An Analysis and Reasoning Framework for Project Data Software Repositories

by

Ioanna Maria Attarian

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2012

Abstract

As the requirements for software systems increase, their size, complexity and functionality consequently increases as well. This has a direct impact on the complexity of numerous artifacts related to the system such as specification, design, implementation and, testing models. Furthermore, as the software market becomes more and more competitive, the need for software products that are of high quality and require the least monetary, time and human resources for their development and maintenance becomes evident. Therefore, it is important that project managers and software engineers are given the necessary tools to obtain a more holistic and accurate perspective of the status of their projects in order to early identify potential risks, flaws, and quality issues that may arise during each stage of the software project life cycle. In this respect, practitioners and academics alike have recognized the significance of investigating new methods for supporting software management operations with respect to large software projects. The main target of this M.A.Sc. thesis is the design of a framework in terms of, first, a reference architecture for mining and analyzing of software project data repositories according to specific objectives and analytic knowledge, second, the techniques to model such analytic knowledge and, third, a reasoning methodology for verifying or denying hypotheses related to analysis objectives. Such a framework could assist project managers, team leaders and development teams towards more accurate prediction of project traits such as quality analysis, risk assessment, cost estimation and progress evaluation. More specifically, the framework utilizes goal models to specify analysis objectives as well as, possible ways by which these objectives can be achieved. Examples of such analysis objectives for a project could be to yield, high code quality, achieve low production cost or, cope with tight delivery deadlines. Such goal models are consequently transformed into collections of Markov Logic Network rules which are then applied to the repository data in order to verify or deny with a degree of probability, whether the particular project objectives can be met as the project evolves. The proposed framework has been applied, as a proof of concept, on a repository pertaining to three industrial projects with more that one hundred development tasks.

Acknowledgements

First of all, I would like to deeply thank my supervisor, Professor Kostas Kontogiannis, for his invaluable assistance. His advice, that was very important and highly appreciated, was crucial and of great help during the conduction of this work. Throughout the four years of our collaboration, he has provided me with all the information and insight I required in the field of Software Engineering. His endless passion for knowledge and research is unique and admirable and his will and energy towards being assigned a large number of tasks simultaneously and completing them all successfully has been a great source of inspiration for me. I would also like to thank him for standing by me all these years not only as a teacher and supervisor but also as a friend, eager to help me through any difficulty I faced. I would like to thank him for his patience and effort while giving me guidance towards completing both my Bachelor's and my Master's degree and for sharing his knowledge with me so willingly.

I would also like to thank all my friends for being close to me while completing all the hard work that a Master's degree requires and, at the same time, adjusting to a very different environment. I would like to thank my long-time friends for showing all their care and interest even through electronic means of communication as well as my new friends that shared the work and laughter with me, even though we have known each other for a short period of time. I wish all of them the best of luck in their lives. May they achieve all the goals and targets they set for themselves through the years and may they be happy and content with everything they get involved in. I would also like to address a special thank you to George Chatzikonstantinou who spent a lot of time collaborating with me and has made the very stressful final period of my degree more enjoyable and easier to confront.

Finally and as importantly, I would like to thank my parents, Tassos and Sue, as well as Kiki and Vasilis for raising me all these years with such love and care and who stood by me in order to achieve all the goals I have set so far. With their support and help, they have been present at each and every moment of my life, assisting me through all difficulties and congratulating me for all achievements. I would also like to address a special thank you to my Aunt, Helen, and cousin, Christina, who were there for everything I needed for the last two years and supported me significantly to adjust in a very new and different environment compared to the one I was raised in and used to. I am very grateful to all of them for being the great people they are and I will always love and cherish them.

Table of Contents

List of Tables ix

List of Figures x

1 Introduction 1

1.1 Problem Description . . . 3

1.2 Motivation . . . 5

1.3 Thesis Contribution . . . 6

1.4 Outline of the Thesis . . . 7

2 Related Work 9

2.1 Research Baseline . . . 9

2.1.1 MOF/XMI . . . 9

2.1.2 Quality Assessment Standards . . . 10

2.1.3 Reasoning Frameworks . . . 12

2.1.4 Goal Trees . . . 20

2.1.5 Goal Tree Model Extensions . . . 22

2.2 Mining Software Repositories . . . 28

3 System Architecture 31

3.1 Architecture Description . . . 31

3.1.1 General Overview 31
3.1.2 Component Description 34
3.1.3 Sequence Diagram of the Functionality of the System 43
3.2 Summary 45

4 Goal Model Specification **50**
4.1 Goal Domain Model 50
4.1.1 HypoCategory Class 51
4.1.2 HypoType Class 51
4.1.3 Hypothesis Class 52
4.1.4 HypoPath Class 52
4.1.5 Iterator Class 53
4.1.6 CNFExpr Class 53
4.1.7 MLN Class 54
4.1.8 Verifier Class 54
4.1.9 GoalTree Class 54
4.1.10 GoalNode Class 55
4.1.11 AtomicGoal Class 56
4.1.12 DecompositionGoal Class 56
4.1.13 ExprProducer Class 56
4.1.14 Strategy Class 57
4.1.15 VerifierA Class 57
4.1.16 VisitorCreator Class 57
4.1.17 Visitor Class 58
4.1.18 AnnotationContainer Class 58
4.1.19 Annotation Class 58
4.1.20 Agent Class 59
4.1.21 Role Class 60

4.1.22 ServiceEngagement Class 60
4.1.23 Capability class 60
4.2 Project Management Goals 61
4.3 Summary 64

5 Goal Model Reasoning **68**
5.1 Goal Model to MLN Transformation 68
5.1.1 Extended Semantics 68
5.1.2 Rule Generation Semantics 71
5.1.3 Handling Uncertainty 73
5.2 DB Overview and Extraction 74
5.3 Process Outline 77
5.3.1 Hypothesis Domain XMI Population and Storage 79
5.3.2 Strategy and Category definition 81
5.3.3 AND/OR Goal Tree Creation 82
5.3.4 Rule Base Generation 82
5.3.5 Configuration XMI Population and Storage 84
5.3.6 Goal Tree Node Annotations Creation 84
5.3.7 Database Fact Extraction 85
5.3.8 MLN Reasoning 85
5.4 Summary 87

6 Case Studies **96**
6.1 Risk and Quality Assessment 96
6.1.1 Training Performance 102
6.1.2 Inferencing 103
6.2 Error Proneness 106
6.3 Threats to Validity 111

7 Conclusion **114**

7.1 Future Work 116

References **118**

List of Tables

4.1 List of Commitments defined for the Goal Model. 63

5.1 Data Related to "Design Database Schema" Task 75

5.2 Subset of FOL Formulas/Atoms Generated from the Goal Model in Figure 2.2. KB1-KB2 Consist the KB while G1-G3 the Ground Atoms Base . . . 76

5.3 A Subset of the KPIs Used 76

5.4 Ground Predicates Added to the DRGA for the Record of Table 5.1 for the corresponding KPIs of Table 5.3 76

6.1 Expected Outcome (E.O) and Probability Assigned by the MLN Analysis (MLN) for the 10 Test Cases. 103

List of Figures

2.1 Simple AND/OR Goal Tree Example 22

2.2 Run-time Goal Model Instance. Project agent adopts two roles in order to achieve its goals. 27

3.1 Block Diagram of the System Architecture 32

3.2 Component Diagram of the System Architecture 47

3.3 Layered architecture of Modeling Module 48

3.4 Layered architecture of UI Module 48

3.5 Functionality Sequence Diagram of the system 49

4.1 Hypothesis Goal Domain Model 65

4.2 Hypothesis Goal Domain Model (cont'd) 66

4.3 AND/OR Goal Trees 67

5.1 Block Diagram of the System Execution 80

5.2 Rule Base Generation 88

5.3 Sample Part of a KB .mln file 89

5.4 Database Fact Extraction 90

5.5 Sample Part of a KB .db file 91

5.6 Example of a .mln input file 92

5.7 Example of a .db input file 93

5.8 WalkSAT algorithm 93

5.9 MaxWalkSAT 94
5.10 Ground Network Construction Algorithm 94
5.11 MCMC : Gibbs sampling Algorithm 95

6.1 Time Required for Training against the Number of Training Cases 104
6.2 Normalized Weight for Formulas KB1 and KB2 of Table 1 105
6.3 Changes in Probabilities Calculated for Goals "Task On Budget", "Task On Schedule" and "High Task Quality" Against Changes in Sub-goals. 107
6.4 Failure Avoidance AND/OR Goal Tree 108
6.5 Failure Avoidance Results 111

Chapter 1

Introduction

The need for better and more complex industrial software systems has established Software Engineering practises as key activities in software specification, design, implementation, testing and maintenance. One of the most important areas where Software Engineering focuses on, is the development of techniques, methods and tools that aim to support the life cycle of complex software systems including the specification, design, implementation, testing and maintenance phases. Furthermore, the need for not only more functionalities but also, better quality has placed significant pressure on reassessing how the different phases of the software life cycle are affected [17]. Overall, we can consider that these elevated needs in market demands have affected the software life cycle as follows:

Initiation/Planning: Developers face additional challenges that concern the completion of software projects in less time, while meeting complex business objectives. Thus, the initiation and planning phase tends to consume a significant amount of time, first because the business analysis needs to be more carefully taken into consideration and planned, and second because of business planning related dependencies, since more parameters and more stakeholders need to be taken into account.

Requirements Gathering and Analysis: As systems become more complex, software development encompasses the fulfilment of more functional and non-functional requirements as well as the utilization of more human, software and hardware resources. Furthermore, the need for utilizing techniques, tools and models to systematically capture, encode and store diverse requirements specification artifacts. Subsequently, the requirements elicitation and analysis phase becomes more extensive, complex and time consuming.

Design: In the design phase where the functions and operations of the project are specified in detail using models, business rules, screen layouts and other documentation, it is evident that the increase in the size and complexity of requirements leads also to the definition of more complex models. That is, the structures need to be specified in more detail and the operations the software system delivers need to be broken down into a larger number of finer grained modules and components.

Build/Coding: The build/coding phase also becomes more challenging, since models which stem from a synthesis of many requirements and goals can be harder to develop in due time, while maintaining high quality as well as robust and reliable source code. In addition, the implementation of a larger number of functions in order to support an expanded set of system operations that encompass more complex dependencies, as mentioned above, is a more time consuming procedure per se.

Testing: The testing phase plays a significant role in the life cycle, since it ensures that the software maintains a high level of quality and reliability. Since the increase in requirements entails the increase in the size and complexity of the software source code, the number of testers and test cases will have to be raised in order to ensure the adequate testing of all parts of the software and the elimination of errors in all possible run time scenarios of execution. This requires again more time as well as monetary and human resources.

Maintenance: A larger amount of high-level and high complexity source code is more fault prone and can lead to maintenance issues. In addition, the maintenance operations need to be performed more frequently and in a larger number of iterations. This process may require more economic funds and human resources to complete it. Data related to these tasks, gathered and stored in specialized repositories can be mined and analyzed to yield important information on the product being built, and the process being followed. This paper proposes a goal and requirements driven framework that allows for the analysis of artifacts stored in such repositories.

All these issues that are pertinent to modern software development and mainly stem from the increase in requirements and design complexity concerning large software systems affecting all the stages of the life cycle, as explained above, has consequently negative results to cost and time required to commence and complete the development of the software system. This fact raises the need of fault prediction, as it can also be seen in [41, 39]. The construction of an analysis system capable of mining software project data and reasoning within a certain degree of probability or confidence, on the overall quality as well as cost, effort and development risks can be an important asset for the software developer. The use of such an analysis system includes not only the prediction of error proneness throughout the software development process but also, the decision making process towards taking actions that can potentially contribute to prevention of faults and development risks ([16]). This can eventually lead to cost reduction, since the developer is given the opportunity to reduce the probability of development risks or prevent their occurrence before they even begin to manifest themselves. It can also contribute to the early detection of possible modeling and planning flaws throughout the phases of the software life cycle.

1.1 Problem Description

The problem that this thesis addresses concerns the matter of developing a framework that mines software project data repositories for the purpose of reasoning for maintaining high software quality and reliability levels by assessing the development risk and quality properties of a software project [54]. The assessment is achieved by the analysis of information stored in project data repositories. The construction of a system that will predict possible risks and quality issues, in order to assist project managers and developers in making the right decisions and, therefore, potentially taking actions to prevent threats, could be of great significance and could become an indispensable tool in the development process. This problem becomes even more evident when taking into consideration the fact that the various risk factors that may produce errors in software development, are not mutually exclusive, but rather exhibit dependencies, which renders the task of predicting the probability of occurrence for each one more difficult. Therefore, many different aspects of the project development need to be taken into account and the dependencies appearing between all of the possible risks, need to be defined in a solid and consistent manner in order for the analysis to produce valid results.

Towards a solution to this problem, we propose the design and implementation of a framework whereby risk factors that can potentially introduce faults in a project, as well as the dependencies existing among them, can be modeled as hypotheses denoted as collections of goal trees. Since the software development process has become so much more complex, it is easily inferred that the definition of such hypotheses reflecting the entire set of risks posing threats to the correct outcome of a software project, along with the connections between them becomes a significant challenge. This procedure is challenging not only in the sense of constructing such hypotheses but also, for validating their completeness and consistency. Nevertheless, the idea of modeling comprehensive scenarios over probable risks and faults that can occur during development and coding can provide a very useful concept towards a possible solution to the prediction analysis ([11]).

One more important aspect to be taken into consideration when attempting to define hypotheses scenarios that fulfil the aforementioned criteria, relates to the different ways that risk factors affect one another. Therefore, it is easily understood that the definition of goals to be satisfied in order to conduct the analysis of software project data is a complex procedure, which requires significant attention at each step, as well as the application of combinatorial thinking in order to cover the multiple ties and dependencies appearing between the various risk factors under investigation which are not always straightforward or obvious.

However, these scenarios should not only take into account the risk factors and their inter-dependencies but should also ensure that they reflect real needs and goals of the industry in order to justify the applicability of the framework. The predictive analysis would be of no use if the hypotheses were not based upon actual goals, the achievement of which is sought by the project managers and developers. The reasoning held towards the calculation of fault occurrence probabilities for each project would perform best if it was ideally conducted over the set of goals and sub-goals that its own project development team has set. Nevertheless, for the purposes of this study, the goals defined have a more general profile and scope, allowing the analysis to be applied on a more general set of project criteria and goals. This should not necessarily be confronted as a negative point in the approach. On the contrary, it provides a notion of generality, which implies that basing the hypotheses definition on elements of modern risk assessment can permit the application of our method on data sets deriving from various different sources with adequate accuracy.

Those hypotheses, though, require the investigation of possible ways of their modeling in a formal manner, as it was also introduced in [9]. Along with a level of formality, the hypothesis modeling should be easily manageable and extendable so as to provide flexibility to the user who will be given the opportunity to set self-defined configurations according to more individual and targeted requirements and objectives.

Another point investigated is the choice of the hypotheses verification technique that will produce the most accurate results. It is evident that a system simply replying deterministically affirmatively or negatively towards the verification of some hypothesis would be rigid and most probably would be inaccurate as project parameters vary. In this respect, we aim for a technique that would take advantage of probabilistic methods so that analysis results can be provided with a degree of confidence as project data may vary or are partially available.

Finally, it is important to investigate ways of verifying whether the results of the above procedure, if such can be found, reflect real scenarios i.e. whether the outcome agrees with actual cases deriving from the industry. For this, the hypotheses should, consequently, be validated or denied using statistical reasoning, as explained above, when cross-examined against actual business intelligence data deriving from project information repositories belonging to real industry sources, such as Data Warehouses which have become and continue to be more and more common in the field of Data Mining ([55]). In order for such a system to be useful towards performing quality checks and obtaining status progress reports on software projects, it needs to be ensured that the outcome of the analysis corresponds to use cases that reflect real life project scenarios.

This thesis examines whether the above points can be fulfilled and proposes a structure of a framework which can give a solution to this problem to some extent.

1.2 Motivation

The problem described above poses the question whether a project data analysis would be a feasible task and whether it would produce results that reflect not only valid but also useful predictions. In this respect, the various factors that can potentially affect the project data analysis and assessment process very much depend on the individual demands of the project objectives and/or the team responsible for the development for the project under evaluation.

As mentioned above, due to the increase in the number of requirements to be met in modern systems, the entire life cycle of a software project becomes more complex, time consuming and requires more monetary and human resources to be completed while maintaining a high standard of quality according to the ”Vicious Square” model, [59]. This fact increases the error proneness and fault occurrence probability throughout all phases of the process. Towards reducing this risk, the construction of a framework that can enable the prediction of the appearance of faults and errors at an earlier stage of the process and even enable the indication of possible factors which, when improved, could assist towards avoiding their occurrence altogether, would be of the utmost significance.

For the above reasons, the challenge posed by this problem and the difficulties of its solution constitute the main theme of this research thesis. Steps towards the solution of such a problem constitute not only a very interesting research topic, but would also open new paths for the future of the fields of Risk Assessment and Management, as well as mining software repositories in general. In addition, the application of such a solution in the industrial sector could improve software development processes to a great extent by increasing productivity and also making products more competitive by establishing a higher level of quality.

1.3 Thesis Contribution

This thesis is presenting a framework which aims to conduct fault and quality prediction in large software projects. More specifically, the thesis proposes the :

- Definition of a Blackboard architecture and a corresponding framework, which provides validation of risk, quality and cost related hypotheses through probabilistic reasoning based on data deriving from project data repositories. The core component of the proposed Blackboard architecture consists of a component which plays the part of the main controlling unit of the system based on a publish/subscribe protocol. More specifically, it collects all the updates from the repositories detected during run-time from all other components which are subscribed, and redistributes information by publishing messages and information to them appropriately. Furthermore, it undertakes the responsibility of gathering all external user input and intermediate data produced during the execution, in order to transmit important information to the components which require it, regarding the state of the process and in order to trigger their execution accordingly. Each component encompasses diagnostic and analysis knowledge in the form of goal models so that to verify or deny specific risk, quality, cost and plan hypotheses.

- Design and implementation of a goal domain model that will enable the definition of the hypotheses to be validated or denied, the type of which can vary depending on the users' requirements. The structure utilized in order to define such a goal domain model aims to provide flexibility and extensibility in order to enable the assembly of different types of hypotheses covering a large range of risk assessment aspects, as well as the inter-dependencies between them, without problems. For this, the notion of AND/OR Goal Trees is utilized. The application of AND/OR Goal Trees in requirement specifications modeling is not new. It constitutes an already established and commonly applied theory in the research community of Requirements Engineering [57]. In addition, towards modeling the inter-dependencies between goals and sub-goals, the notion of Contributions and Commitments through Agents and Roles, has been introduced. These notions have been proposed in the field of Requirements Engineering and their application has led to interesting results [6, 7].

- Representation of the aforementioned goal trees by projection to the first - order logic world which enables the application of Markov Logic Networks as a probabilistic reasoning method. The AND/OR Goal Tree is a structure easily transformed into first - order logic predicates that constitute the knowledge base and simulate the world in effect, due to the by definition already existing connections between the sub-goals through conjunctive and disjunctive relationships. This benefit of the AND/OR Goal Tree structure provides a straightforward way of mapping the goals and sub-goals that constitute the hypothesis under analysis, into logical expressions based on Boolean Algebra and, furthermore, into conjunctive normal form which is also required for the construction of the appropriate for the specific goal model, Markov Logic Network.

- Design of a prototype implementation with goal trees related to the risk, cost and quality hypotheses so that they can be validated against real project data.

1.4 Outline of the Thesis

The text of the thesis is structured as follows :

Chapter 2: In this chapter, a list of related work previously held, is presented. Briefly, the subjects covered in chapter 2 which are relative to the work conducted for the purposes of this thesis, include the MOF (Meta - Object Facility) and XMI (XML Metadata Interchange) for the definition of the domain models and the parsing of the input files to the system respectively. Furthermore, the chapter presents pointers to appropriate KPI libraries as well as the ISO standards ISO/IEC TR 9126 2 and ISO/IEC TR 9126 - 3 in order to provide a level of formality to the definition of the project management and risk assessment goal models so as to ensure they reflect actual risk factors appearing in the real industry world. Finally, the chapter discusses the Markov Logic Network theory for the probabilistic reasoning method followed throughout the analysis, the AND/OR Goal Tree Domain Model theory for the goal modeling of the hypotheses, the Contributions and Commitments theory introduced so as to support the dependencies between various goals and sub-goals defined, as well as other subjects of Business Intelligence and Analytics.

Chapter 3: In this chapter, the system architecture is presented. Specifically, the different architecture styles which are used in various parts of the system are explained and the architecture is described thoroughly for each component in terms of interfaces, offered services and functionality. Finally, a sequence diagram of the system is given.

Chapter 4: In this chapter, the domain goal model and its notation and semantics are presented along with the three goal trees defined for the needs of this thesis. In addition, a discussion regarding the logic behind the goal and sub-goal formalism selection along with the KPI choice is presented. Finally, the necessary configurations in order to run the reasoning analysis successfully and their meaning are described.

Chapter 5: In this chapter, the goal model reasoning is described. More specifically, we explain one of the focal points of the framework that includes the logic with which a goal model can be transformed to a Markov Logic Network, the rules and semantics in effect, in order to conduct this transformation along with some examples of the logic application.

Chapter 6: This chapter includes the case studies i.e. the training procedure as well as the test case scenarios executed in order to prove the validity and, consequently, show the usability of the system, are described and analyzed.

Chapter 7: This chapter concludes the thesis and provides pointers for further research.

Chapter 2

Related Work

The work presented in this thesis, touches upon a number of fields in the area of Software Engineering. First, it utilizes work performed in modeling and in particular the Meta - Object Facility and XMI. Second, it touches upon the area of Quality Assessment and in particular the ISO Standard 9126. Finally, it addresses the area of reasoning and in particular the area of statistical reasoning and Markov Logic Networks. In the following sections we will discuss these areas of related work.

2.1 Research Baseline

2.1.1 MOF/XMI

The **Meta - Object Facility (MOF)** [46] is the core Object Management Group (OMG) standard for model - driven engineering. MOF was concluded by the need for a meta modeling language and yields a taxonomy of models divided into four layers. The M3 level is at the top of the hierarchy and is the language MOF. This layer can be used to describe M2 - models such as UML itself. The M2 - models can be used to denote M1 - models. In the UML case, a particular UML class domain model would belong to the M1 - level. At the bottom of the hierarchy is the M0 - level which consists of the data that is the actual instances of the M1 - level model. The MOF standard is a useful environment for developing models and schemas since it offers flexibility to the developer. Specifically, it provides portability and extensibility to the developed models, since it allows models to be easily exported from one application, imported into others, transferred through a network, stored in and retrieved from a repository or transformed into different formats such as XMI.

The **XML Metadata Interchange (XMI)** [47] is the Object Management Group (OMG) standard format for model metadata transmission and storage through Extensible Markup Language (XML). According to OMG, data can be classified into two categories, abstract models and concrete models. The abstract models represent the semantic information where the concrete models represent the visual diagrams. From this classification, abstract models are the most useful since they are instances of MOF - based modeling languages like UML. In this respect, the XMI format is often used to exchange metadata between UML - based modeling tools and MOF - based metadata repositories in distributed heterogeneous environments. It is also often used as a means of passing models from modeling tools to source code generation tools.

For the purposes of this thesis, the MOF standard was used so as to define the hypothesis, warehouse and goal tree domain models. In order to accomplish that, the Magic Draw tool was utilized. The MOF diagrams were then exported in XMI format compatible with the Eclipse Modeling Framework (EMF) meta model (Ecore) [23]. The produced XMI files were imported into EMF where automated code generation was applied to the model. Exemplary population of the models was done using the runtime environment provided by EMF.

2.1.2 Quality Assessment Standards

A crucial point to the work elaborated for the purposes of this thesis, was the choice of the appropriate criteria [2] in order to ensure that the quality prediction and verification analysis would be held in respect with widely accepted and applied quality standards and features. Two of the most known Quality Standards are the ISO International Standards and the Capability Maturity Model Integration (CMMI) Standards. In addition, the Key Performance Indicator (KPI) Library was a significant contributing source of research concerning risk assessment evaluation features since it describes the most recent fashion followed in Risk Analysis. These are discussed below in more detail.

ISO International Standards

The International Organization for Standardization (ISO) is an international standard setting organization which utilizes representatives from various national standardization organizations. Although it is considered a non - governmental organization, it has the ability to set standards that become technical recommendations in various countries through treaties or national standards. Among ISO's main products are the International Process and Product Quality Standards. The purpose of these International Standards is to assist enterprises to ensure the quality of their products and the integration of the operation environment.

This thesis has focused more on two ISO quality standards and specifically, the ISO/IEC TR 9126 - 2 [49] and ISO/IEC TR 9126 - 3 [50]. There are also other studies focusing on the same subject that were based on the ISO Standards ([19]). These ISO standards were useful to draft the logic by which a process or a product is validated for its quality in the proposed framework.

CMMI

Another commonly used and acknowledged set of Quality Standards is the Capability Maturity Model Integration (CMMI) ([68]). CMMI constitutes a process improvement approach providing the essential elements for enhancing the qualities of processes. CMMI Standards, like the ISO Standards described above, can be utilized in order to conduct process improvement across a project, a division or an entire organization. They assist towards integration of traditionally separate organizational functions, setting process improvement goals and priorities, providing guidance for quality processes and, finally, providing reference points for assessing and appraising current processes.

The CMMI contains directives concerning three areas of interest: the CMMI for Development Model which concerns product and service development, the CMMI for Services Model concerning service establishment, management and delivery, and the CMMI for Acquisition Model which has to do with product and service acquisition.

The above CMMI models are collections of best practices that can be compared to the features and results of a project and that can suggest ways of improvement for its processes. A formal comparison of a CMMI model to the processes of a project is called an appraisal. The Standard CMMI Appraisal Method for Process Improvement (SCAMPI) incorporates the best ideas of several process improvement appraisal methods.

Key Performance Indicator (KPI) Library

The Key Performance Indicator (KPI) Library [42] was launched in September of 2007 by Team42 in an attempt to provide customers with methods that could potentially be useful towards risk assessment of their products and services. The KPI Library contains features that, unlike other techniques of risk analysis, are not qualitative but rather more quantitative. The KPIs provided constitute actual metrics in the form of mathematical formulas that correlate various aspects of a product development that can be measured, in order to calculate values reflecting a certain characteristic of that product. The resulting value of each of those characteristics can be compared to standard thresholds of acceptance, set by the developing team according to their demands and goals, in order to assess the progress of the process and, consequently, make decisions towards the improvement of the project quality status. The generic profile of the KPIs as a notion makes them easily applicable in many additional fields unrelated to Software and Requirements Engineering, which span from agriculture to accommodation, to real estate and to various other types of industry.

The reason why the KPI notion is an important means of quality and risk assessment, involves their quantitative character, as mentioned above. More specifically, by providing mathematical representations for several aspects of a product that are mostly conceived intuitively, KPIs add a layer of objectiveness to the risk analysis procedure. The subjectiveness of any potential intuitive result when conducting risk analysis renders it difficult to present to an audience and inadequately persuasive for prospective clientele since its credibility and validity can be more easily assailed. On the other side, numerical results have always been more appealing to the human logic and, therefore, can be presented as a more standardized and less intuitive method of proving the accuracy of a risk analysis study. Furthermore, the collection of such metrics in an online library, adds an extra notion of formality, since it is an effort to standardize the existence and utilization of such a method towards quality and risk assessment by making it publicly known and available for use. Therefore, the KPI Library can be viewed in some sense as a form of a manual.

2.1.3 Reasoning Frameworks

Reasoning Frameworks refer to all those theories that can provide logical deductions, often with a sense of probability measurement through a generalized method for logic application. Reasoning frameworks in relation to the application of this thesis, are the Markov Logic Networks theory, the Dempster - Shafer theory, Fuzzy Inference ([20]), Function Point Analysis ([18]), Fuzzy Delphi ([18]), Fuzzy Nonlinear Regression Modeling ([68]), Artificial

Neural Networks ([12]) and Support Vector Machines ([12]). The Markov Logic Networks theory was the one utilized in this work.

Markov Logic Networks

Richardson and Domingos [56, 10] have introduced MLNs as a way to combine benefits from both first-order logic (FOL) and probabilistic graphical models in a single representation. More specifically, an MLN constitutes a knowledge base (KB) of predicates and ground atoms. The *Markov Logic Networks* theory combines the benefits of Markov Networks along with the benefits of first order logic. More specifically, an MLN is a first-order knowledge base (KB) that can be used to perform probabilistic inference as opposed to strong inference performed in pure first-order logic (FOL). In FOL a KB is defined as a set of well-formed FOL formulas which are formed by using logical connectives and quantifiers to combine ground atoms (predicate symbols applied to a number of terms). By assigning truth values to each possible ground atom, we construct a possible world in which the KB may or may not be true. In other words, if a world violates at most one of the formulas in the KB then it has zero probability of being an interpretation of the KB. On the contrary, in MLNs a world may still be a possible interpretation with a non zero probability, even if some formulas are violated. The degree of probability for the world to be a proper interpretation, depends inversely on the number of formulas violated and also on how strong the constraints introduced by the violated formulas are [53].

As discussed in [38], Markov Network or Markov Random Field is a model of the joint distribution of an n - set of variables $X = (X_1, X_2, ..., X_n)$. It is composed of an undirected graph G and a series of potential functions φ_k . Each node of the graph corresponds to one variable of the set X and there is one function φ_k for each clique in the graph. The potentials functions are non - negative real - valued and each represents the state of the corresponding clique. Taking the above into account, the resulting joint distribution is :

$$P(X = x) = \tfrac{1}{Z} \prod_k \varphi_k(x_k)$$

where x_k is the state of the variables appearing in the k - clique and Z is the *partition function* :

$$Z = \sum_{x \in X} \prod_k \varphi_k(x_k)$$

Markov Networks are often represented as log - linear models. In this case the potential functions are replaced by an exponentiated weighted sum of features of the state. A feature may be any real - valued function of the state. Thus, the joint distribution takes the form :

$$P(X = x) = \frac{1}{Z} \exp(\sum_j w_j f_j(x))$$

A *first - order logic knowledge base* is a set of first order logic sentences or formulas. These consist of constants, variables, functions and predicates. *Constants* represent instance objects in the domain. *Variables* range over the objects in the domain. *Functions* represent mappings from object tuples to objects. *Predicates* represent relations between objects in the domain or attributes of objects. Apart from the above, the following are also defined :

- An *interpretation* is the definition of the symbolic representation of all the above elements of the first order logic.
- A *term* is any expression representing an object in the domain like a constant, a variable or a function applied to a tuple of terms.
- An *atomic formula* or *atom* is a predicate symbol applied to a tuple of terms.
- A *formula* is constructed through recursion of atomic formulas and logical connectives and quantifiers.
- A *ground term* is a term that does not contain variables. A *ground atom* or *ground predicate* is an atomic formula that consists only of ground terms. The ground term and ground atom or predicate were of significance throughout the conduction of this thesis.

The *Markov Logic Network* (MLN) theory combines the ease of representation provided by the use of a first - order logic knowledge base along with the flexibility of the probabilistic methods of Markov Networks. More specifically, a first - order logic knowledge base can be viewed as a set of hard constraints which define a possible world. Thus, if a world does not satisfy at least one of them, then it has zero probability of being valid. The Markov Logic Network theory attempts to soften those constraints. In this case, a world which fails one or more constraints of the first - order logic knowledge base is considered to be more improbable but not impossible i.e. the less constraints a world violates, the more probable it is considered to be. Each formula of the first - order KB is assigned a weight which indicates the strength of the constraint. According to this, the higher the weight, the greater the difference in log probability between a world that satisfies the formula and another that does not, other things being identical. Let it be noted that for the purposes of this thesis, all formulas and constraints were assumed of equal strength. We give a formal definition of a Markov Logic Network :

A Markov Logic Network L is a set of pairs (F_i, w_i), where F_i is a formula in first - order and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, ..., c_{|C|}\}$, it defines a Markov network $M_{L,C}$ as follows :

1. $M_{L,C}$ contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1 if the ground atom is true, and 0 otherwise.

2. $M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.

An MLN can produce different networks given different sets of constants. These networks which are called *ground Markov networks*, widely vary in size, but all share some regularities in structure and parameters given by the MLN. Thus, the probability distributions over all x specified by the ground Markov network $M_{L,C}$, take the following form :

$$P(X = x) = \tfrac{1}{Z} \exp(\sum_i w_i n_i(x)) = \frac{1}{Z} \prod_i \varphi_i(x_i)^{n_i(x)}$$

where $n_i(x)$ is the number of true groundings of F_i in x, x_i is the state of the atoms appearing in F_i, and $\varphi_i(x_i) = e^{w_i}$.

There are two operations which can be executed via MLNs :

- **Learning** : Via this procedure, we can learn MLN weights from one or more relational databases after having assumed a closed world and, therefore, that every ground atom that does not exist in the database is false. The *Learning* procedure will not be of any concern within this thesis.

- **Inference** : A basic use of MLNs is their capability of providing answers to queries concerning the probability of one formula F_1 being true given that a second probability F_2 holds. Therefore, if F_1 and F_2 are two formulas in first - order logic, C is the set of constants of the world and L is an MLN, then the resulting probability would be :

$$P(F_1|F_2, L, C) = P(F_1|F_2, M_{L,C}) = \frac{P(F_1 \wedge F_2|M_{L,C})}{P(F_2|M_{L,C})} = \frac{\sum_{x \in X_{F_1} \cap X_{F_2}} P(X=x|M_{L,C})}{\sum_{x \in X_{F_2}} P(X=x|M_{L,C})}$$

where X_{F_i} is the set of worlds where F_i holds.

When viewed from a practical perspective, the Alchemy tool was considered a significant asset that permits taking fully advantage of the potential MLNs present. Alchemy is a software package with built-in algorithms implemented for statistical learning and inference based on the MLN theory [37]. The set of rules that constitute the knowledge base along with the set of grounded atoms describing the world, are both input to the Alchemy tool in order to construct the appropriate Markov network. Alchemy also supports the definition of both hard and soft constraints in order to allow weight learning for the later.

Dempster - Shafer theory of Evidence

The Dempster - Shafer theory, as it is analysed in [36], has the advantage of being able to model the narrowing of the hypothesis set with the accumulation of evidence which can be applied to succeed expert reasoning in various fields. It is often that an expert will use evidence that does not have any bearing on a single hypothesis of the hypothesis set but rather a larger set of the original set. Therefore, the Dempster - Shafer theory, supplied with functions and combining rules, can be very suited to represent this type of evidence and its aggregation.

One more advantage of the theory is that, unlike many previous theories like the Bayesian for which one can understand more about its applications in Software Engineering by looking into [4] , it lifts the constraint according to which the belief of one hypothesis implies the remaining belief corresponds necessarily to the negation of the same hypothesis. The fact that, in many cases, evidence supporting partially one hypothesis should not be accounted automatically as evidence partially against the negation of the hypothesis, reflects probably more the reality. Thus, in the Dempster - Shafer theory the beliefs in each hypothesis are allowed to sum up to a number less than or equal to 1.

The Dempster - Shafer theory assigns one number in the range $[0, 1]$ to indicate the belief in each hypothesis provided as a piece of evidence. This constitutes a degree to which the evidence supports the hypothesis. The impact of each distinct piece of evidence on the subsets of Θ, where Θ is the hypothesis set, is represented by a function called a Basic Probability Assignment (BPA). A BPA is a generalization of the traditional Probability Density Function. Next, belief functions denoted as Bel, are defined for each BPA, m, and assigned to every subset A of Θ the sum of the beliefs committed exactly to every subset of A by m.

The Dempster - Shafer theory, as briefly discussed, provides many advantages and, therefore, is firmly considered for the extension of the work projected in this thesis.

Fuzzy Inference

As mentioned in [20], there are many cases that the complexity and ambiguity of some problems does not enable the ease of finding an optimal solution in order to make correct decisions. Especially when it is required that human beings make these decisions, taking into account all criteria and aspects of a problem prior to reaching a final decision can demand much time and effort. For this, using one single criterion or method can prove inadequate, therefore, other solutions based on non deterministic techniques and methods are researched. One of these methods is fuzzy logic.

More specifically, decision making problems often involve multiple criteria that may interact in a negative or positive way, which means there is a kind of redundancy between two criteria or a kind of synergy among them respectively. The subadditivity that the fuzzy measures present, is proven useful towards modeling negative interaction among criteria. Similarly, the superadditivity they present, implies synergy, which allows the modeling of the positive interaction between criteria. The most appropriate method of aggregating these sets of criteria and reaching to a final result is determined after their importance level is determined. The fuzzy integral is considered as one possible solution towards this aggregation of criteria. An important feature of fuzzy integral is the ability of representing a range of interaction between criteria, from redundancy to synergy i.e. both negative and positive interactions between different criteria.

Function Point Analysis and Fuzzy Delphi

Function Point Analysis [18] refers to the use of function-related measures upon software projects towards decision making. Function points are a unit measure for software and they are considered to be ordinal measures. Function Point Analysis is an attempt to define software size through bypassing the difficulties that lines of code (LOC) introduce as a measure, and, at the same time, to provide assistance towards effort requirements prediction associated with software development.

Function Point Measures are technology independent i.e. they are unrelated to a specific hardware platform or a certain programming language implementation. Function Points are composed of the output and input of a program, its interaction with users, the external exchange interface and the number of files the system uses within its execution. The use of Function Points as measures encompasses their enumeration. That is, Function Point Counts at the end of each software life cycle phase, i.e. requirements, analysis, design, code, testing and implementation, can be compared. The function point count at the end of requirements and/or designs can be compared to function points actually delivered. The result can constitute an indication of how well requirements were gathered by and/or communicated to the project team.

Fuzzy Delphi is based on the traditional Delphi theory and its combination with fuzzy set theory. The traditional Delphi method refers to the collection of information from high qualified experts in order to make predictions concerning future events. More specifically, it involves the selection of a panel of experts who, then, release their opinions over certain features of the study held. The answers are collected and analyzed statistically. After the data processing, the experts are again requested to give another written response. This procedure is repeated for several rounds of questioning and obtaining written responses. The outcome is utilized in order to solve a problem or to forecast an event in the future. Fuzzy Delphi combines the maximum-minimum method along with a cumulative frequency distribution and fuzzy scoring to compile the expert opinions into fuzzy numbers.

Fuzzy Nonlinear Regression Modeling

The Fuzzy Nonlinear Regression (FNR) model is defined by a set of fuzzy numbers. The computed output fuzzy form, therefore, is a function of the n-space of input variables and the aforementioned set of fuzzy numbers. Its vagueness is defined as the sum of all widths of the fuzzy parameters. More specifically, a fuzzy nonlinear regression model is constructed by the division of the fuzzy input space into appropriate sub-spaces and the definition of the input/output conventional linear regression relation in each sub-space.

Fuzzy Nonlinear Regression, according to [68], can be used as a modeling technique which provides a method for predicting fault ranges in software modules. As opposed to the classical linear regression, FNR modeling provides a fuzzy number as the output. Since predicting exact numbers when referring to faults in each program module is often not necessary and even impossible, the FNR model can predict the interval that the number of faults of each module falls into with a certain probability.

Artificial Neural Networks and Support Vector Machines

Artificial Neural Networks (ANN) or simply Neural Network models in artificial intelligence are essentially simple mathematical models based upon a function $f : X \rightarrow Y$ or a distribution over X or both X and Y. However, there are cases where the model can be significantly associated with a specific learning algorithm or rule. An Artificial Neural Network is defined through a class of such functions, the members of which are distinguished by varying parameters, connection weights or specifics of the architecture such as the number of neurons or their connectivity. The three main types of parameters that define an Artificial Neural Network are the interconnection pattern between different layers of neurons, the learning process for updating the weights of the interconnections and the activation function that converts a neuron's weighted input to its output activation. In addition, one great advantage of artificial neural networks is their direct association with a specific learning procedure that implies the choice of a cost function which is minimized in order to conclude to the optimal solution. The most widely applied learning algorithms and methods are the supervised learning, the unsupervised learning and the reinforcement learning procedures. The first requires that a set of example pairs of input and output are given and targets to find the function within the set of allowed functions optimally matching the given set. The second requires that some data are given along with the function to be minimized, which can be any function of the data within the output collection of functions of the network. For the third, the data are usually not given but produced by the interaction between some agents, which triggers the generation of a corresponding observation and cost, according to some usually unknown dynamics. This aims towards the definition of a policy for selecting actions and, simultaneously, minimizing some measure of a long-term cost.

A support vector machine (SVM) is related to supervised learning methods i.e. techniques aiming to analyze data and recognize patterns, used for classification and regression analysis. An SVM receives as input a set of data and attempts to predict one possible class between two, which matches that input. Therefore, the SVM is characterized as a non-probabilistic binary linear classifier. More specifically, the SVM creates a model assigning new examples into one of two categories through a training algorithm, which is initially given a set of training examples where for each it is known to which category it belongs. Therefore, an SVM model is a mapping of examples as points in space so that they belong to two categories separated by a distinct gap, which is as wide as possible. Any new example is then mapped into that same space and the machine predicts the category to which it belongs according to which side of the gap it is placed through the aforementioned mapping. In other words, the SVM builds a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, in order to conduct classification, regression or other procedures. From this and after taking all the above into consideration, it can be inferred intuitively that the separation is more accurate if the hyperplane demonstrating the largest distance to the nearest training data point of any class, is chosen.

In [12], the two aforementioned machine learning algorithms are compared. More specifically, a formal model for risk identification has been constructed and actual cases deriving from the software development industry were used in order to build a risk prediction model. The two methods were, then, applied towards the evaluation of the performance of the model. Experiments revealed better performance results when applying the SVM modeling technique upon assembling the risk prediction model.

2.1.4 Goal Trees

In the Software Engineering field, software requirements constitute an important step at the development process. That is, the engineers must specify the requirements of the system to be built so that its quality can be assured as much as possible.

One of the many methods proposed for the specification and modeling of a system's functional and non - functional requirements is the Goal Tree model.

The Goal Tree model is based on the top - down decomposition to which the operational requirements of a system can be modeled through the definition of specific goals to be satisfied. Those goals can be segmented further into sub-goals whose validation will ensure that the original goals are satisfied. One type of Goal Trees that has been accepted by the research community and has been used in Software Requirements Engineering, is the AND/OR Goal Tree Model, [57, 51]. According to this model, a goal can be divided into sub-goals which are represented as the children of the first. The same applies for each sub-goal iteratively. Therefore, it can be easily seen that in this way, a tree form of goals is constructed. A specific goal, that is a goal tree node, can be of type **AND** which implies that the children of the goal are connected by logical conjunction. Thus, the goal is fulfilled if and only if all its sub-goals, i.e. its children nodes, are also satisfied. These are noted as **hard** goals since they demand all of their sub-goals to be fulfilled in order to be satisfied. There are also goals of type **OR**. This implies that the children nodes of such goals, that is their sub-goals, are connected by logical disjunction. Therefore, if at least one of the sub-goals is fulfiled, that leads to their parent goal to be considered as having been satisfied as well. These goals are called **soft** goals since they do not require that all of their sub-goals are satisfied in order to validate as true, but even one of their sub-goals is enough to prove them.

The AND/OR Goal Trees offer a convenient means of representation of goals which was very useful for the modeling of the hypotheses within this thesis. They can easily be constructed and traversed in order to produce the necessary output. Their direct junction with the field of *Boolean Algebra* and first - order logic, [21], also makes them very appealing since this is the base for the construction of the Markov Logic Networks required for the verification of a potential risk hypothesis. Their straightforward traversal and interpretation into logical expressions that can easily be transformed into conjuctive normal form and given as input to the Alchemy tool for the definition of the appropriate MLN for each case in order to verify the validity of the goal tree root, has made them more preferable and a more viable choice for the modeling demands of this thesis. In addition, they offer a greater level of extensibility since with the annotations built and linked to each goal tree node, that is each subgoal, it can be understood that there there is no limitation on the type of information that can be stored in an AND/OR Goal Tree. This allows more freedom to the developer in order to use such Goal Trees for other applications as well, by altering the annotations according to his demands.

An example of an AND/OR Goal Tree can be seen in Figure 2.1 below :

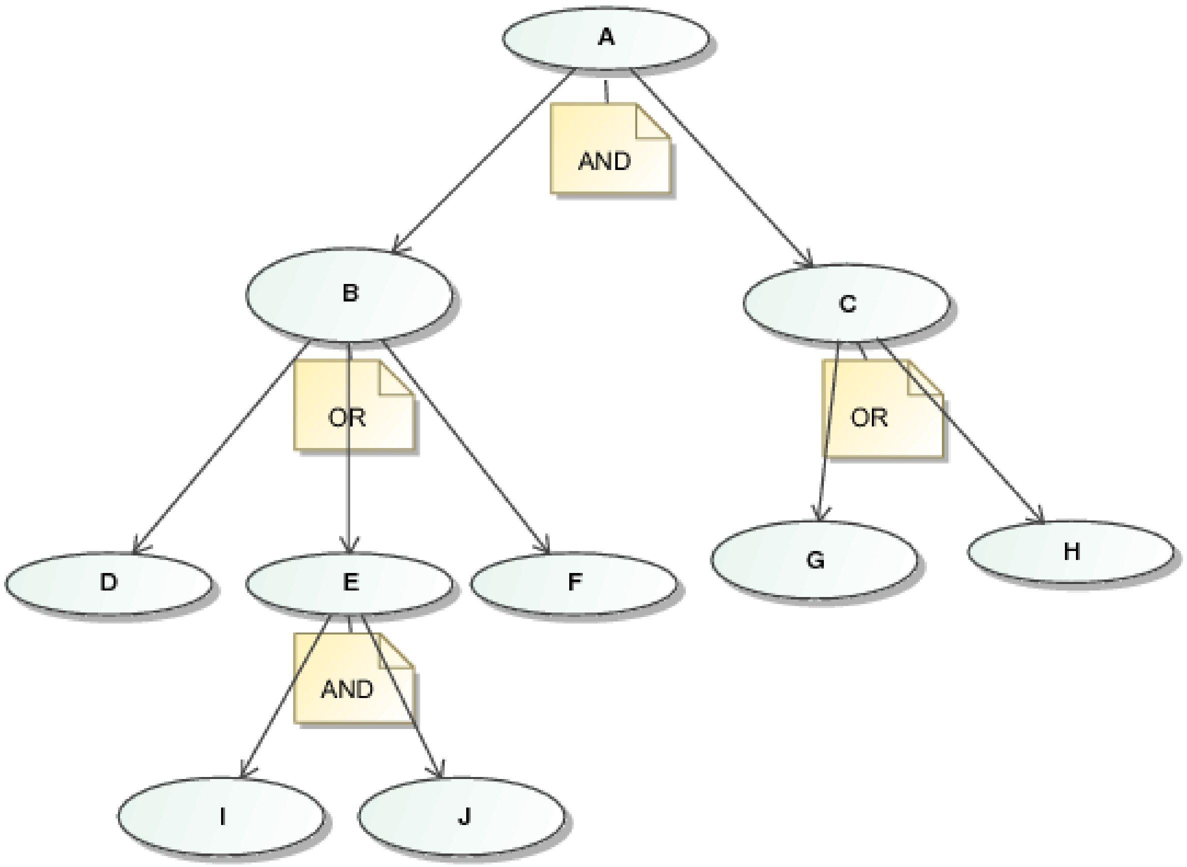


Figure 2.1: Simple AND/OR Goal Tree Example

2.1.5 Goal Tree Model Extensions

In most analysis domains, as mentioned, the various goals and sub-goals defined as the desired set to be achieved, demonstrate inter-dependencies among them that can be numerous and introduce a level of complexity [33] . Therefore, it becomes clear that this is not fully covered by the simple AND/OR Goal Tree model definition. These relations can sometimes constitute an unpredictable factor towards conducting an adequately accurate analysis. In addition, they are often overlooked and hard to understand but, nevertheless, play a very significant part. For this, the necessity of constructing modeling solutions in order to incorporate them to the reasoning method body, rises.

Agents and Roles

Agent Oriented Programming reflects mostly the need for open architectures that can be adaptable to change and evolution in order to accommodate and incorporate new components and also meet new and more complex requirements. In addition, due to the fast pace of hardware evolution, software is required to operate on a larger number of different platforms, without recompilation, and demonstrate compatibility with many operating systems and users without having to make assumptions showing robustness, autonomy and proactiveness. Thus, in order for a software or hardware system to qualify as an agent, it is often important to own attributes such as autonomy, social ability, reactivity, and proactiveness as well as mobility, veracity and rationality.

The notion of agents is defined in terms of cognitive concepts such as goals, beliefs, desires, intentions and obligations, as mentioned in [6]. According to Chopra et al. [6, 7, 8], an agent is specified in terms of a goal model which reflects its motivations and adds a layer of abstraction over the low-level details of control and data flow between the components of an actual system implementation of the goal model. An agent owns a high-level internal decision making logic. More specifically, it owns a set of policies it is obliged to obey to which leads to a certain rationale supporting its decision over making choices that do not produce any contradictions.

Physically, agents represent the participants in an application interacting with each other in order to fulfil their goals and requirements to produce certain results and provide specific services for the user. A level of autonomy and heterogeneity is one the main characteristics of service-oriented applications. Therefore, all participants are engaged to each other through a service enactment by a set of interactions. The term of interaction implies a dynamic character of this service engagement i.e. the participants are free to join or leave at their own will. This gives an even more open sense to the notion of service enactments and interactions, meaning that during the design phase, the identity of the participants is not required to be known []. That allows the dynamic engagement of a specific participant in the run-time environment during execution of the application.

As mentioned above, the characteristic of open architecture systems is that the identity of the agents participating and interacting with each other, is not required to be known in the design phase. In order to achieve this masking successfully, this interaction is held through the adoption of roles from each agent. This introduces the notion of roles. A role is defined as a series of responsibilities and policies that rule the behavior of the participant adopting it according to the goals the later is required to fulfil. Therefore, an agent adopts a certain role in a service-oriented application in order to achieve its set of goals through the interaction with other agents also adopting certain roles, under the constraints and obligations that these roles dictate.

Contributions

The notion of contributions is an approach towards modeling the complex relations between goals and sub-goals. A contribution is defined as the ability of a goal to contribute positively or negatively to the achievement or denial of another goal within the model. More specifically, following [7], contributions between goals can be of four types denoted as : *++S (g, g')* which indicates that the achievement of goal *g* achieves goal *g'*, *–S (g, g')* which means that the achievement of goal *g* denies the achievement of goal *g'*, *++D (g, g')* which specifies that denying the achievement of goal *g* infers the achievement of goal *g'* and, last, *–D (g, g')* which states that the denial of achievement of goal *g* results in the achievement of goal *g'*.

The formal representation of contributions using mathematical definitions, can be introduced by the relations below [7] :

Syn_1. $G \rightarrow AND\alpha$, where $G \subseteq \psi$ and G is not empty. It means that achieving all the goals represented by each of the symbols in G is a way of achieving α.

Syn_2. $G \rightarrow OR\alpha$, where $G \subseteq \psi$ and G is not empty. It means that achieving a goals represented by one of the symbols in G is a way of achieving α.

Syn_3. $\alpha \rightarrow \beta$, equivalent to $++S(\alpha, \beta)$.

Syn_4. $\alpha \rightarrow \neg\beta$, equivalent to $--S(\alpha, \beta)$.

Syn_5. $\neg\alpha \rightarrow \beta$, equivalent to $--D(\alpha, \beta)$.

Syn_6. $\neg\alpha \rightarrow \neg\beta$, equivalent to $++D(\alpha, \beta)$.

Capabilities and Commitments

Taking into consideration all the above definitions and clarifications concerning the notions of agents and roles, it is important to elaborate further on the notion of goal support for an agent. It is defined that a goal is supported by an agent within a service engagement when this goal can potentially be achieved by this agent when it takes some role. The agent can either achieve a goal immediately, if that goal belongs to its set of capabilities or if the goal is being provided to it by another agent through a commitment.

The term capability refers to the set of goals for an agent, which it has the inherent pre-existing ability to satisfy on its own. In other words, the capabilities of an agent are the goals which it can fulfil automatically without any other preconditions. The capabilities of an agent are defined in advance as an agent's characteristic, prior to any execution configuration and they are totally independent of any role the agent can be assigned to i.e. the capabilities of an agent continue to be such, under any interpretation, regardless of the role it undertakes.

The notion of commitments, as defined in [6], is introduced through the form of commitment protocols as a high-level abstraction for interaction between agents. The commitment protocol includes messages along with their meanings in terms of commitments. When there is a commitment binding between two agents participating in an interaction, failure of satisfaction from the part of one agent concerning his obligation in the transaction is interpreted as non-compliance to the protocol and can often lead to punishment, as it would be conceived under a sociological and legal point of view. In other words, commitments reflect the way with which the various agents i.e. application participants, interact and exchange resources in order to achieve their set of goals.

A commitment is represented by the tuple *C(Debtor, Creditor, Antecedent, Consequent)* which means that the debtor is committed to the creditor for the consequent if the antecedent holds. The antecedent and consequent are propositions relevant to the states of the world that can possibly occur within the application upon which the reasoning analysis is applied. For this, one commitment is discharged when the consequent is fulfiled and detached when the antecedent holds. In the case where the antecedent is always true by default, the commitment is considered to be unconditional. The term Service Engagement is also introduced and defines the protocol of communication between two roles that transact through one commitment. A role can be the debtor or the creditor in various commitments. An agent can adopt specific roles that transact through one or more commitments in order to fulfil its set of goals. Therefore, the roles in the model are variables over the agents and the service engagement maps all the possible commitments that can connect the agents of the application, when adopting certain roles. However, the existence of a commitment does not imply immediate satisfaction of the consequent for the agent adopting the role of the creditor. The fulfilment of a commitment is based on a run-time exchange of resources between the agents interacting through this commitment, and, therefore, whether an agent satisfies its part of the exchange is based only upon its own decision. From all the above, it is understood that an important part of commitment satisfaction in a service engagement, is the adoption of a certain role by a specific agent which is denoted as an engagement-bound agent. Such an agent should primarily check its compliance with one service engagement before adopting a role, i.e. whether it is capable of fulfiling its obligations in order to receive the resources desired to achieve its goals. In case of an incapability of doing so, the agent should seek for other service engagements that would enable this successful goal satisfaction.

Considering all the above, goal support for an agent x, as defined in Chopra et al. [7], would be checked by looking into whether the binding of that agent with the service engagement under consideration, by adopting a specific role r supports goal g included in the goal set of agent x. Thus, a goal g is supported if no other goal *g'* within the goal set of agent x contributes negatively to *g'*, if x is capable of g, if x is bound with some other agent playing role *r'* through a commitment *C(r', r, g, g)* and x supports *g'*, if x can commit to another agent playing role *r'* through a commitment of the form *C(r, r', g, g)* i.e. x commits to *g'* if the agent playing *r'* achieves g, if g is and-decomposed or or-decomposed and x supports all or at least one of the sub-goals respectively or, finally, if x intends to achieve another supported goal which contributes positively to g.

An example of such an extended AND/OR Goal Tree can be seen in Figure 2.2.

In Figure 2.2, the goals and sub-goals are depicted showing the conjunctive and disjunctive relationships between them. In addition, contributions existing between nodes are denoted by *++S*. The agents interacting are also stated along with the roles they adopt for a particular run-time execution session and three commitment from one agent to the other are given.

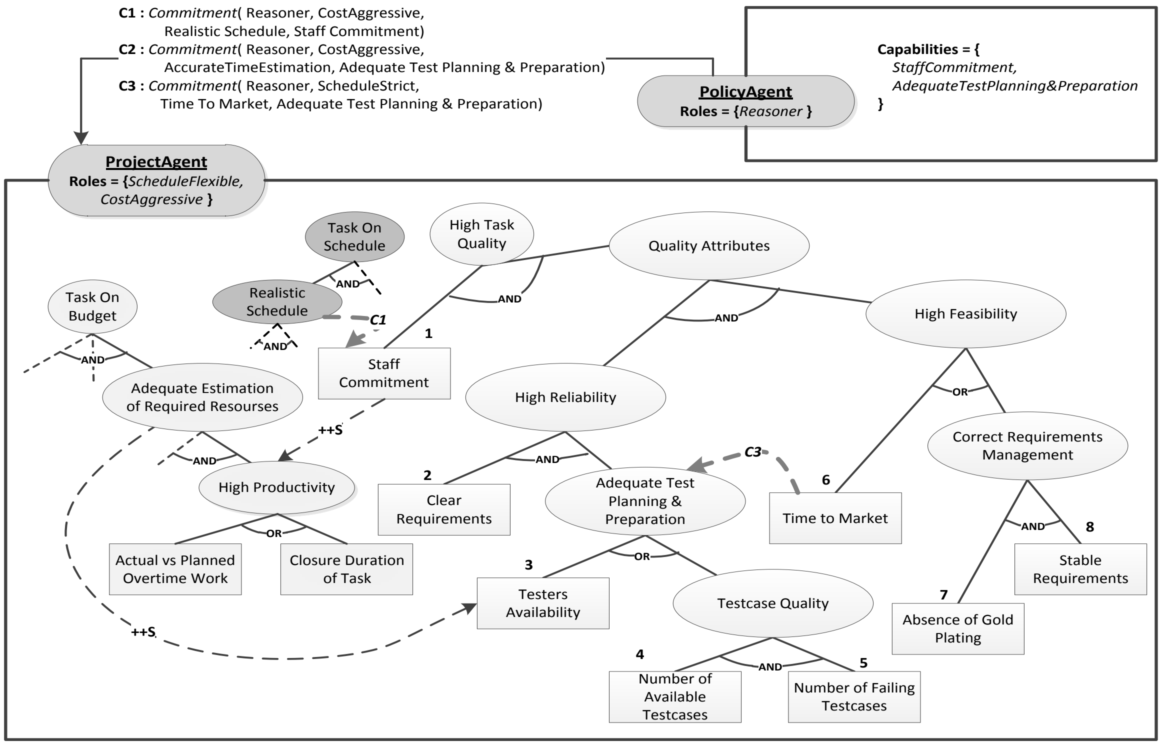

C1 : Commitment(Reasoner, CostAggressive, Realistic Schedule, Staff Commitment)
C2 : Commitment(Reasoner, CostAggressive, AccurateTimeEstimation, Adequate Test Planning & Preparation)
C3 : Commitment(Reasoner, ScheduleStrict, Time To Market, Adequate Test Planning & Preparation)
PolicyAgent
Roles = {Reasoner }
Capabilities = { StaffCommitment, AdequateTestPlanning&Preparation }
ProjectAgent
Roles = {ScheduleFlexible, CostAggressive }
Task On Schedule
High Task Quality
Quality Attributes
Task On Budget
Realistic Schedule
C1
High Feasibility
Staff Commitment
Adequate Estimation of Required Resourses
High Reliability
++S
Correct Requirements Management
High Productivity
Clear Requirements
Adequate Test Planning & Preparation
C3
Time to Market
Actual vs Planned Overtime Work
Closure Duration of Task
Testers Availability
Testcase Quality
Stable Requirements
Absence of Gold Plating
++S
Number of Available Testcases
Number of Failing Testcases
AND
OR
1
2
3
4
5
6
7
8

Figure 2.2: Run-time Goal Model Instance. Project agent adopts two roles in order to achieve its goals.

2.2 Mining Software Repositories

In a nutshell, techniques in mining and reasoning in software repositories are falling into five main areas [32, 1]. The first area deals with statistical and data mining analysis of repository data to uncover statistically significant correlations or interesting trends as the software system evolves. These types of analysis have been extensively used for evaluating software quality and for predicting software defects as well as, build success, code readability, and refactoring activities. In [65] data mining techniques are applied to revision history repositories to uncover dependencies between code segments that are difficult to extract with existing static and dynamic code analysis. In [25] Poisson modeling and generalized linear regression statistical analysis of change management data have been proposed as a way of predicting fault incidence in large long lived software systems. In [29] a time series based statistical analysis of CVS repository data is proposed to predict the number of changes that may occur in a software system within a time span following the analysis. In [43] and association mining technique is proposed for revealing rules related with defect correction effort prediction in medium size software systems.

The second area deals with NLP type of analysis and the use of topic models for search and clustering such as Latent Semantic Indexing (LSI), Probabilistic LSI (PLSI) and variants of the Latent Dirichlet Allocation (LDA). This type of analysis has been used for documenting traceability, concept/feature location through the analysis of informal information (i.e. identifier names, comments), the analysis of mailing lists and chat logs related to textual communication between developers, managers and other project stakeholders. This is used to discover trends in a software development process and for the analysis of software vulnerabilities. In [62] a comprehensive survey of topic model based techniques for mining software repositories is presented. The survey provides a classificatory and comparative study of the different approaches found in the literature. In [24] a text mining approach on natural language descriptions of bug reports is used to identify reports that are related to security issues. In [63] an LDA approach to mining software repositories is applied with the purpose of bug prediction and traceability link recovery. In [67] a PLSI technique is used for log reduction by filter event log data according to specific patterns and keywords, while in [27] a technique that is based on log reduction and Markov Logic diagnostic rules is used for root cause analysis. The main difference of this approach with the approach presented in this paper is that in [27] diagnostic rules are generated form plain AND/OR trees, while in this paper we present a framework that utilizes commitments, roles, and contributions for encoding Markov Logic Network rules from Goal Models. In [15] an informal information analysis technique combined with LSI is proposed for establishing traceability links among source code documents.

The third area deals with the extraction of metrics to compute maintainability indexes, identify code cloning, and predict software quality. In [35] an evaluation of different metrics computed from ten different software repositories as predictors of software quality is presented. The paper presents benchmarking results with respect to top software quality predictors. In [69] the authors analyze large software repositories in order to establish correlations between software metrics and pre and post release defects. In [44] software dependencies and measures of the amount of code change that is taking place within a software unit over time are analyzed as predictors of failure-proneness. In [60] the role of Chidamber and Kemerer suite of OO metrics in determining software defects is empirically evaluated. In [61] an approach combining static code analysis and process analysis with visual analytics is proposed to effectively answer maintenance questions and support decision making during project management. In [26] the architecture and principles of a standardized environment for storing software and project related information with the purpose of supporting the collection, analysis, and distribution of reliability and maintainability data across large software projects, is presented.

The fourth area deals with the analysis of software repositories using social network types of analysis. These types of analysis have been used to identify active project contributors and major project stakeholders as well as to identify correlations between social interactions among project members and software dependencies. In [5] social network techniques are applied on repositories of email correspondents in order to address question related to commit activities. In [22] a social network analysis is applied to reveal team communication patterns and assist on supporting management activities in software development projects. In [48] techniques of collaborative filtering and social networks are applied to open source project repositories to reveal technical relationships between developers and projects. Finally, in [40] social network analysis techniques are applied to development and commit activities activities as predictors of software post release failures.

Finally, the fifth area deals with machine learning where predictions on specific software properties can be inferred by past data trends. These types of techniques have been used to predict successful builds or determine access control policies and software process recovery. More specifically, in [13] a technique that uses association rule mining and the k-Nearest-Neighbor machine learning strategy to generate product specific feature recommendations by mining descriptions of publicly available product specifications is presented. In [34], the authors discuss a technique for predicting latent software bugs that uses a machine learning classifier for determining whether a new software change is more similar to prior buggy changes or to clean changes. The approach is able to classify with a relatively high degree of recall buggy and clean changes. In [3] a machine learning approach is used to identify program properties that are more likely than average to indicate errors in a program. The approach is used to reveal latent software errors.

Most of the approaches discussed above for mining software and project data repositories, relate to knowledge discovery and relationship identification. The work presented in this paper differs from existing approaches in two main points. The first point is that it treats the mining problem as a problem of validating specific goals as opposed of discovering relationships or calculating correlations and metrics. In this respect, in the proposed approach mining does not occur in vacuum but is based on specific analysis requirements (i.e. goals) the software engineer wants to validate in the repository data. This is also different from pattern matching approaches as goal models allow for much richer objectives to be specified than pattern expressions and pattern languages. The second point is that the proposed approach allows for mining even when incomplete or uncertain information has been stored in the repository. This is particularly important for repositories pertaining to open source projects where repository entries may include inconsistencies or ambiguous descriptions due to the large number of contributors involved.

Chapter 3

System Architecture

3.1 Architecture Description

3.1.1 General Overview

In this chapter, we present the architecture of the proposed framework.

In the field of Software Architecture, it is often the case that the best suitable architecture, especially when referred to large-scale software systems, is not always the one that conforms to a single particular architectural style but rather to a combination of different styles forming, thus, a heterogeneous architecture. It is common that in large software systems numerous different architectural styles are combined in order to achieve optimal results regarding the cohesion, coupling and modularity properties of the product under development. These combined styles are referred to as heterogeneous architectures. In an environment where functional and non-functional requirements are becoming more complex, the need for developing flexible and adaptable systems that meet customers' needs in a competitive time to market schedule while simultaneously maintaining a high level of quality, rises. For this, heterogeneous architectural styles are considered to be a reasonable choice since it allows the architect to incorporate within the software product, all the advantages of the styles used and, at the same time, accommodate the disadvantages of the styles through their composition.

The proposed system is based primarily on a BlackBoard style architecture through a Publish/Subscribe protocol with elements of Active DB, Layered and Multi - tier architecture styles. The modules of the system can be viewed at the following block diagram in Figure 3.1 :

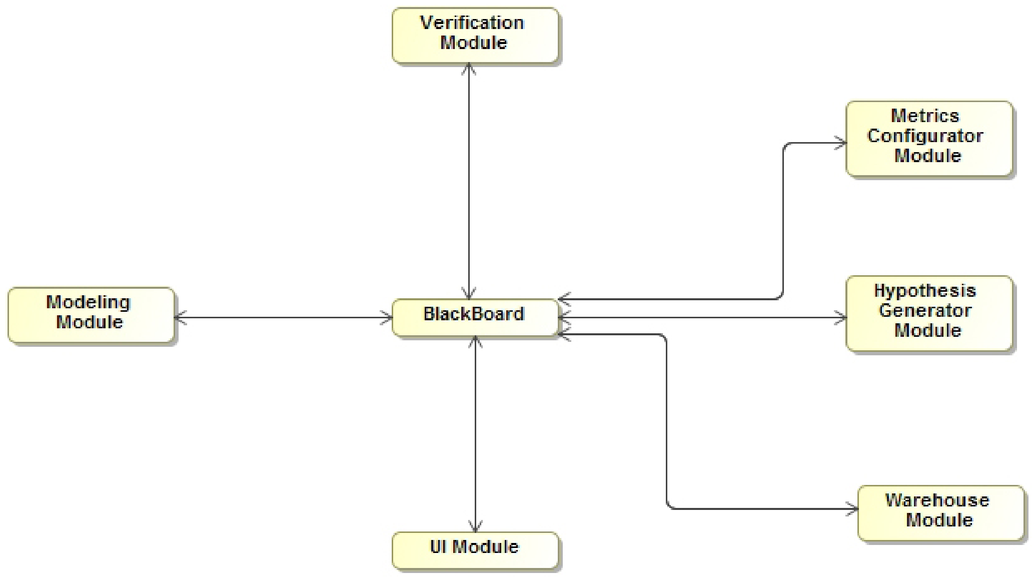


Figure 3.1: Block Diagram of the System Architecture

As depicted in the above diagram, the architecture consists of seven basic modules. A brief description of the functionality of those modules and their relationship is provided below :

BlackBoard : This is the main module which stores the state of the system at each milestone of its operation i.e. whenever there are changes in the state, and also records all events generated by all modules. Its functionality concerns mostly receiving and sending notifications from and to all other modules of the system. That is, whenever a module finishes an operation and declares a state change, it reports this to the BlackBoard where all changes are logged and published to all modules which subscribe to a particular change notification so as to initiate through implicit invocation their operations as well.

Warehouse Module : This is the module storing all the necessary data provided by IDEs, case tools, Project Management tools and any software project, completed or under development, which are required for the reasoning analysis and fault prediction process of the framework. It consists of a *Warehouse Database* containing all the information for the aforementioned projects, to be extracted when needed, a *Mediator*, an *Extractor*, a *Selector* and a *Data Pool* which contains the extracted data from the Warehouse Database.

Modeler Module : This is the module which initiates the construction of the goal model for the hypotheses. That is, the Modeler module requests information from the user for each hypothesis regarding to the goals, sub-goals and their inter-dependencies and relationships i.e. whether they are AND/OR decomposed or atomic, whether they contribute to any other, whether there exists any commitment implicating them with other goals and agents e.t.c. This module is also useful in order to allow the user to insert the necessary metrics and evaluation criteria for the previously defined goals. For these metrics, the user is requested to submit appropriate thresholds and acceptable values so that the system will have the ability to interpret the data values deriving from the database, as accepted or denied according to the user's specifications and requirements. For this, the module consists of an *Editor* which provides the user the means to edit the goal model, the *Hypothesis Modeler* which adjusts the model parameters according to the user's choices, i.e. the goals, contributions, roles and commitments are inserted, and the *Metrics Modeler* which provides metric and threshold insertion support.

Hypothesis Generator Module : This is the module which generates the various hypotheses to be checked for validation. The hypothesis generator module represents each hypothesis based on an AND/OR goal tree model extended through the definition of the notions of contributions, agents, roles and commitments. It consists of the *Hypothesis Generator* which generates the goal tree model and the *Hypothesis DB Server* which plays the role of the *Hypothesis Pool*, that is, it contains all the hypothesis scenaria that can hold as a set of goals for achievement.

Metrics Configurator Module : This is module which handles the evaluation criteria inserted for the various goals i.e. the metrics defined by the user, concerning the goals within the goal model. It consists of the *Metrics Generator* which transforms the inserted metrics into a form suitable for their further application, that is, into the form of appropriate SQL queries. It also consists of the *Validator* which has as its main functionality to examine whether the SQL queries produced are actually mapped to the database, that is, whether they query existing features. Finally, it consists of the *Metrics DB* which plays the role of a *Metrics Pool* storing all metrics generated as well as their thresholds and their corresponding goals for achievement, in order to be retrieved during future execution sessions of the system when the same goal model is invoked for the analysis.

Verification Module : This is the module which runs the verification procedure over the produced goal trees. It consists of a *Verification Pool* which contains all the possible verification strategies that can be imposed i.e. first - order logic, MLNs, e.t.c., a *Subscriber* which receives the generated goal trees, a *Publisher* which publishes the subscription, and a *Controller* which ensures the correct synchronization between the various components of the verification module described, as well as their communication with the *BlackBoard*.

UI Module : This is the module which handles and controls the communication between the user and the system. It consists of a *User Interface* through which the user is prompted to insert the required information to the Modeler regarding the goal model and evaluation specifications preferred, so as to set the configuration profile of the analysis. It also allows the user to store the results of the analysis held under a specific set of parameters for future reference. It also consists of a *Persistance Storage DB* in which these results are stored.

3.1.2 Component Description

In this section, we provide an overview of the architecture in a component level of granularity, which we discuss in more detail in terms of their functionality, provided interfaces and offered services. The component diagram of the system is illustrated in Figure 3.2.

Warehouse Module

The *Warehouse Module* is represented by the *Warehouse component* in Figure 3.2. As it can be seen, the Warehouse component is composite and contains the *Database*, the *Mediator*, the *Selector*, the *Extractor* and the *Data Pool*. The interfaces and services provided by the composite component and its components are explained below :

Interfaces/Services :

supplyData() : The Database contains information concerning a number of IDEs and software projects which can be mined and used towards the execution of the reasoning analysis. The Database is populated by the developers with project management and evaluation information recorded throughout the entire life cycle of the project, when the latter is viewed through various perspectives, such as its cost, the time schedule set for it e.t.c. Therefore, this information can be of different types, which calls for the Mediator to provide this interface to the Database in order to communicate and retract the information that needs to be mediated successfully. Therefore, the interface *supplyData* is used by the Database to supply the Mediator with the appropriate data.

mediateData() : The Mediator receives information provided by the Database in order to mediate it to the Extractor. The latter is assigned to extract out of the raw data deriving from the Database, those that can be of use towards the reasoning analysis. This interface is provided by the Mediator exactly to satisfy this need of data transition to the Extractor and, therefore, enables the correct reception of the appropriate information by the Extractor for further process.

transferData() : The Selector is used to distinguish through the sum of data sent by the Extractor, the subset which contains information relevant to the user's specifications set for the particular execution session, that is, depending on the goal model the user wishes to run the analysis for, and the set of metrics applied to it. This interface is provided by the Extractor in order to permit this selection of the data required, in order to serve as evidence for the specific parameters set by the user for this run-time session.

insertData() : After the appropriate data selection, the Selector stores them into the Data Pool from where the information becomes visible to the external environment of the Warehouse component i.e. the other components of the system. This interface provided by the Selector allows this data transfer to the Data Pool.

validateData() : This interface is provided by the Data Pool and useful to the Validator subcomponent of the Metrics Configurator component. Through this, the Validator is allowed to request all the information contributing to the value calculation of the metrics to be used as parameters for the analysis session, as set by the user, in order to ensure that the corresponding SQL queries involve existing features of the Database. This is a very important step of the process since run-time execution failures of the system can be avoided.

Concerning the architecture of the Warehouse Module, it needs to be noted that this part of the system is based on a 3 - tier architectural style.
The Database plays the role of the back - end of the 3 - tier architecture as illustrated in Figure 3.2. Therefore, being the back - end of a multi - tier architecture, it has the benefits of providing centralized data management, ensuring data integrity, security and consistency, and enabling concurrent operations of simultaneous requests from various clients.

The offered interface *provideData* which will be explained in detail under the BlackBoard subsection, plays the role of the front - end service of the 3 - tier architecture since it ensures that the communication of the back - end of the 3 - tier architecture (Database) with the component environment, that is, the other components of the system, will be held in a consistent and correct manner. This interface, being the front - end of the 3 - tier architecture, has the benefits of flexibility in customization and that of simple front - end processing of the data sent by the back - end as a result of a client request.

Modeling Module

The *Modeling Module* consists of the *Modeler component* in Figure 3.2. The Modeler component is composed by the *Hypothesis Modeler* component, the *Metrics Modeler* component and the *Editor* component. The interfaces and services provided by this module, are :

Interfaces/Services :

editHypoModel() : The Modeler establishes communication with the Editor through the dependency shown in Figure 3.2, in order to accept the goal model selection and edits for the session, among those stored and proposed by the system, or the goal model definition, if none of the already existent is chosen. More specifically, the definition of new and the alteration on existent goal models includes not only the goals, sub-goals and their relationships, but also the agents interacting, their set of goals towards achievement, the roles they can adopt to fulfil this set of goals, the set of capabilities each agent owns, the contributions appearing between goals and the commitments relating roles and goals. It can also receive all the user's preferences concerning goal versus metric correspondence and indicative values for each such evaluation metric tied to the hypotheses chosen for validation. That is, the addition of new metrics to the system, which can correspond to a potential new set of goals defined by the user, or even update already existing metrics corresponding to previously defined goals within the Hypothesis DB Server. Thus, the Modeler Module provides an interface to the User Interface of the UI Module in order to allow the user through the User Interface, to utilize the Editor in order to set all the configurations and parameters required for the analysis to take place successfully.

storeHypo() : The Hypothesis Modeler communicates through this interface with the Hypothesis DB Server in order to retrieve information on previously stored goal models representing various hypotheses, and update the database with new goal trees defined by the user or with potential modifications imposed to the existing, by the user.

sendMetrics() : The Metrics Modeler communicates through this interface with the Validator subcomponent of the Metrics Configurator component, in order to provide the latter with the metrics and their threshold values so that it can verify the accordance and compliance of the SQL queries featuring the metrics, with the database schema in use.

Concerning the architecture style of the Modeling Module, it is based on a Layered architecture style. The layers of the Modeling Module are shown in Figure 3.3 below :

This choice of architectural style offers the advantages of enabling the segmentation of the module in services belonging to different levels of abstraction and permitting the easier maintenance of the module.

Hypothesis Generator Module

The *Hypothesis Generator Module* consists of the *Hypothesis Generator component* as illustrated in Figure 3.2. The Hypothesis Generator component is a composite component and contains the *HypoGen* component and the *Hypothesis DB Server* component. The interfaces and services provided by those components, are explained below :

Interfaces/Services :

loadHypothesis() : As discussed above, the HypoGen component is the main component that generates the hypothesis in the form of a goal model consisting of AND/OR goal trees extended with the notions of contributions and commitments, upon which the analysis will be applied during the current session. The *Hypothesis DB Server* is the database containing all possible effective hypotheses, that need be utilized for the reasoning purposes. This interface is provided by the Hypothesis DB Server in order to communicate with the HypoGen component and supply the latter with hypotheses information necessary for the construction of the goal trees.

genHypothesis() : After constructing the goal trees corresponding to the goal model in effect complying to the user's requirements as set in the initialization of the session, the Hypothesis Generator Module is assigned to notify the Controller of the Verification Module about the event of the goal model construction completion so as to trigger the verification process. This interface offered by the Hypothesis Generator Module implements the interaction between the Hypothesis Generator Module and the Controller of the Verification Module in order for the generated goal model to be transferred to the Verification Module for the main probabilistic inference.

The architecture style on which this module's design is based on, is the Active DB style. An active database extends the idea of a regular database through including active rules and rule processing capabilities, thus providing a flexible and efficient mechanism useful to a large number of applications such as knowledge - based systems which is the case here. The Hypothesis DB Server plays the role of the active database which encapsulates active rules and constraints for the hypothesis properties and features. The HypoGen component communicates with it in order to extract the necessary rules and constraints of the hypothesis it processes during each active validation session. Therefore, it can be seen that the Active DB architecture style design is the most viable choice.

UI Module

The *UI Module* consists of the *UI component* and the *Persistance Storage DB* as they are depicted in Figure 3.2. The interfaces and services provided by these components, are the following :

Interfaces/Services :

applyAnalysis(): Through this interface provided by the Controller component of the Verification Module, the user is permitted to set the desired verification strategy to be applied during the analysis. Different verification techniques can include first - order logic resolutions, MLNs, Dempster - Shafer e.t.c. For the purposes of this thesis, we focus on the use of the Markov Logic Networks theory. The user must, also, specify some configuration settings essential for the verification process, such as threshold and acceptable values for properties and features tied to the hypothesis to be validated. As soon as the user defines the preferred verification strategy and configurations via the UI, the Controller is notified of this event.

connect() : –loadFromDB –storeToDB :

loadFromDB(): This operation of the interface is used by the BlackBoard to load data from the Persistance Storage DB. The Persistance Storage DB is a database used to store the results of verification simulations already executed for specific hypotheses and with certain parameters in order to avoid re - execution of identical simulations. Thus, the BlackBoard uses the interface to answer to clients' requests for retrieval of results of previously ran analyses, by loading the data from the Persistance Storage DB.

storeToDB(): This operation of the interface is used by the BlackBoard to store data to the Persistance Storage DB. After the conclusion of a verification process, the results are stored by the BlackBoard to the Persistance Storage DB in order to populate a log of previously executed verification analyses for the user to refer to before runnning a new analysis in case it has already been executed.

The UI Module is, as well, based on the Layered architecture style design. The layers of its architecture are shown in Figure 3.4.

This choice of architecture style denotes the module in different levels of abstraction enabling thus a more flexible decomposition of services across layers and, therefore, results in a more extensible and maintainable system.

Verification Module

The *Verification Module* consists of the *Controller component*, the *PubSub component* and the *Verification component*. The *PubSub component* is composite and consists of the *Publisher* and *Subscriber* components. The *Verification component* is also composite and consists of a number of *Verifier components*. For example in Figure 3.2 three different verifier components are depicted. The interfaces and services provided by these components, are :

Interfaces/Services :

notify() : After it has received the user's request on which verification strategy to apply during the validation process, the Controller communicates with the Subscriber through this interface in order to pass the verification strategy request so as to be set accordingly.

subscribe() : The Subscriber receives the user's requested strategy to be applied from the Controller and, therefore, is assigned to tie the appropriate Verifier component from the verification pool of the Verification component supported by this interface.

publishSub() : The Subscriber publishes back to the Controller the successful binding of the appropriate Verifier component. This event notification schema is possible through the existence of this interface.

verify() : The Controller is, also, responsible to transfer the command to the Verification component to commence the verification process after all the initializations and bindings have been successfully concluded through the verify interface.

publish() : –State –Notific. –Data : The Publisher component at each phase of the process, publishes through this interface, to the Controller the state of the verification procedure, any event notifications on results as well as the actual results of the analysis when the latter is concluded.

The design of the Verification Module is based on the Publish/Subscribe architecture style. This choice has the advantage of lack of dependencies between the Publisher and the Subscriber since both do not need to be aware of each other's existence in order to operate. Therefore, the scalability of the module can be increased since a certain amount of parallelization can be added to the verification process.

Metrics Configurator Module

The *Metrics Configurator Module* consists of the *Metrics Configurator* component as illustrated in Figure 3.2. The Metrics Configurator component is a composite component and contains the *MetricsGen* component, the *Validator* component and the *Metrics DB Server* component. The interfaces and services provided by those components, are explained below :

Interfaces/Services :

storeMetrics() : The MetricsGen component is the main component that transforms the metrics and thresholds inserted by the user in a form suitable for their utilization in order to cross-examine the data extracted from the Database and the goal model under analysis complying to the user's requirements. The *Metrics DB Server* is the database containing all possible effective metrics along with their corresponding thresholds, that can be applied towards reasoning, in order for the system to keep a record of possible execution parameters and, thus, allow the developer to repeat simulations or reuse past configurations on different goal models and/or different projects. All the aforementioned are the target of this interface.

genMetrics() : This interface is assigned to trigger the metrics and thresholds transformation into a form suitable towards proceeding with the validation of the hypothesis in question, after the Validator has cross-checked the validity of those evaluation criteria towards the Database i.e. after it has been ensured that the database features implicated in all metrics calculations are existent in the Database under scope.

The architectural style on which this module's design finds ground, is again the Active DB style. The Metrics DB Server plays a role similar to the Hypothesis DB Server which is that of the active database. It contains active rules and constraints for the metrics and their corresponding thresholds. The BlackBoard component communicates with it in order to receive the necessary rules and constraints of the queries representing the metrics and the appropriate thresholds for each, in order to redistribute them accordingly in each active session. Therefore, it can be seen that the Active DB architecture style design is the most viable choice.

BlackBoard

The *BlackBoard* is an individual component and plays a very important role in the system's operation. The interfaces provided or used by the BlackBoard, are :

Interfaces/Services :

provideData() : This interface connects the BlackBoard with the Data Pool so as to retrieve the necessary data for the validation of the hypothesis of the current session. The BlackBoard, afterwards, is assigned to forward these data to the other components which will be processing the data.

send() : –Data –State : Via this interface, the Publisher publishes supplementary data occurring during the analysis along with the state of the process, to the BlackBoard in order for the latter to notify other components and to synchronize the various modules.

update() : –Data –Hypoth. –State : Apart from this interface controlling the verification procedure, the Controller is also assigned through this interface to update the BlackBoard with the resulting data, the hypothesis tied to the resulting data sent, and the state of the process.

provideMetrics() : This interface connects the BlackBoard with the Metrics Configurator so that the BlackBoard is notified when the metrics generation has been concluded in order for the verification to proceed. The metrics and thresholds are now in the appropriate form and available through the BlackBoard to be forwarded to and retrieved by the Publisher component and, consequently, by the Verification component, so that the reasoning analysis can be concluded.

This is the core component that implements the BlackBoard architecture style of the proposed system. According to the BlackBoard architecture style ([31]), the BlackBoard component is an information base updated iteratively and incrementally by a set of various specialist knowledge sources, starting with a problem definition and ending with a solution. In our paradigm, the role of the information sources is played by the participating tools and verifier modules of the system. The BlackBoard is updated denoting a partial solution and, therefore, enables the synchronization of the modules of the system in order to make possible a seamless collaboration towards devising a solution.

3.1.3 Sequence Diagram of the Functionality of the System

As it can be understood from the above, the system is complex and contains many different components which have various operations and attributes. This fact raises the complexity level of the functionality of the system. A main *Sequence Diagram* of the overall functionality of the system, can be seen in Figure 3.5.

Therefore, the functionality of the system can be segmented to the following steps :

1. *editHypothesis* : Through the User Interface of the UI Module the user/developer can edit the goal model of already existing simulations through the Hypothesis Modeler, in order to re-execute the simulation over a different goal model.
2. *editMetrics* : Through the User Interface of the UI Module the user/developer can edit the parameters and metrics used in already existing simulations through the Metrics Modeler, in order to re-execute the simulation with different parameters and metrics.
3. *validateMetrics* : The Metrics Modeler requests from the Validator the validation of the inserted metrics, i.e. whether they correspond to valid features of the database, so that attempting to calculate them will return a value.
4. *storeHypothesis* : After edits have been made to existing simulations by the user, the Hypothesis DB Server should be updated with the new data and changes. Thus, the Hypothesis Modeler is assigned to update the hypotheses stored into the Hypothesis DB Server.
5. *confirmMetrics* : The Validator requests the appropriate data from the Database in order to confirm the validity of the metrics inserted.
6. *returnConfirm* : The Database responds to the Validator by returning the data requested, if existent.
7. *storeMetrics* : The Validator stores the metrics into the Metrics DB Server.

8. *updateMetrics* : The Metrics DB Server updates the BlackBoard with the new metrics that were just inserted.

9. *applyAnalysis* : Through the User Interface of the UI Module the user/developer has the ability to apply the preferred analysis parameters to the Controller in order to initialize the verification simulation with his/her own configurations and settings according to his/her demands. The user can, thus, select the hypothesis to be tested, apply the desired verification strategy to be followed, that is MLNs, first - order logic, OCL e.t.c., as well as set the acceptable values and thersholds for the features connected to the hypothesis selected.

10. *createGoalTree* : When it receives the simulation parameters, the Controller notifies the Hypothesis Generator in order to start the creation of the appropriate Goal Tree according to the configurations set by the user.

11. *requestData* : In order to construct the Goal Tree, the Hypothesis Generator needs to retrieve the necessary data for the specific hypothesis type, i.e. the properties and features tied to this hypothesis type along with the AND/OR relationships between them, from the Hypothesis DB Server.

12. *receiveData* : This is where the data from the Hypothesis DB Server are returned to the Hypothesis Generator which requested them.

13. *createAnnotations* : At this step of the procedure, the Hypothesis Generator requests from the MetricsGen to create and add the appropriate annotations at each node of the goal tree.

14. *returnAnnotations* : The MetricsGen returns the annotated goal trees back to the Hypothesis Generator.

15. *returnGoalTree* : After it has concluded with the goal tree construction along with the definition of the node annotations, the Hypothesis Generator notifies the Controller about this completion.

16. *updateHypothesis* : At a next step, the Controller is assigned to update the BlackBoard over the constructed goal tree and the hypothesis.

17. *reqVerify* : The next step of the simulation is for the Controller to trigger the verification process. Therefore, the Controller notifies the Verifier that the system state is ready for the beginning of the verification procedure.

18. *reqData* : In order for the Verifier to execute the verification process, the appropriate Warehouse data need to be retrieved. Thus, the Verifier is obliged to request the necessary data from the Database.

19. *sendMetrics* : The BlackBoard sends the appropriate metric values and thresholds in order to compare towards the data within the Database and produce the corresponding predicates for the cases that hold.

20. *recvData* : After the Verifier's request, the Data Warehouse returns the data needed by the Verifier.

21. *recvVerify* : After the completion of the verification procedure, the Verifier updates the Controller with the results of the validation.

22. *updateResults* : The BlackBoard needs to be updated with the validation results right after they are concluded and transferred to the Controller.

23. *inferMLN* : The BlackBoard, then, uses the resulting predicates of the hypothesis generation and the verification process in order to set them as input to Alchemy which is the MLN tool that will infer the MLN.

24. *updateMLNResults* : The Alchemy tool must update the BlackBoard component with the resulting validation percentages of the MLN inference.

25. *displayResults* : At last, the BlackBoard formats the resulting percentages and conclusions and sends them to the User Interface in order for them to be displayed to the user/developer for reporting and action planning.

3.2 Summary

In this chapter, the overall architecture of the proposed framework was given. More specifically, the architectural styles, designs and patterns were presented initially through the component diagram of the system which was furthermore, described in detail in terms of the components included and their functionality, their provided and required interfaces and their interaction with one to another. Briefly, the proposed architecture for the system is based on a BlackBoard architectural design using a Publish/Subscribe interaction protocol which enables all other components to subscribe in order to receive necessary information and/or publish their results to the BlackBoard component and, therefore, update the state of the entire system. Apart from the main BlackBoard core, the architecture also combines styles such as the ActiveDB and Layered Model in order to describe the structure of other sub-components and sub-modules of the system. This provided the proposed system with more flexibility through the incorporation of the advantages of numerous different architectural designs combined. Finally, the interaction process of the components described is presented in terms of a sequence diagram, which gives an overview of the overall procedure and the order of operation for all components.

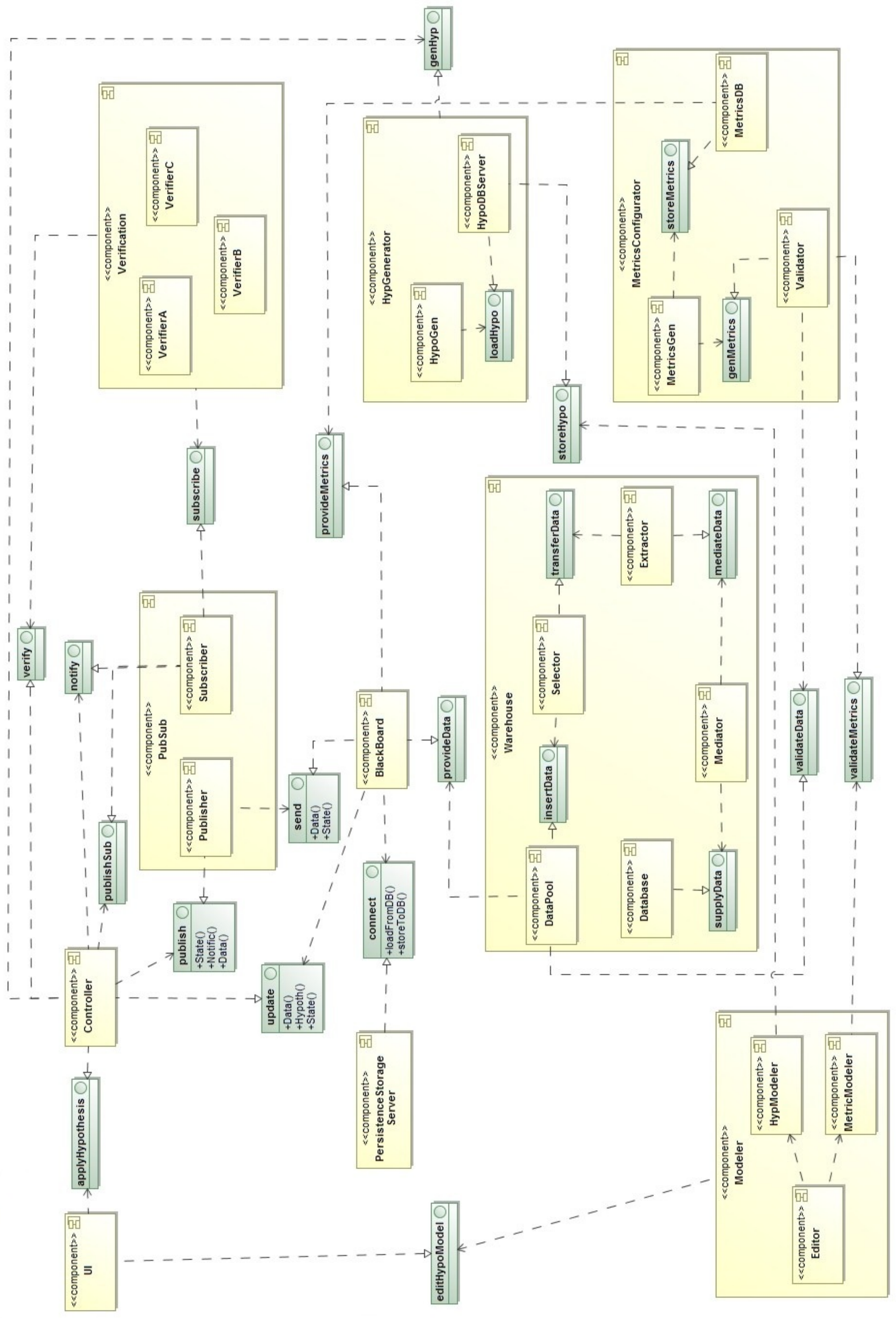


Figure 3.2: Component Diagram of the System Architecture

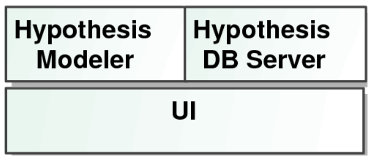


Figure 3.3: Layered architecture of Modeling Module

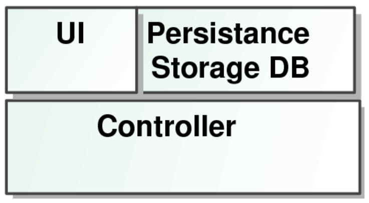


Figure 3.4: Layered architecture of UI Module

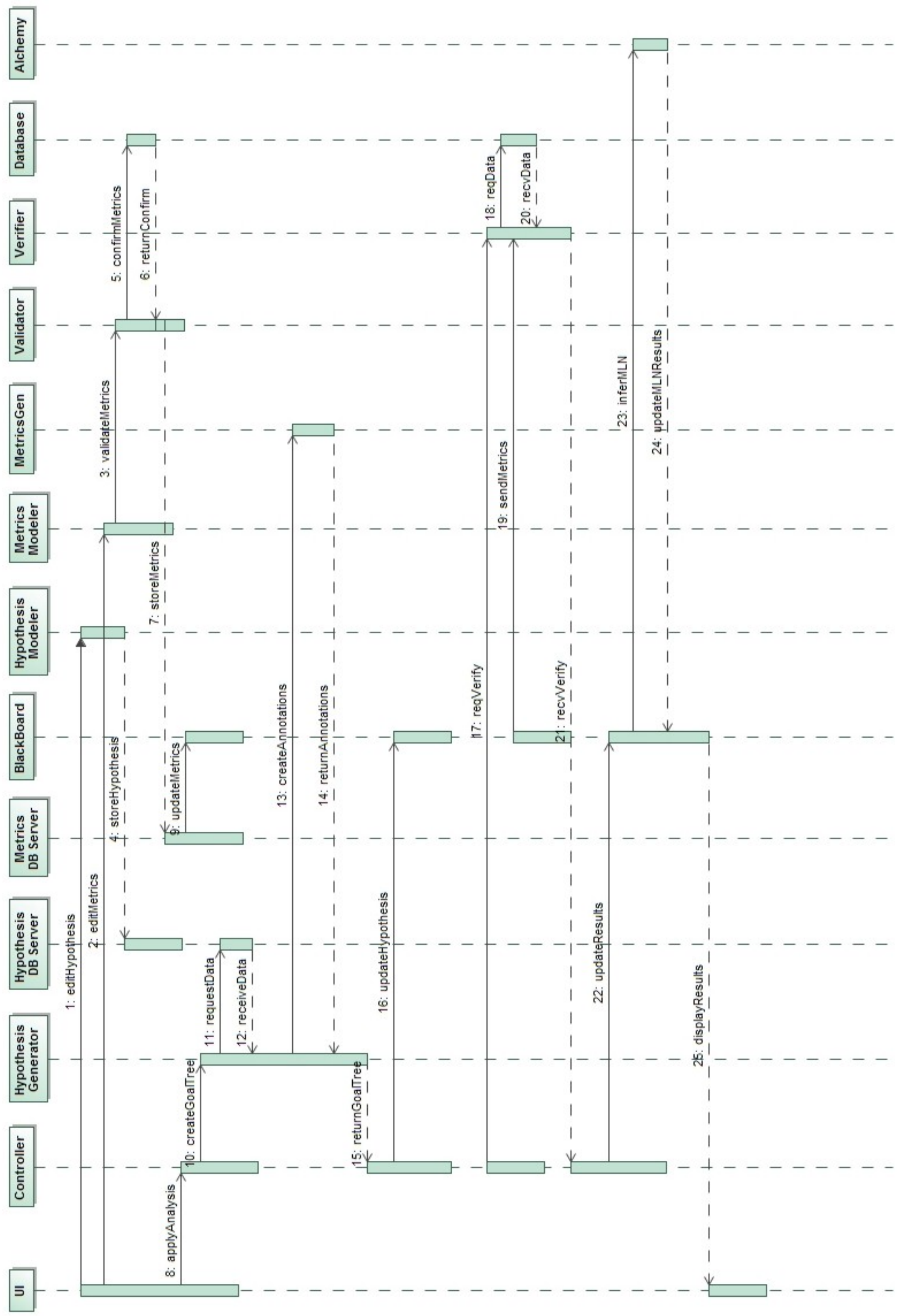


Figure 3.5: Functionality Sequence Diagram of the system

Chapter 4

Goal Model Specification

In this chapter, the specification for the Goal Domain Model developed for the purposes of this framework is being described in terms of MOF classes.

4.1 Goal Domain Model

The goal domain model built to meet the requirements of this framework, denoted as the Hypothesis Domain Model, is used to represent the different hypotheses in the form of AND/OR Goal Trees that we also extended with the notions of contributions and commitments, as discussed in chapter 2. This modeling structure was chosen due to its characteristics and the benefits it offers. More specifically, it provides flexibility regarding the means of modeling each hypothesis, which makes it easier to traverse and conduct the reasoning analysis. In addition, the AND/OR relationships that the nodes of the goal tree share by definition, permit the direct and straightforward interpretation and translation of the trees into first - order logic predicates, since the Goal Tree traversal aims to produce the first-order logic knowledge base, which is required to be provided to the Alchemy tool in order to construct the appropriate MLN for training and inference purposes.

An overview of the Hypothesis Domain Model can be seen in Figures 4.1 and 4.2 below.

Next, let us describe the Goal Domain Model and the classes included in the model.

4.1.1 HypoCategory Class

The *HypoCategory* class models the category in which each hypothesis falls into. The *HypoCategory* class has one attribute Name of type String that plays the role of the primary key, in order to discriminate each instance of the class i.e. each of the different categories defined and populated with hypothesis goal trees, for a specific system execution session. The *HypoCategory* class presents a composition relationship with the *HypoType* class. Thus, the class gains one more attribute from this relationship, the *+type* of type *HypoType* and multiplicity 1..*, which is expected since we have defined that each hypothesis category has various hypothesis types. Through the containment relations the class has with the *Agent* and *Role* classes, it gains the attributes *+agent* of type *Agent* and multiplicity 1..* and *+role* of type *Role* and multiplicity 1..* respectively, which allow the specification of the agents and roles existing and active during a certain session. It also owns an attribute deriving from its containment relation with the *Commitment* class, named *+commitment* of type *Commitment* and multiplicity 0..*, so that the set of commitments determining the specification protocol of the world of the application can be defined for the execution session.

In order to clarify the notion of hypothesis categories, an example of a hypothesis category would be Project Management Risks including hypothesis types such as Be On Budget, Be On Schedule and Maintain High Quality. Another example would be Source Code including hypothesis types such as Number of Lines of Code Correct, Number of Lines of Comments and Complexity of Source Code.

4.1.2 HypoType Class

The *HypoType* class represents the hypothesis type to which each hypothesis belongs to, provided that the specific type corresponds to the category under which the hypothesis is listed. The attribute Name of type String belonging to the *HypoType* class is used as a discriminator for each instance of the class and indicates the type of goal for which the analysis is executed. The *HypoType* class displays a composition association with the *HypoCategory* class, as mentioned above, which explains the existence of the attribute *+category* of type *HypoCategory* and multiplicity 1, since it is obligatory that every instance of the *HypoType* class is assigned to a sole instance of the *HypoCategory* class. The *HypoType* class also consists of many hypotheses via its association with the *Hypothesis* class with one-to-many correspondences. This infers the existence of the attribute *+hypothesis* of type *Hypothesis* and multiplicity 1..*. Each instance of the *HypoType* class contains only one goal tree, due to its composition association with the *GoalTree* class, a relationship which results directly from the fact that the system aims to verify or deny with a level of uncertainty, a hypothesis type through its representation using the notion of AND/OR Goal Trees. From this, it is clear that only one goal tree will correspond to one hypothesis type. In addition, this class owns an attribute *+agent* of type *Agent* and multiplicity 1, due to its association with the *Agent* class, in order to denote the fact that each hypothesis type i.e. each goal tree corresponds to a sole agent during run-time, which is responsible to achieve its root. Last, this class inherits from its association with the *ServiceEngagement* class an attribute *+service* of type *ServiceEngagement* and multiplicity 1..*, to reflect that some hypothesis type can refer or apply to various service engagements depending on their functionality and purpose.

4.1.3 Hypothesis Class

Each instance of the *Hypothesis* class models each hypothesis i.e. each path of any goal tree included into the hypothesis type and category, which are tested for validation through the system execution. It is differentiated from other hypotheses via the attribute *Name* of type String, which stands as a primary key for the class. Through its association with the *HypoType* class, as described above, it gains the attribute *+type* of type *HypoType* and multiplicity 1, which indicates the type that corresponds to the instance of the *Hypothesis* class. The *Hypothesis* class also has composition relationship with the *HypoPath* class, since each individual hypothesis is modeled through a specific path in some goal tree. This assigns one more attribute to the class, named *+hypopath* of type *HypoPath* and multiplicity 1, which denotes the hypothesis path that reflects the specific hypothesis. Finally, the class is associated with the *MLN* class via a one-to-one correspondence, since each hypothesis contributes to the construction of the complete MLN first-order logic knowledge base. Thus, the class obtains the attribute *+mln* of type *MLN* and multiplicity 1, in order to indicate this association.

4.1.4 HypoPath Class

The *HypoPath* class represents the hypothesis paths in a goal tree. More precisely, each goal tree root can be considered as satisfied, if one path of the goal tree leading to it is validated. Therefore, the goal tree root can be satisfied using any appropriate path in the goal tree. Each hypothesis path is interpreted into a certain goal tree and this explains the composition association shown in Figure 4.1 of this class with the *GoalTree* class, from which the attribute *+goaltree* of the class of type *GoalTree* and multiplicity 1 is derived. Each hypothesis path, also, contains one hypothesis and, therefore, the composition association of the class with the *Hypothesis* class is explained. The attribute of the *HypoPath* class gained by this association is named *+hypoth* and is of type *Hypothesis* and multiplicity 1. In addition, the *HypoPath* class has a composition association with the *CNFExpr* class, considering that each hypothesis path during the verification process finally has to be expressed into one conjuctive normal form, in order for the validation to be concluded. The association is manifested via the attribute *+cnf* of type *CNFExpr* and multiplicity 1. In addition, the *HypoPath* class has a one-to-many composition associations with the *GoalNode* class, through the attribute *+node* of type *GoalNode* and multiplicity 1..* which is expected, since, as it was mentioned earlier, each hypothesis path is a path within the corresponding goal tree and therefore consists of goal nodes. Finally, the *HypoPath* class is associated in one-to-one correspondence with the *Iterator* class, which adds the *+iterator* attribute of type *Iterator* and multiplicity 1.

4.1.5 Iterator Class

The *Iterator* class is a class which is helpful mostly to the implementation of the hypothesis verification algorithm. However, it can be seen that the class owns five operations: the *first()* operation that returns the first node of the object over which the iteration will be held, the *next()* operation that returns the next node to be traversed, the *currentnode()* operation that returns the current node that is being visited, the *isDone()* operation that returns a boolean value true or false, if the iteration has reached the end of the object type that is being iterated or not respectively. The Iterator class also owns an attribute under the name *+hypopath* of type *HypoPath* and multiplicity 1, in order to indicate the connection between the two classes.

4.1.6 CNFExpr Class

The *CNFExpr* class represents the conjunctive normal form, in which the hypothesis path is transformed in order for the verification to be held. This is the reason for the composition association between the two classes, which adds the attribute *+path* of type *HypoPath* and multiplicity 1 to the *CNFExpr* class. The *CNFExpr* class also presents a containment association with the *Logical Expression* class, which is expected, since according to the first - order logic theory, an expression in conjunctive normal form consists of logical expressions. The *CNFExpr* also has an attribute *+mln* of type *MLN* and multiplicity 1, due to the convert association with the *MLN* class, which is explained by the fact that each CNF expression raises the construction of one unique MLN. Finally, the class owns one operation named *convertToCNF*, which is assigned to make the conversion of the hypothesis path and its corresponding logical expression into a conjunctive normal form expression. The physical meaning of this class is tied to the representation of the internal component of Alchemy, which transforms the first-order logic predicates into CNF expressions.

4.1.7 MLN Class

The *MLN* class represents the MLN built by Alchemy, in order to validate the hypothesis. Thus, the class owns the operation *constructMLN*, which builds the appropriate MLN for the CNF expression it receives as input. Therefore, the class is associated with the *CNFExpr* class through the attribute *+cnf* of type *CNFExpr* and multiplicity 1. The class is also connected to the *Hypothesis* class, since it has already been mentioned that each instance of the *Hypothesis* class corresponds to an instance of the *HypoPath* class and, finally, is represented by a logical expression which is converted to CNF form. In addition to the previous, the class is tied to the *Verifier* class via the *+verifier* Type attribute of type Verifier and multiplicity 1, gained by the verify association. That is, each MLN is identified by one verifier that will use the MLN produced, in order to finish the verification procedure. For further clarification again, the physical meaning of this class is connected with the modeling of the internal component of Alchemy that produces the MLN and conducts the data training.

4.1.8 Verifier Class

The *Verifier* class represents the verifier which utilizes the MLN produced during the verification process, in order to conclude it. It is connected through the verify association with the *MLN* class via the attribute *+mln* of type *MLN* and multiplicity 1 and also owns an operation *verifyHyp()*, which runs the validation of the hypothesis given to it by the produced MLN. Physically, this class represents the component of Alchemy conducting the inference of the testing data.

4.1.9 GoalTree Class

The *GoalTree* class represents the goal tree structure which is the main structure used to model the hypothesis in a flexible way. This class has an attribute *idname* of type String that acts as a primary key to separate the various instances of the class. The *GoalTree* class also has a composition relationship with the *HypoType* class through the *+typeroot* attribute of type *HypoType* and multiplicity 1, which assigns the hypothesis type to be validated with the goal tree that was built. In addition, the class is associated with the *HypoPath* class via the *+hypopath* attribute of type *HypoPath* and multiplicity 1..*, which indicates that a goal tree can consist of many hypothesis paths and, more specifically, of an equal number to the leaves of the tree.

4.1.10 GoalNode Class

The *GoalNode* class represents the goal nodes of the AND/OR goal tree structure. The *GoalNode* class has two attributes, the *Name* of type String and the *ID* of type Integer. These are the names of the goal node which is also used for the construction and the secure primary key, to separate the different goal nodes and prevent any goal node name aliases within different goal trees, respectively. The *GoalNode* class demonstrates a composition association with the *GoalTree* class, which is shown through its attribute *+tree* of type *GoalTree* and multiplicity 1, that denotes the goal tree for which each goal node belongs to. The class has a supplementary attribute *+parent* of type *GoalNode* and multiplicity 1, which stores the parent node for each goal node. As mentioned earlier, each goal node is also part of some hypothesis path and thus, the attribute *+path* of type *HypoPath* and multiplicity 1 is explained. Furthermore, the class demonstrates an attribute *+visitorcreator* of type *VisitorCreator* and multiplicity 1, due to its association with the *VisitorCreator* class, which is responsible to create a visitor object, in order to process each goal node appropriately during the goal tree traversal. Finally, the *GoalNode* class is part of a composition relationship with the *Annotation Container* class, which adds the attribute *+annotContainer* of type *Annotation Container* and multiplicity 1 to the class, since each goal node is obliged to own an annotation container, in order to store the necessary information for the construction of the appropriate predicates. It is noted that, as it can be seen in Figure 4.1, the *GoalNode* class appears to have inheritance relationships with the classes *AtomicGoal* and *DecompositionGoal* that will be explained below. Also, the *GoalNode* class has an *+exprproducer* attribute of type *ExprProducer*, which is justified by the association of the class with the *ExprProducer* class which serves mostly system control purposes, as it will be explained below. Extending the class in order to cover contribution support, an attribute *+contribution* of type *GoalNode* and multiplicity 0..* is added through a self-association relation, to allow the definition of all the goal nodes to the achievement of which each goal node instance contributes. Furthermore, adding commitment support extensions, the class is connected with the Commitment class through two directed association relations reflecting the antecedent and consequent of each commitment, which are goals, and therefore modeled as goal nodes. Finally, this class also obtains an attribute *+capability* of type *Capability* and multiplicity 1, in order to denote the correspondence of a capability and a goal node, taking into consideration that an agent's capability is defined as a specific goal, which can be automatically satisfied by that agent.

4.1.11 AtomicGoal Class

The *AtomicGoal* class is a subclass of the *GoalNode* class and thus, inherits the properties and operations of the *GoalNode* super class apart from its own. The *AtomicGoal* class represents the atomic goals that can be met in a goal tree i.e. those goal nodes that cannot be further analyzed into simpler logical expressions. These are the leaves of the goal tree which do not have any children.

4.1.12 DecompositionGoal Class

The *DecompositionGoal* class is the second subclass of the *GoalNode* class and also inherits all its properties and operations. The *DecompositionGoal* class represents all the instances of the *GoalNode* class that are composite goals i.e. the goals that have children which present AND - relationships between one to another and the goals that have children which present OR relationships between one to another. This is the reason of the existence of the two subclasses of the *DecompositionGoal* class, the *ANDDecomposition* class and the *ORDecomposition* class. Their purpose is to discriminate the two cases of composite goal nodes and thus permit the individual and separate manipulation of each type of goal node.

4.1.13 ExprProducer Class

The *ExprProducer* class plays the most significant role in the implementation of the hypothesis verification algorithm. It is the main class that composes the BlackBoard component and thus, controls the validation process. The class owns two operations :

constructpath() : This operation is used in order to construct the paths of the goal tree that validate one common hypothesis type and traverse them, so as to conclude to a set of predicates for the first - order logic world that will be used to build the appropriate MLN.

attarchstrategy() : This operation is used in order to infer the attachment of the desired strategy to the process i.e. MLN, first - order logic e.t.c. and the extraction and interpretation of the necessary data deriving from the database that constitutes the Data Warehouse.

The class also presents an aggregation association with the *Strategy* class.

4.1.14 Strategy Class

The *Strategy* class represents the strategy to be attached to the procedure i.e. MLN, first order logic e.t.c. It owns one *verify()* procedure which runs the verification. The *Strategy* class has different *Verifier* classes as subclasses which connected to it through inheritance. In Figure 4.1, three can be seen as an example. The *VerifierA* class is the one corresponding to the MLN verification strategy for the purposes of this thesis.

4.1.15 VerifierA Class

The *VerifierA* class represents the MLN strategy. It is assigned to execute the comparison of the accepted thresholds deriving from the configuration of the simulation, and the metrics values calculated using the data deriving from the Data Warehouse. If the calculated values are within configuration boundaries, a predicate is constructed.

4.1.16 VisitorCreator Class

The *VisitorCreator* class is connected to the *GoalNode* class via the *+goalnode* attribute of type *GoalNode* and multiplicity 1, since one instance of the class is being assigned to each instance of the *GoalNode* class, in order to create the appropriate type of instance of the *Visitor* class among its three subclasses. The *VisitorCreator* class owns the *visitorFactoryMethod()* operation, in order to have the ability to create the appropriate type of *Visitor* for each node. In addition, the class has a dependency relationship visitor with the *Visitor* class. More specifically, each node, according to its type (Atomic, AND Decomposition, OR Decomposition), infers the instantiation of the corresponding subclass of *VisitorCreator* i.e. one of the *ANDVisitorCreator, ORVisitorCreator* and *AtomicVisitorCreator*. The three creator classes, according to their type, are assigned to construct one instance of the appropriate *Visitor*, that is one of its subclasses: *ANDVisitor, ORVisitor* or *AtomicVisitor*. It is necessary to note that the most important benefit of this design choice is that the client does not need to know any details of the type of goal nodes, visitor creators or visitors instantiated, which adds an extra level of flexibility to the implementation, since the traversal of the goal tree becomes a black box procedure.

4.1.17 Visitor Class

The *Visitor* class is dependent by the *VisitorCreator* class and instances of the first are created by the latter. According to the type of the creator that has been instantiated at any point in time, the corresponding visitor will be constructed and inferred. This is the reason explaining the existence of three subclasses of the *Visitor* class, *ANDVisitor, ORVisitor* and *AtomicVisitor*. The Visitor class is assigned an operation under the name visit(), which processes the goal node that is being traversed which also has constructed a creator resulting to the particular visitor. Since the three different types of goal nodes instruct specific manipulation, the operation visit() is individually implemented for each one and overrides the operation of the parent class.

4.1.18 AnnotationContainer Class

The *AnnotationContainer* class provides a container of annotations for each goal tree node for storing the necessary data to construct the appropriate predicates. The class presents a composition relationship with the *GoalNode* class through the *+goalnode* attribute of type *GoalNode* and multiplicity 1. The *AnnotationContainer* class presents also a composition relationship with the Annotation class and thus, obtains one more attribute under the name +annotation of type Annotation and multiplicity 1..*.

4.1.19 Annotation Class

The *Annotation* class represents the general means of attaching the necessary information for constructing the appropriate predicate for each goal node. The instances of the class are contained to one instance of the *AnnotationContainer* class, therefore, the class owns the attribute named *+annotcontainer* of type *AnnotationContainer* and multiplicity 1. The annotations can be of many types, but only one subcategory of them can be of assistance to the process. This is the subclass *Verifiable Annotation* of the *Annotation* class, since only those annotations that can be verified are of interest. A subclass of the *Verifiable Annotation* class is the *Logical Expression* class, since the only type of annotations needed during the goal tree traversal are the logical expressions. This class has one attribute named *+predicate* of type String, which stands for the predicate to be constructed. The *Logical Expression* class has two subclasses that inherit from it the *Simple Expression* class and the *Composite Expression* class. The *Composite Expression* class has an aggregation relationship contain with the *Logical Expression* class, since a composite logical expression consists of many logical expressions, simple or composite. Thus, the Composite Expression class has one attribute *+logicalExpr* of type *Logical Expression* and multiplicity 1..*. The *Annotation* class is defined in such a generic way in order to provide more flexibility to the system. More specifically, in order to extend the system with support of various types of annotations, it is enough for the developers to add more subclasses to it, without being required to know the exact internal structure of the class, and to implement the operations and functionalities of the class, so that they meet their demands by overriding the ones of the parent class.

4.1.20 Agent Class

The *Agent* class represents the notion of agents as described previously. That is, it models the participants of service-oriented application that interact with each other under the specifications of a certain protocol and are bound to fulfil transactions defined through the form of commitments. This class owns an inherent attribute *+Name* of type String, which is used so that the name of the agent can be specified and that instance of the class can be differentiated from others within the same run-time session. The *Agent* class receives an attribute through the composition relationship it maintains with the *Capability* class, called *+capability* of type *Capability* and multiplicity 1..*. This attribute is useful in order to aggregate all the capabilities for one specific agent and bind them to that particular instance of the class. The *Agent* class also owns an attribute *+role* of type *Role* and multiplicity 1..*, which supports the specification at each system execution of the roles each agent instance takes, in order to satisfy its set of goals. Other attributes inherited by the containment relation between the *Agent* class and the *HypoCategory* class and its association with the *HypoType* class respectively, are named *+category* of type *HypoCategory* and multiplicity 1 and *+hypotype* of type *HypoType* and multiplicity 1, in order to specify for each agent instantiation within a single session, the hypothesis category and hypothesis type it is destined to attempt achievement and therefore, determine which role it should take up to do so. The latter is determined through the inheritance of the attribute *+role* of type *Role* and multiplicity 1..*, from the association relation between the class and the *Role* class. Finally, the class also owns an attribute named *+capability* of type *Capability* and multiplicity 1..* deriving from its directed association relationship with the *Capability* class.

4.1.21 Role Class

The *Role* class models the roles that an agent can acquire, in order to achieve a certain set of goals within a run-time execution session. The class owns an intrinsic attribute *+Name* of type String denoting the name of each role instance, which plays the part of the primary key distinguishing the various roles from one to another. The class has an attribute *+category* of type *HypoCategory* and multiplicity 1, to determine under which hypothesis category this role obtains a meaning. The class also inherits the attribute *+agent* of type *Agent* and multiplicity 1 which ties each role to one sole agent depending on the configuration of the execution session and the specifications of the application under analysis. Other attributes acquired are *+commitment* and *+service* of type *Commitment* and multiplicity 1..* and of type *ServiceEngagement* and multiplicity 1 respectively, to allow the commitments and service engagements that the role instance takes part in, either as the creditor or as the debtor of some commitment.

4.1.22 ServiceEngagement Class

The *ServiceEngagement* class models the protocol of a service-oriented application under which the agents interact in order to fulfil their goals by undertaking certain roles. The class owns an attribute *+Name* of type String playing the part of the identification value of each instance. The class also obtains an attribute *+hypotype* of type *HypoType* and multiplicity 1, in order to indicate the hypothesis type connected with the interaction protocol under which the reasoning analysis will be held. Another attribute is named *+role* of type *Role* and multiplicity * and derives from the directed association of the class with the Role class, so that a list of the roles existing as active is contained within the scope of the service engagement. Finally, the attribute *+commitment* of type *Commitment* and multiplicity 1..* is inferred by the directed association between the *ServiceEngagement* and *Commitment* classes and is used to list the commitments existent in the protocol of each service engagement instance presiding in world of the application under analysis.

4.1.23 Capability class

The *Capability* class represents the modeling structure for the capabilities of the agents participating in the application and interacting through commitments towards achieving their sets of goals. This class again has a *+Name* attribute to distinguish each capability of an agent from the others, which is of type String. By definition, capabilities are characteristics of the agents and, thus, are independent from any roles an agent can undertake.

Therefore, the class is connected through a containment relationship with the *Agent* class which results to the existence of the attribute *+agent* of type *Agent* and multiplicity 1, since each capability belongs to a certain agent. Concluding, the attribute *+node* of type *GoalNode* and multiplicity 1, appearing as a member of the class, is useful in order to set the goal node to which one capability corresponds i.e. of which goal node the existence of some capability infers the achievement.

4.2 Project Management Goals

In order to be concurrent with recent tendencies in the field of Software Engineering and to meet the needs of project managers and developers, the reasoning analysis should reflect standard widely applied project management goals. Thus, the need of defining goal models representing these goals in an accurate as well as generic and easily extensible manner rises. Therefore, it becomes clear that goal trees are required to reflect actual realistic goals covering the main current concerns of project teams and managers when developing a new product. For this, it is crucial to search for measurement techniques and methods which are based on standards that are generally accepted and their accuracy is proven.

For the purpose of the reasoning analysis, we selected three types of general goals desired to be achieved by project managers and developers: **a)** High Task Quality, **b)** Task On Schedule and **c)** Task On Budget. These goals were chosen as representative ones from a list of key project objectives found in the literature [30, 52], and for which our data set was complete enough for experimentation purposes. We have also defined two agents, the *PolicyAgent* which can take the *Reasoner* role, and the *ProjectAgent* which can take any of the roles in the sets {*CostAggressive*, *CostConservative*} and {*ScheduleStrict*, *ScheduleFlexible*} or any combination constituted by one member of both sets. Let us note that this choice was exemplary i.e. the framework is constructed in a manner such that it provides the user with the ability of defining his/her own set of goals and customizing his/her own set of roles.

As it can be observed, the types of goals defined for the purposes of this study, refer to project tasks (i.e. they examine a project in a higher level of granularity). We chose to maintain this scope for our analysis due to the nature of the data available. More specifically, the data we had in our disposal, provided an adequate number of specimens regarding project tasks rather than projects as an entity. However, this selection is not considered to harm the generality of the analysis due to the fact that there exists a scaling analogy between project tasks and projects since a project is an aggregation of tasks.

The goal trees corresponding to each of the aforementioned goals, are depicted in Figure 4.3. More specifically in Figure 4.3, three different goals are depicted namely, *High Task Quality*, *Task On Budget* and *Task On Schedule*. Furthermore, there are two agents, *ProjectAgent* and *PolicyAgent*, which adopt for the specific execution session the roles *CostAgressive, ScheduleFlexible* and *Reasoner* respectively. The full list of commitments defined for the goal model shown in Figure 4.3 are given in Table 4.1. Thus, a subset of the commitments of Table 4.1 is denoted to show the interaction protocol between the two agents. A list of contributions is also depicted in Figure 4.3 denoted as *++S*. In order to conclude to these goal trees with AND/OR decompositions, contributions and commitments as seen, we conducted research over various risk factors that can potentially occur on software project development [30, 14]. The literature [30] has provided us with insight over which aspects of a project are the most fault prone, which of those the project managers and developers give most importance on, in which way e.t.c. The combination of sub-goals, as well as the type of relation AND/OR between them in order to achieve a parent goal, was also a matter thoroughly investigated. One also very important issue researched was the appropriate setting of contributions between goals and commitments among different agents. For the latter, not only did we have to research the types of goals that the fulfilment of one would reasonably infer the achievement of the other, but also which agents are participating in the transaction and, therefore, are able to support and potentially satisfy the above goals in order for the commitment to stand, according to the definition of the term.

The goal trees of Figure 4.3 are a direct instance of the domain model depicted in Figure 4.1. Each goal tree reflects to one instance of the *HypoType* class, which is tied to the corresponding instance of the *GoalTree* class. Each of these instances contains many instances of the *GoalNode* class, either as AtomicGoal instances or as *DecompositionGoal* instances i.e. *ANDDecomposition* or *ORDecomposition*. The two agents specified in the figure are also separate instances of the *Agent* class and the roles they adopt are again each one instance of the *Role* class. For the goals contributing to others, as depicted with the notation *++S*, the goal receiving the contribution is set as an attribute to the instance of the *GoalNode* class representing the contributing goal. This is shown in the domain model through the self-association *+contribution* that the *GoalNode* class owns. Finally, each of the commitments denoted is an instance of the *Commitment* class, which is associated with one goal as the antecedent, another goal as the consequent and two agents as the debtor and creditor respectively. This is represented in the domain model through the

Table 4.1: List of Commitments defined for the Goal Model.

Commitment Symbol	**Commitment Predicate**
C1	*Commitment*(Reasoner, CostAggressive, Realistic Schedule, Staff Commitment)
C2	*Commitment*(Reasoner, CostAggressive, Accurate Time Estimation, Adequate Test Planning & Preparation)
C3	*Commitment*(Reasoner, CostConservative, Accurate Cost Estimation, Limited Unjustified Expenses)
C4	*Commitment*(Reasoner, ScheduleFlexible, Time To Market, Adequate Test Planning & Preparation)
C5	*Commitment*(Reasoner, ScheduleStrict, Correct Requirements Management, Adequate Test Planning & Preparation)
C6	*Commitment*(Reasoner, CostConservative, True Node, Absence Of Gold Plating)
C7	*Commitment*(Reasoner, ScheduleStrict, True Node, Stable Requirements)

associations with the *GoalNode* and *Agent* that the *Commitment* class owns.

Taking into consideration that the reasoning analysis aims to verify the root node of a goal tree by traversing it beginning from level 0 i.e. the leaf nodes of the tree, and upwards, it can be understood that, in order to assign truth values to all sub-goals within the tree, we must first do so for the leaf nodes. Therefore, we must define some logistics that will allow this truth value assignment to the leaf nodes in a flexible but also formal manner. As a result, we sought for possible metrics that can be utilized to define a means to quantify and measure the risk factors, which would provide that desired formality and flexibility. During the search for such numeric relations, various appreciated and applied sets of standards were considered as potential candidates, such as the ISO 9126 [49, 50] or the CMMI [58] standards. However, a rather newer collection of such metrics, appreciating various different aspects of the development of a product known as the KPI library, was decided to be applied for the purposes of the study. The reasons for this choice, against a set of standards potentially more widely used and evaluated, was that the KPI library [42], being a more recent set of metrics, appeared to apply more to the current fashion followed in Risk Assessment and to provide metrics more precise and targeted as to the risk factor they represent. Therefore, based on the KPI library, we defined appropriate metrics so as to measure the factors described by the leaf nodes of the goal trees. Our choice provides the formality required since the KPI library refers to all the notions of modern risk analysis in Software Engineering, as well as the flexibility desired, since the metrics we suggest are not absolute i.e. the user is given the opportunity to determine different metrics suitable for his/her project risk evaluation session according to his/her demands and requirements.

4.3 Summary

In this chapter, an overview of the domain Goal Model was given in the form of a class diagram demonstrating MOF compliance. Furthermore, the classes included in the diagram as well as their attributes and operations were presented. The relationships they present from one to another were also described in order to further clarify the usability of the model and the benefits it provides to the purposes of the analysis, which were the reason for this design choice. In addition, the project management goals related to cost, schedule and quality risks, which were used as the exemplary goal set tested for validation, were presented and their relation with the KPI Library was explained. Finally, the relationship between the domain goal model and the project management goal trees is shown and explained i.e. how the latter constitute instances of the first.

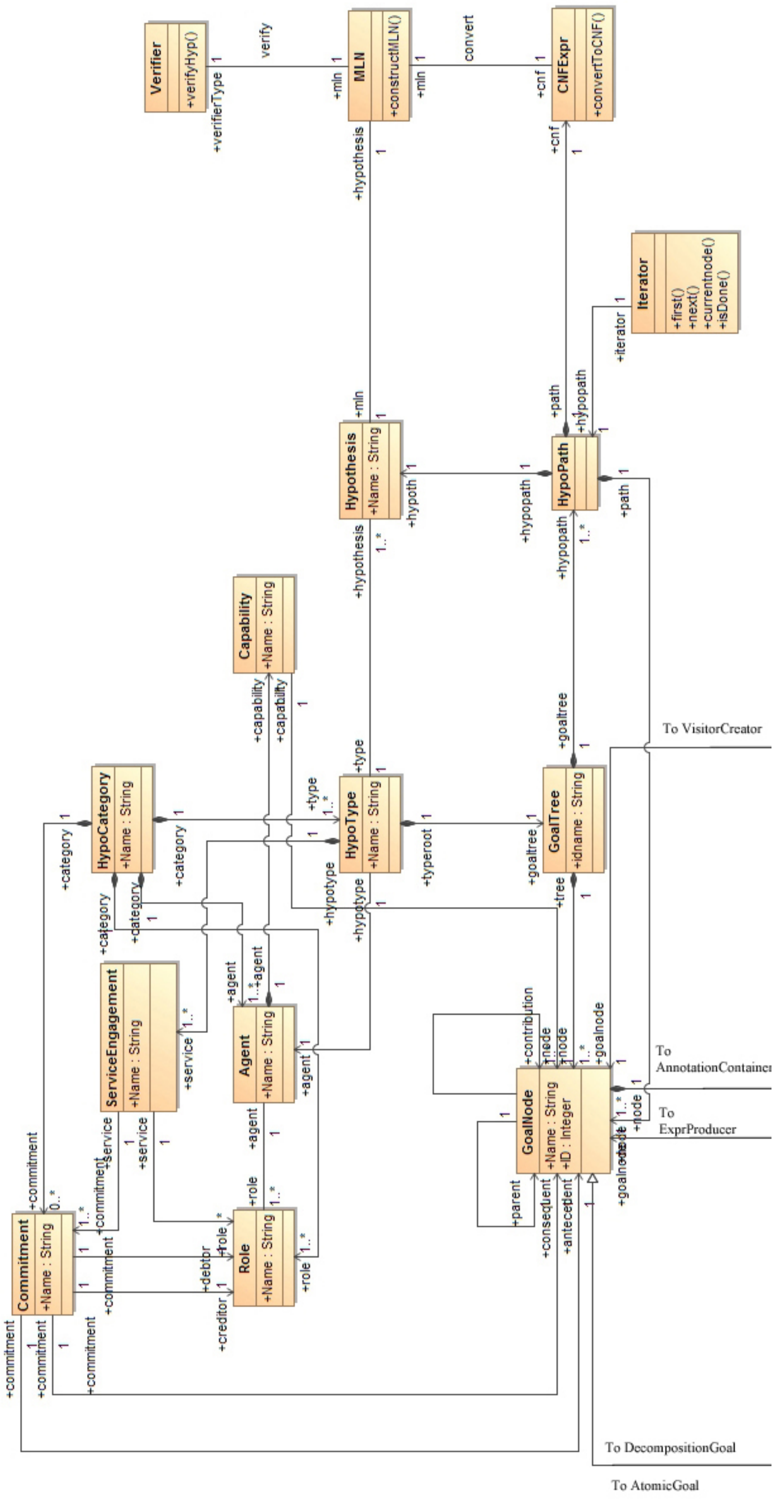


Figure 4.1: Hypothesis Goal Domain Model

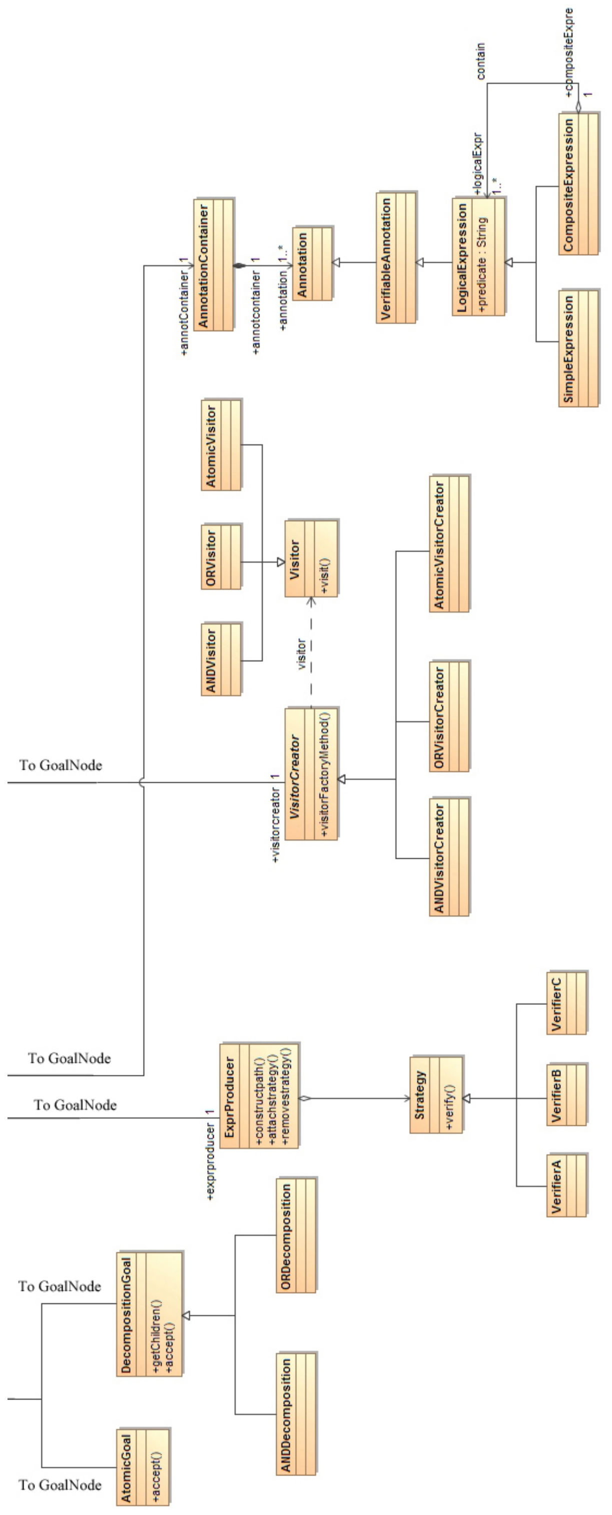


Figure 4.2: Hypothesis Goal Domain Model (cont'd)

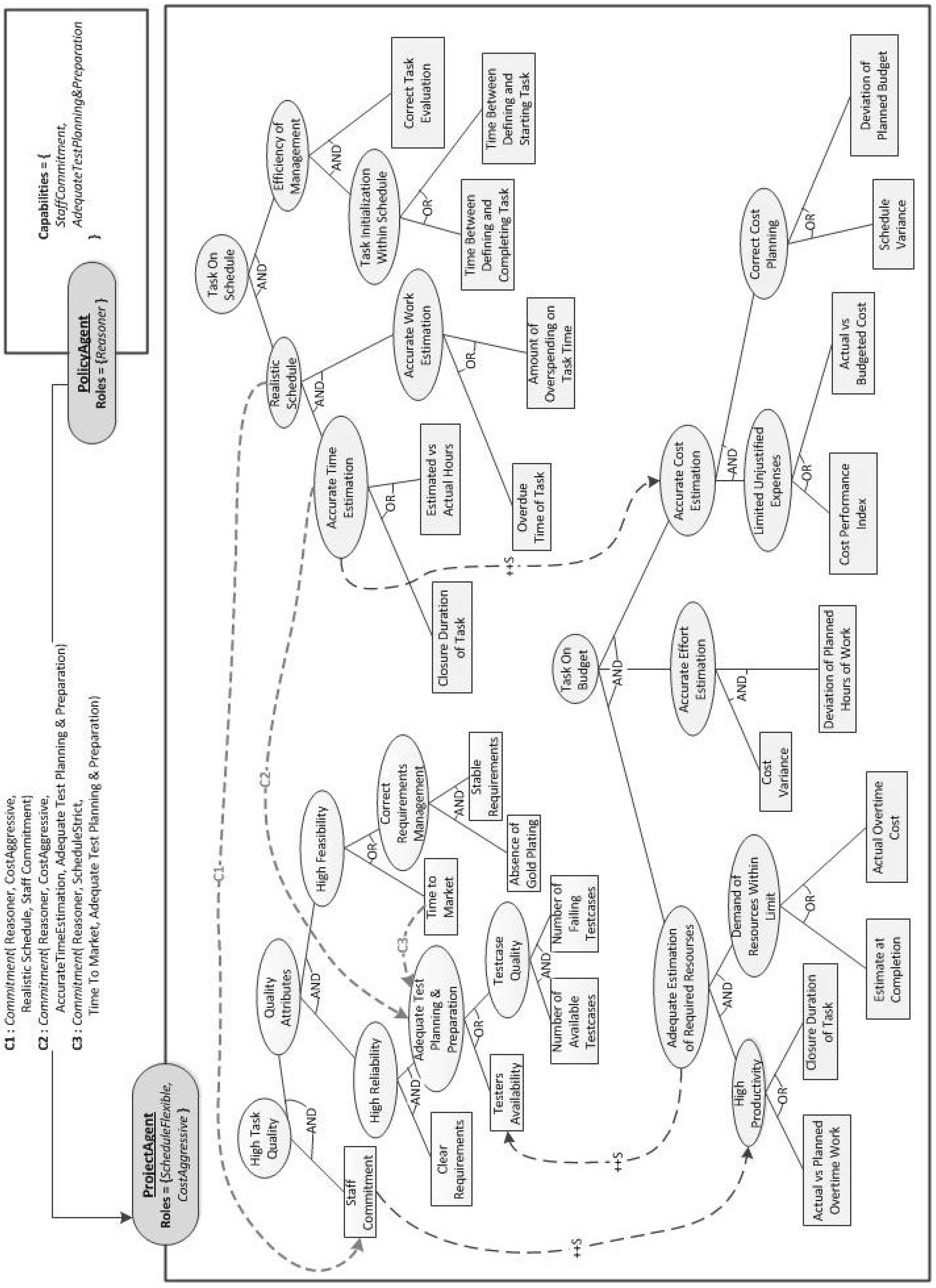

C1 : Commitment(Reasoner, CostAggressive, Realistic Schedule, Staff Commitment)
C2 : Commitment(Reasoner, CostAggressive, AccurateTimeEstimation, Adequate Test Planning & Preparation)
C3 : Commitment(Reasoner, ScheduleStrict, Time To Market, Adequate Test Planning & Preparation)
PolicyAgent
Roles = {Reasoner }
Capabilities = { StaffCommitment, AdequateTestPlanning&Preporation }
ProjectAgent
Roles = {ScheduleFlexible, CostAggressive }
Task On Schedule
Efficiency of Management
Correct Task Evaluation
Task Initialization Within Schedule
Time Between Defining and Starting Task
Time Between Defining and Completing Task
Realistic Schedule
Accurate Work Estimation
Amount of Overspending on Task Time
Overdue Time of Task
Accurate Time Estimation
Estimated vs Actual Hours
Closure Duration of Task
Correct Cost Planning
Deviation of Planned Budget
Schedule Variance
Accurate Cost Estimation
Limited Unjustified Expenses
Actual vs Budgeted Cost
Cost Performance Index
Task On Budget
Accurate Effort Estimation
Deviation of Planned Hours of Work
Cost Variance
Adequate Estimation of Required Resourses
Demand of Resources Within Limit
Actual Overtime Cost
Estimate at Completion
High Productivity
Closure Duration of Task
Actual vs Planned Overtime Work
High Task Quality
Staff Commitment
Quality Attributes
High Feasibility
Correct Requirements Management
Stable Requirements
Absence of Gold Plating
Time to Market
High Reliability
Adequate Test Planning & Preparation
Clear Requirements
Testers Availability
Testcase Quality
Number of Failing Testcases
Number of Available Testcases
AND
OR
++S
C1
C2
C3

Figure 4.3: AND/OR Goal Trees

Chapter 5

Goal Model Reasoning

In this chapter, the semantics of both the theory of the rules utilized to perform the reasoning analysis, as well as of the transformation function of these rules to first-order logic, are presented and described. Concluding, an example of those semantics is given for further clarification.

5.1 Goal Model to MLN Transformation

An important part of the reasoning analysis is the rule base generation i.e. the goal model transformation to the MLN rule base. In order to conduct the analysis in a concise and concrete manner, it is crucial to maintain a level of formalism over the transformation algorithm, when converting any instance of the goal model to the corresponding set of MLN rules, which will be capable of delivering accurate results keeping consistency for all test cases. The transformation function should yield a series of first-order formulas and predicates, which will satisfy all the laws presiding in the world of the goal model and the application under consideration and should express all the relations between goals, including AND/OR decompositions, contributions and agent commitments. For this purpose, it is important to specify the semantics used to perform the reasoning process.

5.1.1 Extended Semantics

The semantics definition available in the literature [7] on which our theoretical base lies, studies the case of goal support for an agent. However, in order to perform our reasoning analysis and conclude to probabilistic measurements accurately, goal support is not adequate. As an addition to goal support, actual goal achievement which derives from the investigation about whether an agent actually fulfils its part of the commitment during run-time execution, is needed. For this, we extended the semantics defined in [7] such that this requirement is met.

Let x be an agent with a set C of capabilities which undertakes a role $\varrho(x)$ to achieve its set of goals G, within an interaction protocol described by a set of commitments Π. Let also I be the interpretation of the world i.e. the assignment of truth values (T or $\perp$) to all symbols representing the goals, and G_C be the set of contributions according to the semantics defined in the previous section.

Therefore, the notation $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I g$ denotes that the protocol-bound specification of x infers the achievement of goal g under the interpretation I.

The extended semantics determined for goal achievement, based on the set given for goal support in [7], are given below:

Sem1 $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I T$

Sem2 $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I \alpha$ iff $G_C \cup I \not\models \neg\alpha$ and:

1. $\alpha \in C$, or
2. $G_A \xrightarrow{AND} \alpha \in G$,
   $\forall g \in G_A : \langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I g$, or
3. $G_O \xrightarrow{OR} \alpha \in G$,
   $\exists g \in G_O : \langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I g$, or
4. $\exists C(\varrho(y), \varrho(x), s, \alpha) \in \Pi.\varrho(x)$ such that
   $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I s$ and
   $\langle\langle G, C\rangle_y, \Pi.\varrho(y)\rangle \models^I \alpha$, or
5. $\exists C(\varrho(x), \varrho(y), s, u) \in \Pi.\varrho(x)$ such that
   $\langle\langle G, C\rangle_y, \Pi.\varrho(y)\rangle \models^I s$ and $s \models \alpha$

Sem3 $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I q \vee r$ iff

1. $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I q$ or $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models r$, or

2. $\exists C(\varrho(y), \varrho(x), s, q \vee r) \in \Pi.\varrho(x)$ such that $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I s$, $\langle\langle G, C\rangle_y, \Pi.\varrho(y)\rangle \models^I q \vee r$ and $G_C \cup I \not\models \neg(q \vee r)$

Sem4 $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I q \wedge r$ iff $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I q$ and $\langle\langle G, C\rangle_x, \Pi.r_x(x)\rangle \models r$

Sem5 $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I \neg q$ iff $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \not\models^I \neg q$

Clarifying all the above semantics, we can conclude to the following:

According to **Sem 1**, any specification models true under any interpretation.

According to **Sem 2**, an agent supports a symbol under a certain specification if and only if the existing constraints do not provoke a contradiction and either one of the following holds true:
(i) The symbol is a capability of the agent and the interpretation infers the satisfaction of the symbol i.e. the symbol is assigned with True as a truth value under the interpretation I.
(ii) There exists an AND-decomposition of the symbol and each sub-goal is satisfied under the interpretation.
(iii) There exists an OR-decomposition of the symbol and at least one sub-goal is satisfied under the interpretation.
(iv) There exists a commitment to the agent for the atomic proposition and both the antecedent and consequent hold.
(v) There exists a commitment from the agent to some other, the atomic proposition is entailed by the antecedent which has to be fulfilled by the later, and the consequent holds.
According to **Sem 3**, a disjunctive goal is supported if :
(i) At least one of its sub-goals is achieved.
(ii) There is a commitment to the agent for the disjunctive goal where the antecedent is achieved and a contradiction does not rise from the interpretation.
According to **Sem 4**, a conjunctive goal is supported if and only if all of its sub-goals are satisfied.
According to **Sem 5**, a negation of some goal is supported if and only if the goal itself is not supported.

When comparing the above semantics with those defined in [], the extensions introduced concern solely the notion of commitments and capabilities i.e. what differentiates the above semantics from those mentioned in the literature [7] is denoted in **Sem2**, cases i), iv) and v). Further elaborating on the aforementioned extensions, in **Sem2**-i) the agent must not only be capable of a goal α but should also fulfil the goal for it to be satisfied. Moreover, in **Sem2**-iv), the relation added signifies the fact that, in order to achieve goal satisfaction and not only goal support for an agent x when participating as the creditor in a commitment, the debtor must fulfill the consequent at run-time execution, so that the first can also consider it as an acquired resource. Consequently, when playing the role of the debtor, in case **Sem2**-v), the creditor must fulfil the antecedent in order for the debtor to consider it as obtained. Originally in [], there are two more semantics defined for $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I q \vee r$ and $\langle\langle G, C\rangle_x, \Pi.\varrho(x)\rangle \models^I q \wedge r$.

5.1.2 Rule Generation Semantics

Proceeding from the theoretical definition of the semantics used to their transformation into first-order logic formulas, in order to construct the corresponding MLN, a generation methodology is required to be defined and its consistency needs to be demonstrated. For clarity purposes, we first define the set of first-order logic predicates which constitute the first-order logic knowledge base used, as also done in [66]:

1. **Satisfied(**x,g**)** means that agent x managed to achieve its goal g i.e. g holds
2. **Capability(**x,g**)** means that agent x is capable of achieving g i.e. $g \in C_x$
3. **HasRole(**x,r_x**)** means that agent x has undertaken role r_x i.e $\Pi.r_x$
4. **Commitment(**r_x,r_y,g_1,g_2**)** means that an agent with role r_x is committed to an agent with role r_y for the g_2 if g_1 holds i.e $C(r_x, r_y, g_1, g_2) \in \Pi.r_x$
5. **PseudoCapability(**x,g**)** means that agent x is capable of achieving g through a commitment. The way this predicate is used is shown in the next section.

Additionally, a contribution of the form - -*S(g, g')* can be translated into the following formula :

$$Satisfied(x, g) \rightarrow \neg Satisfied(x, g') \tag{5.1}$$

while corresponding formulas are used for the remaining three contribution types.

As the KB, we consider the collection of rules governing the world under any interpretation. In addition, the KB includes ground atoms of type *Commitment*, which denote all the possible transactions between agents in the form of commitments. A possible interpretation of the world includes ground atoms of type *Capability* and *HasRole* regarding the agents interacting within the service engagement under scope. The aforementioned rules constituting the KB derive from the theoretical semantics defined in the above section, through their conversion into FOL formulas, as follows:

Sem1 $KB \models^I Satisfied(x, TrueNode)$

Sem2 $KB \models^I Satisfied(x, \alpha)$ iff
$KB \not\models^I \neg Satisfied(x, \alpha)$ and:

i) $Capability(x, \alpha) \in I$, or

ii) $G_A \xrightarrow{AND} \alpha \in G$,
$\forall g \in G_A : KB \models^I Satisfied(x, g)$, or

iii) $G_O \xrightarrow{OR} \alpha \in G$,
$\exists g \in G_O : KB \models^I Satisfied(x, g)$, or

iv) $Commitment(r_y, r_x, s, \alpha) \in KB$ and
$HasRole(x, r_x) \in I$ and $KB \models^I Satisfied(x, s)$ and
$HasRole(y, r_y) \in I$ and $KB \models^I Satisfied(y, \alpha)$ or

v) $Commitment(r_x, r_y, s, u) \in KB$ and
$HasRole(x, r_x) \in I$ and $HasRole(y, r_y) \in I$ and
$KB \models^I Satisfied(y, s)$ and
$KB \cup Satisfied(x, s) \models^I Satisfied(x, \alpha)$

Sem3 $KB \models \neg Satisfied(x, q)$ iff $KB \not\models Satisfied(x, q)$

According to **Sem1** $TrueNode$ holds for all agents specified in the goal model. This special node is introduced so as to be able to define unconditional commitments as:

$$Commitment(r_x, r_y, TrueNode, g)$$

Sem2 represents the rules used to reason towards the satisfaction of a specific goal α for agent x. More specifically, the above semantics are interpreted using the predicates mentioned, in the general forms shown below:

i. $Capability(x, \alpha) \rightarrow Satisfied(x, \alpha)$

ii. $G_A = \{g_1, \ldots, g_n\} \xrightarrow{AND} \alpha \in G,$
$Satisfied(x, g_1) \wedge \cdots \wedge Satisfied(x, g_n) \rightarrow$
$\rightarrow Satisfied(x, \alpha)$

iii. $G_O = \{g_1, \ldots, g_n\} \xrightarrow{OR} \alpha \in G,$
$Satisfied(x, g_1) \vee \cdots \vee Satisfied(x, g_n) \rightarrow$
$\rightarrow Satisfied(x, \alpha)$

iv. $Commitment(r_y, r_x, s, \alpha) \wedge HasRole(x, r_x)$
$\wedge Satisfied(x, s) \wedge HasRole(y, r_y) \wedge$
$\wedge Satisfied(y, \alpha) \rightarrow Satisfied(x, \alpha)$

v. $Commitment(r_x, r_y, s, u) \wedge HasRole(x, r_x) \wedge$
$\wedge HasRole(y, r_y) \wedge Satisfied(y, s) \rightarrow$
$\rightarrow Satisfied(x, s)$

It is important to note that the last rule does not imply direct achievement of α for agent x. On the contrary, the truth value of the ground predicate $Satisfied(x, s)$ must be inferred by the KB. Satisfaction of goal α can be inferred by the application of a contribution or an AND/OR decomposition related rule.

5.1.3 Handling Uncertainty

In order to train the MLN corresponding to the set of rules for a particular goal model, we distinguish the produced formulas into hard and soft constraints. Hard constraints are strictly true or false for all possible worlds depending on the existing evidence. On the contrary, the denial of a soft constraint allows the world to be valid for the knowledge base but with a level of certainty less than 100%. For the latter, the Alchemy tool is instructed to calculate and assign weights. The rules that were considered as soft constraints are those concerning contributions between goals, as shown in equation (1), and commitments between agents. Especially for the commitments rules, we have introduced the notion of *Pseudo-capability*, in order to achieve distinct weight calculation for each of the commitments existing. Taking this into account, the above rules regarding commitments, receive the following form:

Com1 $Commitment(r_y, r_x, s, \alpha) \wedge HasRole(x, r_x)$
$\wedge Satisfied(x, s) \wedge HasRole(y, r_y) \wedge$
$\wedge Satisfied(y, \alpha) \rightarrow PseudoCapability(x, \alpha)$

Com2 $Commitment(r_x, r_y, s, u) \wedge HasRole(x, r_x) \wedge$
$\wedge HasRole(y, r_y) \wedge Satisfied(y, s) \rightarrow$
$\rightarrow PseudoCapability(x, s)$

For each grounded commitment within the goal model specified, we have added one rule of the form :

$$PseudoCapability(x, g) \rightarrow Satisfied(x, g)$$

This constitutes a soft constraint and forces the Alchemy tool to calculate an appropriate distinct weight for each separate commitment, during the training procedure. In analogy with the commitments, a separate rule is also added for each contribution, in order to also assign distinct weights to each one of them.

5.2 DB Overview and Extraction

The data set utilized in order to test the validity of the technique, derived from an information repository pertaining to three industrial software projects. This data set contains information related to various software projects at the task-level of granularity. The per task database schema includes information that involves cost related data, time management related data, human resources data e.t.c. More specifically, the tables we considered from the database, contained information for 100 task specimens spanning throughout three different projects. In order to explain the notion of a task, exemplary tasks of a project would be "Review Feature Requirements with Stakeholders", "Design Database Schema", "Design Web UI" and so on. For each task entry, related data are given such as "Actual Cost", "Budgeted Cost", "Actual Start Date", "Actual Finish Date" e.t.c. Data related to the "Design Database Schema" task is depicted in Table 5.1.

In order to perform weight learning and inference, a set of ground atoms must be added to the KB. Those atoms are used as training examples in the weight learning phase or as evidence in the probabilistic inference phase. The rest of this section describes the process

Table 5.1: Data Related to "Design Database Schema" Task

Feature Name	**Value**
Actual Cost	$352,000
Budgeted Cost	$396,000
Actual Start Date	2008-12-15
Actual Finish Date	2008-12-26
BCWP	$352,000
ACWP	$352,000
BCWS	$396,000
Baseline Cost	$396,000
Baseline Work	2160 man-hours
Actual Work	1920 man-hours

of extracting those ground predicates from the database of the project repository.

First and foremost, a set of predicates called Ground Atoms Base (GAB), is generated from the goal model. GAB includes one capability predicate for each capability that the PolicyAgent has, and the ground atom *HasRole(PolicyAgent, Reasoner)*. For example for the tree in Figure 4.3 some of the GAB elements generated are the atoms G1-G3 given in Table 5.2. As soon as the GAB is generated, capability predicates are extracted from the project information repository, which will be referred to as Data Related Ground Atoms (DRGA). Those are capabilities assigned to the ProjectAgent. To be more specific, for each leaf node, the value of the metric is calculated according to an SQL query provided as an annotation. This value is then compared to the corresponding threshold, also provided as an annotation, and if the value signifies that the node holds, then an atom of the form Capability(ProjectAgent,[nodename]) is added to the DRGA. On the contrary, if the node does not hold the negation of the atom is added. Finally, DRGA also includes one *HasRole* atom for each role ProjectAgent adopts. The list of ground predicates produced for some leaf nodes corresponding to "Design Database Schema" task can be viewed in Table 5.4.

Further explaining the use of the KPI metrics defined in the previous chapter in relation with the database overview and contents, we annotated each of the leaf nodes of the goal tree with two pieces of information necessary for the verdict related to whether the goal stands under the specific world state. More specifically, an SQL query used to extract the appropriate data from the project information repository, along with the threshold of acceptance for the same metric, have been added to each leaf node. A subset of the KPIs

Table 5.2: Subset of FOL Formulas/Atoms Generated from the Goal Model in Figure 2.2. KB1-KB2 Consist the KB while G1-G3 the Ground Atoms Base

	Formula
KB1	PseudoCapability(ProjectAgent,StaffCommitment) $\rightarrow$ Satisfied(ProjectAgent,StaffCommitment)
KB2	Satisfied(ProjectAgent,AccurateTimeEstimation) $\rightarrow$ Satisfied(ProjectAgent,AccurateCostEstimation)
KB3	Satisfied(ProjectAgent,AdequateTestPlanningAndPreparation) $\wedge$ Satisfied(ProjectAgent,ClearRequirements) $\rightarrow$ Satisfied(ProjectAgent,HighReliability)
KB4	Satisfied(ProjectAgent,CorrectRequirementsManagement) $\vee$ Satisfied(ProjectAgent,TimeToMarket) $\rightarrow$ Satisfied(ProjectAgent,HighFeasability)
KB5	Commitment(Reasoner, CostAggressive, RealisticSchedule, StaffCommitment)
G1	HasRole(PolicyAgent, Reasoner)
G2	Capability(PolicyAgent, StaffCommitment)
G3	Capability(PolicyAgent, AdequateTestPlanningAndPreparation)

Table 5.3: A Subset of the KPIs Used

KPI Name	**Description**
Cost Performance Index (CPI)	Earned Value divided by the actual cost ($BCWP/ACWP$.)
Actual vs. budgeted costs	The actual costs relative to the budgeted costs of an activity.
Schedule Variance (SV)	Earned Value minus the planned budget for the completed work ($BCWP - BCWS$.)
Deviation of planned hours of work	The difference in work hours between the planned baseline against the actual statement of work.
Deviation of planned budget.	The difference in costs between the planned baseline against the actual budget.

Table 5.4: Ground Predicates Added to the DRGA for the Record of Table 5.1 for the corresponding KPIs of Table 5.3

Threshold	**Value**	**Ground Predicate**
≥ 1	1	Capability(ProjectAgent,CostPerformanceIndex)
≤ 1	0.89	Capability(ProjectAgent,ActualVSBudgetedCost)
≥ 0	-$44,000	!Capability(ProjectAgent,ScheduleVariance)
≥ 0	240	Capability(ProjectAgent,DeviationOfPlannedHoursOfWork)
≥ 0	$44,000	Capability(ProjectAgent,DeviationOfPlannedBudget)

corresponding to the appropriate leaf node sharing the same name, along with a description of what the metric represents, is given in Table 5.3.

An example of a leaf node and the corresponding KPI metric, along with the leaf annotation including the appropriate configuration threshold, can be seen in Table 5.4.

5.3 Process Outline

Taking all the above semantics into consideration, the process outline of the system encapsulating the algorithms implemented in order to accomplish the desired hypothesis verification are explained below. The basic algorithms used, which are also executed in the order stated below, were:

- **Rule Base Generation**: This algorithm includes all the steps followed towards modeling the various hypotheses into AND/OR Goal Trees enriched with contributions and commitments between goals and sub-goals. It also encapsulates the traversal of the generated goal model for the creation of the predicates that constitute the rule base.
- **Database Fact Extraction**: This algorithm includes all the steps to be completed in order to extract the necessary data from the database, taking into account the contents of the configuration file, and to utilize them to produce the evidence predicates which complete the knowledge base, along with the rule base generated by the the *Rule Base Generation algorithm* stated above.
- **Markov Logic Network Based Reasoning**: This algorithm includes the final verification of the hypothesis, through the use of the predicate files as input to the Alchemy tool for MLN construction and evaluation of the appropriate probabilities.

The process outline of the system is given in the steps below :

Step 1: Hypothesis Domain XMI Population and Storage. In this first step, the Hypothesis Domain Model is being populated with all the goals and their sub-goals along with the contributions and commitments between them, that if satisfied, verify the root goal which is the hypothesis type. This model population is then stored in XMI format, which provides flexibility and portability between different hardware and constitutes an easy to process file format.

Step 2: Strategy and Category definition. In this second step, the user/developer defines the type of strategy that will be applied, such as first - order logic constraints, Markov Logic Networks, OCL constraints e.t.c., and also the hypothesis category for which he or she desires to run the simulation, that is, the specific instance of the goal model he or she wishes to apply for the analysis. For the purposes of this study, the goal model instance is the one defined and populated in **Step 1**. However, ideally the user will be able to define various instances of goal models, store them and choose according to his or her requirements at every session.

Step 3: AND/OR Goal Tree Creation. In this third step of the procedure, the Hypothesis Domain XMI file created in **Step 1**, is loaded and parsed. The data extracted from the populated goal model is transformed into the appropriate AND/OR Goal Trees, which are useful in the depiction of the AND/OR relationships between the various sub-goals. The list of contributions and commitments existing in the goal model between various goals and sub-goals, is also parsed and the nodes of the resulting tree are enriched with information concerning those contributions and commitments in which they are implicated.

Step 4: Rule Base Generation. In this fourth step, the Goal Trees are traversed and for each node that is visited, the appropriate rule regarding its satisfiability is produced, according to the type of relation the node shares with its parent, as described through the semantics of the previous section. Furthermore, the contributions and commitments are parsed resulting to the addition of the corresponding rules to our KB, also according to the semantics of the previous section.

Step 5: Configuration XMI Population and Storage. In this fifth step, the configuration file necessary for the verification process in the form of an XMI file, is constructed and stored. The configuration file includes the leaf nodes of the goal model populated in **Step 1**, as well as the appropriate KPI metric for each leaf. The metric results to assigning a value to the node in the form of an SQL query, which will be automatically executed over the database under analysis, when it is required, in order to construct the evidence predicates in a later step of the procedure. The configuration file also includes for each metric, the corresponding threshold value i.e. that value that defines the margin within which the response of the database information to the query must lie.

Step 6: Goal Tree Node Annotations Creation. In the sixth step, the Goal Trees that were constructed **Step 3**, are further extended with the addition of annotations for each leaf node. These annotations are used to store the necessary information for the construction of the corresponding predicate to each leaf node of the tree, in the form of an SQL query and a threshold value, as previously mentioned in **Step 5**.

Step 7: **Database Fact Extraction**. In the seventh step, the evidence predicates based on the data deriving from the database, are produced and stored in a file, in order to be fed to the Alchemy tool.

Step 8: **MLN Reasoning**. In this last step of the procedure, the two files produced are provided to the Alchemy tool as input, in order to perform either weight learning for the formulas that constitute soft constraints, or as inference in order to conclude to probabilistic rates for goal achievement.

For better understanding, the process outline of the system is also given in the form of a block diagram, as shown in Figure 5.1.

5.3.1 Hypothesis Domain XMI Population and Storage

For the selection of the hypothesis to verify, the user needs to insert the necessary information for the properties that, if validated, ensure the hypothesis verification. The user inserts this information into the Hypothesis Modeler through the Editor of the Hypothesis Modeler Module, which is accessed through the UI Module, as previously mentioned, in accordance to his or her preferences and, also, in compliance with the means of their representation within the database schema under analysis. This implies that we have assumed name accordance between the database schema features and the corresponding nodes of the goal model inserted by the user in the Hypothesis XMI through the Editor and UI Module. The population of the Hypothesis Domain Model is exactly the process where the user populates the model, according to his or her own preferences and requirements, over the hypothesis for which the simulation will be executed. The populated instance of the goal model is then transformed into the Hypothesis Domain XMI document by the Hypothesis Modeler and stored into the Hypothesis DB Server of the Hypothesis Generator component. The storage of the Hypothesis Domain XMI into the Hypothesis DB Server is obligatory, since the user can retrieve information on past hypotheses simulations and hypotheses parameters, as well as edit them to suit his or her new demands.

For the purposes of this thesis, an appropriate Editor and User Interface were not developed but their task was simulated via the Eclipse runtime environment. More specifically, after the construction of the UML MOF class diagram using the MagicDraw UML Design

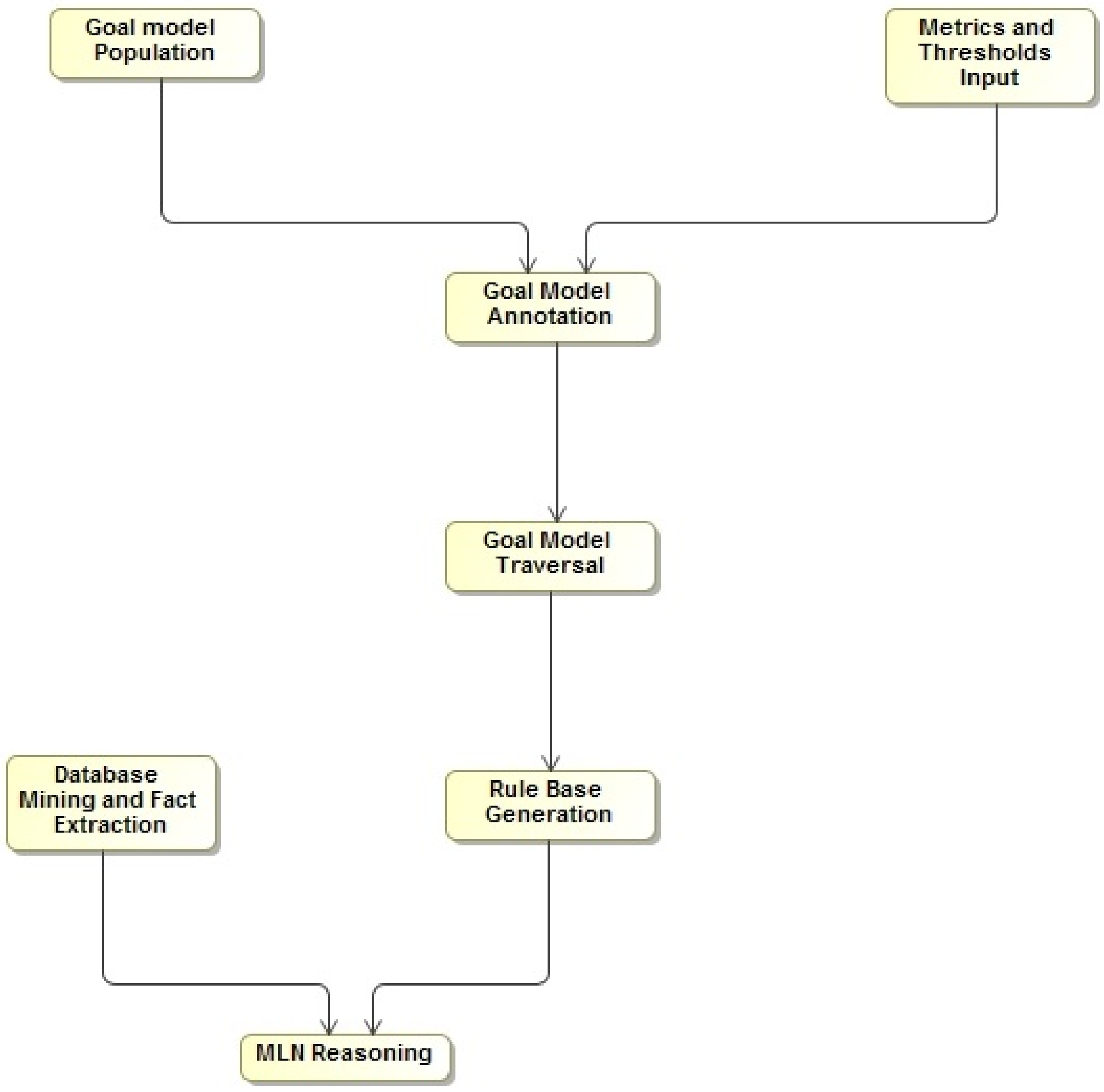


Figure 5.1: Block Diagram of the System Execution

tool, the diagram was exported in XMI E-core compliant form and imported into Eclipse, where the corresponding Java code was auto - generated for the model, as well as for the model editor. Continuing, the model editor was executed as an autonomous Eclipse application. This allowed us to insert exemplary data into the Hypothesis Domain Model and create an XMI, suitable to simulate the aforementioned procedure.

5.3.2 Strategy and Category definition

At a next step, it is requested that the user selects the hypothesis category that will be checked for validation, as well as the verification strategy that the system will apply on the data. This is accomplished again through the User Interface, via the *applyAnalysis* interface and the service it provides. The strategy and the hypothesis category selected, are parameters the user sets in the Controller of the Verification Module through the Subscriber component. The strategy can be based on the first - order logic theory, on the MLN theory, on OCL constraints ([45]) use e.t.c. As for the hypothesis category, the user can select any category from the ones displayed to him or her in a list by the User Interface which are the ones already existent within the Hypothesis DB Server, from previous executions of the framework. In the opposite case whereby the hypothesis category selected is not already existent in the database, the user is expected to insert information on the hypothesis types included into this category, the properties that constitute each type and the AND/OR relations between them, as explained in the previous subsection.

As it was noted above, for the purposes of this thesis, such an Editor and User Interface were not constructed. Therefore, this step of the procedure was encoded within the implementation. More specifically, the category and strategy that were set statically, are:

a) The hypothesis category *Name* was defined during the initialization of the execution according to the desired current test case.

b) The strategy applied, was also defined during the initialization of the execution by invoking a new instance of the appropriate Strategy sub-class which handles the strategy desired. In our case, an instance of the VerifierA class was invoked, since this was the sub-class of the Strategy class designed to handle MLN verification.

The initializations are held within the Test class of the source code implementation package, that includes the main function from which the program control flow starts. This class is also the one that is notified about the conclusion of the state and data changes during the verification procedure and, therefore, plays the role of the BlackBoard.

5.3.3 AND/OR Goal Tree Creation

Here, the hypothesis category requested by the user for verification, is being selected from the parsed Hypothesis Domain XMI document in order to create the appropriate AND/OR Goal Trees extended with contribution and commitment related information. For this purpose, the child nodes of the desired type within the XMI document, are being read one by one and according to their goal node type, i.e. ANDDecomposition, ORDecomposition, AtomicGoal, the corresponding goal node type is being instantiated with the name of the child node read. The type of a goal node indicates whether its children nodes in the AND/OR Goal Tree, if existent, are connected as a conjunction or as a disjunction. For each node its *parent* and *children* fields are filled with the appropriate goal nodes. Consequently, for each goal, it is checked within the XMI file whether it contributes to any other goal of the model and, if affirmative, this information fills the appropriate field of the instantiated goal node as well. In order to complete the AND/OR Goal Trees construction, according to the agents interacting and the roles it has been chosen for them to take, the list of applicable commitments is parsed from the XMI file and that information is also added in the appropriate fields of each goal instance.

It should also be mentioned, that each node can be assigned to a certain metric of the specific hypothesis type. However, for the purposes of this thesis, it was assumed that only nodes of type *AtomicGoal* which are the leaves of the trees, can correspond to metrics, here constructed through the usage of KPIs, while the nodes of type *DecompositionGoal* are iteratively defined as conjunctions or disjunctions of their children nodes.

5.3.4 Rule Base Generation

The inner algorithm implemented for this step, which is one of the core algorithms of the system, can be viewed in Figure 5.2. The output of the algorithm is printed in a file, in order to be used as input for the Alchemy tool.

Further explaining the algorithm shown in Figure 5.2, the input of the algorithm is the goal model under analysis. The output is a set of formulas that define the rule part of the first-order logic knowledge base in the form of a .mln file.

First, the set of nodes, agents and roles existing in the world, as well as the set of predicates that are going to be utilized, as defined in section 5.1.2, are declared.

Second, the rule regarding the notion of capabilities denoting that the existence of a node as capability for an agent infers its immediate satisfaction, is added in the output file.

Third, the set of commitments defined within the goal model is parsed and for each, four rules are added in the output file of the following form :

1. $Commitment(r_y, r_x, s, \alpha) \wedge HasRole(x, r_x)$
   $\wedge Satisfied(x, s) \wedge HasRole(y, r_y) \wedge$
   $\wedge Satisfied(y, \alpha) \rightarrow PseudoCapability(x, \alpha)$
2. $Commitment(r_x, r_y, s, u) \wedge HasRole(x, r_x) \wedge$
   $\wedge HasRole(y, r_y) \wedge Satisfied(y, s) \rightarrow$
   $\rightarrow PseudoCapability(x, s)$
3. $PseudoCapability(x, s) \rightarrow Satisfied(x, s)$
4. $PseudoCapability(x, \alpha) \rightarrow Satisfied(x, \alpha)$

The fourth step of the algorithm is the parsing of each goal tree within the goal model. For each goal node of each tree, its type is identified and those being AND/OR decompositions are extended with their list of children nodes as their annotation. In addition, a rule is added to the output file of the form:

1. $G_A = \{g_1, \dots, g_n\} \xrightarrow{AND} \alpha \in G$,
   $Satisfied(x, g_1) \wedge \dots \wedge Satisfied(x, g_n) \rightarrow$
   $\rightarrow Satisfied(x, \alpha)$ for each AND decomposition
2. $G_O = \{g_1, \dots, g_n\} \xrightarrow{OR} \alpha \in G$,
   $Satisfied(x, g_1) \vee \dots \vee Satisfied(x, g_n) \rightarrow$
   $\rightarrow Satisfied(x, \alpha)$ for each OR decomposition

Finally, for each of the goal nodes, it is tracked whether they contribute to any other goal of the goal model and, if affirmative, a rule is added in the output file of the form :

$$Satisfied(x, g) \rightarrow Satisfied(x, g') \tag{5.2}$$

where g is the goal examined at this point and g' is the goal to which the first contributes.

The above steps are repeated for all goal trees within the goal model, one by one.

An example part of one output file of type .mln is showed in Figure 5.3.

5.3.5 Configuration XMI Population and Storage

At this stage of the procedure, the configuration file has to be populated by the user, with the appropriate metrics in the form of SQL queries corresponding to existent goals of the goal model, as well as threshold values for each. This is useful later in the process, where the data extracted from the database will have to be used to calculate the metrics values. These values will, then, be compared against the appropriate thresholds, in order to decide upon the construction of a ground predicate for each node.

Regarding the Configuration XMI file, since it is already noted that an appropriate User Interface and Editor was not implemented, its population was again done manually through the use of the Eclipse runtime environment as needed for the two previous XMI files, and thresholds were assigned for the features appearing in the Data Warehouse sample constructed.

5.3.6 Goal Tree Node Annotations Creation

Here, the Goal Tree is enriched with annotations for each node. Therefore, one annotation container is instantiated for each node and the configuration file entered by the user, is loaded and parsed. Every time an atomic goal node is met, the configuration file is traversed and the SQL query corresponding to that leaf node is executed. The SQL queries are structured in such a way, that the value returned is already the outcome of the metric assigned to the goal node and, therefore, it is automatically compared towards the threshold or accepted value stated within the file for that metric. The result is stored in a new instance of the Annotation class. Next, this instance of the Annotation class is attached to the current node. This step is concluded as soon as all the leaf nodes have been enhanced with an annotation container containing the correct annotations.

5.3.7 Database Fact Extraction

The algorithm resulting to the database fact extraction is the second core algorithm of the procedure and is shown in Figure 5.4.

Further explaining the algorithm shown in Figure 5.4, the input of the algorithm is the database that includes all the information for the project under scope, as well as the configuration file containing all the queries and their corresponding thresholds, both annotating the leaf nodes. The output is a set of ground atoms that define the evidence part of the first-order logic knowledge base, in the form of a .db file.

First, for all the leaf nodes included in the configuration file, the query contained in the annotation is executed in the database.

Second, the value retrieved is compared with the corresponding threshold, which is also contained in the annotation of the node. If the value is within the boundaries defined by the threshold, then an appropriate ground atom in the form of a predicate is produced and added in the output file. The predicates produced are of the form:

$Satisfied(ProjectAgent, g)$

where g is the leaf node currently under analysis.

An example part of one output file of type .db is showed in Figure 5.5.

5.3.8 MLN Reasoning

On this last part of the process, the two files produced by the Rule Base Generation and the Data Fact Extraction algorithms are given as input to the Alchemy tool, in order for it to execute an **inference** procedure on the MLN that it first has to construct.

An example of a first - order logic world given as the .mln input to Alchemy, can be seen in Figure 5.6 below.

An example of a database of first - order logic predicates given as the evidence .db input file to Alchemy, can be seen in Figure 5.7 below.

The **Inference** algorithm implemented within the Alchemy tool, solves the problem of finding the most likely state of world given evidence. In a nutshell, the algorithm evaluates the maximum probability for a formula y to hold given x, as follows :

$$\max_y P(y \mid x) = \max_y \frac{1}{Z_x} \exp(\sum_i w_i n_i(x, y))$$

where Z_x is the *partition function* as mentioned and given in chapter 2.

This problem is the weighted **MaxSAT** problem and, thus, solved by using a weighted SAT solver e.g. the MaxWalkSAT. The WalkSAT algorithm in pseudocode, can be seen in Figure 5.8 below :

The MaxWalkSAT algorithm differs a little from the WalkSAT algorithm. This algorithm in pseudocode, can be seen in Figure 5.9 below :

In order to compute the probability of a formula given the MLN and the set of constraints C, $P(formula \mid MLN, C)$, the operation uses the *MCMC* algorithm and, therefore, constructs sample worlds and checks if the formula holds. In order to compute the probability $P(formula1 \mid formula2, MLN, C)$, if the formula2 is a conjuction of ground atoms, first it constructs a minimum subset of the network necessary to answer the query and afterwards it applies the MCMC algorithm.

The algorithm for the construction of the Gound Network in pseudocode can be seen in Figure 5.10 below :

Similarly, the MCMC (Gibbs sampling) algorithm in pseudocode can be seen in Figure 5.11.

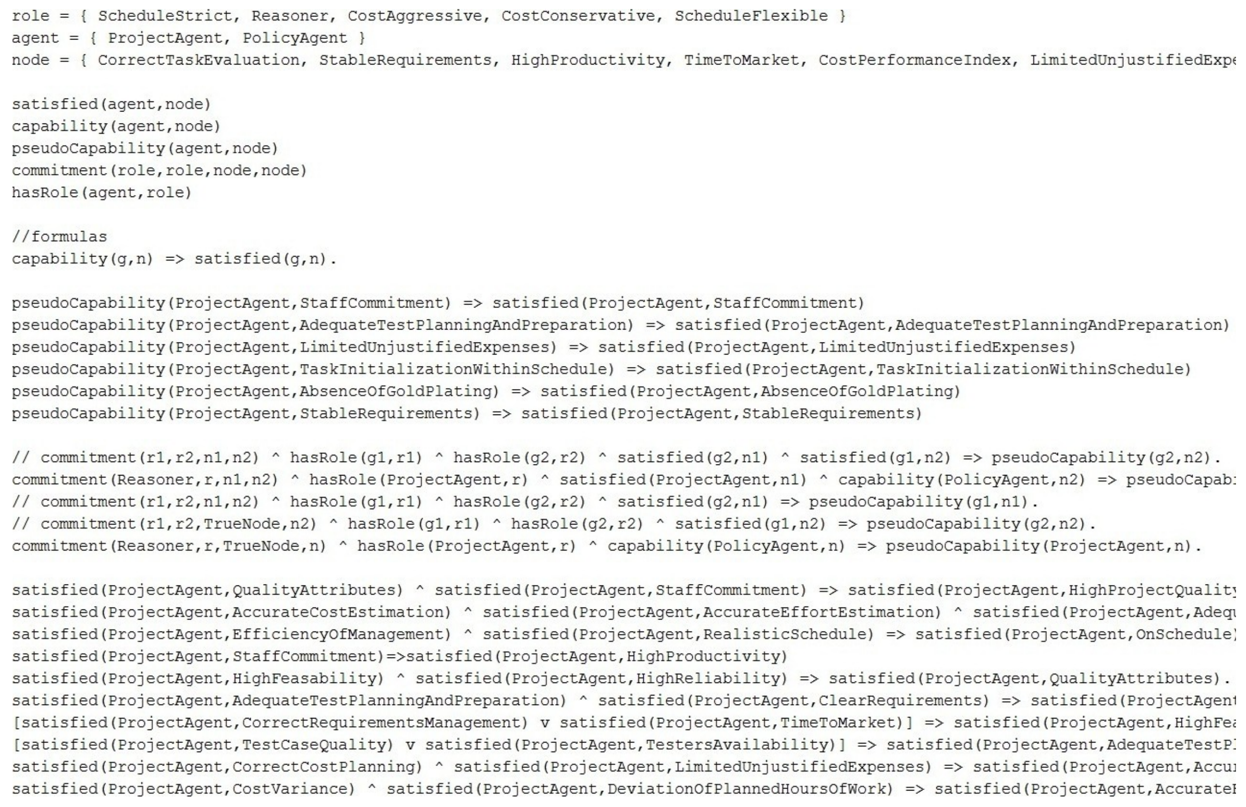
Finally, the output of the Alchemy tool is a set of percentages indicating the probability of occurrence for each goal and sub-goal of the goal model.

5.4 Summary

In this chapter, the theoretical basis of the reasoning analysis performed by the proposed framework is given. That is, the theoretical semantics introduced by [7], which formally define the rules presiding the knowledge base world in order for a goal to be considered as supported. These rules are, then, extended so that reflect goal achievement and not solely goal support. More specifically, the proposed framework has to validate a goal during run-time execution in order to perform the reasoning analysis and assign probabilistic rates correctly. However, goal support does not automatically infer goal achievement during an actual execution session of the system. For this, it was imperative to extend the semantics appropriately in order to explain the theory of approach. At a next step, an overview of the database, which was utilized in order to extract data used as test cases within the bounds of this thesis, as well as a brief description of the database fact extraction are given. This provides a better understanding of a possible schema for databases operating as a candidates for the framework, as well as potential types of data included within such databases. Finally, the process outline of the system execution, along with the main core algorithms implemented towards the reasoning analysis are presented and further explained.

```
Algorithm: Rule Base Generation
Input: goalModel: goal model for the application under consideration
Output: formulas: a set of formulas that define the rules of the first – order knowledge base
Procedure [formulas] generateFormulas (goalTree)   {

    Set CurrNode = top node in goalTree
    ForAll CurrNode in goalTree   {
        if CurrNode is ANDDecomposition {
            Set CurrNode.Annotation = CurrNode.children
            ForAll CurrNode.children except last
                [formulas].add(satisfied(ProjectAgent, childName) ^ )

            [formulas].add(satisfied(ProjectAgent, childName)  →  satisfied(ProjectAgent, CurrNode)  }

        if CurrNode is ORDecomposition  {
            Set CurrNode.Annotation = CurrNode.children
            ForAll CurrNode.children
                [formulas].add(satisfied(ProjectAgent, childName)  →  satisfied(ProjectAgent,
                    CurrNode)) }

        if CurrNode.contribution exists,
            [formulas].add(satisfied(CurrNode)  →  satisfied(CurrNode.contribution))  }

    return formulas  }

main (goalModel)  {
    [formulas] = set of nodes, agents, roles and set of predicate declarations;

    [formulas].add(Capability(PolicyAgent, x)  →  Satisfied(PolicyAgent, x))

    Set CurrCommit = first commitment in the goalModel
    ForAll CurrCommit  in goalModel  {
        [formulas].add(CurrCommit ^ HasRole(PolicyAgent, CurrCommit.debtor) ^
            HasRole(ProjectAgent,CurrCommit.creditor)^Satisfied(Reasoner, CurrCommit.antecedent)
            ^ Satisfied(ProjectAgent,CurrCommit.consequent)  →  PseudoCapability(PolicyAgent,
            CurrCommit.consequent))

        [formulas].add(CurrCommit ^ HasRole(PolicyAgent, CurrCommit.debtor) ^
            HasRole(ProjectAgent, CurrCommit.creditor)^Satisfied(ProjectAgent, CurrCommit.
            Antecedent)  →  PseudoCapability(PolicyAgent, CurrCommit.antecedent))

        [formulas].add(PseudoCapability(PolicyAgent, CurrCommit.consequent)  →
                                        Satisfied(PolicyAgent, CurrCommit.consequent)

        [formulas].add(PseudoCapability(PolicyAgent, CurrCommit.antecedent)  →
                                        Satisfied(PolicyAgent, CurrCommit.antecedent)      }

    ForAll goalTree in goalModel
        [formulas] = [formulas].add(generateFormulas(goalTree))  }
```

Figure 5.2: Rule Base Generation

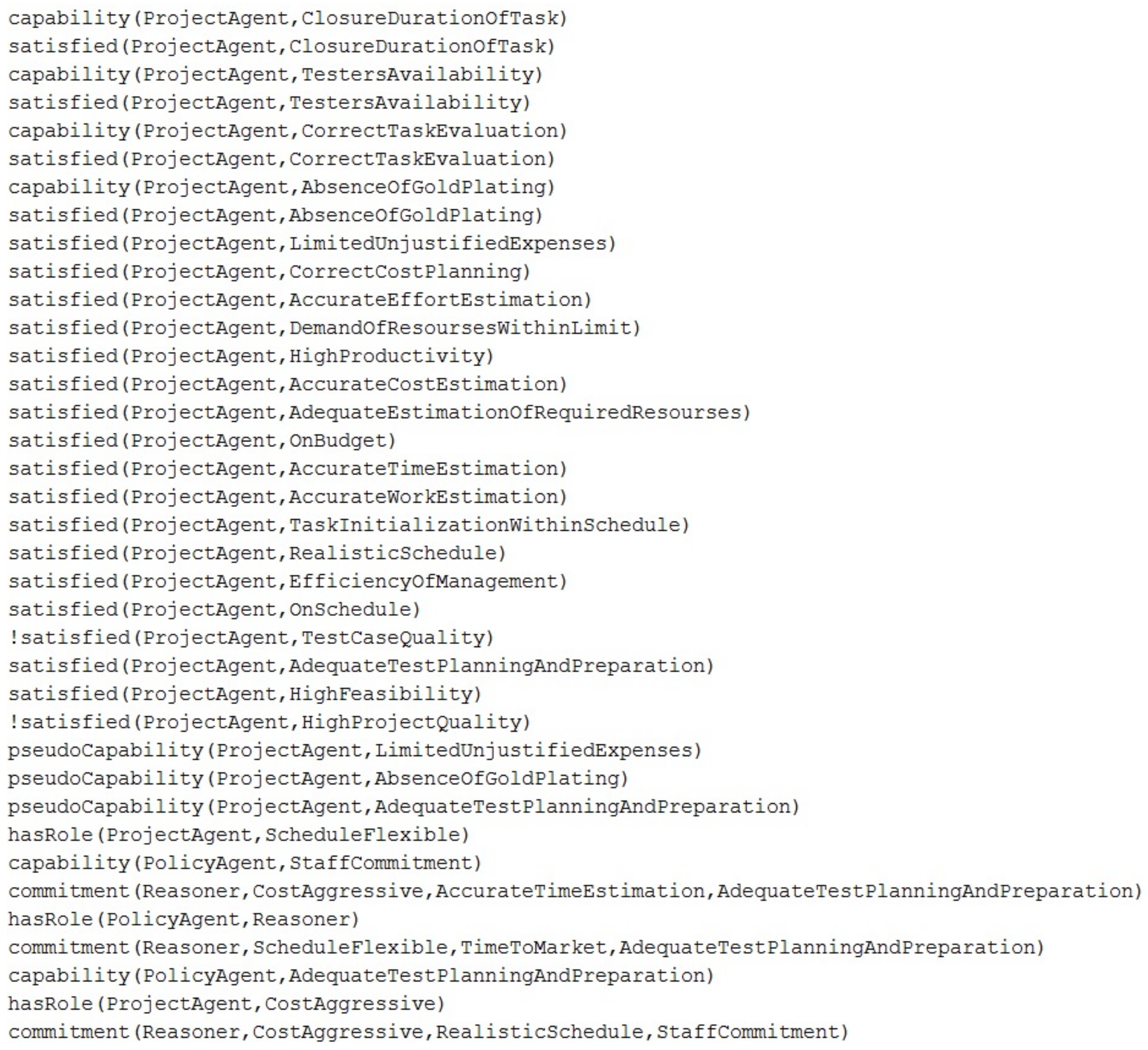


```
role = { ScheduleStrict, Reasoner, CostAggressive, CostConservative, ScheduleFlexible }
agent = { ProjectAgent, PolicyAgent }
node = { CorrectTaskEvaluation, StableRequirements, HighProductivity, TimeToMarket, CostPerformanceIndex, LimitedUnjustifiedExpe

satisfied(agent,node)
capability(agent,node)
pseudoCapability(agent,node)
commitment(role,role,node,node)
hasRole(agent,role)

//formulas
capability(g,n) => satisfied(g,n).

pseudoCapability(ProjectAgent,StaffCommitment) => satisfied(ProjectAgent,StaffCommitment)
pseudoCapability(ProjectAgent,AdequateTestPlanningAndPreparation) => satisfied(ProjectAgent,AdequateTestPlanningAndPreparation)
pseudoCapability(ProjectAgent,LimitedUnjustifiedExpenses) => satisfied(ProjectAgent,LimitedUnjustifiedExpenses)
pseudoCapability(ProjectAgent,TaskInitializationWithinSchedule) => satisfied(ProjectAgent,TaskInitializationWithinSchedule)
pseudoCapability(ProjectAgent,AbsenceOfGoldPlating) => satisfied(ProjectAgent,AbsenceOfGoldPlating)
pseudoCapability(ProjectAgent,StableRequirements) => satisfied(ProjectAgent,StableRequirements)

// commitment(r1,r2,n1,n2) ^ hasRole(g1,r1) ^ hasRole(g2,r2) ^ satisfied(g2,n1) ^ satisfied(g1,n2) => pseudoCapability(g2,n2).
commitment(Reasoner,r,n1,n2) ^ hasRole(ProjectAgent,r) ^ satisfied(ProjectAgent,n1) ^ capability(PolicyAgent,n2) => pseudoCapab
// commitment(r1,r2,n1,n2) ^ hasRole(g1,r1) ^ hasRole(g2,r2) ^ satisfied(g2,n1) => pseudoCapability(g1,n1).
// commitment(r1,r2,TrueNode,n2) ^ hasRole(g1,r1) ^ hasRole(g2,r2) ^ satisfied(g1,n2) => pseudoCapability(g2,n2).
commitment(Reasoner,r,TrueNode,n) ^ hasRole(ProjectAgent,r) ^ capability(PolicyAgent,n) => pseudoCapability(ProjectAgent,n).

satisfied(ProjectAgent,QualityAttributes) ^ satisfied(ProjectAgent,StaffCommitment) => satisfied(ProjectAgent,HighProjectQuality
satisfied(ProjectAgent,AccurateCostEstimation) ^ satisfied(ProjectAgent,AccurateEffortEstimation) ^ satisfied(ProjectAgent,Adequ
satisfied(ProjectAgent,EfficiencyOfManagement) ^ satisfied(ProjectAgent,RealisticSchedule) => satisfied(ProjectAgent,OnSchedule)
satisfied(ProjectAgent,StaffCommitment)=>satisfied(ProjectAgent,HighProductivity)
satisfied(ProjectAgent,HighFeasability) ^ satisfied(ProjectAgent,HighReliability) => satisfied(ProjectAgent,QualityAttributes).
satisfied(ProjectAgent,AdequateTestPlanningAndPreparation) ^ satisfied(ProjectAgent,ClearRequirements) => satisfied(ProjectAgent
[satisfied(ProjectAgent,CorrectRequirementsManagement) v satisfied(ProjectAgent,TimeToMarket)] => satisfied(ProjectAgent,HighFea
[satisfied(ProjectAgent,TestCaseQuality) v satisfied(ProjectAgent,TestersAvailability)] => satisfied(ProjectAgent,AdequateTestP
satisfied(ProjectAgent,CorrectCostPlanning) ^ satisfied(ProjectAgent,LimitedUnjustifiedExpenses) => satisfied(ProjectAgent,Accu
satisfied(ProjectAgent,CostVariance) ^ satisfied(ProjectAgent,DeviationOfPlannedHoursOfWork) => satisfied(ProjectAgent,AccurateE
```

Figure 5.3: Sample Part of a KB .mln file

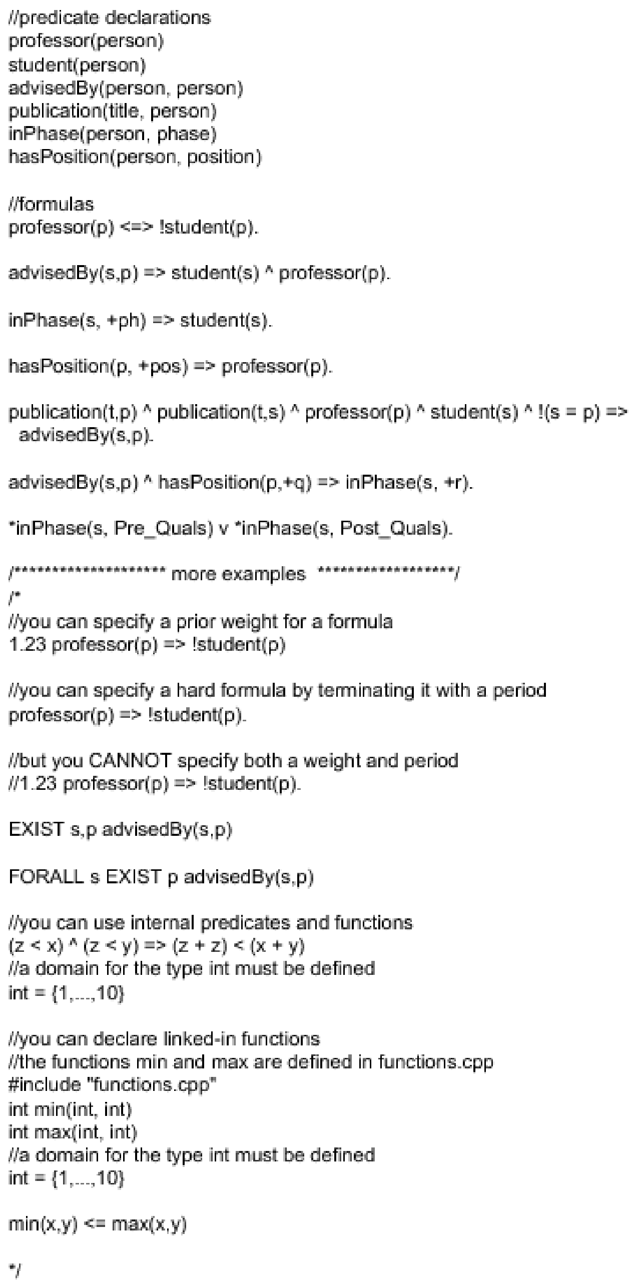
```
Algorithm: Database Fact Extraction
Input: Database: the database containing the data for the project under analysis
        Configuration: the file containing the leaf annotations describing queries and thresholds
Output: groundAtoms: a set of ground atoms as the evidence for the analysis
Procedure [groundAtoms] factExtraction (Database, Configuration)   {

    Set CurrLeaf  = the first leaf in Configuration
    ForAll CurrLeaf in Configuration {
        query = execute (CurrLeaf.annotation.query) in Database
        threshold =  CurrLeaf.annotation.threshold
        if (query within threshold)
            [groundAtoms].add(satisfied(ProjectAgent, CurrLeaf))  }

    return groundAtoms   }

main (Database, Configuration)  {
    [groundAtoms] = factExtraction (Database, Configuration)  }
```

Figure 5.4: Database Fact Extraction



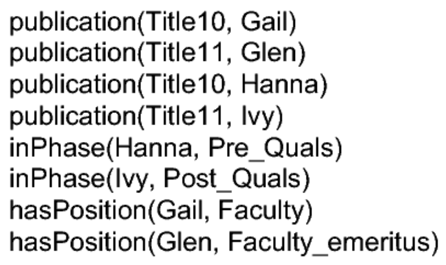
```
capability(ProjectAgent,ClosureDurationOfTask)
satisfied(ProjectAgent,ClosureDurationOfTask)
capability(ProjectAgent,TestersAvailability)
satisfied(ProjectAgent,TestersAvailability)
capability(ProjectAgent,CorrectTaskEvaluation)
satisfied(ProjectAgent,CorrectTaskEvaluation)
capability(ProjectAgent,AbsenceOfGoldPlating)
satisfied(ProjectAgent,AbsenceOfGoldPlating)
satisfied(ProjectAgent,LimitedUnjustifiedExpenses)
satisfied(ProjectAgent,CorrectCostPlanning)
satisfied(ProjectAgent,AccurateEffortEstimation)
satisfied(ProjectAgent,DemandOfResoursesWithinLimit)
satisfied(ProjectAgent,HighProductivity)
satisfied(ProjectAgent,AccurateCostEstimation)
satisfied(ProjectAgent,AdequateEstimationOfRequiredResourses)
satisfied(ProjectAgent,OnBudget)
satisfied(ProjectAgent,AccurateTimeEstimation)
satisfied(ProjectAgent,AccurateWorkEstimation)
satisfied(ProjectAgent,TaskInitializationWithinSchedule)
satisfied(ProjectAgent,RealisticSchedule)
satisfied(ProjectAgent,EfficiencyOfManagement)
satisfied(ProjectAgent,OnSchedule)
!satisfied(ProjectAgent,TestCaseQuality)
satisfied(ProjectAgent,AdequateTestPlanningAndPreparation)
satisfied(ProjectAgent,HighFeasibility)
!satisfied(ProjectAgent,HighProjectQuality)
pseudoCapability(ProjectAgent,LimitedUnjustifiedExpenses)
pseudoCapability(ProjectAgent,AbsenceOfGoldPlating)
pseudoCapability(ProjectAgent,AdequateTestPlanningAndPreparation)
hasRole(ProjectAgent,ScheduleFlexible)
capability(PolicyAgent,StaffCommitment)
commitment(Reasoner,CostAggressive,AccurateTimeEstimation,AdequateTestPlanningAndPreparation)
hasRole(PolicyAgent,Reasoner)
commitment(Reasoner,ScheduleFlexible,TimeToMarket,AdequateTestPlanningAndPreparation)
capability(PolicyAgent,AdequateTestPlanningAndPreparation)
hasRole(ProjectAgent,CostAggressive)
commitment(Reasoner,CostAggressive,RealisticSchedule,StaffCommitment)
```

Figure 5.5: Sample Part of a KB .db file



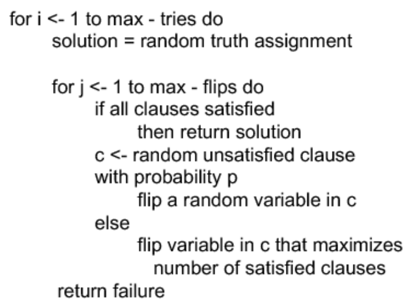
```
//predicate declarations
professor(person)
student(person)
advisedBy(person, person)
publication(title, person)
inPhase(person, phase)
hasPosition(person, position)

//formulas
professor(p) <=> !student(p).

advisedBy(s,p) => student(s) ^ professor(p).

inPhase(s, +ph) => student(s).

hasPosition(p, +pos) => professor(p).

publication(t,p) ^ publication(t,s) ^ professor(p) ^ student(s) ^ !(s = p) =>
  advisedBy(s,p).

advisedBy(s,p) ^ hasPosition(p,+q) => inPhase(s, +r).

*inPhase(s, Pre_Quals) v *inPhase(s, Post_Quals).

/******************** more examples  ******************/
/*
//you can specify a prior weight for a formula
1.23 professor(p) => !student(p)

//you can specify a hard formula by terminating it with a period
professor(p) => !student(p).

//but you CANNOT specify both a weight and period
//1.23 professor(p) => !student(p).

EXIST s,p advisedBy(s,p)

FORALL s EXIST p advisedBy(s,p)

//you can use internal predicates and functions
(z < x) ^ (z < y) => (z + z) < (x + y)
//a domain for the type int must be defined
int = {1,...,10}

//you can declare linked-in functions
//the functions min and max are defined in functions.cpp
#include "functions.cpp"
int min(int, int)
int max(int, int)
//a domain for the type int must be defined
int = {1,...,10}

min(x,y) <= max(x,y)

*/
```

Figure 5.6: Example of a .mln input file



```
publication(Title10, Gail)
publication(Title11, Glen)
publication(Title10, Hanna)
publication(Title11, Ivy)
inPhase(Hanna, Pre_Quals)
inPhase(Ivy, Post_Quals)
hasPosition(Gail, Faculty)
hasPosition(Glen, Faculty_emeritus)
```

Figure 5.7: Example of a .db input file



```
for i <- 1 to max - tries do
    solution = random truth assignment

    for j <- 1 to max - flips do
        if all clauses satisfied
            then return solution
        c <- random unsatisfied clause
        with probability p
            flip a random variable in c
        else
            flip variable in c that maximizes
              number of satisfied clauses
    return failure
```

Figure 5.8: WalkSAT algorithm

```
for i <- 1 to max - tries do
    solution = random truth assignment

    for j <- 1 to max - flips do
        if $\Sigma$ weights (sat.clauses ) > threshold
            then return solution
        c <- random unsatisfied clause
        with probability p
            flip a random variable in c
        else
            flip variable in c that maximizes
              $\Sigma$ weights (sat. clauses)
    return failure, best solution found
```

Figure 5.9: MaxWalkSAT

```
network <- {}
queue <- query nodes

repeat
    node <- front (queue)
    remove node from queue
    add node to network
    if node not in evidence
        then add neighbors (node) to queue
until queue = {}
```

Figure 5.10: Ground Network Construction Algorithm

```
state <- random truth assignment
for i <- 1 to num - samples do
        for each variable x do
                sample x according to P(x | neighbors (x))
                state <- state with new value of x
P (F) <- fraction of states in which F is true
```

Figure 5.11: MCMC : Gibbs sampling Algorithm

Chapter 6

Case Studies

In the previous chapters, the architecture, the goal model and its theoretical base, the structure of the database in our disposition, the project risk management goal model instances used, as well as the algorithms implemented for the purposes of this framework were discussed. In this chapter, some case studies of the framework that were conducted in order to assess the performance, the accuracy and the stability of the proposed mining and reasoning technique, as this is applied to an industrial project data repository, is presented. More specifically, we examine the proposed framework in two areas namely Risk and Quality Assessment and Error Proneness. The goal models for risk and quality assessment were compiled by taking into account elements of the KPI libraries "Project Portfolio" and "Quality, Improvement and Innovation". Similarly, the goal model for error proneness was compiled by taking into account the ISO/IEC TR 9126-2 [49] and ISO/IEC TR 9126-3 [50].

6.1 Risk and Quality Assessment

In order to evaluate the results of the proposed framework, in terms of its ability to conduct risk and quality analysis, three types of experimental studies have been conducted, first to assess performance and second to assess stability. Below, we present in more detail the evaluation results. The software tools utilized for the simulation of the framework were the MagicDraw Enterprise 16.5 edition software tool, for the design of the UML diagrams defining the goal domain model and exporting it to XMI e - core compliant, the Eclipse Modeling Framework (EMF) version Indigo 3.7, for the code generation and development and the Alchemy tool for the Markov Logic Networks learning and inference, so that the final probabilistic reasoning of each goal hypothesis could be performed.

For better understanding, referring to the project management goal trees presented in chapter 4 in Figure 4.3, which were used in order to perform our case studies, a complete list of the nodes is provided with brief explanations of the symbols:

High Task Quality

Staff Commitment: This node is a leaf node and quantifies the commitment of the staff to the task in question, i.e. it reflects parameters that indicate whether the staff is productive and dedicated to the completion of the task.

Clear Requirements: This node is a leaf node and indicates whether the requirements of the task have been set in a clear manner, without allowing misconceptions or possibility for misinterpretations.

Testers Availability: This node is a leaf node and represents the number of available testers for the task which should comply to the threshold determined by the user for the reasoning analysis session.

Number of Available Test Cases: This node is a leaf node and represents the number of available test cases which should be adequate for evaluating the task properly.

Number of Failing Test Cases: This node is a leaf node and represents the number of test cases that fail to return correct results, in order to verify the task's correctness.

Time to Market: This node is a leaf node and reflects the time left until the overall project needs to be released to market.

Absence of Gold Plating: This node is a leaf node and represents whether the task continues to be improved well past the point where the extra effort is worth the value it adds.

Stable Requirements: This node is a leaf node and represents whether the requirements of the task are considered to be in a stable state, i.e. there are no significant changes to be made.

High Reliability: This node is an AND decomposition and represents the level of reliability of the task, i.e. whether the task demonstrates attributes that render it a reliable piece of software for use, integration e.t.c.

Adequate Test Planning & Preparation: This node is an OR decomposition and denotes whether the plans and preparations made for the task regarding its testing, are adequate in terms of the demands set by the developing team and the customer.

Testcase Quality: This node is an AND decomposition and reflects the quality of the test cases for this task, since the test case quality is a crucial factor affecting the testability of the task and, thus, it plays a significant role in the overall quality of the task.

Correct Requirements Management: This node is an AND decomposition and evaluates the management held for the requirements set for the task. The requirements are a very important aspect of the task, therefore, their management and planning in advance, in a correct manner, is tightly connected to the task's quality.

High Feasibility: This node is an OR decomposition and reflects whether a task is evaluated as feasible, i.e. if it demonstrates aspects that advocate towards the capability of it being completed successfully.

Quality Attributes: This node is an AND decomposition and represents the general notion of the set of attributes that must be examined in order to evaluate the quality of a task.

High Task Quality: This node is an AND decomposition and is the top goal in need of fulfilment. That is, whether the task maintains high quality levels.

Task On Budget

Cost Performance Index: This node is a leaf node and represents the quantitative value of the Cost Performance Index (CPI), which is a standard metric used for cost estimation and assessment. The CPI is the rate of the budgeted cost of the work performed versus the actual cost of the work performed. Thus, it becomes obvious that the CPI is optimal when maintaining higher values.

Actual vs Budgeted Cost: This node is a leaf node and represents the comparison between the overall actual cost of the task and the overall budgeted cost of the task. It is again evident that the budgeted cost should be targeted to be kept higher than the actual cost.

Schedule Variance: This node is a leaf node and represents the variance of the schedule mapped for the task. That is, the deviations between the planned schedule and the actual progress of the task development.

Deviation of Planned Budget : This node is a leaf node and gives an estimate of the deviation the planned budget presents with the actual costs of the task.

Cost Variance : This node is a leaf node and represents the variance of the cost of the task, that is, the deviations between the estimated cost and the actual cost of the task development.

Deviation of Planned Hours of Work: This node is a leaf node and gives an estimate of the deviation the planned hours of work presents with the actual hours of work that have been required so far for the task.

Estimate at Completion: This node is a leaf node and represents the estimated time left until the completion of task.

Actual Overtime Cost: This node is a leaf node and represents the actual cost that resulted because of overtime work required.

Actual vs Planned Overtime Work: This node is a leaf node and represents the difference between the actual hours of overtime work that were demanded, as opposed to the amount of hours of overtime work that were planned initially.

Closure Duration of Task: This node is a leaf node and represents the time needed for the task to be considered as closed.

Limited Unjustified Expenses: This node is an OR decomposition and estimates the unjustified expenses arising for a certain task and whether they are under a specific limit.

Correct Cost Planning: This node is an OR decomposition and reflects whether the prior planning made for the task regarding its cost, has been done in a correct manner such that it won't eventually exceed the expected levels to a great extent.

Demand of Resources Within Limit: This node is an OR decomposition and estimates whether the resources (human, monetary, hardware e.t.c.) demanded for the completion of a task are within a certain limit. If this does not hold, then it is possible that the resources will be exhausted during task development.

High Productivity: This node is an OR decomposition and reflects the productivity levels of the human resources assigned to a specific task.

Accurate Effort Estimation: This node is an AND decomposition and evaluates the estimates made regarding the effort required in order for the task to be completed.

Accurate Cost Estimation: This node is an AND decomposition and evaluates the estimates made regarding the monetary resources required, in order for the task to be completed.

Adequate Estimation of Required Resources : This node is an AND decomposition and evaluates the estimates made regarding the overall required resources, that do not fall into the two aforementioned categories, in order for the task to be completed.

Task On Budget: This node is an AND decomposition and is the top goal in need of achievement, that is, whether the task remains eventually within budget.

Task On Schedule

Closure Duration of Task: This node is a leaf node and a duplicate of the one included in the "Task on Budget" goal tree.

Estimated vs Actual Hours: This node is a leaf node and represents the comparison between the actual hours of work required for the task and the estimate that was initially assumed.

Overdue Time of Task: This node is a leaf node and measures the amount of time required for the task development, which exceeds the date planned for its completion.

Amount of Overspending on Task Time: This node is a leaf node and measures the amount of time which was spent on the task but was evaluated as unnecessary or could have been avoided.

Time Between Defining and Completing Task: This node is a leaf node and represents the time period between the definition and completion of the task.

Time Between Defining and Starting Task: This node is a leaf node and represents the time period between the definition and start of the task.

Correct Task Evaluation: This node is a leaf node and assesses the overall evaluation methods applied upon the task, in order to ensure their correctness and completeness.

Accurate Time Estimation: This node is an OR decomposition and evaluates the estimation made for the time required for the accomplishment of the task.

Accurate Work Estimation: This node is an OR decomposition and evaluates the estimation made for the work hours required for the accomplishment of the task.

Task Initialization Within Schedule: This node is an OR decomposition and examines whether the initialization of a task was held within schedule. It is one of the most important phases of the task, since in this stage the requirements are set and the planning is performed.

Realistic Schedule: This is an AND decomposition and examines whether any schedule mapped for the task development has been held within realistic boundaries.

Efficiency of Management: This is an AND decomposition and reflects whether the task development is performed through flexible and accurate management.

Task On Schedule: This node is an AND decomposition and is the top goal in need of achievement, that is, whether the task remains eventually within schedule.

From the set of all the above goals, the subset defined as leaf nodes results to the generation of one corresponding predicate in the knowledge base of a reasoning analysis session for each, if and only if the data deriving from the database within scope and implicated in the appropriate KPI metric, complies to the threshold boundaries set for that metric by the user. That is, if the value of a metric is considered to be accepted on the bases that it is higher (lower) than a threshold, and if the data mined result to the calculation of a value for that metric which is higher (lower) than that threshold, then a predicate is produced and added in the knowledge base.

Each of the predicates produced and added in the knowledge base according to the above method, is of type **Capability(PolicyAgent, g)**, as explained in the previous chapters. That is, the goal g can be inferred as true from the data mined from the database and, therefore, is considered to be a capability for agent *PolicyAgent*, which is the agent responsible for handling the reasoning process.

6.1.1 Training Performance

Prior to proceeding with assessing our system, it was necessary to complete the training process of the Markov Logic Network generated by the goal models defined for the study. More specifically, we trained our MLN varying the cardinality of the training set, in order to make observations concerning the training time and the changes in weight values calculated for the formulas. This experiment investigates time requirements for the training session as a function of the number of test cases included in the training session.

The results are illustrated in Figure 6.1. It can be seen that the training time increases almost linearly with the number of test cases included in the training set. Additionally,

Table 6.1: Expected Outcome (E.O) and Probability Assigned by the MLN Analysis (MLN) for the 10 Test Cases.

		TC1	**TC2**	**TC3**	**TC4**
Task On Budget	E.O.	✓	✓	✓	✗
	MLN	0.883	0.805	0.884	0.553
Task On Schedule	E.O.	✓	✗	✓	✗
	MLN	0.890	0.613	0.945	0.578
High Task Quality	E.O.	✗	✓	✓	✗
	MLN	0.632	0.968	0.976	0.623

		TC5$_A$	**TC6**$_A$	**TC7**$_A$	**TC5**$_B$	**TC6**$_B$	**TC7**$_B$
Task On Budget	E.O.	✓	✗	✗	✓	✗	✗
	MLN	0.814	0.516	0.512	0.844	0.548	0.554
Task On Schedule	E.O.	✗	✓	✗	✗	✓	✗
	MLN	0.562	0.892	0.583	0.544	0.834	0.584
High Task Quality	E.O.	✗	✗	✓	✗	✗	✓
	MLN	0.579	0.585	0.845	0.617	0.655	0.875

we recorded the weight values calculated for formulas **KB1** and **KB2** of Table 5.2. The normalized weight values are depicted in Figure 6.2. The values are normalized on the [0,1] interval for each test case. We chose to present the normalized values of the weights, since their change rate is of interest rather than their absolute values. We observe in Figure 6.2, that, as the number of test cases for which a formula is verified increases, its weight also increases accordingly, which applies for formula **KB1**. In analogy, for each test case added in the set which does not verify a formula, its weight decreases, as in case of formula **KB2**.

6.1.2 Inferencing

In order to validate the inference of the proposed analysis method, two test scenarios have been created and run. The first scenario aims to prove the accuracy of the reasoning. That is, if the outcome is known a priori for a certain test case, the system should be expected to conclude that this test case holds with a high probability. The second test scenario involves the observation of probability changes on specific goals for a project, when various parameters influencing the result are altered gradually over time. Our aim is to show that

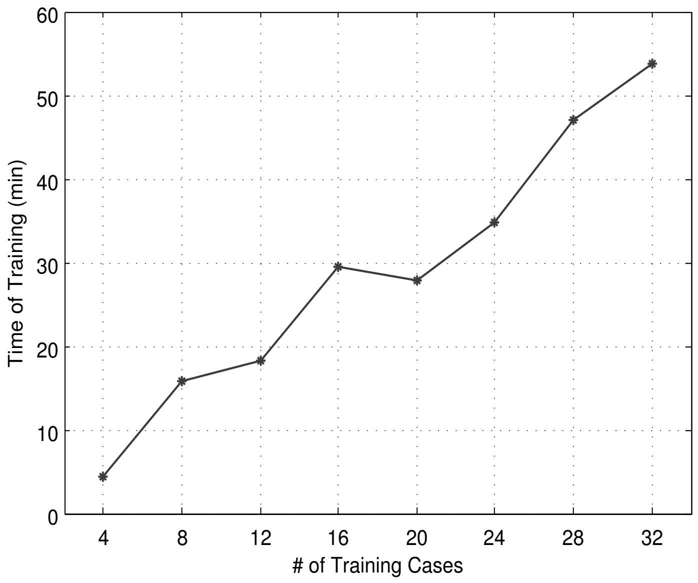


Figure 6.1: Time Required for Training against the Number of Training Cases

the system behaves in a stable manner.

For the two aforementioned test scenarios, we separated a subset of the data available, with indicative cardinality of 30 specimens. In order to ensure the credibility of our training, which plays a crucial role in the testing of our method, the selection of the specimens was based on the most complete test cases in terms of data. This data subset was utilized in order to conduct the training session of the Markov Logic Network, which targets to assign weight values to all formulas of our first-order knowledge base for which we have made an open world assumption. Indicatively, we ran the inference procedure twelve times using a set of input DRGA files varying the number of predicates included in each, i.e. the number of predicates spanning from 2 to 24. The resulting inference time was between 0.12-0.14 seconds for all cases.

Accuracy

This test scenario aims to assess the accuracy of the obtained results using the proposed method. For this, we have isolated ten test cases, where the outcome of the task state regarding whether the task is "On Budget, "On Schedule" and of "High Quality" project trait, was known a priori. The ten test cases can be viewed in Table 6.1. For each test case,

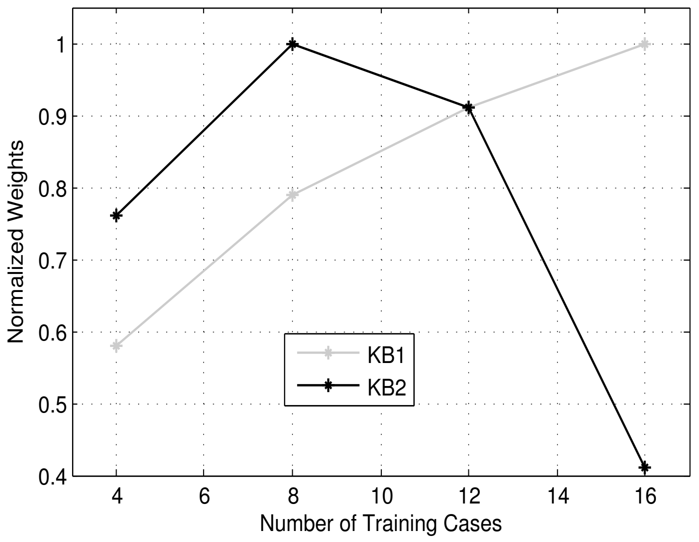


Figure 6.2: Normalized Weight for Formulas KB1 and KB2 of Table 1

the expected outcome and the probability for the resulting state of the project assigned by the MLN analysis, are depicted. For example, test case 1, (TC1) signifies that the task is "On Budget", "On Schedule", but not of "High Quality" and the corresponding probabilities calculated from the MLN are 0.883, 0.890 and 0.632 respectively. From the results presented in Table 6.1, it can be seen that the probability rates assigned by the analysis, agree with the outcome expected for each case with a good level of accuracy, which proves the validity of the reasoning technique.

Moreover, for the couples ($TC5_A$-$TC5_B$), ($TC6_A$-$TC6_B$) and ($TC7_A$-$TC7_B$), the DRGA is identical. It is important to note that for the set $TC5_A$-$TC7_A$, the *ProjectAgent* has adopted the roles *ScheduleStrict* and *CostConservative* at run-time, while for the set $TC5_B$-$TC7_B$, it has adopted the roles *ScheduleFlexible* and *CostAggressive*. It is interesting to observe that the probabilities calculated, differ since for each set of roles, a different set of commitments is used in the reasoning process.

Stability

For evaluating stability, we have based our experimentation upon test case TC4 of the previous section, as it can be seen in Table 6.1. This test case was selected because it shows the worst results (i.e. all objectives do not hold), and therefore, we experimented by gradually improving various data values that affected the outcome negatively, in order to observe how results would also improve. The new results are presented in the form of a diagram illustrated in Figure 6.3, where the changes in the probability rates for each goal are depicted against the changes in the data values.

Looking into these results, some remarks can be made. In the initial case, which is the starting point, all the nodes of our goal model are not satisfied. In a first improvement attempt, we aimed to raise the probability of the task to maintain a high quality level. For this, the data affecting the values of the nodes *"Staff Commitment"*, *"Clear Requirements"* e.t.c., that is, the leaves of the "High Task Quality" tree (see Figure 4.3), have been altered in the order indicated. In a second improvement attempt, we targeted to also raise the probability of the task to be on budget. In order to meet this goal, we attempted to satisfy the nodes *"Cost Performance Index"*, *"Actual vs Budgeted Cost"* e.t.c., that is, the leaves of the "Task On Budget" tree (see Figure 4.3) again in the order shown. Finally, we altered the data affecting *"Closure Duration of Task"*, *"Estimated vs Actual Hours"* e.t.c., that is, the leaves of the "On Task Schedule" tree as numbered (see Figure 4.3) in an attempt to increase the probability of task to be on schedule.

As it can be observed, the gradient of the graph demonstrates expected behavior, that is, as more parameters are improved, the resulting probability is gradually increased without any unexpected peaks or erroneous behaviour. The fluctuations on the probabilities are due to the interaction among tree goals, but the overall trend is for the probabilities to increase, as new supporting evidence comes into the picture.

6.2 Error Proneness

In order to, both, investigate the system's ability to perform fault prediction and, simultaneously, to evaluate the significance of the implication of contributions and commitments as well as the training process prior to the inference process, one more experiment has been

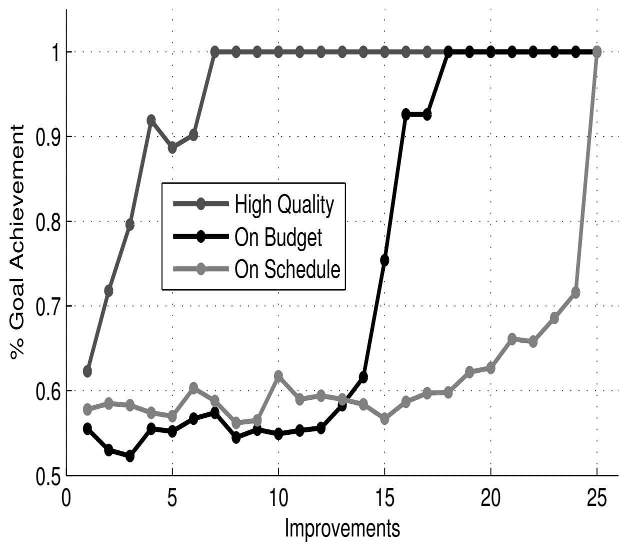
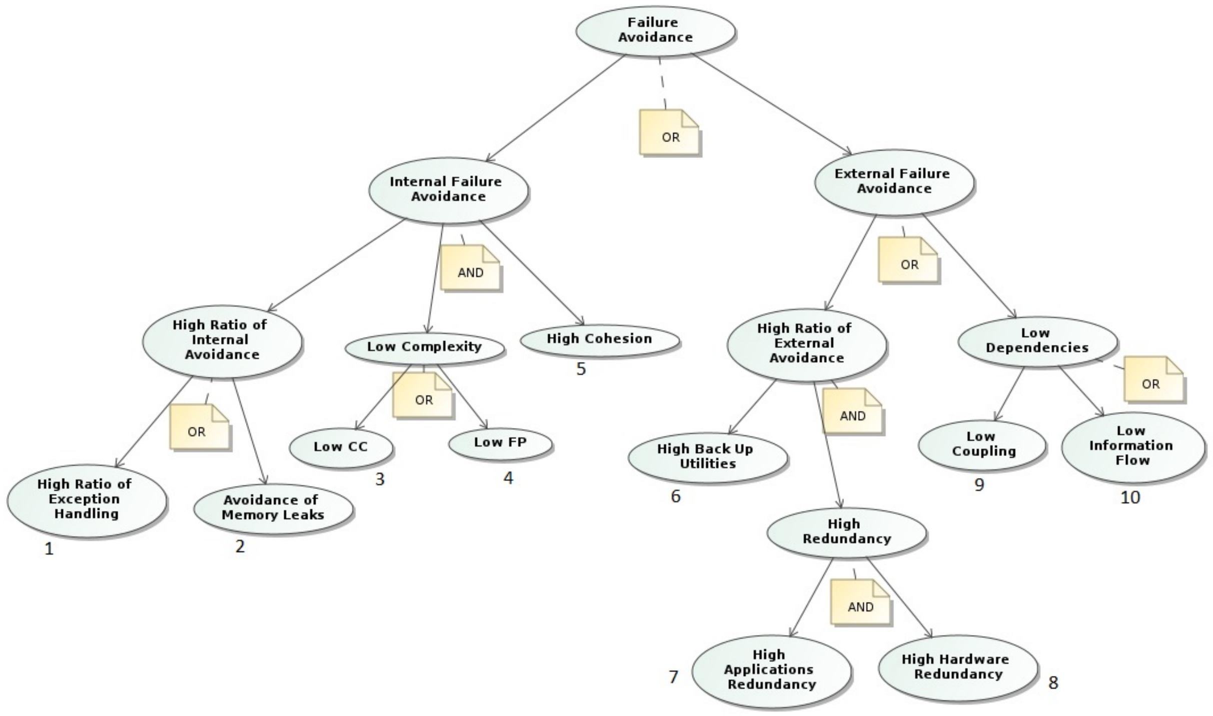


Figure 6.3: Changes in Probabilities Calculated for Goals "Task On Budget", "Task On Schedule" and "High Task Quality" Against Changes in Sub-goals.

conducted. For this, a hypothesis goal model regarding *Failure Avoidance* has been constructed, which concentrates on code related aspects of a task and whether these aspects demonstrate a flawless and expected behavior. Therefore, this can allow us to make deductions in terms of the fault prediction ability of the system. The hypothesis goal model constructed for the purposes of this experiment, is simpler and constituted by a sole goal tree, the root node to be validated is *Failure Avoidance*. Additionally, no contributions or commitments have been defined so as to evaluate their importance towards accuracy of results. Finally, no training was held over the formulas deriving from the goal model in order to examine the importance of the training procedure before inferring the probabilistic rates for the model.

The hypothesis goal model defined for this experiment, in the form of a simple AND/OR goal tree, can be seen in Figure 6.4 below.

In order to define to goal nodes of the AND/OR tree along with the type of relationships, the ISO/IEC TR 9126-2 [49] and ISO/IEC TR 9126-3 [50] were consulted. These two standards provided us with information over the goals included in the tree as well as the metrics and thresholds corresponding to the leaf nodes, which are utilized towards



Failure Avoidance
OR
Internal Failure Avoidance
External Failure Avoidance
AND
OR
High Ratio of Internal Avoidance
Low Complexity
High Cohesion
5
High Ratio of External Avoidance
Low Dependencies
OR
OR
OR
AND
High Ratio of Exception Handling
1
Avoidance of Memory Leaks
2
Low CC
3
Low FP
4
High Back Up Utilities
6
Low Coupling
9
Low Information Flow
10
High Redundancy
AND
7
High Applications Redundancy
High Hardware Redundancy
8

Figure 6.4: Failure Avoidance AND/OR Goal Tree

reasoning. For better understanding, a brief overview of the goal model will be given including a complete list of the nodes with brief explanations of the symbols:

Failure Avoidance

Failure Avoidance: This node is an OR decomposition and denotes that the system presents properties that enable it to avoid failure i.e. ensure error proneness.

Internal Failure Avoidance: This node is an AND decomposition and specifies that the system can avoid internal failure.

External Failure Avoidance: This node is an AND decomposition and specifies that the system can avoid external failure.

High Ratio Of Internal Avoidance: This node is an OR decomposition and denotes that the system presents a high ratio of test cases avoiding internal failure against the total number of test cases executed.

High Ratio Of Exception Handling: This node is a leaf node and indicates that the project presents a high ratio of exceptions handled against the overall number of exception appearing.

Avoidance Of Memory Leaks: This node is a leaf node and denotes that the system presents avoidance of memory leaks.

High Ratio Of External Avoidance: This node is an AND decomposition and denotes that the system presents a high ratio of test cases avoiding external failure against the total number of test cases executed.

High Back Up Utilities: This node is a leaf node and implies that the system has a large number of back up utilities to support its modules and components.

High Redundancy: This node is an AND decomposition and implies that the system has duplicates for many of its components which assists in avoiding failure.

High Applications Redundancy: This node is a leaf node and indicates that the system has various duplicates of its applications in order to ensure a non seizing functionality by the occurrence of an error.

High Hardware Redundancy: This node is a leaf node and indicates that the system provides various duplicates of its hardware components and modules in order to avoid an operation halt in case of an error.

Low Dependencies: The node is an OR decomposition and denotes that the source code of the various components, modules or other structures of the system, present low dependencies among one another.

Low Coupling: This node is a leaf node and denotes that the system presents low coupling, that is, low dependencies among its different modules and components.

Low Information Flow: This node is a leaf node and indicates that the modules and components of the system present low need of exchanging information frequently.

Low Complexity: This node is an OR decomposition and specifies that the source code and architecture of the system present low complexity.

Low C.C.: This node is a leaf node and indicates that the hardware architecture of the system presents low cyclomatic complexity.

Low F.P.: This node is a leaf node and indicates that the system presents low functional points, that is, the system provides a low amount of business functionality to a user.

High Cohesion: This node is a leaf node and denotes that the system presents high cohesion, that is, the functionality expressed by the source code of a software module, is strongly-related.

In order to conclude to some deductions over the extent to which the extensions we've imposed in the AND/OR Goal Tree model using the notions of contributions and commitments, as well as over the ability of the system to perform fault prediction, we repeated the experiment described in the **Stability** paragraph. That is, we chose the test case showing the worst characteristics, i.e. the one for which all leaf nodes are not verified, and made gradual improvements by satisfying one additional leaf at each execution in the order shown in Figure 6.4. The results for the probability of occurrence for failure avoidance for all test cases can be seen in Figure 6.5.

As it can be seen in Figure 6.5, the system maintains its stability, regardless of the definition of contributions and commitments and whether the goal model has undergone a training process. That is deducted from the fact that the probability rate gradually increases as more goals represented by leaf nodes of the tree are satisfied, which induce the satisfaction of the intermediate sub-goals and, finally, the root goal of the tree through AND/OR relationships. However, it can be seen that the accuracy decreases with the absence of contributions and commitments and, especially, with the absence of training prior to the inference process, since the probability rate inferred for the worst test case is 80% which is very high. This is caused mostly by the lack of training. The training session

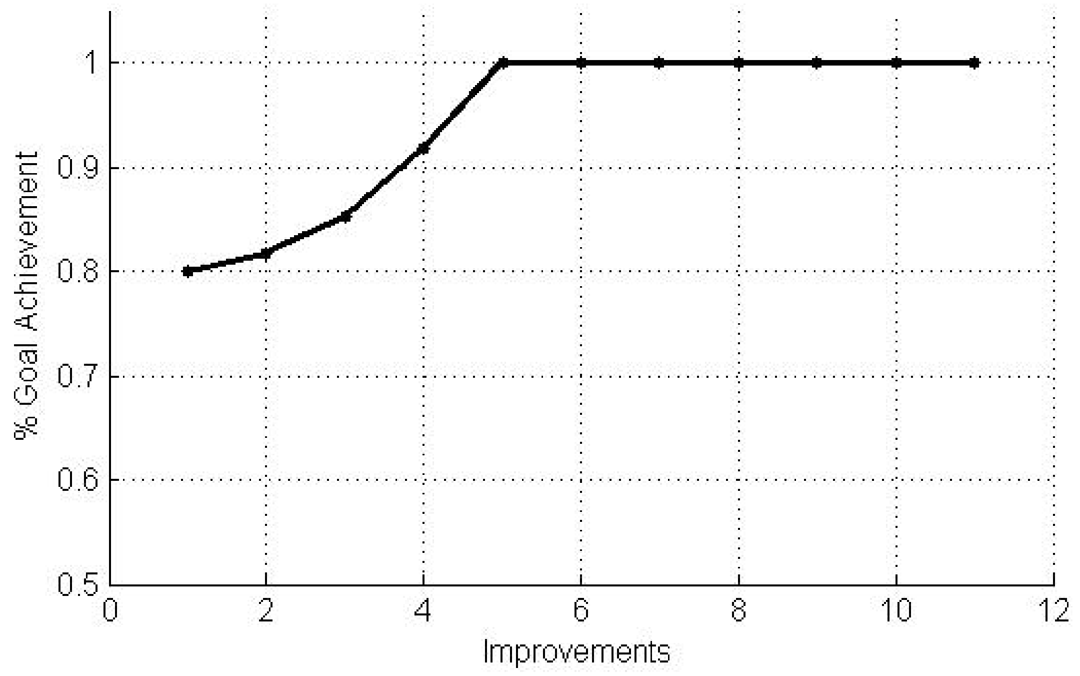


Figure 6.5: Failure Avoidance Results

prior to the inference assigns weight values to all formulas according to the set of probable worlds used as test cases for the training. The weights resulting from this procedure reflect the probability of satisfaction for each formula of the knowledge base in such a manor that it tends more to reality. Consequently, this fact also results to a more accurate inference procedure in terms of the level of probability rate assignments for the goals of the model since the system has been trained in order to reflect real test case scenarios.

6.3 Threats to Validity

It is important to note that there are certain threats to validity concerning this study which should be taken into consideration. First and foremost, the definition of the goal trees and modeling the project management goals has a degree of subjectiveness.

To one extent, this subjectiveness was caused by the fact that the construction of the goal trees was influenced by the type of the data available for use, in order to train our system satisfactorily. This fact can create implications towards the generalization of the reasoning analysis, since it is probable that the goals and sub-goals to be achieved in each goal tree will not reflect accurately or adequately the users' needs and requirements. In addition to this, it is also probable that the various goals and sub-goals needed to be fulfilled by real life industry software projects, might have dependencies from one to another, in a more complex level than the one assumed, while defining commitments and contributions between goals in the way described in the previous sections. In order to overcome these issues, we have attempted to add a degree of objectiveness to our definitions by taking two initiatives. The first is that we based the leaf node selection on the KPI library [42]. The KPIs included in the KPI library are metrics defined, in order to represent the current trends in important factors taken into consideration by today's industry, while conducting risk assessment. The second is that we based the intermediate node selection on risk factors that are mentioned in the literature [64, 28] as common to risk assessment procedures. Therefore, these choices increase the credibility of our goal modeling and ensure that the goal trees used for the analysis will have a practical impact.

Another important point potentially threatening the validity of the experiment is the fact that the data we used towards the training of our system and the testing of our analysis derives from a single business partner, and therefore, this can impede the generalization of the results throughout different sources of data. More specifically, one can argue that the form and structure, as well as the values these data have, may contribute beneficially towards good results of our reasoning, thus, a representative outcome of our analysis cannot be obtained. For this, it should be noted that the data we used, contained information regarding measurable project aspects, like the actual cost or overtime work hours, which are typically measured by large industrial organizations. Therefore, the data set provided by other sources is not expected to vary significantly as to the type of measurable quantities included.

Last, one more important threat towards the generalization of our technique, is the choice of thresholds taken, in order to accept or deny information deriving from the project data repository. The thresholds chosen so as to test our reasoning analysis, were based on the magnitude of values that the data set we had in our disposition, provided. In a more general sense, when applying our method over various data sets deriving from different sources of information, it is crucial to take into consideration, that the thresholds need to be in agreement with the magnitude of the values of each data set tested. The policies and demands of each industrial organization from which the data set under analysis is taken, will also play a great role towards setting the correct threshold for each metric. Therefore, it can be understood that the choice of thresholds made, in order to assess our reasoning framework, does not violate the correctness of the obtained results.

Chapter 7

Conclusion

This thesis focused on the problem of discovering alternative ways to provide assistance to Project Management as well as Risk and Quality Assessment operations, towards meeting the standards that the constantly increasing number of requirements dictate. For this, it is imperative to explore techniques model and denote project management as well as risk and quality project and product attributes, to devise techniques to mine project data repositories and, finally, to develop or adopt reasoning techniques so that project management as well as risk and quality models can be validated.

More specifically, the objective of this thesis was, first, to develop generalized modeling methods for project related information deriving from project data repositories, which would provide flexibility in terms of allowing successful data mining and representation regardless of database schema variations; second to analyze this information, in order to make quality and risk related deductions, that could potentially assist project managers and team leaders in making beneficial decisions towards improving their software or avoiding error appearance; third to investigate over possible modeling choices for common risk factors and quality prediction policies and, specifically, through an extended AND/OR goal tree representation via agent and commitment interoperability; and, finally, forth to utilize reasoning that yields results that hold within a degree of certainty especially in the presence of partial or incomplete information. For this, the Markov Logic Network statistical reasoning was utilized.

The framework has the ability of connecting with various databases that store project related information and extracting data, in order to perform inferences upon different data pertaining to software projects. This was achieved in an extensible manner, which allows the analysis process to remain schema independent, i.e. it permits data extraction regardless of the database schema of the mined repository. This was made possible by providing the user the freedom of choice upon defining the configuration parameters of the extraction and analysis, through the modeling decision of the goals to be fulfiled. In a nutshell, the thesis addresses the following issues:

1. **Architecture:** The development of this framework was based on the BlackBoard architecture style and a combination of multiple architecture design styles for the periphery components of the framework. This choice of architecture design aimed to offer flexibility and extensibility to the system.

2. **Modeling Analysis Objectives:** The use of the AND/OR goal trees annotated with the concepts of contributions and commitments to model goal inter-dependencies, as a modeling structure towards presenting risk avoidance and quality prediction factors, enables the users to define their own project data analysis logic, according to their preferences and requirements. The benefit of the chosen structure is the ease of representation of each goal model node through a First Order Logic expression, that can be transformed into Conjunctive Normal Form (CNF) in a straightforward manner. The resulting set of CNFs can be annotated with weights and be verified through a verifier component which, for the purposes of this study, is the Alchemy tool for conducting MLN reasoning analysis [37]. Therefore, in order to accomplish this probability rate assignment to the aforementioned policies, it was imperative to define a goal domain model which would provide the necessary modeling means that would facilitate the achievement of this target. For this, the Hypothesis Domain Model defined in Chapter 4, exploited the flexibility offered by the AND/OR Goal Tree representation structure, in order to effectively model the various goals and their relationships. The extensions made in order to incorporate the notion of contributions and commitments between goals and sub-goals, also played a very important part, since they reflect the complexity of dependencies existing between various goals, which cannot always be represented by a simple conjunctive or disjunctive relation.

3. **Mining and Reasoning on Project Data Repositories:** The Markov Logic Networks theory was adopted, in order to impose softer constraints to the first - order logic world of the goal model under analysis, and make deductions over the fulfilment of the goals under scope, not by accepting or denying them in an absolute way, but rather by inferring a probabilistic rate to each goal's achievement in the

world state defined by the goal model and the project data mined from the database under consideration.

4. **Evaluation of the Framework:** MLNs have the ability to reason upon those CNFs, by assigning a level of certainty to each, that denotes the confidence by which a predicate holds, given a set of constraints and a First Order Logic Knowledge Base. In order to evaluate the results of this method, we have chosen to encode quality assessment policies and common project goals set by development teams, that stem from the literature and are enforced through the use of KPI metrics within the KPI library. Through the development of this framework, this study aimed at handling issues such as possible application ways of Business Intelligence methods into Software Engineering Management and Quality Control, towards the accomplishment of a dynamic quality check based, on information mined from databases regardless of their schema structure. Furthermore, the study evaluated whether such approaches of business data modeling and analysis could provide project managers and team leaders with the necessary tools, in order to perform evidence based fault prediction over the set of goals they have set for achievement and incorporate the results into a more effective risk assessment process, that would potentially lead to higher quality products. For this, various fields were investigated such as the first - order logic theory, the Markov Logic Networks theory, the AND/OR Goal Trees theory, the agent and role interaction theory through contributions and commitments, as well as various sources for quality assessment standards, such as the KPI library, the ISO standards e.t.c.

7.1 Future Work

As mentioned above, the framework was developed so that it is extensible and flexible. Therefore, its actual structure enables various future additions and optimizations. One necessary optimization would be to create a friendly User Interface, so that the framework can be completely end - user, and to fulfil its purpose as an independent and autonomic system, apart from the simulation within this work.

Another possible extension for this work, is the expansion of the quality assessment models upon which the analysis is held. For the purposes of this thesis and the creation of a pilot prototype, the goal models defined as the set of quality policies for analysis and verification, were based on the investigation of common desired such goals on the part of developing teams, and their evaluation and verification was attempted through the application of KPI metrics that provided us with quantitative measurements for these goals. However, in the future, it is imperative that the developing teams should be given the flexibility to not only set their own metrics and assessment methods for the already defined and available goal models, but to also set their own goals and inter-relationships, in order to conclude to entirely custom results and deductions over the quality of their software, which will be adjusted specifically upon their requirements and criteria. To this direction, the Business Intellegence area of *Profiling* can prove very useful and, thus, should be looked into.

A second area of future work is to devise techniques to define complete models and, consequently, to evaluate the completeness of such goal models against specific standards or guidelines such as the ISO/IEC TR 9126, the CMMI or an industry accepted KPI library.

Within the set of other possible extensions for this work, we can also include the exploration of other techniques other than the Markov Logic Networks theory elaborated in this study. For the purposes of this thesis, the MLN theory was preferred over plain first-order logic, because the first provides the benefit of assigning truth values to all possible occurrences of a goal, with a level of certainty, as opposed to the latter which specifically accepts or denies a goal. However, since in the field of Software Engineering, the requirements for software products are increasing continuously, the goals for achievement desired and the various possible implicating risk factors are becoming more complex. Therefore, the simple acceptance or denial of a goal appears incapable to model adequately such a phenomenon. Apart from the MLN theory though, there are many more theories that investigate the various means of reasoning and probabilistic inferencing upon a set of goals. Such examples are the Neural Network theory, the Dempster-Shafer theory, the Bayes theory e.t.c. Therefore, an idea that could have potential for the future, would be to research whether any of these theories or even their combination could result to better and more accurate results.

Finally, other next steps for future research would include, the completion of the framework development as a stand - alone application and its integration to data collection tools and software, in order to provide useful insights and comprehensive perspectives to developing teams over the state of their projects and to perhaps assist them towards making project related decisions, that will lead to better quality software, while simultaneously and optimally reducing the cost, time spent and work effort.

References

[1] Abraham Bernstein and Jiwen Li. From active towards interactive learning: using consideration information to improve labeling correctness. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '09, pages 40–43. ACM, 2009.

[2] Zhibao Wang Bilong Wen, Qing Shao. Integration enterprise process metrics model and information model based on semantics. In *Proceedings of the 2009 WRI World Congress on Software Engineering*, volume 4, pages 34–37, 2009.

[3] Y. Brun and M.D. Ernst. Finding latent code errors via machine learning over program executions. In *26th International Conference on Software Engineering, 2004. ICSE 2004. Proceedings.*, pages 480 – 490, 2004.

[4] Palo Alto Burton H. Lee, Standford University. Using bayes belief networks in industrial fmea modeling and analysis. In *Annual Reliability and Maintainability Symposium*, pages 7 – 15, 2001.

[5] A. Gourley et al. C. Bird. Mining email social networks. In *Proceedings of the Third International Workshop on Mining software repositories*, pages 137–143. ACM, 2006.

[6] Amit K. Chopra, Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos. Modeling and reasoning about service-oriented applications via goals and commitments. In *Proceedings of the 22nd international conference on Advanced information systems engineering*, CAiSE'10, pages 113–128. Springer-Verlag, 2010.

[7] Amit K. Chopra, Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos. Reasoning about agents and protocols via goals and commitments. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 457–464, 2010.

[8] Amit K. Chopra, John Mylopoulos, Fabiano Dalpiaz, Paolo Giorgini, and Munindar P. Singh. Requirements as Goals and Commitments too. In *Intentional Perspectives on Information Systems Engineering*, chapter 8, pages 137–153. Springer, 2010.

[9] Magnus C. Ohlsson Claes Wohlin, Martin Host. Understanding the sources of software defects : A filtering approach. In *Proceedings of the 8th International Workshop on Program Comprehension*, pages 9–17, 2000.

[10] Pedro Domingos. Real-world learning with markov logic networks. In *Machine Learning: ECML 2004*, volume 3201 of *Lecture Notes in Computer Science*, pages 17–17. Springer Berlin / Heidelberg, 2004.

[11] Dr. James D. Palmer Dr. Joseph J. Romano. Tbrim : Decision support for validation/verification of requirements. In *1998 IEEE International Conference on Systems, Man, and Cybernetics*, volume 3, pages 2489 – 2494, 1998.

[12] Yong Hu Xiangzhou Zhang Xin Sun Mei Liu Jianfeng Du. An intelligent model for software project risk prediction. In *2009 International Conference on Information Management, Innovation Management and Industrial Engineering*, volume 1, pages 629–632, 2009.

[13] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *33rd International Conference on Software Engineering (ICSE), 2011*, pages 181 –190, 2011.

[14] J.S. Collofello D.X. Houston, G.T. Mackulak. Stochastic simulation of risk factor potential effects for software development risk management. *Journal of Systems and Software*, 59(3):247–257, 2001.

[15] A. Lucia et al. Information retrieval methods for automated traceability recovery. In *Software and Systems Traceability*, pages 71–98. Springer London, 2012.

[16] Aman Kumar et al. Enterprise interaction ontology for change impact analysis of complex systems. In *Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference*, pages 303–309, 2008.

[17] I. Mistrik et al. *Collaborative Software Engineering.* Springer Verlag, 2010.

[18] LI Jun et al. Development of the decision support system for software project cost estimation. In *2008 International Symposium on Information Science and Engieering*, pages 299–302, 2008.

[19] Pankaj Jalote et al. Quantitative quality management through defect prediction and statistical process control. Technical report, 2000.

[20] Shahrouz Moaven et al. A decision support system for software architecture-style selection. In *Sixth International Conference on Software Engineering Research, Management and Applications*, pages 213–220, 2008.

[21] Steven Euijong Whang et al. Indexing boolean expressions. *VLDB*, pages 37–48, 2009.

[22] T. Wolf et al. Mining task-based social networks to explore collaboration in software teams. *IEEE Softw.*, 26(1):58–66, 2009.

[23] The Eclipse Foundation. Eclipse modeling framework (emf) [online]. `http://help.eclipse.org/galileo/index.jsp`.

[24] M. Gegick, P. Rotella, and Tao Xie. Identifying security bug reports via text mining: An industrial case study. In *7th IEEE Working Conference on Mining Software Repositories (MSR), 2010*, pages 11 –20, 2010.

[25] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, 26(7):653 –661, 2000.

[26] F J Groen and C Smith. *Concept for the NASA risk and reliability data collection and analysis environment*, pages 134–139. 2009.

[27] J. Mylopoulos H. Zawawy, K. Kontogiannis and S. Mankovskii. Requirements-driven root cause analysis using markov logic networks. In *Proceedings of the 24th International Conference on Advanced Information Systems Engineering*, 2012.

[28] W Han and S Huang. An empirical analysis of risk components and performance on software projects. *Journal of Systems and Software*, 80(1):42–50, 2007.

[29] I. Herraiz, J.M. Gonzalez-Barahona, and G. Robles. Forecasting the number of changes in eclipse using time series analysis. In *Fourth International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR '07.*, page 32, 2007.

[30] Hooman Hoodat and Hassan Rashidi. Classification and analysis of risks in software engineering. *Engineering and Technology*, (56):446–452, 2009.

[31] Dr. John Hunt. Blackboard architectures. Technical report, 2002.

[32] D. Damian I. Kwan. A survey of techniques in software repository mining. Technical report, Software Engineering Global Interaction Laboratory, University of Victoria, 2011.

[33] Ivan J. Jureta, Stéphane Faulkner, and Pierre-Yves Schobbens. Allocating goals to agent roles during mas requirements engineering. In *Proceedings of the 7th international conference on Agent-oriented software engineering VII*, AOSE'06, pages 19–34. Springer-Verlag, 2007.

[34] Sunghun Kim, E.J. Whitehead, and Yi Zhang. Classifying software changes: Clean or buggy? *Software Engineering, IEEE Transactions on*, 34(2):181 –196, 2008.

[35] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485 –496, 2008.

[36] Ronald R. Yager Liping Liu. *Classic Works of the Dempster - Shafer Theory of Belief Functions*. Springer, 2008.

[37] Pedro Domingos Marc Sumner. The alchemy tutorial [online]. `http://alchemy.cs.washington.edu/tutorial/tutorial.pdf`.

[38] Pedro Domingos Matthew Richardson. Markov logic networks. Technical report, 2004.

[39] Robert G. Mays. Applications of defect prevention in software development. *IEEE Journal on Selected Areas in Communications*, 8:164 – 168, 1990.

[40] Andrew Meneely, Laurie Williams, Will Snipes, and Jason Osborne. Predicting failures with developer networks and social network analysis. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 13–23. ACM, 2008.

[41] Ashok Sontakke Meng Li, He Xiaoyuan. Defect prevention : A general framework and its applications. In *Proceedings of the Sixth Conference on Quality Software*, pages 281–286, 2006.

[42] Mirror42. Kpi library [online]. `http://kpilibrary.com/`.

[43] S. Morisaki, A. Monden, T. Matsumura, H. Tamada, and K.-i. Matsumoto. Defect data analysis based on extended association rule mining. In *Fourth International Workshop on Mining Software Repositories, 2007. ICSE Workshops MSR '07.*, page 3, 2007.

[44] N. Nagappan and T. Ball. Using software dependencies and churn metrics to predict field failures: An empirical case study. In *First International Symposium on Empirical Software Engineering and Measurement, 2007. ESEM 2007.*, pages 364 –373, 2007.

[45] Inc. Object Management Group. Object constraint language (ocl) [online]. `http://www.omg.org/technology/documents/formal/ocl.htm`.

[46] Inc. Object Management Group. Omg's metaobject facility (mof) [online]. `http://www.omg.org/mof/`.

[47] Inc. Object Management Group. Xml metadata interchange (xmi) [online]. `http://www.omg.org/technology/documents/formal/xmi.htm`.

[48] Masao Ohira, Naoki Ohsugi, Tetsuya Ohoka, and Ken-ichi Matsumoto. Accelerating cross-project knowledge collaboration using collaborative filtering and social networks. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, 2005.

[49] International Standards Organization. Iso/iec tr 9126 - 2. Technical report, 2003.

[50] International Standards Organization. Iso/iec tr 9126 - 3. Technical report, 2003.

[51] P. Giorgini et al. P. Bresciani. TROPOS: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[52] C.R. Pandian. *Applied Software Risk Management: A Guide for Software Project Managers*. Auerbach/Taylor&Francis, 2006.

[53] Tivadar Papai, Parag Singla, and Henry Kautz. Constraint propagation for efficient inference in markov logic. In *Proceedings of 17th International Conference on Principles and Practice of Constraint Programming (CP 2011)*, number 6876 in Lecture Notes in Computer Science (LNCS), pages 691–705, 2011.

[54] Martin Pinzger, Nachiappan Nagappan, and Brendan Murphy. Can developer-module networks predict failures? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 2–12. ACM, 2008.

[55] Xiaoyan Gao Qingtian Han. Application of data warehouse techniques in enterprise decision support systems. In *9th International Conference on Computer - Aided Industrial Design and Conceptual Design*, pages 1116–1120, 2008.

[56] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[57] Radu Marinescu Robert Mateescu, Rina Dechter. And/or multi-valued decision diagrams (aomdds) for graphical models. *Journal of Artificial Intelligence*, pages 465–519, 2008.

[58] Carnegie Mellon University Software Engineering Institute. Capability maturity model integration (cmmi) [online]. `http://www.sei.cmu.edu/cmmi/`.

[59] Dieter Steinbauer. A repository and other tools in a commercial development center. In *Software Development Environments and CASE Technology*, volume 509 of *Lecture Notes in Computer Science*, pages 114–129. Springer Berlin / Heidelberg, 1991.

[60] R. Subramanyam and M.S. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: implications for software defects. *IEEE Transactions on Software Engineering*, 29(4):297 – 310, 2003.

[61] A. Telea and L. Voinea. Case study: Visual analytics in software product assessments. In *5th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009.*, pages 65 –72, 2009.

[62] S. W. Thomas. Mining software repositories with topic models. Technical Report 2012-586, School of Computing, Queen's University, 2012.

[63] S.W. Thomas. Mining software repositories using topic models. In *33rd International Conference on Software Engineering (ICSE), 2011*, pages 1138 –1139, 2011.

[64] Kenia P. Batista Webster, Kathia M. de Oliveira, and Nicolas Anquetil. A risk taxonomy proposal for software maintenance. In *Proceedings of the 21st IEEE International Conference on Software Maintenance*, ICSM '05, pages 453–461. IEEE Computer Society, 2005.

[65] A.T.T. Ying, G.C. Murphy, R. Ng, and M.C. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, 30(9):574 – 586, sept. 2004.

[66] Pinar Yolum and Munindar R Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. pages 527–534. ACM Press, 2002.

[67] H. Zawawy, K. Kontogiannis, and J. Mylopoulos. Log filtering and interpretation for root cause analysis. In *IEEE International Conference on Software Maintenance (ICSM), 2010*, pages 1 –5, 2010.

[68] T. M. Allen Zhiwei Xu Khoshgoftaar. Prediction of software faults using fuzzy nonlinear regression modeling. In *Fifth IEEE International Symposium on High Assurance Systems Engineering*, pages 281–290, 2000.

[69] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *International Workshop on Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007.*, page 9, 2007.