dimension for Decomposition	Functional	Structural	Functional
Decomposition criteria	Yes	No	Yes
Representation of dependencies between methods in a component	Yes	No	No
Component interfaces carry enough information	Yes	No	-
Component modification method	Yes	No	-
Domain support (Feature model)	Yes	No	-
Guidance	Yes	No	Very limited
Product verification	Yes	No	Yes
All components should be known by developer	No	Yes	-

During our case studies, some shortcomings have been observed and they are listed below:

- Training is required for utilizing ADT for component orientation.
- Tracing of the dependencies becomes a problem in complex systems.
- Complexity of the Feature Models can be a problem during handling consistency of the model and selection of the features. This is a common problem for all the approaches utilizing Feature Models.
- Alternative DPs are not considered.

- Creating & modifying mature domains have difficulties.

- After new configuration based on feature selections, some components need to be modified. This disadvantage will decrease in time because of increased maturity.

- Non-functional requirements are not directly addressed. Although a feature-based approach can treat most of such items similar to functional requirements, an exclusive presentation of non-functional requirements could prove useful.

As an overall assessment we can conclude that the approach can prove useful in the software industry.

5.3. Future Work

This work has been a first attempt trying to enhance component-orientation through design guidance. There are many directions this pioneer work can expand. Some shortcomings are listed in the previous section already. As an initial list of possible improvements, we are planning to enrich the ADCO tool by:

- integrating use case diagrams, collaboration diagrams, information axiom calculation facility which was already implemented in C++ language,

- implementing partitioning algorithms for design matrices in an effort to triangularize coupled designs,

- implementing DP-PV design matrix for showing the connections between the DPs and components,

- implementing a master design matrix for the system under development, that includes only the leaf level FRs and DPs – for easier visualization,

- implementing a capability to propose solution sets from available components, and

- implementing a capability to identify missing components automatically.

We are also planning to adapt ADCO approach to Product Line Architecture (PLA). Some theoretical work is already in progress in this venue.

REFERENCES

[1] Akbiyik, E.K., Suloglu, S., Togay, C. and Dogru, A.H., *Service Oriented Systems Design Through Process Decomposition*, Proceedings of the The Eleventh World Conference on Integrated Design and Process Technology, Tauichung, Taiwan, 2008, pp. 332-338.

[2] Allen, R. and Garlen, D., *A formal basis for architectural connection*, ACM Transactions on Software Engineering and Methodology, Vol. 6*,* No. 3, 1997, pp. 213-249.

[3] Allen, R.J., Garlan, D. and Ivers, J., *Formal modeling and analysis of the HLA component integration standard*, Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering, 1998.

[4] Allen, R.J. and Garlen, D., *A Formal Approach to Software Architecture*, Carnegie Mellon University, 1997.

[5] Baader, F., Calvanese, D., McGuinness, D., Nardi, D. and Patel-Schneider, P., *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.

[6] Batory, d., *Feature-oriented programming and the AHEAD tool suite*, Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), 2004, pp. 702- 703.

[7] Batory, D., *Feature Models, Grammars, and Propositional Formulas*, Lecture Notes in Computer Science, Vol. 3714*,*2005, pp. 7-20.

[8] Batory, D., Benevides, D. and Ruiz-Cortes, A., *Automated Analysis of Feature Models: Challenges Ahead*, Communication of the ACM, Vol. 49*,* No. 12, 2006, pp. 45-47.

[9] Benavides, D., Segura, S., Trinidad, P. and Cortés, A.R., *Using Java CSP Solvers in the Automated Analyses of Feature Models*, Lecture Notes in Computer Science, Vol. 4143*,* No. 2006, 2006, pp. 399-408.

[10] Bergner, K., Rausch, A., Sihling, M. and Vilbig, A., *A Componentware Development Methodology based on Process Patterns*, Proceedings of the Pattern Languages of Programs Conference (PLOP98), Monticello, Illinois, 1998, pp. 11-14.

[11] Bertoa, M.F., Troya, J.M. and Vallecillo, A., *A Survey on the Quality Information Provided by Software Component Vendors*, Proceedings of the tth ECOOP Workshop on Quantitative Approaches in Object Oriented Software Engineering, Darmstadt, Germany, 2003.

[12] Bertoa, M.F. and Vallecillo, A., *Quality Attributes for COTS Components*, Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object Oriented Software Engineering, Malaga, Spain, 2002.

[13] Beugnard, A., Jezequel, J.-M., Plouzeau, N. and Watkins, D., *Making Components Contract Aware*, IEEE Computer, Vol. 32, No. 7, 1999, pp. 38-45.

[14] Bicer, V. and Togay, C., *Representing Feature Models with Semantic Web Ontologies*, Proceedings of the Ulusal Yazilim Mimarisi Konferansi (UYMK), Istanbul, Turkey, 2006.

[15] Browning, T.R., *Applying the design structure matrix to system decomposition and integration problems: a review and new directions*, IEEE Transactions on Engineering Management, Vol. 48, No. 3, 2001, pp. 292-306.

[16] Bühne, S., Lauenroth, K. and Pohl, K., *Why is it not Sufficient to Model Requirements Variability with Feature Models?*, Proceedings of the Automotive Requirements Engineering (AURE04), Nagoya, Japan, 2004, pp. 5-12.

[17] Cechticky, V., Pasetti, A., Rohlik, O. and Schaufelberger, W., *XML-Based Feature Modelling*, Lecture Notes in Computer Science, Vol. 3107, No. 2004, 2004, pp. 101-114.

[18] Christopher, A., *Notes on the Synthesis of Form*, Harvard University Press, Cambridge, 1964.

[19] Clapis, P.J. and Hintersteiner, J.D., *Enhancing Object Oriented Software Development through Axiomatic Design*, Proceedings of the First International Conference on Axiomatic Design, Cambridge, MA, 2000.

[20] Clements, P. and Northrop, L., *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.

[21] Clements, P.C., *From Subroutines to Subsystems: Component-Based Software Development*, The American Programmer, Vol. 8, No. 11, 1995.

[22] Czarnecki, K. and Eisenecker, U., *Generative Programming: Methods, tools, and Applications*, Addison-Wesley, 2000.

[23] Czarnecki, K., Helsen, S. and Eisenecker, U., *Formalizing Cardinality-based Feature Models and Their Specialization*, Formalizing Cardinality-based Feature Models and Their Specialization, Vol. 10,2005, pp. 7-29.

[24] Czarnecki, K., Helsen, S. and Eisenecker, U., *Staged Configuration Using Feature Models*, Procedings of SPLC 2004, Lecture Notes in Computer Science, Vol. 3154,2004, pp. 266-283.

[25] Czarnecki, K., Kim, C.H.P. and Kalleberg, K.T., *Feature Models are Views on Ontologies 10th International Software Product Line Conference (SPLC'06)* 2006, pp. 41-51.

[26] Czarnecki, K. and Pietroszek, K., *Verifying feature-based model templates against well-formedness OCL constraints*, Proceedings of the Proceedings of the 5th international conference on Generative programming and component engineering, Portland, Oregon, USA, 2006 pp. 211-220.

[27] D'Ambrogio, A. and Gianni, D., *Using CORBA to Enhance HLA Interoperability in Distributed and Web-Based Simulation*, Proceedings of the 19th International Symposium on Computer and Information Sciences (ISCIS'04), Lecture Notes in Computer Science, Vol. 3280/2004,2004, pp. 696-705.

[28] Dahmann, J., Salisbury, M., Turrell, C., Barry, P. and Blemberg, P., *HLA and Beyond: Interoperability Challenges*, Proceedings of the The 1999 Fall Simulation Interoperability Workshop, Orlando,FL, 1999.

[29] Dahmann, J.S. and Morse, K.L., *High Level Architecture for simulation: an update*, Proceedings of the Proceedings of the Second International Workshop on Distributed Interactive Simulation and Real-Time Applications, Montreal, Que., Canada, 1998, IEEE Computer Society

[30] Davis, A.M., *The Design of a Familiy of Application-Oriented Requirements Language*, Computer, Vol. 15, No. 5, 1982, pp. 21-28.

[31] Deursen, A.v. and Klint, P., *Domain-specific language design requires feature descriptions*, Journal of Computing and Information Technology2001, pp. 1-20.

[32] Do, S.H. and Park, G.J., *Application of Design Axioms for Glass-Bulb Design and Software Development for Design Automation*, Proceedings of the Third CIRP Workshop on Design and Implementation of Intelligent Manufacturing, Tokyo, Japan, 1996, pp. 119-126.

[33] Do, S.H. and Suh, N.P., *Object Oriented Software Design with Axiomatic Design*, Proceedings of the Proceedings of ICAD2000 First International Conference on Axiomatic Design, Cambridge, 2000.

[34] Do, S.H. and Suh, N.P., *Systematic OO Programming with Axiomatic Design*, IEEE Computer, Vol. 32, No. 10, pp. 121-124.

[35] Dogru, A.H., *Component-Oriented Software Engineering*, The Academy of Learning and Advances Studies (The ATLAS), Dallas, 2006.

[36] Dogru, A.H., *Component Oriented Software Engineering Modeling Language:COSEML*, Proceedings of the Computer Engineering Department, Middle East Technical University,Turkey, TR 99-3, 1999.

[37] Dogru, A.H. and Tanik, M.M., *A Process Model for Component-Oriented Software Engineering*, IEEE Software, Vol. 20, No. 2, pp. 34-41.

[38] Eén, N. and Sörensson, N., *An extensible SAT solver*, Lecture Notes in Computer Science, Vol. 2919, No. 2004, 2004, pp. 502-518.

[39] Ferber, S., Haag, J. and Savolainen, J., *Feature Interaction and Dependencies: Modeling Features for Reengineering a Legacy Product Line*, Lecture Notes in Computer Science, Vol. 2379, No. 2002, 2002, pp. 37-60.

[40] Formal Systems Ltd., *Failures-Divergence-Refinement: FDR2 User Manual*, 2003, http:// www.fsel.com/documentation/probe/probe-doc.pdf, last accessed date: 01/05/2008

[41] Formal Systems Ltd., *Process Behavior Explorer: Probe User Manual*, 2003, http:// http://www.fsel.com/documentation/fdr2/fdr2manual.pdf, last accessed date: 01/05/2008

[42] Grimm, S., Hitzler, P. and Abecker, A., *Knowledge Representation and Ontologies*, in Semantic Web Services, Springer Berlin Heidelberg, 2007, pp. 51-105.

[43] Griss, M.L., Favaro, J. and d'Alessandro, M., *Integrating feature modeling with the RSEB*, in Fifth International Conference on Software Reuse, Victoria, BC, Canada, 1998, pp. 76-85.

[44] Guenov, M.D. and Barker, S.G., *Application of Axiomatic Design and Design Structure Matrix to the Decomposition of Engineering Systems*, Systems Engineering, Vol. 8, No. 1, 2005, pp. 29-40.

[45] Gumus, B., *Axiomatic Product Development Lifecycle*, Mechanical Enginnering, Texas Tech University, PhD Dissertation, 2005.

[46] Gumus, B. and Ertas, A., *Requirement Management and Axiomatic Design*, Journal of Integrated Design and Process Science, Vol. 8, No. 4, 2004, pp. 19-31.

[47] Harmon, S.Y. and Youngblood, S.M., *Leveraging Fidelity to Achieve Substantive Interoperability*, Proceedings of the Spring 2001 Simulation Interoperability Workshop, Orlando, FL, 2001.

[48] Hoare, C.A.R., *Communicating Sequential Processes*, Communication of the ACM, Vol. 21, No. 8, 1978, pp. 666-677.

[49] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B. and Dean, M., *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, 2004, http://www.w3.org/Submission/SWRL, last accessed date: 01/05/2008

[50] Huizing, M., *Component Based Development*, Component Technology, Vol. 6, No. 2, pp. 5-9.

[51] IEEE Std 1278.1-1995, *IEEE Standard for Distributed Interactive Simulation (DIS)--Application Protocol*, 1995.

[52] IEEE Std 1278.2-1995, *IEEE Standard for Distributed Interactive Simulation (DIS)--Communication Services and Profiles*, 1995.

[53] IEEE Std. 830-1998, *IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998)*, IEEE Press New York, 1998.

[54] IEEE Std. 1516-2000, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules*, 2000.

[55] IEEE Std. 1516.1-2000, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Federate Interface Specification*, , 2000.

[56] IEEE Std. 1516.2, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Object Model Template (OMT) Specification*, 2000.

[57] IEEE Std. 1516.3-2003, *IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP)*, 2003.

[58] Inverardi, P. and Uchitel, S., *Proving Deadlock Freedom in Component-Based Programming*, Proceedings of the Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering, 2001, pp. 65-75.

[59] Iribarne, L., Troya, J.M. and Vallecillo, A., *Selecting software components with multiple interfaces*, Proceedings of the Euromicro Conference, 2002, pp. 26- 32.

[60] ISO/IEC 9126-1:2001, *Software Engineering—Product Quality—Part 1: Quality model*, June, 2001.

[61] Jia, Y. and Gu, Y., *The Representation of Component Semantics: A Feature-Oriented Approach*, Proceedings of the Component-based Software Engineering Workshop: Composing Systems from Components (ECBS2002), Lund, SWEDEN, 2002.

[62] Jololian, L.K., Ngatchou, J.C. and Seker, R., *A Component Integration Meta-Framework using Smart Adapters*, Proceedings of the IEEE Proceedings of the 2004 International Symposium on Information and Communication Technologies Las Vegas, Nevada, 2004, ACM, pp. 128-133.

[63] Kalaoja, J., Niemela, E. and Perunka, H., *Feature modelling of component-based embedded software*, Proceedings of the 8th International Workshop on Software Technology and Engineering Practice (STEP '97), 1997, pp. 444-451.

[64] Kang, K.C., Cohen, S.G., Hess, J.A., Nowak, W.E. and Peterson, A.S., *Feature Oriented Domain Analysis Feasibility Study*, Carnegie Mellon University, Pittsburg, PA,, 1990.

[65] Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E. and Huh, M., *FORM: A feature-oriented reuse method with domain-specific reference architectures*, J. C. Baltzer AG, Science Publishers Red Bank, NJ, USA, 1998.

[66] Kang, K.C., Kim, S., Lee, J. and Lee, K., *Feature-oriented engineering of PBX software for adaptability and reuseability*, Software: Practice and Experience, Vol. 29*,* No. 10, 1999, pp. 875-896.

[67] Kang, K.C., Lee, J. and Donohoe, P., *Feature-Oriented Product Line Engineering*, IEEE Software, Vol. 19*,* No. 4, 2002, pp. 58-65.

[68] Kang, K.C., Lee, K., Lee, J. and Kim, S., *Feature Oriented Product Line Software Engineering: Pricinples and Guidelines*, in Hirota, T., Itoh, K. and Kumagai, S. eds, Domain Oriented Systems Development: Perspectives and Practices Routledge, UK, 2003.

[69] Kar, A.K., *Linking Axiomatic Design and Taguchi Methods via Information Content in Design*, Proceedings of the First International Conference on Axiomatic Design, Cambridge, 2000, pp. 219-224.

[70] Kim, I.-G., Bae, D.-H. and Hong, J.-E., *A component composition model providing dynamic, flexible, and hierarchical composition of components for supporting software evolution*, The Journal of Systems and Software, Vol. doi:10.1016/j.jss.2007.02.047,2007.

[71] Kim, M., Yang, H. and Park, S., *A Domain Analysis Method for Software Product Lines Based on Scenarios, Goals and Features* Proceedings of the 10th Asia-Pacific Software Engineering Conference (APSEC'03), 2003, pp. 126-135.

[72] Lee, K., Kang, K.C., Chae, W. and Choi, B.W., *Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse* Sofware practive and experience, Vol. 30, No. 9, 2000, pp. 1025-1046.

[73] Lee, K., Kang, K.C., Kim, M. and Park, S., *Combining Feature-Oriented Analysis and Aspect-Oriented Programming for Product Line Asset Development*, Proceedings of the The 10th International on Software Product Line Conference, 2006 pp. 103 - 112

[74] Lees, M., Logan, B. and Theodoropoulos, G., *Agents, games and HLA*, Simulation Modeling Practice and Theory, Vol. 14,2006, pp. 752-767.

[75] Liu, D. and Mei, H., *Mapping requirements to software architecture by feature-orientation*, Proceedings of the STRAW'03 Second International Software Requirements to Architectures Workshop, Portland, Oregen, 2003.

[76] Lloyd, W.J., *A Common Criteria Based Approach for COTS Component Selection*, Journal of Object Technology, Vol. 4, No. 4, 2005, pp. 27-34.

[77] Lüer, C. and Rosenblum, D.S., *WREN---an environment for component-based development*, Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, Vienna, Austria, 2001, pp. 207-217.

[78] McGuinness, D.L. and Harmelen, F.v., *OWL Web Ontology Language Overview*, 2004, http://www.w3.org/TR/owl-features, last accessed date: 01/05/2008

[79] Mei, J. and Bontas, E.P., *Reasoning Paradigms for SWRL-enabled Ontologies.*, Proceedings of the International Workshop on Protege with Rules, Madrid, Spain, 2005.

[80] Melvin, J.W., *Axiomatic System Design: Chemical Mechanical Polishing Machine Case Study*, Mechanical Engineering, Massachusetts Institute of Technology, PhD Dissertation, 2003.

[81] Morse, K.L., Lightner, M., Little, R., Lutz, B. and Scrudder, R., *Enabling Simulation Interoperability*, IEEE Computer, Vol. 39*,* No. 1, pp. 115-117.

[82] Olewnik, A.T. and Lewis, K., *On Validating Engineering Design Decision Support Tools*, Concurrent Engineering, Vol. 13*,*2005, pp. 111-121.

[83] OMG, *UML 2.0 OCL Specification*, 2003*, http://www.omg.org/docs/ptc/03-10-14.pdf*, last accessed date: 01/05/2008

[84] Oses, N., Pidd, M. and Brooks, R.J., *Critical issues in the development of component-based discrete simulation*, Simulation Modeling Practice and Theory, Vol. 12*,*2004, pp. 495-514.

[85] Pashov, I. and Reiebisch, M., *Using feature modeling for program comprehension and software architecture recovery*, Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2004, pp. 406- 417.

[86] Pashov, I., Reiebisch, M. and Philippow, I., *Supporting architectural restructuring by analyzing feature models*, Proceedings of the Eighth European Conference on Software Maintenance and Reengineering, 2004, pp. 25-34.

[87] Pawletta, S., Drewelow, W. and Pawletta, T., *HLA-based Simulation within an Interactive Engineering Environment*, Proceedings of the 4th International Workshop on Distributed Simulation and Real Time Applications (DS-RT 2000), San Francisco, California, USA, 2000.

[88] Peng, X., Wu, Y. and Zhao, W., *A Feature-Oriented Adaptive Component Model for Dynamic Evolution*, Proceedings of the 11th European Conference on Software Maintenance and Reengineering CSMR '07, 2007.

[89] Peng, X., Zhao, W., Xue, Y. and Wu, Y., *Ontology-Based Feature Modeling and Application-Oriented Tailoring*, Lecture Notes in Computer Science, Vol. 4039*,* No. 2006, 2006.

[90] Philippow, I., Riebisch, M. and Boellert, K., *The Hyper/UML Approach for Feature Based Software Design*, Proceedings of the The 4th AOSD Modeling With UML Workshop, 2003.

[91] Pimentel, A.R. and Stadzisz, P.C., *Application of the Independence Axiom on the Design of Object Oriented Software using the Axiomatic Design*

Theory, Journal of Integrated Design & Process Science, Vol. 10*,* No. 1, 2006, pp. 57-69.

[92] Pimmler, T.U. and Eppinger, S.D., *Integration Analysis of Product Decomposition*, Proceedings of the ASME Design Theory and Methodogy Conference, 1994.

[93] Reiser, M.O. and Weber, M., *Multi-level feature trees A pragmatic approach to managing highly complex product families*, Requirements Engineering, Vol. 12*,* No. 2, 2007, pp. 57-75.

[94] Robak, S. and Franczyk, B., *Modeling Web Services Variability with Feature Diagrams*, Lecture Notes in Computer Science, Vol. 2593*,* No. 2008, 2008, pp. 120-128.

[95] Roubtsova, E.E. and Roubtsov, S.A., *A Feature Computation Tree Model to Specify Requirements and Reuse*, Proceedings of the ICEIS 2006 - Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases and Information Systems Integration, Paphos, Cyprus, 2006, pp. 118-125.

[96] Simon, H.A., *The Science of the Artificial*, The MIT Press, 1969.

[97] Steward, D., *System Analysis and Management: Structure, Strategy and Design*, Petrocelli Books, New York, 1981.

[98] Steward, D. and Tate, D., Integration of Axiomatic Design and Project Planning, Proceedings of the First International Conference on Axiomatic Design, Cambridge, 2000, pp. 286-289.

[99] Su, J.C.Y., Chen, S.J.G. and Lin, L., *A structured approach to measuring functional dependency and sequencing of coupled tasks in engineering design*, Computers and Industrial Engineering, Vol. 45*,* No. 1, 2003, pp. 195-214.

[100] Suh, N.P., *Axiomatic Design Theory for Systems*, Research in Engineering Design, Vol. 10*,* No. 4, 1998, pp. 189-209.

[101] Suh, N.P., *Axiomatic Design: Advantages and Applications*, Oxford University Press, New York, 2001.

[102] Suh, N.P., *Complexity: Theory and Applications*, Oxfor University Press, New York, 2005.

[103] Suh, N.P., *Designing-in of Quality Through Axiomatic Design*, IEEE Transactions on Reliability, Vol. 44*,* No. 2, 1995, pp. 256-264.

[104] Suh, N.P., *The Principles of design*, Oxford University Press, New York, 1990.

[105] Szyperski, C., *Component Software – Beyond Object-Oriented Programming*, Addison-Wesley and ACM Press, 1999.

[106] Tanik, M.M. and Chan, E.S., *Fundamentals of Computing for Software Engineers*, Van Nostrand Reinhold, New York, 1991.

[107] Tate, D. and Cha, J., *The Relationship between Axiomatic Design and Grid Engineering*, Proceedings of the 11th International Conference on Concurrent Engineering, Beijing, China, 2004.

[108] Taylor, S.J.E., *HLA-CSPIF:The high level architecture-COTS simulation package interoperation forum*, Proceedings of the Fall Simulation Interoperability Workshop, Orlando, FL, 2003.

[109] Taylor, S.J.E., Wang, X. and Turner, S.J., *Integrating Heterogeneous Distributed COTS Discrete-Event Simulation Packages: An Emerging Standards-Based Approach*, IEEE Transaction on Systems, Man, and Cybernetics-Part A: Systems and Humans, Vol. 36, No. 1, January 2006, pp. 109-122.

[110] Togay, C., *HLA Tabanli Bilesenler ile Otomatik Uygulama Gelistirme*, Proceedings of the Ulusal Yazilim Muhendisligi Sempozyumu, Ankara, Turkey, 2005.

[111] Togay, C., Aktunc, O., Tanik, M.M. and Dogru, A.H., *Measurement of Component Congruity for Composition Based on Axiomatic Design*, Proceedings of the The Ninth World Conference on Integrated Design and Process Technology, San Diego, CA, 2006.

[112] Togay, C., Bicer, V. and Dogru, A.H., *Deadlock Detection in High Level Architecture Federations Using Axiomatic Design Theory*, Proceedings of the EUROSIM07, Ljubljiana, Slovenia, 2007, pp. 139.

[113] Togay, C. and Dogru, A.H., *Aksiyomatik Tasarim ile Benzetim Bileşen Ara Yüzlerinde Kazanımlar*, Proceedings of the SAVTEK 2006, Savunma Teknolojileri Kongresi, Ankara, 2006.

[114] Togay, C. and Dogru, A.H., *Component Oriented Design Based on Axiomatic Design Theory and COSEML*, Proceedings of ISCIS, Lecture Notes in Computer Science, Vol. 4263/2006,2006, pp. 1072-1079.

[115] Togay, C. and Dogru, A.H., *Federasyonların HLA Tabanlı Simulasyonlara Tümleştirilme Otomasyonu için bir Mekanizma*, Proceedings of the 1. Ulusal

Savunma Uygulamaları Modelleme Simülasyon Konferansı, Ankara, Turkey, 2005.

[116] Togay, C. and Dogru, A.H., *A Framework for Component Integration Using Axiomatic Design and Object Model Template for Simulation Applications*, Department of Electrical and Computer Engineering University of Alabama, Birmingham, Alabama, 2005.

[117] Togay, C. and Dogru, A.H., *Infrastructure Design for HLA Based Automated Federation Development*, Proceedings of the The Eighth World Conference on Integrated Design and Process Technology, Beijing, China, 2005, pp. 698-704.

[118] Togay, C., Dogru, A.H. and Tanik, U.J., *Systematic Component-Oriented Development with Axiomatic Design*, Journal of Systems and Software, Vol. doi: 10.1016/j.jss.2007.12.746*,*2008.

[119] Togay, C., Dogru, A.H., Tanik, U.J. and Grimes, G.J., *Component Oriented Simulation Development With Axiomatic Design*, Proceedings of the The Ninth World Conference on Integrated Design and Process Technology, San Diego, CA, 2006.

[120] Togay, C., Sundar, G. and Dogru, A.H., *Detection of Component Composition Mismatch with Axiomatic Design*, Proceedings of the IEEE Southern Conference, Memphis, TN, 2006, IEEE.

[121] Traas, V. and Hillegersberg, J.v., *The software component market on the internet current status and conditions for growth*, Software Engineering Notes, Vol. 25*,* No. 1, 2000, pp. 114-117.

[122] Tsarkov, D. and Horrocks, I., *FaCT++ Description Logic Reasoner: System Description*, Lecture Notes in Artificial Intelligence, Vol. 4130*,*2006, pp. 292-297.

[123] Tuncel, M.B., *Using Collaboration Diagrams in Component Oriented Modeling*, Computer Engineering Dept., Middle East Technical University, Master Thesis, 2006.

[124] Turner, C.R., Fuggetta, A., Lavazza, L. and Wolf, A.L., *A conceptual basis for feature engineering*, The Journal of Systems and Software, Vol. 49*,* No. 1, 1999, pp. 3-5.

[125] Wang, H., Li, Y.F., Sun, J., Zhang, H. and Pan, J., *A SemanticWeb Approach to Feature Modeling and Verification*, Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE'05), 2005.

[126] Wang, H., Li, Y.F., Sun, J., Zhang, H. and Pan, J., *Verifiying feature models using OWL*, Web Semantics: Science, Services, and Agents on the World Wide Web, Vol. 5*,* No. 2, 2007, pp. 117-129.

[127] Yellin, D.M. and Strom, R.E., *Protocol Specifications and Component Adaptors*, ACM Transactions on Programming Languages and Systems, Vol. 19*,* No. 2, 1997, pp. 292-333.

[128] Yeung, W.L., *Mapping WS-CDL and BPEL into CSP for Behavioral Specification and Verification of Web Services*, Proceedings of the Web Services ECOWS '06, 2006, pp. 297-305.

[129] Yeung, W.L., Wang, J. and Dong, W., *Verifying Choreographic Descriptions of Web Services Based on CSP*, Proceedings of the The IEEE Services Computing Workshops, 2006, pp. 97-104.

[130] Yi, J.W. and Park, G.J., *Development of a design system for EPS cushioning package of a monitor using axiomatic design*, Advances in Engineering Software, Vol. 36*,*2005, pp. 273-284.

[131] Zakongroup, *OpenConf*, 2007, http://www.zakongroup.com/technology/openconf.shtml, last accessed date: 01/05/2008.

APPENDIX A

CONFERENCE MANAGEMENT SYSTEM COMPONENTS

A.1. Author Component

Design matrix of *Author 1* component is represented in Figure A.1. There are two published methods namely *Add_Author* and *Delete_Author*. There is a subscribed method namely *SQL_Execution*. As it can be seen in Figure A.1, both published methods require the *SQL_Execution* method.

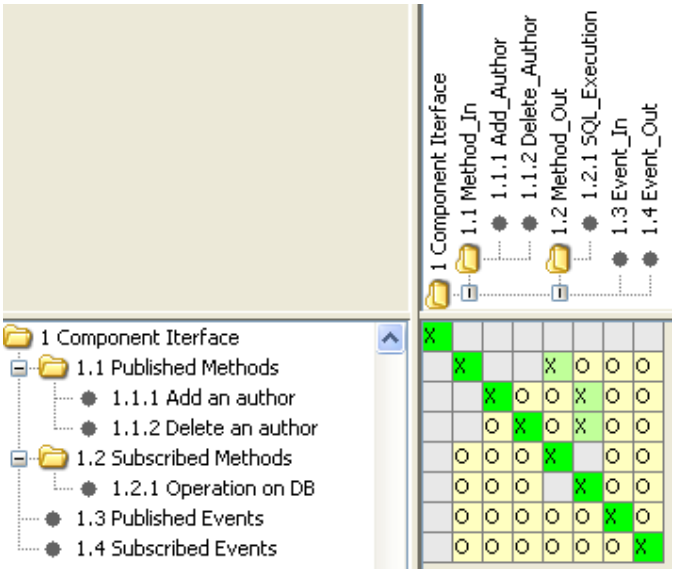

Figure A.1 Design matrix of Author component

A.2. Database Component

Database component is used to utilize database operations. There is no subscribed method. *SQL_Execution* method requires the *Open_DB* and *Close_DB* methods as depicted in Figure A.2. Published methods can be required by other published methods for instance *Open_DB* and *Close_DB* methods utilize *SQL_Execution* method as depicted in Figure A.2.

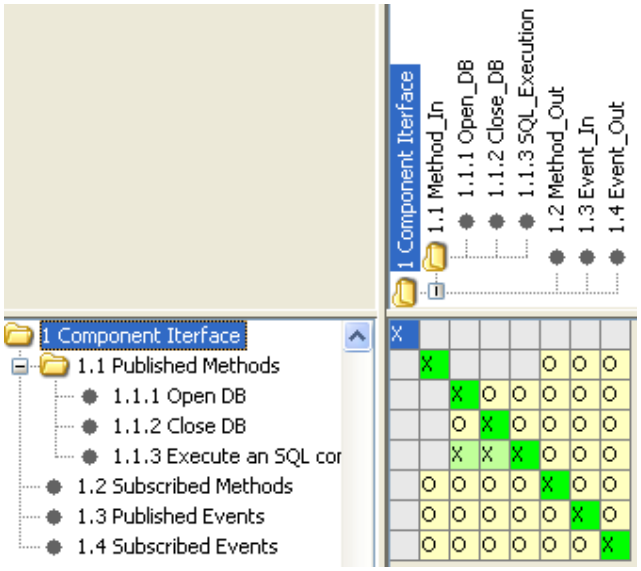

Figure A.2 Design matrix of Database component

A.3. Edit Component

Edit component provides methods for satisfying editing operations. There are two published and three subscribed methods as depicted in Figure A.3. *Login* method utilizes *Get_Contact_Password* method published by *Paper* Component as depicted in Figure A.5 to control password.

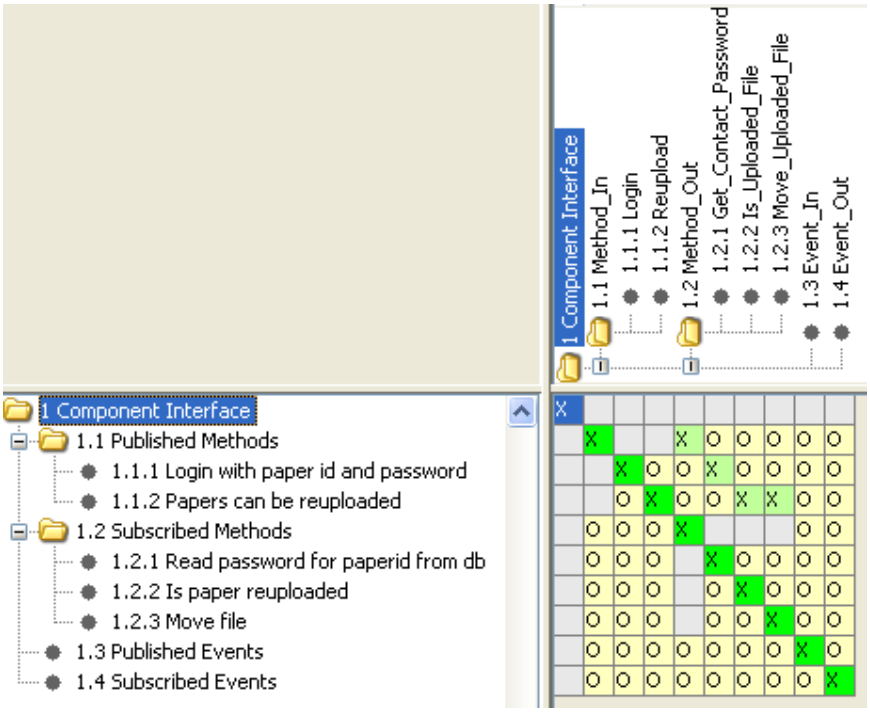

Figure A.3 Design matrix of Edit component

A.4. File System Component

File System component is utilized for file system operations as depicted in Figure A.4. This component is one of the core components and provides upload service. Although, there can be more methods we prefer to represent critical methods for our case study. There are no subscribed methods.

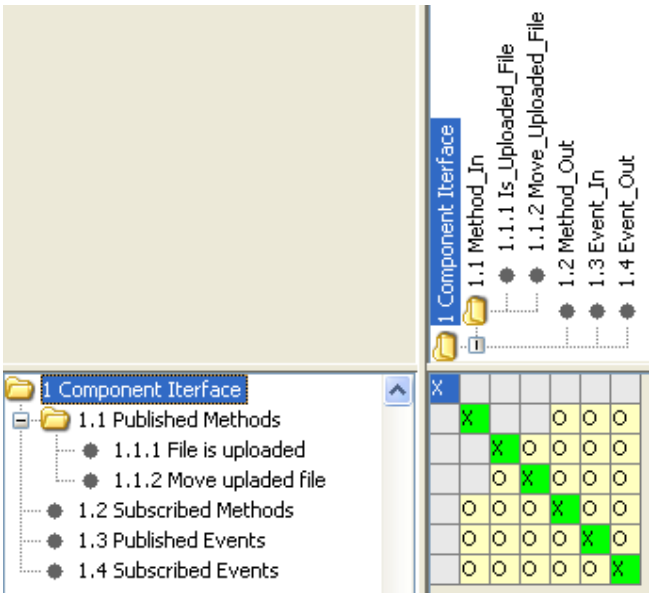

Figure A.4 Design matrix of File System component

A.5. Paper Component

Paper component provides all paper operations on database utilizing *SQL_Execution* method as depicted in Figure A.5.

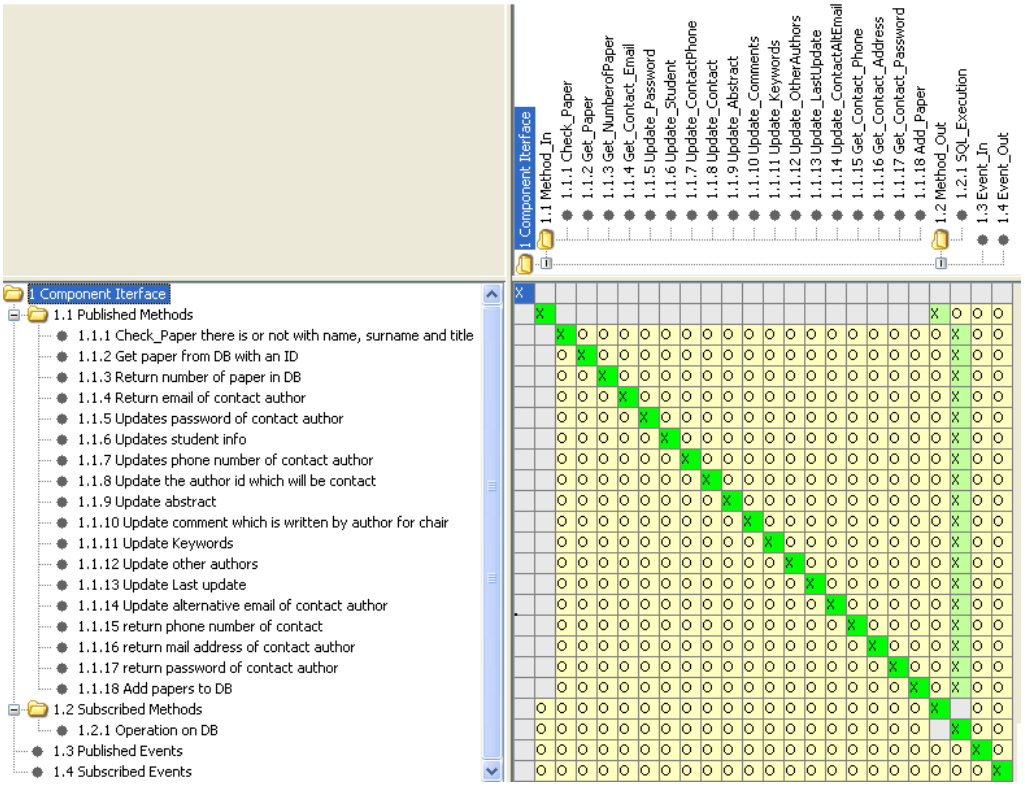

Figure A.5 Design matrix of Paper component

A.6. Paper Topic Component

Paper Topic component provides all topic operations on database utilizing *SQL_Execution* method as depicted in Figure A.6.

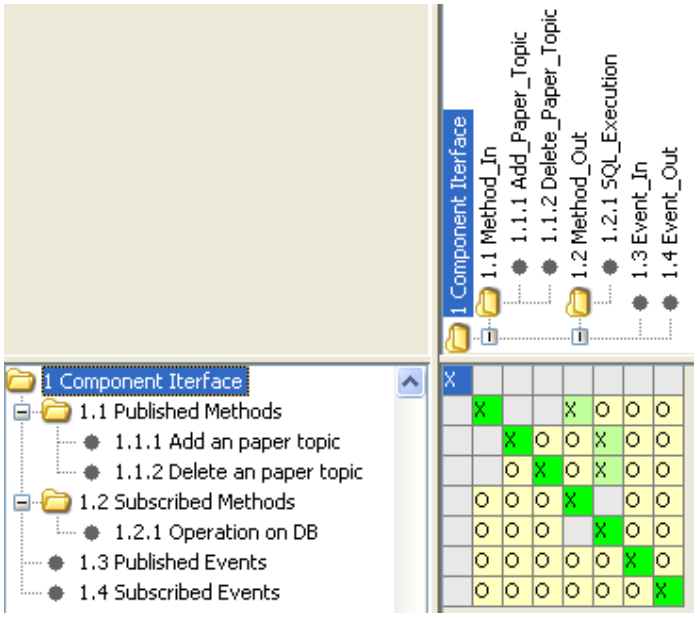

Figure A.6 Design matrix of Paper Topic component

A.7. Submit Component

Submit component is a complex component including methods to satisfy submitting operations with utilizing other components as depicted in Figure A.7. *Submit* method uses other published methods and represented in partial process diagram as depicted in Figure A.8.

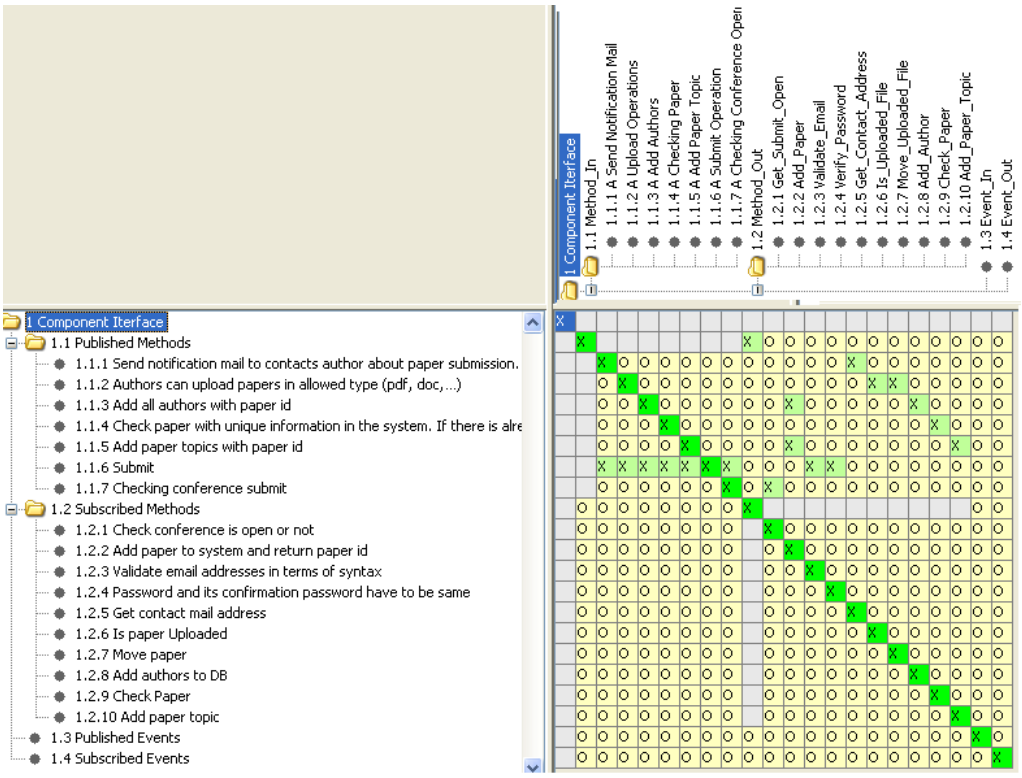

Figure A.7 Design matrix of Submit component

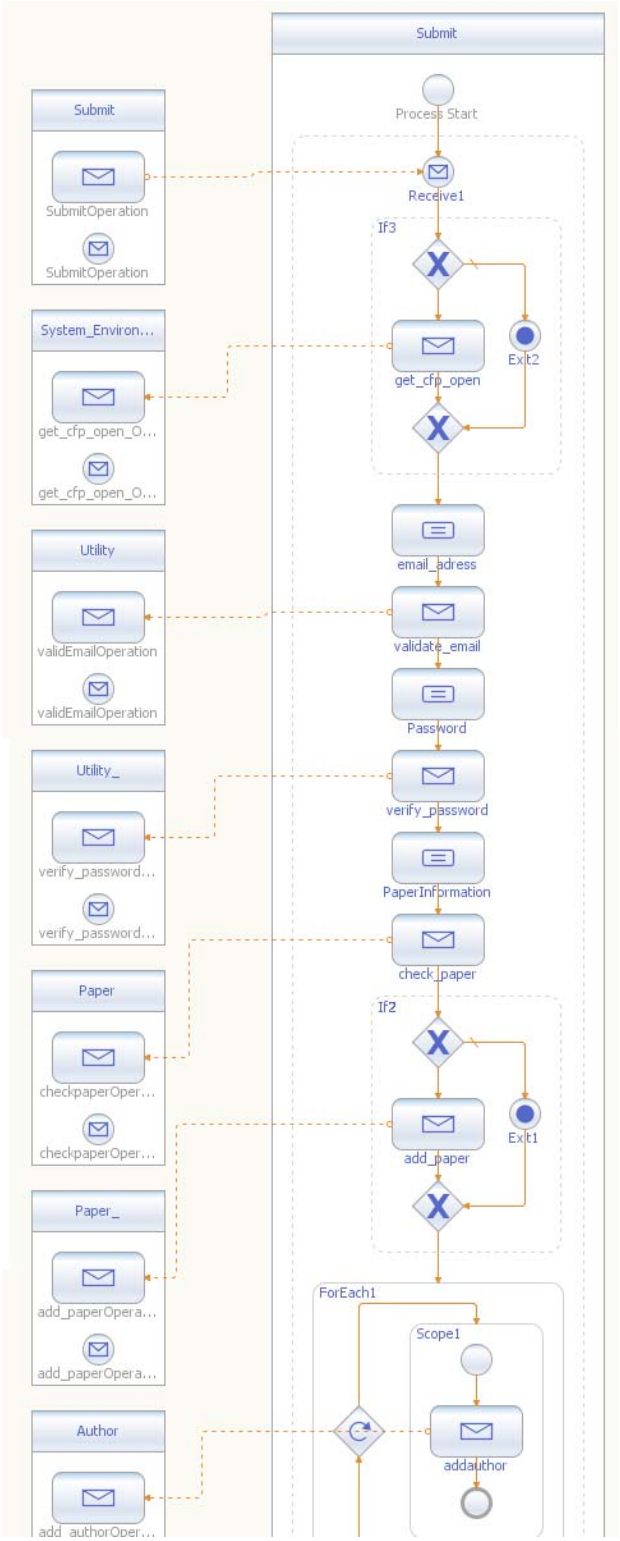

Figure A.8 Partial process diagram of submit method (adapted from [1])

A.8. System Environment Component

System Environment component provides environment information about conference as depicted in Figure A.8. This information is saved in a file.

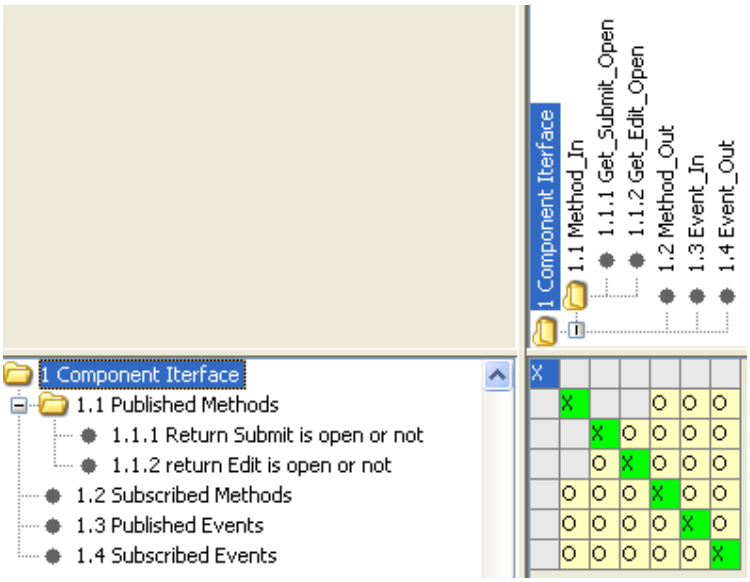

Figure A.9 Design matrix of System Environment component

A.9. Utility Component

Utility component provides some useful methods as depicted in Figure A.9. Such as validate email evaluates the congruity of the email address with standards.

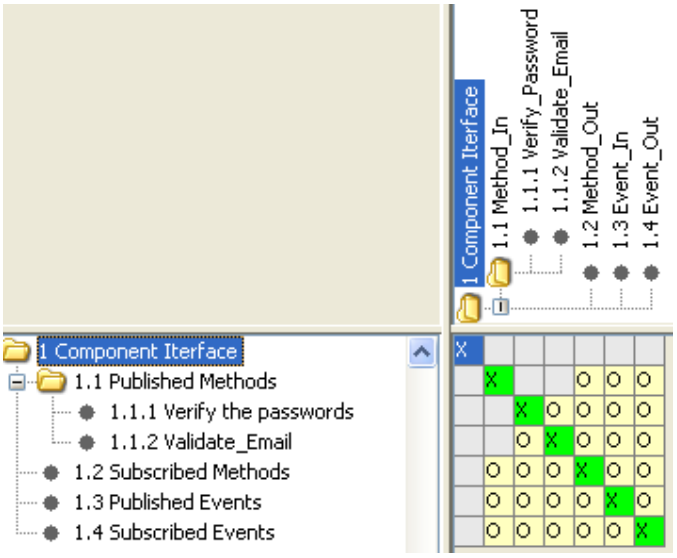

Figure A.10 Design matrix of Utility component

CURRICULUM VITAE

PERSONEL INFORMATION

Surname, Name: Toğay, Cengiz
Nationality: Turkish (TC)
Date and Place of Birth: 30 October 1977, Hatay
Marital Status: Married
Phone: +90 312 2105533
Fax: +90 312 2105544

EDUCATION

Degree	Institution	Year of Graduation
MS	Çanakkale Onsekiz Mart University, Computer Engineering Department	2001
BS	Çanakkale Onsekiz Mart University, Computer Engineering Department	1999
High School	İskenderun Lisesi	1994

WORK EXPERIENCE

Year	Place	Enrollment
2001-Present	Middle East Technical University, Computer Engineering Department	Research Assistant
1999-2001	Canakkale Onsekiz Mart University, Computer Engineering Department	Research Assistant
1997-1999	Figensoft	Programmer

FOREIGN LANGUAGE

English

PUBLICATIONS

National Conferences

1. **Togay, C.,** Dogru, A.H, "Federasyonların HLA Tabanlı Benzetimlere Tümleştirilme Otomasyonu için bir Mekanizma", 1. Ulusal Savunma Uygulamaları Modelleme Simülasyon Konferansı, 2005.

2. **Togay, C.,** "HLA Tabanli Bileşenler ile Otomatik Uygulama Geliştirme", II. Ulusal Yazilim Muhendisligi Sempozyumu (UYMS) 05, pg: 243-251, 22-24 September 2005.

3. **Togay, C.,** Dogru, A.H., "Aksiyomatik Tasarim ile Benzetim Bileşen Ara Yüzlerinde Kazanımlar", SAVTEK 2006, Savunma Teknolojileri Kongresi, vol. 2, Pg: 201-208, 29-30 June 2006, Ankara.

4. Bicer, V. and **Togay, C.,** "Representing Feature Models with Semantic Web Ontologies", Ulusal Yazilim Muhendisligi Konferansi, pg. 70-78, Istanbul, 2006.

International Conferences

1. **Togay, C.,** Dogru, A.H, "Infrastructure Design for HLA Based Automated Federation Development", The Eighth World Conference on Integrated Design and Process Technology, Beijing, China, June 12-16, 2005, pp: 698-704.

2. **Togay, C.,** Sundar, G., Dogru, A.H., "Detection of Component Composition Mismatch with Axiomatic Design", IEEESouthEastCon06, March 31- April 2, Memphis, USA.

3. Bicer, V., **Togay, C.,** Dogru, A.H., "Service-Oriented e-learning Systems with Axiomatic Design", The Ninth World Conference on Integrated Design and Process Technology, San Diego, California, June 25-30, 2006.

4. **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.

5. **Togay, C.,** Aktunc, O., Tanik. M., Dogru, A.H., "Measurement of Component Congruity for Composition based on Axiomatic Design", The Ninth World Conference on Integrated Design and Process Technology, San Diego, California, June 25-30, 2006.

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. Bicer, V., **Togay, C.,** Dogru, A.H., "A model Driven Approach for Service-Centric System Development", The Tenth World Conference on Integrated Design and Process Technology, pg: 175-182, Antalya, Turkey, June 3-8, 2007.

8. **Togay, C.,** Bicer, V., Dogru, A.H., "Deadlock Detection in High Level Architecture Federations using Axiomatic Design Theory", EUROSIM07, Slovenia, September, 2007.

9. Akbiyik, E.K., Suloglu, S., **Togay, C.,** Dogru, A.H.,"Service Oriented Systems Design Through Process Decomposition", The Eleventh World Conference on Integrated Design and Process Technology, pg: 332-338, Tauichung, Taiwan, June 1-6, 2008.

10. Alkislar, L., Ergun, R., **Togay, C.,** Dogru, A.H., "Fault Avoidance for Mission Critical Systems", The Eleventh World Conference on Integrated Design and Process Technology, pg: 326-331, Tauichung, Taiwan, June 1-6, 2008.

International Journals

1. **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

Technical Reports

1. **Togay, C.,** Dogru, A.H., "A Framework for Component Integration Using Axiomatic Design and Object Model Template", Technical Report, 2005-11-ECE-001, Department of Electrical and Computer Engineering, University of Alabama, December, 2005.

2. Kaya, O., Alkislar, L., **Togay, C.,** Dogru, A.H., "Modeling Fault Management Domain with Fault Avoidance Capability", Technical Report, METU-CENG-TR-2007-12, Department of Computer Engineering, METU, December, 2007.