An Approach to Improve Multi-agent System Architecture Design by Minimizing Complexity Aspects

By

HOWAYDA ABDALLAH ALI ELMARZAKI

Supervision

Dr. Twfig Eltwel

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

University of Benghazi

Faculty of Information Technology

Fall 2017-2018

1955

Student name: **Howayda Abdallah Ali Elmarzaki**

Faculty: **information technology**

Title of the thesis: **An Approach to Improve Multi Agent System Architecture Design by Minimizing Complexity Aspects.**

Defended and approved date**: 19/2/2018,**

Examination committee signature

Dr.Tawfig M. Eltaweel	(Supervisor), chairman	[illegible]
Title: **Advisor**		
Dr. Abdelsalam M. Maatuk	Member	[illegible]
Title: **Internal examiner**		
Dr. Mohamed Ahmed Khlaif	Member	[illegible]
Title: **External examiner**		

[illegible]

(Dean of faculty)

University of Benghazi Faculty of Information Technology

Dean of the Faculty

Prof. Mohamed Saleh buamud

(director of graduate studies and training)

اسم الطالب : هويده عبدالله علي المرزكي

اسم الكلية : كلية تقنية المعلومات

عنوان الرسالة : طريقة لتحسين معماريات التصميم للأنظمة المتعددة الوكلاء عن طريق تقليل التعقيد.

تاريخ الإجازة : 2018/2/19م.

أعضاء لجنة المناقشة : **التوقيع**

الدكتور توفيق محمد الطويل (المشرف)، رئيساً ....[illegible]........

الصفة والتخصص : المشرف – هندسة البرمجيات

الدكتور عبدالسلام مراجع معتوق (عضواً) ....[illegible]........

الصفة والتخصص : الممتحن الداخلي – علوم الحاسوب

الدكتور محمد أحمد اخليف (عضواً) ....[illegible]........

الصفة والتخصص : الممتحن الخارجي – هندسة برمجيات

يعتمد عميد الكلية **مدير إدارة الدراسات العُليا والتدريب بالجامعة**

........[illegible]........ ....

جامعة بنغازي ... كلية تقنية المعلومات
عميد الكلية

Dedication

I would like to dedicate this thesis to my beloved parents
and my husband Fathi, who offered me unconditional love and support throughout the course of this thesis.

Acknowledgments

First and foremost, I would like to thank ALLAH, without ALLAH this work would never have been finished.

I would like to express my sincere thanks to my supervisor Dr. Twfig Eltwel for his invaluable guidance and advice, his continuous support and all the useful discussions I had with him from my initial study on this research to the final thesis revisions.

I would like to thank my beloved husband Fathi El faitouri for his unlimited and faithful support as well as his patience and unconditional love. This thesis could not have been completed without his assistance. Thank you!

Also, I would like to take this opportunity to thank my family who provided me over the years with love and affection.

The last but not least, I am profoundly grateful to my kind friend Asya Sohaim for her fruitful collaboration and advice. Also, I am deeply indebted to Ms. Ebtisam Elberkawi and Heba Elnajar who gave me useful advices. Finally, I thank everyone who encouraged me.

Howayda
Jan 2018

Contents

List of Figures

List of Tables

List of Abbreviations

MAS Multi-Agent System

GUI Graphical User Interface

AD Architecture Design

RS Recommendation System

ACL Agent Communication Language

KQML Knowledge Query and Manipulation Language

PA Profiling Agent

TA Translation Agent

RA Retrieval Agent

FIPA Foundation for Intelligent Physical Agent

UCMs Use Case Maps

BBS Black Board System

UCP Use Case Point

FP Function Point

CFA Collaborative Filtering Approach

CBF Content-Based Filtering

KB Knowledge Base

KBS Knowledge Base System

OOE Object Oriented Environment

RB Rule Base

BR Books Resource

BDB Books Data Base

Req Request

Info-M Information (message)

IDR Item Data Base

FG Factors and guidelines

FGA Factors and Guideline of Abstraction

FGM Factors and Guideline of Modularity

FGMOD	Factors and Guideline of Modeling
CCM	Cohesion Communication Measurement
SDLC	Software Development Life Cycle
TCM	Task Complexity Measurement
UTW	Unadjusted Task Weight
UAW	Unadjusted Actor Weight
NST	Number of Simple Tasks
NAT	Number of Average Tasks
NCT	Number of Complex Tasks
NSA	Number of Simple Actor
NAA	Number of Average Actor
NCA	Number of Complex Actor

Abstract

The efficiency of multi-agent system (MAS) design mainly relies on the quality of a conceptual architecture. Thereby, the quality substantially influences the software system architecture, and plays a key role in the description of the initial architecture. Hence, quality properties, such as understandability, complexity, readability, testability, maintainability, reusability, etc should be considered at an early stage in the software development process. It is worth noticing that, large systems such as multi agents systems require many communications and interactions to fulfill their tasks, which may leads to complexity of architecture design (AD).

This thesis attempts to clarify the complexity situations that might happen during the description of architecture design through many aspects represented in the abstraction, modularity, and modeling by introducing an approach that aims to put a set of guidelines to minimize the effects of complexity, provides clarification for each guideline and guides agent systems developers in order to design architectures with high quality, low complexity and understandable.

The approach is applied on a case study to books recommendations system which based on multi agent systems where the complexity is measured by complexity task measurement (CTM) which based on use case point method. The solution has also displayed the complexity results before and after applying the approach.

Keywords: Multi agent system (MAS), a general architectures, Quality attributes, Recommendations systems (RS

CHAPTER 1
Introduction

In the last decade, expert systems and more recently, multi-agent systems emerged as new software technologies which brought together many discipline as (reasoning, knowledge representation, machine learning, planning, coordination, communication , etc.) in an effort to build distributed, intelligent, large scale systems and applications (Oprea, 2004). Multi-agents systems belongs to the field of AI (Artificial Intelligence), the field addressing the approaches of construction of complex systems using a large number of entities (agents) which are altering their behavior in order to accommodate with a particular problem (Markic, 2014). These agents, work together to solve problems which cannot be solved by their individual skills via Agent Communication languages (ACL) (Wood & DeLoach, 2001)**.**

An agent is computer software that contains many features. One of the most important features of an agent is "autonomy" which enable the agent to take decisions without the direct intervention of humans or others (Markic, 2014), (Ahmed Taki, 2014). An intelligent agent can also be reactive, proactive, and social ability, because it responses to the actions and alteration which appears in the working environment, can tack the initiative to establish the goals, and interacts with other agents. Other features that an agent might have include mobility, adaptability, trustworthiness, rationality, and learning capability (Chin, Gan, Alfred, Anthony, & Lukose, 2014).

In software industry, design decision is the most difficult task; particularly, when system is constructed from many components. These components require an organization of an overall system. This organization is called Architecture Design, (AD) which represents the fundamental frame of a system embodied in its elements and relationships. It actually represents the design decision of a software system due it found directly before design phase and after analyses phase in software development life cycle (SDLC). Unfortunately, the complexity becomes the major problem of architecture design (Far, 2002). which affects the software system's quality characteristics such as understandability, reusability, maintainability, testability, due to of the size of software systems, components and interactions increase (Sinha, 2013), (Ghazal Keshavarz, 2011).

Most literatures state that the complexity emerges clearly in architecture design of multi agent systems that assigned many and different tasks.

This research work is an attempt to assist developers to design high quality architectures, not complex, easy to understand and easy to validate for systems based on agents by introducing an approach including a set of factors and guidelines considered during architecture design development.

1.1.Motivations

From available literatures, many researchers have tried to set factors for quality in architecture design that discuss general issues of design quality, without targeting in particular the issue of "complexity" or developing guidelines to reduce it. Other researchers have measured the complexity in multi-agents systems's architectures design using different methods such as:

- Using mathematical logic (Anirban Sarkar, 2012).
- Using matrices (Kl¨ugl, 2008).
- Using certain conventional equations, roads and working in the Object Oriented environment (Iván García-Magariño, 2010).

There is a large number of researchers who have used proposed methodologies and approaches to design architectures based on multi-agents systems (Sara Maalal, 2011).

On one hand, all researchers have attempted to produce high quality architectures, but on the other hand there are no specific guidelines to assist developers avoid the difficulties and complexities in the early stages of architecture design.

Correspondingly, the challenge of complexity is not only large, but also growing (Luiz, 2009). For this reason, developers of these systems strive to design architectures which have highly cohesive, low coupling and low complexity to meet the quality requirements (STARON, 2016), (Saxena & Kumar, 2012).

Moreover, complexity can affect other quality characteristics, e.g. understandability, maintainability, reusability, and testability as it is illustrated in figure (1.1) (Ghazal Keshavarz, 2011).

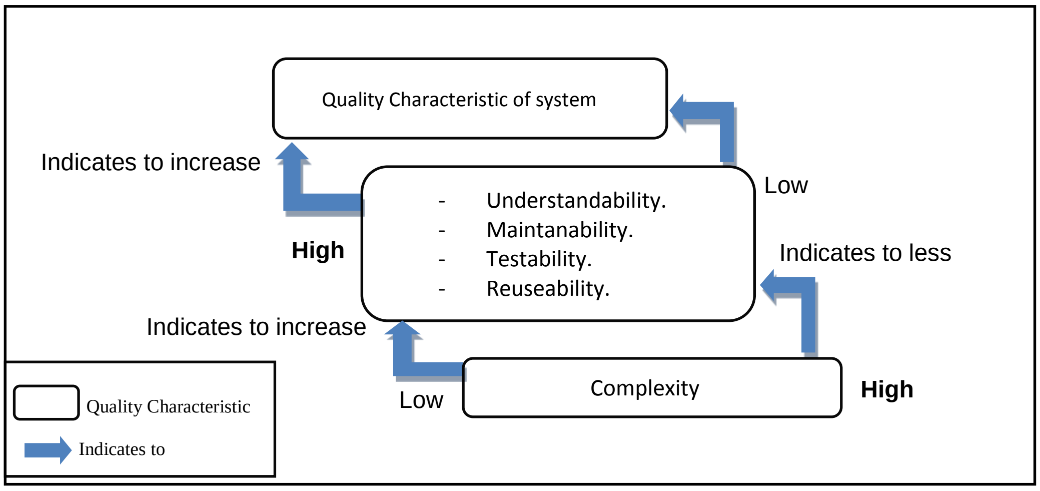


Figure 1.1: Quality characteristics which are affected by complexity

From available literatures, the complexity of the design hides potential defects, and makes it impossible to be certain if the systems will function correctly. This is by a survey introduced by Financial Times and The Economist in the period (2001 – 2009) concerning with complexity in software design (Luiz, 2009).

This means that software engineers need to produce designs that are easy to understand, easy to implement, and easy to reason about (Keating, 2000).

In the same context, quality evaluations of multi-agents systems architecture is a crucial issue for complex systems "design which is comprised of multiple agents" (Anirban Sarkar, 2012).

1.2.The Problem Statement

Complexity will be discussed as a research problem occurring in software architecture design based on multi-agents systems, which requires more interactions among agents to support and achieve its goals (services). These interactions are the main reason for complexities occurrence. Whereas, complexity in design is the most important factor that affects the quality of the design directly, especially in large systems (Ghazal Keshavarz, 2011).

1.3.Aims and Objectives

- **Aim:**

The aim of this research is to introduce an approach to decreas the complexity of architecture design in multi-agent systems.

- **Objectives:**

To achieve the aim, the following steps must to be taken.

- To review literature works.
- To introduce an approach including the affected factors of complexity and guidelines (consider during develomping the architecture design).
- To evaluate the validity and the effectiveness of this approach by a real case study.

1.4. The Solution

- The proposed solution is to achieve the desired goals of this research work. It mainly presents a set of guidelines including the influential factors on the complexity of architecture design. These factors are extracted from several sides of architecture design which should be taken into consideration at the early stage of developing the architecture.

- Introducing a clarification section including illustrative examples to each factor influencing the complexity of architectures design of systems.

- In this research work we suggested that "FG4Complexity" to label the solution. Thereby, "F" letter means Factors, "G" letter means Guidelines, and the "number 4" means for.

1.5. Contribution of the Thesis

The main contribution of the present study is as follows:

- Introducing approach can help the developers of multi-agent system to build their architectures avoiding the complexity influences from initially analysis to architectural design decision.
- Adding clarification part after each complexity factor or guideline in the approach to clarify its affection of complexity on architecture design.
- Introducing new measurement method and applying it on system based on multi-gent. This by adding some modifications to the use case point method and adapt it to the agent environment. The modification aims at estimating the complexity of the tasks in each agent, the complexity of every actor connected with agents, the technical complexity factors, the complexity of environment and the complexity of the tasks assigned to all agents.
- Improving quality of multi-agents systems architectures design by decreasing the complexity sides.
- The research work published in the following:
  - 7th International Conference on Software Engineering and Applications (JSE-2018), the title of research paper is “Increasing the architecture design quality for MAS: an approach to minimize the effects of complexity”. In 2018.
  - International Journal of Software Engineering & Applications (IJSEA), the title of research paper is “Minimizing the Complexity effects to MAS Architectures Design based on FG4Complexity Approach".in 2018.
  - BAMMS Conference, and SPRENGER, the research paper title is “ New Approach to Measure the Architecture Design Complexity of Multi Agent Systems: Recommendations System Case Study”, 2018. In proceeding.

1.6.The Methodology

In this research, an empirical approach is mainly concerned with the selected research methodology. In this thesis, there are three essential aspects of architecture design represented in abstraction, modularity, and modeling. In chapter four, the researcher provides further elaboration on the importance of these concepts to the architecture design complexity.

1.7.Scope and Limitation

The research will focus on architecture design which is between analysis phase and design phase in software engineering lifecycle. Other phases such as (Requirements, Implementation, Maintains, Testing, design,…etc.) will not have more attention. This research actually utilizes the experimental method (case study) to apply the proposed approach of multi-agents systems and the architecture design complexity will be measured by using certain methods of software engineering; however, the fuzzy logic or the mathematical theories will not be used in research.

1.8. Structure of Thesis

This thesis consists of six chapters which are organized as follows:

1. presents the motivation, the research problem, the objectives, outlines of the methodology. This chapter also covers the key contributions, the solution approach, finally scope and limitation of this thesis.

2. Summarizes the background knowledge of multi agent systems and its applications, software architecture design, quality and complexity of software design. It also includes the abstraction, the modularity, the modeling and the Black Board System. This thesis sheds light on Gold Plating, Function Point, Use Case Point, recommendation systems, use case maps and the knowledge based system and the measurement.

3. Provides an overview of the related work such standard and guidelines which support the multi-agents systems quality, analyses the complexity of MAS, and research works on complexity measurement.

4. Describes the proposed approach to solve the problem of architecture design complexity based on specific concepts. It also illustrates the architecture design in software development life cycle, thus; recognizes the concepts and properties which have a crucial impact. Then, it explains the motivate behind selecting those concepts by introducing their roles to reduce the level of complexity and improve quality. Eventually, it ends up with a summary of the whole chapter.

5. Covers the case study based on multi agent system on which the proposed approach, the application steps and the measurement will be applied. The application is through some models used in methodologies related to agents systems.

6. presents conclusion and the scope of future works.

This chapter clarifies motivations, the problem statement, the objectives, the proposed solution approach followed by the contributions, the methodology and the ends scope and limitation of this research.

CHAPTER 2
Background

The purpose of this chapter is to briefly present the background on the research fields impacting the work of this thesis. First, it explains briefly the multi agent systems, and presents the definition, architecture and communication of agent. Then, it outlines the applications that are based on the multi agent systems. After this the software architecture design is described and the quality of software design is defined. Then it provides different definitions of the complexity of software engineering and its types. In addition, it discusses the aims, the levels and the types of abstraction. This chapter also gives an overview about modularity, modeling, black board system, gold plating and function point. It clarifies the use case point and the steps that need to be followed to count processes. Finally, it summarizes the recommendation system, the benefits of use case maps, the knowledge base system, measurement, and task.

2.1. The Multi Agent Systems

Multi-agents systems can be defined as software systems which are comprised of groups of entities called (agents). These agents, work together to solve problems which can be not solved by their individual skills via Agent Communication languages.
(Bhardwaj, 2015), (Muli, 2015). The agents are usually designed to be:

- **Autonomous:** Making decisions without the direct intervention of humans or others.
- **Reactive:** Agents react to events and changes that arise in working environment such as Physical world, Graphic User Interface (GUI), Agents, Internet or combined**.**
- **Proactive:** Agents can exhibit goal directed behavior by taking the initiative.
- **Social ability:** Agents interact with other agents via some kind of Agent Communication language (Markic, 2014).

Business decisions are based on extraction of useful knowledge from various data sources. Those data sources, are usually called the big data, might be huge digital data sets like an internal data warehouse or external sources like the web. Intelligent agents can implement certain tasks such as big data processing, information retrieval, etc. (Markic, 2014). Chapter 5 explains system based on multi-agents systems to use it in apply the proposed approach.

2.2. The Definition of Agent

An agent is a computer system within an environment and with an autonomous behavior made for realizing the objectives that were set during its design (Malika Addou, 2011).

2.3. The Agent Architecture

The agent has several architectures design such as BDI (Believe, Desire, and Intention) architecture, cognitive architecture, reactive architectures, layered and hybrid architectures, etc. These architectures are only related to the internal agent, but they are not a part of the multi agent system or organization (O. Shehory, 1998), (Chin et al., 2014), (Broersen, Dastani, & van der Torre, 2005). This work will focus on the tasks that assigned to each agent in system.

2.4. The Communication of Agents

It is the regular way that agents may interact with each other in order to achieve their delegated goals. The agents can effectively communicate and exchange knowledge with each other by using shared language called agent communication language .Typically, agent communication languages are based on the speech act theory, which is a human knowledge level communication protocol, Knowledge Query and Manipulation Language (KQML) and Foundation for Intelligent Physical Agent (FIPA) which are the best known language used by software agents for their communicative exchanges (Markic, 2014). In this research work, the interaction among agents considered as the fundamental factor to stem the complexity in architecture design.

Figure 2.1 illustrates an overview of interactions among agents via protocols on sharing environment (Zambonelli et al., 2001).

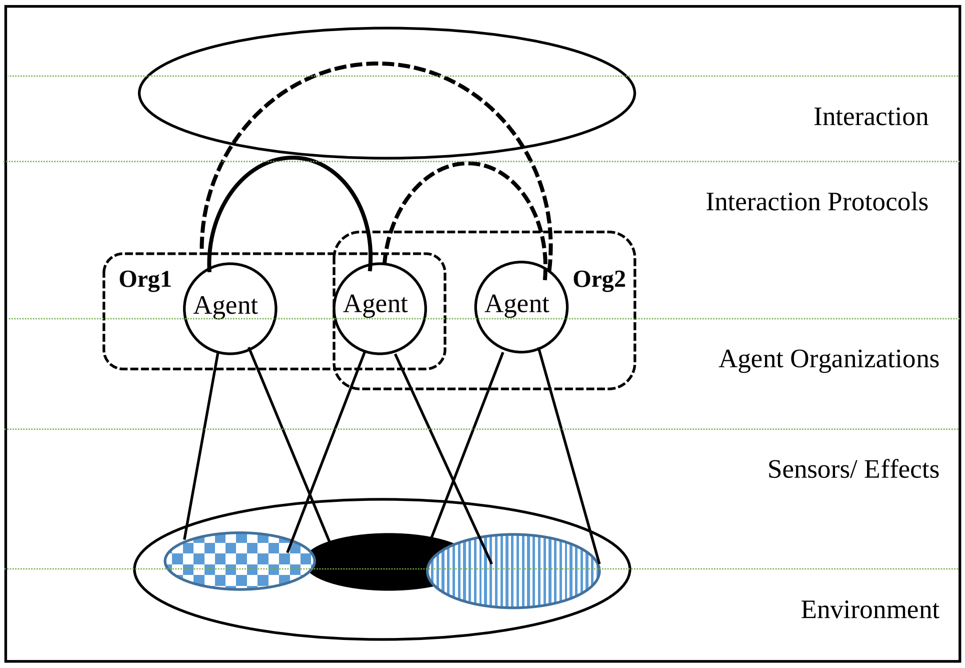


Figure 2.1: The interactions among agents' environment via ACL

2.5. The Applications Based on Multi-agents Systems

There are different application areas of multi-agents systems such as Ecommerce, Economic systems, Distributed information systems, Research engines, Social media, Recommendation systems, Scheduling, planning and other systems as shown in Figure2.2 (Sara Maalal, 2011). Chapter 5 illustrates the complexity which occurs in architecture design by using the recommendations system.

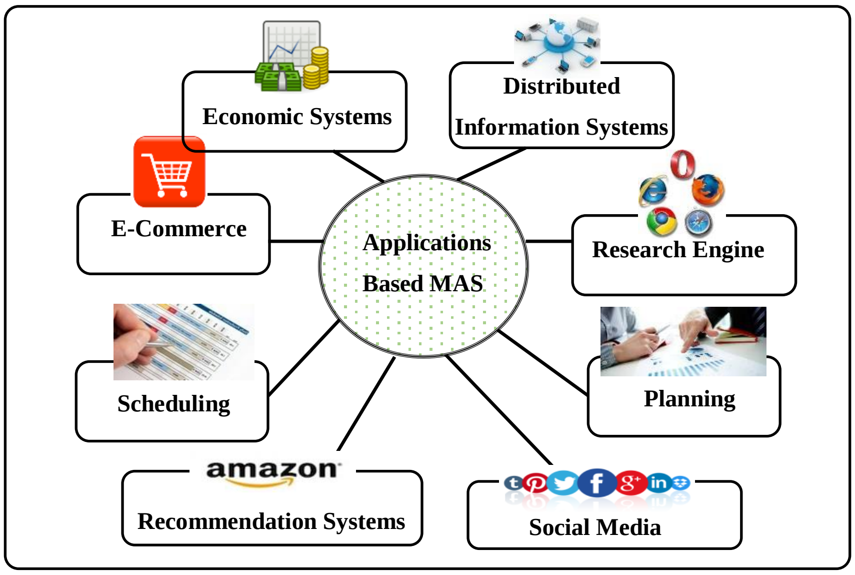


Figure2.2: The multi-agents systems applications in real world

2.6. The Software Architecture Design

The software architecture has become an essential element in designing and discipline of large and complex systems. In fact, software architecture is a description of the system as the blueprint that aids in the understanding of how the system will act and it captures early design decisions. The software architecture introduces many benefits like system understanding, documentation, architectural drifts and reusability (O. Shehory, 1998), (Weyns, 2010).

There are different definitions of the concept of software architecture.

- It is composed of elements, form, components, connectors, and configurations (Serebrenik, 2014).
- It describes solutions for addressing specific quality concerns as scalability, modifiability, availability, security, performance, etc. (Mirakhorli, 2015).
- It is considered as the fundamental organization of a system embodied in its elements, relationships (Serebrenik, 2014).

Consequently, software engineers often describe the architectures of their systems as high level sides of the systems like the overall organization, modularity into components, and the tasks that assignment to components, and the way the components interact. These descriptions often use box and line diagrams and phrases (Mary Shaw, 1995), (Kruchten, 1995).

In multi-agents systems context, the architecture is 95% software engineering and just 5% multi agent systems theories (Sara Maalal, 2011).

2.6.1. The Architectural Issues Including Strategic Decisions:

The decisions of design represented in architecture design including many issues such as the following:

- Structural issues that include all organization and control structure.
- Choosing among design alternatives.
- Assignment of tasks to constituent agents.
- Structure of constituent agents.
- Determining protocols for communication, synchronization, etc.
- Hardware distribution (Far, 2002).

This work, will discuss the complexity, quality, application, and measurment issues through the architecture design of multi-agent system.

2.7. The Quality of Software Design

The quality of a software design can be expressed in terms of several characteristics such as reusability, flexibility, understandability, functionality, extendibility, and effectiveness(Sharma, 2012). Theoretically, the first place in which quality requirements can be addressed is architectural models of software (Evesti, 2007), (ISO, 2016).
Quality requirements are considered as non-functional requirements in the initial steps of software development and influence, importantly the architecture of software. In addition, building a high quality software for real world applications is a hard mission for some problems such as, the large number and flexibility of components, the complexity of interconnections required (O. a. S. Shehory, Arnon, 2001).

Design quality indicates to decrease the rework, costs, and schedules, which lead to decrease prices and increase market share; as a result, it leads to increase profits and business continuity (Chris F. Kemerer, 2009). If so, it is useful realizing the quality in multi agent system which including large number and flexibility of components, and interactions.

2.8. The Complexity in Software Design

The term "Complexity" refers to the effort that is required to understand with the system (Wagner, 2011). There are various definitions of complexity in software design and which can be listed as follows:

- "The term complexity refers to a large number of interacting components in a software design" (Mohamed, 2013).
- "The degree of connectivity between entities in a software design" (STARON, 2016), (Sinha, 2013).
- "Software design complexity is used to indicate the testability, maintainability, readability and understandability of a software" (Sinha, 2013).
- "Complexity in software design refers to the difficulty in understanding and manipulating the set of concepts" (Karageorgos, 2003), (Tran-Cao, Abran, & Lévesque, 2001).

2.8.1. The Types of Complexity

The categorization of complexity is termed as Detail Complexity and Dynamic Complexity.

- **Detail Complexity**: is a type of complex situation that has a great number of possible interconnections between parts.
- **Dynamic Complexity**: is a type of complex situation where cause and effect are subtle and where the effects over time of interventions are not clear (Hanseth, 2010), (Wagner, 2011), (Bouwers, 2010). Chapter 4 discusses these types of complexity.

2.9. The Abstraction

The activity of simplification is composed of reduction of details and the generalization of crucial and common attributes.

2.9.1. Aims of Abstraction

The main aim of abstraction is reducing complexity, and there are two fundamental issues for abstraction. Firstly, abstraction is essential to be able to understand the necessary components for software design. Moreover, interaction between components is too complex to be understood as a whole. Hence, it divides the software into smaller chunks and deletes explicit information in order to understand certain sides. Secondly, reuse is unavoidably connected to abstraction (Tsui et al., 2011), (Tsui et al., 2011).

2.9.2. Levels of Abstraction

There are two levels of abstraction. The first level is called abstraction specification and the other is abstraction realization.

For example, if we have variables and fixed numbers at the same time, they are required to be represented by levels of abstraction. In this case, levels will be divided into abstraction specification and abstraction realization. The first comprises of variables and a fixed parts. The fixed part is what is set by the abstraction. For example, the information that has been abstracted but is still visible as shown in Figure 2.3(Tsui et al., 2011).

Variable
Fixed Number

Figure 2.3: The specification level of abstraction (Tsui et al., 2011)

The second includes further details as illustrated in Figure 2.4.

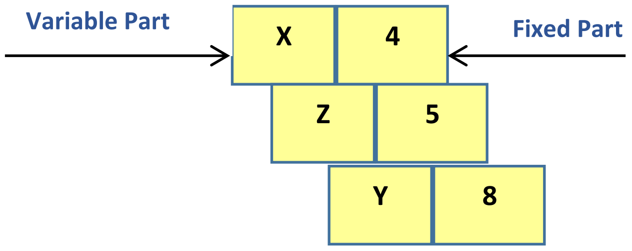


Figure 2.4: The realization level of abstraction (Tsui, Gharaat, Duggins, & Jung, 2011)

There is still abstraction presented in the variable part. Figure 2.5 shows the two levels of abstraction clearly.

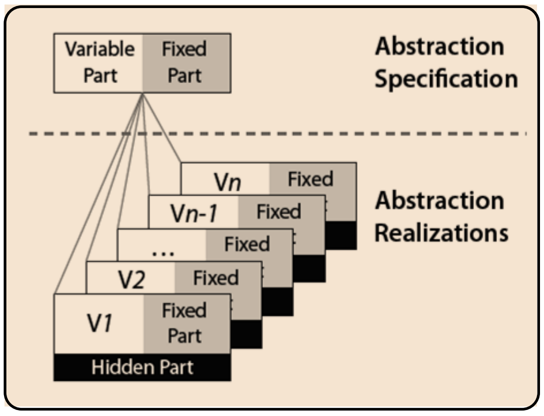


Figure 2.5: The transiting between specification level to realization level of abstraction (Tsui et

2.9.3. Types of Abstraction

The abstraction consists of two different types; the first type is called simplifying abstraction, and the second one is generalizing abstraction. The simplifying abstraction is the type of abstraction that is used when we want to reduce dynamic complexity, for example, removing windows titles if developers do not have to care about it anymore. In Figure 2.6 segment 2 shows how this abstraction is used and how it contributes to reducing complexity. The generalizing abstraction is used if we have several components that have many similarities and only differ in some aspects. In Figure 2.6 segment 3, the differing information between component C1 and C2 is only t1 and t2. Thus, we generalize C1 and C2 to C that has a parameter P. The information of parameter p is removed, and Cg is abstracted to C. This procedure makes the usage of C simpler and less complex in design. The major design goal for generalizing abstraction is reusing (Leopold, Mendling, Reijers, & La Rosa, 2014), (Wagner, 2011).

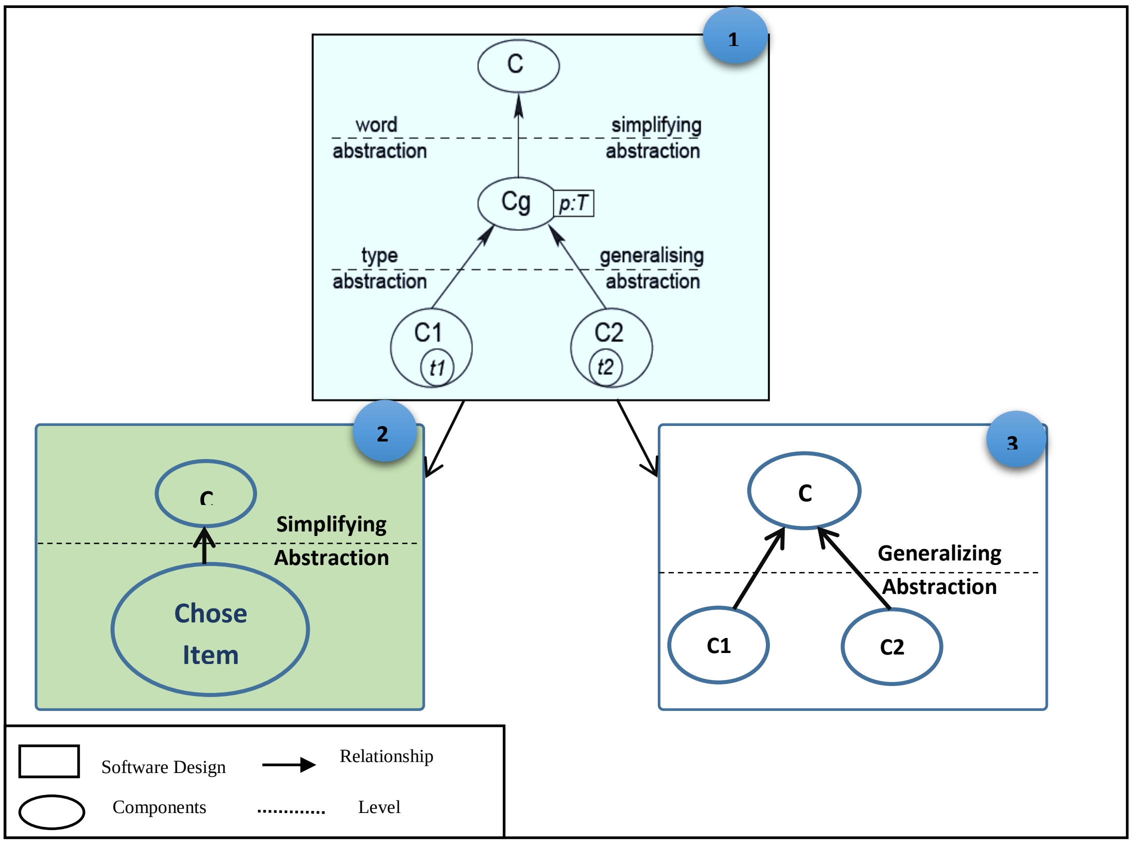


Figure 2.6: Illustrating the different types of abstraction(Wagner, 2011)

2.10. The Modularity

The modularity refers to a crucial concept that developers exercise to reduce the complexity of software systems. The IEEE Standard Glossary Terminology defines the modularity as "The degree to which a software is composed of discrete components where, the changing to one component has minimal impact on other components". This definition is closely related to Booch's (1994).

A modularization generally has three purposes:

- To make complexity manageable.
- To enable parallel work.
- To accommodate future uncertainty (Alessandro Garcia, 2008)**.**

2.11. The Modeling

Modeling a system means identifying its main characteristics, states and behaviour using notations. Whereas, models are the most important engineering tool which allows us to understand and analyze large and complex complications. In architecture design, substantial architectural concepts that need to be modeled are components, connectors, interfaces, and configurations. The goals of modeling include communication, bug finding, quality, analysis, etc. Thus, architectural modeling is the reification and documentation of design decisions (Taylor, Medvidovic, & Dashofy, 2009).

2.12. A Blackboard System (BBS)

A blackboard system is an artificial intelligence methodology based on diverse group of specialist knowledge sources to iterative updates. It is a global accessible database which is used for intermediate, partial results of problem solving. The blackboard system starting with a problem specification and ending with a solution. Each knowledge source updates the blackboard with a partial solution when its internal constraints match the blackboard state. In this method, the specialists work together until the problem is solved. The blackboard system consists of three components: Blackboard (BB), Control Unit, and Knowledge Sources (KS). The blackboard model was originally designed as a technique to handle complex, difficult problems, where the solution is the sum of its parts. The blackboard system will be used in chapter 5 to support one of the agents in system as shown in Figure 2.7 (Rudenko & Borisov, 2007), (Straub, 2014), (Pang, 2000).

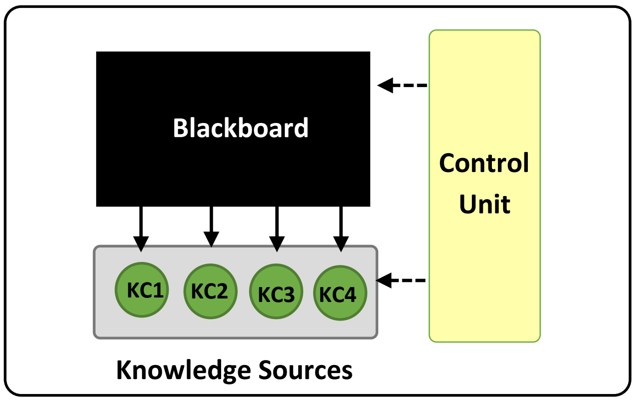


Figure 2.7: The components of blackboard system

2.13. Gold Plating

Gold plating means when extra feature in software is added to delight the customer (a kind of surprise). Gold plating is not a bargain. It can increase operation, complexity, maintenance, costs and decrease quality. In software engineering a gold plating requires hard work to be accomplished. Also, it needs extra effort and time, and it is possible that complexity appears during work (Kirandeep Kaur, 2013). The gold plating concept will be one of the addressed problems in this study.

2.14. Function Point (FP)

Currently, the function point is the most used method to determine the size of a user function and its complexity. FP is used by several organizations all over the world. For many years, FP has become a world standard, and measures the functionality that software should provide, as well as the technical and environmental complexity. Moreover, it measures the size of the user functions of application software or part of it. User functions are the components requested and recognized by the user. The function point takes the user function complexity into consideration (Meli & Santillo, 1999). The point 2.15 will be explained method to estimate the function complexity which is built on function point method.

2.15. Overview of Use Case Point (UCP)

Use Case Points (UCP) is a software estimation technique used to measure the software size with use cases. The concept of UCP is the development of FP that was developed by Gustav Karner to cope with object oriented environment. The work was later licensed by Rational Software that merged in IBM. It estimates the number, the size and the complexity of use case quantitatively by an actor and use cases in use case diagram for estimation of software size (So Young Moon, 2013b), (Ghazal Keshavarz, 2011). It is of general agreement that quality issues should be considered very early in the software development process, to avoid risks and to facilitate the achievement of the overall software system (Francisca Losavio and Ledis Chirinos, 2003b), [21], (Far, 2002).

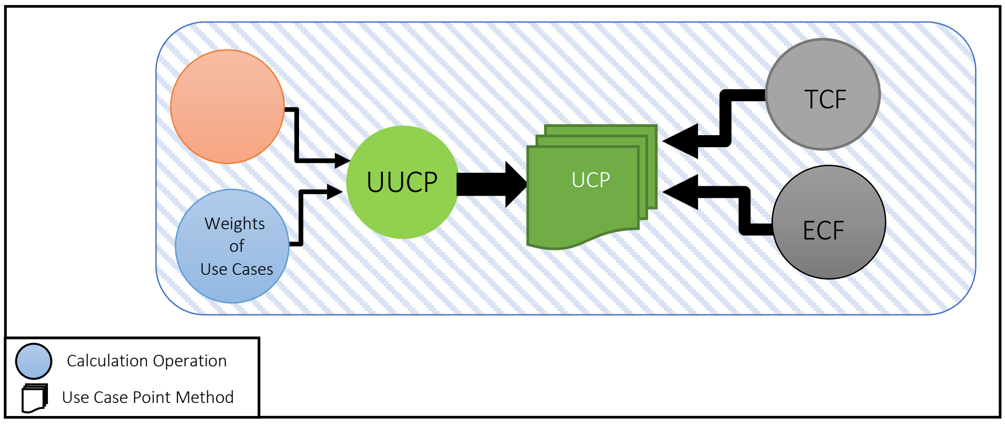


Figure 2.8: The describing of the UCM method

2.15.1. Use Case Point Counting Processes

The Use Case Points counting process has the following steps

- Step1. Calculate unadjusted UCPs
- Step2. Calculate technical complexity
- Step3. Calculate environmental complexity
- Step4. Calculate adjusted UCPs

Step1 Consists of three steps are:

- Step1.1. Determine Unadjusted Use-Case Weight.
- Step1.2. Determine Unadjusted Actor Weight.
- Step1.3. Calculate Unadjusted Use-Case Points.

In step 1.1, the number of transactions is counted in each use case to determine the Unadjusted Use-Case classification and weight. The classification of use case is simple, average or complex. With regard to the weight, it varies according to the classifications. Table 2.1 shows the classifications and weights of each use case.

Use Case Classification	No. of Transactions	Weight
Simple	1 to 3 transactions	5
Average	4 to 7 transactions	10
Complex	8 or more transactions	15

Tab le2.1: The Use Case Classification (So Young Moon, 2013a)

All use cases which are between (1-3) and classified as simple are determined and multiplied by the weight 5; all use cases which are between (4-7) and classified as average and multiplied by the weight 10; and all use cases which are classified as complex and multiplied by the weight 15. After that, the total of the use cases after multiplying by their weights to get the Unadjusted Use-Case Weight (UUCW) as illustrated in Table (2.2).

Use-Case Complexity	Use-Case Weight	Number of Use-Cases	System
Simple	5	NSUC	5 × NSUC

Average	10	NAUC	10 × NAUC
Complex	15	NCUC	15 × NCUC
Unadjusted Use-Case Weight (UUCW)			5 × NSUC + 10 × NAUC + 15 × NCUC

Table 2.2: Calculate the Unadjusted Use Case Weight (So Young Moon, 2013a)

Where:

- NSUC is the no. of Simple Use-Cases.
- NAUC is the no. of Average Use-Cases.
- NCUC is the no. of Complex Use-Cases.

Step1.2. An actor in a use case might be a person, another program,…etc. Classify actors as simple, average, complex. The actor weight depends on the type of actor as Table 2.3 shows.

Actor Classification	Type of Actor	Weight
Simple	External system that must interact with the system using a well-defined API(Application Programming Interface)	1
Average	External system that must interact with the system using standard communication protocols (e.g. TCP/IP, FTP, HTTP, database)	2
Complex	Human actor using a GUI application interface	3

Table 2.3: Showing the actor classification (So Young Moon, 2013a)

Each actor, which is classified as simple, is determined and multiplied by the weight 1; each actor, which is classified as average, is determined and multiplied by the weight 2; and each actor, which is classified as complex, is determined and multiplied by the weight 3. After that, the total is multiplied by the weights to get Unadjusted Actor Weight (UAW) as illustrated in Table (2.4)

Actor Complexity	Actor Weight	Number of Actors	System
Simple	1	NSA	1 × NSA
Average	2	NAA	2 × NAA

Complex	3	NCA	3 × NCA
Unadjusted Actor Weight (UAW)			1 × NSA + 2 × NAA + 3 × NCA

Table 2.4 Calculate the Actor Weight (So Young Moon, 2013a)

Where:

- NSA is the number of simple actors.
- NAA is the number of average actors.
- NCA is the number of complex actors.

Step1.3. The unadjusted use case weight (UUCW) and the unadjusted actor weight (UAW) together give the unadjusted size of the system, referred to as Unadjusted Use Case Points.

(UUCP) = UUCW + UAW

Step2. Calculate the technical complexity:

To calculate the technical complexity we should pay attention to the 13 factors that contribute to the influence of the technical complexity of a software on use case points and their corresponding weights as given in Table 2.5.

Factor	**Description**	**Weight**
F1	Distributed system	2.0
F2	Response time/performance objectives	1.0
F3	End-user efficiency	1.0
F4	Internal processing complexity	1.0
F5	Code reusability	1.0
F6	Easy to install	0.5
F7	Easy to use	0.5
F8	Portability to other platforms	2.0
F9	System maintenance	1.0

F10	Concurrent/parallel processing	1.0
F11	Security features	1.0
F12	Access for third parties	1.0
F13	End user training	1.0

Table 2.5: the Technical Complexity Factors weights (So Young Moon, 2013a)

- For each of the 13 factors, measure the software and rate from 0 (irrelevant) to 5 (very important).
- Calculate the impact of the factor from impact weight of the factor and the rated Value for the project as:

$$\textbf{Impact of the Factor} = \textbf{Impact Weight} \times \textbf{Rated Value}$$

- Calculate the sum of impact of all the factors. This gives the Total Technical Factor (TFactor).
- Calculate the **TCF** as following:

$$\textbf{TCF} = \textbf{0.6} + (\textbf{0.01} \times \textbf{TFactor})$$

Step3. Adjust For Environmental Complexity

Consider the 8 environmental factors that could affect the software execution and their corresponding weights as given in Table 2.6.

Factor	**Description**	**Weight**
E1	Familiarity with development process used	1.5
E2	Application experience	0.5
E3	Object-oriented experience of team	1.0
E4	Lead analyst capability	0.5
E5	Motivation of the team	1.0
E6	Stability of requirements	2.0
E7	Part-time staff	-1.0

E8	Difficult programming languages	-1.0

Table2.6: Showing the environmental factors weight (So Young Moon, 2013a)

- Calculate the Environmental Factor (EF) where, **1.4 + (-0.03 × EFactor)**

Step 4: Calculate Adjusted Use-Case Points (UCP)

- Calculate Adjusted Use-Case Points (UCP) where **UCP = UUCP × TCF × EF** (So Young Moon, 2013b).

In chapter 5 use case point method is used to measure the complexity of tasks assigned to agents in system after adding some modification to adapt the agent environment. Figure 2.8 describes the mechanism of use case point.

2.16. Recommendation Systems (RS)

The recommendation is a very common phenomenon in our daily life. Nowadays, recommender systems appear as a developing application and a research field in several domains of computing research from artificial intelligence to information systems. Recommender systems, also known as personalization systems, are a popular technique for reducing information overload and finding items that are of interest to the user (Chaptini, 2005).

They mainly rely on many approaches such as Collaborative Filtering Approach(CFA), Content-Based Filtering (CBF) and other more complex approaches (Lenhart & Herzog, 2016), (Yan, 2014). These systems are effective means of selling more products because they work to filtering vital information fragment out of large amount of dynamically generated information according to user’s preferences, interest, or observed behavior about item. In e-commerce setting, recommender systems enhance revenues. In scientific libraries, recommender systems support users by allowing them to move beyond catalog searches. Therefore, the need to use efficient and accurate recommendation techniques within a system that will provide relevant and dependable recommendations for users (Isinkaye, Folajimi, & Ojokoh, 2015).

2.17. Use Case Maps (UCM)

Use case maps fill the gap between verbal descriptions and detailed descriptions in terms of interaction diagrams. These maps are useful between analysis stage and design stage. UCM notations permit the description of complex software driven systems in terms of high level scenarios and allow us to know the responsibilities of the components without going into the details about the messaging between them. UCMs provide an integrated view of behavior and structure at the system level where puts scenario paths on a structure of abstract components (Khan & Mahmood, 2012). Chapter 5 in application part, the UCM notations will be used with focus on some notations such as: Task, component, interaction, and (begins and ends of scenarios).

2.18. Knowledge Base Systems (KBS)

These systems are at the applied edge of research in Artificial Intelligence (AI) to solve the type of problems that normally require human experts. These systems such as medical diagnosis, financial analysis, factory production scheduling, and multi agents systems…etc. The knowledge base consists of the following: a domain knowledge, usually provided by human experts, very specialized for a particular problem domain, it is often known as IF-THEN rules, and it incorporates heuristics or probabilities (Akerkar & Sajja, 2010). Figure 2.9 shows the knowledge phase.

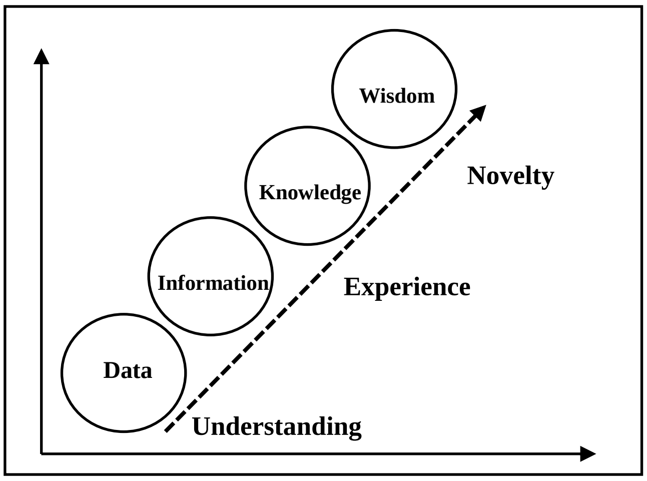


Figure 2.9: Showing the knowledge phase

2.19. The Measurement

A set of processes having the object of determining a value of a measure. It can include assigning a qualitative class (ISO, 2016). This concept is applied in chapter 5 to measure the agent's tasks.

2.20. Task

A set or sequence of activities is required to accomplish a given goal. This work focuses primarily on the tasks of each agent.

CHAPTER 3
Related Work

This chapter describes an overview of existing available literature of complexity and quality issues.

- **Complexity**

Complexity is one of the most interesting criteria for researchers, some of them as zambonelli in 2001, discussed a number of issues related to the analysis, design, abstraction and complexity of multi-agent systems by introduce some general guidelines for multi-agent system analysis and design that are centered around organizational abstractions. The research showed some complexity situations that occur during the agent's interactions without addressing the means to avoid such situations. Actually, the architecture design in these systems was not the main concern; although, it exists between the analysis and design phases, and the guidelines were only centered around abstractions concept without approaching the complexity (Zambonelli et al., 2001). The quality and complexity issues of multi-agent system were addressed by Behrouz Homyoue Far in 2002, which proposed some metrics to measure the complexity and introduced some metrics used for knowing a candidate set of agents for multi-agent system design. The research is dedicated to the complexity measurement, and does not provide solutions for avoiding or decreasing the intensity of the problem of complexity and ignores the complexity of architectures (Far, 2002). Jose Luiz, explained the nature of complexity as it arises in software design and discussed some of the challenges that still remain in design complexity. The research illuminated some concepts which influence complexity such as abstraction and documentation. It also discussed the role of architectures in large systems, but did not address the multi agent systems or the complexity of architecture design (Luiz, 2009). Eric Bouwers, attempted to provide a Software Architecture Complexity Model (SACM) which can be used to reason the complexity of a software architecture. It is based on theories from reasoning science and system attributes. The SACM can be used as a formal model to explain existing quality models and as a starting point within architecture evaluation methods. The complexity in multi-agents systems architectures, the affected factors on complexity and how to decrease it from these systems were not addressed in this research (Bouwers, 2010). Iván García in 2010, attempted to introduce a suite metric to measure certain

quality characteristics of multi-agents systems's architectures considering agents and their organization. Most of these metrics were based on object-oriented environment (OOE), the research adapted agent oriented environment. In other words, the research work, is only concerned with measurement of complexity, and does not address any means to avoid or reduce it (Iván García-Magariño, 2010). Ghazal Keshavarz in 2011, analyzed the software complexity issue, especially in the first phase of software development, and proposed a requirement based on a metric. Actually, this metric enables software engineers to assess the complexity before starting the actual design and implementation. The research does not focus on complexities which occur during interactions in large systems such like multi agent systems (Ghazal Keshavarz, 2011). Anirban Sarkar and Narayan Debnath in 2012, suggested some standards to complexity measurements of system based on multi-agents systems. They also, presented a case study, including conceptual architecture of MAS, and described a set of quality metrics based on multi-agents systems architecture design. These metrics addressed many architecture sides such as dynamic, structure, and agent side. The research work concentrates basically on the measurement of the complexity of architecture design to multi agent system, but did not decrease it in systems architectures (Anirban Sarkar, 2012).
Sinha in 2013, described the measurement method for software complexity including the factors affecting the complexity. But they did not address the large system such as systems based multi-agents systems and did not propose any methods to decrease the complexity in architecture design (Sinha, 2013). Alenezi and Almustafa in 2015, studied the complexity evolution of five open source projects from various domains; then, they conducted an analytical procedure for the growth of ten releases of these systems in design phase. Then, they displayed how complexity evolves over time. The research work focused on complexity analyses and proved its growth with the work progress of the large systems, but paid no attention to some specific sides of the complexity. In addition, the research did not provide any guidelines or approaches on how to avoid those complexities (Alenezi & Almustafa, 2015).

▪ Quality

Umapathy Eaganathana in 2016, introduced a survey of literature reviews which proposed various object oriented design metrics by different researchers to guarantee production of high quality software design that is free from defects and programming errors. The main goal of this systematic literature reviews was to investigate the role of metrics in software development lifecycle. It actually aims to help developers to produce qualitative software design and also to improve its productivity. This research depends on object oriented environment in proposing these metrics. It also focused on the overall quality of production without stating the issue of reducing the complexity (Umapathy Eaganathana, 2016). Punam Bedi, Vibha Gaur in 2007, proposed a methodology to obtain prioritization of quality specifications that assists quality engineer in achieving the desired level of quality for multi-agent systems (Jomi Fred Hubner, 2007). The research also, addressed general quality factors of systems based on multi-agents without solve the issue of complexity. Francisca Losavio and Ledis Chirinos in 2003, introduced an approach facilitates the choice of the right decisions during the architecture analysis process. It could be easily integrated into a general software development process or into specific architectural design methods. The research ignored big systems which have complex interactions and the design decisions does not considered the complexity factore (Francisca Losavio and Ledis Chirinos, 2003a).

Mike Keating in 2000, measured design quality based on measured design complexity by proposing a method quantifying design complexity which enabls design developers to produce architectures cabable of manging complexity, and enhance the quality (Keating, 2000). The research primerly focused on complexity, architecture and quality of design, but did not provide any guidelines or approaches on how to avoid those complexities. Furthermore, large systems such as multi-agents systems were not addressed in this research.

▪ Discussion

The complexity measurement was the major subject of interest among researchers which explained it from different point of views, e.g. measuring a dynamic complexity, structure complexity, function complexity, coupling degree and cohesion degree views. The researchers also, attempted to introduce suitable metrics to measure general quality of architecture. It is worth noticing that all previous works did not address the complexity in architecture design resulted from agents interactions which appear explicitly in multi-agents systems and found out solutions to decrease these complexities. That’s why this research work covers the complexity gap in the architecture design of multi-agents system by presenting many factors affecting the complexity and guide the developers to decease it in an early stage of architecture design development.

CHAPTER 4
The FG4Complexity Approach

This chapter describes the proposed approach to solve the problem of architecture design complexity called FG4Complexity (Factors and Guidelines for Complexity) which produced to facilitate understanding and to avoid the complexity aspects of architectures design based on specific concepts. It illustrates the architecture design in software development life cycle (SDLC). Thus, recognizes the concepts and properties which have a real impact on complexity. Then, it provides the motivate behind selecting those concepts by introducing their rule in decreasing the complexity and how to boost the quality. This chapter also introduces the factors and guidelines (FG) of each concept and explains their arrangement with some clarifications of FG4Complexity approach. Finally, introduces the summary of the chapter.

4.1. The Architecture Design in Software Developments Life Cycle (SDLC)

It is worth noticing that the term "an architecture" is used as a general description of how the subsystems join together to form the system (Pohl, 2010).

The architecture design represents the final approved decision which is made by system developers, and it is also an output of requirements analysis (van der Ven, Jansen, Nijhuis, & Bosch, 2008), (Ahmed Taki, 2014) which means that the architecture design always comes between requirements analysis phase (which also known as "what phase") and design phase (which also known as "how phase") in SDLC (Tekinerdogan & Demirli, 2013) as shows in Figure 4.1.

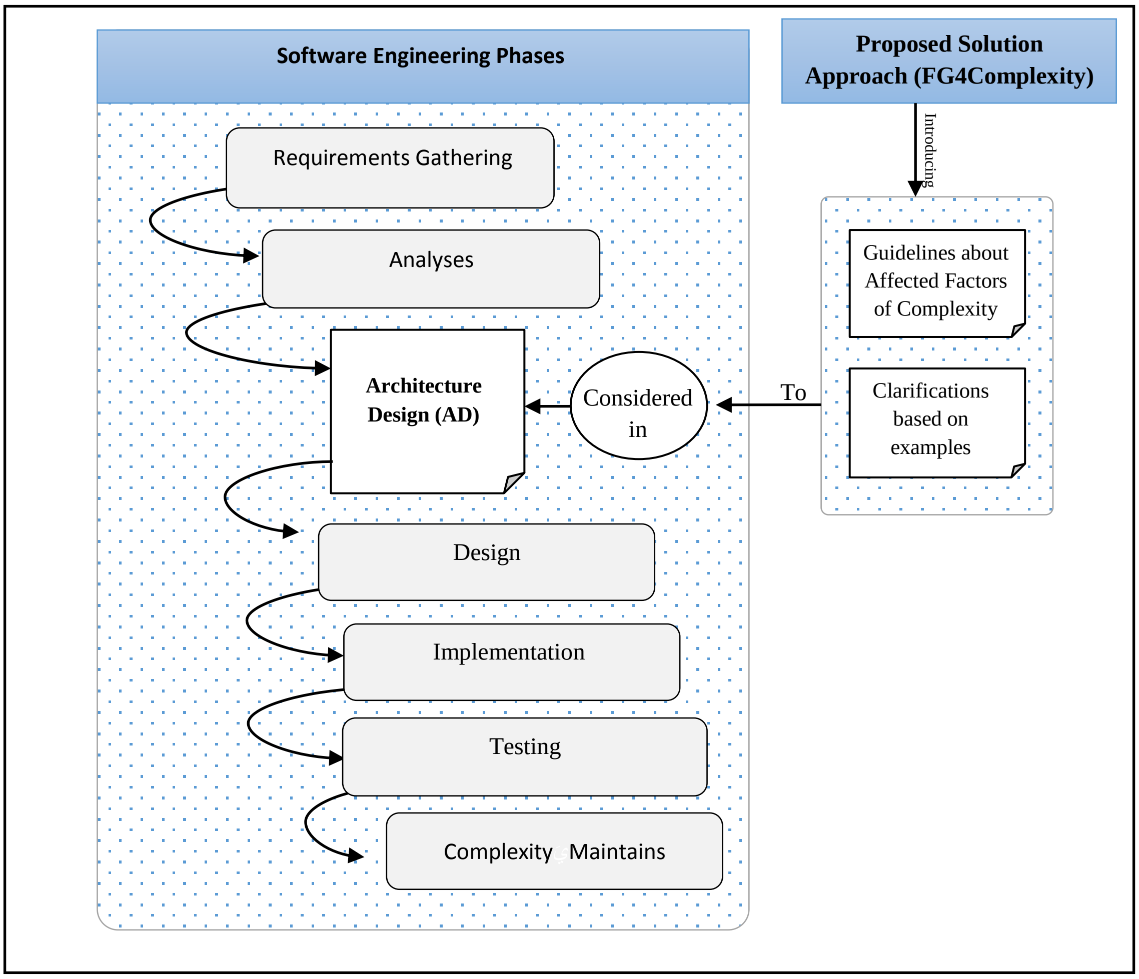

Software Engineering Phases
Proposed Solution Approach (FG4Complexity)
Introducing
Requirements Gathering
Analyses
Architecture Design (AD)
Considered in
To
Guidelines about Affected Factors of Complexity
Clarifications based on examples
Design
Implementation
Testing
Complexity Maintains

Figure 4.1: The position of architecture design in SDLC

It is useful to consider some of the characteristics and the concepts of both phases including the essential effects on analysis and design quality of architecture leading to approaching the problem of complexity. Figure 4.3 illustrates the methodology steps.

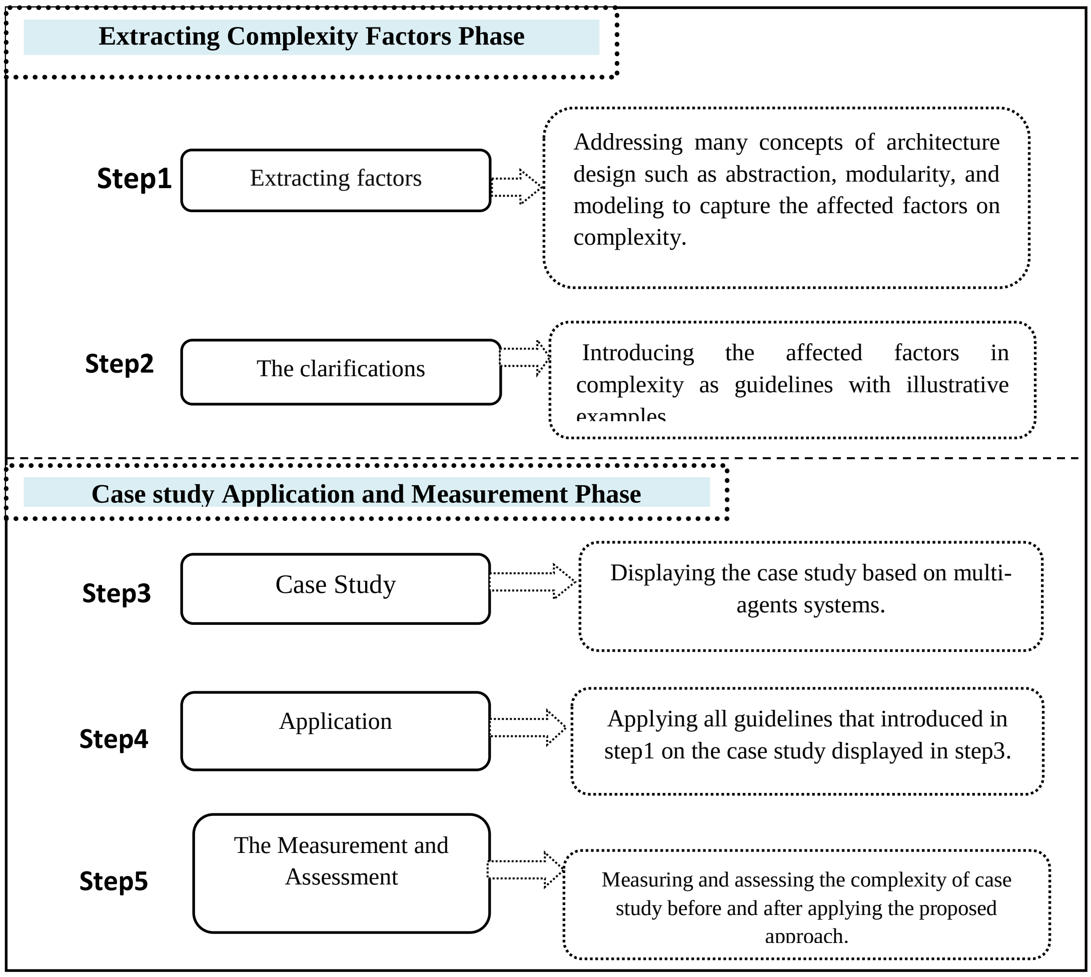

Extracting Complexity Factors Phase
Step1
Extracting factors
Addressing many concepts of architecture design such as abstraction, modularity, and modeling to capture the affected factors on complexity.
Step2
The clarifications
Introducing the affected factors in complexity as guidelines with illustrative examples
Case study Application and Measurement Phase
Step3
Case Study
Displaying the case study based on multi-agents systems.
Step4
Application
Applying all guidelines that introduced in step1 on the case study displayed in step3.
Step5
The Measurement and Assessment
Measuring and assessing the complexity of case study before and after applying the proposed approach.

Figure 4.3: An overview Methodology Steps

4.2. The Concepts of Analyses and Design

There are many characteristic concepts of analysis and design phases during the development process. For example, analyses phase is expressed by using the diagrams, maps, models, prototypes … etc. which work as a link between developers and users of software system to understand the requirements (Chakraborty, Baowaly, Arefin, & Bahar, 2012), (S.Mary Helan Felista1, 2014), (Moertini, Heriyanto, & Nugroho, 2014).

Correspondingly, the design phase (especially in object oriented environment in which agents systems are able to adapt) is expressed by abstraction, patterns, modularity, information hiding, function independency…etc. (Sękala, Foit, Banaś, & Kost, 2015), (Ghasemi, Sharafi, & Arman, 2015).

In this research work, we introduce some guidelines and factors that have a great impact on complexity of architectures design based on three concepts. These concepts are modeling, abstraction and modularity as shown in Figure 4.2. Each concept plays an important role to manipulate the complexity, especially in large and complex systems as illustrated in following sections.

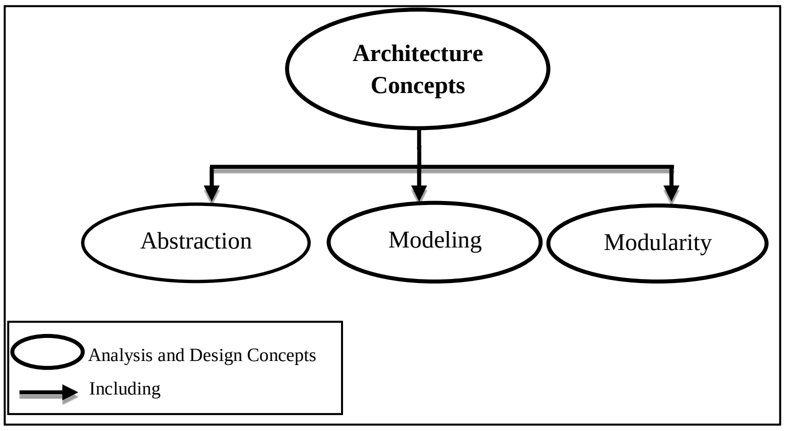


Figure 4.2: The architecture concepts addressed in FG4 Complexity

4.3. Reducing the Complexity of architecture design using Abstraction, Modularity and Modeling

The following sections provide a description of some concepts that can affect complexity of architecture design and how they reduce the complexity.

4.3.1. The Role of the Modeling

The modeling in software engineering is considered as a fundamental to understand and clarify many complex systems. By complex system, we mean a system that requires many components and interactions like systems based on multi-agents system (Goel, Rugaber, & Vattam, 2011). It depicts the software system by using multiple graphical and textual notations. Furthermore, the software modeling is taken as the documentation process for software systems in analysis and designs phases. In analyses phase, the modeling is used to help the analysts to understand and document the requirements from the users for example using prototypes method. However, in the design phase, modeling facilitates the understanding and the documentation of the design, especially in complex systems.

Modeling is an integral part of modern software development, in particular these models can be reused in different systems (Haber, 2011).

The modeling also reduces the ambiguities that occur in natural language descriptions, helps to understand the software system, help the developers to envision the system, discuss alternative designs; and make an architecture design decision.
There are many models used in analysis and design phases. For example:

- **Entity Relationships Diagrams** (ERD) which were originally proposed by Peter Chen in 1976. They are graphical representation of entities and their relationships to each other. Typically, they are used for modeling the organization of data within databases or information systems (Al-Masree, 2015).

- **Data Flow Diagrams** (DFD)
  Which demonstrate the data store, external entities, data flow in system and connecting data flow in other systems. There are only four notations for a data flow diagram: squares, circles (or Rounded Rectangles), arrows, and open-ended rectangles. Squares or Ovals represent external entities which are person or a group of people outside the control of the system being modeled. The circle or rounded rectangles represent processes within the system to show a part of the system that transforms inputs into outputs. The name of the process in the notations regularly clarifies what the process does. The arrows represent the data flows which can be either electronic or physical or both. The name of the arrows represents the meaning of the packet (data or items) that flow along. Moreover, arrows in data flow diagrams show direction to indicate whether data or items are moving out or into a process. The open-ended rectangles represent data stores, including both electronic stores and physical stores (Aleryani, 2016).

- **Unified Modeling Language** (UML) which was developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in 1994–1995. It is originally based on the notations of the Booch method, the object-modeling technique (OMT) and object-oriented software engineering (OOSE), which has integrated into a single language. UML has been developed into several versions as (UML, UML1.x and UML2.x). It includes diagrams, the static, the behavior and the interactions in software systems. These diagrams can be used in analysis and design phases (Alhumaidan, 2012), (Bartz, 2013).

At any rate, as we attempted to point out that the architecture design between the analysis phase and the design phase in SDLC. In this sense, there is a conceptual gap between analysis diagrams (such as Use Case diagram) and design diagrams (such as sequence, activity, and state diagrams) in architecture design's level (Alhumaidan, 2012).

For this reason, in this chapter, we promote establishing the architecture design by using a kind of notations to bridge the gap between these diagrams, in particular, to illustrate the details of agent system scenarios. Figure 4.3 clarifies using the notations of use case maps.

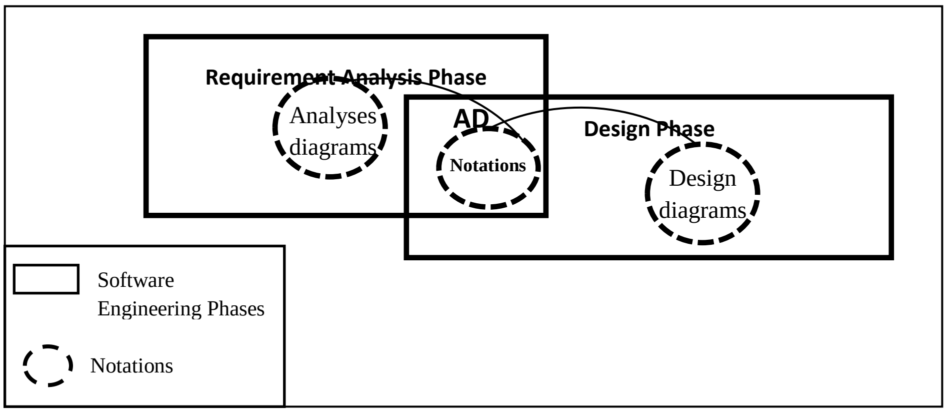


Figure 4.3: Using the maps to describe the architecture design

4.3.2. The Role of the Abstraction

The essence of abstraction is to extract necessary properties while omitting unnecessary details (Ross, Goodenough, & Irvine, 1975). One of the most fundamental objectives to engage in the task of abstraction in software analysis and design is to reduce the complexity. It is an essential concept of software design which controls the system description to a certain level based on components, relationships, interactions among the components, and simplify the composition of components into systems (Tsui et al., 2011). Modeling and abstraction are sometimes used together. For example, when architecture is modeled, the concentration is at a high level, while modules and their relationships ignore their internal structure.

From available literatures, there are various levels of abstraction based on main components that should be taken in consideration. Most developments are based on two levels. The first level is including basic components and its relationships while the second level is including more details than the first level. Also, these levels have different names such as (reduction, generalization), (specification, realization) and etc (Tsui et al., 2011**).**

In this research work, we also have two levels for using the specification to characterize the first level of abstraction and realization to characterize the second one.

4.3.3: The Role of the Modularity

In software engineering, modularity (Decomposition) refers to the extent to which a software is divided into smaller modules and how to keep the complexity of large systems under control and manageability (Ghasemi et al., 2015).

Modularity is the key property of software quality. Therefore, a high modularity improves the flexibility and understandability of the agent systems.

Moreover, complexity is revealed by both cohesion and coupling (were illustrated in chapter2). Higher cohesion indicates lower complexity, when coupling multiplies, the complexity also multiplies. Coupling, cohesion, and complexity relate strongly to the software maintenance work (Alenezi & Zarour, 2015), (Darcy, Daniel, & Stewart, 2010), (Alenezi & Almustafa, 2015).

A module having high cohesion and low coupling is functionally independent of other modules (Saxena & Kumar, 2012).

In addition, the agent system has the same concepts and phases in software developments life cycle. Figure 4.4 illustrates the main concepts which are addressed in FG4 Complexity approach.

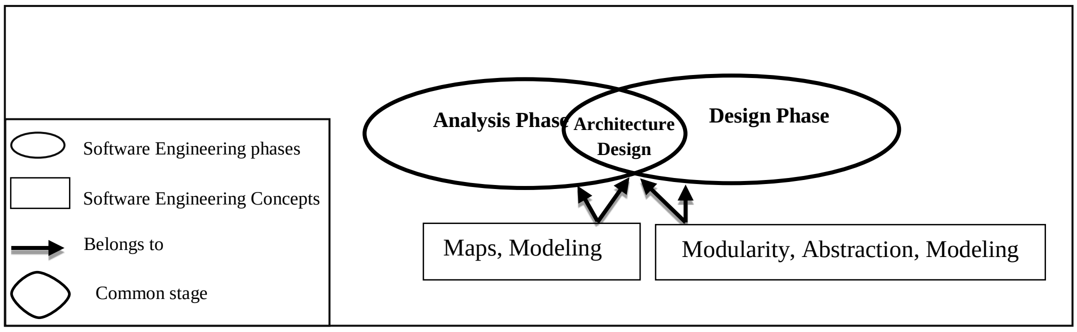

Analysis Phase
Architecture Design
Design Phase
Software Engineering phases
Software Engineering Concepts
Belongs to
Common stage
Maps, Modeling
Modularity, Abstraction, Modeling

Figure 4.4: The concepts of analyzing and design which were addressed in FG4 Complexity approach.

4.4. Factors and Guidelines (FG)

In this section several factors and guidelines are presented to decrease the complexity in architectures of multi agent systems. Each FG is established based on developer's previous practice or experimental methods. Each FG has a clarification part which is provided to illustrate the FG role in reducing the complexity and how to use certain rules if those rules were found. The FG is extracted from concepts which are related to software architecture ,then presented as symbols to beused in application phase. For example, the FG is related to modeling concept and represented by FGMOL symbol. The FG is related to abstraction concept and represented by factors and guidelines of abstraction (FGABS) symbol and the FG is also related to modularity concept and represented by factors and guidelines of modularity (FGMOR) symbol. Each FG should be numbered for example, FGABS 4 means the factor and guideline number 4 in abstraction concept section, FGMOL 2 means the factor and guideline number2 in modeling concept section as illustrated in Table 4.1.

Table 4.1: The symbols interpretation of architecture concepts

Architecture Concept	symbols	Symbols Interpretation	Instances
Abstraction	FGABS	Factors and Guideline of Abstraction	FGA1….i where I is Integer number
Modularity	FGMOR	Factors and Guideline of Modularity	FGM1….i where I is Integer number
Modeling	FGMOL	Factors and Guideline of Modeling	FGMOD1….i where I is Integer number

4.4.1. Factors and Guidelines for Modeling (FGMOL)

FGMOL1. Using Use Case Maps (UCM) to clarify the most relevant, interesting, and critical tasks of multi-agents systems (Lawgali, 2017).

The Clarification:

UCM act as a bridge between requirements analysis and design phases. It provides a behavior structure for evaluating architecture decisions at a high level of design. In this context, these maps can become applicable on architecture design at the same stage (After requirements analyses and before design).

It can also be used to emphasize the tasks (Responsibilities) of multi-agents systems along paths among components and clarify the interaction. There are many notations using in UCM. The following example illustrates the usage of UCM method through focus on some notations such as: Task, component, path of scenario, (start and end points of scenario), and the interactions among components.

Example

The example describes a simple UCM where a user (Nancy) attempts to make a phone call with another user (Jack) through a network of agents. Each user has an agent responsible for managing subscribed telephony features. Nancy first sends a connection request (req) to the network through her agent. This request causes the called agent to verify (vrfy) whether the called outcome is idle or busy (conditions are italicized). If he is, then there will be some status update (upd) and a ring signal will be activated on Jack's side (ring). Else, a message stating that Jack is not available will be prepared (mj) and sent back to Nancy (msg). A scenario starts with a pre-condition (filled circle labeled req) and ends with one or more resulting events and/or post-conditions (bars), in our situation ring or msg.

The responsibilities (vrfy, upd, mj) have been activated along the way. In this example, the responsibilities are allocated to abstract components (boxes Nancy, Agent A, Jack and Agent B), which could be realized as objects, processes, agents, databases, even roles, actors, or persons.

The structure of a UCM can be formed in different ways (views). For example, one may start by identifying the responsibilities (Figure 4.10 (a)). They can then be allocated to scenarios (Figure 4.10 (b)) or to components (Figure 4.10 (c)). Eventually, the views are merged to form a finishing map (Figure 4.10 (d)).

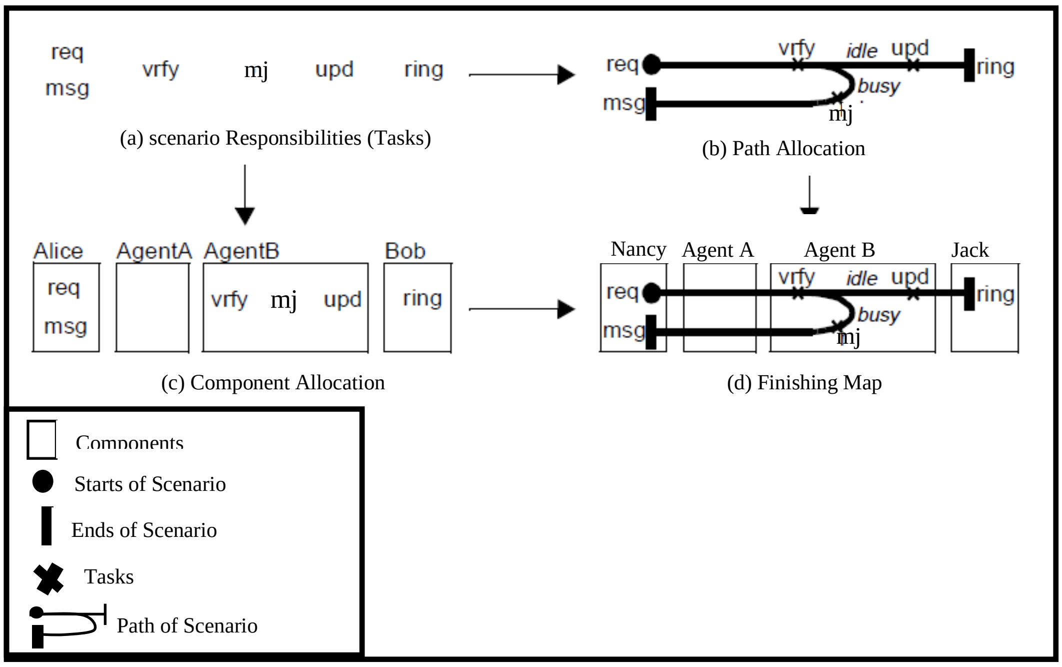
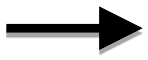
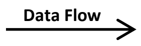
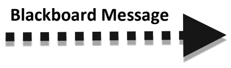
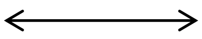


Figure 4.5: The Use Case Map construction

FGMOR 2. Using simple notations is very important to enhance understandability and decrease complexities in architecture design such as arrows, components, domain and etc. (Zalewski, 2013).

The Clarification:

According to available literatures, there are a lot of various notations used to describe the architecture design of software systems. Some of these notations are simple and intuitive while others need to be understood. To model the software architecture, we need to capture some aspects such as components, interactions, and context then model them. In the context of avoiding the complexities that arise from misunderstanding we suggest some simple notations are proposed and used to describe the architecture as shown in Table 4.3.

Description	Notations
Bold arrows to represent the messages among agents through the interactions.	
Normal arrow to represent the dataflow	
Dotty arrows to represent the messages which are exchanged from extra system such as the black board system.	
Doubly directions arrows represent the dataflow if it is the same exchanged between two components.	

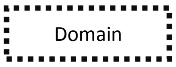
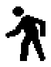
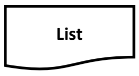
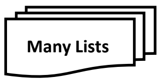
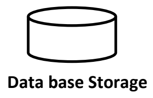
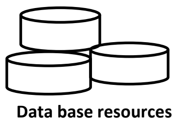
Dotty rectangles to represent the domains.	
Distinguish component to represent Agent.	
Distinguish component to represent list.	
Distinguish component to represent many lists.	
Distinguish component to represent data base storage.	
Distinguish component to represent data base resources.	

Table 4.2: The proposed notations

4.4.2. Factors and Guidelines for Abstraction (FGABS)

FGABS1. Developers should use Simplifying Abstraction type if they want to decrease the dynamic complexity type (Wagner, 2011).

The Clarification:

As we pointed out in chapter 2, there are two types of abstraction. The first type is called Simplifying Abstraction (the transition from the middle level to the top level of abstraction), and the second one is generalizing abstraction (the transition from the lowest level to middle level of abstraction). Simplifying Abstraction is the type of abstraction that is used when we want to reduce dynamic complexity and generalizing abstraction is used if we have several components that have many similarities and only differ in some aspects. In fact, this type is very useful if we need to reuse the design. The first type of abstraction is more abstract than the second one. Although, the developers always make a generalizing abstraction before they use Simplifying Abstraction. By this, the parameters and their types are identified before bringing them together to a more abstract design.

There is simple example of Class (Ahmed.A, 2013) or software module of library system to clarify the alteration to Simplifying Abstraction as follows.

Suppose we have GUI modules of agent system that describes many dialogs, for example:

- A is the root dialog which includes a chosen item from the library.
- A1, A2 are both GUI dialogs windows.
- Ag is the window title (String) and linked to the root dialog.
- P is the (parameter) which consists of variable T (Title Name).
- t1, t2 are different titles, for example t1 is "Choose the Book" and t2 is "Choose the Magazine".

By using simplifying abstraction we should abstract the modules of agent system from detailed concept in Figure 4.5 part (A) to make it more comprehensible. This means we should apply the following steps.

- Transition from the middle level to the top level of abstraction.
- Low level will be ignored.
- Removing each parameter in middle level (in fig. 4.5 B we should remove the parameter (P) completely by abstracting from Ag to A. This makes the usage of A simpler and less complex than the usage of Ag).
- Adding appropriate name of abstraction to describe what have been removed in fig (4.5 B) we using (Choose item) as appropriate name.

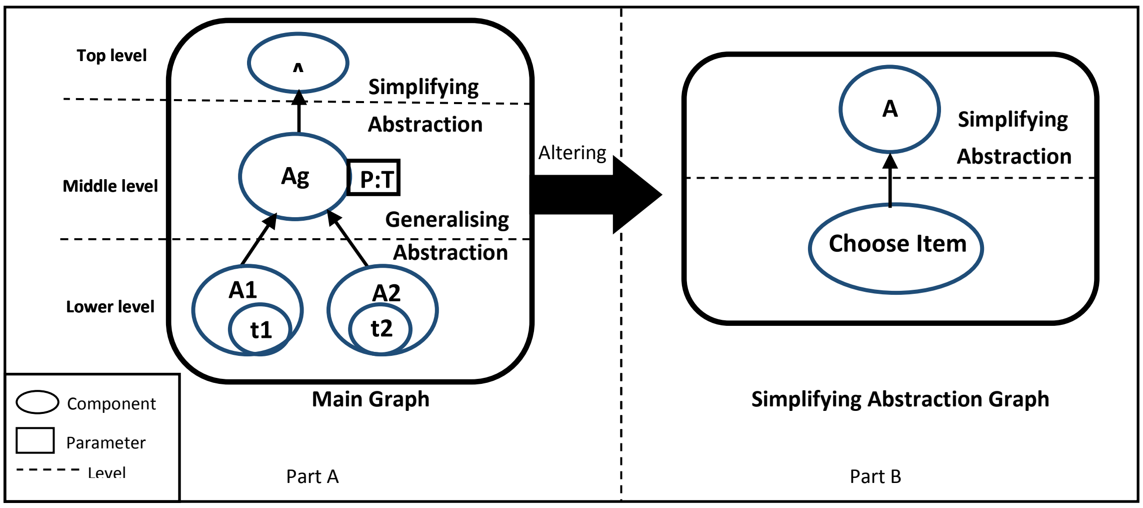


Figure 4.6: The altering to simplifying abstraction

FGABS2. Choosing the appropriate level of abstraction (Tsui et al., 2011).

The Clarification:

Taking the appropriate level of abstraction is a very important task for developers to increase understanding; thus, decreasing the complexity by using the abstraction levels. In this work, the architecture design will be described based on two levels of abstraction high level (specification) and detailed level (realization).

Figure 4.6 explains the high level and the detailed level. The first level specifies the main components and its relationships; while, the second level realizes more details than the first one.

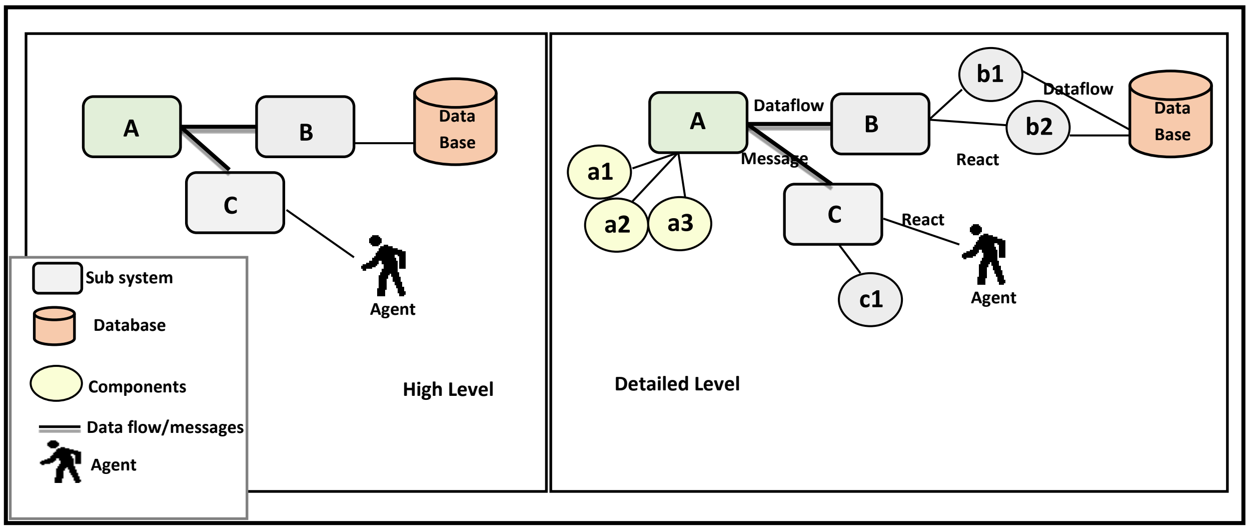


Figure 4.7: The high and detail levels of abstraction.

Moreover, the highest level of abstraction should be avoided as possible because it leads to difficulty in understanding the architecture design; therefore, more complex design is produced. It is a common saying in software designing "we cannot understand something if it is too abstract" (Wagner, 2011). By the same mechanism for architecture design, we should represent the components, the abbreviations, notations, names of components and dataflow and messages by using clear terms. Whereat, the purpose of abstract is to simplify complex systems but not to abstract them at the miss information degree, which also leads to the difficulty of understandability. This means it rises the level of the complexity again. Figure 4.7 illustrates the relationship between abstraction and complexity.

FGABS 3. Avoid to adopt the concept of gold plating (Kirandeep Kaur, 2013).

The Clarification:

Gold plating is the act of giving the customer more than what he originally asked for. This addition of system functions is reflected on the abstraction task of software system that is undesirable. It is usually performed to make the client happy and pleased; although, it makes the architecture design more have complex components.

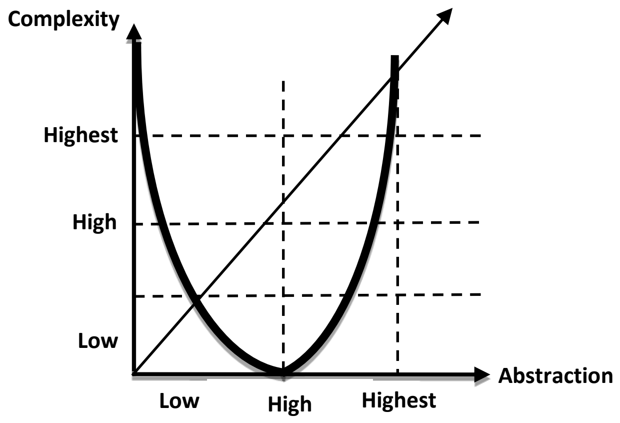


Figure 4.8. The relation between complexity and abstraction

4.4.3. Factors and Guidelines for Modularity (FGMOR)

FGMOR 1. Using Hierarchical Decomposition Approach (HDA) which considers a major method of handling complexity in conventional software analysis and design (Far, 2002), (Medeiro, Pérez-Verdú, & Rodríguez-Vázquez, 2013), (De Bruin, 2003).

The Clarification:

HDA involves the top-down design which starts by defining the top level components. This design contains the main components. After this, sub components are defined in the lower-level. This decomposition in each level is effective for controlling complexity (if it enforces information hiding) by demanding lower level components as explained in the next example (Far, 2002).

Example: The example illustrates how using HDA to design particular software of digital clock as Figure 4.8 shows.

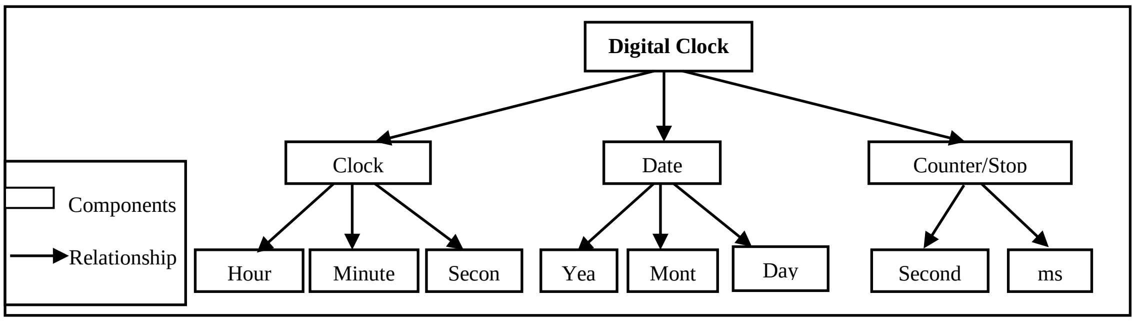


Figure 4.9: Illustrating the Hierarchical Decomposition Approach (HDA)

FGMOR 2. It is useful to establish the software modularity based on roles or measurements such as Cohesion Communication Measurement (CCM) (Misra, 2011).

<u>The Clarification:</u>

It is crucial to realize that the complexity of any system stems from a large number of system components and interaction required between these components. This is brought out clearly in large and complex system as multi-agents systems. If this is the case, then, the modularity rules needs to be taken as the important issue to manage a complex system designs. This complex design is comprised of multiple agents and interactions. In this sense, the modularity concept could be decomposed in components and again the components into sub-components till some basic entities are obtained. The measurement of communication cohesion introduces approximate ratio to internal interactions on external interactions for each agent. After applying CCM, the observed results if CCM ≥ 0.91 of the Agent, then it will be targeted for further decomposition. Hence, FGMOR2 is based on measurement principle during architecture design phase. According to this measurement decomposition produces independent results. Figure 4.9 illustrates CCM mechanism, and Table 4.2, demonstrates more decomposition.

CCM Mechanisem	
$\mathrm{CCM}\,(A_i) = \dfrac{R_{\mathrm{internal}}}{R_{\mathrm{internal}} + R_{\mathrm{External}}}$	
Abbreviation	**Illustration**
CCM	Communicative Cohesion Measurement defined in terms of the ratio of internal relationships (interactions) to the total number of relationships.
(Ai)	**(A)Agent,** where i=1 to N. Example: Agent1, Agent2 and etc.
R internal	Internal interaction
R external	External interaction

Table 4.4: The abbreviations of CCM metric

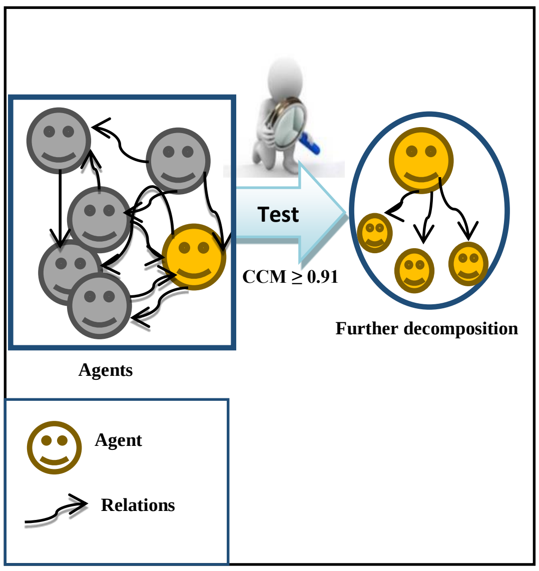


Figure 4.10: The agent targeted to further decomposition

4.5. The Clarifications of FG4Complexity Approach.

Some clarifications of FG4Complexity approach are summarized in Table 4.4.

FG4Comlexity Approach	Based on	RULE	Supports the understandability and analysis of AD	Supports the understandability and design of AD	Notes
FGABS 1	Experimental Method	-	-	✓	An example was Explained
FGABS2	Previous Experiences	-	-	✓	Clarify it by illustrative figure
FGABS 3	Previous Experiences	-	-	✓	Clarify it by illustrative figure
FGABS 4	Experimental Method	-	-	✓	-

FGMOR 1	Experimental Method	-	✓	✓	An example was Explained
FGMOR 2	Measurement rule	$\mathbf{CCM\ (Ai)} = \frac{\mathbf{R}\ \text{internal}}{\mathbf{R}\ \text{internal}+\mathbf{R}\ \text{External}}$	-	✓	how to use the rule was explained In FGMOR2
FGMOL 1	Maps	-	✓	-	An example was Explained
FGMOl 2	Notations	-	✓	✓	Suggested some simple notations

Table 4.4: Some clarifications about FG4Complexity approach

4.6. Summary

- The chapter addressed the problem of architectures design de complexity and proposes an approach called FG4Complexity.
- FG4Complexity comprises of a set of factors and guidelines extracted from many concepts related to analysis and design.
- Abstraction, Modularity and Modeling are three substantial concepts of architecture design and were addressed to support the approach.
- Each FG of FG4Complexity approach has a direct impact on the analysis or the design processes for the architecture.
- More details about FG4Complexity approach were given in Table 4.4.

CHAPTER 5
Case Study Application and Measurement

This chapter is mainly concerned with the case study which is based on multi agent system, for applying the proposed approach. The application steps will be applied via some models used in methodologies related to agents systems such as HLIM (Elammari & Lalonde, 1999), MASD (Abdelaziz, Elammari, Unland, & Branki, 2010). Figure 5.1 gives a simple overview of models of HLIM methodology.

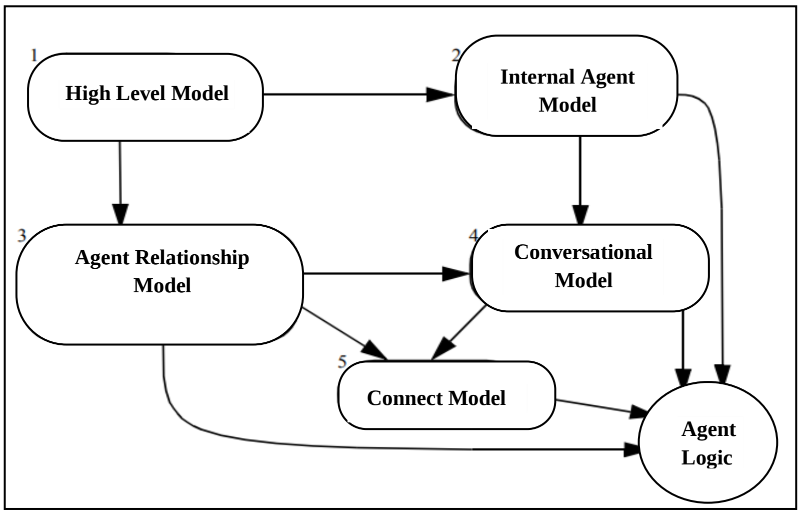


Figure 5.1: High-Level and Intermediate Models (HLIM)

This case study addresses briefly the nature of the system, the possible scenarios of the system, the internal structure of the agents, their tasks, the agent's relational model the agent conversational model, and finally demonstrates the conceptual architecture of the system. The application includes the application steps. Each step consists of proposed FG which to apply later on the system in this case study. Then, it introduces the tools for measuring the complexity of certain aspects of the system by using appropriate software engineering techniques. Eventually it leads to the results of measurement. Based on the results, the system is assessed by using the FG4Complexity approach followed by the summary of this chapter.

5.1. The Case Study

The case study is a "books recommendations system" based on multi –agents system to help users to select books by providing three scenarios. The system can switch to three recommendation approaches Content-based filtering approach (CBF) (M. Montaner, 2003), (Castillo, 2007) Collaborative Filtering approach (CF) (Itmazi, 2005), (Obando, 2008) and knowledge based approach (KBA) (Burke, 2002), (Cohen, 2000). The first scenario is that the user wants recommendations based on his/ her current needs entered into the system. The second scenario, the new user requests books recommendations based on his/her preferences entered into the system, and the third scenario registered user requests the books recommendations based on his/her preferences entered into the system. The agents within the system can exchange the messages among each other via one of agent communication languages. In this case study, the messages exchanged will be via Knowledge Query and Manipulation Language (KQML).

- **A brief illustration of recommendations system (RS).**

As we pointed out in Chapter 2, the recommendations system filters information fragment out of large amount of dynamically generated information according to each user's preferences, interests, or observed behavior about item by many approaches such as CBF, CF, KBA and other.

- The essential characteristics of the book recommender system in this case study are summarized below.

  - **User Profile**: Each user registered in the system has a profile that contains his/her preferences and all the information related to the books.

  - **Books Resources**: We assume that the books resources exist in external database contain also the additional information about these books, for example, their description and availability.

  - **The Knowledge Base**: Contains the essential knowledge about how a specific book meets the user's needs and it should be represented by knowledge base rules used with expert systems. (Engin et al., 2014)

▪ The scenarios.

The scenarios of this case study are the following:

• The First Scenario

A user wants recommendations based on his current needs entered into the system.

In this scenario, the Need Determination Agent (NDA) will interact with the user and gives him/her a range of questions to answer, thereby the agent can interact with the users via dialogues as the following:

- **System:** Please select the book category from the list.
- **User:** Selects a book of Artificial Intelligent
- **System:** Please select a book you like from the list.
- **User:** Selects a book he/she is interested in.

After the user has answers all the given questions, the NDA sends user's requirements to the Filtering Agent (FA). Figure 5.2 illustrates the messages from the need determination agent to the filtering agent via KQML.

```
(Inform
:Sender          Need determination agent

:receiver         filtering agent

:language         Agent Speaks

:ontology         Books recommendations

:content          User requirements
```

Figure 5.2: The message from the NDA to the FA

After this, FA sends a message to RA to request available books as shown in Figure 5.3.

```
(Request
:Sender          Filtering agent
:receiver         Retrieval agent
:language        Agent Speaks
:ontology        Books  recommendations
:content         Give me available books
```

Figure 5.3: The message from the FA to the RA

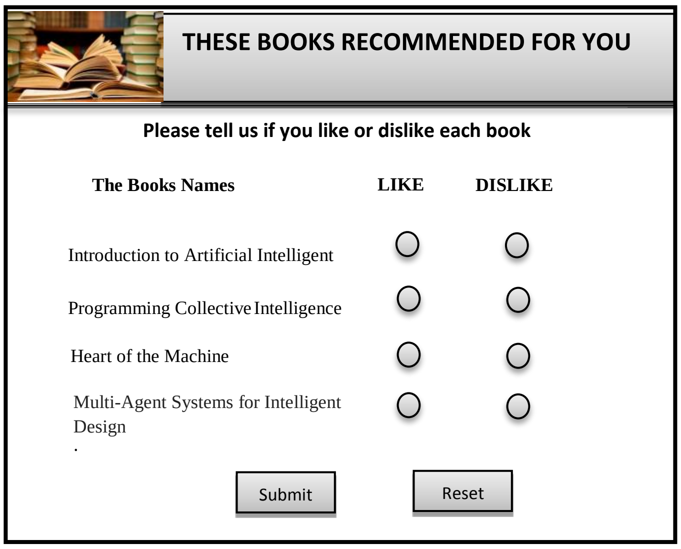
The RA receives the message and searches in the books database and then retrieves available books to send the content to the filtering agent as illustrated in Figure 5.4.

(Inform

:Sender	Retrieval agent
:receiver	Filtering agent
:language	Agent Speaks
:ontology	Books recommendations
:content	List of available books

Figure 5.4: The message from the RA to the FA

When the filtering agent receives the message, it applies the Knowledge based recommendation approach, then the recommendations are transferred to Graphical User Interface (GUI). Figure 5.5 shows the recommendations of books on GUI.



THESE BOOKS RECOMMENDED FOR YOU

Please tell us if you like or dislike each book

The Books Names	LIKE	DISLIKE
Introduction to Artificial Intelligent	○	○
Programming Collective Intelligence	○	○
Heart of the Machine	○	○
Multi-Agent Systems for Intelligent Design	○	○

Figure 5.5: Illustrating the GUI of scenario case1

- **The Second Scenario**:

A new user requests books recommendations based on his/her preferences entered into the system.

In this scenario, primarily the system has no knowledge about the user; as a result, the PA will create a new user profile and inform the filtering agent to give the recommendations to the user as illustrated in the next steps and Figure 5.6.

- **User:** Enter as a new user.
- **System:** Gather user information through his/ her behaviour via GUI to build his/her profile.
- **User*:*** Enters books names and browses the list of available books.

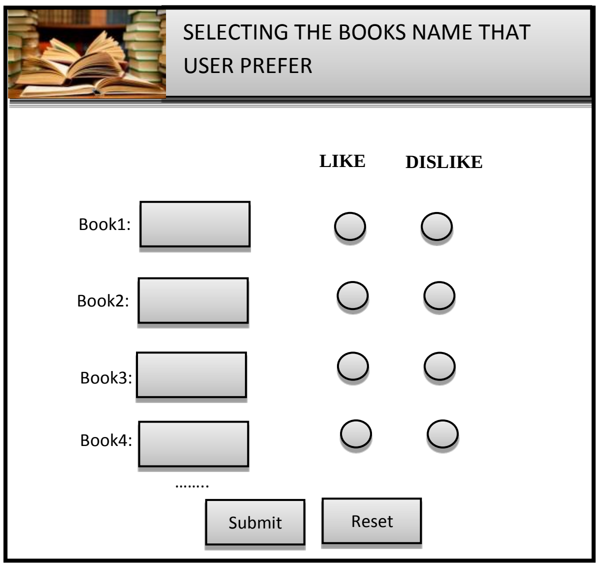


Figure 5.6: Illustrating the GUI of scenario case2

After the new user enters the preferred book names, and browses the available details on the book, the following steps will be done:

1. The PA will build a profile for the user which contains the user's preferences.
2. It will then send a message to the filtering agent. " There is a new recommendation request ".
3. After FA receives the message, it will apply the KB recommendation approach.
4. To compute the recommendations, the filtering agent will filter the content, and display the results to the user via GUI.
5. When the user submits his/her feedback about the recommendations to the system; the profiling agent will update the user profile.

Figure 5.7 showing the message from PA to FA.

(Inform

:Sender	Profiling agent
:receiver	Filtering agent
:language	Agent Speaks
:ontology	Books recommendations
:content	There is recommendation request

Figure 5.7: The message from the PA to the FA

- **The Third Scenario**:

A registered user requests the book recommendations based on his/her preferences entered into the system which can be summarized as:

- **User:** Enters the username and the password.
- **System:** Asks the user to enter additional books he/she prefers then monitors the user behavior within the page. After this, the system observes which books he/she is browsing to gather user's preferences and feedback for updating his/her profile. The system realizes all these via GUI.
- **User:** Enters books names and provides his/her feedback. Then, the PA will update the user's profile, and inform the filtering agent that there is a recommendation request.

  When the filtering agent receives the message in this scenario, there are two possible cases**:**

Case1: There are new books available, in this case the filtering agent will apply the CBF approach to produce the recommendations to the user.

Case2: There are no new books available, and in this case the filtering agent will apply the CF approach to produce the recommendations to the user.

5.2. The Agents Participant in System

The books recommender system consists of four basic agents as following:

1. Profiling Agent (PA).
2. Need Determination Agent (NDA).

3. Filtering Agent (FA).
4. Retrieval Agent (RA).
5. Translation Agent (TA).

5.3. The Internal Structure of Agents

This section illustrating the agents' characteristics such as goals, tasks, precondition, and postcondition to clarify the structure and behavior of each agent in the system as shows in Tables (5.1, 5.2, 5.3, 5.4. 5.5).

Profiling Agent				
N	**Goal**	**Precondition**	**Postcondition**	**Task**
1	Making the user profile	User wants recommendations based on his/her preferences	The user profile made	• Checking if a new user • Goal (gather the preferences)
2	Gathering the Preferences	New user log-in	Profile built	• Observing user behavior • Gathering explicit preferences • Building active user profile
3	Gathering the Preferences	Existing user log-in	Existing profile updated	• Observing user behavior • Gathering explicit preferences • Gathering relevance feedback • Updating the existing profile

Table 5.1: A profiling agent internal structure

Filtering Agent				
N	**Goal**	**Precondition**	**Postcondition**	**Task**
1	Delivering recommendations to the GUI.	Receiving a message from profiling agent or need determination	The recommendations transferred to GUI.	• Requesting books from Retrieval agent. • Goal (Check the message resource) • Goal (Filter the recommendations list).

		Agent.		
2	Checking the Message resource.	Receiving a message came from need determination Agent.	The recommendations Generated.	• Applying KB approach.
3	Checking the Message resource.	Receiving a message came from profiling Agent.	The recommendations Generated.	Checking if a new user, apply KB approach • Checking if there is a new book, apply CBF approach else applying CF approach • Checking if CF approach Failed, apply CBF.
4	Filtering the recommendations list.	Receiving recommendation list	The recommendations transferred to GUI.	• Comparing the active user profile with the recommendation list • Removing the book that user has known before from the recommendation list • Transferring the recommendations to GUI
5	Filtering the recommendations list.	Receiving recommendation List.	The recommendations transferred to GUI.	• Comparing the recommendation list with the available books list. • Removing the unavailable book from the recommendation list. • Transferring the recommendations to GUI.

Table 5.2: A filtering agent internal structure

Need Determination Agent				
N	**Goal**	**Precondition**	**Postcondition**	**Task**
1	Gathering user requirements.	User wants recommendation based on his/her current needs.	User requirements are gathered.	• Showing queries to the user. • Gathering user requirements.

Table 5.3: Need Determination Agent (internal structure)

Retrieval Agent				
N	**Goal**	**Precondition**	**Post-condition**	**Task**
1	Delivering the available books to the filtering agent.	Receiving the message from Filtering agent.	Available books List is sent.	• Extracting the books from the books database. • Preparing the available books list. • Sending the books list to filtering agent.

Table 5.4: A retrieval agent internal structure.

Translation Agent				
N	**Goal**	**Precondition**	**Post-condition**	**Task**
1	Translating books which user required.	Receiving the request from NDA.	Books are translated.	• Sending the translation request to BBS. • Receiving the translation books from BBS. • Resending the translation books to FA which sends it to GUI.

Table 5.5: Translation Agent (internal structure)

5.4. Agents and Their Tasks.

A brief summary of agents and their tasks in Table 5.6.

Agents	Roles (Tasks)
Profiling agent	• Gathering the user's preferences. • Gathering the relevance feedback. • Building and updating the active user profile.
Need determination agent	• Gathering the user current needs.
Filtering agent	• Producing the recommendations. • Removing the books that are not currently offered from the recommendation list. • Transferring the recommendation to the GUI.
Retrieval agent	• Retrieving the books that are currently offered from the books database. • Storing the available books in the recommender system database.
Translation agent	• Producing books translation service for users.

Table 5.6: The agents and their tasks

5.5. The Relational Model of Agents

The relational model of agents is represented in the interactions among the agents. Figure 5.8 shows these interactions.

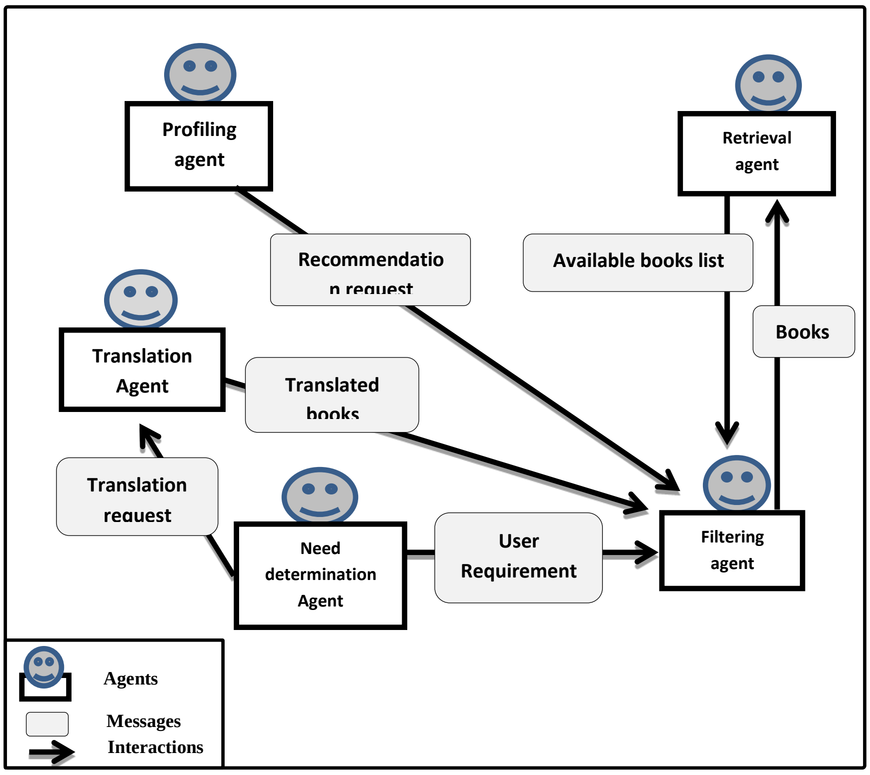


Figure 5.8: The agent relational model

5.6. Conversational Model

Conversational model is used to explain the exchanged messages among the agents in books recommender system as shows in Tables (5.7, 5.8, 5.9, 5.10, and 5.11).

	Receive	Send	Comment
1		inform(from :PA to: :FA msg: there is recommendation request)	

Table 5.7: The profiling agent conversational model

	Receive	Send	Comment
1		inform(from :NDA to: :FA msg: list of user requirements)	
2		inform(from :NDA to: :TA msg: Translate books requirements)	

Table 5.8: The need determination agent conversational model

	Receive	Send	Comment
1	inform(from :PA to: :FA msg: there is recommendation request)	Requesting (from :FA to : :RA msg: give me the list of available books)	
2	inform(from :NDA to: :FA msg: list of user requirements)	Requesting (from :FA to : :RA msg: give me the list of available books)	
3	inform(from :RA to : :FA msg: list of available books)		
4	infrom (from :TA to : :FA msg: translated books)		

Table 5.9: The filtering agent conversational model

	Receive	Send	Comment
1	Requesting (from :FA to : :RA msg: give me the list of available books)	inform(from :RA to : :FA msg: list of available books)	

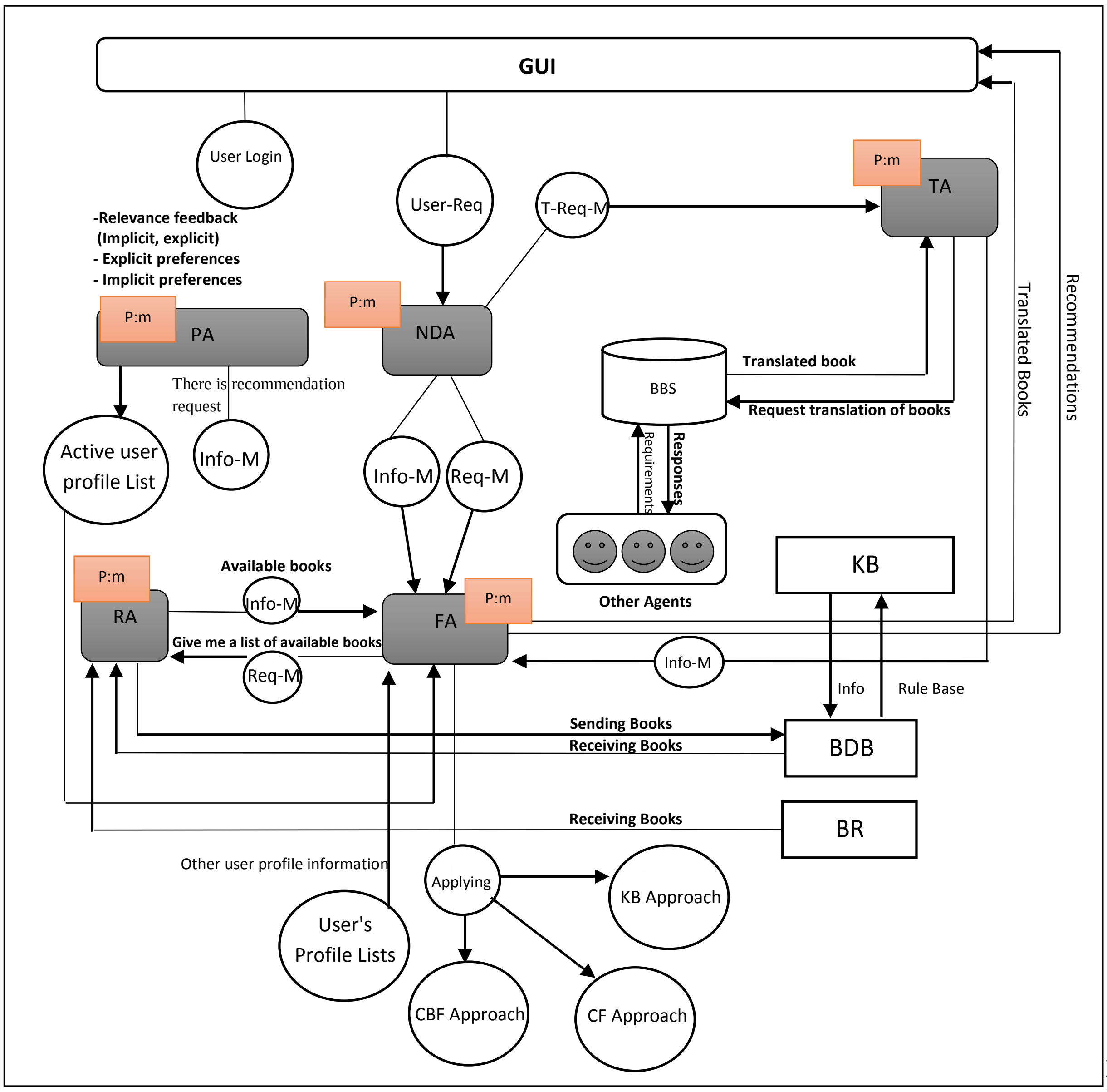
Table 5.10: The retrieval agent conversational model

	Receive	Send	Comment
1	inform(from :NDA to: :TA msg: requests list of translation books)	Requesting (from :TA to: :BBS msg request translation of a book)	

Table 5.11: Translation agent conversational model

5.7. Conceptual Overview of Architecture Design.

Figure5.9 showing an original architecture design of books recommendations system which consist of agents, graphical user interface, parameters, messages, dataflow, and components.



12

Figure 5.9: Conceptual overview of books recommendations system architecture design

shows abbreviations and meaning which used in Figure 5.9.

Abbreviations	Meaning
TR	Translation Request
T	Translation
P:m	Parameter (Message)
BDB	Books Data Base
BR	Book Resource
KB	Knowledge Base
Req	Request
Info-M	Message of information
Req-M	Message of Request
T- Req-M	Message of Translation Request
GUI	Graphic User Interface

Table 5.12: Illustrating the abbreviations of architecture design of the books recommendations system

5.8. The Application and Measurement

5.8.1. The Application

In this section, we first, introduce the application strategy of Factors and Guidelines for Complexity (FG4Complexity) in the proposed approach. The strategy includes four steps, and each step consists of Factors or Guidelines for complexity (FG) addressed in chapter which were be applied on the previous case study. These steps are intended to decrease the complexity in architecture design. Some steps are useful to apply in system analysis such as Step1 and step2. Correspondingly; Step3 and step4 attempt directly to decrease complexity in architecture design. Having applied all the steps, the same architecture will be reviewed using the guidelines. Figure 5.35 and Figure 5.36 illustrate how the architecture has become less complex than original architecture design as shown in Figure 5.9.

In general, the guidelines provided by the proposed approach allow us to use them according the situation. This means it is not necessary to apply them in arrangement.

- **The FG4 Complexity Approach Application Strategy**

As we have earlier pointed out that all the previous FG will be within 4 steps to correspond to the current case study as Figure 5.10 shows.

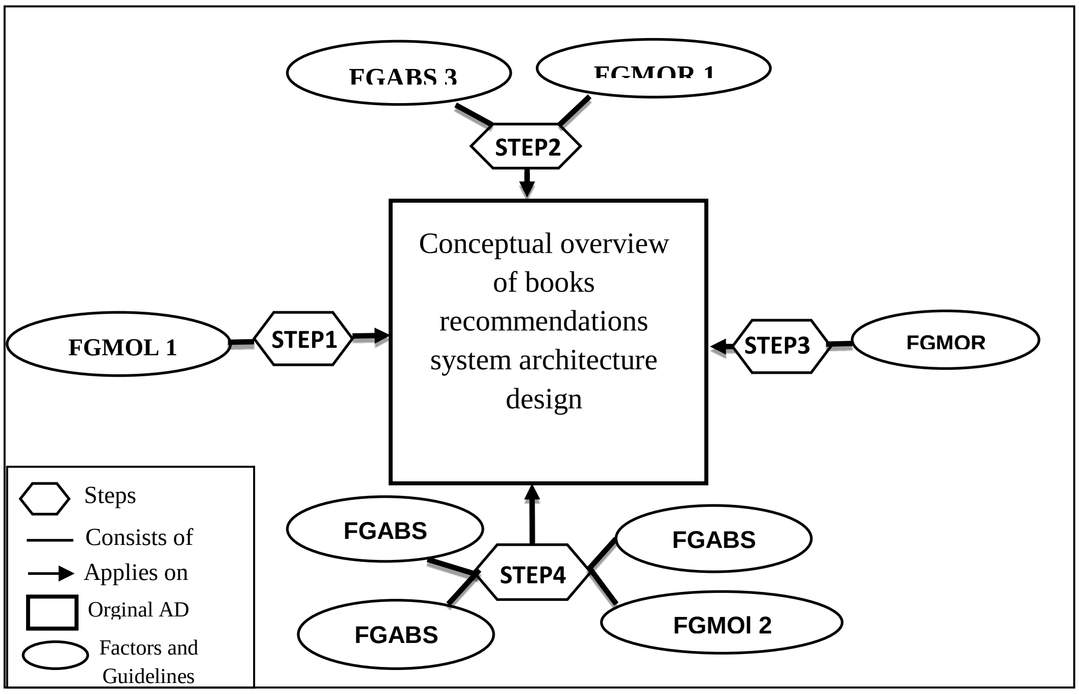
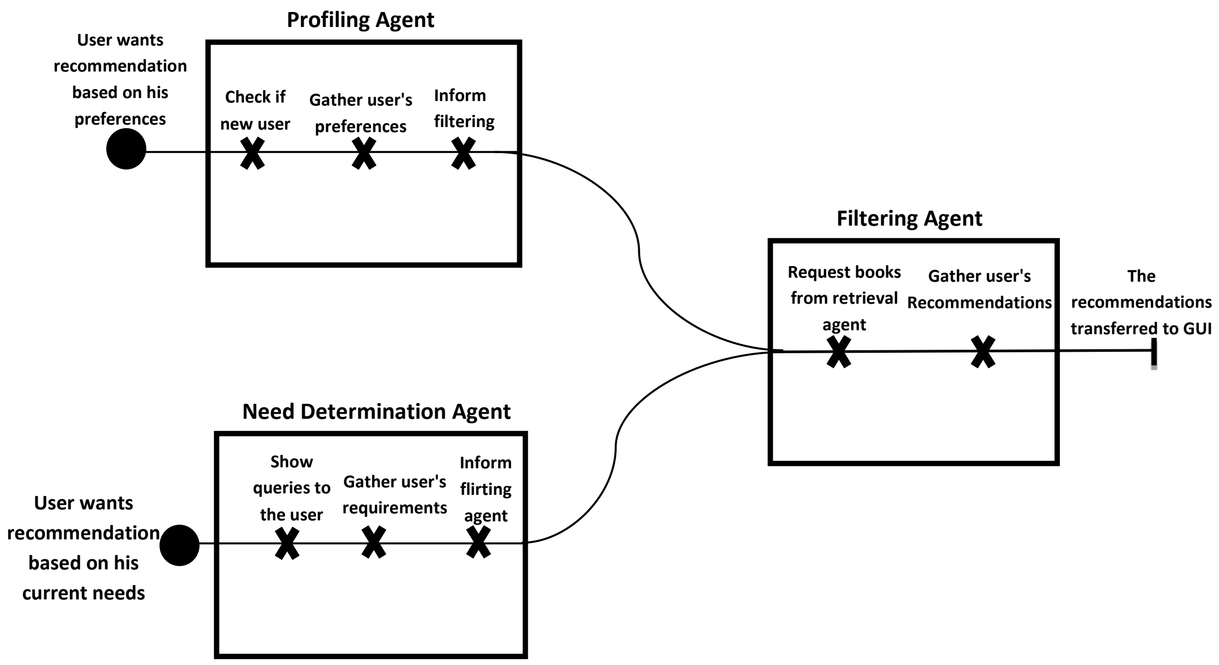


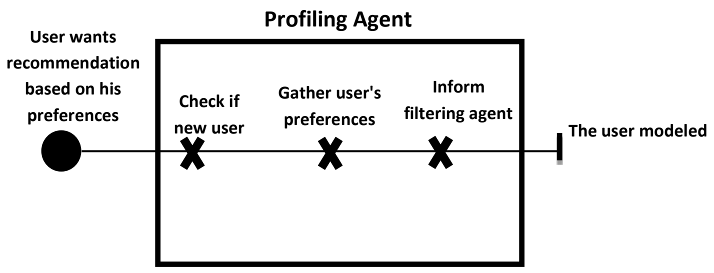
Figure 5.10: Illustrating of the applied steps on architecture Design

Step1. Initially, this step is based on applying use case maps represented in FGMOL1 of FG4Complexity approach which used in between analysis and design phases. These maps give high view of system specifically the responsibilities (Tasks) and interactions in a simple way, reinforce system understanding and overcome some situations of complexity such as intercommunication among agents. Figure 5.11 to Figure 5.30 illustrate how to use UCM's in descript the system through its tasks, some scenarios and the most significant interactions among agents system such as describe the mechanisms of a profiling, a filtering, a translation, a retrieval, a need determination agents, the messages among them, and etc through focused on some notations such as tasks components, and scenario's path. (frerjani, 2010), (Saleh, 2014).



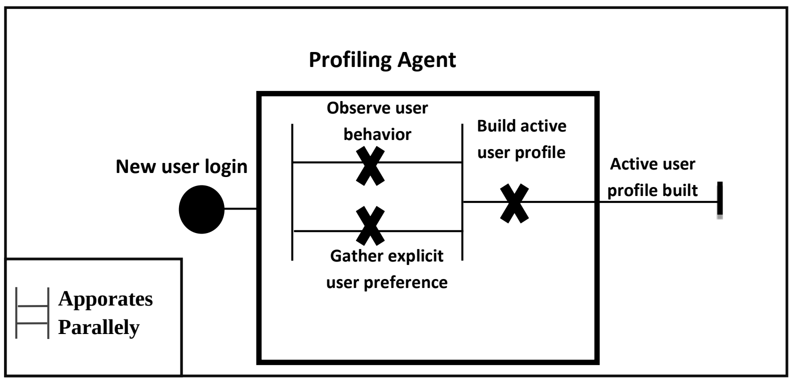
Profiling Agent
User wants recommendation based on his preferences
Check if new user
Gather user's preferences
Inform filtering
Filtering Agent
Request books from retrieval agent
Gather user's Recommendations
The recommendations transferred to GUI
Need Determination Agent
User wants recommendation based on his current needs
Show queries to the user
Gather user's requirements
Inform flirting agent

Figure 5.11: Using the UCM to describe the interactions between filtering agent and (NDA or Profileing agents)



Profiling Agent
User wants recommendation based on his preferences
Check if new user
Gather user's preferences
Inform filtering agent
The user modeled

Figure 5.12: Using the use case maps for profiling agent.



Profiling Agent
New user login
Observe user behavior
Gather explicit user preference
Build active user profile
Active user profile built
Apporates Parallely

Figure 5.13: Gathering the preferences in case of new user.

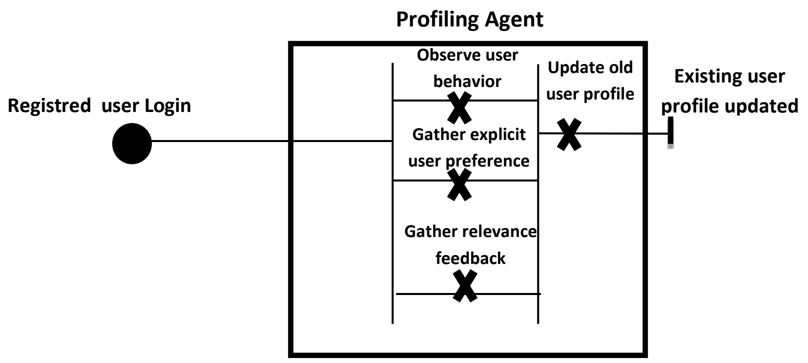

Profiling Agent
Registred user Login
Observe user behavior
Gather explicit user preference
Gather relevance feedback
Update old user profile
Existing user profile updated

Figure 5.14: Gathering the preferences in case of existing user.

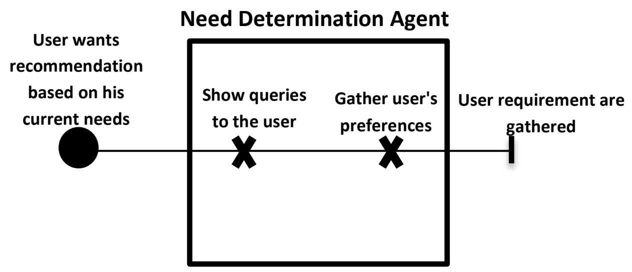

Need Determination Agent
User wants recommendation based on his current needs
Show queries to the user
Gather user's preferences
User requirement are gathered

Figure 5.15: Using the use case maps for NDA.

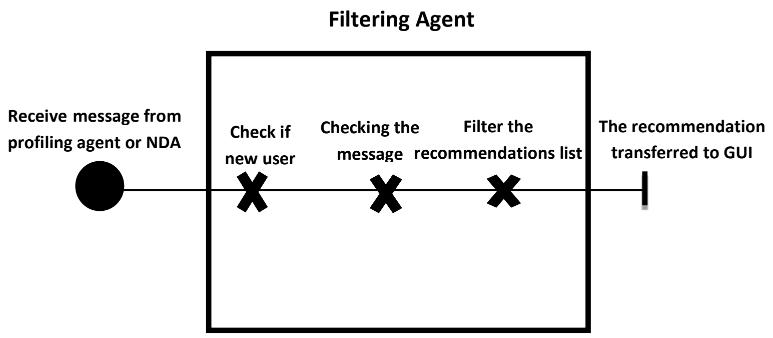

Filtering Agent
Receive message from profiling agent or NDA
Check if new user
Checking the message
Filter the recommendations list
The recommendation transferred to GUI

Figure 5.16: Using the use case maps for Filtering agent.

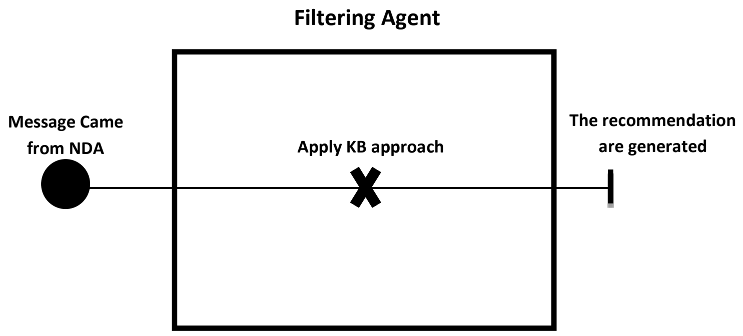


Figure 5.17: Receiving a message from NDA

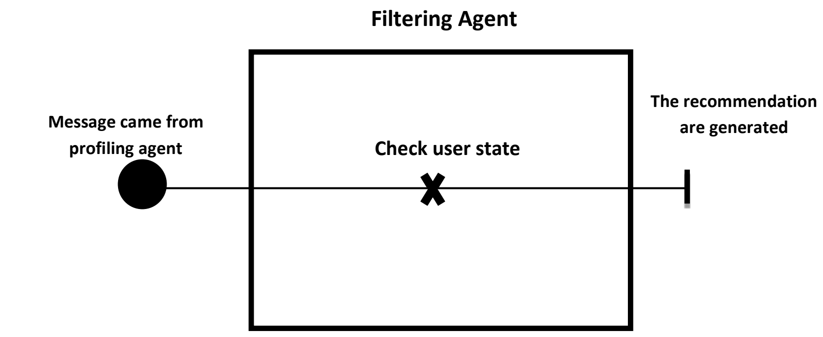

Filtering Agent
Message came from profiling agent
Check user state
The recommendation are generated

Figure 5.18: Receiving a message from profiling agent.

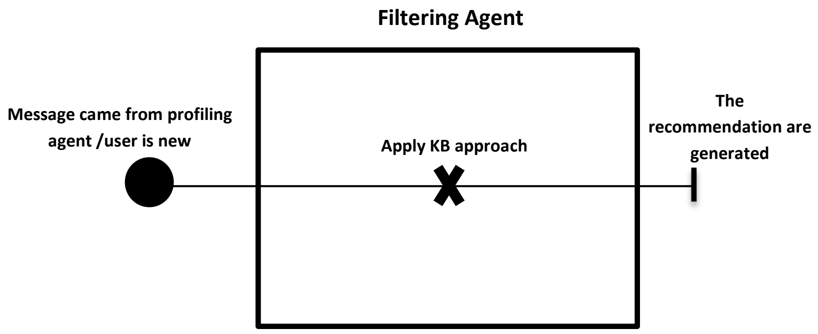

Filtering Agent
Message came from profiling agent /user is new
Apply KB approach
The recommendation are generated

Figure 5.19: Receiving a message from profiling agent case of new user.

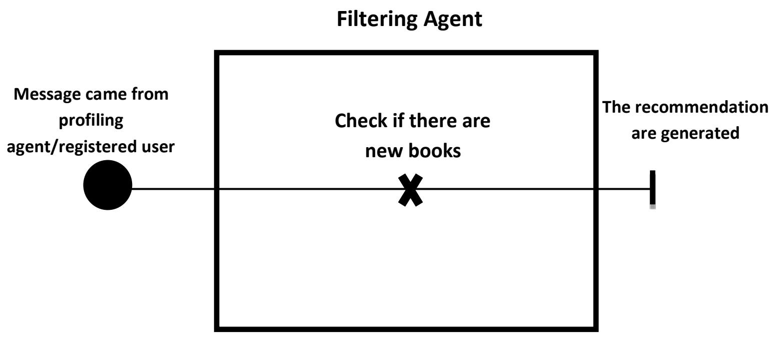

Filtering Agent
Message came from profiling agent/registered user
Check if there are new books
The recommendation are generated

Figure 5.20: The case of receiving a message from profiling agent for registered user.

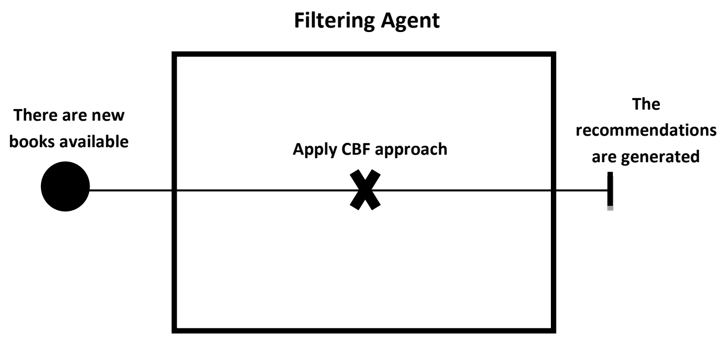

Filtering Agent
There are new books available
Apply CBF approach
The recommendations are generated

Figure 5.21: The case of availability of new books.

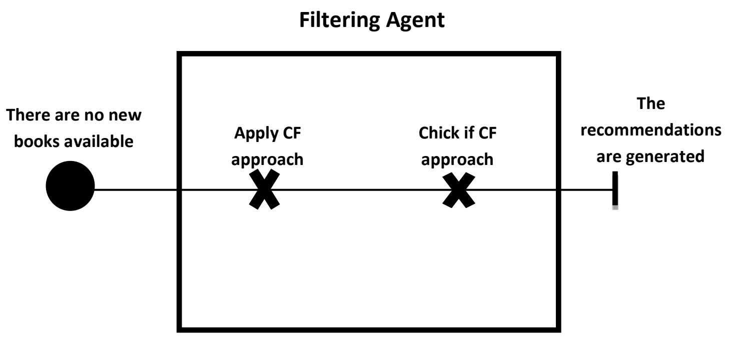

Filtering Agent
There are no new books available
Apply CF approach
Chick if CF approach
The recommendations are generated

Figure 5.22: No new books are available case.

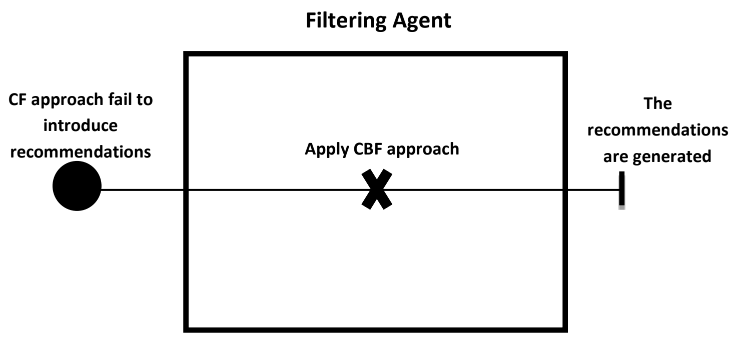

Filtering Agent
CF approach fail to introduce recommendations
Apply CBF approach
The recommendations are generated

Figure 5.23: The failure of CF approach to introduce recommendations case**.**

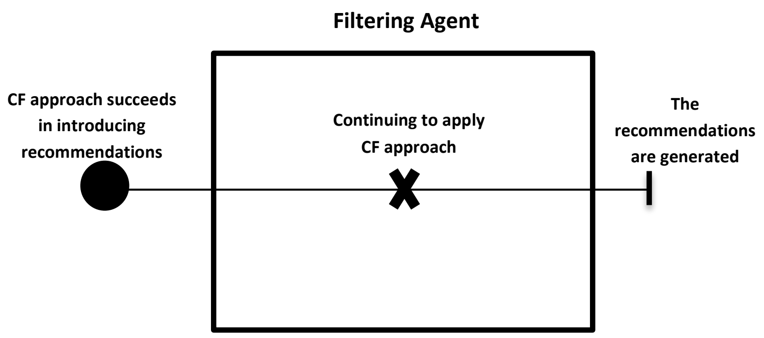

Filtering Agent
CF approach succeeds in introducing recommendations
Continuing to apply CF approach
The recommendations are generated

Figure 5.24: The success of CF approach to introducing recommendations case.

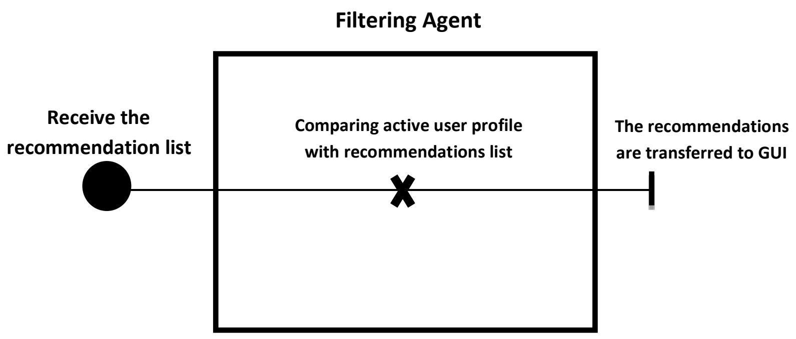

Filtering Agent
Receive the recommendation list
Comparing active user profile with recommendations list
The recommendations are transferred to GUI

Figure 5.25: Comparing the active user profile with recommendations list

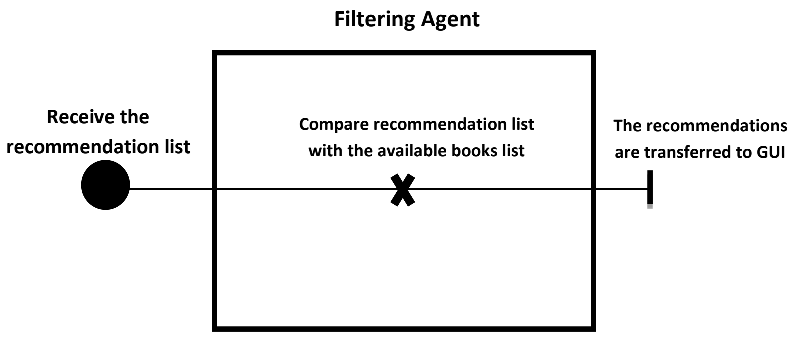

Filtering Agent
Receive the recommendation list
Compare recommendation list with the available books list
The recommendations are transferred to GUI

Figure 5.26: Comparing the recommendations list with the available books list.

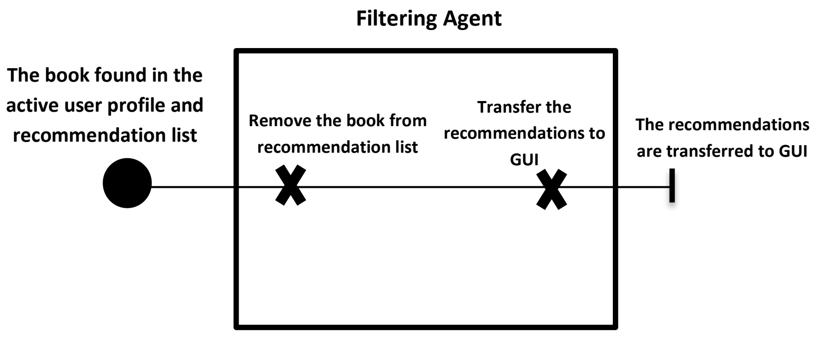

Filtering Agent
The book found in the active user profile and recommendation list
Remove the book from recommendation list
Transfer the recommendations to GUI
The recommendations are transferred to GUI

Figure 5.27: Comparing the active user profile with recommendations list.

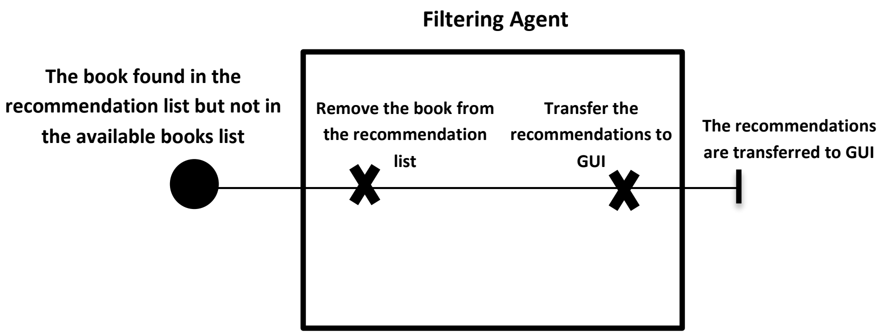

Filtering Agent
The book found in the recommendation list but not in the available books list
Remove the book from the recommendation list
Transfer the recommendations to GUI
The recommendations are transferred to GUI

Figure 5.28: Comparing the recommendations list with available books list.

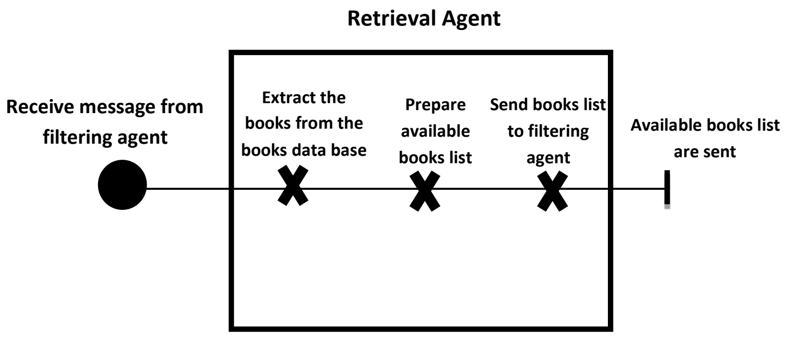

Retrieval Agent
Receive message from filtering agent
Extract the books from the books data base
Prepare available books list
Send books list to filtering agent
Available books list are sent

Figure 5.29: UCM of retrieval agent.

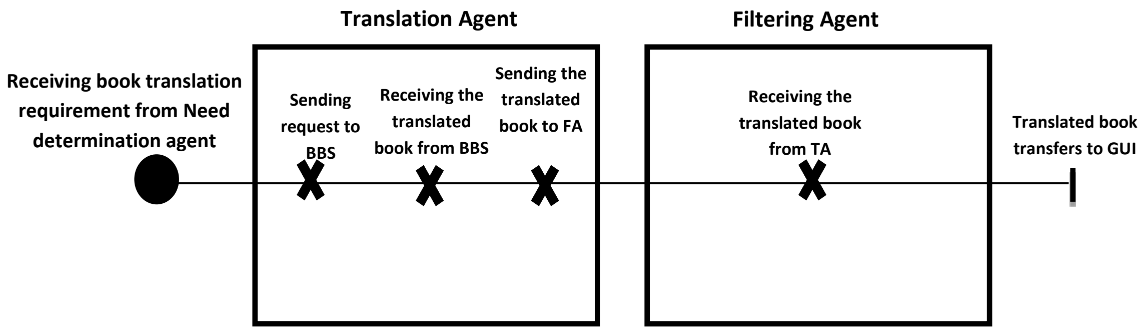

Translation Agent
Filtering Agent
Receiving book translation requirement from Need determination agent
Sending request to BBS
Receiving the translated book from BBS
Sending the translated book to FA
Receiving the translated book from TA
Translated book transfers to GUI

Figure 5.30: The UCM of translating book mechanism.

Step2. If the system requirement specifications (SRS) (Thitisathienkul & Prompoon, 2015) of a previous system do not have a translation function; then, this function is considered as Gold Plating concept; therefore, we should apply the FGA3 which avoid the part of gold plating represented in translation agent (TA) and all components connected from architecture design as illustrated in Figure 5.31.

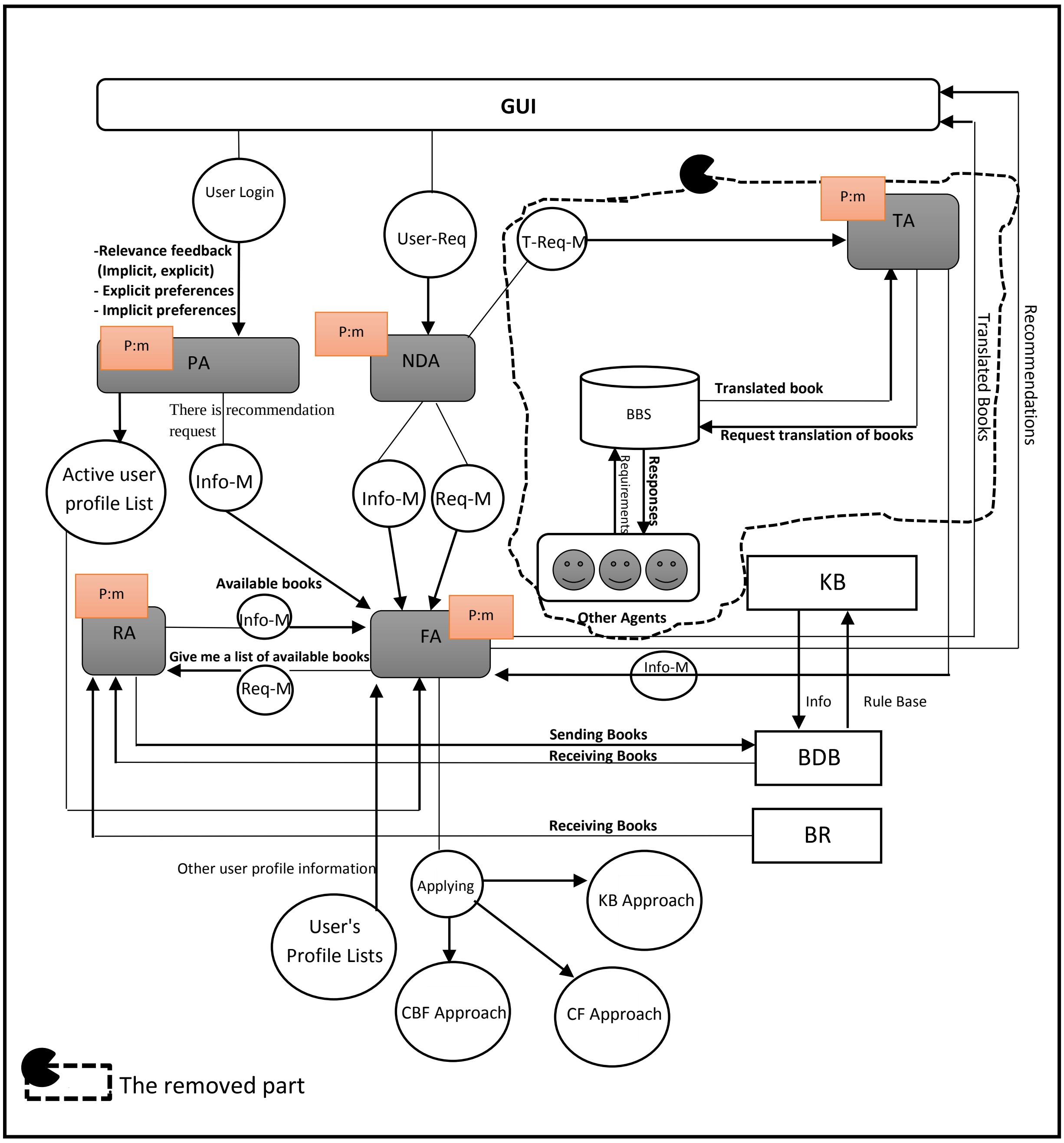


Figure 5.31: Omitting the part representing the gold plating

Depending on FGMOR1 the hierarchical decomposition approach (HDA) could be applied on books recommendation system to demonstrate the main components in visual manner to increase the understandability. Table 5.13 shows the main components and their connected components in books recommendations system.

Main Components	Connected component(1)	Connected component(2)	Connected component(3)
Retrieval Agent	Book Data Base	Filtering Agent	Book Resource
Filtering Agent	Knowledge Base	GUI	Retrieval Agent
Profiling Agent	GUI	-	-
Need determination Agent	GUI	-	-
Book Data Base	Retrieval Agent	-	-
Book Resource	Retrieval Agent	-	-
Knowledge Base	Filtering Agent	-	-
GUI	Profiling Agent	NDA	Filtering Agent

Table 5.13: The main components and their connected components in books recommendations

Figure 5.32, demonstrates the majeure components in case study by applying HDA.

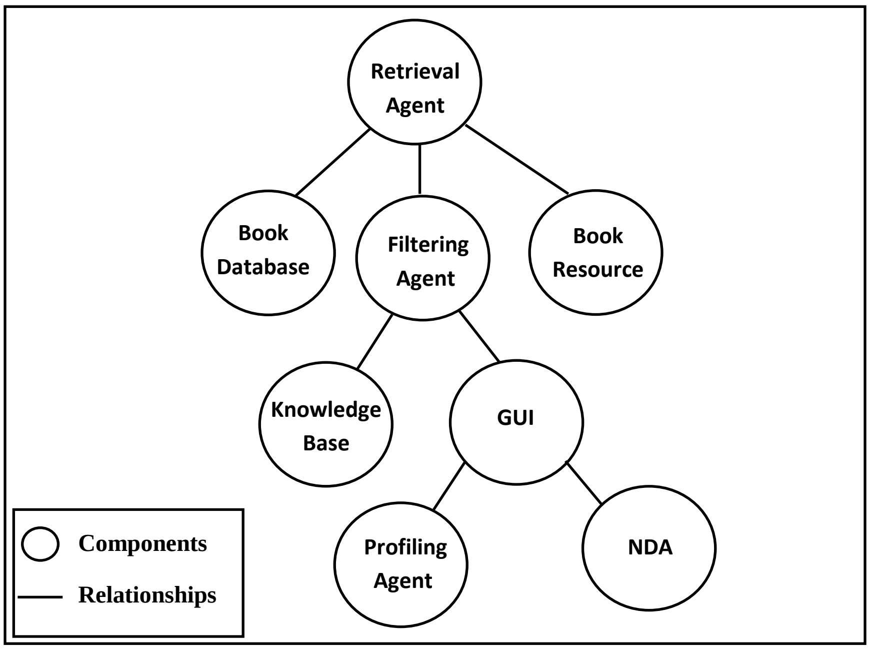


Figure 5.32: Conceptual system after applying **HDA**

Step3. As we have pointed out, the modularity has a major role in decreasing the complexity in software design since the interaction among agents to accomplish their tasks can lead to system complexity. This step totally relies on cohesion measurement principle which uses the Communication Cohesion Measurement (CCM). This measurement works as a testing tool. This enables us to discover which agent needs more decompositions. In this research work, we have four agents described in the case study: Filtering agent, profiling agent, need determination agent, and retrieval agent in respect that the translation agent has been omitted in the last step. Next formulation illustrates the communication cohesive measurement.

$$\mathbf{CCM\ (A_i)} = \frac{\mathbf{R}_{\text{internal}}}{\mathbf{R}_{\text{internal}} + \mathbf{R}_{\text{External}}}$$

Based on the architecture design of book recommendation system, the filtering agent has 4 internal relationships and 2 external relationships, profiling agent has just one internal relationship and 4 external relationships, need determination agent has one internal relationship and 2 external relationships and retrieval agent has 4 internal relationships and 3 external relationships as shown in Table 5.14.

Filtering agent		**Profiling agent**		**Retrieval agent**		**NDA**	
R internal	7	**R** internal	1	**R** internal	3	**R** internal	1
R external	4	**R** external	2	**R** external	2	**R** external	3
CCM(**FA**)	7/11	CCM(**PA**)	1/3	CCM(**RA**)	3/5	CCM(**NDA**)	1/4
Assessment		**Assessment**		**Assessment**		**Assessment**	
CCM(**FA**) = 0.6		CCM(**PA**) = 0.3		CCM(**RA**) = 0.6		CCM(**NDA**) = 0.3	

Table 5.14: The calculating by using CCM technique

- **The results are:**
  - CCM(FA)<0.91
  - CCM(NDA) <0.91
  - CCM(RA) <0.91
  - CCM(PA)<0.91

All results less than 0.91 by this, they do not need more decomposition.

Step4. Applying a group of FG on the architecture design. This group consists of FGABS 1, FGABS 2, FGABS 3 and FGMOL2 which influence the architecture directly and the changes can clearly be observed . Table 5.15 illustrates the symbol of each FG to and also shows which parts of architecture design are influenced after the application in architecture design. Figure 5.33 demonstrates the architectural places affected by applying step4.

N	FG	Description	Place Impact Symbol	Impacts	Notes
1	**FGABS 1**	Applying simplifying abstraction to decrease the dynamic complexity	✖	- Complexity in filtering agent will be reduced. abstracting some details as parameters to support understandability	Transferring from middle level of abstraction to high level.
2	**FGABS 2**	Dividing the architecture design into two levels and avoids adopting the highest level of abstraction to avoid the complexity in architecture design.	■	These two levels allow us to understand the architecture design gradually (basic components then detailed components)	-
3	**FGABS 3**	The Gold plating	◖⬚	Translation agent, blackboard system and all related components will be removed from architecture design.	With supposing it was not from SRS document,
5	**FGMOL2**	Using the simple notations on architecture design.	▲	Simple notations will support the understandability and reduce the complexity.	In this work was proposed and used simple arrows and components

Table 5.15: Clarifying the impact marks on architecture design

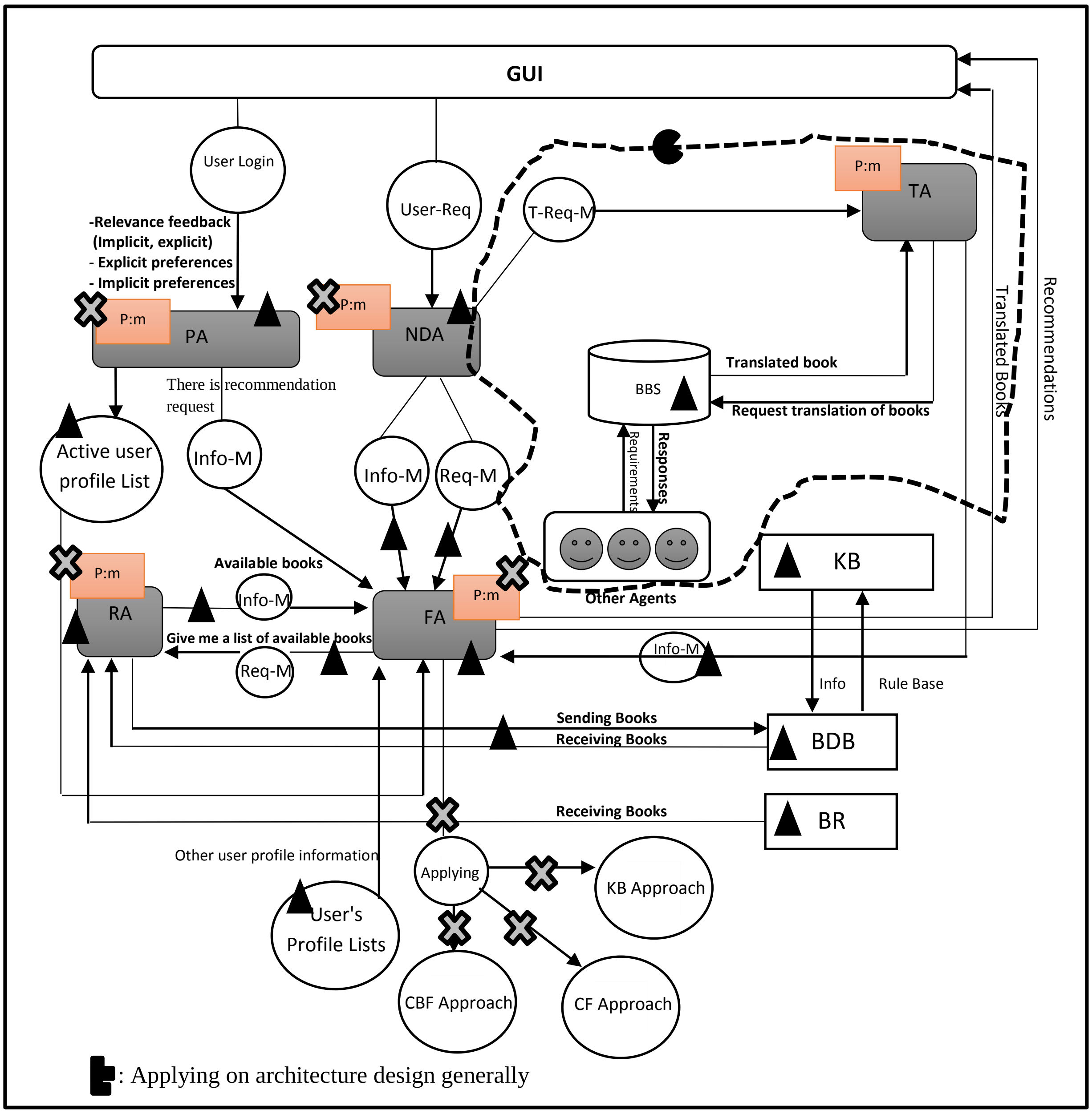


Figure 5.33: Architectural places affected by applying step4

▪ Displaying the Architecture Design

In this section, the architecture design of books recommendations system is viewed after applying the FG4Complexity approach which consists of two graphs. The first graph focuses on FG that addresses the abstraction, notations and avoid the highest level of abstraction. It also represents the first level of abstraction which is indicated in chapter4. The second graph is more detailed than the first one since it comprises more information about the system like showing important data flow, messages, and domains. Figure 5.34 and Figure 5.35 illustrate both graphs.

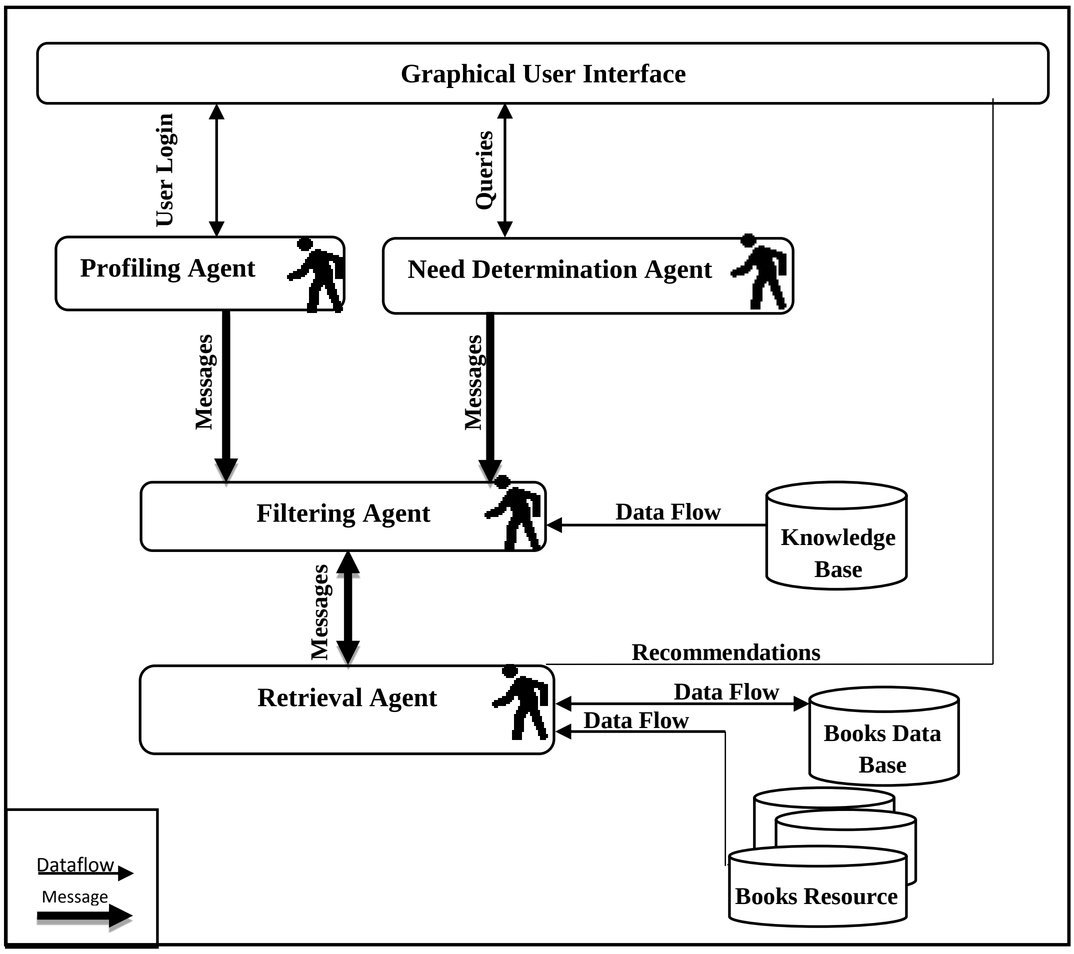


Figure 5.34: First level of Abstraction-Graph1.

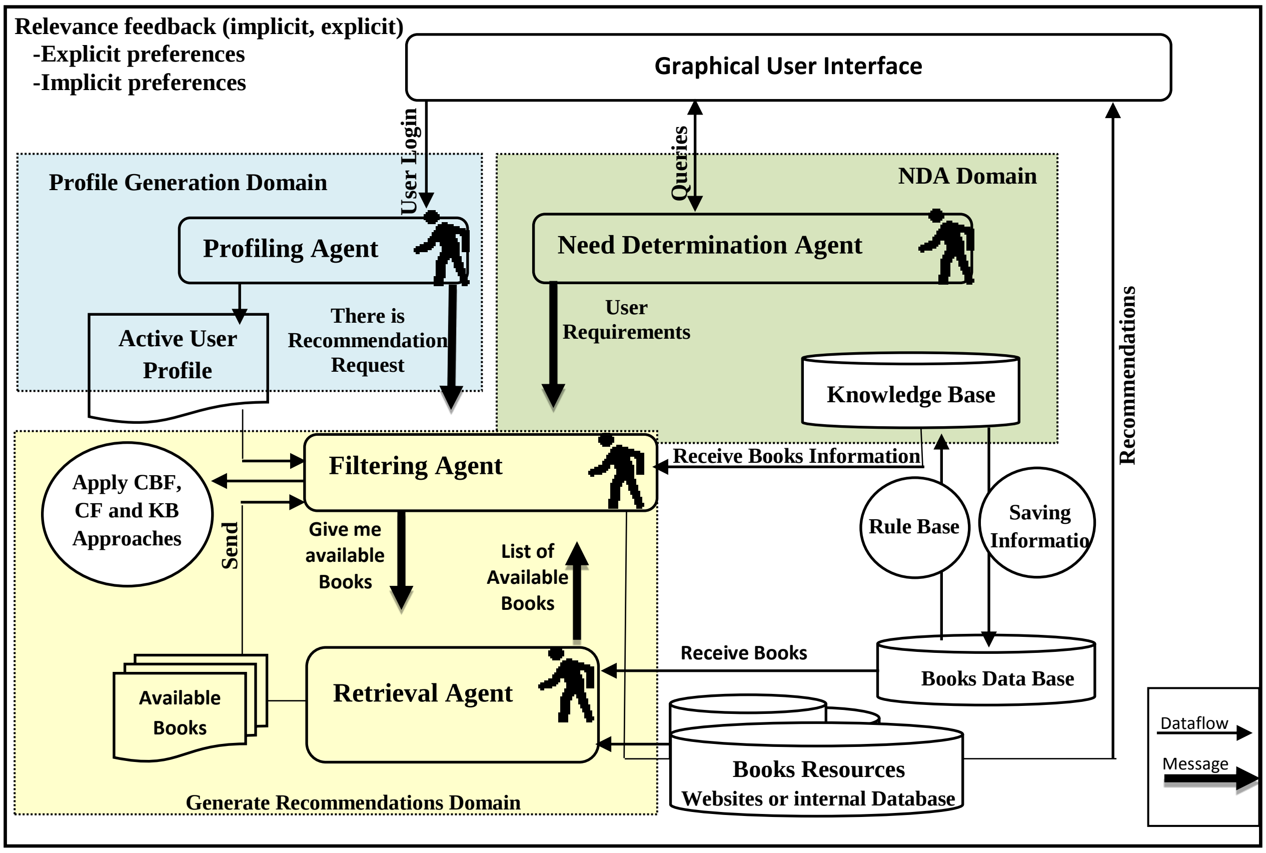


Figure 5.35: Second level of Abstraction-Graph2.

5.8.2. The Measurement

In this section ,we will measure the architecture design from substantial perspective of complexity represented in the tasks assigned to agents and the measurement we labeled as the Complexity Task Measurement (CTM). This is based on Use Case Point technique (UCP) (So Young Moon, 2013a) with adding some modifications to adapt to the agent environment to estimate the following:

- The complexity of the tasks in each agent.
- The complexity of an actors connected with agents.
- The technical complexity factors
- The complexity of environment.
- The complexity of the tasks assigned to all agents.

• The Complexity Task Measurement (CTM)

First of all, the abbreviations and their means used in this chapter will be illustrated. Table 5.16 shows them in full forms and also the added modifications to make the adaption between UCP technique which is based on object oriented environment (OOE) and CTM technique is based on multi-agents systems environment (Kendall, 1997). Besides, Table 5.17 illustrating the adapting between UCP and CTM techniques.

Abbreviations in UCP	**Meaning**	**Abbreviations in CTM**	**Meaning**
UCP	Use Case Point	CTM	Complexity Task Measurement
UUCP	Unadjusted Use Case Point	UT	Unadjusted Task
TCF	Technical Complexity Factor	TCF	Technical Complexity Factor
ECF	Environment Complexity Factor	ECF	Environment Complexity Factor
UUCW	Unadjusted Use Case Weight	UTW	Unadjusted Task Weight
UAW	Unadjusted Actor Weight	UAW	Unadjusted Actor Weight

Table 5.16: The abbreviations and their meaning to techniques

Abbreviations in UCP Measurement	**Formulations in UCP Measurement**	**Abbreviations MAS Environment**	**Formulations in MAS Environment**	**Interpretation**
UCP	UCP = (UUCP) x TCF x ECF	CTM	CTM = (UT) x TCF x ECF	Each use case is changed to a task which assigned to a certain agent.
UUCP	UUCP =UUCW + UAW	UT	UT = UTW + UAW	Calculating unadjusted tasks to an agent
TCF	TCF =0.6+(0.01 *total(TF))	TCF	TCF =0.6+(0.01 *total(TF))	Calculating the technical complexity factor to system
ECF	ECF =1.4+(-0.03 *total(EF))	ECF	ECF =1.4+(-0.03 *total(EF))	Calculating the environment complexity factor to system
UUCW	Estimating the use case as that (Simple, Average, Complex)	UTW	Estimating the tasks as that (Simple, Average, Complex)	Calculating the unadjusted tasks weight of agents
UAW	Estimating the actors as that (Simple, Average, Complex)	UAW	Estimating the agents as that (Simple, Average, Complex)	Calculating the unadjusted agents weight

Table 5.17: The adapting between UCP and CTM

Figure 5.36 shows a brief illustration of the Complexity Task Measurement method.

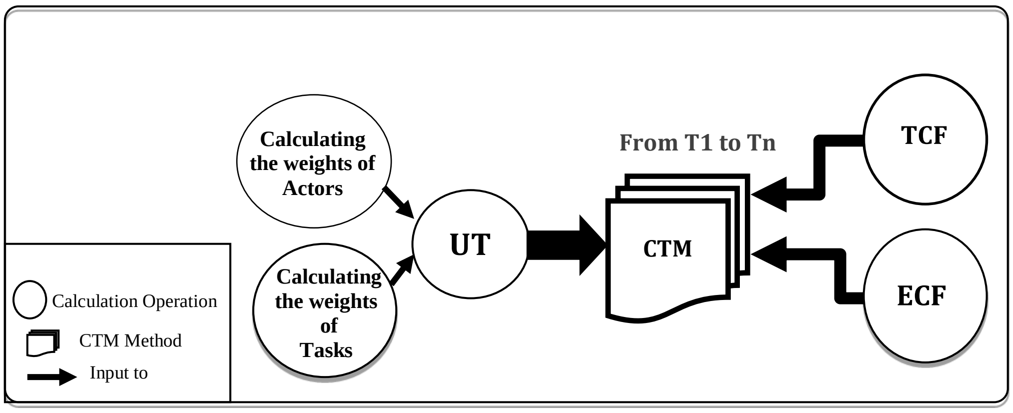
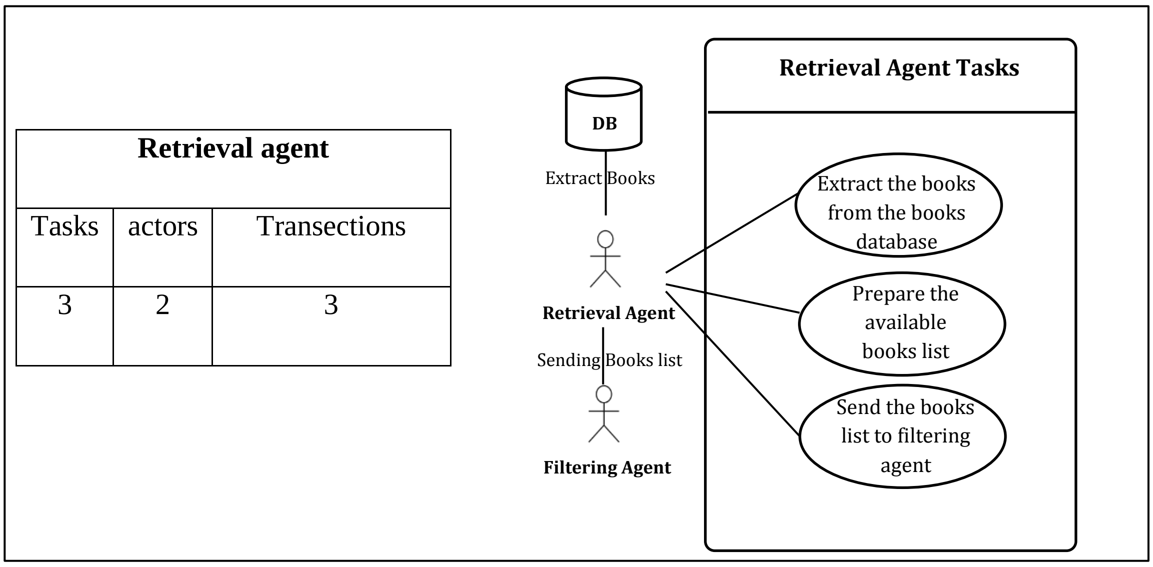

Calculating the weights of Actors
Calculating the weights of Tasks
UT
From T1 to Tn
CTM
TCF
ECF
Calculation Operation
CTM Method
Input to

Figure 5.36: Describing the complexity task measurement (CTM)

▪ Complexity Calculation of the Retrieval Agent Tasks.

Retrieval agent consists of three main tasks, two actors and three transactions as illustrate in Figure 5.37.



Retrieval agent		
Tasks	actors	Transections
3	2	3

DB
Extract Books
Retrieval Agent
Sending Books list
Filtering Agent
Retrieval Agent Tasks
Extract the books from the books database
Prepare the available books list
Send the books list to filtering agent

Figure 5.37: An overview of retrieval agent tasks

Step1. Calculating unadjusted tasks.

Step1.1. Determine the unadjusted task weight as illustrated in Table 5.18 where **NST** is Number of Simple Tasks, **NAT** Number of Average Tasks and **NCT** Number of Complex Tasks.

Task Complexity	**Task Weight**	**Number of Tasks**	**UTW of (RA)**
Simple	5	NST	5 × NST
Average	10	NAT	10 × NAT
Complex	15	NCT	15 × NCT
Unadjusted Task Weight (UTW)			5 × NST + 10 × NAT + 15 × NCT

Table 5.18: Illustrating the Unadjusted Task Weight method (UTW)

Table 5.19 demonstrates the calculating of UTW for retrieval agent.

Task Complexity	**Tasks Weight**	**Number of Tasks**	**UTW of (RA)**
Simple	5	3	5 × 3
Average	10	0	0
Complex	15	0	0
Unadjusted Task Weight (UTW)			5 × 3 + 10 × 0 + 15 × 0=15

Table 5.19: Calculating the UTW of retrieval agent

Step1.2. Determining unadjusted actors weight (UAW) as illustrated in Table 5.20 where **NSA** is Number of Simple Actor, **NAA** Number of Average Actor and **NCA** Number of Complex Actors.

Actor Complexity	**Actor Weight**	**Number of Actors**	**UAW**
Simple	1	NSA	1 × NSA
Average	2	NAA	2 × NAA

Complex	3	NCA	3 × NCA
Unadjusted Actor Weight (UAW)			1 × NSA + 2 × NAA + 3 × NCA

Table 5.20: Illustrating the unadjusted actor weight (UAW) method.

Table 5.21 is an exemplify of calculating the UAW for retrieval agent.

Actor Complexity	Actor Weight	Number of Actors	UAW of (RA)
Simple	1	1	1 × 1
Average	2	0	2 × 0
Complex	3	1	3 × 1
Unadjusted Actor Weight (UAW)			1 × 1 + 2 × 0 + 3 × 1=4

Table 5.21: Calculating the UAW of retrieval agent.

- UT= UTW+UAW.
- UT=15+4.
- UT of Retrieval Agent is (**19**).

- **Step2. Calculating Technical Complexity Factors (TCF).**

To calculate the TCF we should select the factors which affect the system we want to measure. In this case study, some of these factors are determined in Table 5.22.

Factor	Description	Weight
F1	Distributed system	2.0
F2	Response time/performance objectives	1.0
F3	End-user efficiency	1.0
F4	Internal processing complexity	1.0
F5	Code reusability	1.0

F6	Easy to install	0.5
F7	Easy to use	0.5
F8	Portability to other platforms	2.0
F9	System maintenance	1.0
F10	Concurrent/parallel processing	1.0
F11	Security features	1.0
F12	Access for third parties	1.0
F13	End user training	1.0

Table 5.22: The weights of technical complexity factors (TCF)

In this research work, we assume that eight factors are affected by the entire system as following: **F1, F2, F3, F5, F7, F8, F10, and F11**. After selecting the affected factors, their weights must be calculated by the following way.

- **Total Factors** (TFactor) = weights * Rated Values (RV), where RV is from 0 to 5 to each factor.
  = (F1*RV+F2*RV +F3*RV +F5*RV +F7*RV +F8*RV +F10*RV +F11*RV).
  = (5*2.0+3*1.0+2*1.0+4*1.0+5*0.5+3*2.0+4*1.0+3*1.0).
- **TFactor**= (34.5).
- **TCF** = 0.6 + (0.01 × TFactor).
  = 0.6 + (0.01 × 34.5).
- **TCF to all system**= (0.945).

▪ Step3. Calculating the Environmental Complexity Factor (ECF).

To calculate the environmental complexity factor to the entire system we should select the environmental factors that affect the system execution and calculate their weights as shown in Table 5.23.

Factor	Description	Weight
E1	Familiarity with development process used	1.5
E2	Application experience	0.5
E3	Agent-oriented experience of team	1.0

E4	Lead analyst capability	0.5
E5	Motivation of the team	1.0
E6	Stability of requirements	2.0
E7	Part-time staff	-1.0
E8	Difficult programming language	-1.0

Table 5.23: The weights of environmental complexity factors (ECF)

- In books recommender system, we assume that there are six factors influencing the entire system: **E1, E2, E3, E4, E6, and E8.**
- **Total EF**= calculates the weights of (**E1+ E2+ E3+ E4+ E6+E8**).
- **Total EF**= (4.5).
- **EF** = 1.4 + (-0.03 × **Total EF**).
- EF= 1.4 + (-0.03 × 4.5).
- EF= (1.265).
- **ECF of entire system= (1.265).**

▪ **Step4. Calculating the Complexity of Tasks.**

- CTM = UT × TCF × EF.
- CTM=19*0.945*1.265.
- CTM=22.7.
- **Complexity tasks of Retrieval Agent are (22.7).**

▪ Complexity Calculation of Need Determination Agent Tasks.

Need determination agent consists of two main tasks, one actor and two transactions as illustrate in Figure 5.38.

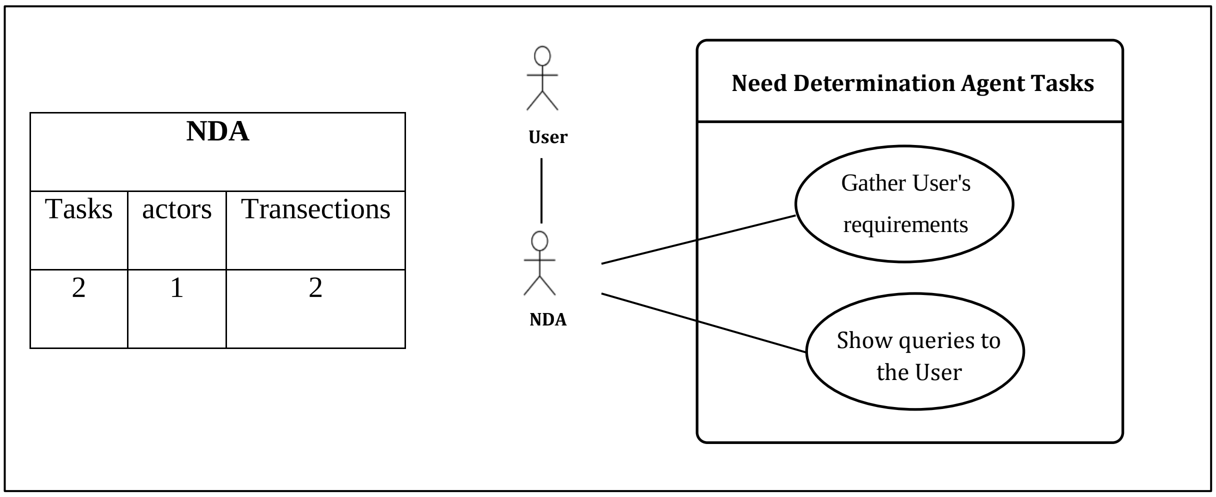


Figure 5.38: An overview of need determination agent tasks

Step1. Calculating the unadjusted tasks.

- **Step1.1**. Determining the Unadjusted Task Weight as Table 5.24:

Task Complexity	Task Weight	Number of Tasks	UTW of (NDA)
Simple	5	2	5 × 2
Average	10	0	0
Complex	15	0	0
Unadjusted Task Weight (UTW)			5 × 2 + 10 × 0 + 15 × 0=10

Table 5.24: The UTW of Need Determination Agent

Step1.2. Determining the Unadjusted Actors Weight (UAW) as Table 5.25.

Actor Complexity	Actor Weight	Number of Actors	UAW of (NDA)
Simple	1	0	1 × 0

Average	2	0	2 × 0
Complex	3	1	3 × 1
Unadjusted Actor Weight (UAW)			1 × 0 + 2 × 0 + 3 × 1=3

Table 5.25: The UAW of need determination agent

- UT= UTW+UAW.
- UT=10+3.
- UT of **NDA** is (**13**).

▪ **Step2. Calculating technical complexity factors.**
  - TCF of NDA is (0.945).

▪ **Step3. Calculating Environmental Complexity Factor.**
  - ECF of NDA is (1.265).

▪ **Step4. Calculating Complexity Tasks.**
  - CTM = UT × TCF × EF.
  - CTM=13*0.945*1.265.
  - CTM=15.5
  - **Complexity tasks of NDA (15.5).**

▪ Complexity Calculation of Profiling Agent Tasks

Profiling agent consists of eight main tasks, one actor and eight transactions as illustrate in Figure 5.39.

Profiling Agent		
Tasks	actors	Transections
8	1	8

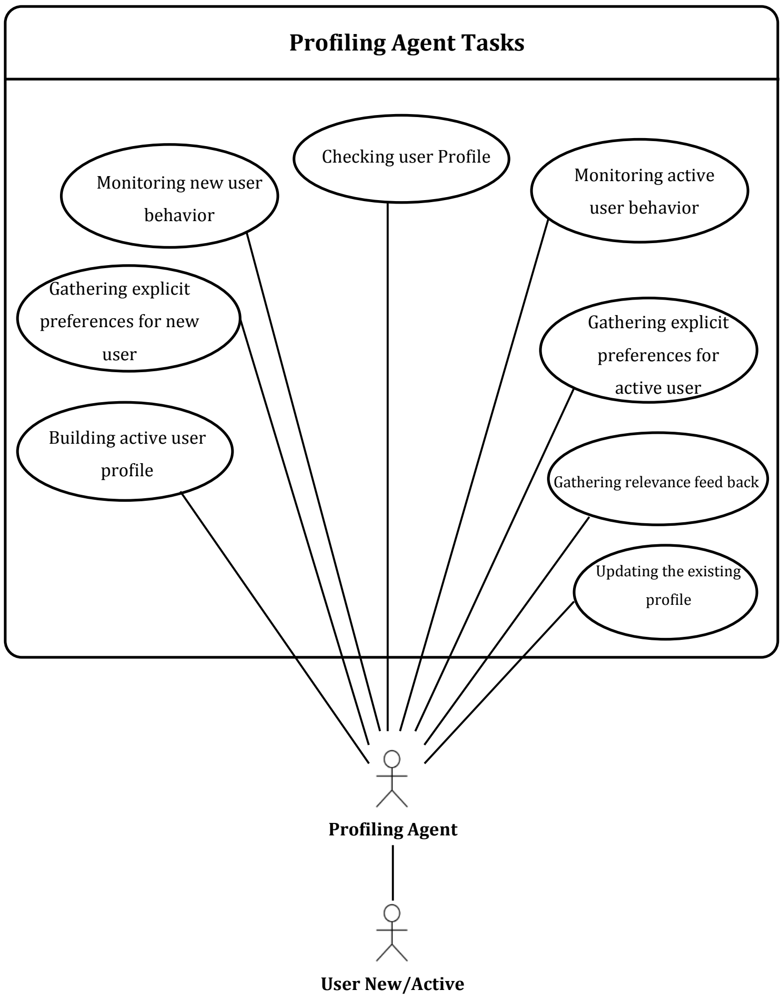

Profiling Agent Tasks
Checking user Profile
Monitoring new user behavior
Monitoring active user behavior
Gathering explicit preferences for new user
Gathering explicit preferences for active user
Building active user profile
Gathering relevance feed back
Updating the existing profile
Profiling Agent
User New/Active

Figure 5.39: An overview of profiling agent tasks

- **Step1**. Calculating the unadjusted tasks.
    - **Step1.1**. Determining the Unadjusted Task Weight as Table 5.26.

Task Complexity	**Task Weight**	**Number of Tasks**	**UTW of (PA)**
Simple	5	0	0
Average	10	0	0
Complex	15	8	15*8
Unadjusted Task Weight (UTW)			5 × 0 + 10 × 0 + 15 × 8=120

Table 5.26: The unadjusted tasks weight of profiling agent

Step1.2. Determines Unadjusted Actor Weight (UAW) as Table 5.27.

Actor Complexity	Actor Weight	Number of Actors	UAW of (PA)
Simple	1	0	1 × 0
Average	2	0	2 × 0
Complex	3	1	3 × 1
Unadjusted Actor Weight (UAW)			1 × 0 + 2 × 0 + 3 × 1=3

Table 5.27: The UAW of profiling agent

- UT= UTW+UAW.
- UT=120+3.
- UT of Profiling Agent is (**123**).

▪ **Step2. Calculating the technical complexity factors.**
  - TCF of Profiling Agent is (0.945).

▪ **Step3. Calculating the Environmental Complexity Factor.**
  - ECF of Profiling Agent is (1.265).

▪ **Step4. Calculating the Complexity Tasks.**
  - CTM = UT × TCF × EF.
  - CTM=123*0.945*1.265.
  - CTM=147.03.
  - **Complexity tasks of Profiling Agent are (147.03).**

▪ Complexity Calculation of Filtering Agent Tasks

Filtering agent consists of eleven main tasks, four actors and eleven transactions as illustrate in Figure 5.40.

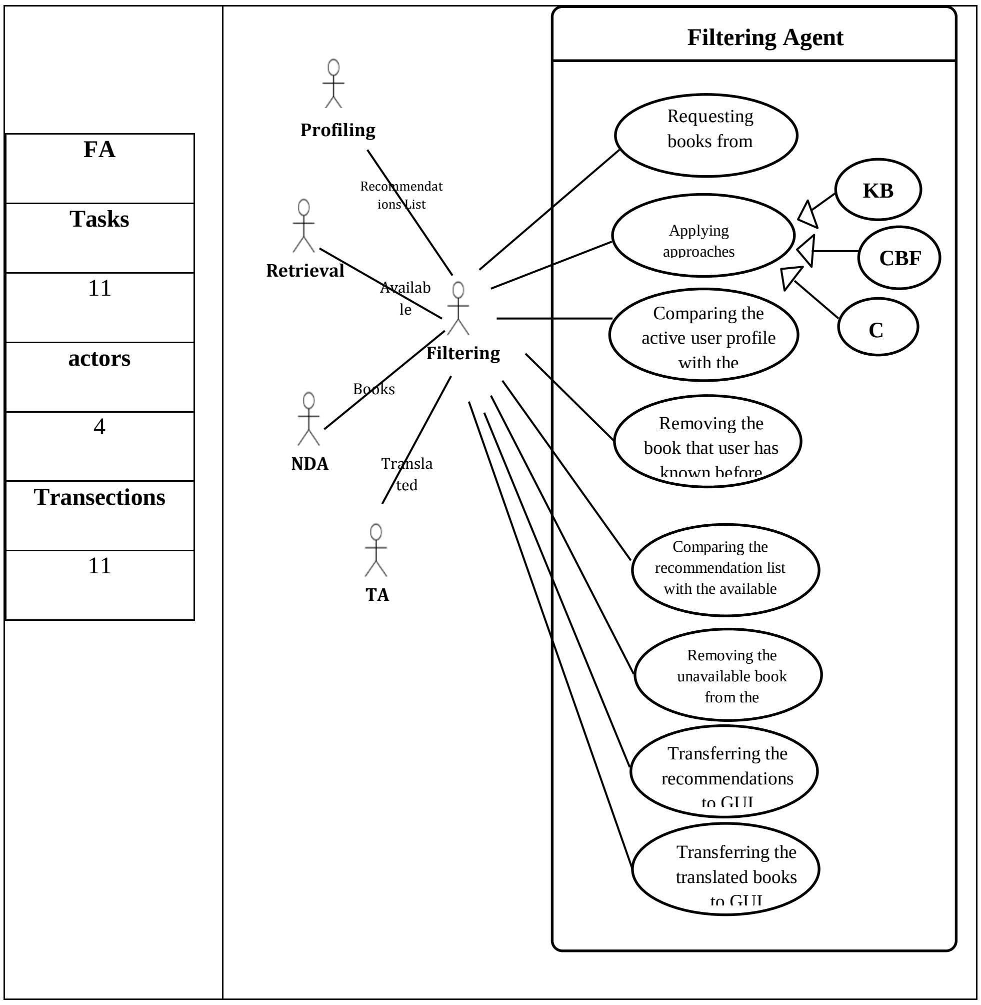

FA
Tasks
11
actors
4
Transections
11
Profiling
Recommendations List
Retrieval
Available
Filtering
Books
NDA
Translated
TA
Filtering Agent
Requesting books from
Applying approaches
Comparing the active user profile with the
Removing the book that user has known before
Comparing the recommendation list with the available
Removing the unavailable book from the
Transferring the recommendations to GUI
Transferring the translated books to GUI
KB
CBF
C

Figure 5.40: An overview of filtering agent tasks

- **Step1**. Calculating unadjusted tasks.
    - **Step1.1**. Determining Unadjusted Task Weight as Table 5.28:

Task Complexity	Task Weight	Number of Tasks	UTW of (FA)
Simple	5	0	0*5
Average	10	11	11*10
Complex	15	0	0*15
Unadjusted Task Weight (UTW)			5 × 0 + 10 × 11 + 15 × 0=110

Table 5.28: The UTW of filtering agent

Step1.2. Determining Unadjusted Actors Weight (UAW) as Table 5.29.

Actor Complexity	Actor Weight	Number of Actors	UAW of (FA)
Simple	1	0	1×0
Average	2	4	2×4
Complex	3	0	3×0
Unadjusted Actor Weight (UAW)			$1 \times 0 + 2 \times 4 + 3 \times 0=8$

Table 5.29: The UAW of filtering agent

- UT= UTW+UAW.
- UT=110+8.
- UT of Filtering Agent is (**118**).

▪ **Step2. Calculating the technical complexity factors.**
  - TCF of Filtering Agent is (0.945).

▪ **Step3. Calculating the Environmental Complexity Factor.**
  - ECF of Filtering Agent is (1.265).

▪ **Step4. Calculating the Complexity Tasks.**
  - CTM = UT × TCF × EF.
  - CTM=118*0.945*1.265.
  - CTM=141.5.
  - **Complexity tasks of Filtering Agent are (141.5).**

▪ Complexity Calculation of Translation Agent Tasks.

Translation agent consists of four main tasks, three actors and four transactions as illustrate in Figure 5.41.

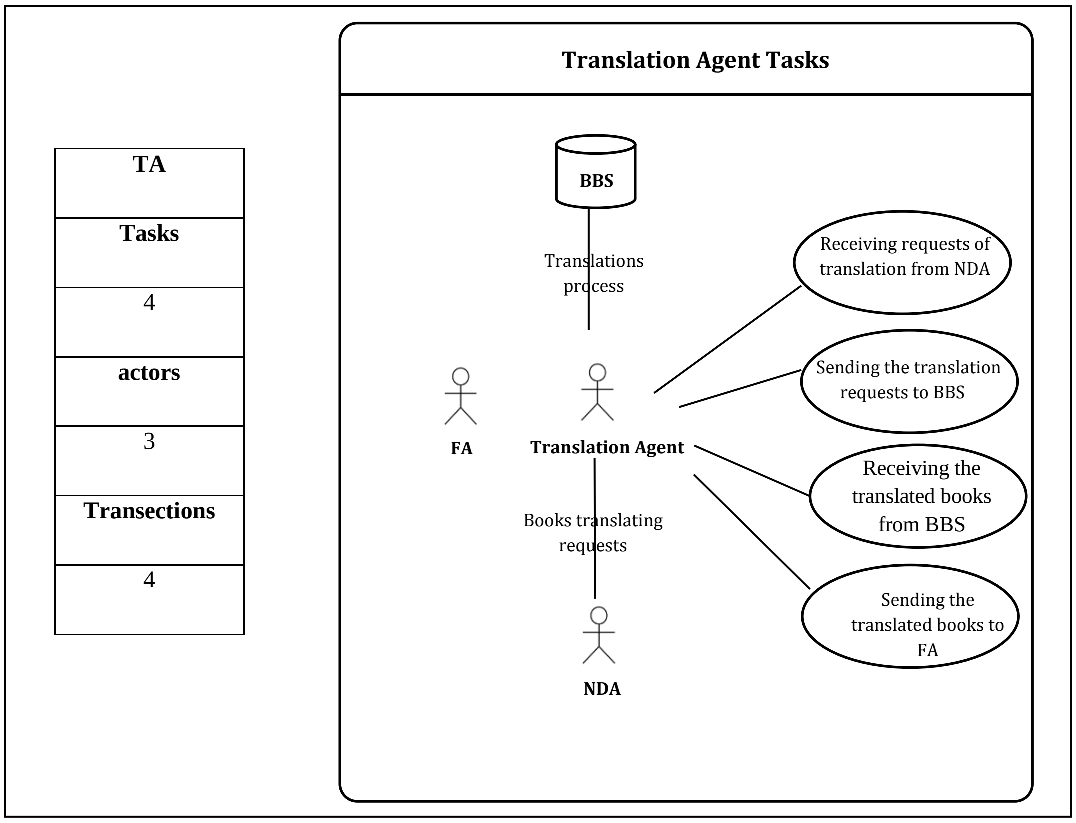

TA
Tasks
4
actors
3
Transections
4
Translation Agent Tasks
BBS
Translations process
FA
Translation Agent
Books translating requests
NDA
Receiving requests of translation from NDA
Sending the translation requests to BBS
Receiving the translated books from BBS
Sending the translated books to FA

Figure 5.41: An overview of translation agent tasks

Step1. Calculating the unadjusted tasks.

- **Step1.1**. Determining the Unadjusted Task Weight as Table 5.30.

Task Complexity	Task Weight	Number of Tasks	UTW of (TA)
Simple	5	0	0*5
Average	10	04	4*10
Complex	15	0	0*15
Unadjusted Task Weight (UTW)			5 × 0 + 10 × 4 + 15 × 0=40

Table 5.30: The UTW of translation agent

Step1.2. Determining Unadjusted Actor Weight (UAW) as Table 5.31.

Actor Complexity	Actor Weight	Number of Actors	UAW of (TA)
Simple	1	0	1 × 0
Average	2	2	2 × 1
Complex	3	1	1 × 1
Unadjusted Actor Weight (UAW)			1 × 0 + 2 × 2+ 3 × 1=7

Table 5.31: The UAW of translation agent.

- UT= UTW+UAW.
- UT=40+7.
- UT of Translation Agent is (**47**).

- **Step2. Calculating the technical complexity factors.**
  - TCF of Translation Agent is (0.945).
- **Step3. Calculating the Environmental Complexity Factor.**
  - ECF of Translation Agent is (1.265).
- **Step4. Calculating the Complexity Tasks.**
  - CTM = UT × TCF × EF.
  - CTM=47*0.945*1.265.
  - CTM=56.1.
  - **Complexity tasks of Translation Agent (56.1).**

Calculating the CTM for Overall System.

- In this section, the complexity of multi-agents systems tasks is calculated by summation of CTM to each agent.

- In original architecture design (before applying FG4Complexity approach) there are five agents, so, we can calculate the total CTM as following:

- $\text{Total CTM} = \sum^{\text{CTM}}_{\text{Profiling, Filtering, Need determination, Retrieval, and Translation Agents}}$

- Total(CTM) =(174.03)+(141.5)+(15.5)+(22.7)+(56.1)

Total (CTM) of original AD= (409.8).

- **To measure the architecture design after applying FG4 Complexity we should remove the translation agent as shown below.**

$\text{Total CTM} = \sum^{\text{CTM}}_{\text{Profiling, Filtering, Need determination, and Retrieval Agents}}$

Total (CTM) = (174.03) + (141.5) + (15.5) + (22.7).

Total (CTM) of AD after applying the FG4Complexity approach =(353.7).

- **Comparison by Results and the Assessment**
    - **The Comparison:**

In this section of research work, results of architectural designs are compared after have been measured by Complexity Tasks Measurement as Table 5.32 shows.

The Architecture design Assessment	
Original AD	**After applying FG4Complexityapproach**
CTM	**CTM**
409.8	**353.7**

Table 5.32: The assessment to CTM before and after applying the FG4Complexity approach

Figure 5.42 illustrates the results of CTM.

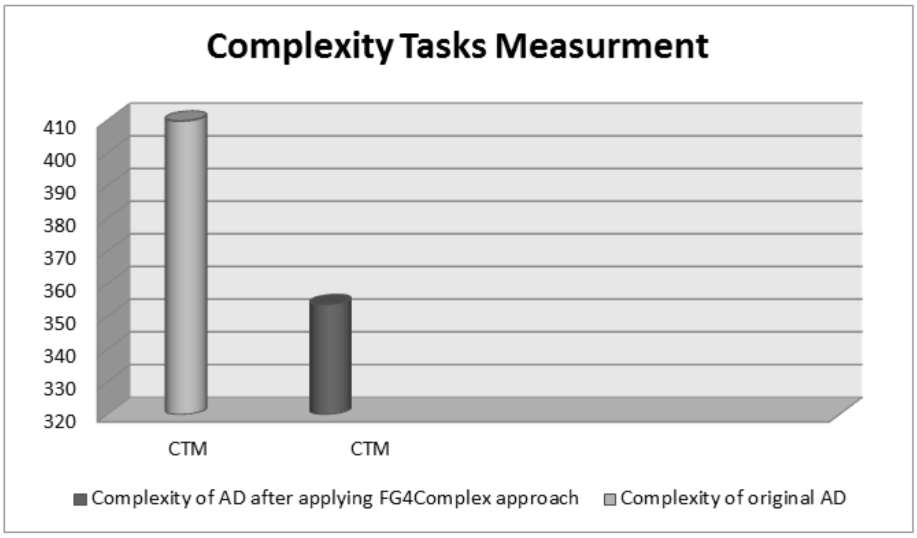


Figure 5.42: The results of (CTM) measurment

- **The Assessment**
  - **In Terms of Analysis:**

- Using use case maps improves the understandability of agent tasks (Agent responsibilities) and the interactions which occur in the system, and this enables us to avoid the complexity in the early stages of architecture design.
- Decreasing the complexity by clarifying the basic modularity of software system depending on (major components and their relationships).

  - **In Terms of Architecture Design Form:**

- Classifying the AD into two forms. Basically, this classification gradually allows us to understand the essential components and their relationships. As a result, more comprehended information will be displayed in the second architecture design form.
- Using clearing notations plays a major role in recognizing the system components and their communications in a natural way.
- Avoiding the additional function that was not required in a system such as illustrated in FGABS3 also contributes to the decrease of the complexity.

Figure 5.43 shows the architectural design before and after applied FG4Comlexity approach.

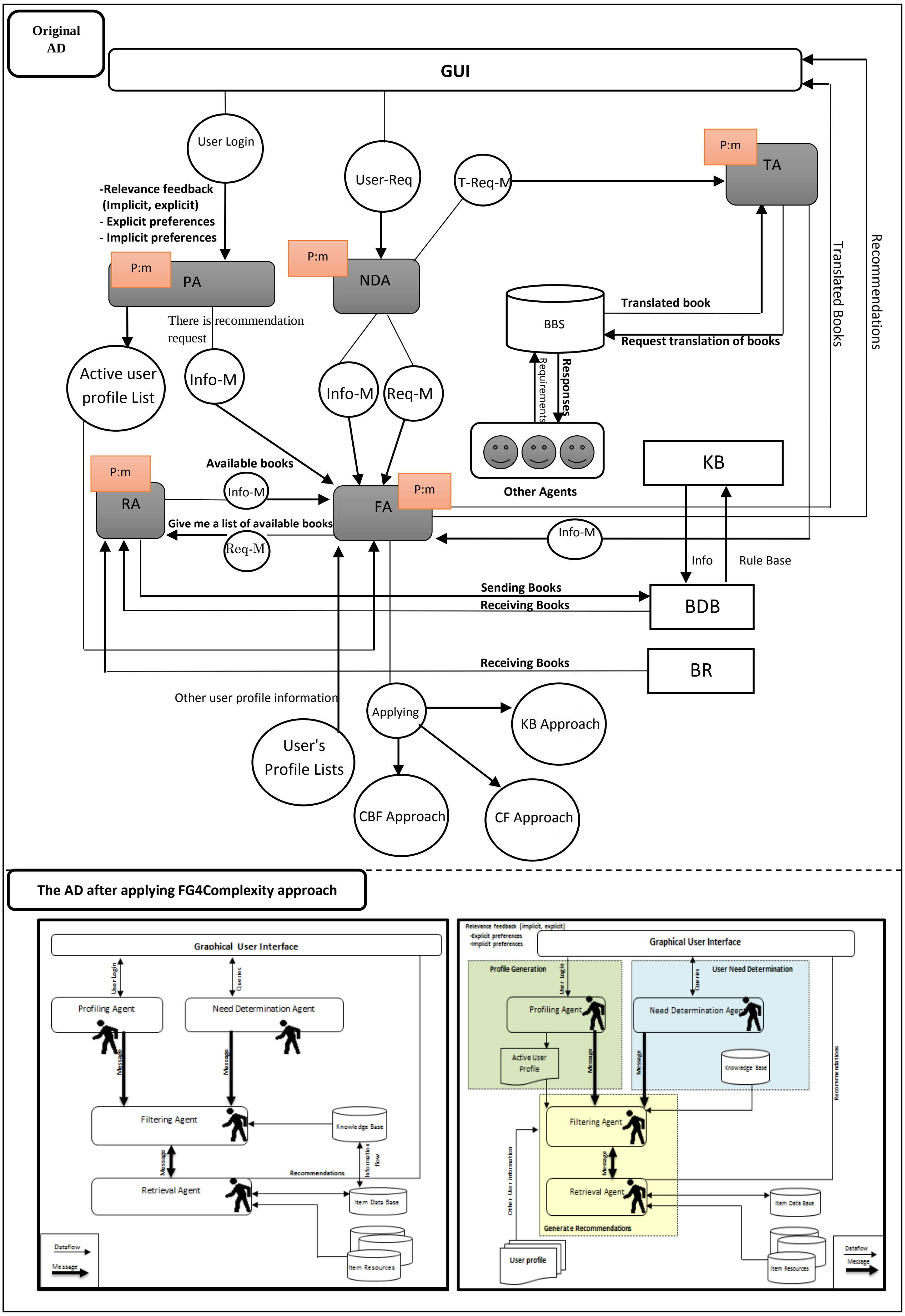


Figure 5.43: Displaying the AD before and after applying the FG4Complexity approach

5.9. Summary

This chapter introduced the case study: Books Recommendations System, which is based on multi agents systems. Hence, it applied all factors and guidelines proposed in FG4Complexity approach on the case study. In addition, it introduced the CTM method to measure the system before and after applying the proposed approach. And finally it illustrated the assessment of FG4complexity of architecture design in terms of analysis and form.

CHAPTER 6
Conclusion and Future Work

6.1. The Conclusion

This thesis describes a method to reduce the complexity of architecture design to multi-agents systems. This method includes set of guidelines related to abstraction, modularity and modeling concepts. It labels as "FG4complexity" and discusses the decrease of complexity which usually occurs during the interactions among agents. In other side, the research developes a method entitled (CTM) to estimate the complexity of multi-agents systems, based on use case point method by adding some modifications on this method to adapt it with agent environment. This modification aims at estimating the complexity of the tasks in each agent, the complexity of every actor connected with agents, the technical complexity factors, the complexity of environment and the complexity of the tasks assigned to all agents. The FG4complexity approach is useful for large systems such as recommendation systems that are based on multi-agents systems to avoid the complexity problems found in the most existing architectures. Thus, it enhances the quality standards, the reduction of complexity from architecture design, and eventually reinforces the reusability concept.

6.2. The Future Work

For future work, other aspects of architecture design could be addressed to attempt to make the proposed approach more effective. Those aspects may be represented in the style, design patterns, documentation and etc. Correspondingly, the research will introduce more effective means including guidelines for complexity measurements. In other words, assisting developers of systems based multi –agents systems to select suitable measurements for their systems. Measuring the complexity of architecture design of systems through other measuring techniques, e.g., to measure the coupling among agents. Finally, we hope to apply the FG4complexity approach on other larger and more complex systems.

REFERENCES

Abdelaziz, T., Elammari, M., Unland, R., & Branki, C. (2010). MASD: Multi-Agent Systems Development Methodology. *Multiagent and Grid Systems Journal, 6*(1), 71-101 .

Ahmed Taki, Z. S. (2014). *Formal Specification of Multi-Agent System Architecture*. Paper presented at the International Conference on Advanced Aspects of Software Engineering ICAASE, ALGERIA .

Ahmed.A, I. B. (2013). *From UML 2.0 Interaction Fragments to PROMELA using a Graph Transformation Approach*. Paper presented at the The International Arab Conference on Information Technology (ACIT'2013), Tunisia .

Akerkar, R., & Sajja, P. (2010). *Knowledge-based systems*: Jones & Bartlett Publishers.

Al-Masree, H. K. (2015). Extracting Entity Relationship Diagram (ERD) from relational database schema. *International Journal of Database Theory and Application, 8*(3), 15-26 .

Alenezi, M., & Almustafa, K. (2015). Empirical analysis of the complexity evolution in open-source software systems. *International Journal of Hybrid Information Technology, 8*(2), 257-266 .

Alenezi, M., & Zarour, M. (2015). *Modularity measurement and evolution in object-oriented open-source projects.* Paper presented at the Proceedings of the The International Conference on Engineering & MIS 2015.

Aleryani, A. Y. (2016). Comparative Study between Data Flow Diagram and Use Case Diagram. *International Journal of Scientific and Research Publications, 6*(3), 124-127 .

Alessandro Garcia, U. K., Carlos Lucena. (2008). On the modularity assessment of aspect-oriented multiagent architectures: a quantitative study. *International Journal of Agent-Oriented Software Engineering, 2* .

Alhumaidan, F. (2012). A critical analysis and treatment of important UML diagrams enhancing modeling power. *Intelligent Information Management, 4*(05), 231 .

Anirban Sarkar ,N. C. D. (2012). Measuring Complexity of Multi-Agent System Architecture. *IEEE* .

Bartz, R. (2013). Mapping data models of the standardized automotive testing data storage to the unified modeling language *Industrial Technology (ICIT), 2013 IEEE International Conference on* (pp. 1327-1332). South Africa: IEEE.

Bhardwaj, M. S. (2015). *A Framework for Designing Modeling and Analysis Agent Based Software Systems*. Paper presented at the 4th International Conference on System Modeling & Advancement in Research Trends (SMART), Beijing, China .

Bouwers, E. a. V., Joost and Lilienthal, Carola and van Deursen, Arie. (2010). A Cognitive Model for Software Architecture Complexity. *IEEE,* 152-155 .

Broersen, J., Dastani, M., & van der Torre, L. (2005). Beliefs, obligations, intentions, and desires as components in an agent architecture. *International Journal of Intelligent Systems, 20*(9), 893-919 .

Burke, R. (2002). Hybrid Recommender Systems:Survey and Experiments. *User Modeling and User-Adapted Interaction, 12*(1), 33 .370 - 1

Castillo, H. (2007). *Hybrid Content-Based Collaborative-Filtering Music Recommendations.* MSc., University of Twente, Netherlands .

Chakraborty, A., Baowaly, M. K., Arefin, A., & Bahar, A. N. (2012). The role of requirement engineering in software development life cycle. *Journal of emerging trends in computing and information sciences, 3*(5), 723-729 .

Chaptini, B. H. (2005). *Use of discrete choice models with recommender systems.* PHD., Massachusetts Institute of Technology, Cambridge. USA .

Chin, K. O., Gan, K. S., Alfred, R., Anthony, P., & Lukose, D. (2014). Agent Architecture: An Overviews. *Transactions on science and technology, 1*(1), 18-35 .

Chris F. Kemerer, M. C. P. (2009). The Impact of Design and Code Reviews on Software Quality. *IEEE, VOL. 35*(4 .(

Cohen, T. T. a. R. (2000). Hybrid Recommender Systems for Electronic Commerce. *in the 17th National Conference on Artificial Intelligence AAAI* .

Darcy, D. P., Daniel, S. L., & Stewart, K. J. (2010). *Exploring Complexity in Open Source Software: Evolutionary Patterns, Antecedents, and Outcomes.* Paper presented at the hicss.

De Bruin, H. a. v. V., Hans. (2003). Quality-driven software architecture composition. *Journal of Systems and Software, Elsevier, 66*(3), 269--284 .

Elammari, M., & Lalonde, W. (1999). *An agent-oriented methodology: High-level and intermediate models.* Paper presented at the Proc. of the 1st Int. Workshop. on Agent-Oriented Information Systems.

Engin, G., Aksoyer, B., Avdagic, M., Bozanlı, D., Hanay, U., Maden, D., & Ertek ,G. (2014). Rule-based expert systems for supporting university students. *Procedia Computer Science, 31,* 22-31 .

Evesti, A. (2007). *Quality-oriented software architecture development*. Paper presented at the International Conference on Information Technology: Coding and Computing (ITCC'05), Las Vegas, NV, USA .

Far, B. H. (2002). Software Agents: Quality, Complexity and Uncertainty Issues. *IEEE*. doi: 10.1109/COGINF.2002.1039290

Francisca Losavio and Ledis Chirinos, N. L. a. A. R.-C. (2003a). Quality Characteristics for Software Architecture. *JOURNAL OF OBJECT TECHNOLOGY, 2* .

Francisca Losavio and Ledis Chirinos, N. L. a. A. R.-C. (2003b). Quality Characteristics for Software Architecture*. *ETH Zurich, Chair of Software Engineering, 2* .

frerjani, R. N. A .A. E. (2010). *Towards A General Architecture for Building Intelligent, Flexible, and Adaptable Recommender System Based on MAS Technology.* post graduation, benghazi IEEE journal .

Ghasemi, M., Sharafi, S. M., & Arman, A. (2015). Towards an Analytical Approach to Measure Modularity in Software Architecture Design. *JSW, 10*(4), 465-479 .

Ghazal Keshavarz, D. N. M., Dr. Mirmohsen Pedram (2011). Metric for Early Measurement of Software Complexity. *International Journal on Computer Science and Engineering (IJCSE) 3* .

Goel, A. K., Rugaber, S., & Vattam, S. (2011). Structure, behavior, and function of complex systems: The structure, behavior, and function modeling language. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 23*(01), 23-3 .5

Haber, A. a. R., Holger and Rumpe, Bernhard and Schaefer, Ina. (2011). *Delta Modeling for Software Architectures*. Paper presented at the Tagungsband des Dagstuhl-Workshop MBEES .

Hanseth, O. a. L., Kall. (2010). Design theory for dynamic complexity in information infrastructures: the case of building internet. *Springer, 15,* 1-19 .

Isinkaye, F., Folajimi, Y., & Ojokoh, B. (2015). Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal, 16*(3), 261-273 .

ISO. (2016). ISO/IEC 25023 *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality*.

Itmazi, J. (2005). *Flexible Learning Management System To Support Learning In The Traditional And Open Universities* .

Iván García-Magariño, M. C., Valeria Seidita. (2010). A metrics suite for evaluating agent-oriented architectures. *ACM* .

Jomi Fred Hubner, J. S. S., and Olivier Boissier2. (2007). *Prioritizing Quality Specifications of Multi-agent Systems*. Paper presented at the IAENG (International Association of Engineers), London (UK .(

Karageorgos, A. a. M., Nikolay and Thompson, Simon. (2003). A Design Complexity Evaluation Framework for

Agent-Based System Engineering Methods. *Springer* .

Keating, M. (2000). measuring design quality by measuring design complexity. *IEEE Computer Society Washington*, 314 .

Kendall, E. A. a. M., Margaret T. and Jiang, Chong H. (1997). THE APPLICATION OF OBJECT-ORIENTED ANALYSIS TO AGENT BASED SYSTEMS. *JOOP* .62-56 ,(1)9 ,

Khan, Y. A., & Mahmood, S. (2012). Deriving Sequence Diagrams from Use Case Map Scenario Specifications .

Kirandeep Kaur, B. J., Rekha Rani. (2013). Analysis of Gold Plating: A Software Development Risk. *International Journal of Computer Science and Communication Engineering, 2*(1 .(

Kl¨ugl, F. (2008). Measuring Complexity of Multi-agent Simulations – An Attempt Using Metrics. *Springer,* 128-138 .

Kruchten, P. (1995). Architectural Blueprints—The “4+1” View Model of Software Architecture. *IEEE*

Lawgali, A. (2017). TRACEABILITY OF UNIFIED MODELING LANGUAGE DIAGRAMS FROM USE CASE MAPS. *International Journal of Software Engineering & Applications (IJSEA* .(

Lenhart, P., & Herzog, D. (2016). Combining Content-based and Collaborative Filtering for Personalized Sports News Recommendations. *CBRecSys 2016*, 3 .

Leopold, H., Mendling, J., Reijers, H. A., & La Rosa, M. (2014). Simplifying process model abstraction: Techniques for generating model names. *Information Systems, 39*, 134-151. Retrieved from

Luiz, J. (2009). The Many Faces of Complexity in Software

Design.

M. Montaner, B. L., and J. De La. (2003). A Taxonomy of Recommender Agents on the Internet. *Artificial Intelligence Review, 19*(4), 285–330 .

Malika Addou, S. M. (2011). A new approach of designing Multi-Agent SystemsInternational Journal of Advanced Computer Science and Applications, (Vol. 2): (IJACSA) International Journal of Advanced Computer Science and Applications .

Markic, I. a. S., Maja and Maras, Josip. (2014). Intelligent Multi Agent Systems for Decision Support in Insurance Industry *Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 1118--1123): IEEE.

Mary Shaw, R. D., Daniel V. Klein, Theodore L. Ross, David M. Young, Gregory Zelesnik. (1995 .(Abstractions for Software Architecture and Tools to Support Them. *IEEE Transactions on software engineering, 21*(4), 314--335. doi: 10.1109/32.385970

Medeiro, F., Pérez-Verdú, B., & Rodríguez-Vázquez, A. (2013). *Top-down design of high-performance sigma-delta modulators* (Vol. 480): Springer Science & Business Media.

Meli, R., & Santillo, L. (1999). *Function point estimation methods: A comparative overview.* Paper presented at the Proceedings of FESMA.

Mirakhorli, M. a. C., Hong-Mei and Kazman, Rick. (2015 .(Mining Big Data for Detecting, Extracting and Recommending Architectural Design Concepts *Proceedings of the First International Workshop on BIG Data Software Engineering* (pp. 15--18): IEEE press.

Misra, S. (2011). An approach for the empirical validation of software complexity measures. *Acta Polytechnica Hungarica, 8*(2), 141-160 .

Moertini, V. S., Heriyanto, S., & Nugroho, C. D. (2014). REQUIREMENT ANALYSIS METHOD OF E-COMMERCE WEBSITES DEVELOPMENT FOR SMALL-MEDIUM ENTERPRISES, CASE STUDY: INDONESIA. *International Journal of Software Engineering & Applications, 5*(2), 11 .

Mohamed, N. a. S., Raja Fitriyah Raja and Endut, Wan Rohana Wan. (2013). The Use of Cyclomatic Complexity Metrics in Programming Performance's Assessment. *Procedia-Social and Behavioral Sciences, 90,* 497--503 .

Muli, E. a. K., James. (2015). A multi-agent based model for self motivated learners: self study tool. *International Journal of Advanced Computer Research, 5,* 298 .

Obando, J. (2008). *Methodology to obtain the user's Human Values Scale from Smart User*

Models .

Oprea, M. (2004). Applications of multi-agent systems. In R. R. (Ed.), *Information Technology* (Vol. 157, pp. 239-270). Boston, MA: Springer.

Pang, G. K.-H. (2000). BLACKBOARD ARCHITECTURE FOR INTELLIGENT CONTROL .

Pohl, K .(2010) .*Requirements engineering: fundamentals, principles, and techniques*: Springer Publishing Company, Incorporated.

Ross, D. T., Goodenough, J. B., & Irvine, C. (1975). Software engineering: process, principles, and goals. *Computer, 8*(5), 17-27 .

Rudenko, D., & Borisov, A. (2007). *An overview of blackboard architecture application for real tasks.* Paper presented at the Scientific Proceedings Of Riga Technical University, Ser.

S.Mary Helan Felista1, S. S., S. Jebapriya3. (2014). A Genteel Requirement Engineering for Web

Applications. *International Journal of Innovative Research in Computerand Communication Engineering, 2*(5 .(Saleh, E. M. (2014). *Architecture for Design Pattern Selection*

based on Multi-Agent System. post graduation, benghazi university .

Sara Maalal, M. A. (2011). A new approach of designing Multi-Agent Systems. *(IJACSA) International Journal of Advanced Computer Science and Applications, , Vol. 2, No. 11, 2011* .

Saxena, V., & Kumar, S. (2012). Impact of coupling and cohesion in object-oriented technology. *Journal of Software Engineering and Applications, 5*(09), 671 .

Sękala, A., Foit, K., Banaś, W., & Kost, G. (2015). Design of robotic work cells using object-oriented and agent-based approaches. *Journal of Achievements in Materials and Manufacturing Engineering, 73*(2 .(

Serebrenik, A. (2014). Software architecture: Introduction

Sharma, T. (2012). Quantifying Quality of Software Design to Measure the Impact of Refactoring *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual* (pp. 266--271): IEEE.

Shehory, O. (1998). Architectural Properties of MultiAgent Systems .

Shehory, O. a. S., Arnon. (2001). *Evaluation of modeling techniques for agent-bases systems*. Paper presented at the Proceedings of the 5th International Conference on Autonomous Agents,, New York .

Sinha, B. R. a. D., Pradip Peter and Amin, Mohammad and Badkoobehi, Hassan. (2013). SOFTWARE COMPLEXITY MEASUREMENT USING MULTIPLE CRITERIA. *Journal of Computing Sciences in Colleges, 28,* 1 .162--55

So Young Moon, R. Y. C. K. (2013a). Verification of Requirements Extraction and Prioritization Based on Goal Oriented Use Case Approach by using Use Case Points. *nternational Journal of Software Engineering and Its Applications, 7*(4), 105-114 .

So Young Moon, R. Y. C. K. (2013b). Verification of Requirements Extraction and Prioritization Based on Goal Oriented Use Case Approach by using Use Case Points. *International Journal of Software Engineering and Its Applications, 7* .

STARON, M. (2016). SOFTWARE COMPLEXITY METRICS IN GENERAL AND IN THE CONTEXT OF ISO 26262 SOFTWARE VERIFICATION REQUIREMENTS. *Scandinavian Conference of Systems and Software Safety*, 1-23. doi: 10.13140/RG.2.1.1531.1763

Straub, J. (2014). *Comparing the blackboard architecture and intelligent water drops for spacecraft cluster control.* Paper presented at the AIAA SPACE 2014 Conference and Exposition.

Taylor, R. N., Medvidovic, N., & Dashofy, E. M. (2009). *Software architecture: foundations, theory, and practice*: Wiley Publishing.

Tekinerdogan, B., & Demirli, E. (2013). *Evaluation framework for software architecture viewpoint languages.* Paper presented at the Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures.

Thitisathienkul, P & ,.Prompoon, N. (2015). *Quality assessment method for software requirements specifications based on document characteristics and its structure.* Paper presented at the Trustworthy Systems and Their Applications (TSA), 2015 Second International Conference on.

Tran-Cao, D., Abran, A., & Lévesque, G. (2001). *Functional complexity measurement.* Paper presented at the Proceedings of the „International Workshop on Software Measurement (IWSM'01)”, Montreal, Quebec, Canada.

Tsui, F., Gharaat, A., Duggins, S., & Jung ,E. (2011). *Measuring Levels of Abstraction in Software Development.* Paper presented at the SEKE.

Umapathy Eaganathana, M. K. D., Abdifatah Farah Alic. (2016). A STUDY OF OBJECT ORIENTED SOFTWARE COMPLEXITY AND SIZING MEASURE. *International Journal of Pharmacy & Technology* .

van der Ven, J. S., Jansen, A. G., Nijhuis, J. A., & Bosch, J. (2008). Design decisions: The bridge between rationale and architecture *Rationale management in software engineering* (pp. 329-348): Springer.

Wagner, S., Florian. (2011 .(Abstractness, Specificity, and Complexity in Software Design. *ACM*, 35--42 .

Weyns, D. (2010). *Architecture-based design of multi-agent systems*: Springer Science & Business Media.

Wood, M. F., & DeLoach, S. A. (2001). An Overview of the Multiagent Systems Engineering Methodology. *Springer, 19*(57 .(

Yan, S. (2014). A Collaborative Filtering Recommender Approach by Investigating Interactions of Interest and Trust *Knowledge Engineering and Management* (pp. 173-188): Springer.

Zalewski, A. (2013). Modelling and evaluation of software architectures. *Prace Naukowe Politechniki Warszawskiej. Elektronika* .

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2001). *Organisational abstractions for the analysis and design of multi-agent systems.* Paper presented at the Agent-oriented software engineering.

الملخص

تصميم الانظمة المتعددة الوكلاء الكفؤة تعتمد على جودة المعمارية التصورية، حيث الجودة تؤثر بشكل جوهري في معمارية النظام البرمجي وتلعب دور رئيسي في وصف المعمارية. بالتالي خواص الجودة كقابلية الفهم، التعقيد، قابلية القراءة، قابلية الاختبار، وقابلية اعادة الاستخدام وغيرها يجب ان تؤخذ بعين الاعتبار في مرحلة مبكرة اثناء تطوير المعمارية البرمجية. من الجذير بالذكر، ان الانظمة الكبيرة مثل الانظمة المتعددة الوكلاء تتطلب الكثير من الاتصالات والتفاعلات لإنجاز مهامها، والذي قد يؤدي الى التعقيد في معمارية التصميم. هذه الاطروحة تحاول توضيح مواضع التعقيد في الذي قد يحدث اثناء وصف معمارية التصميم من خلال عدة جوانب، التجريد، التجزئة، و النمذجة وذلك عن طريق تقديم طريقة تهدف لوضع مجموعة من الارشادات لتقليل تأثير التعقيد، وضع توضيحات لكل ارشاد، وارشاد مطوري انظمة المتعددة الوكلاء لتصميم معماريات عالية الجودة، منخفضة التعقيد، وقابلة للفهم.

الطريقة يتم تطبيقها على حالة دراسة لنظام توصيات الكتب الذي يعتمد بدوره على الانظمة المتعددة الوكلاء حيث التعقيد سيتم قياسه عن طريق مقياس تعقيد الوظيفة (CTM) الذي يعتمد على طريقة نقطة حالة الاستخدام. ايضاً سيتم عرض نتائج التعقيد قبل وبعد تطبيق الطريقة.

الكلمات المفتاحية: انظمة المتعددة الوكلاء، المعماريات بشكل عام، خواص الجودة، انظمة التوصيات(RS).

طريقة لتحسين معماريات التصميم للأنظمة المتعددة الوكلاء
عن طريق تقليل جوانب التعقيد

قدمت من قبل:

هويده عبدالله على المرزكي

تحت اشراف:

د. توفيق محمد الطويل

قدمت هذه الرسالة استكمالاً لمتطلبات الحصول درجة الماجستير في علوم الحاسوب

جامعة بنغازي

كلية تقنية المعلومات

خريف 2017-2018