4+1 View Architectural Model Pattern for Software-Intensive Systems

By

Noor Rasheed Hameed

Supervisor

Dr. Moayad A.fadhil

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Computer Science

Deanship of Academic Research and Graduate Studies

Philadelphia University

2012

جامعة فيلادلفيا

نموذج تفويض

أنا ....[illegible]........ ، أفوض جامعة فيلادلفيا بتزويد نسخ من رسالتي للمكتبات أو المؤسسات أو الهيئات أو الأشخاص عند طلبها.

التوقيع: [illegible]

التاريخ: ١٧ / ٥ / ٢٠١٢

Philadelphia University

Authorization Form

I, Noor. Rashed........ authorize Philadelphia University to supply copies of my Thesis to libraries or establishments or individuals upon request.

Signature: [illegible]

Date: 2012 /5 /17

Successfully defended and approved on ___2012_/5/17

Examination Committee Signature

Signature

Dr. Mazyad A. Fadhil , Chairman. [illegible]

Academic Rank: Associative Prof .

Dr. Rashid Al-Zwaidy , Member. [illegible]

Academic Rank: (Associative Prof.)

Dr. Samir Tartir , Member. [illegible]

Academic Rank: Assistant Professor

Dr. Raed Shatnawi , External Member. [illegible]

Academic Rank:
(Assistant Professor)

Dedication

I fully dedicate this thesis to my love, and my whole family; father, mother, sister and brothers

Also to my professors and all my friends

Noor

Acknowledgment

I would like to express my regards and appreciation to all of those who have helped me and encouraged me and I gift them all of existed beautiful words from thanks and love.

Noor

Table of Contents

List of Figures

List of table

List of Abbreviations

AUML	Agent Unified Modeling Language
ATC	Air Traffic Control
ATCT	airport traffic control tower
AMOLA	Agent Modeling Language
OMG	Object Management Group
OOSE	Object Oriented Software Engineering
SIS	Software Intensive System
SORAP	A Source of Repair Assignment Process
Sys ML	System Model Language
TEC	Tower En Route Control
TRACON	Terminal Radar Approach Control
UML	Unified Modeling Language
4+1	4+1 view architecture model

Abstract

The architecture is general has no specific recommendation on using special method or language. Software-intensive systems are considered as one of the major and essential roles in software engineering, where software-intensive systems affect on the design, constructions, deployment and also the evolution of the whole system. In this thesis we purpose an enhancement on a specific model (4+1 view model) in order to reduce challenges a software intensive system. *The problem* of architecture model for software intensive system is lack of software pattern (Template). The new pattern must enhance the 4+1 view model by providing the model of how to design and implement two types of software-intensive system (monitoring and control) , (Integration and composition) .

So, that we created a new pattern (template) that solves the problem mentioned above.

As a result we will have pattern (template) that ensure an adaptable methodology which can integrate and synchronize all the a activities of software intensive system.

CHAPTER ONE
INTRODUCTION

1.1 Introduction

Software has become a key feature of a rapidly growing range of products and services from all sectors of economic activity. Software intensive systems (SIS) include large-scale heterogeneous systems, embedded systems for automotive applications, telecommunications, wireless ad hoc systems, business applications with an emphasis on web services etc. Our daily lives depend on complex software-intensive systems, from banking to communications to transportation to medicine. Software technology is a driving factor for many high tech products; competence in software technology defines more and more the innovation capability of the whole industry (Wising et al, 2004).

A software intensive system is a system where software represents a significant segment in any of the following points: system functionality, system cost, system development risk, development time. (TNO/IDATE, 2005)

1.2 Problem Statement

During studying some challenges such as (complexity), (adapt changing technology) we found an increase in challenges in software-intensive system and there is no a specific pattern (template) for solving these problems and problem in architecture model.

So our research aims to solve the problem of architecture model for software-intensive system by creating a new software pattern (Template). The new pattern will enhance the 4+1 view model by providing the model of how to design and implement two types of software-intensive system

1.2.1-Monitoring and Controlling of Large Systems or Environments: These applications are characterized by the need to collect data from environments which can partially control the environment.
For example: Systems controlling critical infrastructures such as electricity, transportation , and weather monitoring and prediction.

1.2.2-Integration and composition of highly complex software systems: These applications are characterized by the need to adapt in response to unforeseen changes of requirements, technology, or environment, and to integrate highly dynamic unpredictable diverse knowledge.

For example: Telecommunications, wireless ad hoc systems which are decentralized and have a dynamically changing topology, with nodes constantly entering and leaving the system.

1.3-Software Intensive Systems and 4+1 Architectural Model

Software intensive systems have become a major part of an increasingly growing range of applications, services and products. Software intensive systems are systems in which software interacts with other systems, software, sensors and devices with people. Such systems are like telecommunications, wireless, heterogeneous systems, business applications with web services.

People's activities depends on complex software intensive systems increasingly, such systems are becoming more heterogeneous, decentralized, and operating more in dynamic and often unpredictable; this development has several consequences; as software systems grew increasingly, the focus has moved from the complexity of developing algorithms to the complexity structuring large systems, and then to creating complex distributed concurrent systems. Show (figure (1.1)) as time goes, we will have to face another level of complexity rise up from the fact that systems have to operate in open, large and non-deterministic environments.

Current engineering methods are not powerful enough to design, deploy and maintain software intensive systems. However there is no realistic alternative to such systems, we can't afford to stop building software intensive systems. (Hwang .K, 2008)

Following are some factors responsible for the development of systems which are playing   an increasingly huge role in our lives and daily activities.

1.3.1 Our personal and business life in an interconnected world

Living in a highly interconnected world, which has expanded from wire bound telephone and mail to more rapidly with the internet, portable computers with Bluetooth and mobile phones, and so on.

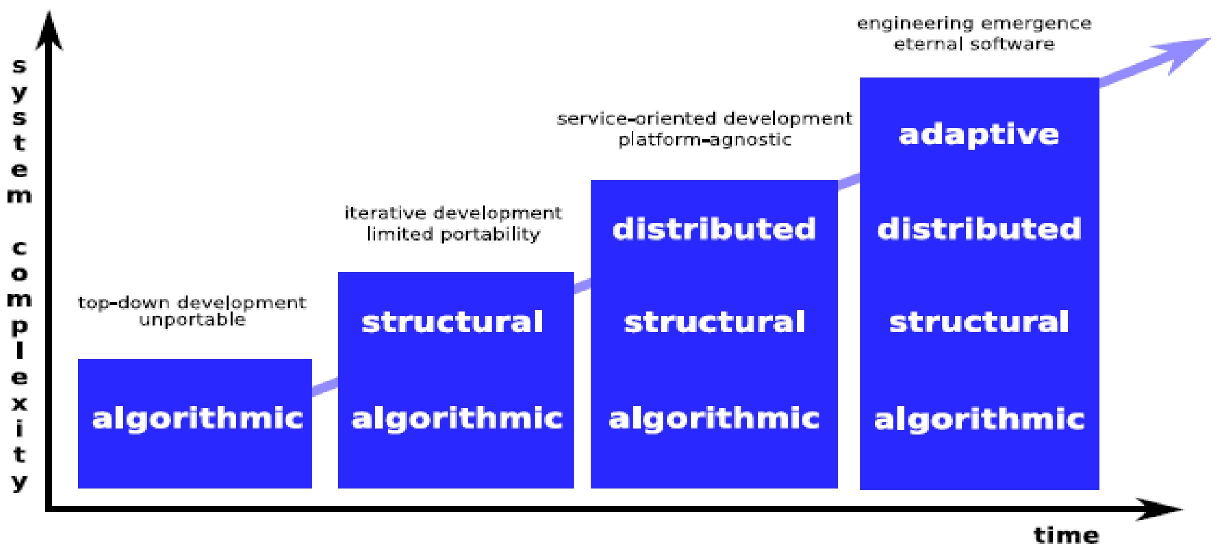


Figure 1.1: Increasing Complexity of Software over Time. (Wirsing et al, 2004)

Communications have diverted from sending letters by mail to writing SMS's, we are no longer storing our documents or books in a public or private libraries, they are being stored now on a network that can be accessed from personal computers or laptops or even mobile devices. Show (figure (1.2)).

In business, we have replaced formal communication from sending letters and faxes to emails, employees are accessing to their company's resources through their personal computers. Business processes are increasingly operated to be more involved with automatic communication by computers and less manual interaction by humans. We depend very much on these systems that a few hours of downtime can have severe consequences, from mild discomfort that many people experience when a network fail for few hours, to millions of dollars of damage when software intensive systems used in international trade fail. (Wirsing et al, 2004)

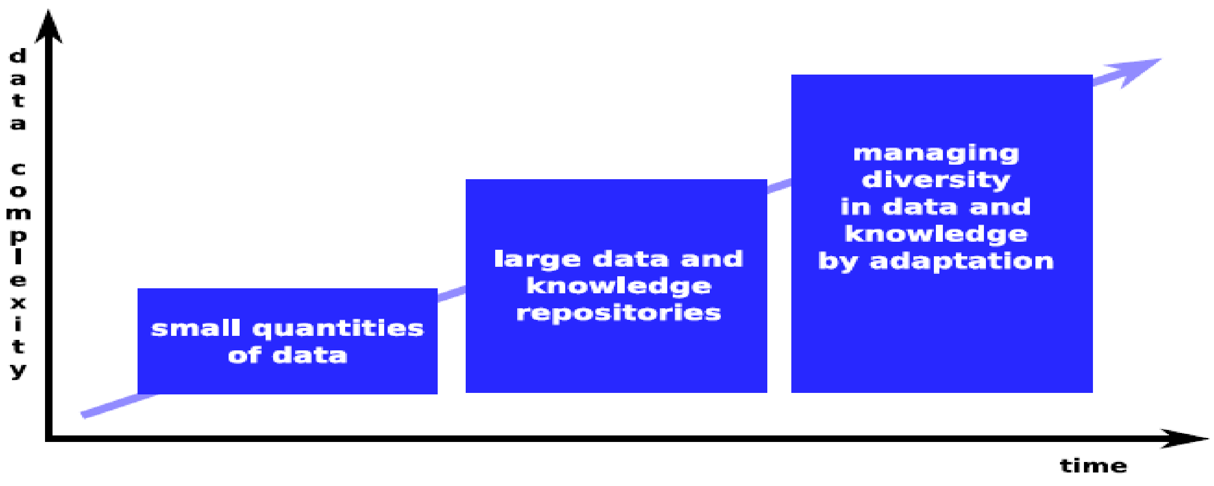

data complexity
small quantities of data
large data and knowledge repositories
managing diversity in data and knowledge by adaptation
time

Figure 1.2: Increasing Complexity of Data over Time (Wirsing et al, 2004)

1.3.2 Service-Oriented systems

In the service-oriented paradigm, services act as autonomous, platform-independent entities that can be published discovered, described, and dynamically assembled for developing massively distributed, evolved systems. Service oriented facilitates personalization of information to applications and end users by different means of context services, in which that the quality of the service is organized according to the context of its use, transaction user, customer type, location, and so on. It also addresses the requirements of dynamic e-business which includes alternative modes of user interaction (e.g. portals), innovative business models (e.g., auctions and e-marketplaces), and devices (mobile phones). Service-oriented system also allow for multi-step transactions to be supported across any device (e.g. web browsers, internet appliances). The promise that Service-oriented systems or architecture offers is a world of cooperating services, where application components are assembled into services that can be gathered to create dynamic business processes and applications. (TNO/IDATE, 2006)

1.3.3 Software-Intensive Systems

Most of the systems depend on software that control the behavior of individual components and the interaction between components, and also it depend on software which interacts with other software, devices, sensors, people, and systems; which means they all depend on software-intensive systems. As a result for the unique ways that software offers, it is possible to build more advanced systems more cheaply and more flexibly than ever before, for example, the possibilities given by embedded systems; their computational power increased exponentially, while the cost and energy consumption decreases; this will accelerate the process of transitioning from manual or mechanical systems to computers in the near future. Software intensive systems will become even more spread in the next years. (Lapham, 2006)

-Exploitation, integration and composition of highly complex software systems

These applications are characterized by the need to adapt in response to unforeseen changes of requirements, technology, or environment, and to integrate highly dynamic unpredictable diverse knowledge, but also by the need to integrate and compose the execution of autonomous software systems. A possible solution here is to develop seamless adaptive service systems: namely systems embedded in human-centered environments, which meaningfully interact with humans and offer seamless adaptive services, thus hiding the complexities of their internals for example:

a) **SIS for communication and entertainment.** Telecommunications systems, e.g., wireless ad hoc networks, are decentralized and have a dynamically changing topology, with nodes constantly entering and leaving the system. The cable television system is being combined with the Internet and therefore integrated into a software-intensive system by the introduction of multi-purpose home entertainment centers.
b) **SIS for business**. The infrastructure of large companies becomes increasingly software-intensive with increasing integration of computer-aided design (CAD), manufacturing (CAM), procurement, supply management, etc. Business applications are increasingly built from distributed, dynamically assembled services and endowed with decision-making capabilities. (Soares et al , 2009)

1.4- “4+1” Architectural view model

1.4.1 Architectural model

Software architecture deals with high-level structure of the software. Software architecture is the result of assembling a number of elements in some well-chosen forms to achieve the major functionality requirements of the system, along with non-functional requirements such as scalability, portability, reliability and availability. Each view defines a set of *elements* to use (components, containers, and connectors). We capture the *forms* and patterns that work, and we capture the *rationale and constraints,* connecting the architecture to some of the requirements. Perry and Wolfe put it in the following formula which has been modified by (Boehm, Kruchten, 1995).

Software architecture = {Elements, Forms, Rationale/Constraints}

Most of architectural documents are overemphasizing the aspect of development or do not address the concerns of all stakeholders; various stakeholders of software system, such as, end-user, developers, system engineers, and project managers. Software engineers used to struggle to represent more on one blueprint, and so architectural documents contain complex diagrams. 4+1 is a view model, was designed by Philippe (Kruchten, 1995) for "describing the architecture of software-intensive systems, based on the use of multiple, concurrent views". The views are used to describe the system for the viewpoint of different stakeholders. The four views are *logical, development, process* and *physical* view. as description in (figure (1-3)) .

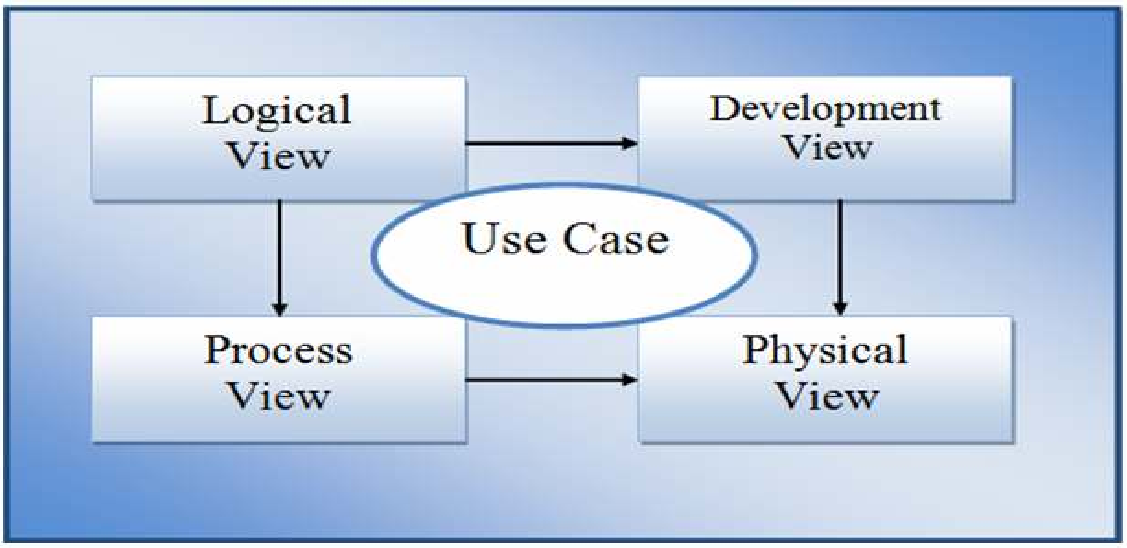


Figure 1.3: 4+1 Architecture View model

Logical view: The logical view is concerned with the functional requirements; what the system should provide in terms of services to its end-users. The system is decomposed into a set of keys that are taken from the problem domain, which are represented in the form of *objects* or *object classes*. These objects use the principles of encapsulation, abstraction and inheritance

The decomposition also serves to identify common mechanisms and design elements across the various parts of the system beside the functional analysis.

Sets of related classes can be grouped into class categories. Class templates focus on each individual class; they emphasize the main class operations, and identify key object characteristics. If it is important to define the internal behavior of an object, this is done with state transition diagrams, or state charts. Common mechanisms or services are defined in class utilities. Alternatively to an OO approach, an application that is very data-driven may use some other form of logical view, such as E-R diagrams. UML diagrams used in this view to represent the logical view include Class diagram, Sequence diagram, and Communication diagram. Show (figure (1-4)). Illustrates how logical view can be represented using a UML tool like Rational Rose.

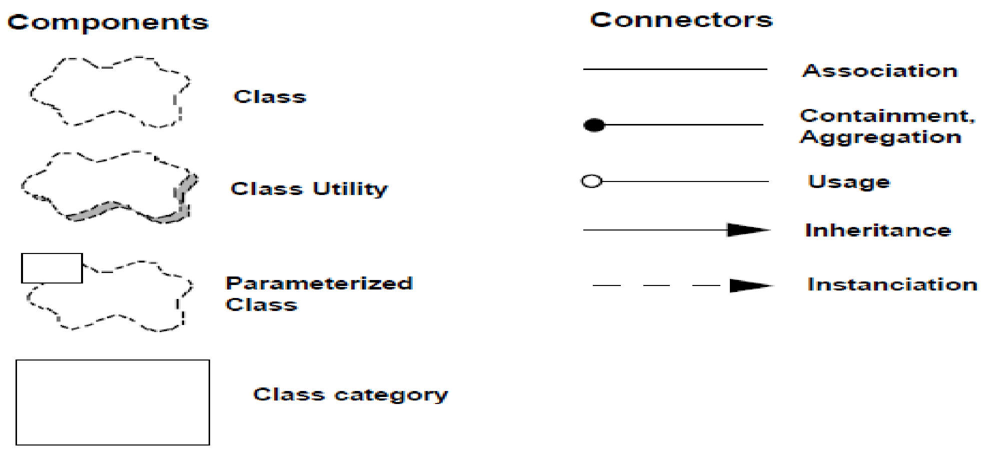


Figure 1.4: Notation for the logical blueprint

Development view: The development view illustrates a system from a programmer's perspective; it is concerned with software module organization (Hierarchy of layers, software management, reuse, constraints of tools). The software is packaged in small chunks (program libraries or subsystems) that can be developed by on or small number of developers. The subsystems are organized in a hierarchy of layers, each layer providing a narrow well defined interface to the layer above it. The development architecture of the system is represented by module and subsystem diagrams, showing the 'export' and 'import' relationships. The complete development architecture can only be described when all the elements of the software have been identified. It is, however, possible to list the rules that govern the development architecture: partitioning, grouping, visibility. Development view is also known as the implementation view. UML diagrams are also used in development view to represent it and to represent the Package diagram. Show (figure (1-5)). (Mikko, 2005)

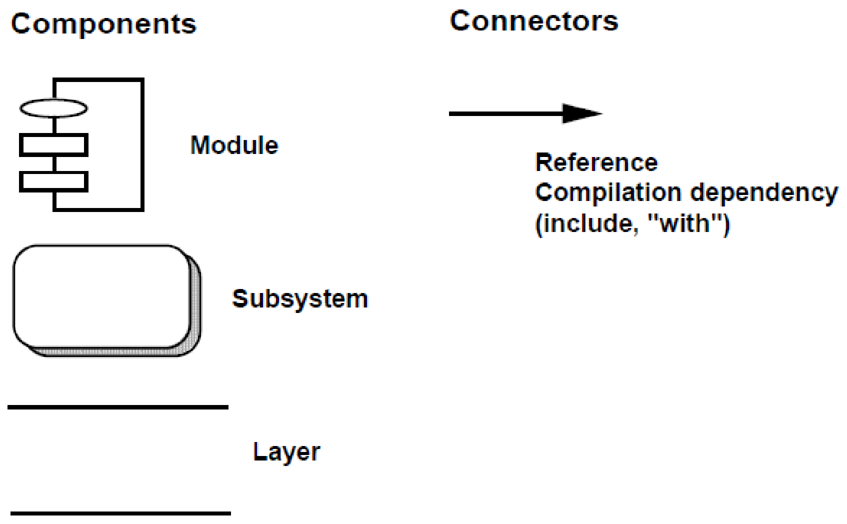


Figure 1.5: Notation for the Development blueprint (Kruchten, 1995).

Process view: The process view is considering a non-functional requirement, it addresses concurrency, integrators, distribution, performance, and scalability, it explains the system processes and how do they communicate and focuses on the runtime behavior of the system. A process is a grouping of tasks that form an executable unit. Processes represent the level at which the process architecture can be tactically controlled (i.e., started, recovered, reconfigured, and shut down). In addition, processes can be replicated for increased distribution of the processing load, or for improved availability. The software is partitioned into a set of independent tasks. A task is a separate thread of control that can be scheduled individually on one processing node. We can distinguish then: major tasks, that are the architectural elements that can be uniquely addressed and minor tasks, that are additional tasks introduced locally for implementation reasons (cyclical activities, buffering, time-outs, etc.). They can be implemented as light-weight threads. Major tasks communicate via a set of well-defined inter-task communication mechanisms: synchronous and asynchronous message-based communication services, remote procedure calls, event broadcast, (Kruchten, 1995).etc. Minor task may communicate by rendezvous or shared memory. Major tasks shall not make assumptions about their collocation in the same process or processing node. Show (figure (1-6, 1-7)).illustrates how a process views would be represented using a UML tool. (Mikko, 2005).

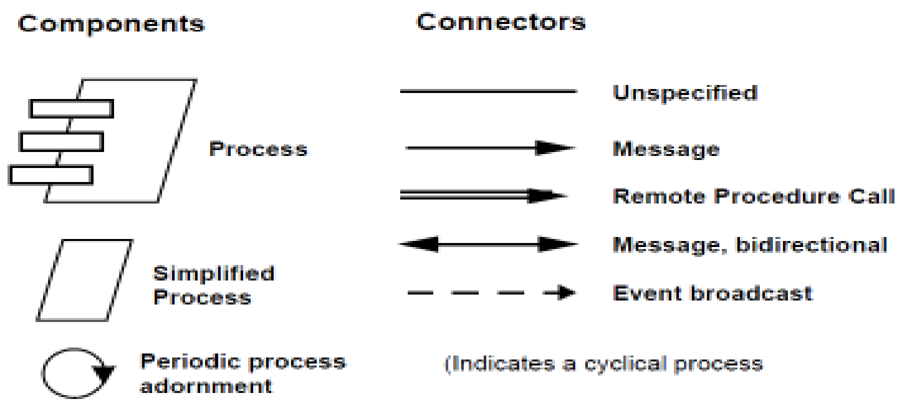


Figure 1.6: Notation for the Process blueprint

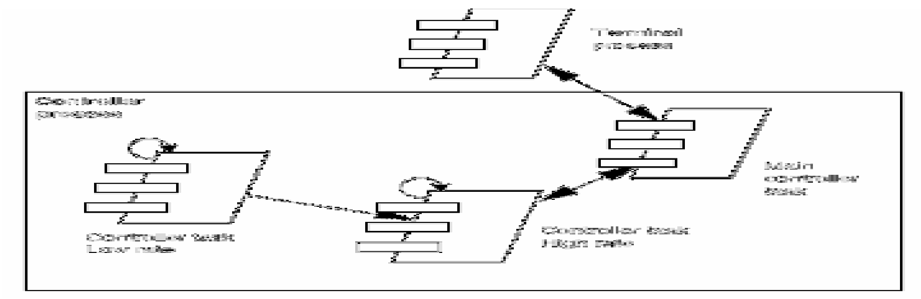


Figure 1.7: Process view on PABX example (Kruchten, 1995).

Physical view: The physical view describes a system from a system engineer's point of view. It is concerned with a non-functional requirements regarding to underlying the topology of software components and communication on the physical layer. It is also known as the deployment view. The software executes on a network of computers, or processing nodes (or just nodes for short). The various elements identified— networks, processes, tasks, and objects—need to be mapped onto the various nodes. We expect that several different physical configurations will be used: some for development and testing, others for the deployment of the system for various sites or for different customers. The mapping of the software to the nodes therefore needs to be highly flexible and have a minimal impact on the source code itself (Kruchten, 1995) .UML diagrams used to represent physical View. Show Fig (1-8).illustrates how a physical view can be represented using a UML tool. (Mikko, 2005)

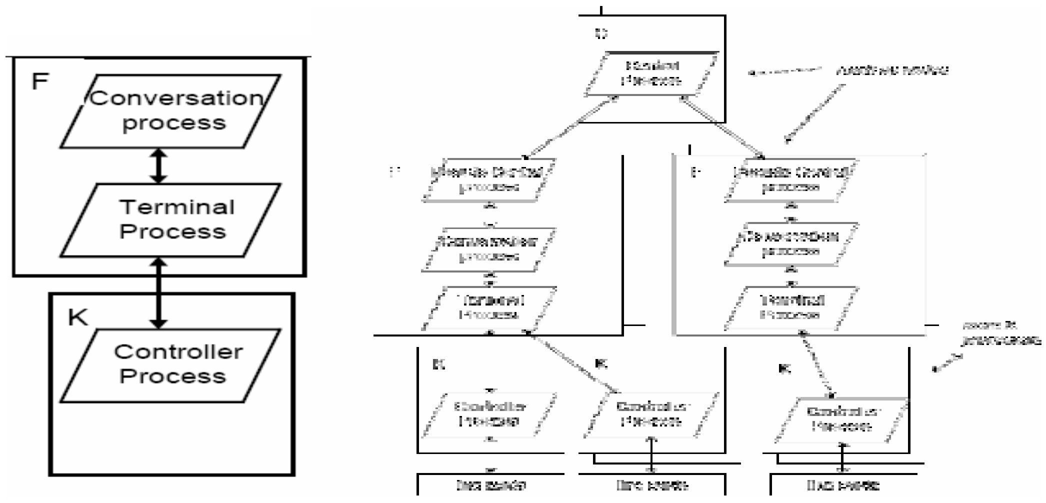


Figure 1.8: Physical view example (Kruchten, 1995).

Scenarios: The scenarios are the description of architecture that is illustrated using a small set of use cases. It helps to illustrate and validate the document and describe sequences of interactions between objects and between processes. They also help as a starting point for testing an architecture prototype. scenarios are in some sense an abstraction of the most important requirements. Their design is expressed using object scenario diagrams and object interaction diagrams. This view is redundant with the other ones (hence the "+1"), but it serves two main purposes: (Kruchten, 1995).

First: As a driver to discover the architectural elements during the architecture design as we will describe later, Second: As a validation and illustration role after this architecture design is complete, both on paper and as the starting point for the tests of an architectural prototype.

1.5 Modeling language

Modeling languages, like programming languages, need to be *designed if* they are to be practical, usable, accepted, and of lasting value. It presents principles for the design of modeling languages to arrive at these principles; it considers the intended use of modeling languages. Principles of modeling language are applicable to the development of new modeling languages, and for improving the design of existing modeling languages that have evolved, perhaps through a process of unification. The principles are illustrated and explained by several examples, drawing on object-oriented and mathematical modeling languages. There are many types of modeling language can be used in this research starting with the used modeling language in 4+1 view model which is Rational Rose. After this tow authors make a research to compare which is the best modeling language can be used with 4+1 view models. They compared between two modeling languages Sys ML and Unified Modeling Language (UML).

1.5.1 Unified Modeling language (UML)

The Unified Modeling Language (UML) is a family of design notations that is rapidly becoming a de facto standard software design language. UML provides a variety of useful capabilities to the software designer, including multiple, interrelated design views, a semiformal semantics expressed as a UML meta model, and an associated language for expressing formal logic constraints on design elements. The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

1.5.2 Goals of UML

The primary goals in the design of the UML were:

1-Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.

2-Provide extensibility and specialization mechanisms to extend the core concepts.

3- Be independent of particular programming languages and development processes.

4- Provide a formal basis for understanding the modeling language.

5- Encourage the growth of the OO tools market.

6- Support higher-level development concepts such as collaborations, frameworks, patterns and components.

7- Integrate best practices.

1.5.3 Using UML in this Research

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. The Unified Modeling Language (UML) was designed to respond to these needs. UML is a result of the evolution of object-oriented modeling languages. It was developed by Rational Software Company by unifying some of the leading object-oriented modeling methods:

- Booch by Grady Booch .
- OMT (Object Modeling Technique), by Jim Raumbaugh and
- OOSE (Object-Oriented Software Engineering), by Ivar Jacobson.

UML is used for modeling software systems; such modeling includes analysis and design. By an analysis the system is first described by a set of requirements, and then by identification of system parts on a high level. The design phase is tightly connected to the analysis phase. It starts from the identified system parts and continues with detailed specification of these parts and their interaction. For the early phases of software projects UML provide support for identifying and specifying requirements as use cases. Class diagrams or component diagrams can be used for identification of system parts on a high level. During the design phase class diagrams, interaction diagrams, component diagrams and state chart diagrams can be used for comprehensive descriptions of the different parts in the system.

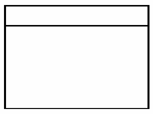
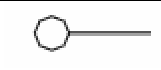
1.5.4 Basic building blocks of UML

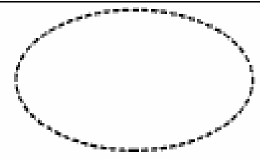
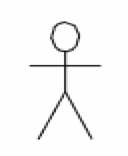
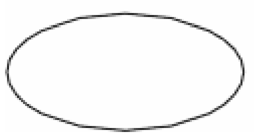
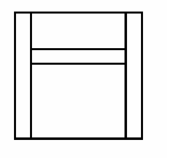
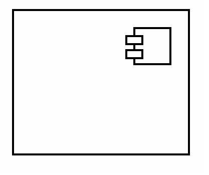
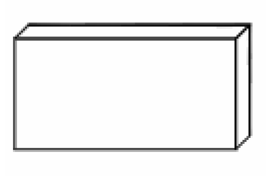
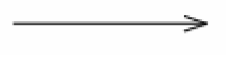
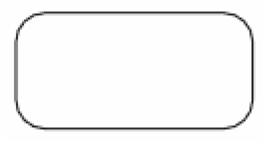
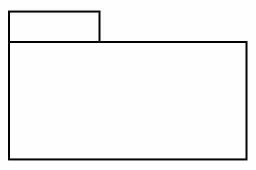
The basic building blocks in UML are things and relationships; these are combined in different ways following different rules to create different types of diagrams. In UML there are 13 types of diagrams, below is a list and brief description of them. The more in depth descriptions in the document, will focus on the first five diagrams in the list, which can be seen as the most general, sometimes also referred to as the UML core diagrams.

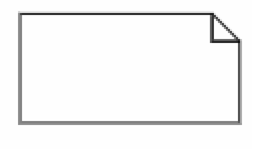
1. Use case diagrams; shows a set of use cases, and how actors can use them.
2. Class diagrams; describes the structure of the system, divided in classes with different connections and relationships
3. Sequence diagrams; shows the interaction between a set of objects, through the messages that may be dispatched between them.
4. State chart diagrams; state machines, consisting of states, transitions, events and activities.
5. Activity diagrams; shows the flow through a program from an defined start point to an end point.
6. Object diagrams; a set of objects and their relationships, this is a snapshot of instances of the things found in the class diagrams.
7. Communication Diagrams (Collaboration diagrams in UML-1); a way to show how objects are linked together and how messages are sent between them.
8. Component diagrams; shows organizations and dependencies among a set of components. These diagrams address static implementation view of the system.
9. Deployment diagrams; show the configuration of run-time processing nodes and components that live on them.
10. Package diagrams; is used to group classes at compile time to get an easier overview of a bigger system with a lot of classes.
11. Composite Structure diagrams; runtime decomposition of a class, it is like a package diagram but it shows the grouping at runtime instead of compile-time.
12. Interaction Overview diagrams; a mix of sequence diagrams and activity diagrams.
13. Timing diagrams; shows interaction between objects base on timing.

1.5.5 Representing "Things" in UML

Things are used to describe different parts of a system; existing types of things in UML are presented in (table (1.1). (Table 1.1 Type of things)

Type of Things	Symbol	Description
Class		Description of a set of objects that share the same: attributes, operations, relationships and semantics
Interface		A collection of operations that specify a service of a class or component.

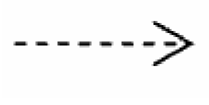
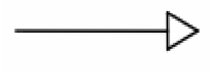
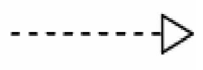
Collaboration		An interaction and a society or roles and other elements that work together to provide some cooperative behavior that is bigger than the sum of all the elements. Represent implementation of patterns that make up the system
Actor		The outside entity that communicates with a system, typically a person playing a role or an external device
Use Case		A description of set of sequence of actions that a system performs that produces an observable result of value to a particular actor. Used to structure behavioral things in the model.
Active class		A class whose objects own a process or execution thread and therefore can initiate a control activity on their own.
Component		A component is a physical and replicable part that conforms to and provides the realization of a set of interfaces.
Node		A physical resource that exists in run time and represents a computational resource.
Interaction		Set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose.
Activity		A behavior that specifies the sequences of steps a computational process performs during its lifecycle.
Packages		General purpose mechanism of organizing elements into groups.

Note		A symbol for rendering notes and constraints attached to an element or a collection of elements.

1.5.6 Representing of “Relationships” UML

The types of UML relationships are shown in the (table (1.2) , relationships are used to connect things into well defined models (UML diagrams).

(Table 1.2 relationships)

Name	Symbol	Description
Dependency		A semantic relationship between two things in which a change to one thing may affect the semantics of the dependent thing.
Generalization		Specialization/generalization relationship in which objects of the specialized element are substitutable for objects of the generalized element.
Realization		Semantic relationship between two classifiers, where one or them specifies a contract and the other guaranties to carry out the contract. They are used between: - Interfaces and classes or components. - Use cases and collaborations that realize them.

CHAPTER TWO

PREVIOUS STUDIES

2.1 Previous studies:

Kruchten (1995), published a paper titled “Architectural Blueprints: The “4+1” View Model of Software Architecture”. This paper presents a model for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views. This use of multiple views allows addressing separately the concerns of the various ‘stakeholders’ of the architecture: end-user, developers, systems engineers and project managers. It describes how to handle separately the functional and non functional requirements. Each of the five views is described, together with a notation to capture it. The views are designed using an architecture-centered, scenario driven, iterative development process.

Our search works more useful because we have to ensure:

1-Enhancement and development on the 4+1 view model not only using the language of representation, but in the same architectural model by finding a new pattern (template).
2 - After the development and the improvement of ”4+1 view model”, we create a new pattern (template)which will be used to solve the existing problems with the design and implementation of the SIS.

Lapham (2006), wrote a technical note discussing some particular questions including definitions, relevant concerns, future considerations, along with some suggestions for maintaining software-intensive systems. Well-done sustainment and assists to lead to well supported software-intensive systems, in addition to decreasing the total ownership costs, along with supporting the organizations to meet the latest mission area and capabilities requirements. The issues that are facing systems are generally the result of their very nature. A solution for organizations so they could be more effective in sustaining software-intensive systems is by guiding these organizations to be more aware of the issues which are addressing them practically. Some of the issues mentioned are: the lack of funding for transition planning, the lack of signed SORAP or any equivalent document, including unclear AI requirements, and the un-addressed support database and tools transition logistics. But we are done on found pattern (template) using in software intensive system that solution the one on challenges in SIS.

Tiako (2008), mentioned in “Designing Software-Intensive Systems” that the methods and principles which address as the complex subjects, that are associated with software engineering environment capabilities for designing real-time embedded software systems. This provides relevant theoretical foundations, principles, methodologies, frameworks, and the latest research findings in the field to deliver a superior knowledge base for those in computer science, software engineering, and fields alike.
have described Software-intensive systems are large, complex systems, in which software is an essential component, interacting with other software, systems, devices, actuators, sensors and with people, and

that it is an essential key attribute of a growing rising variety of products and services in all sectors of the economic activity design for software-intensive systems requires adequate methodology and tool support in order for researchers and practitioners to make use of and develop very large and complex systems. Software engineering environments help reduce the design costs of very large and intricate software systems while improving the quality of the software produced. But otherwise we create pattern (template) to solving problem (increase complexity) for software intensive system

Buettner (2008) had mentioned that the development of schedule-constrained software-intensive space systems is very challenging and difficult. Case study information provided by the national security space programs developed at the U.S. Air Force Space and Missile Systems Center (USAF SMC) have provided evidence of the strong desire by suppliers to omit or severely trim down software development design, and the early defect detection methods in these schedule-constrained environments. His own research findings had suggested some recommendations to entirely address these issues at several levels. Though, the observations lead him to investigate modeling and theoretical methods to basically show us and help us to understand what motivated this behavior. The result of the study clears that the inspection-based system dynamics model is adapted to embrace unit testing and an integration test feedback loop. This Modified Madachy Model (MMM) is used as a tool to investigate the consequences of this behavior on the observed defect dynamics for two remarkably different case study software projects.

Soares and *Vrancken* (2009), introduced the "4+1" View Model of Software Architecture, whereas they developed this model by including the Systems Modeling Language (SysML) with the model especially with software intensive system. The authors argued software intensive systems are normally large scale systems in which software plays a fundamental role, but other elements are also important. The purpose of this article is to show where and how Sys ML applied to model other elements than just the software in a software-intensive system, can be included in the 4+1 View Model of Architecture. For each view, UML and SysML diagrams are jointly used. The methodology used in this research is basically based on Action Research, where as the researchers here work as a team all together with the practitioners so they could be able to apply the theories which they have developed, and then test these theories in practice. In this research and approach, the result was obviously feasible while evaluating the company's employees. In this research, the scientists doing research and study singled out by comparing the two modern methods of representation of the system or the regulations are the SysML, UML while we've improved on this research so that we use and like us in a language representation of the UML to solve the existing problems in SIS, and we came up with pattern (template) solve the problems found by the SIS, our search job more useful because we have to represent and apply the architecture pattern (template) to language representation UML and developed and improved on the model used by the Papers himself, which is the 4+1 View model and solve a problem of the SIS .

Boehm and valerdi (2011) “Impact of Software Resource Estimation Research on Practice: Achievements, Synergies, and Challenges”, this paper is an involvement of the Impact Project in the area of software resource estimation. The main purpose of the Impact Project has been analyzed concerning the impact of software engineering research investments on software engineering perform. In addition to have exposed that: by means of well-defined systems engineering approaches for software- intensive systems development results in improved and enhanced cost and schedule performance, and amplifies the likelihood that the implementation will meet the user's needs. Additional benefits conclude the production of adaptable and resilient systems, along with improved reuse and better documentation.

Muller (2011), have mentioned in his paper “The Role of Software in Systems” that the amount of software in various types of systems raises exponentially. This augment impacts the reliability of these systems. In the source code of software many hidden faults are present. In which these hidden faults could transform into errors through the life cycle of the system itself, due to the changes that occur in the system, or due to the context of the system software which is a dominating issue in the development process. In addition to the role of software in the broader system context; an advanced comprehending of the functionality of software, whereas it enables the system architect and the stakeholders of the product creation process, of integrating the software development in a better way. Our search works more useful because:

3- It is clear that there is an increase in the growth of complexity and this is what we found that our search has provided results that are better in terms of reduced complexity in SIS through using pattern (template). Where the produced pattern (template) could be flexible (customized) to work with any system or Application in software intensive systems.

4- Through the creation and the application of the new pattern (template), we have solved one of the challenges in SIS (Increasing the complexity because of having an increase in functions).

5- Handling Complexity and the Two-Way Alignment between the Business (Problem) and the Solution (Software).

The studies presented, were either concerned about software intensive system (SIS), or the 4+1 view model. Each of *Buettner (2008), Boehm and valerdi* (2011), *Tiako* (2008), *Lapham* (2006), addresses the software-intensive systems; whereas they generally defined the software-intensive as the following: Software-intensive systems are considered as one of the major and essential roles in software engineering, where software-intensive systems affect on the design, constructions, deployment and also the evolution of the whole system. Yet, software-intensive systems face some challenges, two of the main challenges which affects on the software-intensive systems are: the enormously *growing complexity* of software-intensive systems, along with the growing and continuous requiring of adapting a fast changing technology and environment. Where this research aims to solve the problem of architecture model for software-intensive systems, and that's through creating a new software pattern (Template). On the other hand, the following researchers *Soares* and *Vrancken* (2009), and *Kruchten* (1995), have concentrated more on the 4+1 view model, which basically concludes the four main views or architecture (logical, process, deployment, and implementation), the +1 is the (use case) which is the crosscutting view that integrates the four views mentioned previously. In addition to the common modeling language UML which associates with the 4+1 view model including Sys ML. However, some specific methods, or languages for design are not supported, in this part this research is considering a solution to support the system of the 4+1 view model, and make it more flexible by designing this new pattern (template), by using the 4+1 view model in order to find a solution in designing and implementing the software-intensive system.

CHAPTER THREE

METHODOLOGY

3.1 Introduction (Research Methodology)

In this chapter we will be touching and explain the methodology of this research and some important paragraphs and the relations among them as

The aim of this research is create software pattern (Template), Which will enhance the 4+1 view model and it's based on "4+1 view model" software architecture to solve one problem or challenge of Software Intensive System (SIS). It is a description or template for how to solve a problem that can be used in many different situations. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

(Figure (3-1)) shows the steps that will be taken in order to achieve the main aim of this research.

This research will adopt one challenge of software-intensive system to be solved by 4+1 view model.

Figure (3-1) explain the steps that research will follow to achieve the main purpose.

Step 1: there are many software architectures available to design and implement the systems. After studying the software architectures, it can be seen the most appropriate architecture that will be used is 4+1 view models.

Step 2: there are many challenges that face software intensive systems. After studying the software-intensive system, later each challenge will be discussed and figure out the main reasons of each challenge.

Step 3: study the selective software architecture and compare between SIS's challenges and each view in order to find a relationship between them.

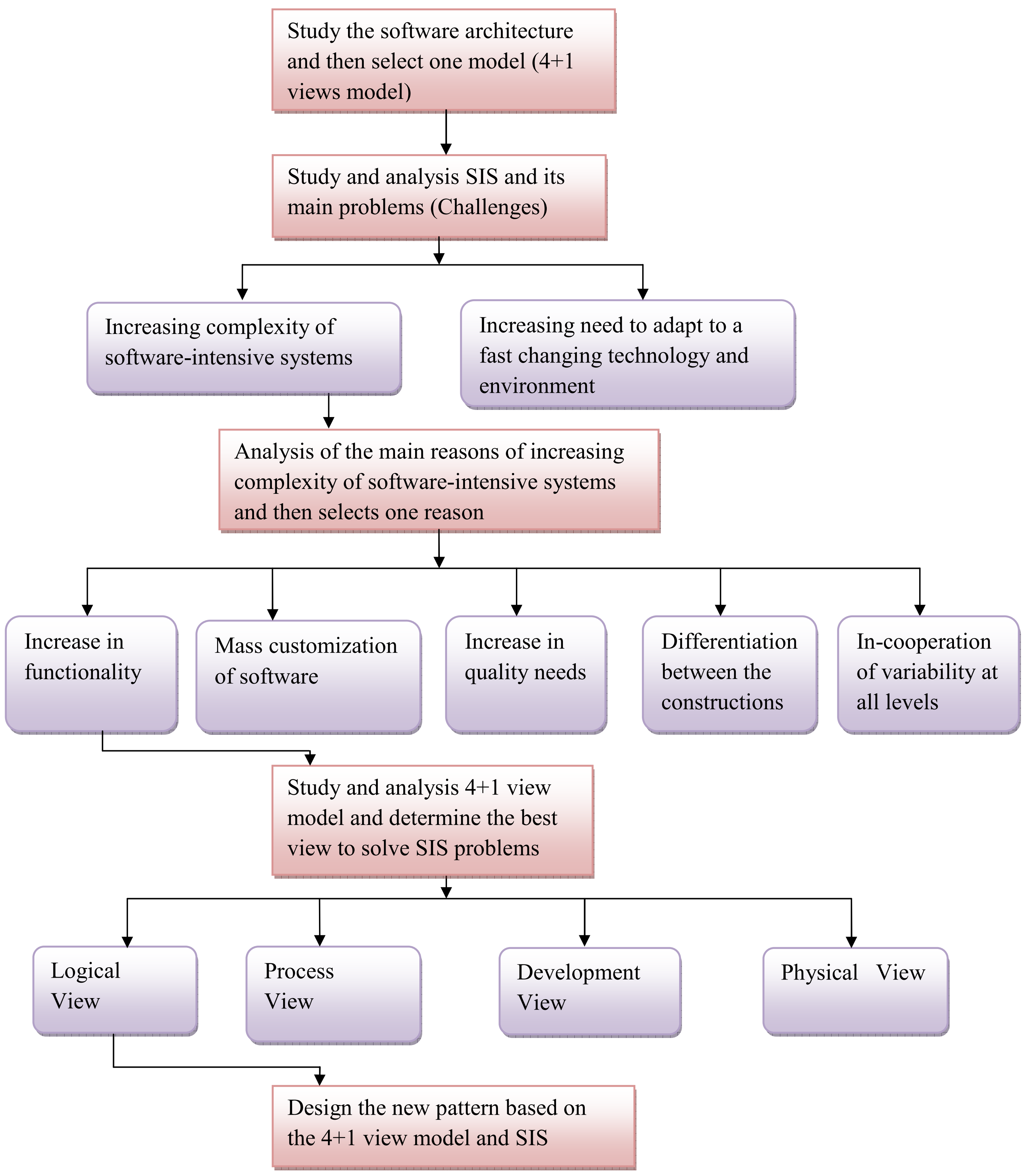
Step 4: Design the new template based on SIS and 4+1 view architecture models.

3.2 Software-Intensive Systems Challenges

There are two challenges that face SIS in general:

a) The *increasing complexity* of software-intensive systems.
b) The increasing need to adapt to a fast changing technology and environment.

Each challenge has its own reason.



Study the software architecture and then select one model (4+1 views model)
Study and analysis SIS and its main problems (Challenges)
Increasing complexity of software-intensive systems
Increasing need to adapt to a fast changing technology and environment
Analysis of the main reasons of increasing complexity of software-intensive systems and then selects one reason
Increase in functionality
Mass customization of software
Increase in quality needs
Differentiation between the constructions
In-cooperation of variability at all levels
Study and analysis 4+1 view model and determine the best view to solve SIS problems
Logical View
Process View
Development View
Physical View
Design the new pattern based on the 4+1 view model and SIS

Fig. 3-1 Architecture Methodology of this research (Step-by-step)

3.2.1 Main reasons of "*increasing complexity*" challenge:

1. *Increase in functionality*: As observable, e.g., in the automotive or telecommunication industry, the functionality of software increases dramatically. Each new product (or even software version) must of course provide all the "old" functionality and, in addition, in cooperation a lot of new functions – quite often, a platform change is required in parallel, and other devises such as sensors change their behavior so that the "old" functionality must be adjusted accordingly.

2. Increase in quality needs: The time where the delivery of faulty software or software requiring a significant, not intended adaptation of its environment still satisfied the customer is over. Today, software – as all other products – must be delivered without failure and must fit in the environment. In addition, there are increasing demands for the quality of the internal software "structure; one main reason for demanding an increase in software quality. Another main reason is the increase of quality features like security, safety or mobility (which often are used to express opaque functional features).

3. Mass customization of software: Customer demands an adaptation of the software-intensive product to their needs – like with other products like cars. This leads to a mass customization of software-intensive products which quite easily lead to the existence of several hundred or even thousands of software systems/versions. Maintaining and managing their evolution is a main reason for the increase of complexity – orthogonal to the above. The challenge that will be solved in this research is "Increase in functionality" which is a very important reason for complexity of software-intensive systems.

3.2.2 Main reasons of "need to adapt to a fast changing technology and environment"

1- Requirements engineering methods, techniques and tools for facilitating the two way alignment establishing requirements as a "bridge" between the problem and the solution space.

2- Dynamic adaptation and run-time evolution as well as run-time evaluation of software systems.

3- Quality assessment of components (open source and COTS) during runtime, certification of such components.

4- Standardization of components and semantic component interfaces to allow semantic run-time evaluations; even more important as basis for standardization.

3.3-Relationship between SIS and 4+1 view models

The methodology that will be adopted in this research is finding the best view of the 4+1 view models.

Let us check each view and determine the best view can solve one of SIS challenges

1. The Logical Architecture:

   This view concerns about the functions and services that are provided to clients. The logical architecture primarily supports the functional requirements. Logical view can be formulated by one question: "what the system should provide in terms of services to its users". The system is decomposed into a set of key abstractions, taken (mostly) from the problem domain.
2. The Process Architecture:

   The process architecture takes into account some non-functional requirements, such as performance and availability. It addresses issues of concurrency and distribution, of system's integrity, of fault-tolerance, and how the main abstractions from the logical view fit within the process architecture "on which thread of control is an operation for an object actually executed".
3. The Development Architecture: The development architecture focuses on the actual software module organization on the software development environment. The software is packaged in small chunks—program libraries, or *subsystems*; that can be developed by one or a small number of developers. The subsystems are organized in a hierarchy of *layers*, each layer providing a narrow and well-defined interface to the layers above it. The development architecture of the system is represented by module and subsystem diagrams, showing the 'export' and 'import' relationships. The complete development architecture can only be described when all the elements of the software have been identified. It is, however, possible to list the rules that govern the development architecture: partitioning, grouping, visibility.

4. The Physical Architecture:

The physical architecture takes into account primarily the non-functional requirements of

The system such as availability, reliability (fault-tolerance), performance (throughput), and scalability. The software executes on a network of computers, or processing nodes (or just nodes for short). The various elements identified— networks, processes, tasks, and objects need to be mapped onto the various nodes. We expect that several different physical configurations will be used: some for development and testing, others for the deployment of the system for various sites or for different customers. The mapping of the software to the nodes therefore needs to be highly flexible and have a minimal impact on the source code itself.

it can be seen there is a relationship between SIS challenges and 4+1 view models. This relationship that between logical view and the first reason of increasing complexity of software- intensive systems which increase in functionality. So, logical view is concerned about the system functions and it can be seen the main reason of the increasing

Complexity which is increasing the functionality in SIS.

CHAPTER FOUR

(ANALYSIS AND DESIGN)

4-analysis and design

In this chapter we will be touching and explain the analysis and design

of this research and some important paragraphs and the relations among them.

4.1 Analysis the research problem

This research aims to solve one of the Software Intensive System (SIS) challenges. This challenge is "Increasing complexity of software-intensive systems".

There are many criteria can effect on reasons for the increase in the complexity of software intensive systems are, among others:

a) *Increase in functionality*: As observable, e.g., in the automotive or telecommunication industry, the functionality of software increases dramatically.

b) *Increase in quality needs:* The time where the delivery of faulty software or software requiring a significant, not intended adaptation of its environment still satisfied the customer is over.

c) *Mass customization of software:* Customer demands an adaptation of the software-intensive product to their needs – like with other products like cars.

The challenge that will be solved in this research is "Increase in functionality" which is a very important reason for complexity of software-intensive systems.

4.2 Solving the problem

Software architecture deals with the design and implementation of the high-level structure of the software. It is the result of assembling a certain number of architectural elements in some well-chosen forms to satisfy the major functionality and performance requirements of the system, as well as some other, non-functional requirements such as reliability, scalability, portability, and availability. The logical architecture primarily supports the functional requirements what the system should provide in terms of services to its users. The system is decomposed into a set of key abstractions, taken (mostly) from the problem domain. This research will use the logical view to solve the main reason of increasing the complexity in SIS which is the increasing in functionality.

4.2.1 Design of the New Pattern (template)

Now it can be seen that is one the challenges that faces the SIS which is the complexity of functions. After studying the main purpose of the logical view, it can be seen there is a relationship between the challenge and the logical view, especially both logical view and the challenge are concerned about the functions. So, logical supports the functional requirements and what the system should provide in terms of services to its users. So, the new pattern (template) will consist of solving the SIS problem by using the logical view as shown in (figure (4-1.)).

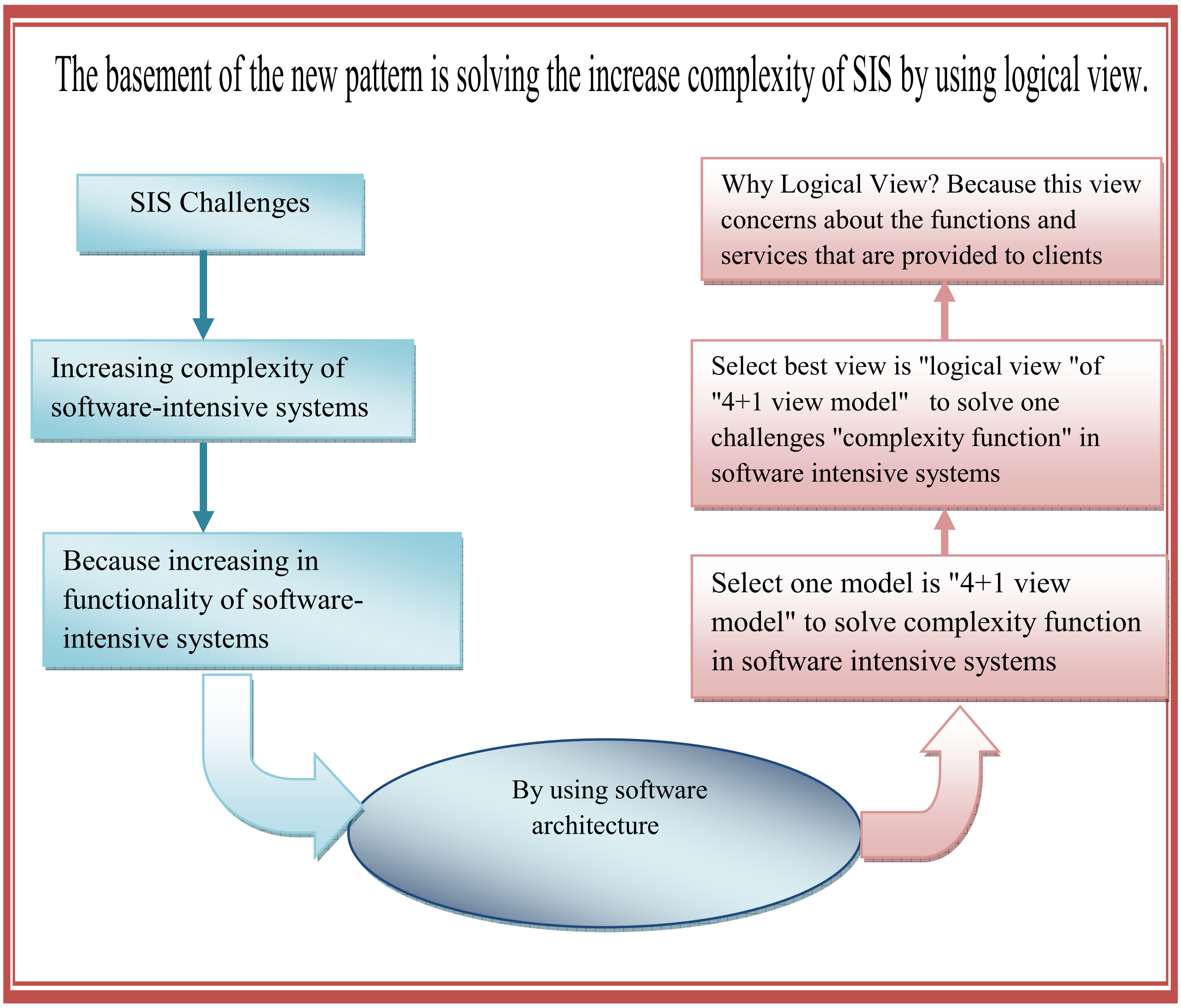

The basement of the new pattern is solving the increase complexity of SIS by using logical view.
SIS Challenges
Increasing complexity of software-intensive systems
Because increasing in functionality of software-intensive systems
By using software architecture
Select one model is "4+1 view model" to solve complexity function in software intensive systems
Select best view is "logical view "of "4+1 view model" to solve one challenges "complexity function" in software intensive systems
Why Logical View? Because this view concerns about the functions and services that are provided to clients

Fig 4-1 General pattern (template) for relationship between SIS and Logical view

4.3 Example of Logical View Modeling by UML

This example will show how to model the logical view via UML. It is a case study for a system to visualize measurements at junctions. Measurements obtained by sensors and the effects of the application of actuators are important quantities to be visualized for decision makers. Thus, having application systems that can visually show numerical values resulting from measurements is fundamental. This case study is about a system that provides visualization of road traffic measurements at junctions controlled by traffic signals. It can be seen as a decision support tool for traffic operators, providing reliable, real-time information of road traffic measurements, which can be useful to improve traffic signals planning. This application will be used as a subsystem for other road traffic management systems. The location where is traffic can change its routes, directions and sometimes even the mode of travel is named Junction as shown in (figure (4-2.).

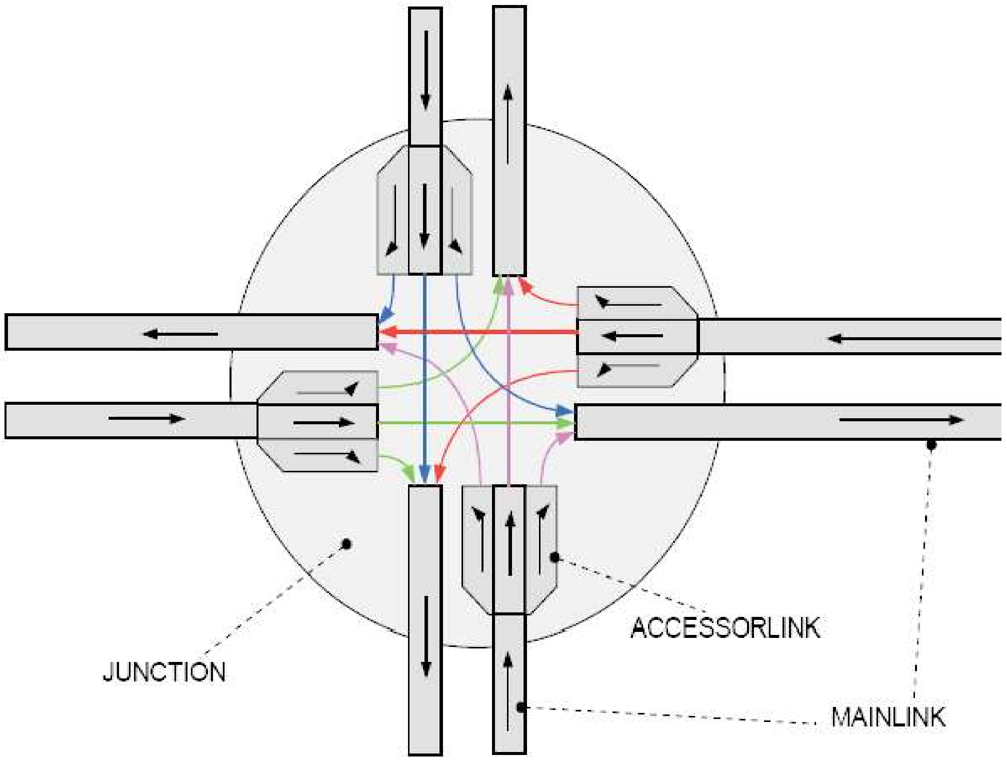


Fig. 4-2 A junction representation (soares et al, 2009)

A junction comprises the outgoing Main link and the incoming Access or links of a crossing or motorway. Each Main link and Access or link only belongs to one specific junction. For each direction of the Access or link (turn left, go ahead, or turn right), information about velocity, waiting time, intensity and density. So each system needs to list all requirements in order to know what the system should provide. In this example, there are many requirements listed as following:

1. For each direction, the user wants to visualize information about split factor and waiting time.
2. The number of directions depends on the number of Access or links.
3. The symbol “-” should be used in the place of numerical values when there is no direction.
4. Junction must have an automatic created image based on Links information;
5. Junction image must be by default on the geographical map (Trinivision).
6. Junction image on the geographical map (Trinivision) must be shown in a small form
7. Clicking in the junction image will result in an overview from this image containing the junction information.
8. The junction image must have an understandable and represent-able layout.
9. The information shown shall be from all Access or links into the junction.
10. The information shall be shown for each traffic stream into the junction.
11. Access or link information should contain the split factor.
12. Information about intensity and waiting time must be shown when there is a Direction with an actuator or sensor from the Access or link.

As it has been explained the main purpose of logical view which is concerns about the functions and services that should the system. So, in this example there are two main functions (junction and direction). (Figure 4-3)) is a class diagram and shows the main functions and the relationships between them.

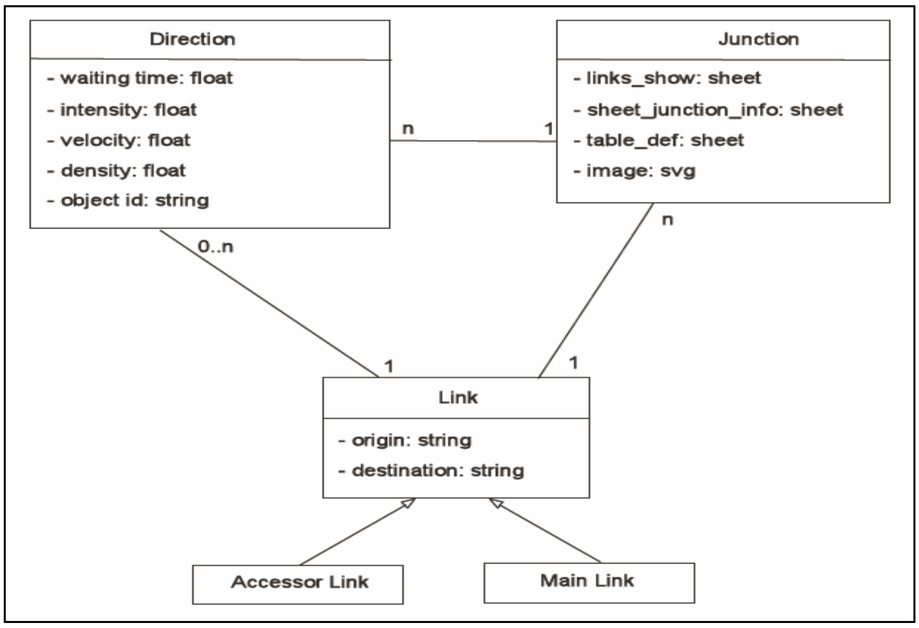


Fig. 4.3 UML class diagram for representing of logical view

CHAPTER FIVE

(SYSTEM MODELLLING,

CASE STUDY MODELING)

5.1 Introduction

The idea of the research will be implemented in this chapter, which is the implementation of software architecture model "4+1 view model "to solve the SIS challenges.   In order to check the efficiency of the new pattern (Template) , architecture the research consider the “Air Traffic Control System” as a case study and implementation issue Components mean the parts of the system depend on the 4+1 view model.(Table (5.1) shows the meaning of components for each view.

Table 5.1: "4+1 architecture model"

View	View Logical	Process	Development	Physical	Scenarios
Components	Class	Task Module	Subsystem	Node Step	Scripts

In this research the logical will be used. So, the components of logical can be seen as:

1- Class.
2- Class Utility
3- Parameterized Class

In order to check the efficiency of the new pattern, “Air Traffic Control System” will be implemented using the logical view as shown in figure (5-1). in this research based on the principle of 4+1 view models. As shown (figure (5-1))

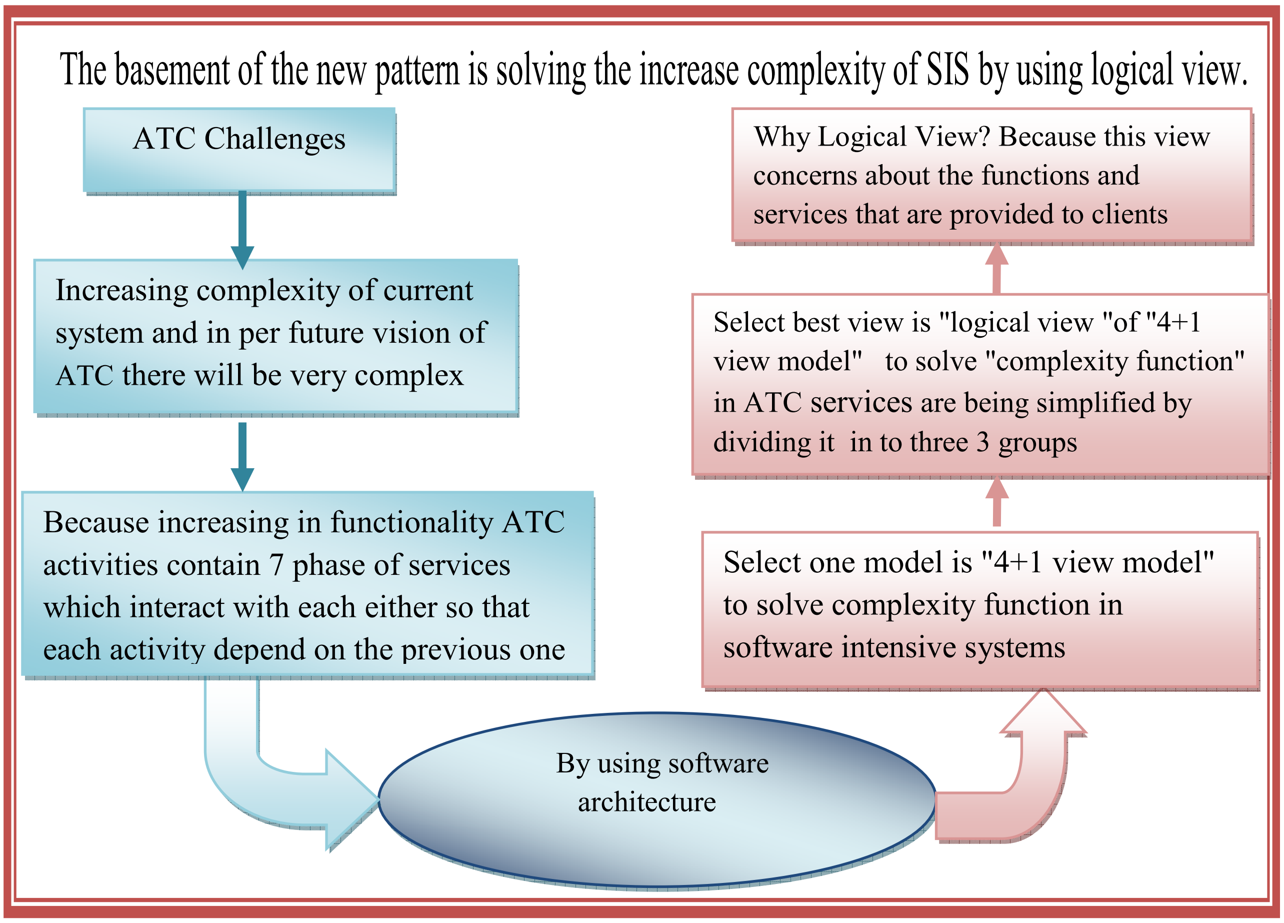

The basement of the new pattern is solving the increase complexity of SIS by using logical view.
ATC Challenges
Increasing complexity of current system and in per future vision of ATC there will be very complex
Because increasing in functionality ATC activities contain 7 phase of services which interact with each either so that each activity depend on the previous one
By using software architecture
Select one model is "4+1 view model" to solve complexity function in software intensive systems
Select best view is "logical view "of "4+1 view model"   to solve "complexity function" in ATC services are being simplified by dividing it in to three 3 groups
Why Logical View? Because this view concerns about the functions and services that are provided to clients

Fig 5-1 ATC by using pattern (template)

5.2 Air Traffic Control (ATC) (Prasad Rahul,2011)

ATC system is a service that gives guidance to airplane, prevents collisions and manages safe and orderly traffic flow. ATC system has been developing since 1945; it is a vast network of people and equipment that ensures safe operation of airplanes. The first air traffic control (ATC) system was originally built in the 1960s; since then, air traffic has increased immensely, and has become increasingly more difficult to maintain safety in the sky. As air travel has become an essential part of modern life, the ATC system has become strained and overworked. The ATC system has been in a process of continuous improvement change. In the earliest days of aviation, few airplanes were in the skies that there was little need for automated control of airplane. As the volume of air traffic increased and the control was still fully manual; the system was considered unsafe as human error has been cited as a major factor in the majority of aviation accidents and incidents. In today's Air Traffic Control system, air traffic controllers are primarily responsible for maintaining airplane separation. Every airplane follows several activities during a flight. The primary role of an air traffic controller is to separate airplane and provide them safe, orderly, and expeditious movement from one place to the next. Air Traffic Controllers are responsible for virtually every bit of airspace around the world. Most of the major facilities are staffed 24 hours a day, seven days a week, 365 days a year. This means air traffic controllers are subject to long hours, shift work, and working on holidays.

Air Traffic Controllers can be employed by various agencies. In the United States, the Federal Aviation Administration (FAA) employees the most controllers, staffing the nations busiest facilities. The Department of Defense, military, and other private contractors are responsible for staffing all of the other facilities. Though the FAA doesn't employ every controller, they oversee and provide the guidelines to which all air traffic controllers in the United States must adhere by. As many new air traffic controllers enter the workforce, more and more are coming from professions outside the aviation industry. The most common facilities a new hire with the Federal Aviation Administration can be assigned are: Center/EnRoute, Tower only, TRACON only, or Tower/TRACON combined. The tower handles the entire airplanes on the airport and immediately surrounding the airport. Once the airplane leaves the boundaries of the tower, the tower controller hands them off to the TRACON. The TRACON is responsible for the area immediately surrounding the airport and extending from 20 to over 100 miles and generally from 10,000 to 15,000 feet. Once the airplane passes those boundaries, they are handed off to the Center/En Route controller. This process repeats itself backwards as the airplane begins its descent to its destination.

5.3 ATC as example of SIS

ATC systems are highly complex pieces of machinery, they employ standard verification and modeling technique to coordinate, distribute and track airplane as well as weather information. The currently used systems need to employ procedures for improved safety and efficiency which include flexibility, potential cost savings and reduction in staffing. As per future vision of air traffic control there will be large number of aircraft and their route will be quite busy and very complex. This means that there is a lack of advanced technology and desire to support the controller. Thus there is a need to build ATC system based on a method which can handle increased air traffic capacity/congestion to provide a safety critical interactive system. The following are the major Obstacle of current ATC system: (Prasad Rahul, 2011)

a) *Lack of well-defined human/software interface* – The idea of full automation or minimum human intervention of the ATC system still remains unfulfilled. The existing systems do require human interaction as the system only guides but actual decision is taken by the controller's in charge (ground, local).
b) *Need for high maintenance* – Maintenance of the system is also an issue which can cause problem as about an incidence in which voice communication between the pilot and controllers broke down and the reason behind this was found to be a lack of maintenance.
c) *Outdated design/technology* – Obsolete software design and programming language are major barriers to upgrades and efficient software maintenance of the currently used ATC systems because of which improved capacity and efficiency can't be achieved with the current system. The current computer software limits the number of airplane that can be tracked at any given time, and the dated architecture makes enhancements, troubleshooting and maintenance more difficult. Computer outages, planned or unplanned, are covered by a backup system that cannot handle the same level of air traffic as the main system. The result is significantly limited capacity during backup mode.
d) *Mixed communication* – The communication between the controllers and pilot currently is a combination of voice and data link. The results of test conducted show that the mixed communication leads to slow speed which can be overcome only when the whole communication takes place in a well defined manner.

5.4 ATC system services

There are <u>seven</u> activities for this system. (figure (5-2)) shows these seven activities.

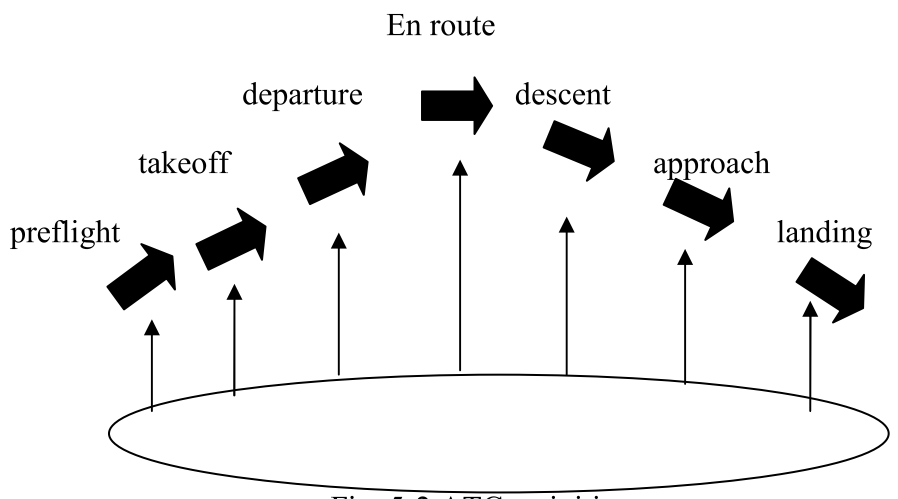


Fig. 5-2 ATC activities

5.4.1 Activities should be controlled

ATC system will control each main activity for the plane starting from preflight to landing. So the main function for this system is preventing accidents happen in the air traffic. (lubos ,2008)

These activities for ACT are explain below, figure 5.2 shows these activities

a) Preflight -This portion of the flight starts on the ground and includes flight checks, push-back from the gate and taxi to the runway.

b) Takeoff - The pilot powers up the airplane and speeds down the runway.

c) Departure - The plane lifts off the ground and climbs to a cruising altitude.

d) En route - The airplane travels through one or more center airspaces and nears the destination airport.

e) Descent - The pilot descends and maneuvers the airplane to the destination airport.

f) Approach - The pilot aligns the airplane with the designated landing runway.

g) Landing - The airplane lands on the designated runway, taxis to the destination gate and parks at the terminal.

5.4.2 Grouping ATC Activities

ATC system activities aim to provide control on three main services:

1- Airplane departure control service

This service will include the following activities:

a) Preflight

b) Takeoff

c) Departure

2- Airplane en route control service

3- Airplane landing control service

This service will include the following activities:

a) Descent
b) Approach
c) Landing

(Figure (5-3)) shows grouping activities in three main services for ATC system

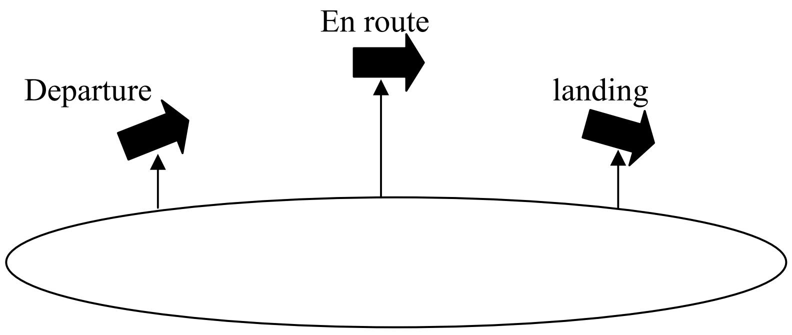


Fig. 5-3 ATC services simplified by grouping

These services will be the objects of system classes and they will be modeled using UML in order to provide the main functions. Logical view focuses on the services for any system that will provide them for system users. In this chapter, ATC will be designed using 4+1 view model in order to implement the system without error and failure.

5.5 Use case ATC:

Now will explain the use case and the scenario of the system and to clarify all the functions of the system and the results of this work, Show figure (5-4).

Shows Use Case Model diagram for all function the ATC.

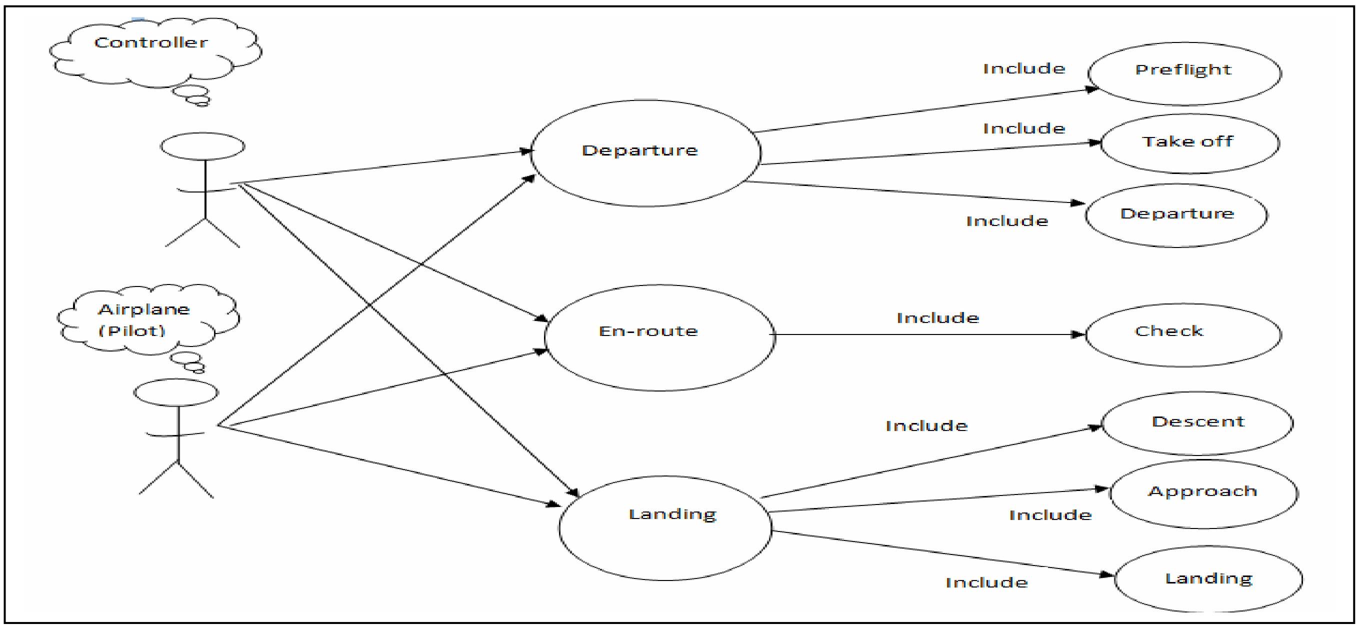


Fig 5-4 UML Use Case Model on all function the system

5.6 Design the system using 4+1 view model in UML

The ATC systems are quite complex and inefficient. To adapt to the changing demands of speed and efficiency a reliable software system for ATC is required to be developed. Software architecture based on UML models will help in handling complexities and drawbacks of existing ATC systems and also help to better understand the domain. The complexity of the problem domain requires extensive efforts for the clarification of the initial problem statement. Moreover, due to the extremely long lifespan of ATC systems, stable and robust analysis models enabling the integration of new operational scenarios are needed which can be efficiently obtained using UML models. The diagrams obtained using the UML models are highly optimized which is one of the main requirements for the design of a cooperative ATC system.

5.7 Logical View and ATC Modeling

In order to implement 4+1 view model of logical view on the ATC, all services of ATC should be placed. Logical view in the 4+1 view model will primarily supports the functional requirements of Air Traffic Control system which means what the ATC system should provide in terms of services to controller and airplane. The system is decomposed into a set of key abstractions, taken from the air traffic control domain, in the form of *objects* or *object classes* for ATC services. The principles of abstraction, encapsulation, and inheritance will be exploited based on ATC requirements. This decomposition of ATC is not only for the sake of functional analysis, but also serves to identify common mechanisms and design elements across the various parts of the ATC system.

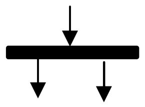
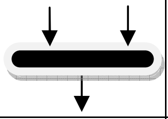
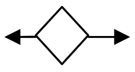
5.7.1 Modeling Departure Control Services

In order to model this service, many of UML symbols will be used to present the actors, process and relationships.

-The following is a description in the introduction for each notation which will be used in presenting Departure, En route, landing services in ATC.

(Table 5.2 notation)

Number	Notation	Description
1-	●	**Initial node**. The filled circle is the starting point of the diagram. An initial node isn't required although it does make it significantly easier to read the diagram.
2-	◉	**Activity final node**. The filled circle with a blod border is the ending point. An activity diagram can have zero or more activity final nodes.
3-	▭	**Activity**. The rounded rectangles represent activities that occur. An activity may be physical, such as *Inspect Forms*, or electronic, such as *Display Create Student Screen*.
4-	→	**Flow/edge**. The arrows on the diagram. Although there is a subtle difference between flows and edges I have never seen a practical purpose for the

		difference although I have no doubt one exists. I'll use the term flow.
5-		**A fork node**. Is a control node that splits a flow into multiple concurrent flows.
6-		**A join node**. Is a control node that synchronizes multiple flows.
7-		**Branches -** These divide the sequence into several alternatives specified by different conditions (guards).

Based on the system requirements of the ATC, all services that should provide to their users be molded, logical view will concern each class and activity should be involved. The following activity should be presented as classes in order to provide this service.

There are two main actors for providing this service which is "airplane departs" from pilot to the controller. Actors and process will be presented based on UML standard.

1st Step: determine the two users (actors) for this system:

Airplane (Pilot)
Controller

Relations between actors will be presented as Arrow

2nd Step: airplane (pilot) send request for controller to get permission for pushback clearance.

3rd step: controller will give the airplane a pushback clearance.

4th step: craft will pushback from gate **then** it will leave the ramp area. (Hint: **then** here means process after process)

5th step: airplane will request from controller the clearance taxi permission.

6th step: airplane taxing

7th Step: airplane requests a departing permission from controller

8th Step: Controller gives airplane departure clearance.

9th Step: airplane departs.

(figure (5-5)) shows UML activity diagram of departure services

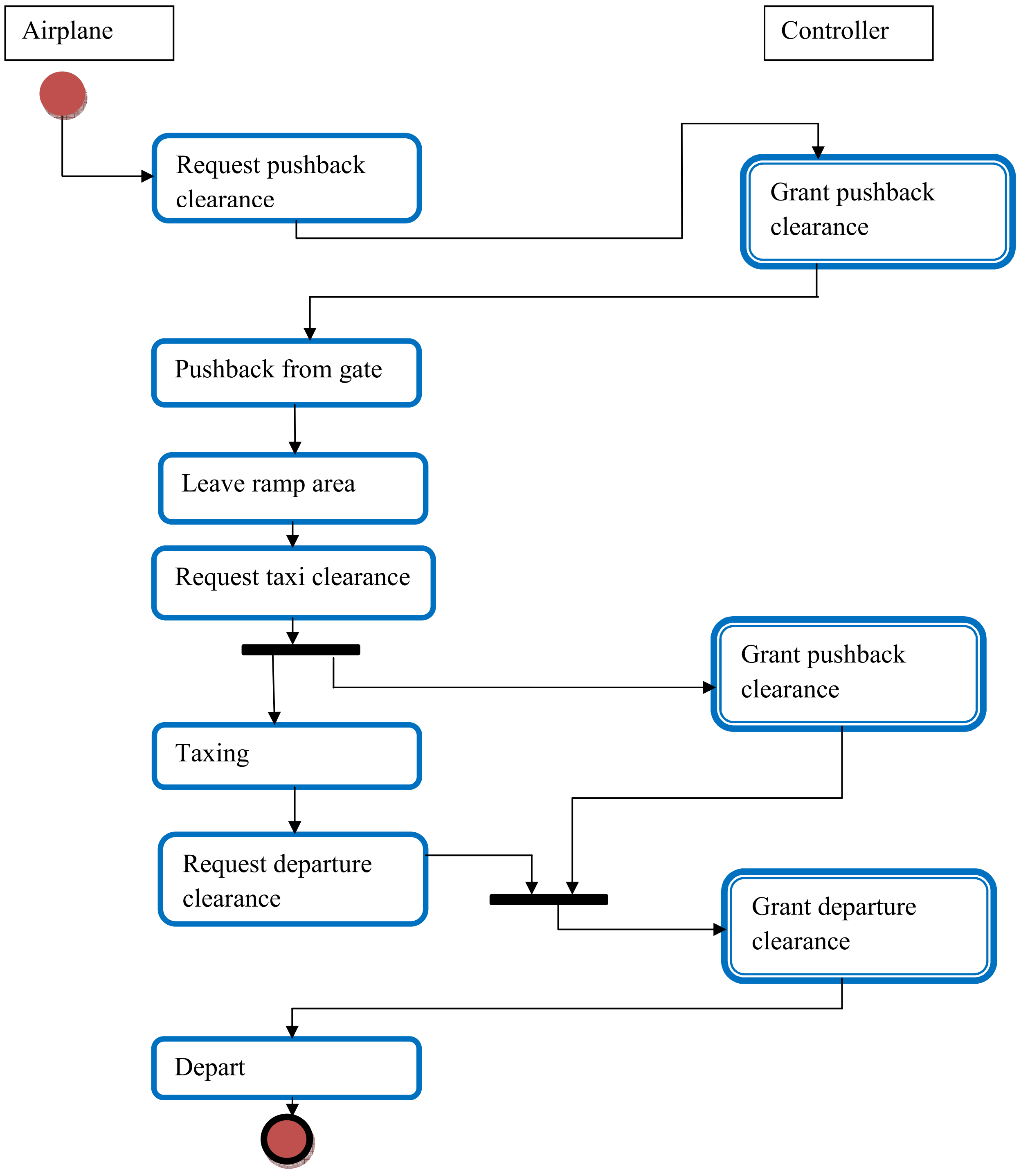


Fig. 5-5 UML activities for departure control service

Consequence process of Departure Service

While ATC system is SIS contains complex function varied with many circumstances, it shouldn't has no chance to happen any error or accident in the system. There are 5 actors will be involved in the departure service. Each actor will communicate with airplane to insure the fight departures safely without accidents.

The following section will show the relationships between airplane and others actors.

1- Gate Controller with Airplane.
2- Ramp Controller with Airplane.
3- Ground Controller with Airplane.
4- Local Controller with Airplane.

Subsystem 1:

This will include the airplane and gate controller. ATC will make airplane communicate the gate controller to assign which gate will the airplane takes.

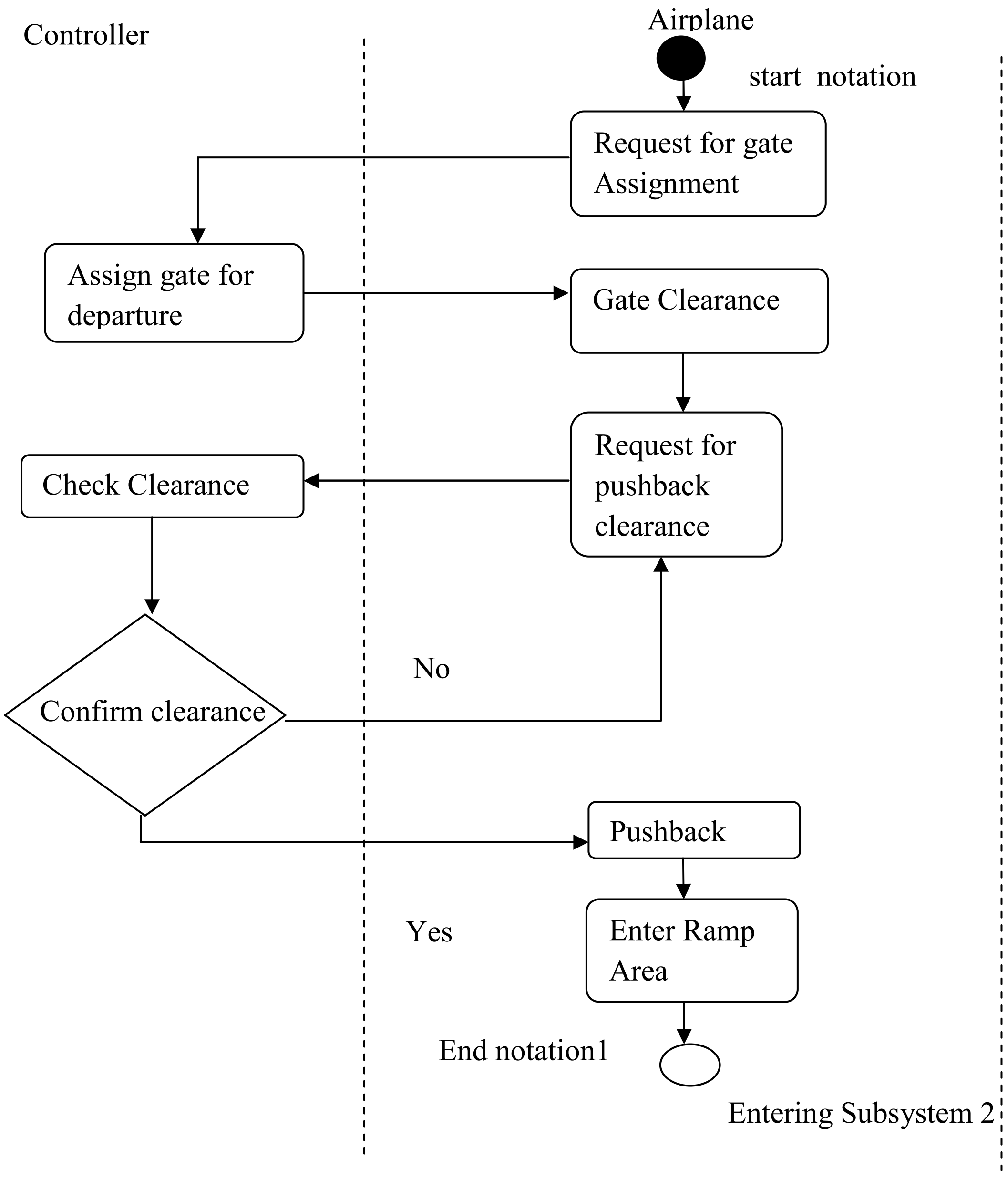


Fig. 5-6 communicate the gate controller

Subsystem 2

This subsystem will take over the next step of ATC system that will control the communications and functions between airplane and ramp and guarantee

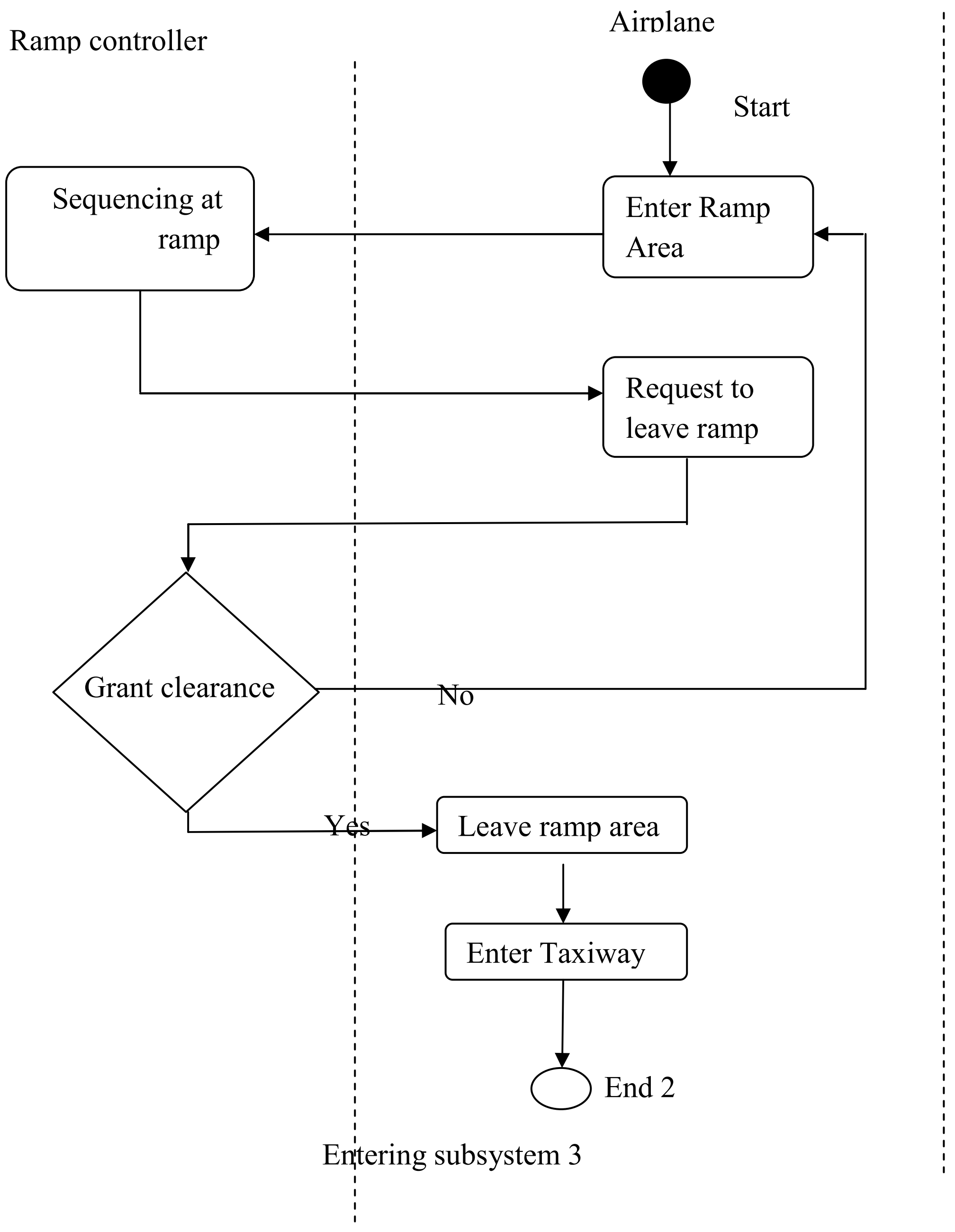


Fig.5-7 communicate the ramp controller

Subsystem 3

The main function for this subsystem is controlling the communication between airplane and both of ground controller and local controller in order to provide the information to airplane, at the end of this subsystem, the airplane will be processed safely to be on runway.

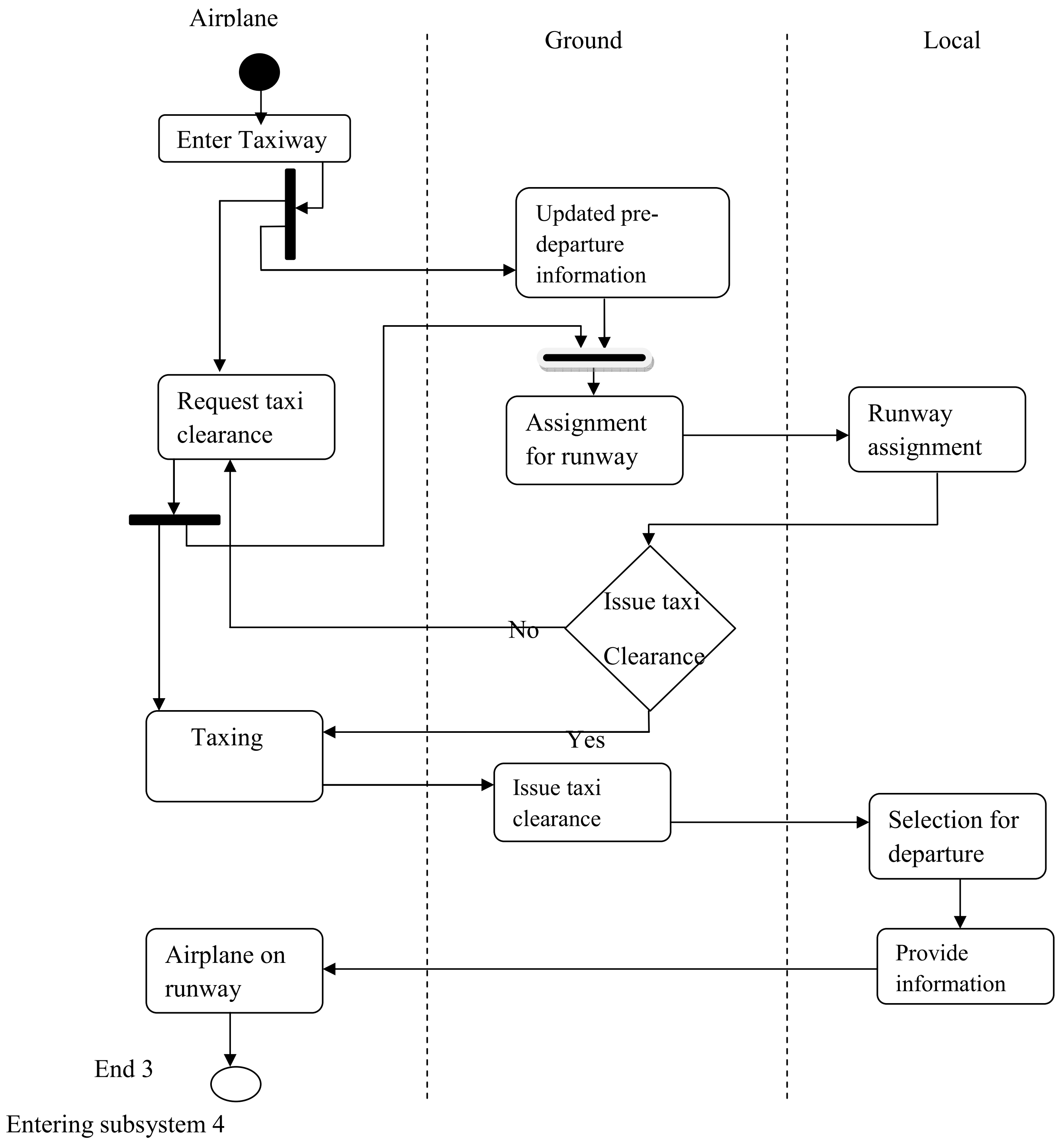


Fig. 5-8 communications both of ground controller and local controller

Subsystem 4

It will detect, track and display the progress of airplane in the runway and start departure. This subsystem will include airplane, ground controller and local controller.

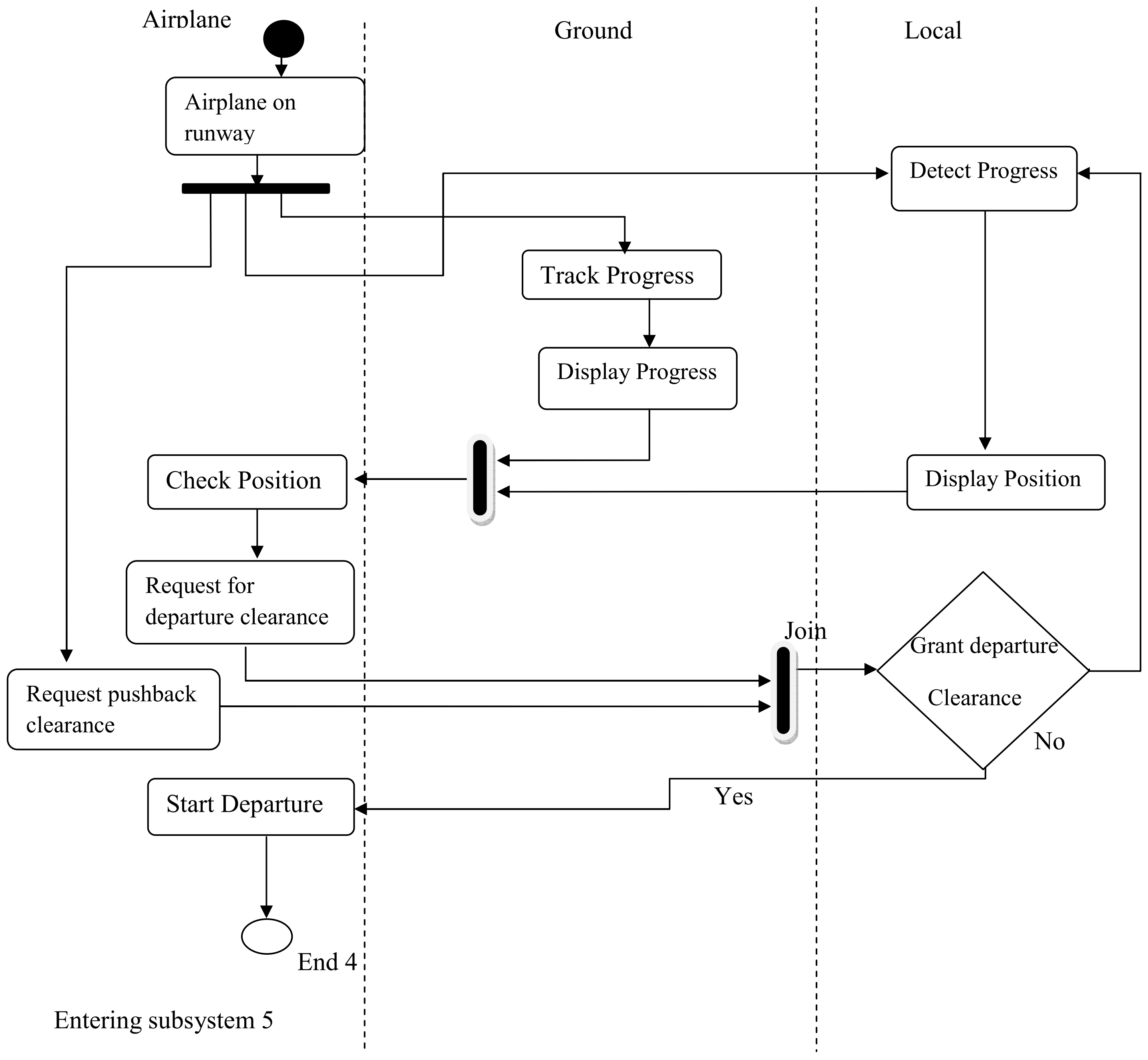


Fig.5-9 detect track in the runway and start departure.

Subsystem 5

It is final phase for departure. It will detect if there is another flight will departure or conflict. If there is any conflict instructs controller or pilot himself will decide what to do when conflict alert arises. At the end of this process the fight will departure safely. If there is any conflict alert, the pilot can send alert in emergency condition for local controller.

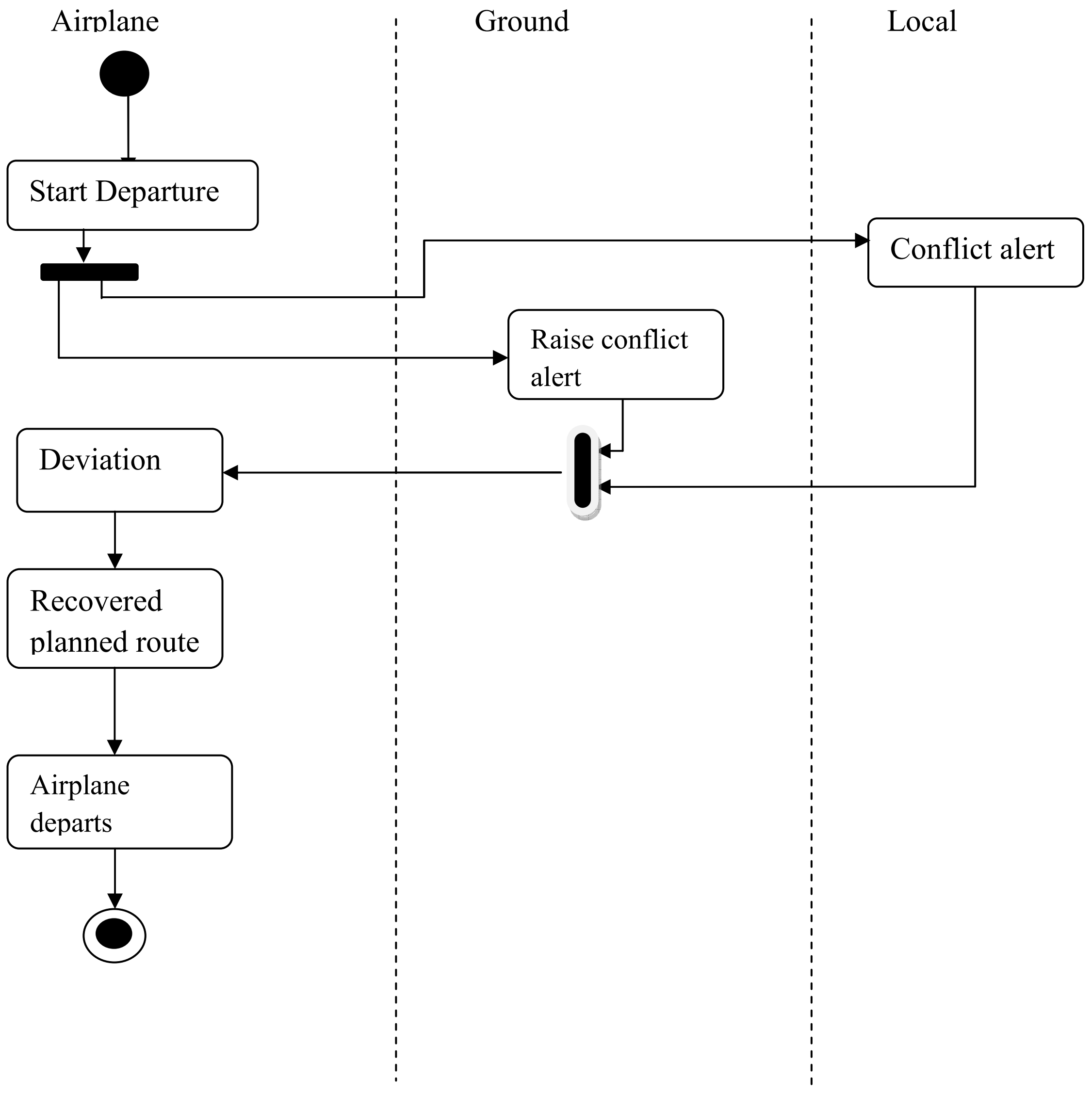


Fig.5-10 Final phase for departure.

5.7.2 Modeling En-Route Control Services

After each plane departs, terminal controllers notify en route controllers, who take charge next.

Airplanes usually fly along designated routes; which center is assigned a certain airspace containing many different routes. En route controllers work either individually or in teams of two, depending on how heavy traffic is; each team is responsible for a sector of the center's airspace. As the plane proceeds on its flight plan to its destination it is handed off from sector to sector both within the center and to adjoining centers. To prepare for planes about to enter the team's sector, the radar associate controller organizes flight plans output from a printer into strip bays. If two planes are scheduled to enter the team's sector in conflict, the controller may arrange with the preceding sector unit for one plane to change its flight path or altitude. As a plane approaches a team's airspace, the radar controller accepts responsibility for the plane from the previous sector. The controller also delegates responsibility for the plane to the next sector when the plane leaves the team's airspace. Based on the requirement of this service, the following steps will show how to control this service:

1st step: Airplane flies along designated routes

2nd step: terminal controller will notify en route controller

3rd step: airplane sends strip bay to controller based on the designated routes

4th step: if there is conflict between two airplanes, controller will arrange with the preceding sector unit for one plane to change its flight path or altitude.

All these steps will provide control for En route service. (Figure (5.11)) shows the activity diagram for controlling en route service.

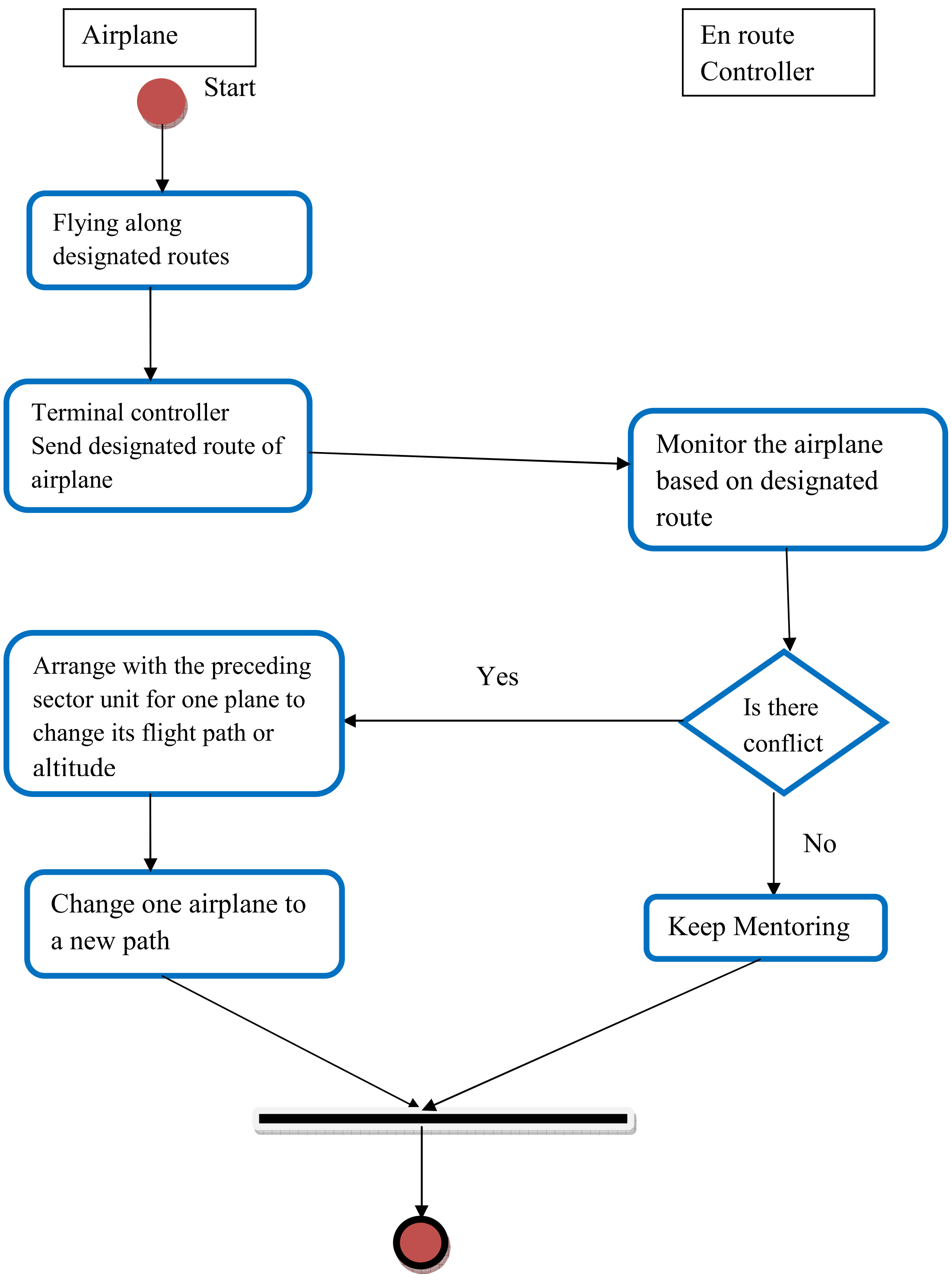

Airplane
Start
En route Controller
Flying along designated routes
Terminal controller Send designated route of airplane
Monitor the airplane based on designated route
Is there conflict
Yes
Arrange with the preceding sector unit for one plane to change its flight path or altitude
No
Change one airplane to a new path
Keep Mentoring

Fig 5-11 UML activity diagram of en route service controlling

5.7.3 Modeling Landing Control Services

When the plane is approximately 50 miles from the destination airport, it is handed off to that airport's terminal radar arrival controller who sequences it with other arrivals, and issues an approach clearance. As the plane nears the runway, the pilot is issued a clearance to contact the tower. The local controller issues the landing clearance. Once the plane has landed, the ground controller directs it along the taxiways to its assigned gate. The local and ground controllers usually work entirely by sight, but may use airport surface radar if visibility is very poor. Both airport tower and en route controllers usually control several planes at a time, often making quick decisions about completely different activities. For example, a controller might direct a plane on its landing approach and at the same time provide pilots entering the airport's airspace with information about conditions at the airport. While instructing these pilots, the controller also might observe other planes in the vicinity, such as those in a holding pattern waiting for permission to land, to ensure that they remain well separated. In addition to airport towers and en route centers, air traffic controllers also work in flight service stations at 17 locations in Alaska. These flight service specialists provide pilots with preflight and in-flight weather information, suggested routes, and other aeronautical information important to the safety of a flight. Flight service specialists relay air traffic control clearances to pilots not in direct communications with a tower or center, assist pilots in emergency situations, and initiate and coordinate searches for missing or overdue airplane. At certain locations where there is no airport tower or the tower has closed for the day, flight service specialists provide airport advisory services to landing and departing airplane. However, they are not involved in actively managing and separating air traffic.

The following steps show how to control the landing service

1st Step: airplane informs the controller of airport destination at 50 miles away and then request for approach clearance.

2nd Step: controller provides airplane with approach clearance.

3rd Step: pilot is issued a landing clearance.

4th Step: controller provides landing clearance for the airplane.

5th Step: pilot request taxi clearance

6th Step: controller provides the clearance and directs it along the taxiways to its assigned gate.

(Figure (5-12)) shows UML activity diagram of landing services

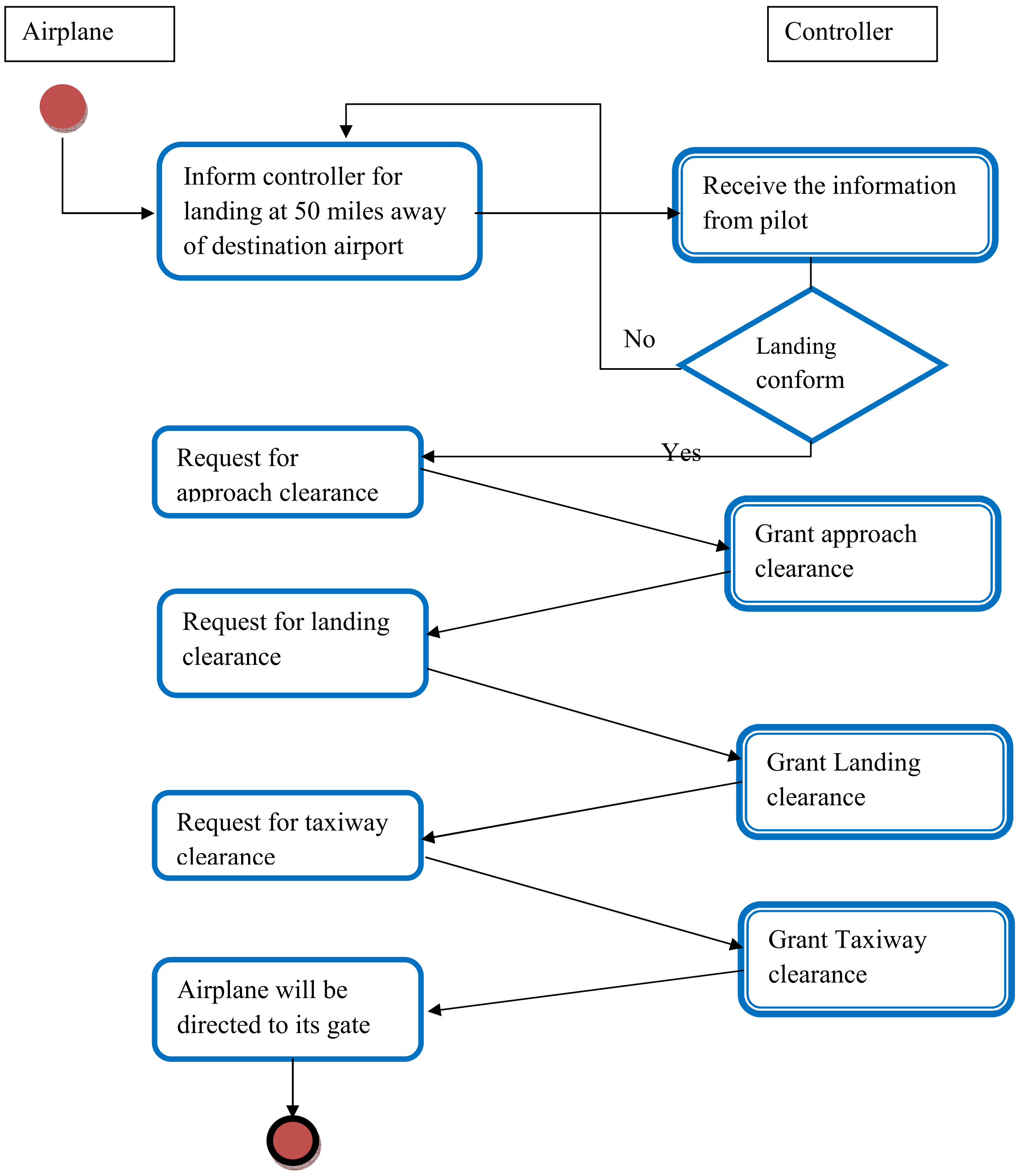

Airplane
Controller
Inform controller for landing at 50 miles away of destination airport
Receive the information from pilot
Landing conform
No
Yes
Request for approach clearance
Grant approach clearance
Request for landing clearance
Grant Landing clearance
Request for taxiway clearance
Grant Taxiway clearance
Airplane will be directed to its gate

Fig. 5-12 UML activities for landing control service.

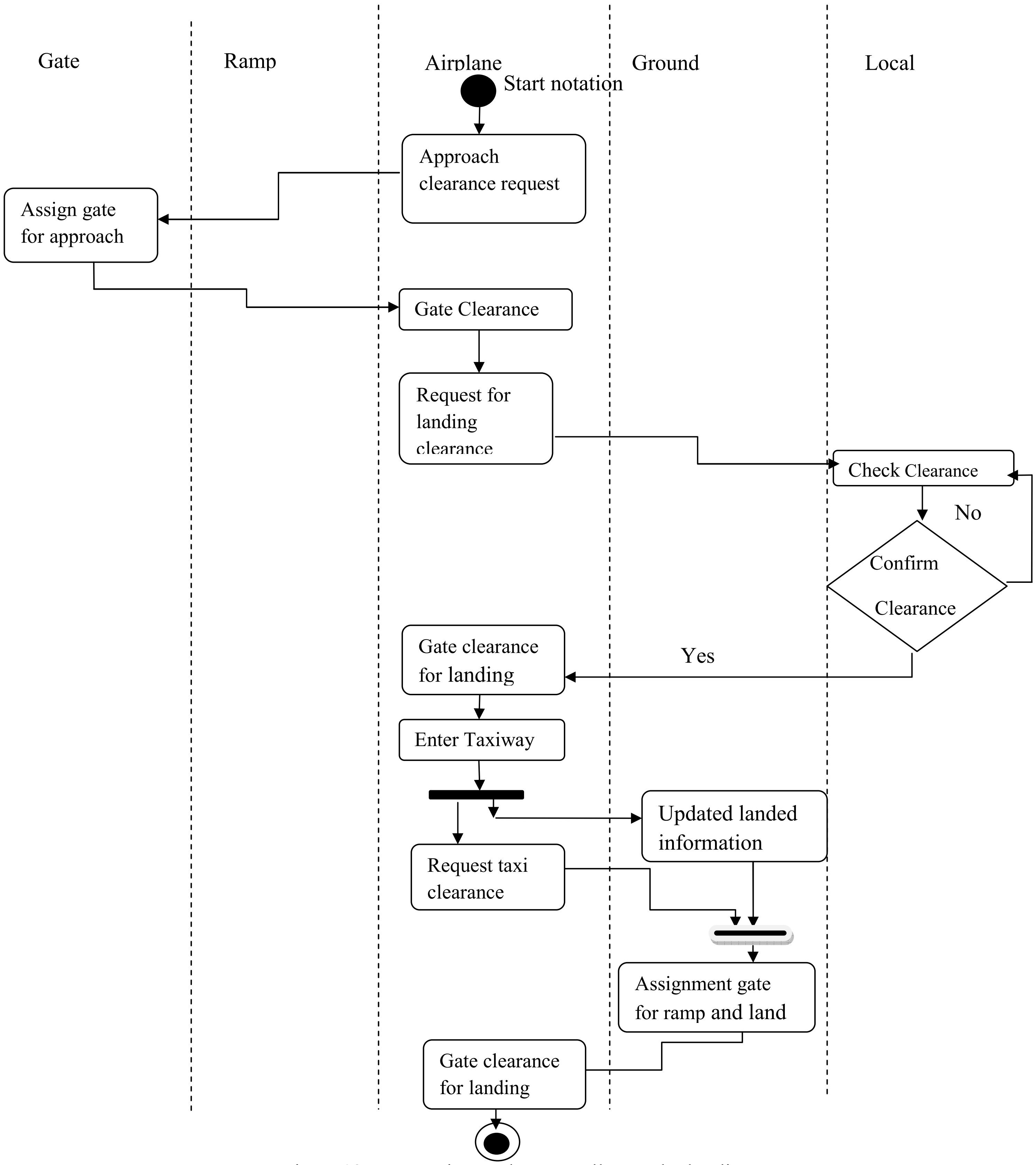

Gate
Ramp
Airplane
Ground
Local
Start notation
Approach clearance request
Assign gate for approach
Gate Clearance
Request for landing clearance
Check Clearance
No
Confirm Clearance
Yes
Gate clearance for landing
Enter Taxiway
Updated landed information
Request taxi clearance
Assignment gate for ramp and land
Gate clearance for landing

Fig. 5-13 communicates the controller on the landing.

If two planes request landing clearance, what the system should do?

In this case, the plane which is moving faster than the other will get the clearance. This condition concerned about the speed for each plane. The other plane will slow down till the first plane lands then get the clearance, The result in the final will be according to existing condition the controller on the takeoff and landing in system .

5.8 ATC UML Class Diagram

Class diagram identifies and describes the static structure of the system i.e. the system architecture. The purpose of a class diagram in this research is to depict the classes within ATC model. ATC classes have attributes (member variables), operations (member functions) and relationships with other classes. (Figure 5-14)) Shows UML class diagram

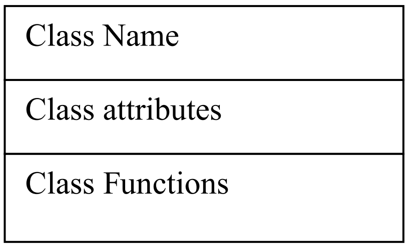


Fig 5-14 UML Class Diagram

Implemented class diagram for ATC system means there are two classes

1- Controller
2- Airplane

Each class will contains variables and functions, these classes integrated together in order to provide departure control service.

5.8.1 Controller UML Class Diagram

A-Controller class members will consist of:
1-Controller name
2-Location
3-Area

B-Controller class functions will consist of:
1- monitor
2- grant clearance
3- tracking
4- movement handling

5.8.2 Airplane UML Class Diagram

A. Airplane class members will consist of:
1- Airplane name
2. Airline name
3. Departure time

B. Airplane class functions will consist of:
1- Push back
2-Taxi
3-Depart

After determining each class members and functions now we can model the class diagram by UML and the relationships between them and it is this generalization type as Shown in (figure (5-15.)) This diagram will help the developer and programs to make basement for implementing this services.

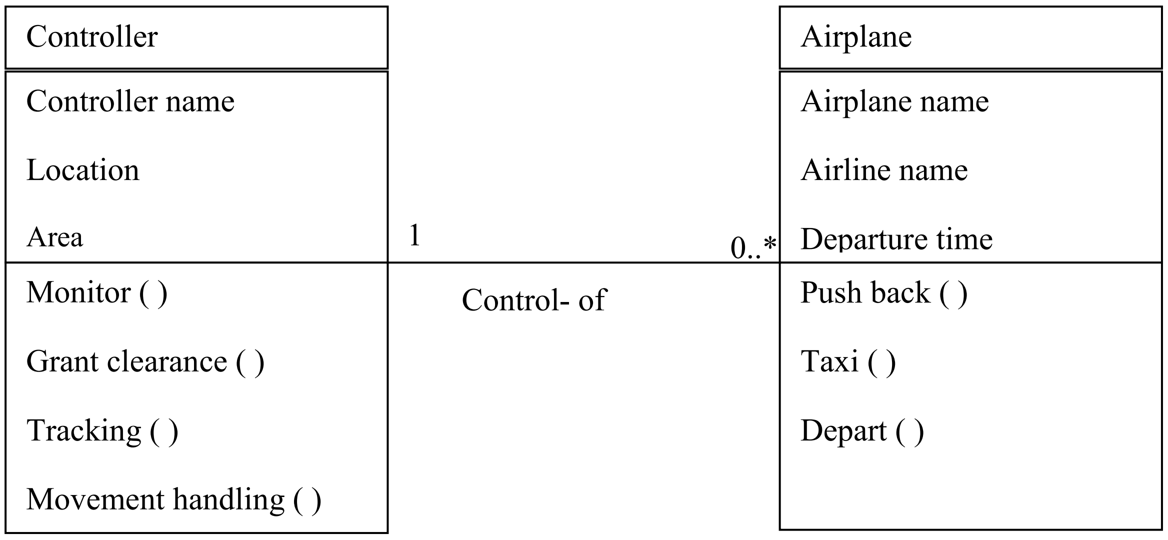


Fig. 5-15 Departure UML Class Diagram

5.8.3 Class diagram for ATC departure service

In order to achieve this purpose, we have to show all attributes for each UML class diagram to be complete and all class compatible with each other. By doing this strategy, SIS will be more efficient and easy to implement because all attributes are fully defined in independent classes. Each class contains its main name, attributes and functions. In addition, there define for each function such as long, string and Boolean. There will be no more complex in the process of implementing SIS such as ATC.

The main advantage of describe the type of each variable is preventing the error and conflict between the classes in the future. As it’s be mentioned before, logical view will decompose the ATC system into a set of key abstractions, in the form of *objects* or *object classes*. After we figured out the main classes for departure service of ATC, it can be easy to involve logical view concept and implement the software architecture model.

(Figure (5-16)) shows UML class diagram for ATC departure service.

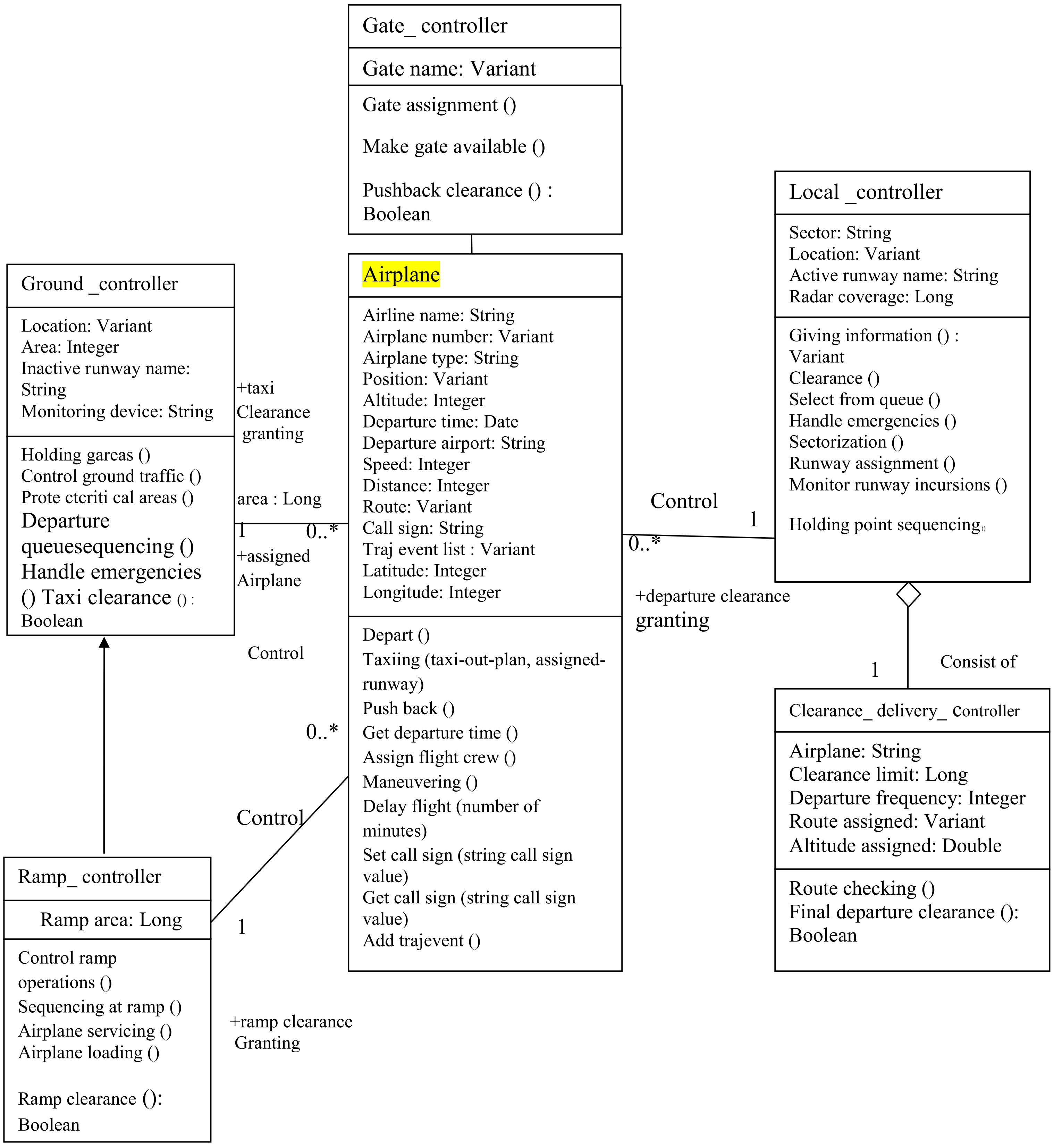


Fig. 5-16 UML class diagram of departure service

5.9 Logical View to Process View

The process architecture takes into account some non-functional requirements of ATC system, such as performance and availability.

It addresses issues of:

a) ATC concurrency and distribution.
b) ATC system's integrity.
c) ATC fault-tolerance.

The main point important of process view is addressing how the main abstractions from the logical view fit within the process architecture on which thread of control is an operation for an object actually executed. The following section shows the addressed issues. Bad weather conditions like wind, rain and snow can pose threats for planes ATC and shows some faults between airplane and controller. Terminal controllers keep pilots informed about weather and runway conditions.

5.9.1 Process view fit logical view abstraction

Services of ATC are addressed and the process that involved in these services is located between airplane controller (pilot) and local controller (tower). For example logical view addressed the departure service and the process involved in this abstraction is how to make communication between controller with airplane controller and with the others airplane controllers while there are many airplanes departure at the same time. Other service is the controller are immediate concern is safety, but controllers also must direct planes efficiently to minimize delays. So the process will be executed immediately. Some regulate airport traffic through designated airspaces; others regulate airport arrivals and departures. Terminal controllers watch over all planes traveling in an airport's airspace. Their main responsibility is to organize the flow of Airplane into and out of the airport. They work in either the control tower or the terminal radar approach control room or building. Relying on visual observation, the tower local controllers sequence arrival Airplane for landing and issue departure clearances for those departing from the airport. Other controllers in the tower control the movement of Airplane on the taxiways, handle flight data, and provide flight plan clearances. Terminal radar controllers manage Airplane departing from or arriving to an airport by monitoring each Airplane's movement on radar to ensure that a safe distance is maintained between all Airplane under their control.

In order to accomplish the goals of safety, efficiency, and cost-effective operation, the present ATC system offers the following services to the aviation community:

1- *Separation* assurance: tracking Airplane in flight, primarily with surveillance radars on the ground and airborne transponders, in order to ensure that adequate separation is maintained and to detect and resolve conflicts as they arise;

2-navigation aids —maintaining a system of defined airways and aids to navigation and establishing procedures for their use

3-weather and flight information— informing users of the conditions that may be expected along the intended route so they may plan a safe and efficient flight;

4-Traffic management-processing and comparing the flight plans, distributing flight plans to allow controllers to keep track of intended routes and anticipate potential conflicts, and ensuring the smooth and efficient flow of traffic in order to minimize costly congestion and delays.

5-Landing services-operating airport control towers; instrument landing systems, and other aids that facilitate the movement of air traffic in the vicinity of airports and runways, particularly during peak periods or bad weather that might affect safety or capacity.

These services together comprise an integrated program, no part of which can be fully effective without the others. Flight plans must take into account weather and traffic, for instance, and traffic must be routed to destinations so that it arrives on time and can be handled at the airport with a minimum of delay. Similarly, clearances have to be modified so that traffic can be routed around severe weather or away from bottlenecks that develop in the system. In a practical sense, the aircrew and ground controllers cooperate as a team using various human and electronic resources to maintain safety and to move traffic expeditiously. While the ultimate responsibility for safety of flight rests with the pilot, he remains dependent in many ways on data or decisions from the ground.

CHAPTER SIX

EVALUATION

6.1 System Evaluation

Table 6-1 shows the results of systems evaluation by compare the function of the current research with previous works. Table 6-1 "Table Evaluate Comparison"

	FUNCTION	Current Research	Others
1	Pattern (template) design	Yes (create new pattern (template) based on 4+1view model , Ch4	No (they did not use pattern (template)) and Sys ML extended UML**(soares ,vranken 2009)**
2	Complexity reduction	Yes (simplified enhancement on architecture by using pattern template and enter 4+1 view model decrease function)	(only description) (soares ,vranken 2009) and (kruchten 1995)
3	Construction simplification	Yes (simplified to be grouping 3 phases in ATC activities) Ch5	No (It is more complicated by using 7 phases in ATC activities) (lubos 2008)
4	Software Engineering Approach	Yes (standard work phases are integrated through 4 +1 view to produce pattern (template)	(Standard work phases-description only) (soares ,vranken 2009) and (kruchten 1995)
5	synchronization	Yes (Full synchronized) As using activates diagram in functionality at using activities diagram ,so that will be concurrent interaction between multi function .	(Not fully Synchronized) (lubos 2008) ,(Vipin ,Saxena 2009)
6	Automatic	Full Automatic (ATC) (no human interaction)	Semi- Automatic (existing of human interaction) **(Vipin Saxena 2009)**

7	Development	Yes (Full development of integration among all activities in ATC system)	No (one function is covered in the whole ATC system) (Vipin,Saxena 2009)(lubos 2008) glenn(2003)
8	Flexibility	Yes (enables the implementation to achieve executable code , such that it can be customized to any environment and technology)	(No pattern template is developed to achieve executable code) (soares ,vranken 2009) and (Vipin Saxena 2009) and (kruchten 1995)
9	Advanced Approach	Yes (the benefit of using Pattern (template) architecture is to solve the challenges in SIS)	No (their approach does not solve the challenges in SIS) (soares ,vranken 2009) and (Vipin Saxena 2009) and (kruchten 1995) and (Tiako 2008)
10	Cost Reduction of SIS	Yes(by providing adequate methodology and tools)	(No methodology provided) (Tiako 2008)

CHAPTER SEVEN

CONCLUSIONS AND FUTURE WORKS

7-1 CONCLUSION:

7.1.1- The current proposed research, we produced a new pattern (template) that solved one of the challenges in SIS (software intensive system) which is the increasing complexity. Show (Figure (4-1)).

Through using the proposed pattern (template) we can enhance the architecture model "the 4 +1 view model" to provide an enhancement of how to design and implement SIS, besides having a flexible approach to be adapted in SIS.

1.1- Monitoring and Control of Large Systems or Environments: These applications are characterized by the need to collect data from environment which can also be partially control the environment. For example: Systems controlling such as ATC monitoring and check.

1.2- Integration and composition of highly complex software systems: These applications are characterized by the need to adapt in response to unforeseen changes of requirements, technology, or environment, and to integrate highly dynamic unpredictable diverse knowledge as the integration of the functions of ATC activities.

7.1.2- The new pattern (template) enhance the 4+1 view model by providing it a new feature that allows to design and implement tow type (1.1 and 1.2) that are mentioned above of software intensive system like ATC.

7.1.3 -In order to prove the proposed pattern (template), we applied the use case diagram, activity diagram and class diagram, knowing that the Unified modeling language (UML) is involved in this research because it will be the official modeling language in the use case.

7.2 Future Works:

Our work can be extended and developed in future to:

1-Study and analysis the other reasons (Mass customization of software, Increase in quality needs, Differentiation between the constructions, in–cooperation of variability at all levels) that affect the complexity of SIS.

2-Increasing the development and the enhancement on 4+1 view model architecture by focusing the studies and making deep analysis on the other views (Process view, Physical view and Development view).

3- The proposed pattern (template) can be modified and customized to be reused in new technologies such as Cloud Computing and applied in institution of higher education.

4- Developing a new pattern (template) that combines and integrates more than one plane, so that we can create an integrated and synchronized pattern (template) for multiple planes in the same airport. Further, we can we can have an integrated and synchronized pattern (template) between different ATC systems in multi-airports among the world.

References

- André, C., Mallet, F., de Simone, R.,(2007) Modeling Time(s), Proceedings of the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML), USA. Springer Verlag, LNCS 4735, pp. 559-573(2007).

- Anda, B., Hansen, K., Gullesen, I., Thorsen, H.K.(2006) “Experiences from introducing UML-based development in a large safety-critical project”, Empirical Software Engineering 11(4), 555-581.(2006).

- Booch, G., (2007). “The Economics of Architecture- First”. IEEE Software, 24(5) 18-20(2007) .

- Boehm, B. and Valerdi, R., (2011) Impact of Software Resource Estimation Research on Practice: Achievements, Synergies, and Challenges.

-Buettner, Douglas John, (2008), **designing an optimal software intensive system acquisition: A game theoretic approach**, ProQuest Dissertations and Theses; Thesis (Ph.D.)--University of Southern California, 2008. Publication Number: AAI3341575; ISBN: 9780549969211; Source: Dissertation Abstracts International, Volume: 69-12, Section: B, page: 7639; 391 p.

- Booch, G. (1993). *Object-oriented Analysis and Design with Applications* (2nd ed. Ed.). Redwood City: Benjamin Cummings. (*1993)*

- Clements, P., Garlan, D., Bass, L. Stafford, J., Nord, R., Ivers, J. Little, R. (2002), *Documenting Software Architectures: Views and Beyond*, Pearson Education.

- Dobing, B., Parsons, J., (2006). “How UML is used”. Communications of the ACM 49(5), 109-113(2006).

- Harel, D., Rumpe, B., (2004) “Modeling languages: Syntax, semantics and all that stuff (or, what's the semantics of "semantics"?)”. , *IEEE Software* (2004).

- Henderson-Sellers, B., Cook, S., Mellor, S., Miller, J., Selic, B. (2005). “ UML the Good, the Bad or the Ugly? Perspectives from a panel of experts”. Software and Systems Modeling 4(1) 4-13(2005).

- ISO/IEC, ISO/IEC 42010: (2007). Systems and Software Engineering - Recommended practice for architectural description of software-intensive systems, (2007).

- Glenn brown, (2003)"remote intelligent Air traffic control systems for non-controlled Airports" thesis of Griffith University, 29 January 2003.

-Kruchten P., (1995). " The 4+1 View Model of Architecture", IEEE Software 12(6) 42-50, November (1995).

- Kruchten, P., (2003).The Rational Unified Process: An Introduction, Addison-Wesley Longman Publishing Co., Inc., Boston, MA(2003).

- Hwang, K., (2008)"from grids and p2p to clouds", The 3rd International Conference on Grid and Pervasive Computing – gpc -workshops.

- Kruchten, P. (1995), "The 4+1 View Model of architecture", published by IEEE Software, 1995, vol. 12, iss. 6, pp. 42-50.

- Klaus pohl "challenges for the design of software –intensive system", university of Duisburg.essen, 45117 Essen, Germany.

- Lapham, M.A, **Acquisition Support Program,** (2006), Contributor: Carol Woody, PhD, Technical Note CMU/SEI-2006-TN-007. (2006).

-Luboh. A, (2010). "Trends in government services in the GCC countries". A Middle East point of View, 2010, PP: 32 to 37\.

- Lapham, M. A., (2006). "Sustaining Software-Intensive Systems ". Published by the U.S. Department of Defense, 2006, U.S.

- Lubos brim (2008),"fundamentals of air traffic control ". Paradise seminar, 25 February 2008.

- Muller, G., (2011), **the Role of Software in Systems**, Buskerud University College.

- OMG,(2003) "Unified Modeling Language: Superstructure", (final adopted spec, version 2.0, 2003-01-02). Object Management Group (2003).

- OMG, (2003), "UML for Systems Engineering RFP", OMG: Request for Proposal, ad/03-03-41.

- OMG, (2007), Systems Modeling Language (OMG Sys ML) v1.0. OMG available specification. Document number: formal/2007-09-01.

- Prasad Rahul, (2011). "Recent Advancement in Air Traffic Control system". University of petroleum and Energy studies, July 2011.

- Soni, D., Nord, R., Hofmeister, C. (1995), "Software Architecture in Industrial Applications," *Proc. 17th Int'l Conf. Software Eng. (ICSE 95)*, ACM Press, 196–207.

- Soares, M.S., Julia, S., Vrancken, J., (2009). " Rea l-time Scheduling of Batch Systems using Petri Nets and Linear Logic", Journal of Systems and Software 81(11) 1983-1996(2008,2009).

- Simons, A.J.H., (1999). "Use cases considered harmful", Proceedings of Technology of Object-Oriented Languages and Systems, pp.194-203(1999).

- Soares, M. S, Vrancken, J., (2009). " Including Sys ML in the 4+1 View Model of Architecture for Software-Intensive System".7th Annual Conference on Systems Engineering Research, 2009, Lough borough University, UK.

- Tiako (2009), mentioned in "Designing Software-Intensive Systems".(2009) ,Langston University, USA.

- Website /flyingway.com/air lecture/airspace%20 control.pdf, Appendix B
" AIR SPACE AND AIR TRAFFIC CONTROL" .

-
http://symposium.itea2.org/symposium2006/main/publications/TNO_IDATE_study_ITEA_SIS_in_the_future_Final_Report.pdf TNO/IDATE, “Software intensive systems in the future”, September (2005) V5.

- Wisnosky, D.E., Vogel, J. (2004). DODAF Wizdom: a Practical Guide to Planning, Managing and Executing Projects to Build Enterprise Architectures using the Department of Defense Architecture Framework, Wizdom Systems, Inc.2004.

- Wirsing, M., and Ronchaud, Rémi, (2004). “Engineering Software Intensive Systems”, European Commission US National Science Foundation, Edinburgh, UK, 22 - 23 May 2004.

الملخص

تعتبر معمارية البرامجيات مفهوماً عاماً بحيث أنه لا توجد توصيةْ محددة على إستخدام طريقةِ أو لغةِ خاصّةِ بها.

تلعب أنظمة البرامجيات المكثفة أحد الأدوارَ الرئيسية والضرورية في هندسة البرامجيات، حيث أنها تُؤثِرُ على التصميم ، والبناء , والإنشاء، والتوزيع بالإضافة إلى تطوير النظام بشكل كامل.

و الغرض من هذه الرسالة هو التحسين على نموذج محدد وهو نموذج (4+1) وذلك لتقليل التحديات في أنظمة البرامج المكثفة, المشكلة من أنظمة البرامج المكثفة أنها تفتقد لنمط (قالب) البرامجيات . وهذا لنمط (القالب) الجديد يجب أن يحسن على نموذج (4+1) من خلال تزويد النموذج لكيف يصمم وينفذ نوعين (المراقبة والسيطرة), (التكامل والتراكب) من انظمة البرامج المكثفة , لذلك فقد قمنا بإنشاء نمط (قالب) جديد بحيث يتم من خلاله حل التحديات الآنفة الذكر.

ومن النتائج المتوقعة لهذا الرسالة, أننا سنحصل على نمط (قالب) جديد بحيث يَضمن منهجية أوطريقة قابلة للتكيف بالشكل الذي يُمْكِنُ أنْ يُكاملَ ويُزامنَ كُلّ نشاطات نظام البرامج المكثفة .

نــمط نمذجـــي لمعمارية (1+4) للـــــنظـم كثيـفــة الـبـرمجيـات

بواسـطة

نـــور رشـــيد حمـيد

بأشـــــراف

د.مـؤيــــد فضـــــــل

قدمت هذه الرســـالة استكمالاً لمتطلبـات الحصـول على درجة الـماجسـتيـر في علم الحاسوب

عمــادة البحث العلمـــي والدراســـات العلـــيا

جامعة فيلادلفـيا

2012