A SURVEY ON SYSTEMS ARCHITECTURE DEVELOPMENT AND A HELICOPTER COMMUNICATION SYSTEM CONCEPTUAL DESIGN

by

Mubin FAKIOĞLU

Submitted to the Institute of Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
in
System Engineering

Yeditepe University
2009

A SURVEY ON SYSTEMS ARCHITECTURE DEVELOPMENT AND A HELICOPTER COMMUNICATION SYSTEM CONCEPTUAL DESIGN

Approved by:

Assoc. Prof. Dr. Kudret YURTSEVEN ……..
(Advisor)

Prof. Dr. Murat TUNÇ ……..

Asst.Prof.Dr. Zeynep OCAK ……

Date of Approval: / / 2009

ACKNOWLEDGEMENTS

I would like to thank to, my adviser Assoc.Prof. Dr. M. Kudret YURTSEVEN for his help and advices, Zehra ALAŞAN for her encouraging thoughts, Gürkan EĞRİCAN and Cüneyt ÖZEN for their support, my colleques in TAI, Turkish Aerospace Industries Inc., especially Gökhan ÇİYAN, for their comments, and my family for their support.

ABSTRACT

A SURVEY ON SYSTEMS ARCHITECTURE DEVELOPMENT AND A HELICOPTER COMMUNICATION SYSTEM CONCEPTUAL DESIGN

A system is commonly defined to be a collection of hardware, software, people, and procedures organized to accomplish some common objectives. These objectives are required by the stakeholders of the system. Systems are not developed at a point in time. The system development process to bring a system into being and into operational use from user requirements, requires a systems development life cycle approach that includes analysis, design, implementation, integration, maintenance and retirement. To obtain efficient systems, in the design process of the system, system's architecture is built to manage to prevent design conflicts and undesired solutions. System's architecting contributes to the development of a system from its initial concept until its retirement from use in this life cycle process.

This thesis mainly focuses on system's architecture design context and systems architecture design methodologies. In the first chapter of the study, system development life cycle models and system's architecting design process is introduced. In addition, the context of the systems architecure is explained. In the next chapter, Structured Architecture Methodology and Object Oriented Architecture Methodology are introduced and explained. In the last chapter the study is concluded by architecting a communication system of an attack helicopter by Structured Systems Architecture Methodology.

Key Terms: Systems Architecture Context, Systems Architecture Methodologies, Structured Architecture Methodology, Object-Oriented Architecture Methodology

ÖZET

SİSTEM MİMARİSİ GELİŞTİRİLMESİ ÜZERİNE ARAŞTIRMA VE BİR HELİKOPTERİN HABERLEŞME SİSTEMİNİN KAVRAMSAL TASARIMI

Sistemler, ortak bir amaca ulaşmak için biraraya gelmiş donanım, yazılım, ve insanlar gibi farklı işlevleri olan olguların bir bütünüdür. Sistemlerin ulaşmaya çalıştığı amaç, sistemlerin kullanıcıları tarafından belirlenir. Sistemler, zaman içerisinde bir anda meydana gelmezler. Kullanıcı isteklerini operasyonel kullanım alanlarında yerine getirebilen bir sistem geliştirmek, analiz, tasarım, bütünleştirme, uygulama, bakım ve emeklilik fazlarını içeren bir süreç içerisinde gerçekleşir. Bu süreç, Sistem Geliştirme Yaşam Döngüsü olarak adlandırılır. Bu süreç içerisinde, istenilen özelliklerde ve performansta sistemleri geliştirebilmek için, sistem mimarileri oluşturulur. Sistem mimarileri, sistemlerin hatalı, çelişkili ve istenmeyen özelliklerde olmasını engeller. Sistem mimarileri, sistem tasarımı sürecinde başından sonuna kadar varolurlar.

Bu tez, sistem mimarilerinin içeriği ve sistem mimarisi geliştirme metodolojileri hakkındadır. Çalışmanın ilk bölümü, sistem geliştirme yaşam döngüsünü ve modellerini içerir. Bu bölüm daha sonra sistem mimarisinin özellikleri ve sistem mimarisi tasarımı ile sistem mimarisi içeriğinde yer alması gereken unsurların anlatıldığı bölümleri içerir. İkinci bölüm, sistem mimarisi geliştirmede kullanılan, Yapısal Sistem Mimarisi, ve Nesnesel Sistem Mimarisi metodolojilerini içerir. Çalışmanın son bölümünde, Yapısal Sistem Mimarisi Metodolojisi kullanılarak bir saldırı helikopterinde kullanılmak istenen haberleşme sisteminin kavramsal sistem mimarisinin geliştirildiği örnek uygulama yer almaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANSI	American National Standards Institute
COTS	Commercial of the Shelf
DAG	Defense Acquisition Guidebook
DoD	U.S. Department of Defense
DoDAF	U.S. Department of Defence Architecture Framework
IEEE	Institute for Electrical and Electronics Engineers
INCOSE	International Council on Systems Engineering
ISO	International Standards Organization
MoDAF	Ministry of Defence Architecture Framework
NDIA	National Defense Industries Association
OO	Object Oriented
SE	Systems Engineering
SoS	System of Systems
SysML	System Modeling Language
UML	Unified Modeling Language

1. INTRODUCTION

A System is a collection of components organized to accomplish a specific function or set of functions [1], due to the stakeholders system considerations such as performance, reliability, security, etc. For instance, a government wants to meet an international challenge by safely sending astronauts to the moon and getting them back. Military services needing nearly undetectable strike aircraft is another example for bringing a system into being to achieve a set of specified requirements.

The system development process to put systems into operational use from user requirements requires a systems development life cycle approach. There are various types of SDLC models in use in the literature, but most are grounded in one of three models, which are Royce's Waterfall Model, Boehm's Spiral Model, and Forsberg and Moog's "Vee" Model [2]. Almost all of the SDLC models have some common steps that can be abtracted as follows: development begins with analysis of the problem, user needs, then building design of the system which is followed by implementation and verification, and finalized by retirement. The details and differences of these models will be introduced in the following chapters.

To obtain efficient systems, system's architecture is built to manage integrated design process of the system to prevent design conflicts and undesired solutions. Maier and Rechtin, defines systems architecting as a process driven by a client's purpose or purposes [3]. According to Muller, system architecting is a means to create systems that are efficient and effective, by supplying overview, by guarding consistency and integrity, and by balancing [4].

It has long been recognized that "architecture" has a strong influence over the life cycle of a system. However, the concepts of architecture have not been consistently defined [1], and applied within the life cycle of integrated systems. To built consistent, integrated, well-structured architectures, there is a need for defining the way of building architectures. For this purpose a number of methodologies have been proposed, such as Structured Systems Architecture Yourdan - Demacro, Hatley-Pirbhai, Ward – Mellor,

Harel, OMG groups Object-Oriented Systems Architecture, Activity Based Systems Architecture etc., in the last decades for the specification of systems. Hovewer there is not any approved or agreed common methodology for systems architecting in literature. Consequently, *there is certainly a strong need to conduct a comparative study between mostly recognized methodologies to evaluate their performances*. It is important to point out that, while conducting this research work, the following fact was observed: what systems architecture needs to be made up of is viewed differently by different methodologies.

This thesis attempts to provide a context to show what the systems architecture design process is, and what the role and use of the systems architecting design methodologies is in developing the appropriate systems architecture, and also to compare the advantages and disadvantages of the selected methodologies. Figure 1.1 shows the context of systems architecting design.

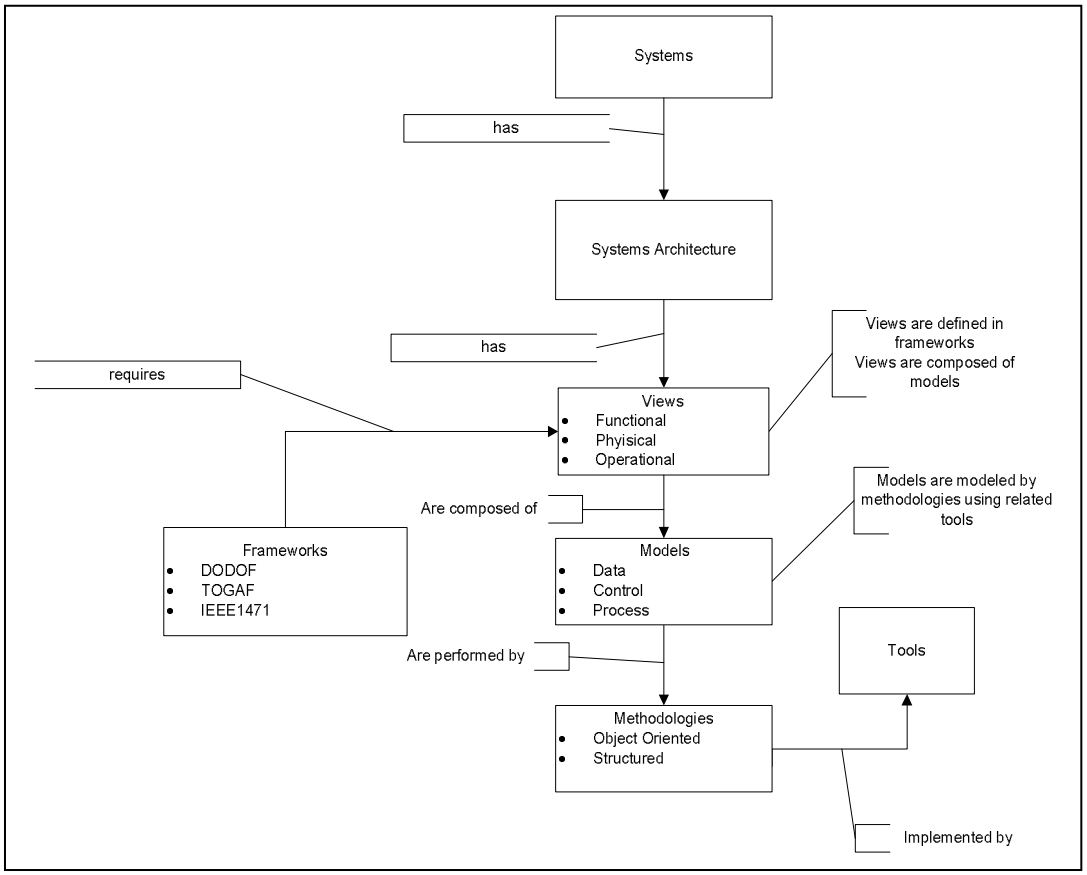

Figure 1.1. Systems architecture context diagram.

A number of articles as well as the current and popular textbooks on Systems Analysis and Design - which include but are not limited to those mentioned in the references (Rechtin and Maier 2000, Muller 2009, Yourdan 1989, Hatley 1989, Buede 2000, Brook, Stevens, Jackson and Arnold 1998, Kossiakoff and Sweet 2003, Wood 1989) – are surveyed during this study. It was realized that there was a considerable amount of discussion on the use of Object-Oriented Analysis and the Structured Architecture Methodology in literature. Moreover, according to the Department of Defence Architecture Framework, Structured Architecture Methodology and Object Oriented Methodology are the most widely used methodologies in military systems architecture design process. Due to this fact, the Structured Architecture Design and Object Oriented Architecture Design methodologies wll be compared and evaluated in this thesis.

Even though there are different types of systems (such as air traffic control systems, air logistics, computer networks, army defense systems, etc.) can be considered in a system architecture development study, the scope of this thesis is limited to military applications.

The approach used in this thesis begins with a literature survey of existing systems architecting methodologies, models, views and frameworks. The literature survey is then extended to relevant military and industrial standards that are being used for systems architecting and system design. The study is continued by examining the selected methodologies, and tools that are being used to implement these methodologies. furthermore, the key parameters for succesful integrated architectures are explored and used as criteria for evaluating the selected methodologies and to propose one of them to be employed in the case study given in the last chapter.

The plan of the thesis is as follows. In the first chapter, systems development life cycle models and systems architecting design process are introduced. In this part, important concepts used in systems architecture design are explained in order to avoid confusion. As it will be pointed out, in literature, multiple terms are used to describe a single concept in some cases, and different concepts are denoted by the same term in other cases. Here, the reader will be warned on these matters so that the rest of this study can be followed easily. In the next chapter, Structured Architecture Methodology and Object Oriented Architecture Methodology are introduced and explained. In addition, the key

parameters to be used in the selection of a system architecture are described, and a methodology is introduced – which is to be used for the selection of the appropriate methodology for the design process given in the case study. In the last chapter the study is concluded by architecting a communication system of an attack helicopter by using the chosen methodology - which is Structured Systems Architecture Methodology.

2. SYSTEM ARCHITECTURE AND ARCHITECTURE DEVELOPMENT

According to Maier and Rechtin, a system is a set of different elements so connected or related to perform a unique function not perfomable by the elements alone [3]. Kossiakoff and Sweet defines a system as a set of interrelated components working together toward some common objective [5]. Following to these definitions of systems it is obvious that every system consists of subsystems. In another saying, every system can be viewed as a part of another system, up to the whole universe. Thus it can be said that systems can be investigated in a level context where on the top the ultimate system is placed and every subsystem that come together to form its upper level system belongs to one level down. This context is shown on figure 2.1 below.

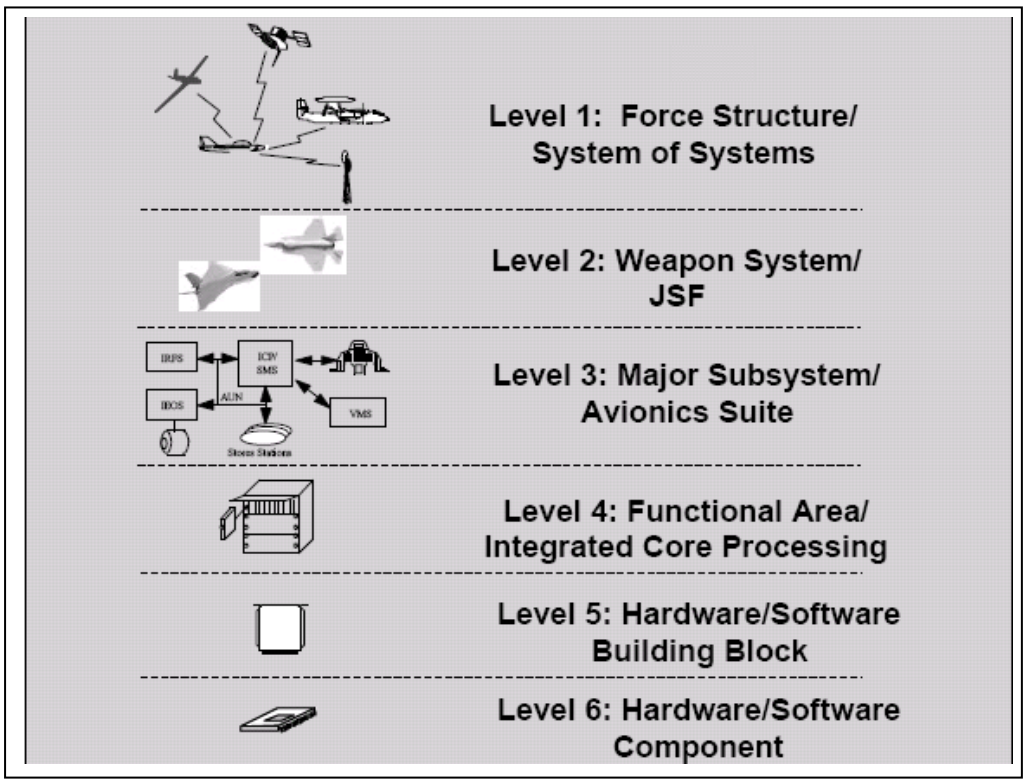

Figure 2.1. Systems level context [6].

Systems are not developed at a point in time. System development is a life cycle that includes analysis, design, implementation, integration, maintenance and retirement. And architecting contributes to the development of a system from its initial concept until its

retirement from use in this life cycle process. As such, architecting is best understood in a life cycle context, not simply as a single activity at one point in that life cycle [4]. In section 2.1 Waterfall, Spiral and Vee models which are some of the most important life cycle models that are widely being used in different system domains will be introduced.

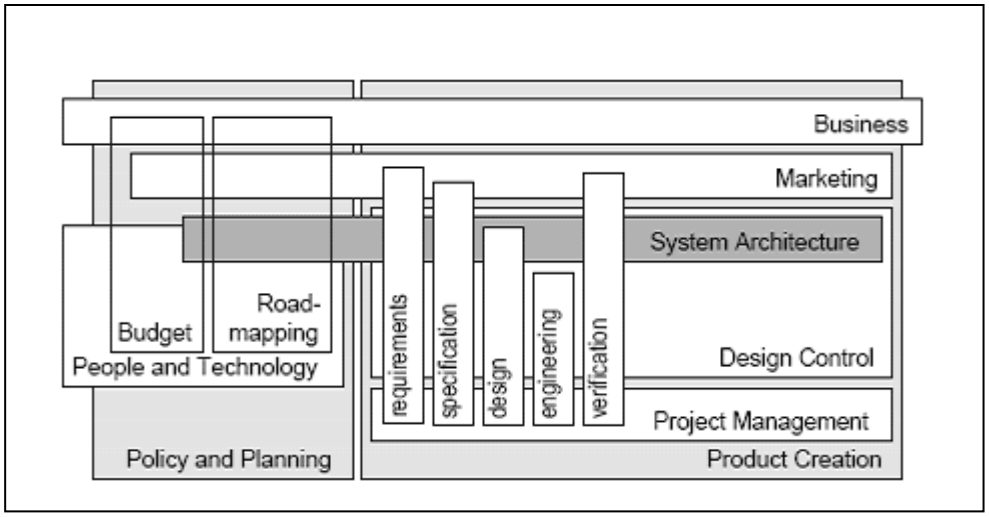

Figure 2.2. Systems architecture development life cycle [4].

As it is mentioned above, systems are built from subsystems and an external environment which it interacts with due to this fact there is a need for an organizing mechanism which provides the consistency of integration of these subsystems to form the required system. The need for system architecture development arose as a result of this need. Systems architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution [1]. Recommended Practice for Architectural Description of Software Intensive Systems also mentiones that system's architecture provide communication among the system stakeholders [1]. According to Stevens, Brook, Jackson and Arnold, architectural design defines clearly what is to be built [7]. When it is complete, each design component can be seen seperately by the group tasked to produce it, and so the design forms the basis for management of the implementation.

Carnegie Mellon University's Software Engineering Institute defines systems architecture as a representation of a system in which there is a mapping of functionality

onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and human interaction with these components.

The Open Architecture Framework defines an architecture as the most important, pervasive, top-level, strategic inventions, decisions, and their associated rationales about the overall structure (i.e., essential elements and their relationships) and associated characteristics and behavior [8].

IEEE standard glossary of software engineering terminology defines architecture as the structure of components, their relationships, and the principles and guidelines governing their design and evolution over time.

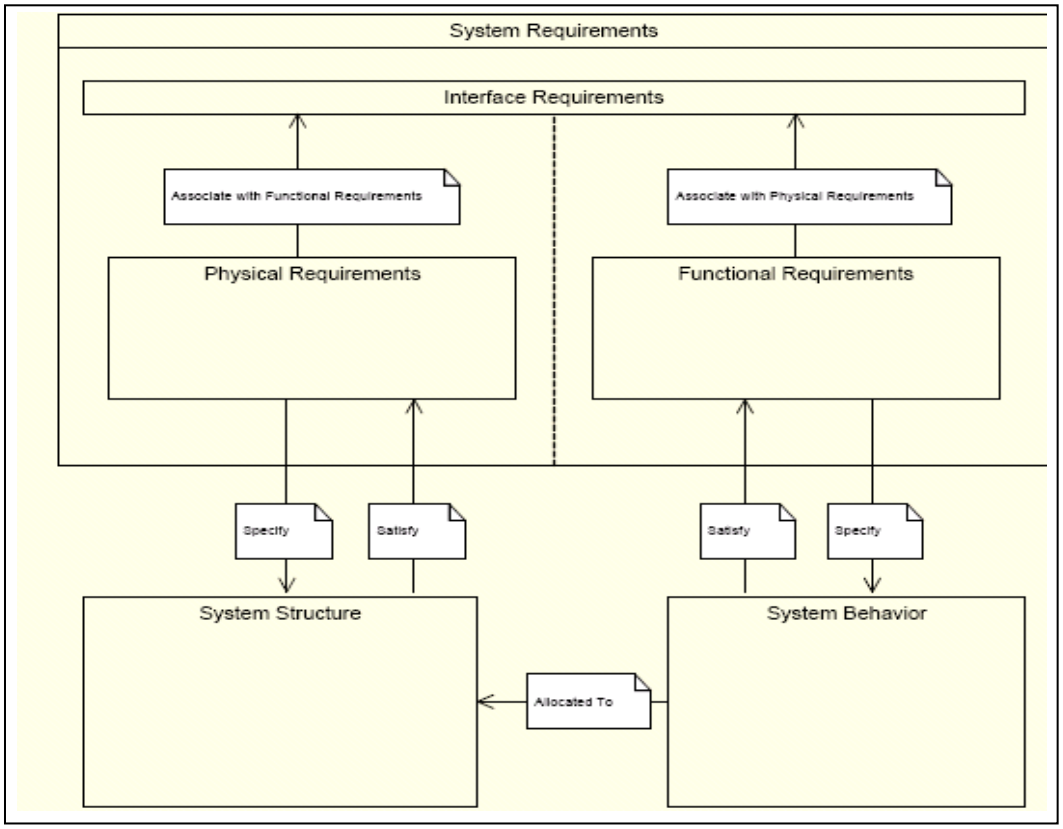

Figure 2.3. Systems architecture overwiev [9].

The present work, that is the research of the concepts of system's architecture design was found to be very challenging. The absence of agreed terminology made the work interesting but difficult; the study took longer time than it was anticipated. In almost each

reference different terminolgy is used to describe the same concept. For instance, even though the final or overall expectations from a systems architecture study is the same, behavioral or functional architectures (views), and structural model and physical architecture are used to describe the same thing in different references. The following sections are written in order to clarify these terms and concepts while the systems architecture design is explained.

2.1. SYSTEMS ENGINEERING LIFE CYCLE MODELS

The system development process is a complex effort which is an evolution of a particular new system from the time when a need for it is recognized and a feasible technical approach is identified though its development and introduction into operational use. The term "system life cycle" is commonly used to refer to the step-wise evolution of a new system from concept through development and on to production, operation and ultimate disposal. As the type of the work evolves from mainly analysis in the early conceptual phases, to engineering development and testing and then on to support of production and operational use [5].

The system development life cycle process begins with analysis of user needs to establish system requirements for the purposed system. Likewise, system requirements are analyzed to establish functional requirements, performance parameters, interface requirements and constraints which are the inputs for the architectural design process. A consensus seems to be present about the fact that requirements deal with the what and do not describe the how [4]. And also a good requirement can be explained as it should be specific, unambiguous, verifiable, quantifiable, measurable, complete, and traceable. The details of requirements analysis process and types of requirements will not be explained in detail here as this topic is out of the scope of this study.

A number of lifecycle development models have been created and applied to large-scale system and software development projects used in government, industry, and academia, but most are grounded in one of three seminal models. These are Royce's Waterfall Model, Boehm's Spiral Model, and Forsberg and Moog's "Vee" Model as Estefan cites [2]. These models are briefly explained in the following section.

2.1.1. Waterfall Model

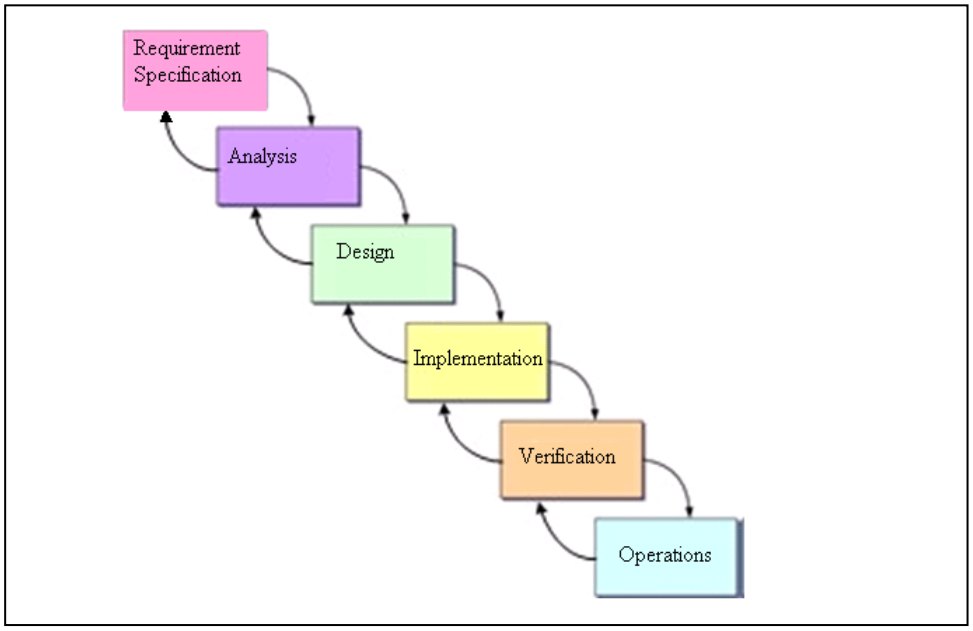

Figure 2.4. Waterfall model [2].

The waterfall model is a popular version of the systems development life cycle model. It is especially used in software engineering. It is considered as the classic approach to the systems development life cycle, the waterfall model describes a development method that is linear and sequential. Waterfall development has distinct goals for each phase of development. Once a phase of development is completed, the development proceeds to the next phase and feedback is very limited in waterfall.

The advantage of waterfall development is that it allows for managerial control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process, and theoretically, be delivered on time. Development moves from requirements specification, through design, implementation, verification (this stage includes testing, trouble shooting), and ends up at operation and maintenance. Each phase of development proceeds in strict order, with limited overlapping or iterative steps.

IEEE Standard for Application and Management of the Systems Engineering Process defines the sequential life cycle is a logical way of expressing many core concepts about

development. Hovewer in development of complex systems, it is almost not possible to chase straight forward path, each phase of the design may require feedback to its preeceding phases. So the disadvantage of waterfall development is that it does not allow for much reflection or revision. For instance an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the requirements analysis stage.

2.1.2. The Spiral Model

The Spiral Model, also known as the spiral lifecycle model, is a systems development lifecycle model which is mostly used in large, expensive, and complicated projects. This model of development combines the features of the prototyping model and the waterfall model.

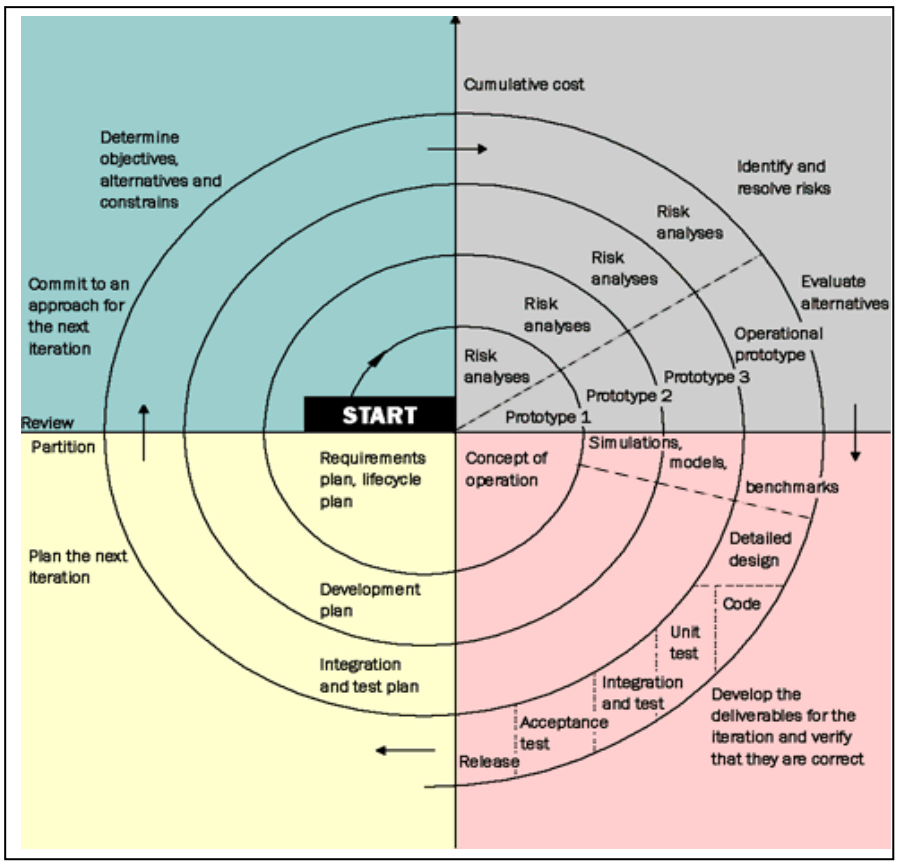

Figure 2.5. The spiral model [2].

The steps in the spiral model can be generalized as follows:

- The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.

- A preliminary design is created for the new system.

- A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.

- A second prototype is evolved by a fourfold procedure: Evaluating the first prototype in terms of its strengths, weaknesses, and risks; Defining the requirements of the second prototype; Planning and designing the second prototype; Constructing and testing the second prototype.

- At the customer's option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer's judgment, result in a less-than-satisfactory final product.

- The existing prototype is evaluated in the same manner as was the previous prototype, and, if necessary, another prototype is developed from it according to the fourfold procedure outlined above.

- The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.

- The final system is constructed, based on the refined prototype.

- The final system is thoroughly evaluated and tested. Routine maintenance is carried out on a continuing basis to prevent large-scale failures and to minimize downtime.

Stevens, Brook, Jakson and Arnold from Boehm 1986 that the major disadvantage or ambiguity that the spiral model is, it does not describe the criteria and issues which are drivers for the successive product prototypes, releases, and partial builds.

2.1.3. Vee Model

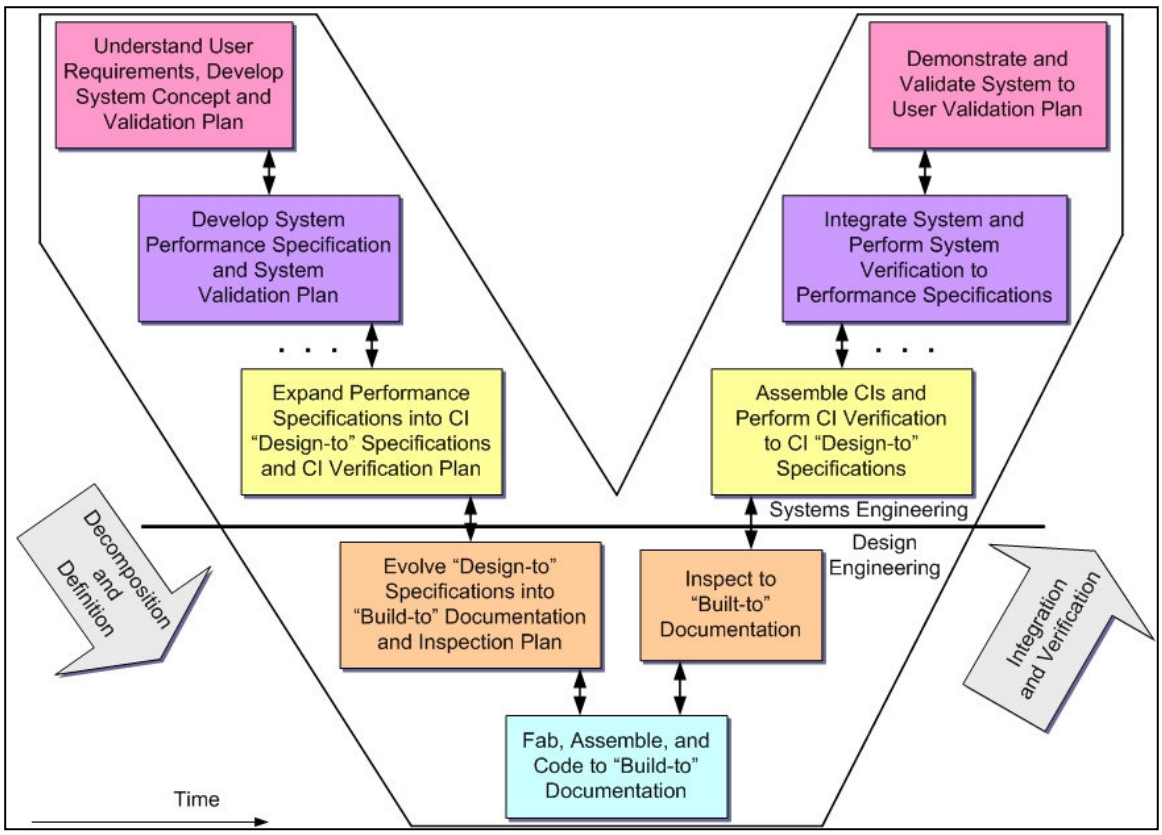

Figure 2.6. The Vee model [2].

Vee Model is developed by Forsberg and Mooz in 1992. It especially emphasizes the engineering activities during development process. Vee Model starts with the determination of the operational needs. It aims to transform the needs in to a detailed definiton of the system through a set of engineering activities. In this process operational needs are transformed into a system definiton documented by a system specification. Then follows item/subsystem definitons. In the following steps, this definiton process focuses on lower level system constituents. After each element of the sytem is defined, design, production, test and integration know-how is built [11].

2.2. ARCHITECTURE DESIGN PROCESS

The purpose of the Architectural Design Process is to synthesize a solution that satisfies system requirements. As was introduced in Section 2.1, there are different system life cycle models in literature; any of these life cycle models can be used to design a system. This decision purely belongs to the organization. Hovewer, there are some common concerns for the design of system's architecture for all of the life cycle models. These common concerns will be investigated in this section as system's architecture design process activities. The organizations could implement the following activities in accordance with applicable organization policies and procedures with respect to the architectural design process.

- Systems architecture design process begins with evaluation of the requirements, grouping and categorizing them. This includes identifying and defining derived requirements for describing functional and performance requirements, services and attributes, timeline requirements, data flow requirements, etc. [12]. The quality of a system architecture depends largely on the inputs provided to the architect.

- Developing the behavior model (functional model). The model that describes the functionality desired to be accomplished by the system. These models come together to form systems behavior (functional) view(architecture).

- Developing physical models (structure models) of the alternative sets of things, components to build the system. These models form the systems physical view (structural) view.

- Allocation of functions onto physical component to obtain operational architecture. The interface requirements are incorporated into the architectural design solution.

- Determine which system requirements are allocated to operators. This determination takes account of the context of use factors and considers, as a minimum, the following factors for the most effective, efficient and reliable human-machine interaction [12]. Yurtseven [13], expained in a private meeting that, in recent studies

Cognitive System Engineering, Joint Cognitive Systems Paradigm is revealed, which focuses on how the joint system performs as a whole for Human – Machine interface design process.

- Trade-off - selects among the alternative designs or architectures. Any design to be feasible must meet all of the performance requirements at system level. It is a key practice in the engineering of complex systems. One possible branch from this step is an iteration back to the beginning made necessary by no alternative design or architecture meeting the requirements. When this occurs, the steps are repeated to find feasible solutions, or requirements are relaxed so that a previous non-feasible solution is accepted, or the project is terminated for budget and schedule overrun, or simple impossibility [10].

- Maintain traceability between architectural design and system requirements [12]. This goal is achieved during the hole process. Figure 2.7. shows the flow diagram of systems architecting design process cited from [14].

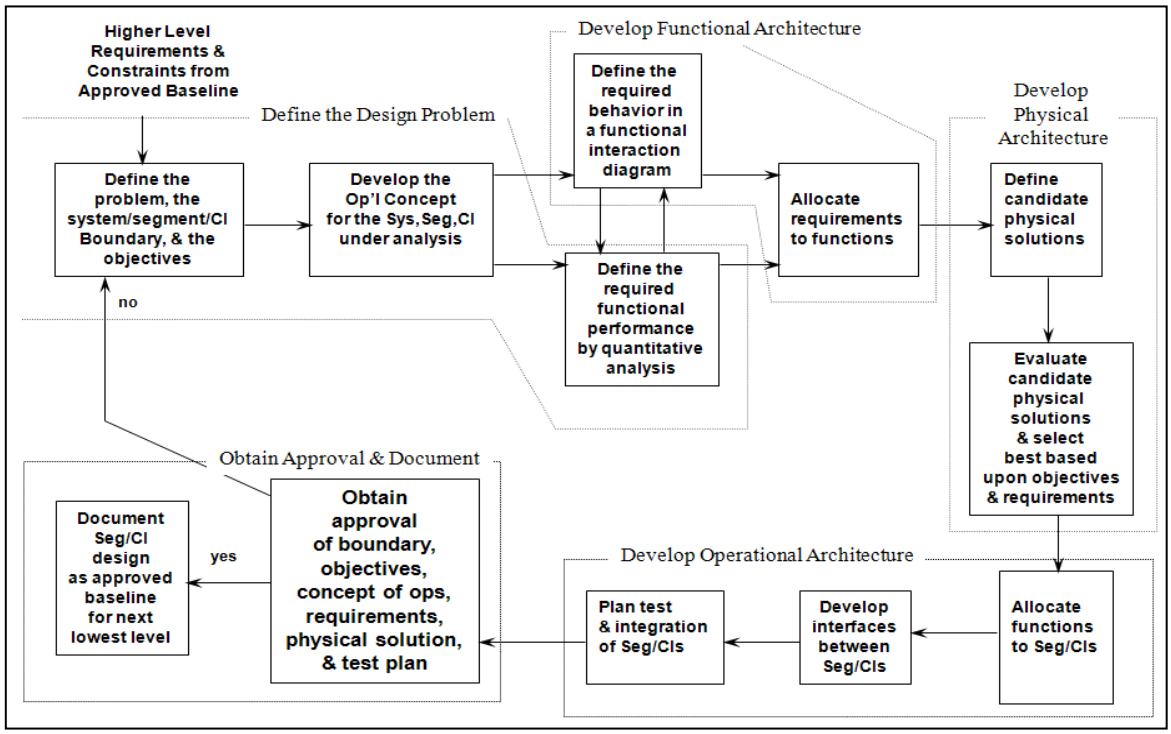

Figure 2.7. Systems architecting design process flow diagram [14].

While conducting the design process (as explained above), the content of the views may vary due to organizaional procedures and the framework that is used for the design. For instance in the USA, Depatment of Defence asks contractors to use DoDAF in order to provide communication between different contractors and in order to follow and control the design process in terms of their understanding.

Figure 2.8, shown below, which was given in the introduction section, shows how a system's architecture is composed of models which come together to form related views of the system's architecture. Models are developed by systems architecting methodologies and the framework requires views of the systems architecture. These concepts will be explained in detail in the following sections.

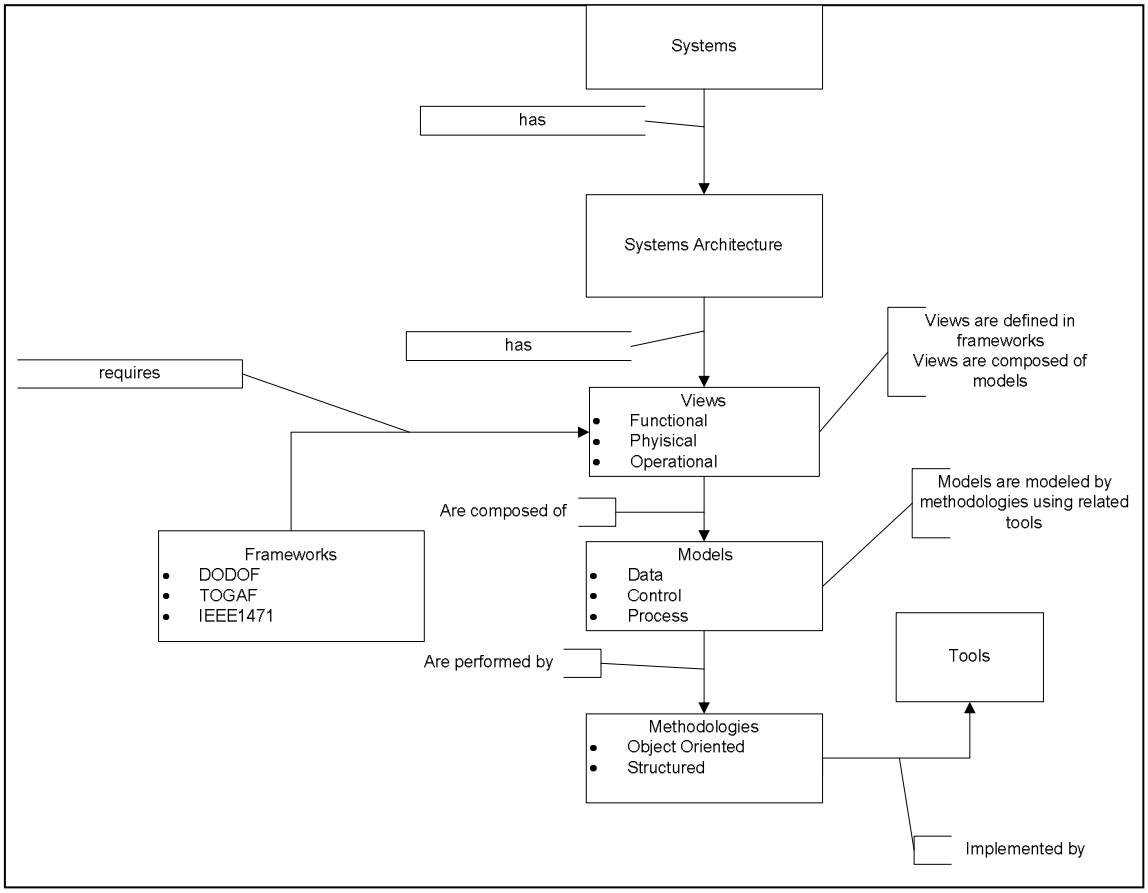

Figure 2.8. Systems architecture context diagram.

2.3. KEY PARAMETERS OF A SYSTEMS ARCHITECTURE

Most often, users judge the value of a product, system, or service by looking at its external interfaces and their function and form. They frequently treat the product/system as a black box for their use and for their value. Due to this fact a good architecture is required to meet the needs of the stakeholders (especially the users) to their satisfaction. Hovewer there is a very critical point that should be considered in the design. A succesfull design needs to satisfy set of known requirements, but in practice it can not meet every requirement.

Thus a systems architecture has some key parameters to fulfill an expected way of reflecting stakeholder concerns while it is enabling the affordability of the design. According to Muller, these parameters are as follows: consistency, integrated, simple and balanced [4]. Stevens, Brook, Jackson and Arnold, introduces key parameters as simple, well-structured, elegant, durable and relevant; and also the resultant product should be easy to use, as inexpensive as possible, and convenient to upgrade [7].

In this section these key parameters, elegancy and simplicity, balanced, consistent and integrated, tracibility and modularity will be explained.

2.3.1. Elegancy and Simplicity

Elegancy is being clean of unnecessary complexities. It can direct a builder to cost-effective structures that can be completed within a reasonable time frame, conceptually pleasing to all stakeholders, especially the user [7].

An enabling factor for an optimal result is simplicity of all technical aspects. Any unnecessary complexity is a risk for the final result and lowers the overall efficiency [7]. The system architecture should be as simple as possible without conflicting with other design principles. Architectures that are more complex than necessary will result in sub-optimal systems.

2.3.2. Balanced

The System Architecture is requred to be balanced amongst the goals of the system's external and internal requirements, short term needs and long term interests, efforts and risks from requirements to verification, value and costs [4]; such a balance is obtained by making trade-offs between the design alternatives.

2.3.3. Consistent & Integrated

It is the purpose of the System Architecture Process to maintain the consistency throughout the entire system, from roadmap and requirement to implementation and verification. On top of this consistency, the integrity in time must be ensured between the levels of systems an subsystems and external systems.

The true challenge for the architect is to design decompositions, that in the end will support an integration of components into a system. Most effort of the architect is concerned with the integrating concepts. How do multiple components work together? [15]. Decomposing the system continues until a level is found when the operations to be performed within a transformation need not be subdivided further. Whenever something is decomposed the resulting components will be decoupled by interfaces. The architect will invest time in interfaces, since these provide a convenient method to determine system structure and behavior, while separating the inside of these components from their external behavior.

2.3.4. Tracebility

The architecture required to provide tracebility between functions, components and requirements. Also tracebility between functions from high level to functions from sub levels need to be developed.

2.3.5. Modularity

One of the architect's roles is to ensure the best modularization of the system architecture, so as to allow for all the benefits of modularity: easier testing, easier accommodation of new requirements at the component level, and easier accommodation of new components at the system level [7].

2.4. SYSTEMS ARCHITECTURE VIEWS

A view is representation of a whole system from the perspective of a related set of concerns [1]. Maier and Rechtin defines view is a collection of models that share the property that they are relevant to the same concerns of a system stakeholder [3]. For example, a functional view collects the models that represent a systems functions. Moreover, they explain that the idea of view is needed because complex systems tend to have complex models and require a higher-level organizing element. Due to this fact views are composed of models.

In the introduction part of the thesis, it was mentioned that multiple terms have been introduced to describe a single concept and in some cases, and very different conepts are denoted by the same term in the literature. For instance, views, models and architecture are widely used to designate the same thing in literature, as mentioned earlier. In this thesis, the tem **view** will be used. This is consistent with the definition given Recommended Practice for Architectural Description of Software Intensive Systems, according to it, functional architecture, physical architecture, are frequently used informally. In the conceptual framework of the recommended practice, the approximate equivalents of these informal terms are accepted as functional view, physical view, respectively.

Architectures provide a description of how subsystems join together to form a system [16]. An integrated systems architecture consists of components, interfaces, interdependencies. Buede defines three architectures to form a complete architecture of a system [14]. These three architectures are functional architecture, which is the hierarchical model of the functions performed by the system, physical architecture, which is hierarchical description of the resources that comprise the system, and finally operational

architecture, which stands for complete description of the system design, including the functional archiecture allocated to physical architecture.

Stevens, Brook, Jakson and Arnold define three systems architecture descriptions (views) as follows: system structure which defines what the major components are, how they are organized and decomposed, their functionality and interfaces and the ties to the system requirements; system behaviour defines the dynamic response of the system to events, providing a basis for reasoning the system and the final is system layout which defines the physical arrangement, packaging and location aspects of design. Packaging adresses how components are allocated to physical resources such as the layout of a vehicle or how software is mapped to hardware. Packaging provides the basis for understanding the non-functional properties of the system such as weight, power consumption and performance. The layout of components adresses installation and environmental issues such as vibration or mutual interference, either withiin the system or between the system and its environment [7].

Estefan cites from Long, and gives three models that are necessary and sufficient to completely specify a system: (1) control (functional behavior) model, (2) interface (I/O) model, and (3) physical architecture (component) model [2].

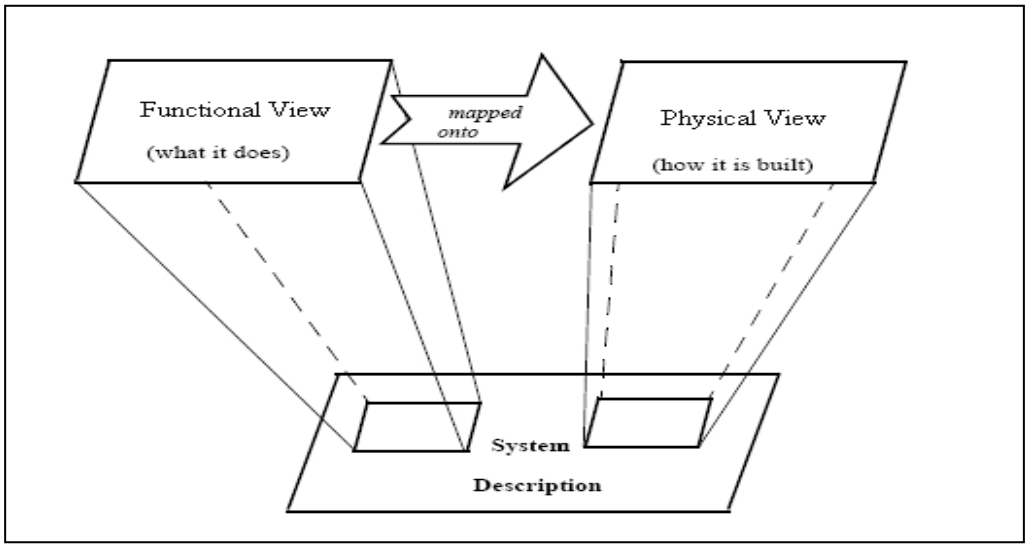

Figure 2.9. Architectural compositon of a system [10].

2.4.1. Physical View

The physical Architecture represents the partioning of physical resources available to perform the systems functions. A physical architecture subdivides the problem into manageable parts permitting and encouraging an iterative process, and providing excellent documentation [14].

The physical Architecture depicts the system product by showing how it is broken down into subsystems and components [16]. Since large complex systems are built from thousands or hundreds of thousands of parts, the models that form physical view are mostly required to developed hierarchically [10]. This hierarchy begins with the system and sytem's top level components and progress down to the configuration items that comprise each intermediate component. The configuration items can be hardware or sowftware elements or combinations of sowftware, hardware, people, facilities, procedures and documents [14].

Stevens, Brook, Jakson and Arnold use system structure term instead of the physical view term. According to him, structure defines what the major components are, their organization and interrelationships. Moreover the structure shows which components are to built, links the architecture back to the system functions and defines how the system is partitioned. It frames the design as a configuration of interacting components. Also, he introduces some ciritical principles for system structure. These principles can be summurized as follows: critical interfaces must not be seperated across the system, because they will infect non-critical elements; the design must always be product aware to maximize the chances of using off-the-shelf components, and also he influences on simplicity of the design.

Neglegting how we name the term, and looking for what has to be provided in the physical (structural) view, the part tree model that shows the physical connection of components in a hierarchical way and flow model that shows the data flow between the components of the system should be formed. Flow model defines the interfaces of the components. Moreover tracebility to requirements should be established.

2.4.2. Functional View

Buede defines need for functional architecture/view because engineering of systems has shown that the design process for a system has to consider more than the physical side of the system; the functions or activities that the system has to perform are critical element fot the design process to be successful on a consistent basis. Thus the design of functional and physical resources should proceed as providing checks on each other and complementing each other's progress [14].

A function is a transformation process that changes inputs into outputs [14]. The function tells something about the black box, but without prescribing how to realize it. To get the requirements more specific, all interfaces are identified; human interfaces as well as interfaces to other systems. Specifying only the functions is insufficient. The specification must also describe the desired quantified characteristics, such as how fast, how much, how large, how costly, etc [4].

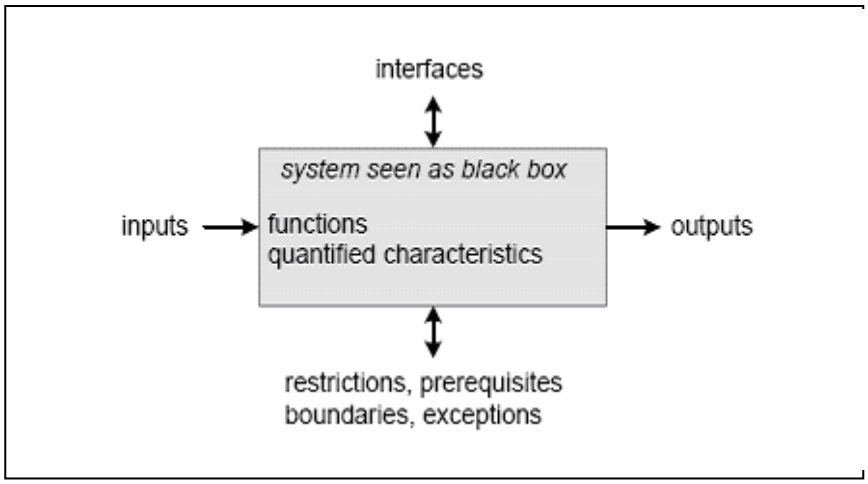

Figure 2.10. Overview of a function [4].

Mil-Std 499 Engineering Management defines function production process as "functions and sub-functions shall be developed in an iterative process system functions and subfunctions shall be progressively identified and analyzed as the basis for meeting system performance and design requirements [17].

A particular function might be assigned to a person, a machine, a slow computer, or a fast computer. The time to execute that function will depend upon the choice made [4].

The functional view/architecture of a system contains a hierarchical model of the functions performed It defines what the system must do, that is, describes the systems functions and the data flows between them. A system is modeled in functional view as having a single, top-level function that can be decomposed into a hierarcy of subfunctions [14].

Maier and Rechtin defines functional view as an arrangement of functions and their subfunctions and interfaces (internal and external) that defines the execution sequencing, conditions for control and data flow, and the performance requirements to satisfy the requirements baseline [3].

Keegan, Kelliher and Oliver use the behavior model term instead of functional architecture, and he defines behavior model as it captures what any thing, or object is to do. The model must capture all of the steps or functions involved in the behavior, how the functions are ordered, and all of the inputs and outputs of the functions. If the ordering of the functions allows alternative responses (paths) then the conditions for the alternative paths must be captured [10].

The primary difference between the behavior model of Keegan, Kelliher and Oliver and the functional architecture of Buede is control information of the functions. Control information can be explained as follows: the function is activated as soon as the resource for carrying out the function is available. When the appropriate triggering input arrives, the function is then ready to receive the input and begin transformation process. Buede defines functional architecture to have process and data models of the functions, he adds the control information on operational view of the system [14]. On the other hand Keegan, Kelliher and Oliver defines behavior model that has the process, data and control model of the functions [10].

This terminology difference can be summarized as follows: a systems functional view should contain three functional models to decribe the system funcitons in a proper

context. These are functional data model (information model) and functional control model and process model (function flow). These models are explained in system architecture models section. In these models the following can be used as the exit criteria for a functional architecture: coherent matching of input/output requirements with the functions and items in the functional architecture. Every input/output requirement should be traced to at least one function and one item in the functional architecture. In additon, every function associated with an external item in the functional architecture should have at least one input/output requirement traced to the function, as should every external item [14].

2.4.3. Operational View

The operational architrecture integerates the requirements decomposition with the functional and physical architectures. The process of developing operational architecture provides the raw materials for the definition of the system's external and internal interfaces and is the only activity in the design process that contains the material needed to model the systems performance and enable trade-off decisions [14].

Operational architecture covers the issues as allocation of functions to subsystems, trace non- input/output requirements and derive requirements, conduct performance and risk analysis, and document subsystem specifications.

- Allocate functions and system-wide requirements to physical subsystems
- Define and analyze functional activation and control structure
- Conduct performance and risk analysis
- Document architectures and obtain approval as an exit criteria
- Document sub-system specificaitons

The operational view is the mapping of functions to resources in a manner that is suitable for discrete – event simulation of the systems functions [14]. The design process proceeds several times, at decreasing levels of abstraction.

2.5. SYSTEMS ARCHITECTURE FRAMEWORKS

Architecture description standards are named as architecture frameworks. An architectural framework defines what products the architect must deliver and how those products must be constructed. The frameworks generally does not constrain the contents of any of those products, although such constraints could be incorporated [3].

An architecture framework specifies, information about architectures, thus a framework needs to distinguish what information is "architectural" as opposed to something else. Several architecture frameworks have been introduced to support enterprise and systems of systems modeling. The Zachmann framework, developed in the 1980's, and the Federal Enterprise Architecture Framework (FEAF) are used in selected industries. Military frameworks include the US Department of Defense Architecture Framework (DODAF), 12the UK Ministry of Defence Architecture Framework (MODAF) and the NATO Architecture Framework (NAF) [18] and IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.

The development of the DoD Architecture Framework and the earlier C4ISR Architecture Framework can be viewed as first generation attempts at creating a descriptive vocabulary for expressing architecture concepts and for creating structures for collecting and organizing data describing specific architectures [19].

2.5.1. Department of Defence Architecture Framework

DoDAF defines the need for architecture framework as "from a practical perspective, experience has demonstrated that the management of large organizations employing sophisticated systems and technologies in pursuit of joint missions demands a structured, repeatable method for evaluating investments and investment alternatives, as well as the ability to effectively implement organizational change, create new systems, and deploy

new technologies". Towards this end, the DoD Architecture Framework (DoDAF) was established as a guide for the development of architectures.

The DoD Architecture Framework specifies a set of "standard" views capturing various system perspectives. As with nearly all frameworks, the outline and contents are defined, but the methodology and support aids are left to the developmental organization's discretion. Many organizations implement processes that develop and manage the various DoDAF artifacts as independent deliverables leading to artifacts which are often inconsistent. Removing these inconsistencies occupies much of the time and resources at every stage of development. Failing to recognize inconsistencies leads to actual developmental, integration, and operational problems along with expensive retrofit efforts [21].

The goal of the DoDAF was to ensure that future military systems are interoperable and provide the warfighter with the support and effectiveness required for successful missions [21]. The DoDAF provides the guidance and rules for developing, representing, and understanding architectures based on a common denominator across DoD, Joint, and multinational boundaries [8].

Department of Defence Architectural Framwork defines a set of architectural products and views in three perspectives: Operational, System, and Technical Figure below shows the relationships between the DoDAF views.

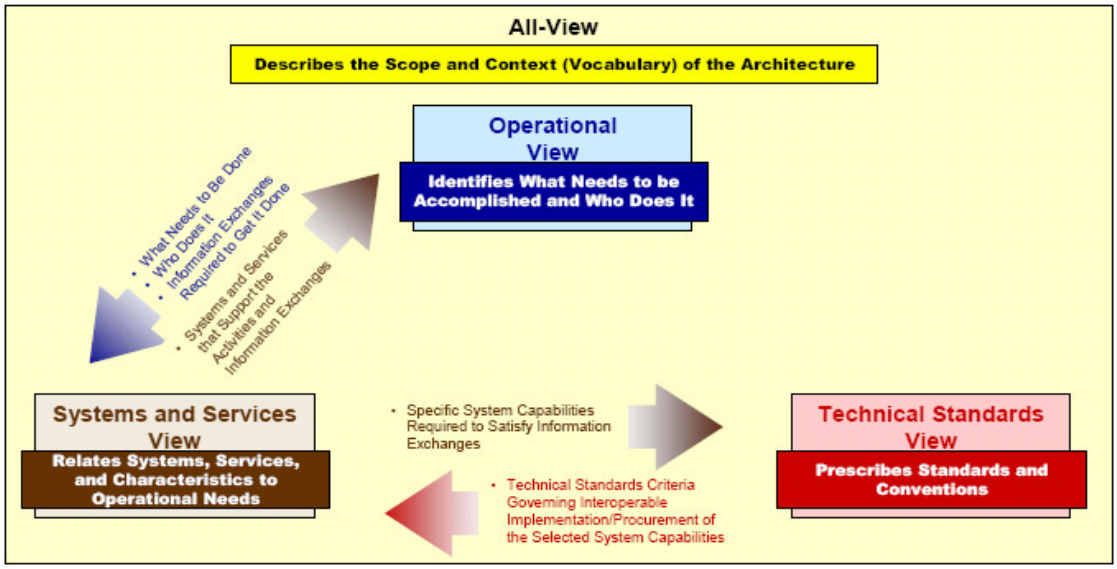

Figure 2.11. Relationship between DoDAF views [20].

Moreover, DoDAF provides direction on how to describe architectures and defines output products that are intended to provide a common basis for comparing and evaluating architectures [21].

2.5.2. IEEE 1471 Recommended Practice for Architectural Description of Software Intensive Systems

IEEE 1471 addresses the architectural description of software-intensive systems. IEEE 1471 is intended to reflect generally accepted trends in practices for architectural description and to provide a technical framework for further evolution in this area. Furthermore, it establishes a conceptual framework of concepts and terms of reference within which future developments in system architectural technology can be deployed. This recommended practice codifies those elements on which there is consensus; specifically the use of multiple views, reusable specifications for models within views, and the relation of architecture to system context.

IEEE 1471 addresses general expectations from a software architecture; it provides specific viewpoint explanations to be established. Structural viewpoint and behavioral viewpoint. In addition to these viewpoint examples, it tries to give explanations of the

expected deliverables of sofware architectures in general. The main difference of the IEEE 1471 is, it mentions viewpoint in addition to views. The following explanations are cited from the recommmended practice in order to distinguish the view and the viewpoint as mentioned in [1]:

"An architectural description is organized into one or more constituents called (architectural) views. Each view addresses one or more of the concerns of the system stakeholders. A view is a partial expression of a system's architecture with respect to a particular viewpoint."

"A viewpoint establishes the conventions by which a view is created, depicted and analyzed. In this way, a view conforms to a viewpoint. The viewpoint determines the languages (including notations, model, or product types) to be used to describe the view, and any associated modeling methods or analysis techniques to be applied to these representations of the view. These languages and techniques are used to yield results relevant to the concerns addressed by the viewpoint."

"An architectural description selects one or more viewpoints for use. The selection of viewpoints is typically based on consideration of the stakeholders to whom the architectural description is addressed and their concerns. A viewpoint definition may originate with an architectureal description, or it may have been defined elsewhere (a library viewpoint)."

"A view may consist of one or more architectural models. Each such architectural model is developed using the methods established by its associated architectural viewpoint. An architectural model may participate in more than one view."

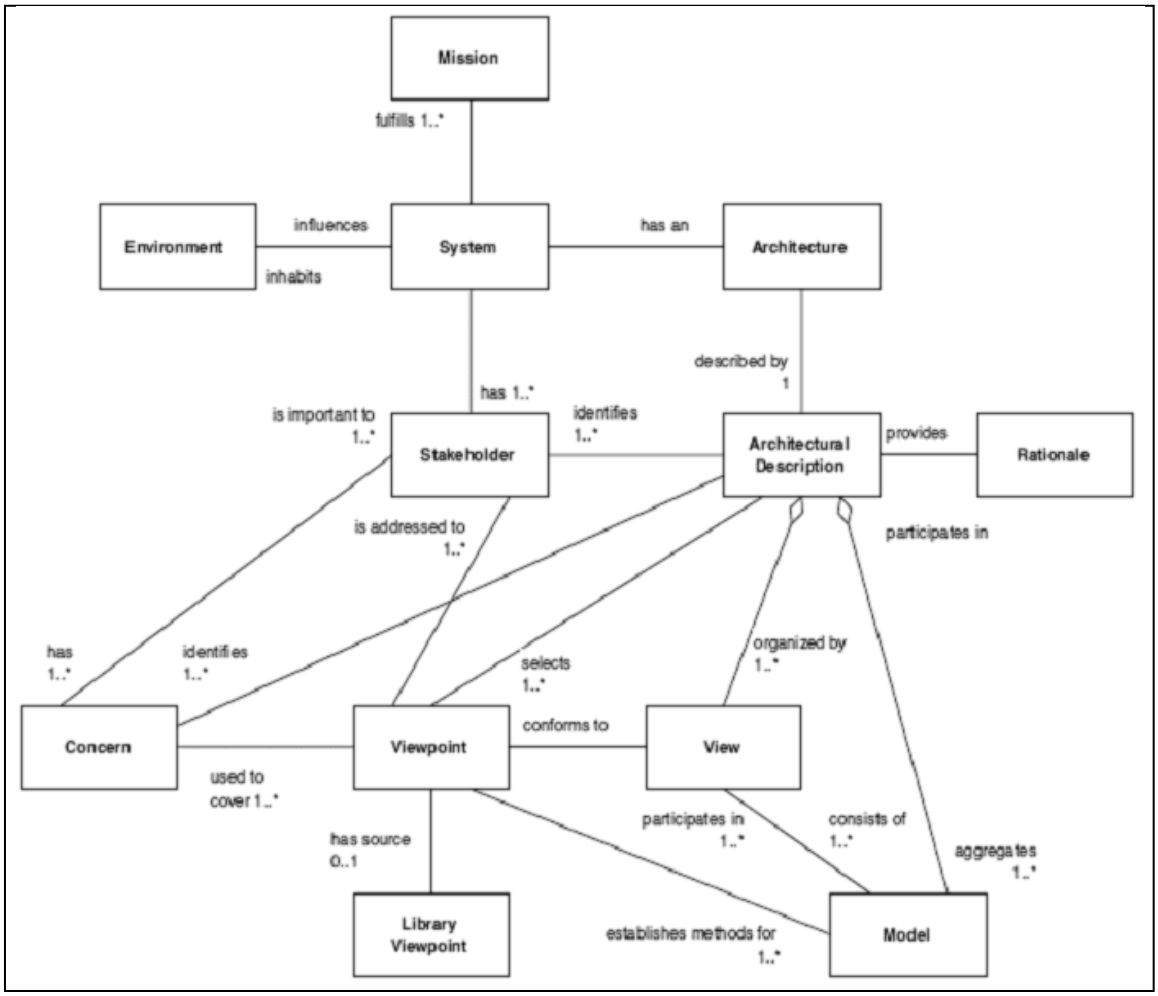

Figure 2.12. Conceptual model of architectural description in IEEE 1471:2000 [1].

2.5.3. ISO Reference Model of Open Distributed Processing

The Reference Model of Open Distributed Processing (RM-ODP) defines an architectural framework for distributed processing systems; systems “in which discrete components may be located in different places, or where communication between components may suffer delay or may fail.

The RM-ODP framework defines five view for specifying ODP systems. For each view, there is an associated viewpoint language that defines “the concepts and rules for specifying RM-ODP systems from the corresponding view.

- Enterprise viewpoint, explains the purpose, scope, and policies for an ODP system, roles played by the system, activities undertaken by the system, policy statements about the system

- Information viewpoint, explains the semantics of information and information processing in an ODP system

- Computational viewpoint, explains a functional decomposition of the system into objects that interact at interfaces

- Engineering viewpoint, explains the mechanisms and functions required to support distributed interaction between objects in the system.

- Technology viewpoint, captures the choice of technology in the system, how specifications are implemented, specification of relevant technologies, support for testing

2.6. SYSTEMS ARCHITECTURE MODELS

Models are the most important constituent of the systems architectures since architecture views are composed of related models, this section introduces the model terminology in the litereature, importance and benefits of models in systems architectures.

In the development of large complex systems there is substantial effort and engineering cost expended in assessing the large requirements documents that are made available. These efforts and costs can be reduced substantially with modeling. With modeling, the information is substantially condensed. A page of modeling is equivalent to five to ten pages of text. Furthermore, the models can be checked for correctness by engineers and tools, When models are used fully, text descriptions are not lost. Instead they are created as data dictionary items whenever a modeling element is created. This provides for traceability without having to create traceability for large volumes of text [10].

Models are abstraction of reality constructed for a purpose consisting of formal notations, building blocks, ways to model interfaces, interdependencies, and other relationships among the model components [22]. Ang, Nicholson and Mercer define models as they are architecture description products that are, graphical, textual, or tabular, for capturing and presenting a defined set of architecture description elements and their relationships in a visually consistent way [19].

Models can scale up to complex systems to analyze complex relationships and dependencies where complexity can be considered as a measure of how well knowledge of a system's component parts explains the system's behavior and also by the number of mutually interacting and interwoven parts, entities or agents [18]. To model complex systems architects first define the system concept model. As the concept is found satisfactory and feasible, the models progress to the detailed, technology specific models [3]. Therefore, every architecture design process should involve iteration: the process should be designed to be conducted over and over again until a satisfactory solution is reached.

In an another expression way: at the highest abstraction level, a system can be characterized by its core function and the key performance figure. Via multiple decomposition steps the description is detailed to units that can be engineered. The implementation shows orders of magnitude more details. The source description of today's products is in the order of millions lines of code [15].

Advantages of Model Based Development

- The models are composed of simple graphics thus they are really understandable by observers.

- Defects should be identified and eliminated as early as possible in the product development process [23].

- The models encourage completeness and avoidance of unnecessary content [24].

- The information is substantially condensed [10].

- Reduces impact of changes.

- Reduces cost of downstream activities (design, code).

- Model consistency through the modeling language and tools as opposed to PowerPoint engineering.

- Improved traceability between requirements and model elements.

Buede defines five views to capture hole system context, but the terminology used as view in his textbook actually refers to models as mentioned in this study. He cites from Karangelen, Hoang that for many systems five modeling views are critical for capturing the totality of a system: environment, data or information, process, behaviour, and implementation.

- The environmental view captures the system boundary, the operational concept, and the objectives of the system's performance.

- The data or information view adresses the relationships among the data elements that cross the system's boundary and those that are internal to the system; this view can be critical for information and software systems but incidental to mechanical systems.

- The process view examines the functionality of the system and is used to create the functional architecture.

- The behaviour view addresses the control structures in which the systems functions are embedded.

- The implemantation view examines the marriage of the physical architecture with the process and behaviour views; the operational architecture represents the implementation view.

Hatley proposes five models to capture required information of a systems architecture.

- Process Model - Models Functional Requirements
- Control Model - Models System Control
- Information Model - Models Data Relationships
- Architecture Flow Model - Models Data Flows Between Components
- Architecture Interconnect - Models Physical Connection of Components

After examining the views on literature, it is possible to classify models as follows: Models that are required to form functional view of a systems architecture are:

- Process model of the system. This model also named as functional model. It contains functional decomposition and functional ordering information of the system.
- Control model of the system. Where control mechanism of the system shown.
- Data model. This model also named as information model of the system. It contains input and output data information of the system.

Models that are required to form physical view of a system architecture are:

- Part tree model. Which contains physical connection of components.
- Flow model. This model contains data flow between components.

2.7. SYSTEMS ARCHITECTURE DESIGN METHODOLOGIES

The job of the architect is not only to drive ambiguity out of the system by defining the boundaries of the system and creating the concept of the system by allocating functionality and defining interfaces, but the architect need to be able to communicate these goals completely and clearly in the deliverables. For this reason, a common language is needed for continuous communication among team members throughout the developmental process.

The primary limiting factor in large system architecture development fundamentally results from inadequacies in the semantic foundations of architecture description—a knowledge deficiency. Because the first step toward reliable, mature practice in any discipline is the definition of the fundamental vocabulary, semantics, and models upon which the practice is built and shared [19].

A methodology is a particular implementation of a process. The steps in the process are specified in great detail and alternatives in the ordering of the work steps or in notation and views of information are removed and standardized. A methodology insures that a large number of workers performing the same process will do each step in the same way. On large projects, this is essential for intercommunication among the people and ability to perform the work reproducibly [10]. A methodology defines the fundamental vocabulary, semantics and models it is a recipe that stakeholders can speak about their system in common terms. It is a procedure for resolving the problems [26].

Structured Architecture Methodology is a well defined, widely used methodology in system's architecture design process. On the other hand Object Oriented Methodology has an increasing usage in systems architecting process. Object Oriented approach is originally a software development methodology. Advances in technology in recent years has been increasing the usage of software in systems. To improve system designes that are cascaded with software, the systems engineers adopted Object Oriented Methodology for designing systems to increase the communication betwen software engineers and systems enegineers to obtain better and faster results.

3. AN EVALUATION AND COMPARISON OF STRUCTURED ARCHITECTURE AND OBJECT ORIENTED METHODOLOGIES

3.1. STRUCTURED ARCHITECTURES AND CORE

During the research work for structured methodology, various types of structured methodology have been seen in literature. Some of the most famous ones are as follows: Yourdon/ DeMacro, Hatley / Pirbhai, Ward/Mellor, Harel, FFBD. All of these methods have a proven track record in the industry. In this thesis, FFBD has been since it is a well supported and widely used methodology by the program CORE. Moreover, by using CORE the products and views specified in the Department of Defense Architecture Framework can be generated via specialized view generators and/or queries to the CORE design repository. This ensures that the DoDAF views are consistent with each other as well as with the current system design.

Structured methodologies allow the analyst to break down complicated systems into smaller, clearly defined and more manageable parts. The Structured architecture is based on the concept of functional decomposition where the analyst breaks down the system into the basic processes that make it up and then breaks these down into smaller ones and so on until the analyst understands all the essential components of the system being investigated. Here the system is considered in its entirety where the analyst first tries to understand the key features of the system, ignoring the smaller details until later.

The high level description of a system is considerably simpler than describing the more detailed aspects of the lower-level system activities. This activity transforms the higher-order abstract functions the system must perform into more refined and detailed descriptions, or lower-level functions. The system-level performance requirements are decomposed as well and allocated to the lower-level functions. By adding control model, these functions may be sequenced or arranged such that a control viewpoint is established, input/output data dependencies established, and timing relationships determined.

In this section CORE, which uses the structured architecture methodology for designing system's architectures, will be explained. CORE uses functional flow block diagrams (FFBD) to obtain the functional or process model of the system; FFBD contains the sequence of the functions to be performed and control information for the functions. However it does not contain any data flow information. To obtain data flow information of the system's architecture, N2 charts are used. Moreover to combine these three models, another diagram called Enhanced Functional Flow Diagram is used to obtain process, control and data flow of the system in one diagram. To design the physical side of the system, Block Diagrams are used. These diagrams also show the electromechanical interfaces between systems components.

3.1.1. Function Flow Block Diagram

A FFBD shows the functions that a system is to perform and the order in which they are to be enabled (and performed). The order of performance is specified from the set of available control constructs. The control enablement of the first function is shown by the reference nodes which precede it, and the reference nodes at the end of the function logic indicate what functions are enabled next. The FFBD also shows completion criterion for functions as needed for specification. The FFBD does not contain any information relating to the flow of data between functions, and therefore does not represent any data triggering of functions. The FFBD only presents the control sequencing for the functions [27]. Figures below show the FFBD examples.

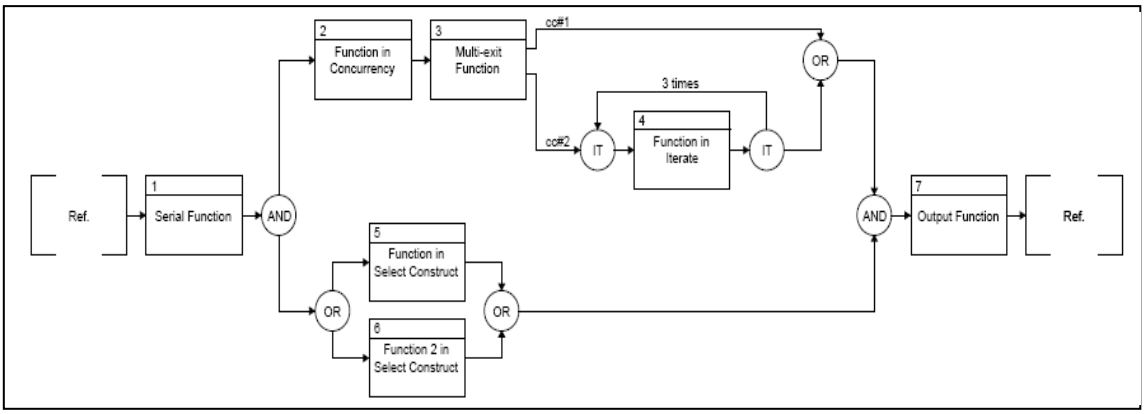

Figure 3.1. Functional flow block diagram example [27].

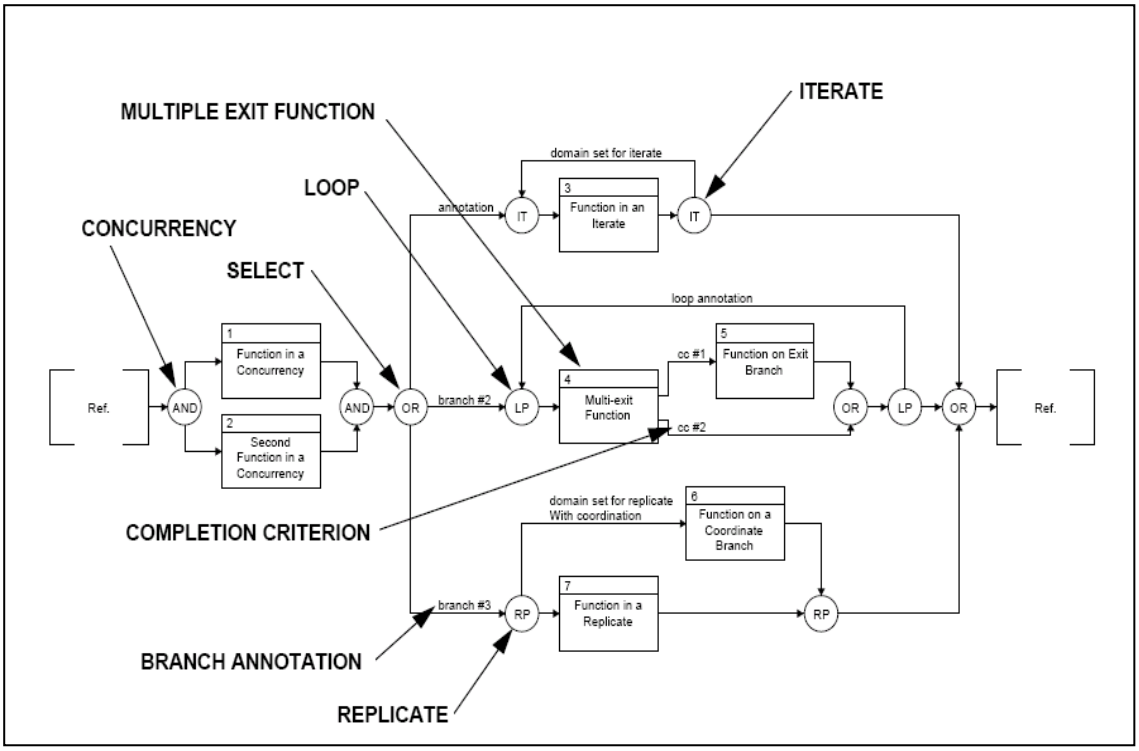

Figure 3.2. Functional flow block diagram example [27].

3.1.2. The N-Squared (N2) Chart

The N-Squared (N2) Chart shows and specifies interfaces between the elements of a system Long cites from Long and Lano [27] . The N-Squared (N2) Chart used to show the interfaces between the functions in a system, the N2 chart is equivalent to a Data Flow Diagram introduced by Yourdan; it contains all the information and differs only in with format from Data Flow Diagrams. The N2 chart is commonly used as a complement to the FFBD to provide the data flow information as inputs and outputs of the system functions.

The N2 Chart is structured by locating the functions on the diagonal, resulting in an N x N matrix for a set of N functions. For a given function, all outputs are located in the row of that function and all inputs are in the column of the function. If the functions are placed on the diagonal in the nominal order of execution, then data items located above the diagonal represent normal flowdown of data. Data items below the diagonal represent data item feedback. External inputs can optionally be shown in the row above the first function on the diagonal, and external outputs can be shown in the right-hand column. If desired, data repositories can be represented by placing them on the diagonal with the functions [27]. Figure 3.3 shows an example of an N squared Chart.

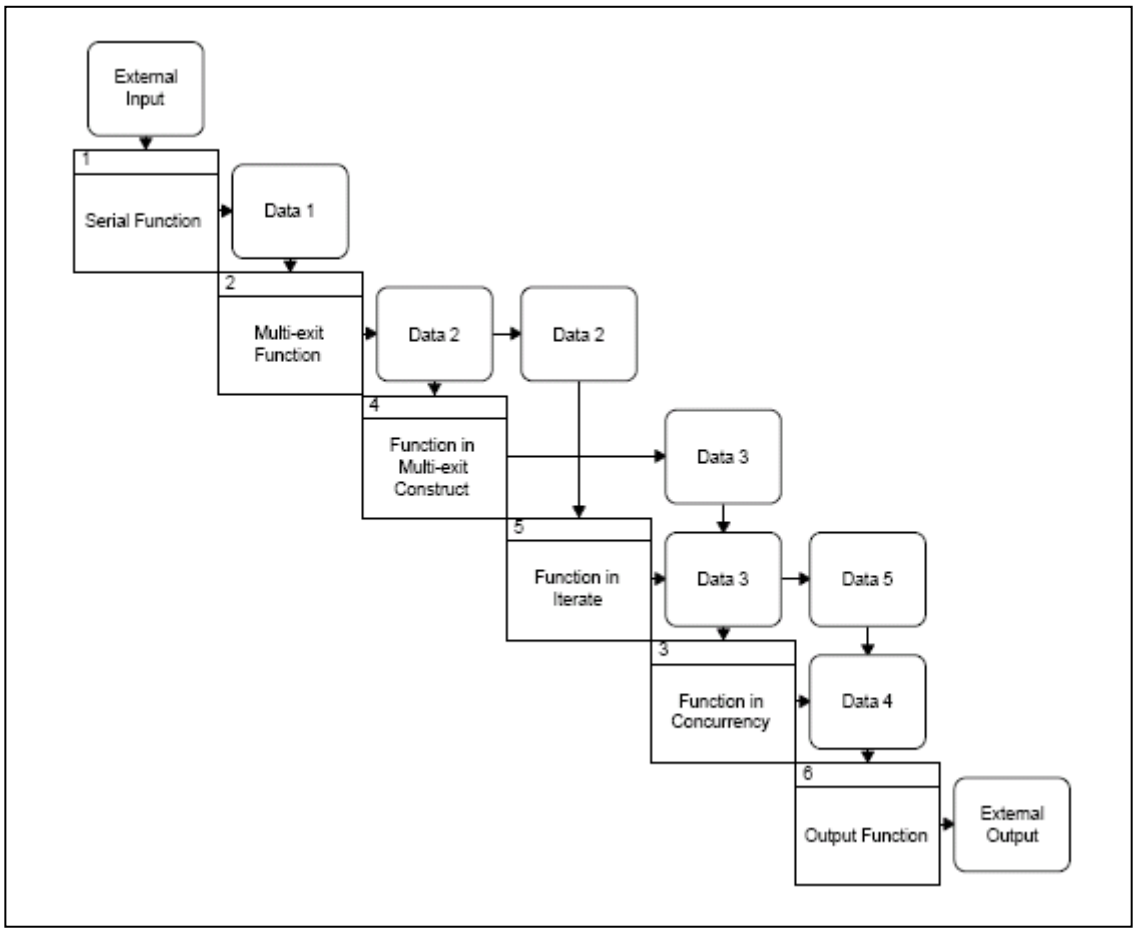

Figure 3.3. N Squared chart (N2) example [27].

3.1.3. Enhanced Functional Flow Block Diagram

The EFFBD displays the control dimension of the functional model in an FFBD format with a data flow overlay to effectively capture data dependencies. Thus, the Enhanced FFBD represents: (1) functions, (2) control flows, and (3) data flows. The logic constructs allow the designer to indicate the control structure and sequencing relationships of all functions accomplished by the system being analyzed and specified. When displaying the data flow as an overlay on the control flow, the EFFBD graphically distinguishes between triggering and non-triggering data inputs. Triggering data is required before a function can begin execution. Therefore, triggers are actually data items with control implications. In Figure 3.4, triggers are shown with green backgrounds and with

the double-headed arrows. Non-triggering data inputs are shown with gray backgrounds and with single-headed arrows.

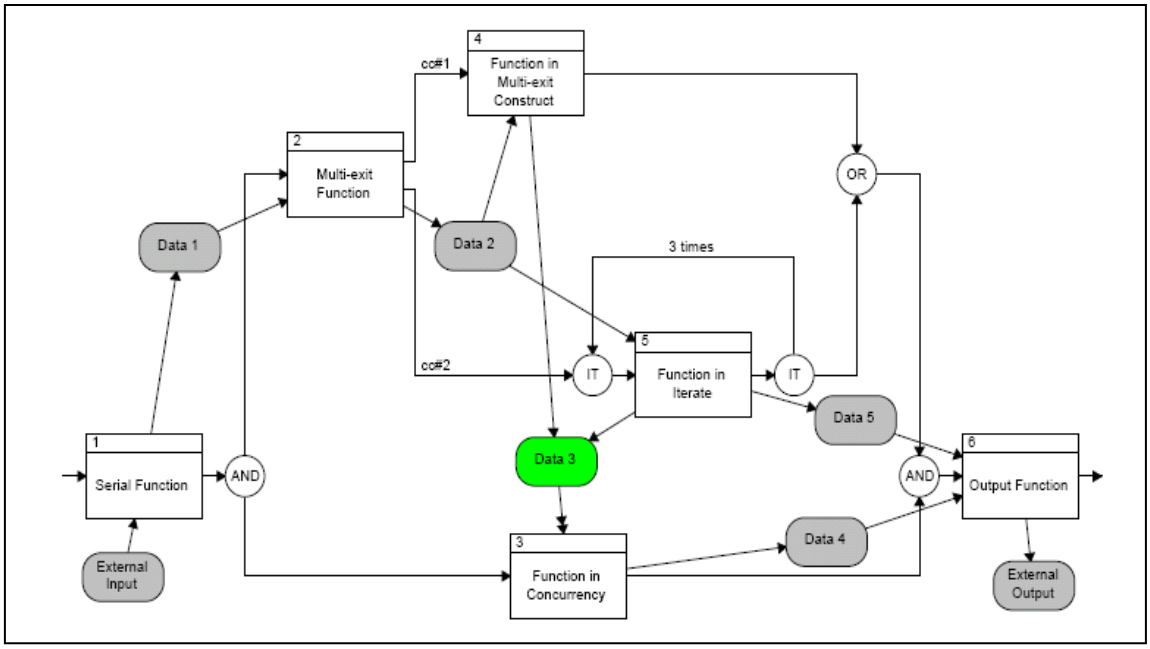

Figure 3.4. Example of an enhanced functional flow block diagram [27].

The Enhanced FFBD specification of a system is complete enough that it is executable as a discrete event model, providing the capability of dynamic, as well as static, validation. A fundamental rule in the interpretation of an EFFBD specification is that a function must be enabled (by completion of the functions preceding it in the control construct) and triggered (if any data input to it is identified as a trigger) before it can execute. This allows the engineer maximum freedom to use either control constructs or data triggers (or a combination of both) to specify execution conditions for individual system functions.

3.2. OBJECT- ORIENTED ARCHITECTURES AND SYSML

While structured methdology has its roots in both software and systems engineering, the Object Oriented (OO) design is a product of computer science and software systems engineering [14]. The Object-oriented design began in the late 60's as software programs became more and more complex. The idea behind the approach was to build software systems by modeling them, based on the real-world objects that they were trying to represent. In the last two decades it is being used for designing systems in general, too. It is not as widely used as Structured System's Architecture Methodology, but has an increasing usage for system's architecture design.

Objects are the real and conceptual things we find in the world around us. An object may be hardware, software, a concept e.g., velocity. Objects are complete entities, Software objects strive to capture as completely as possible the characteristics of the "real world" objects which they represent. Objects are "black boxes", their internal implementations are hidden from the outside world, and all interactions with an object take place via a well-defined interface. Object Oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis. The Object Oriented methodology uses an Object Oriented perspective rather than a functional perspective as in the structured architecture design methodology to design behavior of the system.

Object Oriented approach to system development is a collection of interacting objects that work together to accomplish tasks. Conceptually there are no separate processes or programs; there are no separate data entities or files. The system in operation consists of objects. An object is a thing in the computer system that is capable of responding to messages. Consequently, the Object Oriented methodology can be broken up into two major areas: Object Oriented analysis is concerned with developing an object-oriented model of the problem (application) domain. These identified objects represent entities, and possess relationships and methods that are necessary for the problem to be resolved. Object Oriented design is concerned with developing an object-oriented model of the system necessary to implement the specified requirements.

The major focus of the object model is object decomposition as opposed to functional decomposition, where a complex system is decomposed into several objects. An object-oriented system will consist of these various objects each of which will collaborate and cooperate with other objects to achieve specified tasks. Consequently, object decomposition allows the analyst to break down the problem into separate and more manageable parts. Objects do not stand alone. They work together in a cooperative manner to achieve the goals of the designer. Interconnection is the abstraction we use to think about how things (systems and objects) interrelate physically or logically.

The Object-Oriented System's Architecture Methodology integrates a top-down, model based approach that uses OMG SysML to support the specification, analysis, design, and verification of systems [2]. This section will introduce the foundation of SysML and the diagrams used in SysML to obtain systems architecture.

SysML supports the specification, analysis, design, and verification and validation of a broad range of complex systems. These systems may include hardware, software, information, processes, personnel, and facilities. The origins of the SysML initiative can be traced to a strategic decision by the International Council on Systems Engineering's (INCOSE) Model Driven Systems Design workgroup in January 2001 to customize the Unified Modeling Language (UML) for systems engineering applications. This resulted in a collaborative effort between INCOSE and the Object Management Group (OMG), which maintains the UML specification, to jointly charter the OMG Systems Engineering Domain Special Interest Group (SE DSIG) in July 2001 [28].

SysML reuses a subset of UML 2 and provides additional extensions needed to address the requirements for systems design that are not included in the UML. It is particularly effective in specifying requirements, structure, behavior, and allocations and constraints on system properties to support engineering analysis [28]. Figure 3.5 shows the structure of SysML. Structure diagrams that are shown below are used for designing the physical (structural) view of the system's architecture. They give the static model of the architecture. They are used to model the things that make up an architecture, classes, objects, interfaces and physical components. In addition, they are used to model the relationships and dependencies between elements. Behavior diagrams capture the varieties

of interaction and instantaneous states within a model as it executes over time; tracking how the system will act in a real-world environment, and observing the effects of an operation or event, including its results.

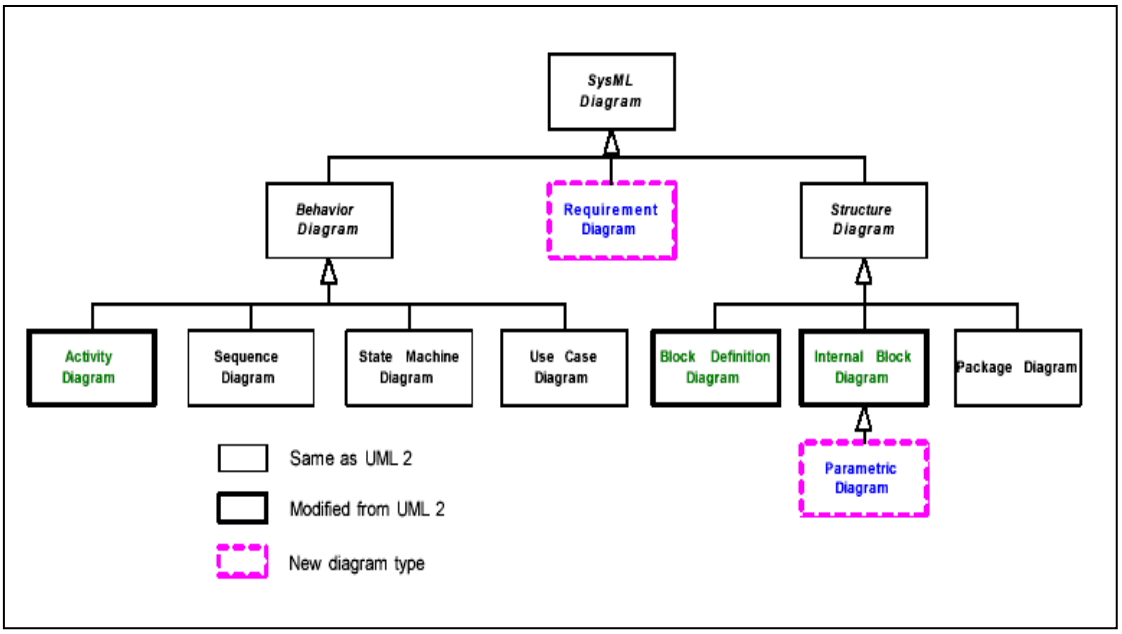

Figure 3.5. SysML diagrams [28].

3.2.1. Use Case Diagrams

A use case diagram is a set of scenarios that describe an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors.

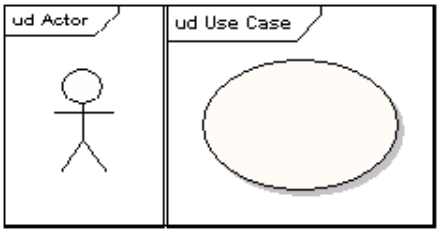

Figure 3.6. Shows the actor and the use case of a use case diagrams [29].

An actor represents a user or another system that will interact with the system. Actors represent roles which may include human users, external hardware or other systems. Use case is a single unit of meaningful work. It provides a high-level view of behavior observable to someone or something outside the system. Use case diagrams are used for capturing the functional requirements of a system. A use case typically Includes: Name and description, requirements, constraints, scenarios.

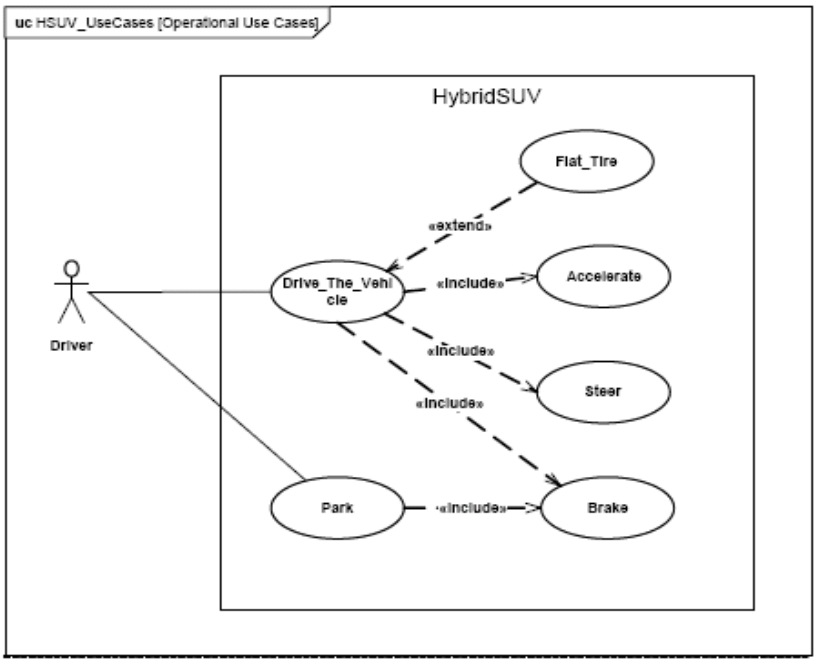

Figure 3.7. Use case diagram example [30].

3.2.2. Sequance Diagrams

Sequance diagrams describe shows objects as lifelines running, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline. Sequence diagrams are good at showing which objects communicate with which other objects; and what messages trigger those communications. Sequence diagrams are not intended for showing complex procedural logic. Typically a sequence diagram captures the behavior of a single scenario. Weakness of Sequence diagrams are, they are not good at to show looping and conditional behaviour. It is better to use activity diagrams to show control structure.

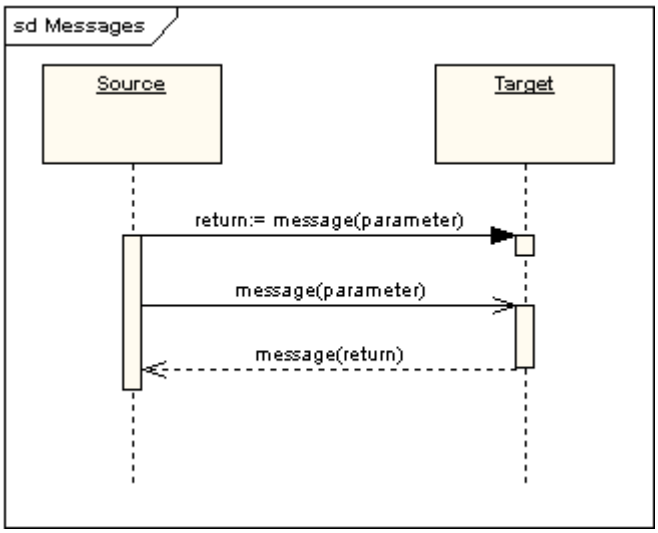

Figure 3.8. A view of a sequance diagram [29].

3.2.3. Activity Diagrams

Activity Diagrams are used to display the sequence of activities. Activity diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity [29]. Activity diagrams are similar to state diagrams because activities are the state of doing something. The activity diagrams describe the state of activities by showing the sequence of activities performed. Activity diagrams can show activities that are conditional or parallel.

Activity Diagrams are also useful for: analyzing a use case by describing what actions required to take place and when they should occur; describing a complicated sequential algorithm; and modeling applications with parallel processes.

However, activity diagrams should not take the place of state diagrams. Activity diagrams do not give detail about how objects behave or how objects collaborate.

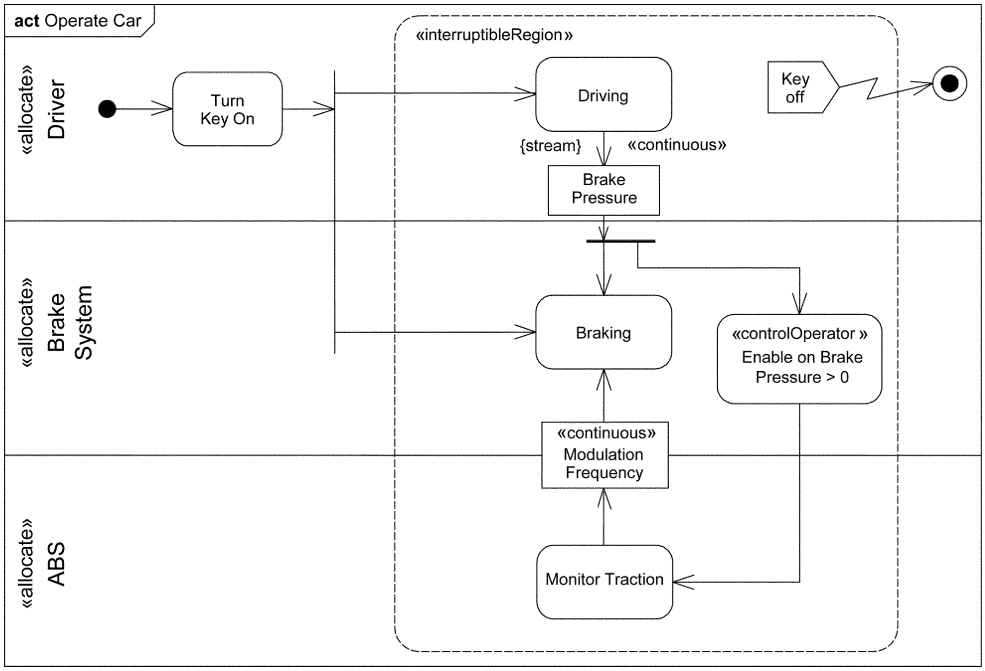

Figure 3.9. A view of an activity diagram [31].

3.2.4. State Machine Diagrams

A state machine diagram models the behaviour of a single object, specifying the sequence of events that an object goes through during its lifetime in response to events [29]. The state diagram shows the rules for the controller to change from state to state. These rules are in the form of transitions (the line that connects the states). The transition indicates a movement from one state to another. And each transition has a label that has three parts as; Trigger-[guard]/activities

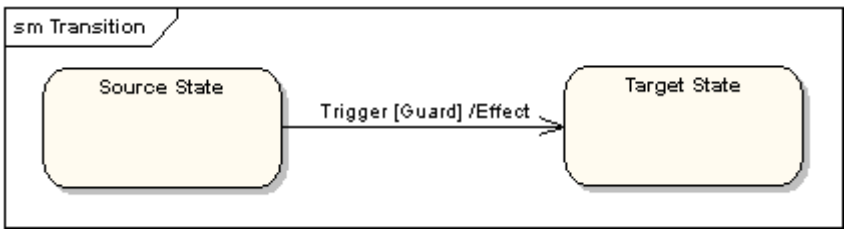

Figure 3.10. A view of an state machine diagram [29].

Where, Trigger is a single event that triggers a potential change of state. Guard is a Boolean condition that must be true for the transition to be taken however a transiton line do not have to have a Guard information. Activity is some behaviour that is executed during the transition.

3.2.5. Package Diagrams

Package diagram is used to organize the model. Package diagrams groups model elements into a name space, often represented in tool browser, it supports model configuration management [30].

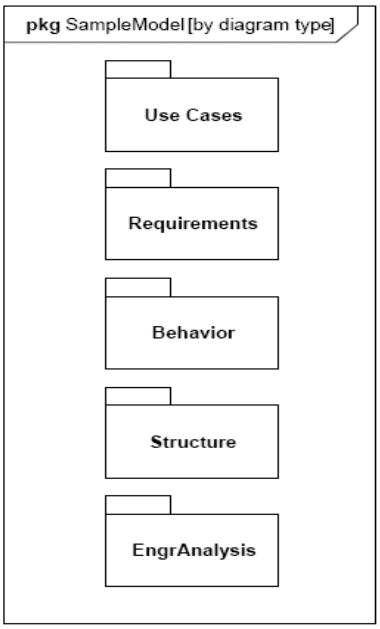

Figure 3.11. Package diagram example [30].

3.2.6. Parametric Daigrams

Parametric diagram is used to express constraints (equations) between value properties. Parametric diagarams provides support for engineering analysis (e.g., performance, reliability). It facilitates identification of critical performance properties. Parametric diagram represents the usage of the constraints in an analysis context. Binding of constraint parameters to value properties of blocks [30].

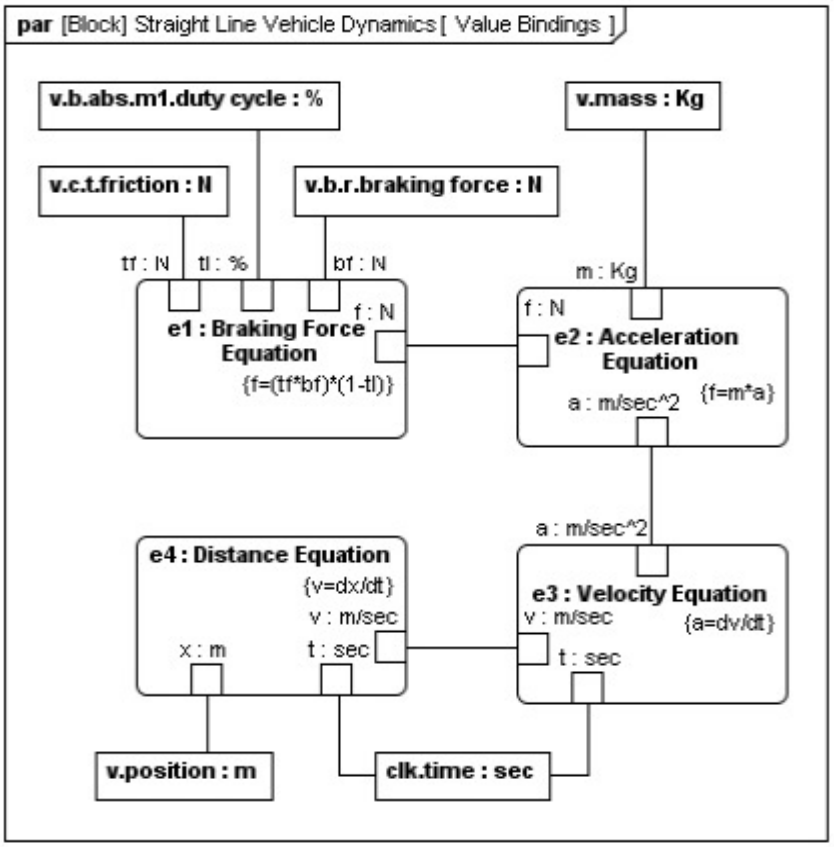

Figure 3.12. Parametric diagrams example [30].

3.2.7. Block Definition Diagram

Block Definition Diagrams shows the connections of objets of system.

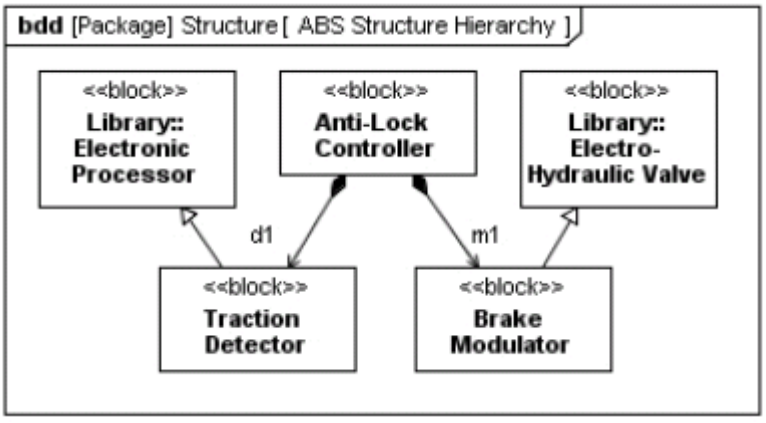

Figure 3.13. Block definition diagram [30].

3.2.8. Internal Block Diagram

Internal Block Diagram shows the inside of a block, that is defined in Block definition Diagram.

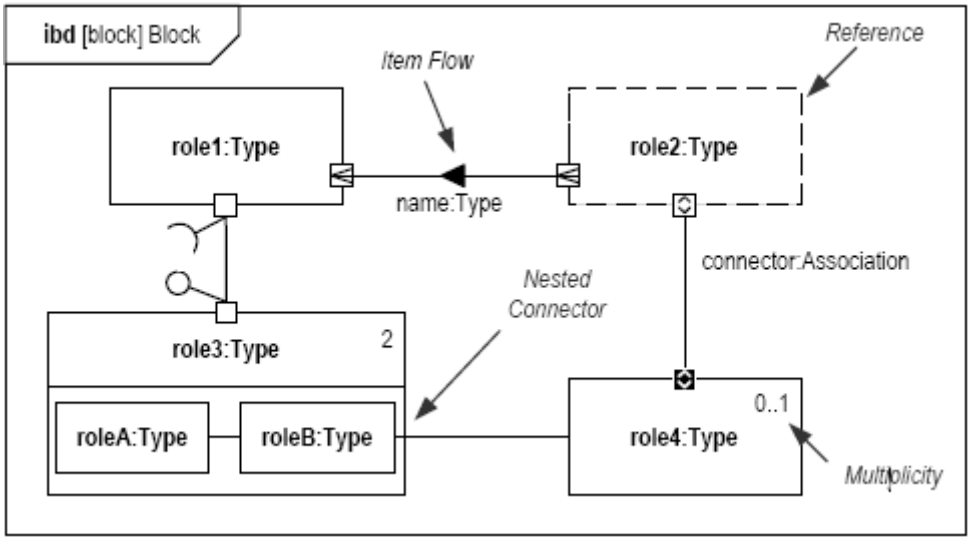

Figure 3.14. Internal block diagram [30].

4. AEROSPACE CASE STUDY: AN ATTACK HELICOPTER'S COMMUNICATION SYSTEM CONCEPTUAL DESIGN

4.1. GENERAL DESCRIPTION OF THE REQUIRED SYSTEM

This thesis introduces a case study of a communication system for an attack helicopter. In this thesis, the work will be concentrated on developing the requirements of the user, and derived system and functional requirements for these user requirements. Moreover, the function sequences that satisfy the system requirements of the communication system will also be developed. These function sequences are the primary driver of the functional architecture. And finally physical architecture of the communication system which contains components that perform functions of the system will be developed. The purpose of the case study is to provide demonstration of the system architecture develpment process, explained in this thesis, and usage of structured system architecture methodology.

One of the major problems that designers come across is when to stop in terms of level of details of the system's architecture; a system's architecture can be detailed as it is preferred. However, it must not be forgotten that the architectures are mediums that helps stakeholders of the system to the speak same language. In addition, architectures provides abstraction of detailed design issues. In this case study in order to not to lose the control of the system design, the conceptual design of the communication system will be performed. For this purpose, avionics systems physical view will be built as the first level of the architecture. Furthermore, communication system's first level and second level architectures will be built and the case will be finalised at that level; further detailed work is out of scope of this case study.

The user of the attack helicopter requires the communication system as follows: the communication system of the helicopter shall provide internal communication capability between pilot/copilot and external communication capability between aircrew and third parties. Also the system required to have encrypted external communication capability. The external communicataion need to be performed at HF, VHF or UHF bands.

Retransmission capability is also required. Finally, the system is required to have emergency transmitting capability which can be activated manually or automatically in crash conditions.

4.2. DECISION OF THE METHODOLOGY USED IN CASE STUDY

There are few significant eliminating factors that can be used to denote any methodology as the best. However, important factors can be presented as compatibility with life cycle goals, capability of the diagrams to form required models of the system, tool supportability, availability of training and personal expertise.

Both of the explained methodologies are widely used in systems design. However, Structured Methodology is more widely used in system's architecture development. On the other hand Object Oriented Methodology is excessively used for Software Systems Design. Also, it has been observed that Structured Architecture Methodology supports systems design completely when a certain level of abstraction is needed; this prevents design conflitcs due to complexity of the systems. For Object Oriented Methodology, that fact is a little bit different since Object Oriented Methodology is derived from Software Systems. It can be said that Object Oriented Methodology is more suitable for detailed design of the systems. Especially for the software systems, the models of Object Oriented Methodology can easily be transformed into software design models.

Rickman explains the advantages of Structured Methodology as follows: mature discipline for large systems design, better understood by developers and customers [23]. Maier and Rechtin explains that functional decomposition, communicates to stakeholders better than specification by use cases, contract requirements are specified in terms of functions [3], where Structured Architecture Methodology provides complete functional requirements model. Disadvantages of Structured Methodology are the following: it does not readily support commercial of the shelf reuse, since requirements developed top-down do not map well to reusable components. On the other hand Rickman explains advantages of Object Oriented Methodology as Object Oriented Methodology supports inclusion of COTS and Reusable Components, and disadvantages as Object Oriented Methodologies lacks of system functional model which can lead to missed requirements [23].

When we look to the tool supportability and availability of training, both Structured and Object Oriented Methodologies have tools that support the use of these methodologies. For instance in this thesis CORE is used for Structured Methodology which is a commertial tool that can be obtained for free for academic studies and can be bought for commercial works. Object Oriented Methodology has various tools available for use, too. But most of them are primarily focused on software development like UML. The examined language was SysML, which is supported by different commercial companies.

In terms of personel expertise, it is very important for the designers to use the methodology which they are best at it. Differences in modeling systems between these two methodologies is not sufficient to guide designers to make a choice. Moreover, it has been seen that, including Department of Defence Architectural Framework, official frameworks do not designate any of these two methodologies as the best. They leave the decision of chosing the "right" methodology to the designer. However, they ask for compatibility between the contractors designs, which can be maintained by using the same methodology for the same project.

In this thesis Structured Methodology and CORE is chosen in order to provide a sound conceptual architectural design of the communication system. The reasons for this choice are as follows: (1) the system designers are familiar with this methodology; this will help avoiding confusion and contradictions in the design process; (2) the tool availability; CORE is already available for the design work

4.3. REQUIREMENT ANALYSIS

In section 2.1 and 2.2. it has been mentioned that Systems Development and System's Architecture Development are performed in a Life Cycle Approach which begins with analysis of the user needs. Furthermore, this processs contuniues with transforming these user needs to Systems Requirements. System requirements are the basis for the architecture. Architecture is required to conform to System Requirements. So that, allocation of System's Requirements to Architecture should be provided. In order to inform stakeholders that each requirement is covered by the design.

4.3.1. User Requirements

It has already been mentioned in the introduction part of this case study section. The user of the attack helicopter requires the communication system which shall provide internal communication between pilot/copilot and external communication between aircrew and third parties. Also the system required to have encrypted external communication capability. The external communicataion need to be performed at HF, VHF or UHF bands. Retransmission capability is also required. Finally, the system is required to have emergency transmitting capability which can be activated manually or automatically in crash conditions.

User requirements for the Attack Helicopters Communication System are provided in Table 4.1. below. The table has four attributes, "ID", "Requirement Text", "Type", "Traceability". "ID" attribute is unique for each requirement, and it has a hierarchical structure. "Requirement Text" is the body of the User requirement, "Type" Attribute shows whether a requirement is "Heading" or a "Requirement". "Traceability" attribute establishes traceability to System Requirements.

ID	**Requirement Text**	**Type**
UR_1	Communication System	Heading
UR_1.1	Internal Communication System	Heading
UR_1.1.1	Communication system shall have internal communication capability between pilot and co-pilot.	Requirement
UR_1.2	External Communication System	Heading
UR_1.2.1	Communication system shall have external communication capability, between aircrew and third parties.	Requirement
UR_1.2.2	External communication system shall have HF (high frequency) communication capability.	Requirement
UR_1.2.3	External communication system shall have VHF (very high frequency) communication capability.	Requirement
UR_1.2.4	External communication system shall have UHF	Requirement

	(ultra high frequency) communication capability	
UR_1.2.5	Communication between aircrew and third parties shall be encrypted when selected by aircrew.	Requirement
UR_1.2.6	External communication system shall have total of four radios for HF, VHF, UHF communication.	Requirement
UR_1.2.7	External communication system shall have adequate means to allow pilot/copilot to select HF, VHF, UHF radios.	Requirement
UR_1.2.8	External communication system shall allow pilot/copilot to select two radios at a time.	Requirement
UR_1.2.9	Chosen radios shall be displayed to pilot/copilot.	Requirement
UR_1.3	Communication system shall be identical for pilot/copilot.	Requirement
UR_1.4	Volume of the selected radios and internal communication shall be adjusted by pilot/copilot.	Requirement
UR_1.5	Communication system shall have head microphones for pilot/copilot.	Requirement
UR_1.6	Communication system shall have retransmission capability.	Requirement
UR_1.7	Emergency Locator Transmitter (ELT)	Heading
UR_1.7.1	ELT shall be activated automatically or manually for the emergency landing.	Requirement
UR_1.7.2	ELT system shall be portable and integrated under the pilot seat.	Requirement
UR_1.8	Communication system shall receive Pilot/Copilot requests, as mentioned in Systems Requirements Document	Requirement
UR_1.9	Communication System shall provide feedback to Pilot/CoPilot, as mentioned in Systems Requirements Document	Requirement

Table 4.1. User requirements for communication system.

4.3.2. Systems Requirements

System requirements for the Attack Helicopters Communication System are provided in Table 4.2. below. System requirements are the transformation of User requirements into structured, clearly defined and verifiable requirements. The table below has four attributes, “ID”, “Requirement Text”, “Type”, “Traceability”. “ID” attribute is unique for each requirement, and it has a hierarchical structure. “Requirement Text” is the body of the System requirements, “Type” Attribute shows whether a requirement is “Heading” or a “Requirement”. “Traceability” attribute establishes traceability to User Requirements.

ID	**Requirement Text**	**Type**	**Tracea-bility**
SR_1	Communication System	Heading	UR_1
SR_1.1	Internal Communication System	Heading	UR_1.1
SR_1.1.1	Internal communication System shall have two working modes	Requirement	UR_1.1.1
SR_1.1.1.1	Mode_1 shall be full duplex mode,	Requirement	UR_1.1.1
SR_1.1.1.2	Mode_2 shall be half duplex mode	Requirement	UR_1.1.1
SR_1.1.2	Internal Communication System shall receive communication mode information from Pilot/CoPilot	Requirement	UR_1.1.1 UR_1.8
SR_1.1.3	Volume of the internal communication shall be adjusted by pilot/copilot from communication system control and display unit..	Requirement	UR_1.4 UR_1.8
SR_1.2	External Communication System	Heading	UR_1.2 UR_1.2.1
SR_1.2.1	External communication system shall have ON/OFF switch to enable Radio usage.	Requirement	UR_1.2
SR_1.2.2	Initial Radio frequency shall be 243 MHZ when ON/OFF switch is turned ON.	Requirement	UR_1.2
SR_1.2.3	HF Radio	Heading	UR_1.2.2

SR_1.2.3.1	External communication system shall have HF (high frequency) communication capability with FM/AM modulations in the military communication frequency bands.	Requirement	UR_1.2.2
SR_1.2.3.2	External communication system shall have at least one HF (high frequency) radio.	Requirement	UR_1.2.6
SR_1.2.4	V/UHF Radio	Heading	UR_1.2.3 UR_1.2.4
SR_1.2.4.1	External communication system shall have V/UHF (very/ultra high frequency) communication capability with FM/AM modulations in the military communication frequency bands..	Requirement	UR_1.2.3 UR_1.2.4
SR_1.2.4.2	External communication system shall have at least 3 V/UHF radios.	Requirement	UR_1.2.6
SR_1.2.5	Encryption	Heading	UR_1.2.5
SR_1.2.5.1	External Communication system shall have encryption mode for all of the radios.	Requirement	UR_1.2.5
SR_1.2.5.2	Encryption mode shall be enabled/disabled by Pilot/CoPilot.	Requirement	UR_1.2.5
SR_1.2.5.3	Encryption code shall be zeroized by Pilot/CoPilot under emergency conditions.	Requirement	UR_1.2.5
SR_1.2.5.4	Encryption enabled/disabled information will be shown to the Pilot/CoPilot	Requirement	UR_1.2.5
SR_1.2.6	Radio Selection and Display	Heading	UR_1.2.7 UR_1.8 UR_1.9
SR_1.2.6.1	External communication system shall allow pilot/copilot to select HF, V/UHF radios from cyclic controller.	Requirement	UR_1.2.7
SR_1.2.6.2	External communication system shall allow pilot/copilot to select HF, VHF, UHF	Requirement	UR_1.2.7

	radios from communication system control and display unit.		
SR_1.2.6.3	Each Pilot/CoPilot shall be able to select one radio at a time.	Requirement	UR_1.8
SR_1.2.6.4	Radio frequencies shall be adjusted by Pilot/CoPilot.	Requirement	UR_1.8
SR_1.2.6.5	Volume of the selected radios shall be adjusted by pilot/copilot from communication system control and display unit.	Requirement	UR_1.4 UR_1.8
SR_1.2.6.6	FM/AM modulation shall be selected by Pilot/Copilot.	Requirement	UR_1.8
SR_1.2.6.7	Chosen radios shall be displayed to pilot/copilot by communication system control and display unit.	Requirement	UR_1.2.9 UR_1.9
SR_1.2.6.8	Adjusted frequencies shall be displayed to pilot/copilot by communication system control and display unit.	Requirement	UR_1.9
SR_1.2.6.9	Chosen modulation method shall be displayed to pilot/copilot by communication system control and display unit.	Requirement	UR_1.9
SR_1.2.7	Antennas		UR_1.2
SR_1.2.7.1	Each Radio shall have one antenna placed on the helicopter.		UR_1.2
SR_1.3	Communication system shall be identical for pilot/copilot.	Requirement	UR_1.3
SR_1.4	Communication system shall have head microphones for pilot/copilot.	Requirement	UR_1.5
SR_1.5	Communication System shall provide retransmission services between to other external communicating parties.	Requirement	UR_1.6

SR_1.6	Emergency Locator Transmitter (ELT)	Heading	UR_1.7
SR_1.6.1	ELT shall be activated automatically or manually for the emergency landing.	Requirement	UR_1.71
SR_1.6.2	ELT system shall be portable and integrated under the pilot seat.	Requirement	UR_1.7.2
SR_1.7	Communication System will be energized when the master avionics switch is ON.		UR_1
SR_1.8	Communication System power supply shall be compatible with MIL-STD-704	Requirement	UR_1

Table 4.2. System requirements for communication system.

4.4. ARCHITECTURE

4.4.1. Physical View

The Physical Hierarcy Model (Part Tree Model) which establishes the Pysical View of the System's Architecture is shown figure below. This diagram shows the highest level Physical Hierarchy of the Communication System. As it can be seen from the Figure below, The Commmunicaiton System has Communication System Component on top of the Pyhsical Hierarcy Model. Below that the composing components are Communication Controller and Distributor Sub-System, External Communication Sub-Sysytem, Emergency Location Transmitter Component, Communication System Control And Display Unit, Headphones, Microphones.

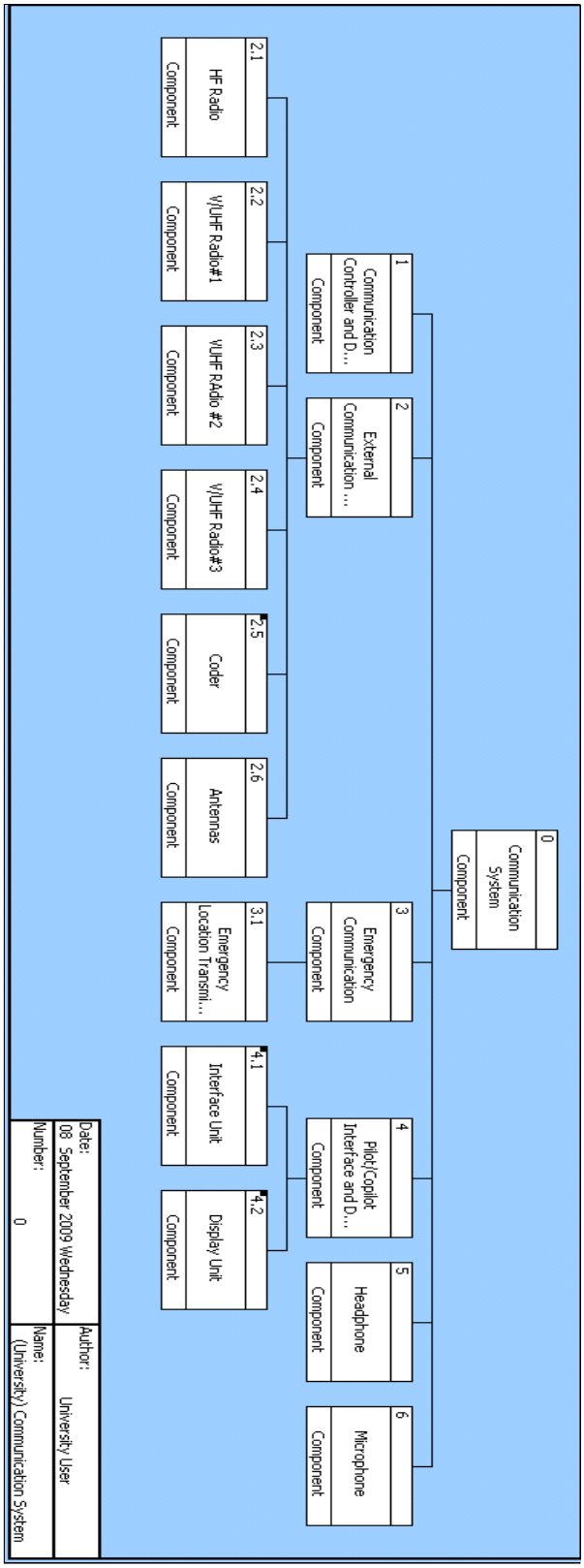

Figure 4.1. High level physical hieracy model of the communication system.

Figure 4.2. below, shows the Flow Model of the communication system.

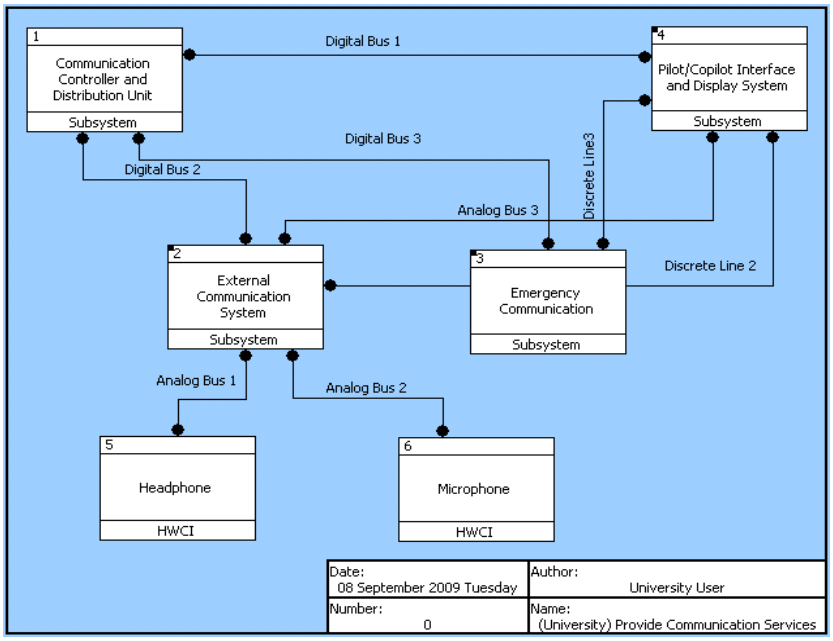

Figure 4.2. Physical flow model of the communication system.

It has already been explained that system's are composed of several levels. Through lower levels details are increasing. Figure below shows the details of "External Communication Component" of Figure 4.1. It shows the components of "External Communication Sub-System"

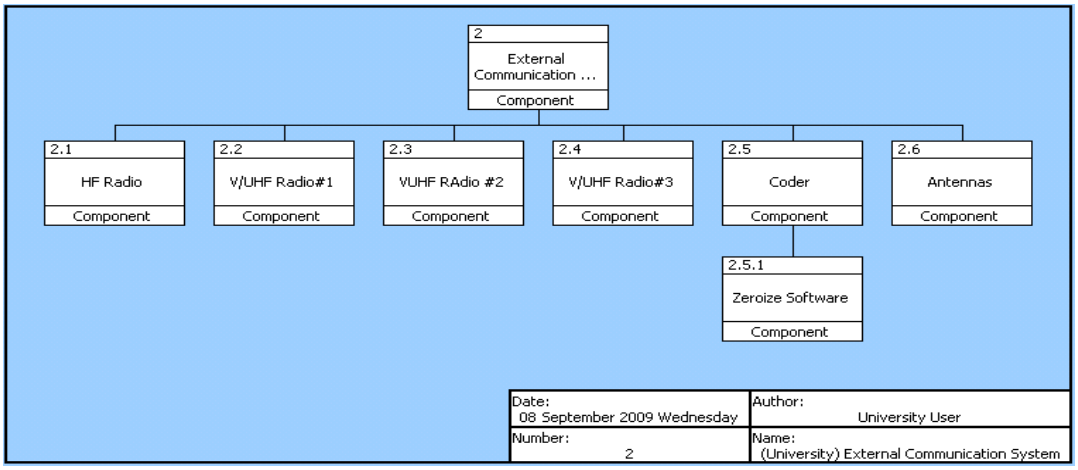

Figure 4.3. Physical hieracy model of the external communication sub-system.

4.4.2. Functional View

Figure 4.4. below shows the first level functions and their Process Model for the Communication System. This model contains the high level functions which performs multiple low level tasks. Each of those low level tasks are low level functions (Figure 4.5. shows lower level functions of "Provide Internal Communication" function which shown in Figure 4.4.) of related high level functions.

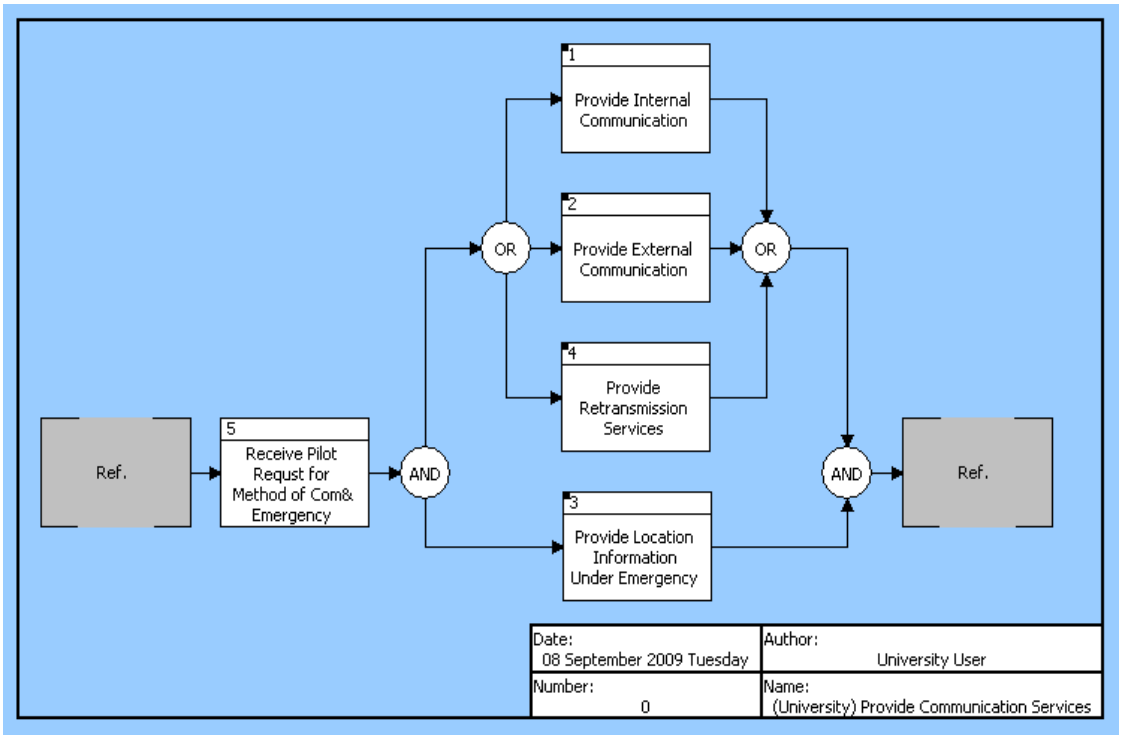

Figure 4.4. First level functional flow block diagram of communication system.

In Figure 4.4. Functional Model is as follows; Communication System, receives Pilot commands for "Method of Communication" and a "Emergency Condition". Then, it provides "Internal Communication, or External Communication, or Retransmission" for Pilot. Meanwhile, providing one of the communication services, it can also provide "Emergency Condition info" to external systems, if any command for it is received from Pilot.

In Figure 4.5. Low level functions of “Provide Internal Communication” function are shown. In this model Internal Communicaiton System, receives Pilot requests for one of the two communication modes. That can be “Full Dublex Mode” or “Half Dublex Mode”

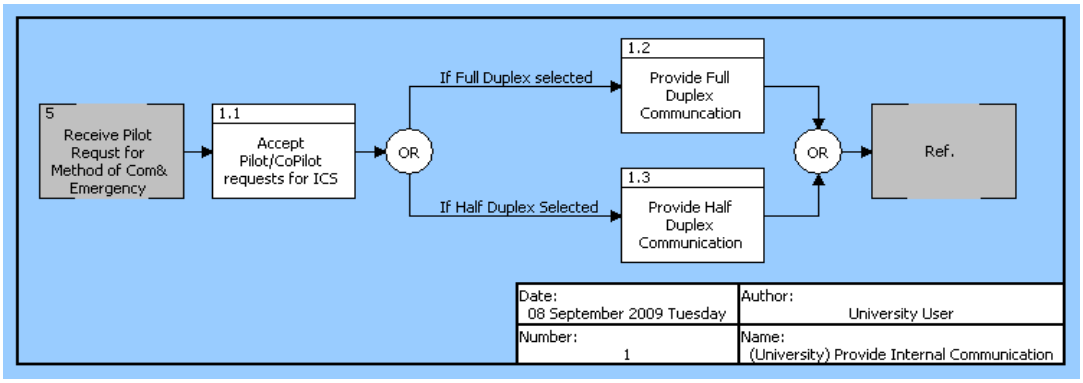

Figure 4.5. Functional flow block diagram of ınternal communication system.

Figure 4.6. shows, Data flow for functions which have been shown in Figure 4.5. Green ballons represent that Data is a “trigger” for the function. Grey ballons represent that Data is only “input” for functions or “output” from functions. The direction of the arrow shows wheather the Data is “input” or “output”.

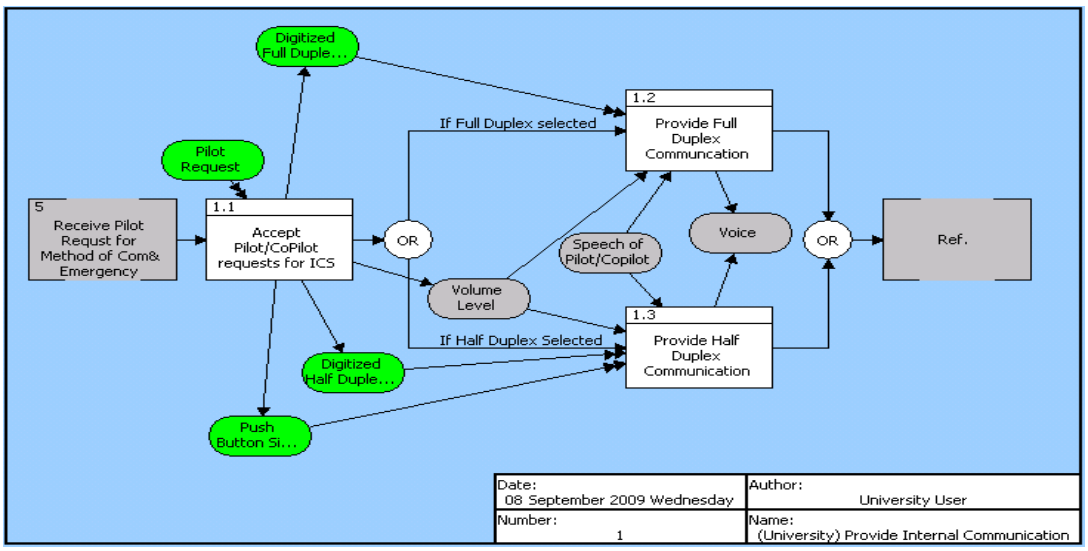

Figure 4.6. Enhanced functional flow block diagram for ınternal communication system.

Figure 4.7. shows the Data relationships between functions of Internal Communication System.

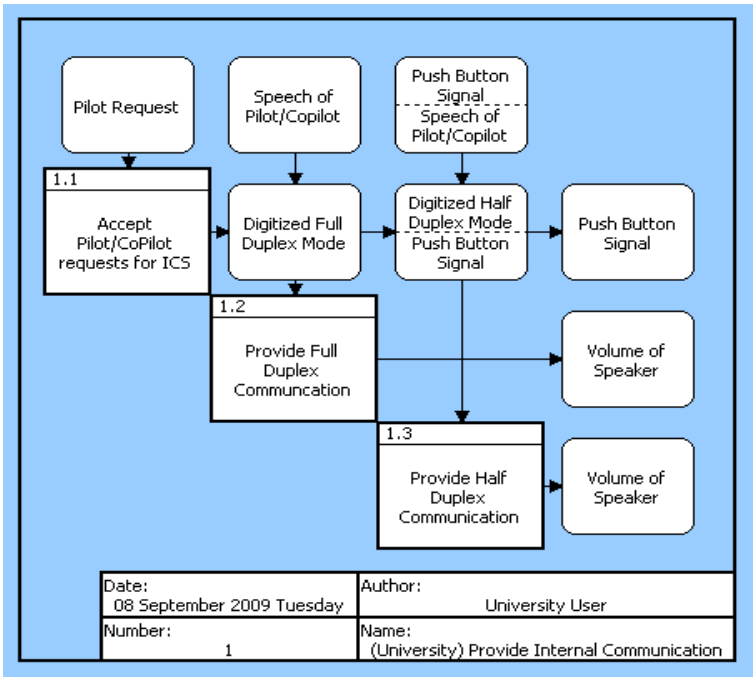

Figure 4.7. N2 chart of ınternal commmunication system.

4.4.3. Simulation Validation

The designed system in case study is validated by using the simulator of the CORE program. Figure 4.8. below shows the simulation results of the system.

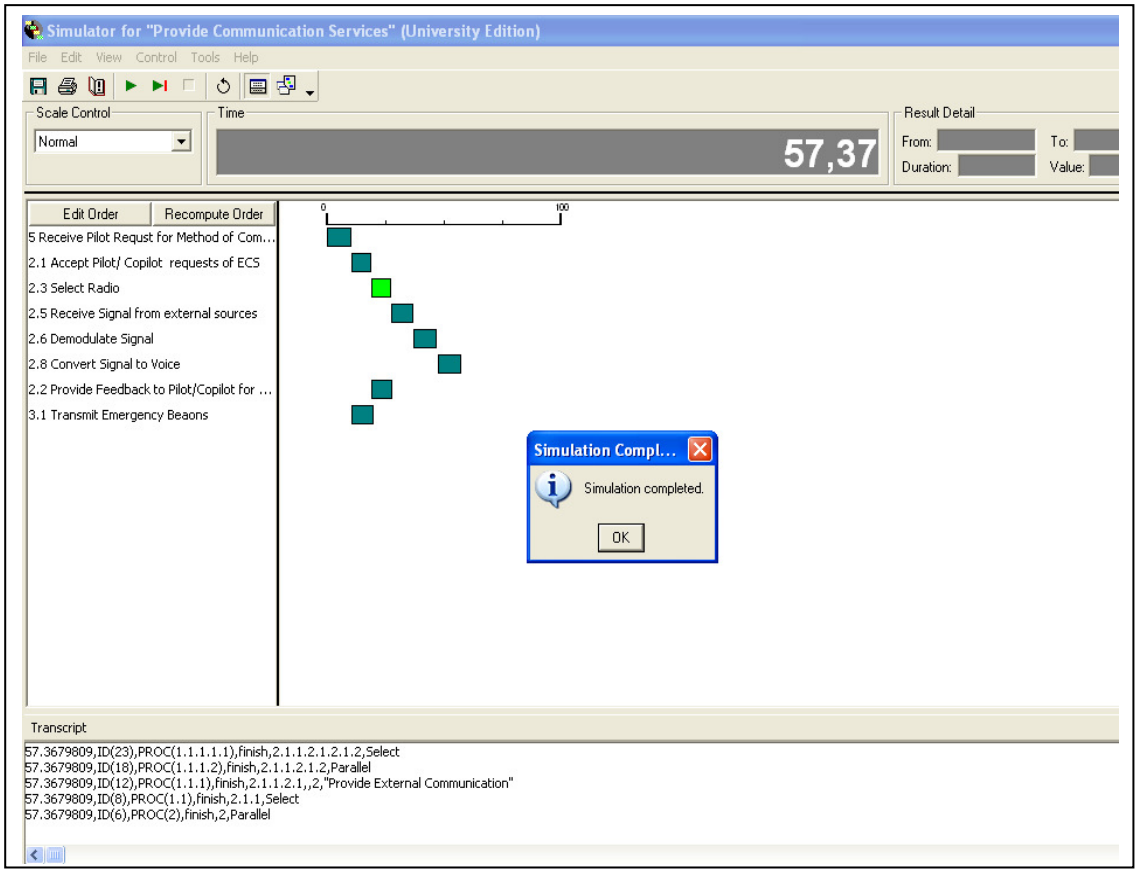

Figure 4.8. System simulation completed.

5. CONCLUSION

Throughout this thesis, the importance and need for system architecture development is discussed. A system's architecture is a means to create systems that are efficient and effective by supplying overview, by guarding consistency and integrity, and by balancing. System's architectures are required to manage the integrated design process, to prevent design conflicts and undesired solutions. Systems architectures are developed to satisfy the needs of stakeholders.

The importance of system architecture in system development life cycle is emphasized by majority of the authors in literature. Moreover, some organizations, governments, and academic studies propose some frameworks to guide systems engineers for developing appropriate system architectures as required by their organizations or for specific system domains. Hovewer, despite all this effort, there still is no consensus on terminology or a common approach to architecture development. The resaerch study reported in this thessis addressed these questions. It was realized that system architectures are formed of views that describe the behaviour and structure of a system. These views are composed of some models (process, data, control, and physical interface models). In turn, models are formed by diagrams which are guided by methodologies; it is possible to obtain a certain model by using different diagrams or methodologies.

Views represent the whole system from the perspective of a related set of concerns. The physical view represents the partioning of physical resources available to perform the systems functions. The functional view of a system contains a hierarchical model of the functions performed; it defines what the system must do, that is, it describes the systems functions and the data flows between them. Operational architecture covers the issues related to allocation of functions to components.

Models are the most important constituent of a system's architecture. They are abstraction of reality constructed for a purpose consisting of formal notations, building blocks, interfaces, interdependencies, and other relationships among the model components. Well supported integrated models are very powerful because they use

graphical notations to provide information of how components are related to each other. With models, the information is substantially condensed in comparison to text. Further, the models can be checked for correctness by engineers and tools. Models that are required to obtain system's architecture. Process model of a system contains functional decomposition and functional ordering information of the system. Control model of the system contains control mechanism of the system. Data model, on the other hand, also named as information model, contains input and output data information of the system. , part tree model which contains physical connection of components, and flow model which shows interfaces between components.

A methodology defines the fundamental vocabulary, semantics and models. It is a recipe that stakeholders can speak about their system in common terms. It is a procedure for resolving the problems. In this thesis two of the most important methodologies are examined. These are Structured Architecture Methodology and Object Oriented Methodology. The Structured Architecture Methodology is a well defined, widely used methodology in system architecture design process. The Structured architecture is based on the concept of functional decomposition. It allows the analyst to break down complicated systems into smaller, clearly defined and more manageable parts. The Object Oriented Methodology uses an object-oriented perspective rather than a functional perspective to design behavior of the system. The major focus of the Object Oriented Methodology is objects, where a complex system is decomposed into several objects. An object-oriented system will consist of these various objects each of which will collaborate and cooperate with other objects to achieve specified tasks.

These two methodolgies are evaluated for their compatibility with life cycle goals, capability of the diagrams to form required models of the system, tool supportability, availability of training and personal expertise. The results found in this work can be summurized as follows. Structural Architecture Methodology has an advantage in explaning system's functional decomposition. This helps in communicating the design process to stakeholders (better than specifications by use cases) since contract requirements are specified in terms of functions. Moreover, it has been seen that Structured Architecture Methodology supports systems design completely when a certain level of abstraction is needed. This prevents design conflitcs due to complexity of the

systems. On the other hand, Object Oriented Methodology is more suitable for detailed design of the systems, particularly for software systems. This is because the models of Object Oriented Methodology can easily be transformed into software design models. In terms of tool supportability and availability of training, both Structured and Object Oriented Methodologies are good. In conclusion, it was concluded that differences in system modeling provided by these methodologies is not sufficient for justifying a choice. Hence, it appears that designers' familiarty or expertise with a particular methodolgy will determine the outcome.

In the case study, Structured Methodology and CORE are used since the designers are familiar with this methodology and CORE. Results of the comparison between two methodologies, and personal expertise on Structured Methodology, were the decision criteria for chosing this methodology. The aim of the case study was to provide practical example to the theoretical study on Systems Architecting, which is explained in this thesis. Hence, it was important to see the results obtained by the case study. The chosen system was complicated enough to examine the disadvantages and advantages of the chosen methodology. Moreover, it was also simple enough to provide perceptible satisfactory results of system's models. The reader should note that in the beginning of the study an attempt was made to develop a system architecture without using a methodology. No hierarchical functional or structural model was built; a pragmatic approach was adopted for system design. As expected, the result was not successful and considerable time was wasted. It was then decided to apply Structured Methodology for the reasons discussed earlier. The system was modeled in the way that Structured Methodology is proposes. The design process, began with forming the high level system functions, and their process model, in accordance with system requirements. After completion of high level functional process models, lower level functions, which come together to compose the high level functions, were produced. Also, their process models were formed. Meanwhile, the components which perform intended functions were produced using the same logic. The result was satifactory for the architecture of required communication system.

Finally, during the case study it has seen that it is not sufficient to use a good methodology for architecture development. In order to built a sound system architecture, it is also essential to conduct and manage requirement analysis properly.

If you don't know, or clearly understand, the customer needs, then you cannot know if you are building the right system-which then makes the technical correctness of the functional spec (what we intend to build) or the design spec (how we think it should work) a moot point [Richard Zultner].

I have made this letter longer than usual because I lack the time to make it shorter [Blaise Pascal].

During the literature research for this thesis, various types of system's architecture design methodologies, models, frameworks and views have been encountered for different system domains. Future work can be conducted to form a matrix which gives a detail comparison of these different types of methodologies, models, frameworks and views for different system domains - in particular, some studies on Business Systems domain needs to be conducted. This thesis established an initial study for this purpose. The following list summarizes the items that will form the entities of such a matrix.

Methodologies to be considered:

- Structured System's Architecture Yourdan – Demacro
- Hatley – Pirbhai
- Ward – Mellor
- Harel
- Object-Oriented System's Architecture
- Activity Based Methodology
- Architecure Specification Model
- Jackson
- Quantitative Quality Function Deployment
- ADARTS

Frameworks to be considered:

- DoD Architecture Framework
- TOGAF
- CORBA

- Zachman Framework
- TEAF: Treasury Enterprise Architecture Framework
- TAFIM: Technical Architecture Framework for Information Management
- SPIRIT Platform Blueprint Issue 3.0
- ISO RM-ODP
- ISO/IEC TR 14252 (IEEE Std 1003.0)
- Federal Enterprise Architecture

APPENDIX A: CASE STUDY SOFTWARE

Conceptual communication system design which is performed in case study is attached into compact disk as software. The software is an executable CORE program.

REFERENCES

1. IEEE 1471:2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, New York, 2000.

2. Estefan J. A., *Survey of Model-Based Systems Engineering Methodologies*, California Institude of Technology, California, 2007.

3. Maier, M.W., Rechtin E., *The Art of Systems Architecting,* CRC Press, Second Edition, Florida, 2000.

4. Muller, G., *Systems Architecting*, Available on site: http://www.gaudisite.nl/, 2009.

5. Kossiakoff A., Sweet W. N., *Sytems Engineering Principles and Practices,* John Wiley & Sons, New Jersey, 2003.

6. Hardy D., Murdock J., *Open Systems Architecting*, Open Systems Joint Task Force, 2000.

7. R. Stevens, P. Brook, K. Jackson & S. Arnold., *Systems Engineering Coping With Complexity,* Printice Hall Europe, England, 1998.

8. The Open Architecture Framework, TOGAF, available on website; http://www.togaf.org/.

9. Fogarty K., Total System Modeling: *A System Enginnering Application of the Higraph Formalism*, NDIA 11th Annual Systems Engineering Conference, San Diego ,2008.

10. Keegan G. J., Kelliher T. P., Oliver D. W., *Engineering Complex Systems,* 1992.

11. Yuksel, S. M., *ESYE 506 Introduction to Systems Engineering Management Course Notes,* Istanbul , 2007.

12. International Standart, ISO/IEC 15288 *Systems engineering — System Life Cycle processe*s, Switzerland, 2002.

13. Yurtseven K. M., Buchanan W. W., Bas S., *Automation and Control System Design: A General Systems Theory Perspective.*

14. Buede, D. M., *The Engineering Design Of Systems ,* John Wiley & Sons, New York , 2000.

15. Muller G., *CAFCR:A Multi-view Method for Embedded Systems Architecting Balancing Genericity and Specificity*, Thesis Book, 2006.

16. Salvatore F., *The Value of Architecture*, , NDIA 11th Annual Systems Engineering Conference, San Diego ,2008.

17. MIL-STD-499A, *Engineering Management*, Washington, 1974.

18. Harry E., *Systems Engineering Vision 2020*, Incose, San Diego, 2007.

19. Huei Wan Ang, Nicholson D., and Mercer B., *Improving the Practice of DoD Architecting with the Architecture Specification Model*, Mitre Coorpration.

20. *Department of Defence Architectureal Framework v1.5*., 2007.

21. Long J., Maley J., *A Natural Approach to DoDAF:Systems Engineering and CORE*,2005.

22. Olson T., Armstrong C., *Architecture and Model Based Systems Engineering for Lean Results*, NDIA Systems Engineering Conference, 2008.

23. Dale M. Rickman, *A Process for Combining Object Oriented and Structured Analysis and Design*, Raytheon Systems Company, 2000.

24. Grady J. O., *Universal Architecture Description Framework*, JOG Systems Enigneering, Inc.

25. D. J. Hatley and I. A. Pirbhai, *Strategies for Real-Time System Specification*., Dorset House, New York, 1988.

26. Pefkaros K., *Using Object-Oriented Analysis and Design Over Traditiıonal Strutured Analysis and Design*, 2003.

27. Long J., *Relationships Between Common Graphical Representations in Systems Engineering*, available on web site www.vitechcorp.com.

28. *OMG Systems Modeling Language Version 1.1*., available on web site www.**omg**.org, 2008.

29. UML 2.1 tutorial, available on web site http://www.sparxsystems.com/resources/uml2_tutorial/index.html.

30. Friedenthal S., Moore A., Steiner R., *OMG Systems Modeling Language (OMG SysML™) Tutorial*, Netherland, 2008.

REFERENCES NOT CITED

Eisner, H., *Essentials of Project and Systems Engineering Management*, John Wiley & Sons, Second Edition, New York , 2002.

Adamsen II, P. B., *A Framework for Complex System Development*, CRC Press, Florida, 2000.

IEEE 1220:2005, *IEEE Standard for Application and Management of the Systems Engineering Process*, New York, 2005.

Moir, I., Seabridge A., *Military Avionics Systems*, John Wiley & Sons, England, 2006.
Military Standart, *MIL-STD-490A Specification Practices*, Washington, 1985.

Cloutier R., *Model Driven Architecture for Systems Engineering ,Stevens Institute of Technology*, Conference On Systems Engineering Research, 2008.

Moinul Khan and Vijay K. Madisetti, *Multi-domain Model Based System Engineering using SysML for Networked Systems*, Georgia Institute of Technology, Conference On Systems Engineering Research, 2008.

Atasoy S., *Aerospace Systems Engineering Course Notes*, Middle East Technical University, Ankara, 2006.

E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*., Prentice-Hall, Englewood Cliffs, 1979.

Hardy D., *Open Sandarts for Architecture Modeling*, Open Systems Joint Task Force, 2005
Madigan M., *Requirement Analysis*, University of Colorado, 2006.

Dos Santos Soares M., Vrancken J., *Requirement Specification and Modeling Through SysML*, IEEE, 2007.

SW Life Cycle Models, Available on site http://searchsoftwarequality.techtarget.com/

U.S Department of Energy, *Systems Engineering Methodology The DOE Systems Development Lifecycle for Information Technology Investments*, 2002.

Wood D., Wood W., *Comparative Evaluations of Four Specification Methods for Real Time Systems*, Carnegie Mellon University, Pennsylvania, 1989.

Gantzer D., Reuss L., *Implications from Standarts and Models Applied to DoD Acquisition Programs*, NDIA 11th Annual Systems Engineering Conference, San Diego, 2008.

Vitech Coorp., *Integrating Architecting and Systems Engineering*, NDIA 11th Annual Systems Engineering Conference, San Diego, 2008.

Systems and Proposal Engineering Company, *Knowledge Based Analysis and Design*, NDIA 11th Annual Systems Engineering Conference, San Diego, 2008.

Scott Derby, *Stop the Pain: Effective Requirements Definition and Management for Project Success*, NDIA 11th Annual Systems Engineering Conference, San Diego, 2008.

Roberts N., Edson R., *System Concept of Operations: Standarts, Practices and Reliability*, NDIA 11th Annual Systems Engineering Conference, San Diego, 2008.

Sui E., *The Challenges of Requirements Decomposition*, NDIA 11th Annual Systems Engineering Conference, San Diego, 2008.

Andrulis Research Coorpration, *Department of Defence Records Management Functions and Information Models*, USA, 1995.

Martin R., C., *Design Principles and Design Patterns*, availlable on site www.objectmentor.com.

Green J. M., Miller G., *Applyimg Open Architecture Concepts to Mission and Ship Systems*, Naval Post Graduate School available on site www.nps.edu.

Talbot I., *Defining 100 Best Practices for Systems Engineering*, NDIA 11th Annual Systems Engineering Conference, San Diego, 2008.

Horner N., Topper S., *Domain modeling Roadmap to Convergence, NDIA 11th Annual Systems Engineering Conference*, San Diego, 2008.

Collins M. R., *Enabling Systems Engineering with an Integrated Approach to Knowledge Discovery and Architecture Framework*, NDIA 11th Annual Systems Engineering Conference, San Diego, 2008.

Sharper C. D., *Enhanced Systems Engineering- Starting Programs Right*, NDIA 11th Annual Systems Engineering Conference, San Diego, 2008.

Fluhr J. H., Macdonald P., *Interoperability Between the DOORS Requirements Management Tool and the CORE Systems Engineering Tool*, available on web site www.vitechcorp.com, 2002.

Muller G., Hole E., *Architectural Descriptions and Models*, White Paper Architecture Forum Meeting, Washington, 2006.

Herzog E., Pandikow A., *SysML- an Assessment*, Syntell AB, Stockholm, 2002
Vitech Coorpration, System Definition Guide, available on web site, www.vitechcorp.com, 2007.

M. Holocher, R. Michalski, D. Solte , F. Vicufia, *MIDA: An Open Systems Architecture for Model-Oriented Integration of Data and Algorithms*, Germany, 1996.

Rob M. A., *Issue of Structured vs Object Oriented Methodology of Systems Analysis and Design*, 2004.

Meilich A., *Integration Model Based Systems Engineering and Human Systems Integration,* NDIA 11th Annual Systems Engineering Conference, San Diego, 2008.

A Comparison of Object-Oriented Development Methodologies, available on web site http://www.ipipan.gda.pl/~marek/objects/TOA/OOMethod/mcr.html.

Champeaux D., Lea D., Faure P., *Object-Oriented Systems Development,* available on web site http://g.oswego.edu/dl/oosd/ch2.html#yourdonb.

Structured Analysis and Design, available on web site http://www.hit.ac.il/staff/leonidM/information-systems/ch03.html.