Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Περιβάλλον Εκτίμησης Ποιότητας Λογισμικού Βασισμένο στην Ανάλυση Δομικών και Σχεδιαστικών Στοιχείων του Έργου

Διπλωματική Εργασία

της

Ιωάννας Μαρίας Χ. Ατταριάν

Επιβλέπων: Κώστας Κοντογιάννης
Αν. Καθηγητής Ε.Μ.Π.

Εργαστήριο Τεχνολογίας Λογισμικού
Αθήνα, Ιούλιος 2010

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού

Περιβάλλον Εκτίμησης Ποιότητας Λογισμικού Βασισμένο στην Ανάλυση Δομικών και Σχεδιαστικών Στοιχείων του Έργου

Διπλωματική Εργασία

της

Ιωάννας Μαρίας Χ. Ατταριάν

Επιβλέπων: Κώστας Κοντογιάννης
Αν. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 16η Ιουλίου, 2010.

....	....	....
Κώστας Κοντογιάννης	Ιωάννης Βασιλείου	Γεώργιος Στάμου
Αν. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Λέκτωρ Ε.Μ.Π.

Αθήνα, Ιούλιος 2010

....
Ιωάννα Μαρία Χ. Ατταριάν
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Τεχνολογίας Λογισμικού

Ευχαριστίες

Θα ήθελα, αρχικά, να ευχαριστήσω θερμά τον επιβλέποντά μου Αναπληρωτή Καθηγητή Ε.Μ.Π. κύριο Κώστα Κοντογιάννη ο οποίο υπήρξε πολύτιμη βοήθεια για μένα. Με τις συμβουλές του και τις επισημάνσεις του, με καθοδήγησε σε όλη την πορεία κατά τη διάρκεια εκπόνησης της παρούσας διπλωματικής εργασίας. Μου μετέδωσε πάρα πολλές και χρήσιμες γνώσεις γύρω από τον τομέα της Τεχνολογίας Λογισμικού ο οποίος ήταν ακόμα πολύ άγνωστος για εμένα και η αστείρευτη διάθεσή του για δουλειά και το ασταμάτητο ερευνητικό του ενδιαφέρον υπήρξαν το έναυσμα και για μένα να ασχοληθώ ερευνητικά με τον τομέα της Τεχνολογίας Λογισμικού. Τον ευχαριστώ πάρα πολύ για όλη την υπομονή και την προσπάθειά του να με διδάξει και να μοιραστεί τις γνώσεις του μαζί μου.

Θα ήθελα, επίσης, να ευχαριστήσω τους φίλους και συμφοιτητές μου οι οποίοι, κατά τα πέντε χρόνια σταδιοδρομίας μου στο Ε.Μ.Π., υπήρξαν στήριγμα και συνοδηπόροι μου. Γελάσαμε, διανύσαμε δυσκολίες, δουλέψαμε μαζί και, προπάντων, μάθαμε πολλά ο ένας από τον άλλον μέσα από τις συζητήσεις μας και την ανανταλλαγή απόψεων, ακόμα και από τις αντιδικίες μας. Τους ευχαριστώ πολύ, λοιπόν, και τους εύχομαι καλή σταδιοδρομία από εδώ και πέρα, να επιτύχουν κάθε τους στόχο και να μη σταματήσουν να επιδιώκουν πάντα να μαθαίνουν από τους ανθρώπους γύρω τους όπως κάναμε ο ένας από τον άλλον αυτά τα χρόνια.

Θα ήθελα, τέλος, να ευχαριστήσω πολύ τους γονείς μου καθώς και την Κική και το Βασίλη που με μεγάλωσαν με αγάπη και υπομονή για να βρεθώ σήμερα εδώ που είμαι. Υπήρξαν στηλοβάτες για μένα στις πιο κρίσιμες στιγμές της ζωής μου τις οποίες με βοήθησαν να ξεπεράσω, και ήταν παρόντες σε κάθε χαρούμενη στιγμή της ζωής μου να με συγχαρούν. Τους είμαι ευγνώμων για όλα όσα μου πρόσφεραν αυτά τα χρόνια και για όλα όσα μου έμαθαν.

Περίληψη

Στη σημερινή εποχή, στον τομέα των Συστημάτων Εφαρμογών Λογισμικού παρατηρείται συνεχής αύξηση των απαιτήσεων για μεγαλύτερη λειτουργικότητα και, άρα, τα συστήματα λογισμικού γίνονται μεγαλύτερα και πολυπλοκότερα. Εξαιτίας αυτού του γεγονότος και καθώς η αγορά γίνεται όλο και περισσότερο ανταγωνιστική, ανακύπτει η ανάγκη των διαχειριστών συστημάτων και των επικεφαλών ομάδων να αποκτήσουν μία περισσότερο ολιστική οπτική και εποπτεία της κατάστασης του συστήματός τους και να αναγνωρίζουν εκ των προτέρων δυνητικούς κινδύνους και ελαττώματα που μπορεί να αντιμετωπίσουν. Οι ερευνητές και οι ακαδημαϊκοί εξίσου, έχουν αναγνωρίσει την ανάγκη εύρεσης νέων τεχνικών που στόχο θα έχουν τη στήριξη των λειτουργίων Διαχείρισης Λογισμικού για μεγάλα συστήματα λογισμικού παραθέτοντας κάποιας μορφής πρόβλεψη κινδύνων και ποιότητας με χρήση κατάλληλων αυτοματοποιημένων εργαλείων. Ο κύριος στόχος της διπλωματικής εργασίας είναι η διερεύνηση μεθόδων ώστε να γίνει επεξεργασία και ανάλυση δεδομένων προερχόμενων από Ολοκληρωμένα Αναπτυξιακά Περιβάλλοντα και άλλα εργαλεία Διαχείρισης Συστημάτων με σκοπό οι διαχειριστές συστημάτων και οι προγραμματιστές να αποκτούν μία ακριβή εκτίμηση της προόδου ενός συστήματος λογισμικού και να λαμβάνουν εμπεριστατωμένες αποφάσεις αναφορικά με το σύστημα που αναπτύσσουν, σχετικές με το κόστος και την ποιότητα. Προς αυτήν την κατεύθυνση, η κύρια ιδέα είναι να αναπτυχθεί ένα αυτόματο περιβάλλον - πλαίσιο το οποίο να μοντελοποιεί πολιτικές διαχείρισης και κινδύνους με τρόπο ευέλικτο και εύκολο και να εφαρμόζει μια προβλεπτική ανάλυση λαμβάνοντας υπόψην πληροφορία προερχόμενη από Ολοκληρωμένα Αναπτυξιακά Περιβάλλοντα, εργαλεία Διαχείρισης Συστημάτων Λογισμικού και αποθήκες δεδομένων.

Λέξεις Κλειδιά

αρχιτεκτονική υπηρεσιοκεντρικών συστημάτων, μηχανική οδηγούμενη από στόχο, πρόβλεψη σφάλματος, αποθήκη δεδομένων, δένδρο στόχων, Μαρκοβιανό Λογικό Δίκτυο

Abstract

As software applications increase in size, complexity and functionality, so does the complexity of the corresponding specification, design, implementation and, testing efforts. Furthermore, as the software market becomes more and more competitive, and time to market schedules shrink, the need arises for project managers and team leaders to obtain a more holistic view and perspective of the status of their project and 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 need for the development of new techniques aiming to support software management operations for large software projects by providing some form of risk and quality prediction utilizing appropriate automated tools. The main goal of this diploma thesis is the design of a framework in terms of a reference architecture and supporting tools to process and analyze data obtained from IDEs and other project management tools so that, project managers and developers can obtain an accurate assessment on the progress of a software project and make educated cost and quality related decisions on their projects. More specifically, the thesis presents a reference Blackboard architecture for such project analysis and evaluation systems and techniques that allow for information obtained from various tools to be modeled and stored in a centralized data repository, for management policies and risks to be modeled as Goal Trees, and for reasoning techniques to be applied so that policies and risks can be verified or denied with a level of confidence or probability assessment. The proposed environment has been applied for the evaluation of the quality of a system according to the ISO 9126 standard.

Keywords

service oriented architecture, goal driven engineering, fault prediction, data warehouse, goal tree, Markov logic network

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Introduction

During the past recent years, one of the focal areas in Software Engineering is to provide techniques, methods and tools to support the specification, design, implementation and testing of complex software systems. This has led to the production of integrated development environments (IDE) and Software Project Management tools. This has caused changes thoughout all the phases of the life cycle of large scale software projects. More precisely it has affected :

Initiation/Planning : Due to the constant increase of users' demands, developers are challenged to complete projects that fulfill more high - level goals. Thus, the initiation and planning of projects tends to become more time consuming and complex since it has to be in consideration of more parameters.

Requirements Gathering and Analysis : As a result of the above, the project development requires more sources. In addition to this, the analysis, that is the process of identifying faults in order to fix them, becomes more difficult since the source code fulfills more requirements and is more extensive.

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 models entail functions that relate to a larger and more complex number of operations.

Build/Coding : The build/coding phase also becomes more demanding since models which stem from a synthesis of many requirements and goals can be harder to develop correctly.

Testing : Taking all of the above into consideration, it is clear that the testing phase as well has become even more demanding. In order to test successfully larger volume of source code, resources and entities such as testers and test cases will have to be raised in size and complexity, a fact that also produces a significant rise in cost and time consumption.

Maintenance : A larger amount of high - level source code is more fault prone and can raise more difficulties in maintaining its flawless operability and may require more economic funds and human resources to ensure it.

Thus, as explained above, this increase in demands and complexity has negative results such as simultaneous increase in cost and time acquired to conclude development. This fact raises the need of fault prediction, as it can also be seen in [13, 12]. It can be easily understood that the development of a framework which can provide information whether a module or component of a project has high probability to present a certain type of error or fault can provide significant assistance to the developer towards preventing the occurrence of failures ([1]). This can eventually lead to cost reduction since the developer will be given the opportunity to reduce the probability of faults. It can also assist throughout all the phases of the life cycle by detecting possible modeling and planning flaws at each phase separately which also ensures the maximization of project management granularity.

1.1 Problem Description

The problem which this diploma thesis is attempting to propose a solution to, concerns the prediction modeling, planning and design of faults and flaws that may occur during the specification and implementation of a large software project. Due to the increased level of demands and considering that the market continues to grow more and more in a competitive manner, the need of a way to effectively run some risk and product quality prediction analysis so as to assist project managers in making correct decisions in order to prevent possible threats, is useful.

To this direction, we propose a framework whereby we assume hypotheses which can correspond to actual problems that can appear in projects. Consequently, the framework is used to validate or deny each hypothesis utilizing statistical reasoning. The idea of building hypotheses scenaria over probable risks and faults that can occur during the development and coding of large integrated systems, can provide a very useful concept towards a possible solution to the prediction analysis ([5]).

It would be also interesting to consider whether there is a way of modeling those hypotheses in a formal manner, as it was also introduced in [4]. In the field of Software Engineering, the requirements specification plays an important part in the development of a software project. Therefore, the hypothesis modeling should be also accomplished in a flexible manner which would be easily handled and extended in order to provide more flexibility to the developers by allowing them to set their own configurations according to their requirements and objectives.

Next, it would be wise to consider whether a way of verifying the hypotheses can be found so that each hypothesis can be valid with some percentage of certainty. It can be easily understood that developing a procedure which would result to an affirmative or negative answer on the matter of some risk occurring, can be rigid and error prone since no flaw can possibly be predicted with complete certainty.

Finally, it should be searched whether the results of the above procedure, if such can be found, reflect accurately scenaria of real cases of errors and flaws that can be detected in software projects using data provided by various sources and repositories such as Data Warehouses which have become and continue to be more and more common in the field of Data Mining ([21]). The invention of such procedure would only be useful if and only if it corresponds to real paradigms so that project managers and team leaders can utilize it as a tool to perform quality checks and obtain status progress reports on their projects which would be critical information in order to make correct decisions on the continuation of the development.

This diploma 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 Thesis Contribution

This diploma 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 which has the ability to validate fault hypotheses with a probabilistic measure depending on the project data it will receive. The BlackBoard architecture consists of a core component which collects all the updates and alterations occurring during the execution of the framework. It gathers all the data produced during the procedure or added externally by the user and are required for the current analysis in order to transfer to other components that request them. It also receives updates from all modules with the status of the system at all times in order to notify the appropriate modules and trigger the execution of their operation.

- Definition of a hypothesis domain model in order to define hypotheses which belong to various hypothesis categories and types and construct the corresponding AND/OR goal trees with the appropriate goal tree node annotations using the Goal Tree Domain Model which is a tool of Requirements Engineering. The AND/OR Goal Tree Model has already been used by the research community in modeling requirement specifications and provides a flexible and extensible means of representation of the hypotheses, therefore chosen for the purposes of this diploma thesis.

- Representation of the goal trees by projection to the first - order logic world which enables the use of Markov Logic Networks as a probabilistic validation method. The AND/OR Goal Trees are easily projected into first - order logic predicates that simulate the world in effect, due to their structure which already connects the subgoals by conjuctive and disjunctive relationships. This benefit of the AND/OR Goal Tree structure provides a straightforward way of transforming the goals and subgoals that constitute a hypothesis, into logical expressions based on *Boole Algebra* and, next, into conjuctive normal form which is also required for the construction of the appropriate for the hypothesis, Markov Logic Network.

1.3 Outline of the Thesis

The text of the diploma thesis will be structured as follows :
Chapter 2 : In this chapter, a list of the related work is presented. More specifically, the relative subjects covered in chapter 2 which were very useful for the conduction of this diploma thesis, are 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, the ISO standards ISO/IEC TR 9126 - 2 and ISO/IEC TR 9126 - 3 for the definition of the appropriate quality measures that a module needs to have in order to be accepted as fault free, the Markov Logic Network theory for the the probabilistic validation of each hypothesis, the Goal Tree Domain Model theory for the hypotheses modeling, 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 three domain models defined for the purposes of this diploma thesis, the **Hypothesis Domain Model**, the **Warehouse Domain Model** and the **Goal Tree Domain**

Model, are described in detail and each class is explained in terms of its attributes, operations and functionality.

Chapter 5 : In this chapter, the hypothesis analysis algorithms which were implemented, are described. Specifically, details are given for the **Hypothesis Selection** algorithm, the **Hypothesis Verification** algorithm and the **Markov Logic Network** based analysis.

Κεφάλαιο 2

Related Work

The work presented in this diploma 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 touches upon 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 MOF/XMI

The **Meta - Object Facility (MOF)** [15] 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)** [16] 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. 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 diploma 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) [7]. 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.

2.2 Quality Assessment Standards

A crucial point to the work elaborated for the purposes of this diploma 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. These are discussed below in more detail.

2.2.1 ISO International Standards

The International Organization for Standardization (ISO) is an international stardard setting organization which utilizes representatives from various national standardization organizations. Although it is considered a non - governmental organization, it has the ability of setting 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 entreprises to ensure the quality of their products and the integration of the operation environment.

This diploma thesis focused on more at two ISO quality standards and specifically, the ISO/IEC TR 9126 - 2 [17] and ISO/IEC TR 9126 - 3 [18]. There are also other studies focusing on the same subject, that were based on the ISO Standards ([19]). These ISO standars were useful to draft the logic by which a process or a product is validated for its quality in the proposed framework. T

2.2.2 CMMI

Another commonly used and acknowledged set of Quality Standards is the Capability Maturity Model Integration (CMMI) ([24]). 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 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.

2.3 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. Two reasoning frameworks studied in this diploma thesis, are the Markov Logic Networks theory and the Dempster - Shafer theory of Evidence. Other reasoning frameworks have also been applied for the development of decision support systems such as Fuzzy Inference ([23]), Function Point Analysis ([9]), Fuzzy Delphi ([9]), Fuzzy Nonlinear Regression Modeling ([26]), Artificial Neural Networks ([6]) and Support Vector Machines ([6]). The Markov Logic Networks theory was the one utilized in this work.

2.3.1 Markov Logic Networks

The *Markov Logic Networks* theory combines the benefits of Markov Netwoks along with the benefits of first order logic. As discussed in [11], Markov Network or Markov Random Field is a model of the joint distribution of a 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 :

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

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 diploma 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. 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 diploma thesis, all formulas and constraits 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 sizewise 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 diploma 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.

2.3.2 Dempster - Shafer theory of Evidence

The Dempster - Shafer theory, as it is analysed in [10], 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 shall use evidence that do not bear 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 [3] , 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 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 assign to every subset A of Θ the sum of the beliefs commited 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 diploma thesis.

2.4 Goal Trees

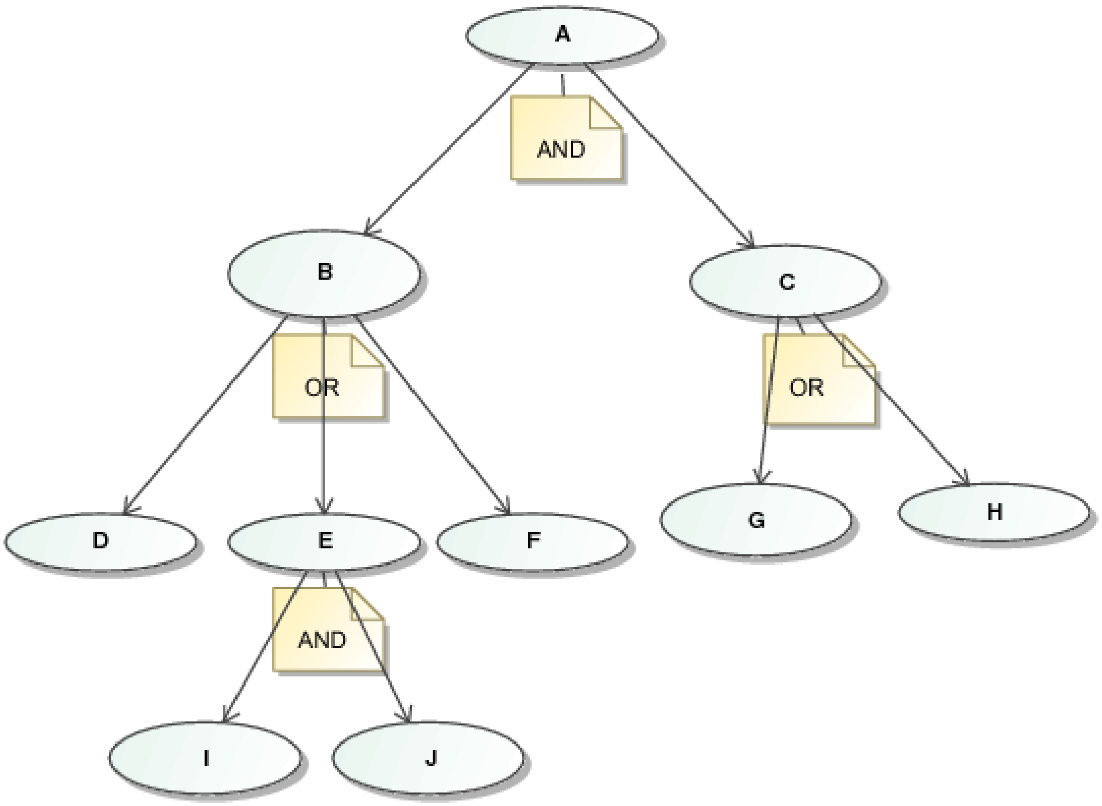
In the field of Software Engineering, a role of great significance for the development of a project is played by the operational requirement specifications. That is, the developer must specify the requirements of the project so that, under a certain input, its behavior considering the output, will be as expected.

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 subgoals 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, [22]. According to this model, a goal can be divided into subgoals which are represented as the children of the first. The same applies for each subgoal 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 his subgoals, i.e. his children nodes, are also satisfied. These are noted as **hard** goals since they demand all of their subgoals 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 subgoals, are connected by logical disjunction. Therefore, if at least one of the subgoals is fulfiled, that leads to their parent goal to be considered as been satisfied as well. These goals are called **soft** goals since they do not require that all of their subgoals are satisfied in order to validate as true, but even one of their subgoals 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 diploma 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, [25], 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 diploma thesis. In addition, they offer a great level of extensibility since with the annotations built and linked to each goal tree node, that is each subgoal, it can be understood 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 :



Σχήμα 2.1: Simple AND/OR Goal Tree Example

Κεφάλαιο 3

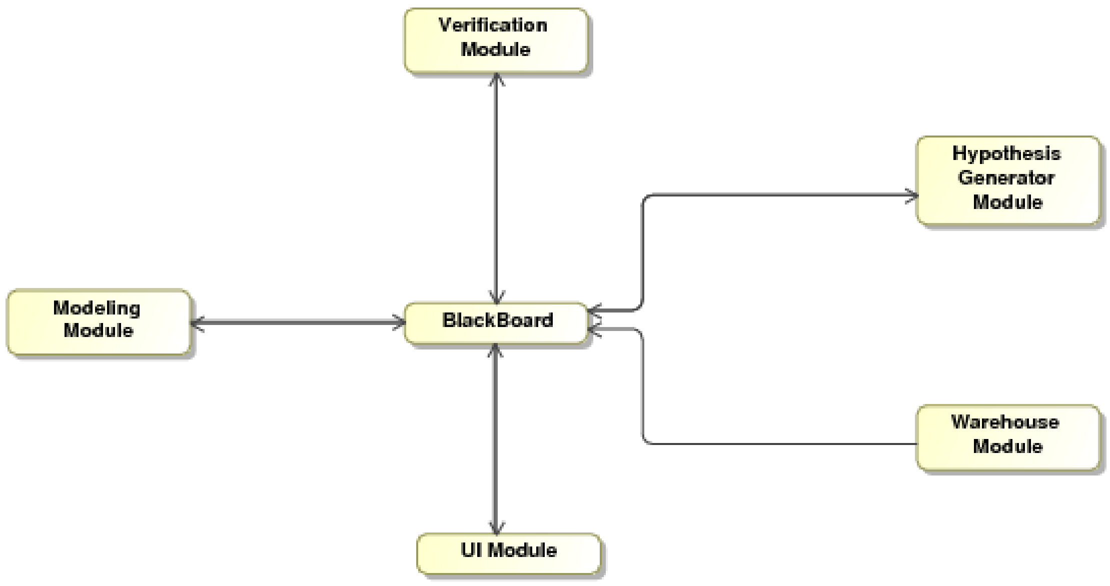
System Architecture

3.1 Architecture Description

3.1.1 General Overview

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

In the field of Software Architecture it is common that the best suitable architecture for an application is not always one individual architecture style but rather a mixture of different architecture styles combined in order to achieve optimal cohesion and coupling properties. The proposed system is based primarily on a BlackBoard style architecture with elements of Pub/Sub, 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 :



Σχήμα 3.1: Block Diagram of the System Architecture

As it is shown in the above diagram, the architecture consists of six basic modules. A brief description of the functionality of those modules is discussed below :

BlackBoard : This is the main module which stores the state of the operation or the system and all events generated by all other 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 which provides the necessary information provided by IDEs, case tools and Project Management tools which is required in order for the system to run the fault prediction process. It consists of an *IDE Container* with the IDEs from which data are extracted, a *Mediator*, an *Extractor*, a *Selector* and a *Data Pool* which contains the extracted data from the IDEs of the IDE Container.

Modeling Module : This is the module which models the hypotheses. That is, the modeling module requests information from the user for each hypothesis so as to model the hypothesis correctly. The user is asked to give the necessary thresholds and acceptable values for each property feature that plays a role in each hypothesis to be tested for validation. It consists of an *Editor* which provides the user the means to edit the hypothesis model, and the *Hypothesis Modeler* which adjusts the model parameters according to the user's choices.

Hypothesis Generator Module : This is the module which generates the hypotheses to be checked for validation. The hypothesis generator module represents each hypothesis based on an AND/OR goal tree model. It consists of the *Hypothesis Generator* which generates the hypotheses goal trees and the *Hypothesis DB Server* which plays the role of the *Hypothesis Pool*, that is, it contains all the hypothesis scenaria that can possibly hold.

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.

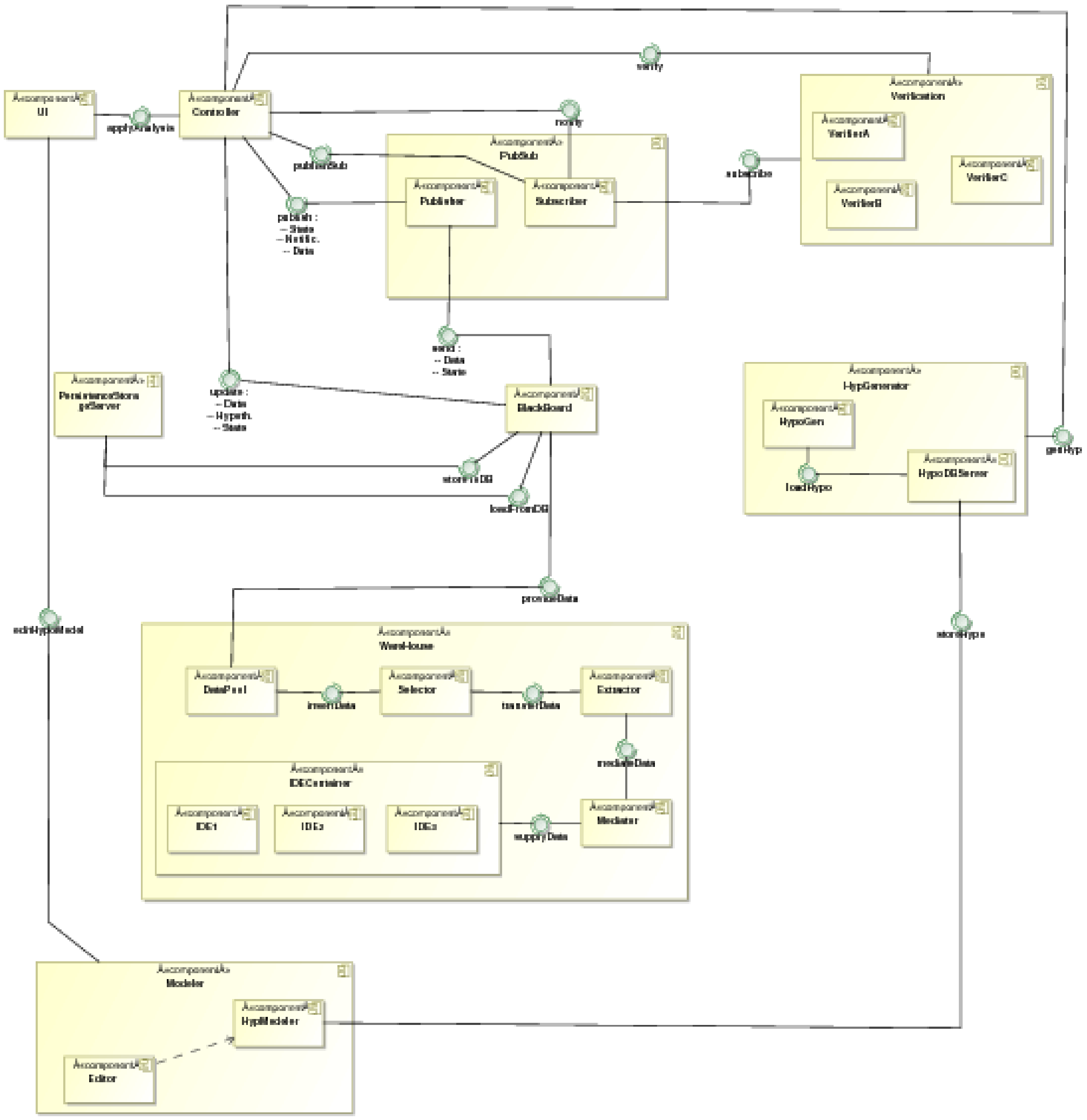
UI Module : This is the module which controls the communication between the user and the system. It consists of a *User Interface* through which the user can choose to insert the required information by the modeler so as to be applied to the analysis, or choose to store the results of the analysis held under certain parameters of his interest. It also consists of a *Persistance Storage DB* which is used to store the analysis results via the UI if the user desires so.

3.1.2 Component Description

In this section, we provide an overview of the architecture and discuss in more detail each component and its functionality along with the interfaces it provides and the services it offers. The component diagram of the system is illustrated in figure 3.2.

3.1.2.1 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 *IDE Container*, the *Mediator*, the



Σχήμα 3.2: Component Diagram of the System Architecture

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 IDE Container contains a number of IDEs for which the analysis will be held. The IDE Container is populated by the developers with the IDEs of their interest over which the analysis will be run. For this purpose, the Mediator provides this interface to the IDE Container in order to communicate and retract the information that needs to be mediated. Therefore, the interface *supplyData* is used by the IDE Container to supply the Mediator with the appropriate data.

mediateData() : The Mediator receives information provided by the IDE Container in order to mediate it to the Extractor for the extraction of the useful for the fault prediction analysis, data out of the raw data. This interface is provided by the Mediator exactly to satisfy this need of data transition to the Extractor. The Extractor, then, uses this interface in order to receive the appropriate data that it has to process so as to extract the appropriate information.

transferData() : The Selector is used to select the useful data for the analysis out of the data sent by the Extractor according to the user's demands, that is, depending on which type of hypothesis the user wishes to run the testing for. This interface is provided by the Extractor in order to permit to the Selector to select the data needed to run the analysis for the parameters set by the user.

insertData() : After it has selected the appropriate data, the Selector stores them into the Data Pool from where the data is now visible to the component's environment i.e. the other components of the system. This interface provided by the Selector allows the data storage to the Data Pool.

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 IDE Container plays the role of the back - end of the 3 - tier architecture as illustrated in figure 3.2. The IDE Container being the back - end of the multi - tier architecture, has the benefits of providing centralized data management, ensuring data integrity and security along with database 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 makes possible the communication of the back - end of the 3 - tier architecture (IDE Container) with the component environment, that is the other components of the system. 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.

3.1.2.2 Modeling Module

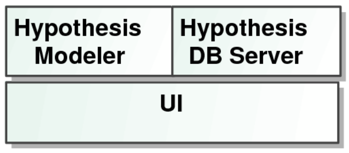
The *Modeling Module* is represented by the *Modeler component* in figure 3.2. The Modeler component is composed by the *Hypothesis Modeler* component and the *Editor* component. The interfaces and services provided by this module, are :

Interfaces/Services :

editHypoModel() : The Hypothesis Modeler is assigned to communicate with the Editor through the dependency that can be seen in figure 3.2, in order to accept the user's preferences concerning thresholds and indicative values for the property features tied to the hypotheses which were chosen to be tested, or even insert new hypotheses according to his demands and edit existing ones. Thus, the Modeler Module provides an interface to the User Interface of the UI Module in order to allow to the user through the User Interface to use the Editor and edit the hypothesis model by inserting new parameters to the system.

storeHypo() : The Hypothesis Modeler communicates through this interface with the Hypothesis DB Server in order to retrieve information on the hypotheses, the properties and the features to which the data inserted by the user, correspond and update the database with the new parameters and information added or altered by the user.

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 :



Σχήμα 3.3: Layered architecture of Modeling Module

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

3.1.2.3 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 which generates the hypothesis that will be tested for validation during the current session, and its corresponding goal tree. The *Hypothesis DB Server* is the database containing all possible effective hypotheses, that need be verified. 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 for the hypotheses which will be tested during the session according to the user's requests, the Hypothesis Generator Module is assigned to notify the Controller of the Verification Module about the event of goal tree construction completion so as to trigger the start of 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 hypotheses to be transfered to the Verification Module for validation.

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 by 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.

3.1.2.4 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 :

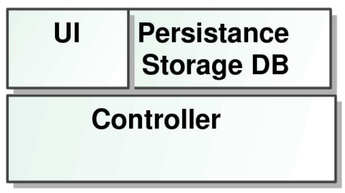
Interfaces/Services :

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 diploma 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.

loadFromDB() : This is the interface 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 is the interface 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.



Σχήμα 3.4: Layered architecture of UI Module

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.

3.1.2.5 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 independency 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 scaleability of the module can be increased since a certain amount of parallelization can be added to the verification process.

3.1.2.6 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.

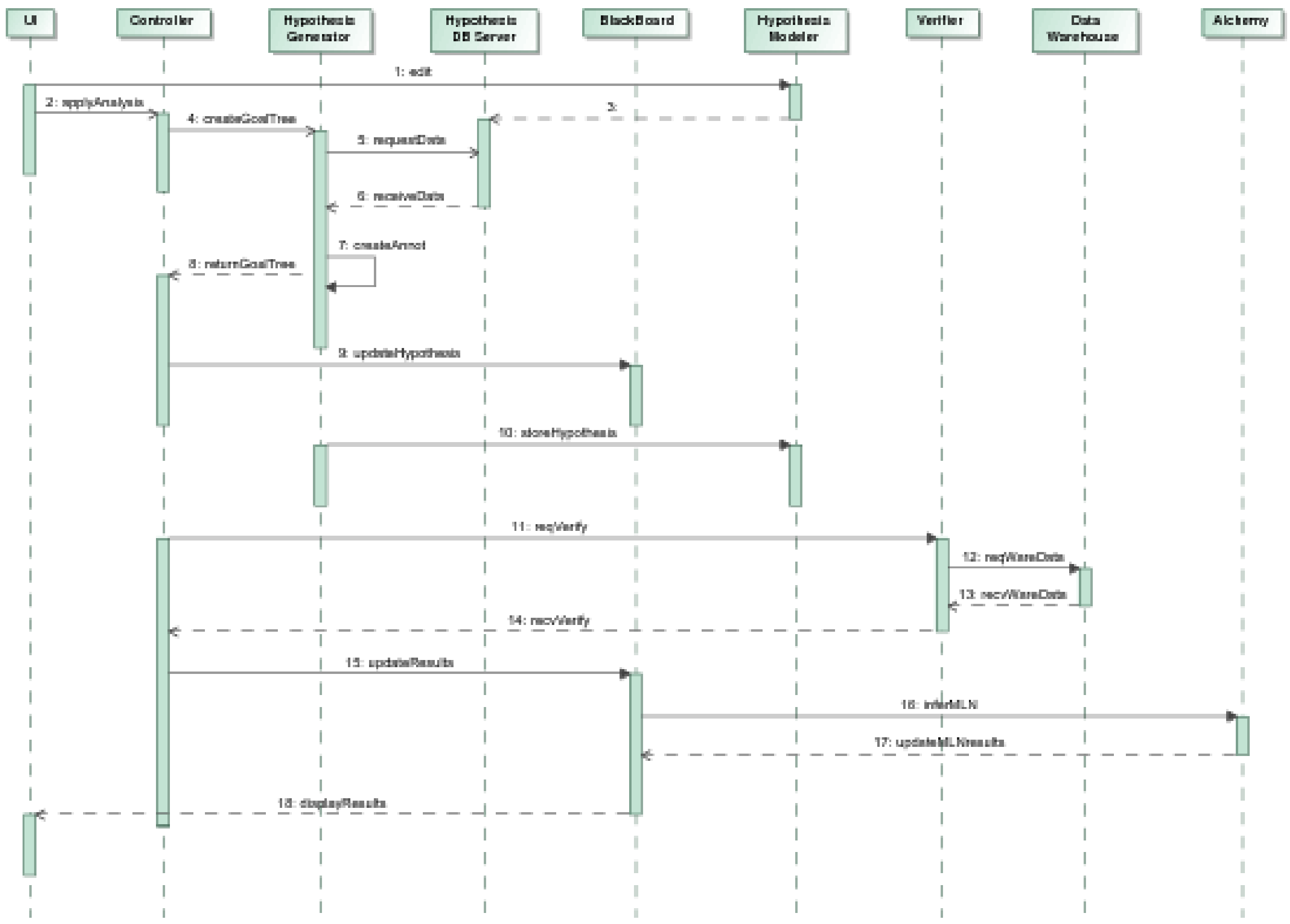
send() : --Data --State : Via this interface, the Publisher publishes supplementary data occurring during the analysis along with the state of the process, to the BlackBoard in order for the latter to notify other components and to synchronize the various modules.

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.

This is the core component that implements the BlackBoard architecture style of the proposed system. According to the BlackBoard architecture style ([8]), 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.



Σχήμα 3.5: Functionality Sequence Diagram of the system

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

1. *edit* : Through the User Interface of the UI Module the user/developer can edit the parameters of already existing simulations through the Hypothesis Modeler, in order to re-execute the simulation with different parameters.

2. *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.

3. *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.

4. *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.

5. *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.

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

7. *createAnnot* : This is an internal operation of the Hypothesis Generator. At this step of the procedure, the Hypothesis Generator adds the appropriate annotations at each node of the goal tree.

8. *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.

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

10. *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.

11. *reqWareData* : 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 Data Warehouse.

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

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

14. *updateResults* : The BlackBoard needs to be updated with the validation results right after they are concluded and transferred to the Controller.
15. *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.
16. *updateMLNResults* : The Alchemy tool must update the BlackBoard component with the resulting validation percentages of the MLN inference.
17. *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.

Κεφάλαιο 4

Domain Models

In this chapter the domain models developed for the purposes of this framework, are being described in terms of MOF class.

4.1 Warehouse Domain Model

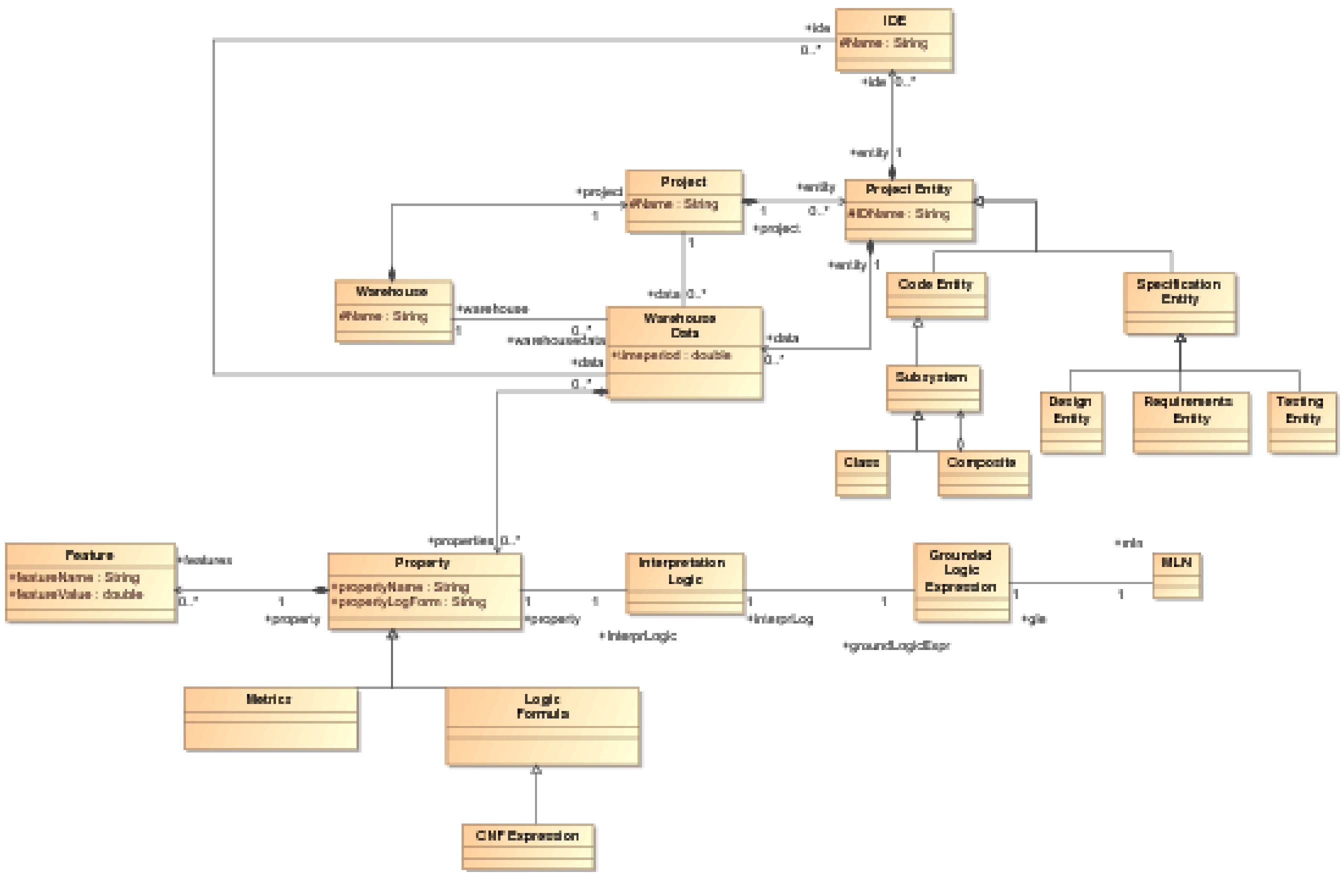
The first domain model that was developed, is the Warehouse Domain Model. The Warehouse serves the purpose of storing exemplary data concerning IDEs and/or their individual modules. These data are ideally provided by developers and are related with code, structure or development issues. Within this diploma thesis, the type of data included into the Data Warehouse Model as well as the type of properties of the hypotheses included into the Hypothesis Model, conform to the ISO Standards as it was mentioned earlier.

An overview of the Warehouse Domain Model can be seen in figure 4.1 below.

Continuing, let us describe the Warehouse Domain Model and the classes included in the model.

4.1.1 Warehouse Class

The Warehouse class represents the Warehouse entity, that is the Data Pool. It is the main class which contains all the data that developers have added concerning projects and IDEs for which they desire to execute the verification analysis. The Warehouse class owns an attribute under the name *Name* and of type String so that each instance of the Warehouse class will have a unique identifier to differentiate it from all the other instances of the same class. As it can be seen in figure 4.1, the Warehouse has a composition relationship under the identifier $+project$ of type *Project* with the *Project* class, that is each instance of the Warehouse class will contain one instance of the Project class which will be explained next. It should be noted that an instance of the Warehouse class can contain one sole instance of the Project class since it is assumed that a discrete warehouse will be constructed by the developer for each project he is working on. The Warehouse class is also associated with the *Warehouse Data* class. More precisely, it owns a $+warehousedata$ attribute of type *Warehouse Data* with multiplicity $0..*$ which indicates that each instance of the Warehouse class can associate with zero or many instances of the Warehouse Data class.



Σχήμα 4.1: Warehouse Domain Model

4.1.2 Project Class

The Project class represents the project that the user is developing and, thus, desires to run the analysis on. As it was mentioned previously, it is reminded that one warehouse corresponds to a specific project. Therefore, the data within the instance of the Warehouse class which contains the instance of the Project class will concern solely this individual project. The Project class owns an attribute named *Name* of type String which plays the role of the key identifier of different instances of the Project class. It is advised to initialize the Name attribute of each instance of the Project class with the name of the actual project it is referring to so as to avoid any cohesions. The Project class has a composition relationship with the *Project Entity* class of multiplicity $0..*$, that is each instance of the Project class contains zero or many instances of the Project Entity class. From this relationship one more attribute is added to the Project class, the $+entity$ attribute of type *Project Entity*. The Project class is also associated with the *Warehouse Data* class with multiplicity $0..*$. This adds an attribute named $+data$ of type *Warehouse Data* to the Project class. It is obvious that each instance of the Project class can be associated with many instances of the Warehouse Data class since a warehouse can contain a lot of information for a project about various fields.

4.1.3 Project Entity Class

The Project Entity class represents each individual part of a project on which we can obtain discrete data, independent from other entities. This class has an attribute named *IDName* of type String which is the primary key to discriminate instances of the class. As it is shown in figure 4.1, the

Project Entity class has a composition relationship with the *IDE* class of multiplicity $0..*$. That is, one instance of the Project Entity class can be associated with zero or more IDEs regarding that different IDEs can display partial similarities. This relationship is useful in order to present the direct linkage of each project entity with certain IDEs and, therefore, adds an attribute named $+ide$ of type *IDE* with multiplicity $0..*$ to the Project Entity class. The relationship of the Project Entity class with the Project class described previously also adds the attribute $+project$ of type *Project* with multiplicity 1 since various instances of the Project Entity class can be contained by the same instance of the Project class. The Project Entity class, also, has a composition relationship with the *Warehouse Data* class since data are added in the warehouse for each project entity individualy. This relationship adds an attribute $+data$ of type *Warehouse Data* and multiplicity $0..*$ to the Project Entity class. The Project Entity class displays dependencies with the classes *Code Entity* and *Specification Entity* which inherit the first. A Project Entity can either be a Code Entity or a Specification Entity. A Code Entity concerns an actual code segment of a project. A Specification Entity refers to non code related information on the project but rather refers to design, requirements and testing models. The *Code Entity* class has one subclass which inherits it, the *Subsystem* class. The Subsystem class has two subclasses that inherit it, the *Class* class and the *Composite* class. This means that a code entity can be a subsystem and each subsystem can be an individual class of the project or a composite which, as shown in figure 4.1, can be an aggregation of one or more classes and other composites iteratively. Therefore, the code entity can be an individual class or a more complex segment consisting of many classes. The *Specification Entity* class has three subclasses which inherit the first, the *Design Entity* class, the **Requirements Entity** class and the *Testing Entity* class. The Design Entity class is an entity of a project concerning its desing method, architecture e.t.c. The Requirements Entity class is an entity of a project concerning the requirements of the project. Finally, the Testing Entity class is an entity of a project concerning anything related to the testing process of the project.

4.1.4 IDE Class

The IDE class represents all the IDEs for which the warehouse can contain data. This class has an attribute named *Name* of type String that is used as a key to identify the IDEs. As it can be seen in figure 4.1, the IDE class is associated with the Project Entity class which adds an attribute $+entity$ of type *Project Entity* and multiplicity 1, to it. The IDE class is, also, associated with the Warehouse data class which adds another attribute $+data$ of type *Warehouse Data* and multiplicity $0..*$. This is interpreted as that one instance of the IDE class can be tied with multiple instances of the Warehouse Data class since for one IDE there can exist various data added within the Data Warehouse.

4.1.5 Warehouse Data Class

The Warehouse Data class represents the actual data within the Warehouse. This class has an attribute under the name *timeperiod* of type Double which is useful in the comparison of faults given their time period of appearance considering that it is more accurate to compare the number of faults within a period of time than the absolute number of faults. In addition, the Warehouse Data class is associated with the IDE class and, therefore, from this relationship it has gained another attribute $+ide$ of type *IDE* and multiplicity $0..*$ which is expected considering that the Warehouse data can correspond to multiple IDEs. The Warehouse Data class, also, is associated with the Warehouse class and, because of this, has the attribute $+warehouse$ of type *Warehouse* and multiplicity 1 since some instance of the Warehouse Data class can only belong to one instance of the Warehouse class. It is, also, associated with the Project class from which it gained another attribute to which was not of great significance to the further analysis, to assign a name. As mentioned previously, the Warehouse Data class has a composition relationship with the Project Entity class and, thus, the attribute gained is $+entity$ of type *Project Entity* and multiplicity 1 which indicates to which project entity each warehouse data belongs. Last, the Warehouse Data class has a composition association with the Property class and, therefore, an attribute $+properties$ of type *Property* and multiplicity $0..*$ since one instance of the Warehouse Data class can be tied with zero or many instances of the Property class.

4.1.6 Property Class

The Property class represents each property that can appear referring to a warehouse datum. This class has two attributes under the names *propertyName* and *propertyLogForm* of type String. The first plays the role of the primary key to differentiate each property and the second is the logical form of the property. Each instance of the class can be linked to zero or multiple instances of the Warehouse Data class. At the same time, the class has a composition association with the Feature class which attaches another attribute to it, the attribute $+features$ of type *Feature* and multiplicity $0..*$ which represents all the features that can belong to each property. The Property class is associated, also, with the Interpretation Logic class which adds one more attribute $+InterprLogic$ of type *Interpretation Logic* and multiplicity 1. It must be noted that the Property class has two subclasses which inherit the first, the *Metrics* class and the *Logic Formula* class. The Metrics class refers to the metrics defined by the features tied with each property. An example of such a metric would be the ratio of two features connected to a particular property. The Logic Formula class includes the logic formulas defined by the property name and each feature name. The Logic Formula class also has a subclass which inherits the first, the *CNF Expression* class, which represents all the logic formulas that are given in a conjuctive normal form expression.

4.1.7 Feature Class

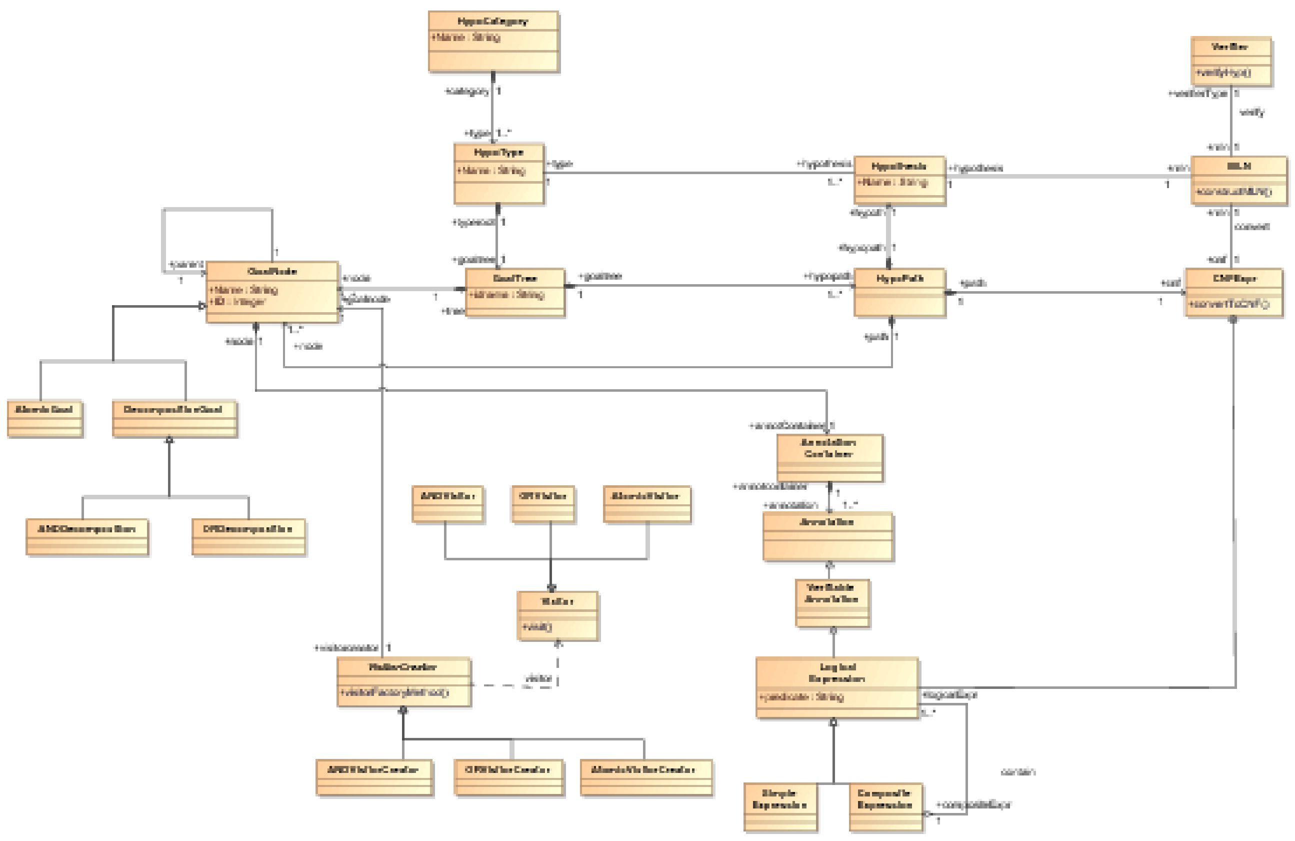
The Feature class represents each feature that can appear to belong to some property of a warehouse datum. This class has two attributes by names *featureName* of type String and *featureValue* of type Double. This first is the key to discriminate the different features and the second is the value for the specific instance of the class. In addition, the Feature class has a composition association with the Property class and, therefore, gains the attribute $+property$ of type *Property* and multiplicity 1 which indicates the property to which each feature belongs.

4.1.8 Interpretation Logic Class

The Interpretation Logic class represents the means of interpreting each property. For each property there exists only one way of interpretation and, thus, the class obtains one attribute $+property$ of type String and multiplicity 1. The Interpretation Logic class is also associated with the *Grounded Logid Expression* class which explains the existence of the attribute $+groundLogicExpr$ of type *Grounded Logic Expression* and multiplicity 1 since one grounded logic expression can occur from a certain interpretation logic.

4.1.9 Grounded Logic Expression Class

The Grounded Logic Expression class represents the grounded first - order logic predicate which will be used finally as input to Alchemy for the MLN inference. It is associated with the Interpretation Logic class which explains the gained attribute $+interprLog$ of type *Interpretation Logic* and multiplicity 1 and, also, with the *MLN* class which explains the second gained attribute $+mln$ of type *MLN* and multiplicity 1.



Σχήμα 4.2: Hypothesis Domain Model

4.1.10 MLN Class

The MLN class represents the inferred MLN by the Alchemy tool. Each grounded logic expression associates with one MLN, therefore their one - to - one correspondence that can be seen in figure 4.1, is justified. From this association, the class has as attribute the $+gle$ of type *Grounded Logic Expression* and multiplicity 1.

4.2 Hypothesis Domain Model

The second domain model that was built for the development of these framework, was the Hypothesis Domain Model. The Hypothesis Domain Model is used in order to model the different hypotheses into the form of a AND/OR Goal Tree. The AND/OR Goal Tree was the form of hypothesis representation utilized within the bounds of this diploma thesis because of the benefits it offers. First, it is a very flexible means of representation which makes it easier to traverse. Second, the AND/OR relationships between the nodes of the goal tree permit the direct and straightforward interpretation and translation of the nodes into first - order logic predicates which is the target of the Goal Tree traversal in order to feed the results to Alchemy as input to the MLN inference.

An overview of the Hypothesis Domain Model can be seen in figure 4.2 below.

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

4.2.1 HypoCategory Class

The HypoCategory class represents the hypothesis category in which the hypothesis falls into. The HypoCategory class has one attribute *Name* of type String that plays the role of the primary key in order to discriminize each instance of the class. 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. An example of a hypothesis category would be Fault Tolerance and some hypothesis types corresponding to it would be Failure Avoidance, Incorrect Operation Avoidance and Break Down Avoidance.

4.2.2 HypoType Class

The HypoType class represents the hypothesis type to which each hypothesis corresponds provided that the specific type falls into the category into which the hypothesis was placed. The attribute *Name* of type String belonging to the HypoType class is used so as to differentiate each instance of the class. The HypoType class displays a composition association with the HypoCategory class, as mentioned above, which provides it with one more attribute, the $+category$ of type *HypoCategory* and multiplicity 1 since it is obligatory that every instance of the HypoType class belongs to only one instance of the HypoCategory class. The HypoType class is, also, linked with many hypotheses via its association with the *Hypothesis* class with one - to - many correspondence. This fact enriches it with the attribute $+hypothesis$ of type *Hypothesis* and multiplicity $1..*$. Last, 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 target of the framework is to validate a hypothesis type through their representation in AND/OR Goal Trees. Thus, it is vital that only one Goal Tree will correspond to one hypothesis type.

4.2.3 Hypothesis Class

Each instance of the Hypothesis class represents each hypothesis included into the hypothesis type and category for validation. It is separated from other hypotheses via the attribute *Name* of type String which plays the part of 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 is an indicator of the type that corresponds to the instance of the Hypothesis class. The Hypothesis class has, also, a composition relationship with the *HypoPath* since each path in the goal tree corresponds to one different hypothesis. This gives one more attribute to the class named $+hypopath$ of type *HypoPath* and multiplicity 1 and indicates the hypothesis path that reflects the specific hypothesis. Finally, the class is associated with the *MLN* class via a one - to - one correspondence since for each hypothesis one different MLN is produced. Thus, the class obtains the attribute $+mln$ of type *MLN* and multiplicity 1 in order to indicate this association.

4.2.4 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 derives the attribute $+goaltree$ of the class of type *GoalTree* and multiplicity 1. 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 association 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 correspondance with the *Iterator* class which adds the $+iterator$ attribute of type *Iterator* and multiplicity 1.

4.2.5 Iterator Class

The Iterator class is a class helpful mostly to the implementation of the hypothesis verification algorithm and, therefore, will be covered more thouroughly in the next chapter. 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 which returns the current node that is being visited, the *isDone()* operation which 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.

4.2.6 CNFExpr Class

The CNFExpr class represents the conjuctive 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 conjuctive normal form is constituated by 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 conjective normal form expression.

4.2.7 MLN Class

The MLN class respresents 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 CN form. In addition to the previous, the class is tied with the *Verifier* class via the $+verifierType$ 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.

4.2.8 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.

4.2.9 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 be consisted of many hypothesis paths and, more specifically, of equal number to the leaves of the tree.

4.2.10 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. The first is the name of the goal node which is also used for the construction of the predicates while the second is used as a secure primary key to separate the different goal nodes and prevent any goal node name aliases within different goal trees. The GoalNode class presents a composition association with the GoalTree class which is shown through its attribute $+tree$ of type *GoalTree* and multiplicity 1, that defines for each goal node to which goal tree it belongs. The class has a supplementary attribute $+parent$ of type *GoalNode* and multiplicity 1 which stores for each goal node its parent node. As mentioned earlier, each goal node is also contained into some hypothesis path and, thus, explains the attribute $+path$ of type *HypoPath* and multiplicity 1. Furthermore, the class presents an attribute $+visitorcreator$ of type *VisitorCreator* and multiplicity 1 due to its association with the *VisitorCreator* class which is assigned to create a visitor object in order to process each goal node during the goal tree traversal. Finally, the GoalNode class shows a composition relationship with the *Annotation Container* class which adds to the class the attribute $+annotContainer$ of type *Annotation Container* and multiplicity 1 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. The GoalNode class, finally, has an $+exprproducer$ attribute of type *ExprProducer* which is justified by the association of the class with the *ExprProducer* class.

4.2.11 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 analysed into simpler logical expressions. These are the leaves of the goal tree which do not have any children.

4.2.12 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 another and the goals that have children which present OR - relationships between one another. This is the reason why the DecompositionGoal class has two subclasses, the *ANDDecomposition* class and the *ORDecomposition* class in order to separate the two cases of composite goal nodes and thus permitting the individual and separate manipulation of each type of goal node.

4.2.13 ExprProducer Class

The ExprProducer class plays the most significant role to 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 Data Warehouse.

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

4.2.14 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 present inheritance from it. In figure 4.2 three can be seen as an example. The *VerifierA* class is the one corresponding to the MLN verification strategy for the purposes of this diploma thesis.

4.2.15 VerifierA Class

The VerifierA class represents the MLN strategy. It is assigned to execute the check between the accepted thresholds deriving from the configurations of the simulation, and the data deriving from the Data Warehouse. If the data are within configuration boundaries, a predicate is constructed.

4.2.16 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 building of one instance of the corresponding subclass of VisitorCreator i.e. one of the *ANDVisitorCreator*, *ORVisitorCreator* and

AtomicVisitorCreator. The three types of creators, 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 built which adds more flexibility to the implementation since the traversal of the goal tree becomes a *black box* procedure.

4.2.17 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 instanciated at any time point, the corresponding visitor will be built and inferred. This is the reason for 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 who also built a creator resulting to the particular visitor. Since the three different types of goal nodes request specific manipulation, the operation *visit()* is individually implemented for each one and overrides the operation of the parent class.

4.2.18 AnnotationContainer Class

The AnnotationContainer class provides a container of annotations for each goal tree node in order to store 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 has also a composition relationship with the *Annotation* class and, thus, obtains one more attribute under the name $+annotation$ of type *Annotation* and multiplicity $1..*$.

4.2.19 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 into one instance of the AnnotationContainer class so, therefore, the class owns one more attribute named $+annotcontainer$ of type *AnnotationContainer* and multiplicity 1. The annotations can be of many types but the one subcategory of them can be 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 $1..*$.

Κεφάλαιο 5

Hypothesis Analysis Algorithms

Apart from the architecture and the domain models of the system described in the previous chapters, of great significance to the project were the algorithms implemented in order to accomplish the desired hypothesis verification and which we are about to explain in this section. The basic algorithms used were :

- **Global Hypothesis Formation** : This algorithm includes all the steps followed towards selecting the hypothesis type to be verified and its modeling into the appropriate AND/OR Goal Tree. It also encapsulates the traversal of the generated goal tree for the creation of the predicates.
- **Data and Fact Gathering** : This algorithm includes all the subprocedures to be completed in order to extract the necessary data from the Data Warehouse and the configuration file, and utilize them to produce the evidence predicates.
- **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.

It is also noted that the above algorithms - procedure are executed by this order.

5.1 Global Hypothesis Formation

The first stage of the procedure is the **Global Hypothesis Formation**. The Global Hypothesis Formation algorithm presents the following steps :

Step 1 : Hypothesis Domain XMI Population and Storage. In this first step of the algorithm, the Hypothesis Domain Model is being populated with all the goals and their subgoals that, if satisfied, then 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 of the algorithm, 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.

Step 3 : Hypothesis Domain XMI Load and Parsing. In this third step of the algorithm, the Hypothesis Domain XMI file created in Step 1, is loaded and parsed in order to transform the data included into form suitable for further processing according to the object - oriented programming direction since the development was elaborated with the use of the Java Eclipse framework.

Step 4 : AND/OR Goal Tree Creation. In this fourth step of the algorithm, the data extracted from the populated Hypothesis Domain XMI in step 3 is transformed into the appropriate AND/OR Goal Tree which is useful in the depiction of the AND/OR relationships between the various subgoals.

Step 5 : Goal Tree Node Annotations Creation. In the fifth step of the algorithm, the Goal Tree that was constructed in Step 4, is enriched with annotations for each goal tree node. Those annotations are used to store the necessary informantion for the construction of the corresponding predicate each goal node of the tree.

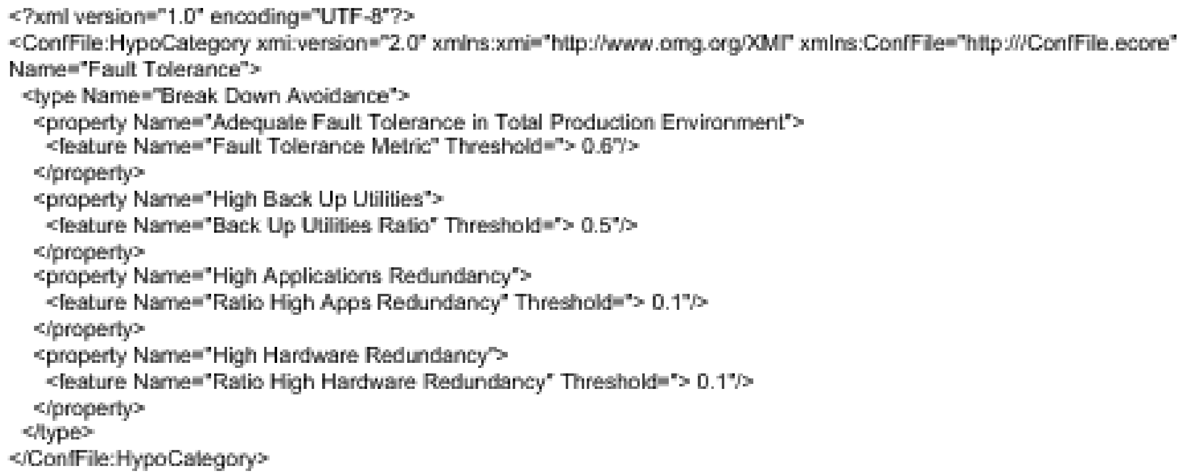
Step 6 : Goal Tree Traversal and Predicates Construction. In this sixth step of the algorithm, the Goal Tree is traversed and for each node that is visited, a predicate or a composite logical expression are constructed always in compliance to the demands of the Alchemy tool to which the file will be set as input for the verification of the hypothesis through a Markov Logic Network.

5.1.1 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 formentioned, in accordance to his or her preferences and, also, in compliance with the means of their representation within the warehouse XMI. That is, if some property mentioned into the Data Warehouse XMI is referred to under the name *Failure Avoidance*, the same property should be referred to under the same name within the Hypothesis XMI inserted by the user through the Editor and UI Module. The population of the Hypothesis Domain Model is exactly the process where the user populates the model with his own data over the hypothesis for which the simulation will be executed. The populated instance of the Hypothesis Domain 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 diploma 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 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 afformentioned procedure.

An example of an XMI document can be seen in figure 5.1.



```
<?xml version="1.0" encoding="UTF-8"?>
<ConfFile:HypoCategory xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:ConfFile="http:///ConfFile.ecore"
Name="Fault Tolerance">
  <type Name="Break Down Avoidance">
    <property Name="Adequate Fault Tolerance in Total Production Environment">
      <feature Name="Fault Tolerance Metric" Threshold="> 0.6"/>
    </property>
    <property Name="High Back Up Utilities">
      <feature Name="Back Up Utilities Ratio" Threshold="> 0.5"/>
    </property>
    <property Name="High Applications Redundancy">
      <feature Name="Ratio High Apps Redundancy" Threshold="> 0.1"/>
    </property>
    <property Name="High Hardware Redundancy">
      <feature Name="Ratio High Hardware Redundancy" Threshold="> 0.1"/>
    </property>
  </type>
</ConfFile:HypoCategory>
```

Σχήμα 5.1: Simple XMI Example

5.1.2 Strategy and Category definition

At a next step, it is demanded 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 throuth 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 ([14]) 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 existant within the Hypothesis DB Server, from previous executions of the framework. In the opposite case that the hypothesis category selected is not already existant in the database, the user is anticipated 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 it was noted above, for the purposes of this diploma 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 subclass which handles the strategy desired. In our case, an instance of the VerifierA class was invoked since this was the subclass 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 after the conclusion of the state and data changes during the verification procedure and, therefore, plays the role of the BlackBoard.

5.1.3 Hypothesis Domain XMI Load and Parsing

At the third step, the Test class, creates one instance of the ExprProducer class which encapsulates the main operations that expedite the majority of the full verification process body. It is, thus, seen that the ExprProducer class corresponds to the Controller component that invokes the hypothesis generation process by notifying the Hypothesis Generator component via publishing an event through the Publisher component, renews the BlackBoard condition when the generation is concluded with the hypothesis generation results, invokes the appropriate verifier from the Verifier component through the Subscriber component and updates the BlackBoard with the results of the verification process when the latter is concluded.

At this point, the ExprProducer class notifies the Hypothesis Generator component which is simulated by the **constructpath** procedure of the class. Within its body, the Hypothesis Domain XMI that was constructed earlier, is being loaded and parsed using DOM. The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. The Java programming language provides very useful libraries for the implementation of DOM in order to parse the desired XMI document.

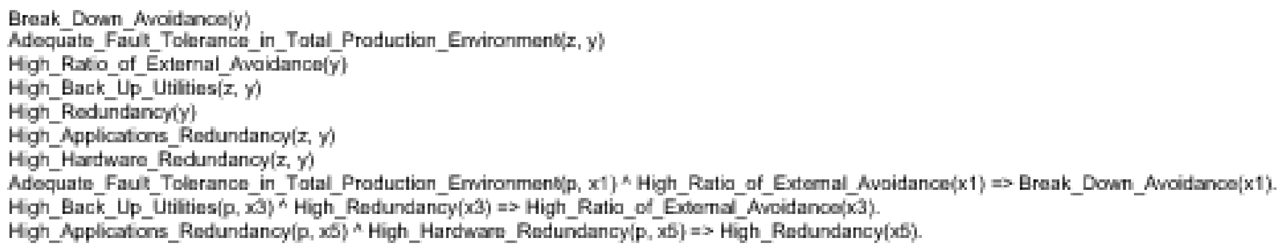
5.1.4 AND/OR Goal Tree Creation

At this fourth step of the algorithm, the hypothesis category and type 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 Tree. 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 existant, are connected as a conjunction or as a disjunction. Finally, for each node its *parent* and *children* fields are filled with the appropriate goal nodes in order to complete the AND/OR Goal Tree construction.

It should also be mentioned that the name of each node can correspond to an existing property of the specific hypothesis type within the Data Warehouse. For the purposes of this diploma thesis in order to simplify the process, it was assumed that only nodes of type *AtomicGoal* which are the leaves of the tree, can correspond to properties of the Data Warehouse while the nodes of type *DecompositionGoal* are iteratively defined as conjunctions or disjunctions of their children nodes.

5.1.5 Goal Tree Node Annotations Creation

At the fifth step, 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 again with DOM usage. Every time an atomic goal node is met, the configuration file is traversed and the name of the feature along with its threshold or accepted value that corresponds to the property defined by the atomic goal node name, are extracted and stored into a new instance of the Annotation class. Next, this instance of the Annotation class is attached to the current node. When an ANDDecomposition goal node or an ORDecomposition goal node are met, the annotations attached to the children nodes, are read, transformed into an appropriate form and stored into a new



```
Break_Down_Avoidance(y)
Adequate_Fault_Tolerance_in_Total_Production_Environment(z, y)
High_Ratio_of_External_Avoidance(y)
High_Back_Up_Utilities(z, y)
High_Redundancy(y)
High_Applications_Redundancy(z, y)
High_Hardware_Redundancy(z, y)
Adequate_Fault_Tolerance_in_Total_Production_Environment(p, x1) ^ High_Ratio_of_External_Avoidance(x1) => Break_Down_Avoidance(x1).
High_Back_Up_Utilities(p, x3) ^ High_Redundancy(x3) => High_Ratio_of_External_Avoidance(x3).
High_Applications_Redundancy(p, x5) ^ High_Hardware_Redundancy(p, x5) => High_Redundancy(x5).
```

Σχήμα 5.2: Simple AND/OR Goal Tree Predicates

instance of the Annotation class. This new annotation is again attached to the current node. This step is concluded as soon as all the nodes have been enhanced with an annotation container containing the correct annotations.

5.1.6 Goal Tree Traversal and Predicates Construction

At this sixth and final step of the algorithm, the Goal Tree is traversed in order to construct the predicates that will compose the first - order logic world on which the MLN construction will be based later on. In order for the traversal of the Goal Tree to be held, a new instance of the Iterator class is generated which iterates over all the nodes of the Goal Tree. When it has reached a new node, the iterator calls the operation *accept()* which is defined for the GoalNode class and is overriden by the individual definitions of the same operation within the AtomicGoal class, the ANDDecomposition class and the ORDecomposition class.

Each individual operation *accept()* within each of the above classes creates a new instance of the appropriate subclass of the VisitorCreator class according to the type of the goal node. The visitor creator built, invokes the *visitorFactoryMethod()* which is defined within its class implementation in order to create an instance of the appropriate subclass of the Visitor class again according to the goal node type. The visitor created, is attached to the current node that is visited and the *accept()* operation returns the result of the *visit()* operation of the visitor instance.

The *visit()* operation of each type of visitor instance, is assigned to read the annotations within the annotation container of the current node, transform the information read into the predicate form and return a list of predicates that correspond to this node. The resulting list is returned to the iterator instance which adds the contents of the list to a universal list of predicates that will be returned to the Controller, that is the instance of the ExprProducer class, for printing.

Finally, the instance of the ExprProducer class initializes a file with the *.mln* extension which is the extension of the first - order logic world file expected from the Alchemy tool, and prints the predicates contained in the list.

A simple example of the predicates form that correspond to an AND/OR Goal Tree, can be seen in figure 5.2 below :

5.2 Data and Fact Gathering

The second stage of the procedure is the **Data and Fact Gathering**. The Data and Fact Gathering algorithm presents the following steps :

Step 1 : **Warehouse Domain XMI and Configuration Population and Storage**. In this first step of the algorithm, the Warehouse Domain Model is being populated with test data, that is properties and features of project modules along with their feature values. The properties are in compliance with the goal node names appearing in the hypothesis AND/OR Goal Tree in order to identify them as the same property by applying pattern matching on the names, and, therefore, make the verification procedure possible. This model population is then stored in XMI format in order to be used as a sample input file to the framework developed.

Step 2 : **Verification Process Start**. In this second step of the algorithm, after the BlackBoard component which synchronizes the procedures of the different framework components, sends a message to the Contoller of the Verification Module, the latter triggers the beggining of the verification process.

Step 3 : **Isolation of Features and Thresholds**. In the third step of the algorithm, the features along with their values are isolated from the populated Data Warehouse Domain XMI file. The corresponding thresholds for each feature are also extracted from the Configuration XMI file.

Step 4 : **Predicates Construction**. In the fourth step of the algorithm, the evidence predicates based on the data deriving from the Data Warehouse, are produced and printed into a file in order to be fed to the Alchemy tool.

5.2.1 Warehouse Domain XMI and Configuration XMI Population and Storage

In order for the verification to take place, it is a precondition that the Data Warehouse is populated with properties, features and feature values. The data within the Data Warehouse originally derive from the information given by developers on IDEs and projects that are of interest to them. This stage is represented in the system architecture by the IDEContainer component. Next, the data pass through the Mediator, Extractor and Selector components in order to reach a form suitable to the analysis i.e. suitable to the kind of input the Verifier component expects and considers usable, and, consequently, be stored into the Data Pool component as an XMI compliant document form. At this stage of the procedure, the configuration file set by the user before the execution of the system, will be needed.

For the purposes of this diploma thesis, we used simulated data and, therefore, exemplary Warehouse Domain XMI files were populated through the use of the Eclipse runtime environment as it was described for the Hypothesis Domain XMI population step respectively. The data added into the Warehouse Domain XMI conform to the constraint mentioned above according to which, the properties within the Data Warehouse must correspond by name, to the properties stated into the Hypothesis Domain XMI in order to associate a property validating our hypothesis type appearing at the AND/OR Goal Tree, with the features within the Data Warehouse, appearing under the property with the same name. As for 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 like the two previous XMI files needed, and thresholds were assigned for the features appearing into the Data Warehouse sample constructed.

5.2.2 Verification Process Start

After the conclusion of the Global Hypothesis Formation algorithm, the BlackBoard, i.e. the instance of the Test class, is notified of this event and sends a message to the Controller of the Verification Module, i.e. the instance of the ExprProducer class running, that the verification is ready to begin by invoking the *attachstrategy()* operation which is defined within the class implementation body of the latter. This operation ultimately returns the predicates deriving from the Data Warehouse that correspond to features with acceptable values according to the thresholds read by the configuration file. The invocation of the *attachstrategy()* operation simulates the *subscribe()* service provided to the Subscriber component and the operation itself simulates the Subscriber and Publisher components of the system architecture.

5.2.3 Isolation of Features and Thresholds

In this next step of the procedure, the target is to isolate the features of the properties appearing into Data Warehouse along with their values, and also extract the thresholds and acceptable values of those features which are provided by the Configuration XMI file. Therefore, the instance of the ExprProducer class begins the execution of the *attachstrategy()* operation body by loading and parsing the Warehouse Domain XMI and Configuration XMI documents using the DOM library provided by the Java programming language.

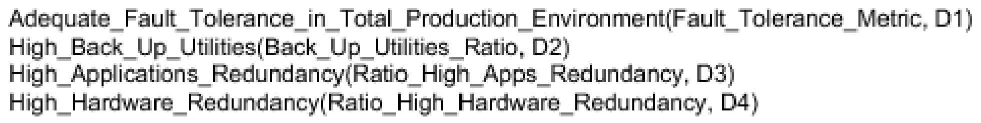
Continuing the execution, for each property into the Warehouse XMI, the features are isolated. For every feature, the Configuration XMI file is searched and, when the feature is found, its threshold or acceptable value is extracted. Finally, all the properties with their feautures, values and thresholds are stored temporarily for the next step of the algorithm to take place.

5.2.4 Predicates Construction

At this final step of the algorithm, the target is to print the correct predicates according to the information extracted by the Data Warehouse. In order to accomplish this, for each feature the Subscriber simulation, i.e. the *attachstrategy()* operation, invokes the *verify()* operation defined into the Strategy class implementation and, more precisely, invokes the *verify()* operation defined into the VerifierA class implementation which overrides the one defined into its parent class Strategy, since this is the subclass that handles the MLN verification in our case.

The *verify()* operation of the VerifierA class checks whether the feature value is within the boundaries set by the corresponding threshold and, if so, constructs the appropriate predicate and returns it to the Controller instance of the ExprProducer class or else it returns null. This is executed for each feature individually. Every time it receives a predicate, the Controller adds it to a list of predicates. When the features are all processed and the valid predicates are added to the universal list, the Controller prints the predicates into a file with the extension *.db* which is the extension expected by the Alchemy tool for the input database file including all the predicates that are assigned a true boolean value. Finally, the Controller notifies and updates the BlackBoard, i.e. our Test class, when the *attachstrategy()* operation returns.

An example of the predicates form that are produced using data from the Data Warehouse, can be seen in figure 5.3.



```
Adequate_Fault_Tolerance_in_Total_Production_Environment(Fault_Tolerance_Metric, D1)
High_Back_Up_Utilities(Back_Up_Utilities_Ratio, D2)
High_Applications_Redundancy(Ratio_High_Apps_Redundancy, D3)
High_Hardware_Redundancy(Ratio_High_Hardware_Redundancy, D4)
```

Σχήμα 5.3: Simple Data Warehouse Predicates

5.3 Markov Logic Network Based Reasoning

On this last part of the process, the two files produced by the Global Hypothesis Formation and the Data and Fact Gathering algorithms are given as input to the Alchemy tool in order for it to execute an **inference** procedure on the MLN 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 5.4 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 5.5 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 5.6 below :

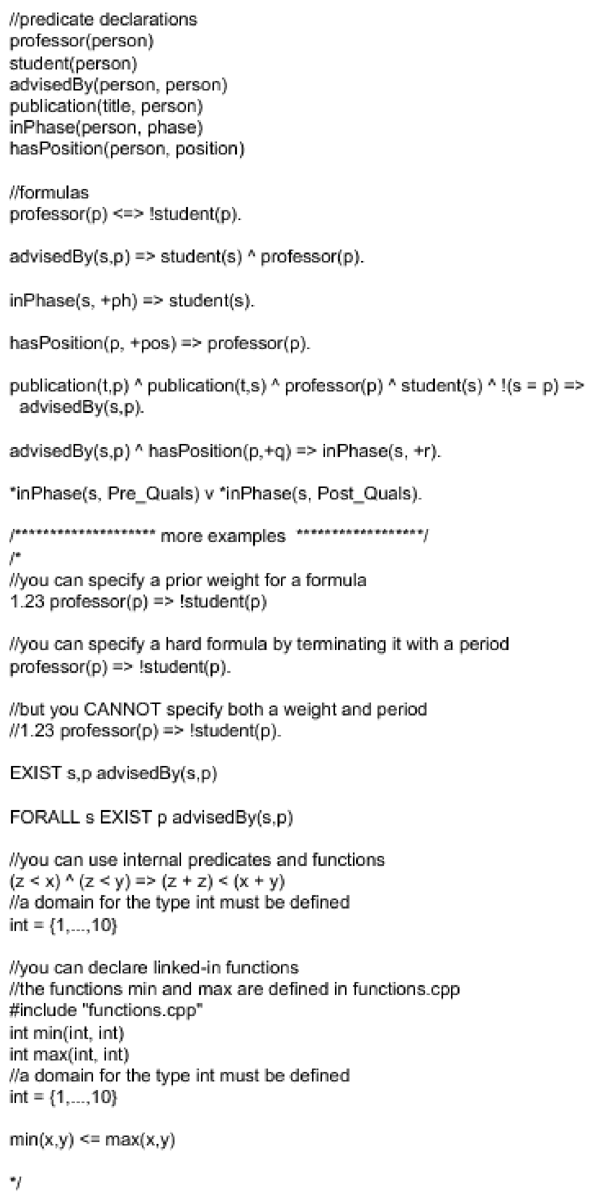
The MaxWalkSAT algorithm differs a little from the WalkSAT algorithm. This algorithm in pseudocode, can be seen in figure 5.7 below :

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 apply the MCMC algorithm.

The algorithm for the construction of the Gound Network in pseudocode, can be seen in figure 5.8 below :

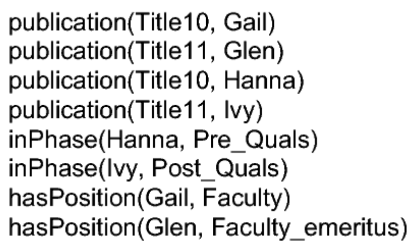
Similarly, the MCMC (Gibbs sampling) algorithm in pseudocode, can be seen in figure 5.9.

Finally, the output of the Alchemy tool is a set of percentages, one for each atomic goal node, i.e. leaf of the goal tree, to indicate the probability of the path starting from this atomic goal node, to occur.



```
//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)

*/
```

Σχήμα 5.4: 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)
```

Σχήμα 5.5: 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
```

Σχήμα 5.6: 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
```

Σχήμα 5.7: 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 = {}
```

Σχήμα 5.8: 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
```

Σχήμα 5.9: MCMC : Gibbs sampling Algorithm

Κεφάλαιο 6

Case Studies

In the previous chapters the architecture, the domain models and the algorithms implemented for the purposes of this framework were discussed. In this chapter, we shall look into some case studies of the framework. More specifically, three examples will be illustrated as case studies. For all three examples the Hypothesis Category is Fault Tolerance. In this respect, Fault Tolerance is validated with certainty if and only if all three of its Hypothesis Types are validated. The three Hypothesis Types that belong to the Hypothesis Category Fault Tolerance, are **Failure Avoidance**, **Incorrect Operation Avoidance** and **Break Down Avoidance**. For the execution of the above test casing, an IBM ThinkPad 1.8Ghz Pentium M and 1.5GB DDR RAM Memory running the Debian Squeeze Linux distribution, was used. The software tools utilized for the simulation of the framework, were the MagicDraw Enterprise edition software tool for designing UML diagrams and exporting to XMI e - core compliant, the Eclipse Modeling Framework (EMF) version 3.5 for the code generation and development and the Alchemy tool for the Markov Logic Networks construction and evaluation of the hypotheses.

For better understanding, a complete list of the predicates that can be seen in the above figure, is provided with brief explanations of the symbols :

Failure_ Avoidance(y) : This predicate denotes that the system presents properties that enable it to avoid failure.

Internal_ Failure_ Avoidance(y) : This predicate specifies that the system can avoid internal failure.

External_ Failure_ Avoidance(y) : This predicate denotes that the system can avoid external failure.

High_ Ratio_ of_ Internal_ Avoidance(y) : This denotes that the system presents a high ratio of test cases avoiding internal failure against the total number of test cases executed.

Low_ Complexity(y) : This predicate specifies that the source code and architecture of the system present low complexity.

High_ Ratio_ of_ External_ Avoidance(y) : This predicate denotes that the system presents a high ratio of test cases avoiding external failure against the total number of test cases executed.

Low_ Dependencies(y) : The predicate denotes that the source code of the various components, modules or other structures of the system, present low dependencies among one another.

High_ Redundancy(y) : This predicate implies that the system has duplicates for many of its components which assists in avoiding failure.

Incorrect_ Operation_ Avoidance(y) : This predicate denotes that the system has properties that permit the avoidance of incorrect operations.

Internal_ Incorrect_ Operation_ Avoidance(y) : This predicate specifies that the system can avoid internal incorrect operations.

External_ Incorrect_ Operation_ Avoidance(y) : This predicate denotes that the system can avoid external incorrect operations.

High_ Ratio_ of_ Internal_ Operation_ Avoidance(y) : This predicate denotes that the system has a high ratio of test cases not presenting incorrect internal operations against the total number of test cases executed.

High_ Ratio_ of_ Internal_ Logic_ Checks(y) : This predicate denotes that the system presents a high ratio of internal logic checks per time unit.

Break_ Down_ Avoidance(y) : This predicate indicates that the system has properties that enable it to avoid breaking down.

Each of the above predicates resolves to a *true* boolean value if the path in the goal tree starting from the atomic goal, i.e. leaf, *y*, is satisfied and, therefore, the module for which the test case is executed, will have properties defined by the name of each predicate.

High_ Ratio_ of_ Exception_ Handling(z, y) : This predicate indicates that the project presents a high ratio of exceptions handled against the overall number of exception appearing.

Avoidance_ of_ Memory_ Leaks(z, y) : This predicate denotes that the system presents avoidance of memory leaks.

Low_ CC(z, y) : This predicate indicates that the hardware architecture of the system presents low cyclomatic complexity.

Low_ FP(z, y) : This predicate indicates that the system presents low functional points, that is, the system provides a low amount of business functionality to a user.

High_ Cohesion(z, y) : The predicate denotes that the system presents high cohesion, that is, the functionality expressed by the source code of a software module, is strongly-related.

High_ Back_ Up_ Utilities(z, y) : This predicate implies that the system has a large number of back up utilities to support its modules and components.

High_ Applications_ Redundancy(z, y) : This predicate indicates that the system has various duplicates of its applications in order to ensure a non seizing functionality by the occurence of an error.

High_ Hardware_ Redundancy(z, y) : This predicate indicates that the system provides various duplicates of its hardware components and modules in order to avoid an operation hault in case of an error.

Low_ Coupling(z, y) : This predicate denotes that the system presents low coupling, that is, low dependencies among its different modules and components.

Low_ Information_ Flow(z, y) : This predicate indicates that the modules and components of the system present low need of exchanging information frequently.

High_ Ratio_ of_ Incorrect_ Sequence_ of_ Data_ Input(z, y) : This predicate denotes that the system presents a high ratio of incorrect sequence of data input against total number of data streams inserted to it as input.

High_ Ratio_ of_ Incorrect_ Operation_ Patterns(z, y) : This predicate indicates that the system presents a large number of incorrect operation patterns against the total number of its operation patterns.

High_ Ratio_ of_ Incorrect_ Data_ Types_ as_ Parameters(z, y) : This predicate denotes the appearance of a large number of errors related to incorrect data types set as parameters.

High_ Ratio_ of_ Incorrect_ Sequence_ of_ Operation(z, y) : This predicate denotes that the system presents a high ratio of incorrect operation sequences per time unit.

High_ Ratio_ of_ Preconditions_ Checks(z, y) : This predicate indicates that the system runs a high number of preconditions checks per time unit.

High_ Ratio_ of_ Postconditions_ Checks(z, y) : This predicate indicates that the system runs a high number of postconditions checks per time unit.

High_ Ratio_ of_ Compliance_ Checks(z, y) : This predicate indicates that the system runs a high number of compliance checks per time unit.

Correct_ Business_ Process_ Specs(z, y) : This predicate denotes that the system presents correct business process specifications.

Correct_ Service_ Registry_ Specs(z, y) : This predicate denotes that the system presents correct service registry specifications.

Correct_ Deployment_ Description_ Specs(z, y) : This predicate denotes that the system presents correct deployment description specifications.

Correct_ Service_ Description_ Specs(z, y) : This predicate denotes that the system presents correct service description specifications.

Adequate_ Fault_ Tolerance_ in_ Total_ Production_ Environment(z, y) : This predicate indicates that the system presents adequate fault tolerance within the total production environment.

Each of the above predicates resolves to a *true* boolean value if the variable *z* takes as value a valid feature for the property described by the name of the predicate and if the path in the goal tree starting from the atomic goal, i.e. leaf, *y*, is satisfied and, therefore, the module for which the test case is executed, will have properties defined by the name of each predicate.

A listing of the evidence predicates deriving from the Data Warehouse, is also provided :

High_ Back_ Up_ Utilities(Back_ Up_ Utilities_ Ratio, D1)
High_ Applications_ Redundancy(Ratio_ High_ Apps_ Redundancy, D2)
High_ Hardware_ Redundancy(Ratio_ High_ Hardware_ Redundancy, D3)
Low_ Coupling(Ratio_ Low_ Coupling, D4)
High_ Ratio_ of_ Incorrect_ Sequence_ of_ Data_ Input(Ratio, D1)
High_ Ratio_ of_ Incorrect_ Operation_ Patterns(Ratio, D2)
High_ Ratio_ of_ Preconditions_ Checks(Ratio, D5)
Correct_ Service_ Description_ Specs(Ratio, D10)
Adequate_ Fault_ Tolerance_ in_ Total_ Production_ Environment(Fault_ Tolerance_ Metric, D1)
High_ Back_ Up_ Utilities(Back_ Up_ Utilities_ Ratio, D2)
High_ Applications_ Redundancy(Ratio_ High_ Apps_ Redundancy, D3)
High_ Hardware_ Redundancy(Ratio_ High_ Hardware_ Redundancy, D4)

As it can be seen and according to the explanation for each predicate given above, the first value to all predicates resolves to a valid feature for the property described by each predicate and the second variable D1, D2, D3, D4, D5, D10 in the above predicates, represents a particular atomic goal, i.e. leaf, of a goal tree by which each path to the root begins and for which the Alchemy tool will calculate a probability of occuring within the set world of first - order logic predicates constructed by the AND/OR Goal Tree.

6.1 Example 1

In this first example presented, the Hypothesis Type to be validated is **Failure Avoidance**.

After populating the Hypothesis Domain Model using the Eclipse runtime environment, the XMI document which is also compliant to e - core, is shown in figure 6.1 below.

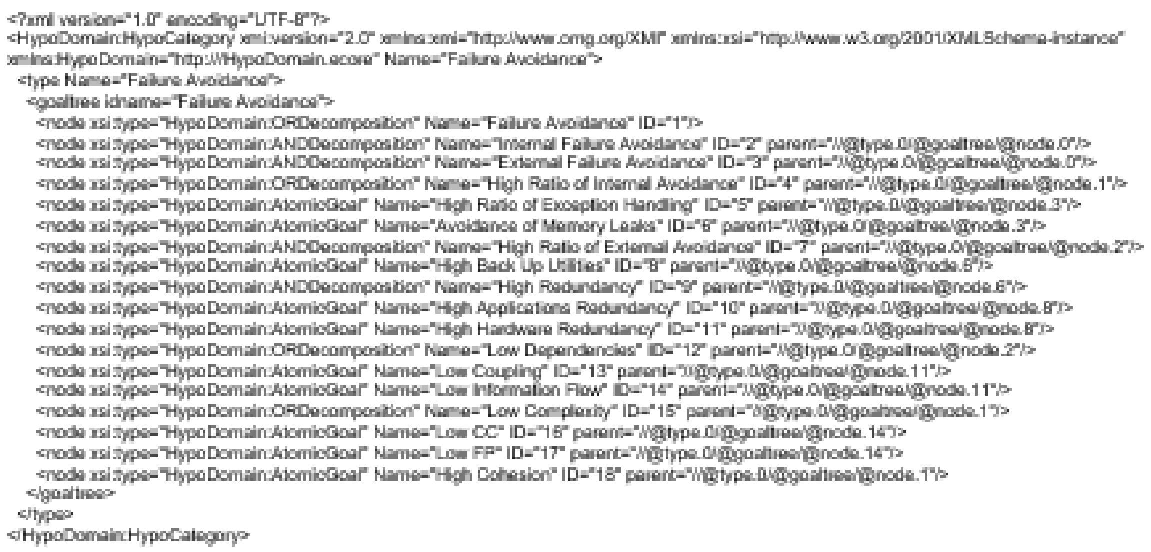
In order for the reader to better understand the structure of the XMI document in figure 6.1 and how this represents an AND/OR Goal Tree, let us provide the same schema in the form of a tree in figure 6.2.

The file *hypoth.mln* produced by the execution of the Global Hypothesis Formation algorithm, can be seen in figure 6.3 below.

After populating the Warehouse Domain Model using the Eclipse runtime environment, the XMI document which is also compliant to e - core, is shown in figure 6.4 below.

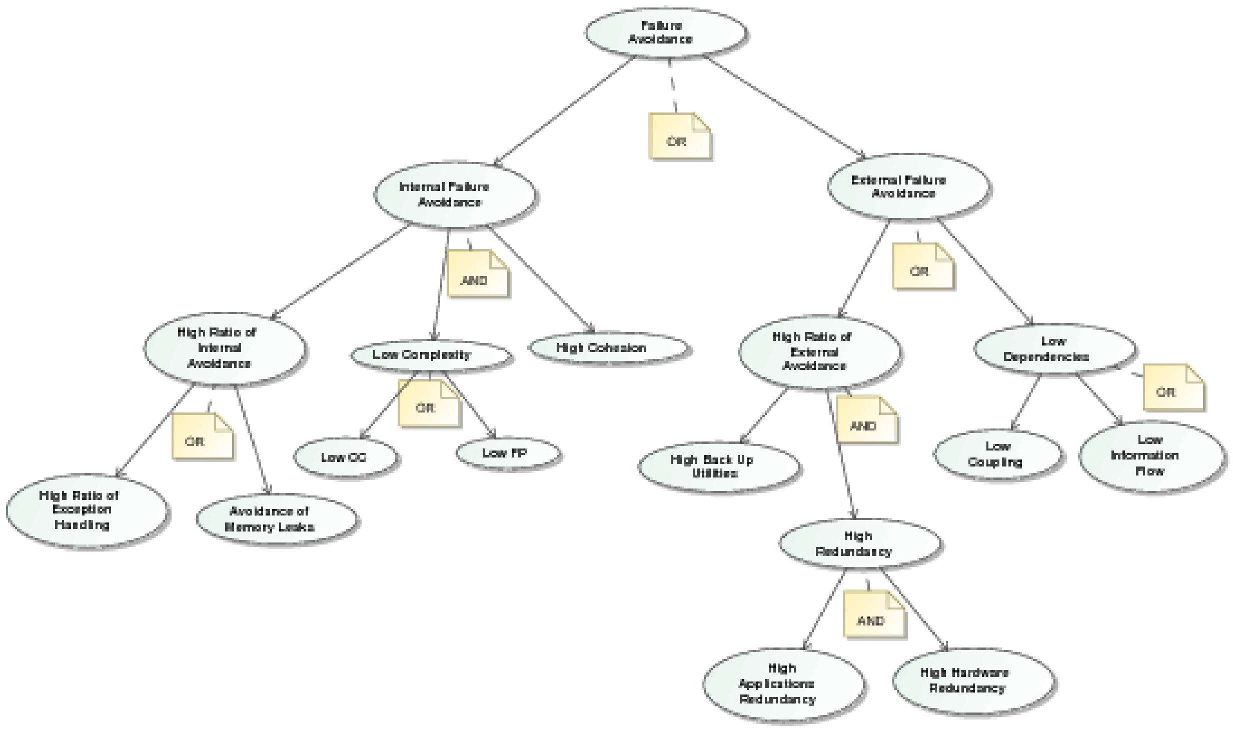
In the above XMI document displaying the Data Warehouse, the properties and corresponding features along with their values, can be seen.

After populating the Configuration Model using the Eclipse runtime environment, the XMI document which is also compliant to e - core, is shown in figure 6.5 below.



```
<?xml version="1.0" encoding="UTF-8"?>
<HypoDomain:HypoCategory xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:HypoDomain="http:///HypoDomain.ecore" Name="Failure Avoidance">
  <type Name="Failure Avoidance">
    <goaltree idname="Failure Avoidance">
      <node xsi:type="HypoDomain:ORDecomposition" Name="Failure Avoidance" ID="1"/>
      <node xsi:type="HypoDomain:ANDDecomposition" Name="Internal Failure Avoidance" ID="2" parent="//@type.0/@goaltree/@node.0"/>
      <node xsi:type="HypoDomain:ANDDecomposition" Name="External Failure Avoidance" ID="3" parent="//@type.0/@goaltree/@node.0"/>
      <node xsi:type="HypoDomain:ORDecomposition" Name="High Ratio of Internal Avoidance" ID="4" parent="//@type.0/@goaltree/@node.1"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="High Ratio of Exception Handling" ID="5" parent="//@type.0/@goaltree/@node.3"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="Avoidance of Memory Leaks" ID="6" parent="//@type.0/@goaltree/@node.3"/>
      <node xsi:type="HypoDomain:ANDDecomposition" Name="High Ratio of External Avoidance" ID="7" parent="//@type.0/@goaltree/@node.2"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="High Back Up Utilities" ID="8" parent="//@type.0/@goaltree/@node.6"/>
      <node xsi:type="HypoDomain:ANDDecomposition" Name="High Redundancy" ID="9" parent="//@type.0/@goaltree/@node.6"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="High Applications Redundancy" ID="10" parent="//@type.0/@goaltree/@node.8"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="High Hardware Redundancy" ID="11" parent="//@type.0/@goaltree/@node.8"/>
      <node xsi:type="HypoDomain:ORDecomposition" Name="Low Dependencies" ID="12" parent="//@type.0/@goaltree/@node.2"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="Low Coupling" ID="13" parent="//@type.0/@goaltree/@node.11"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="Low Information Flow" ID="14" parent="//@type.0/@goaltree/@node.11"/>
      <node xsi:type="HypoDomain:ORDecomposition" Name="Low Complexity" ID="15" parent="//@type.0/@goaltree/@node.1"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="Low CC" ID="16" parent="//@type.0/@goaltree/@node.14"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="Low FP" ID="17" parent="//@type.0/@goaltree/@node.14"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="High Cohesion" ID="18" parent="//@type.0/@goaltree/@node.1"/>
    </goaltree>
  </type>
</HypoDomain:HypoCategory>
```

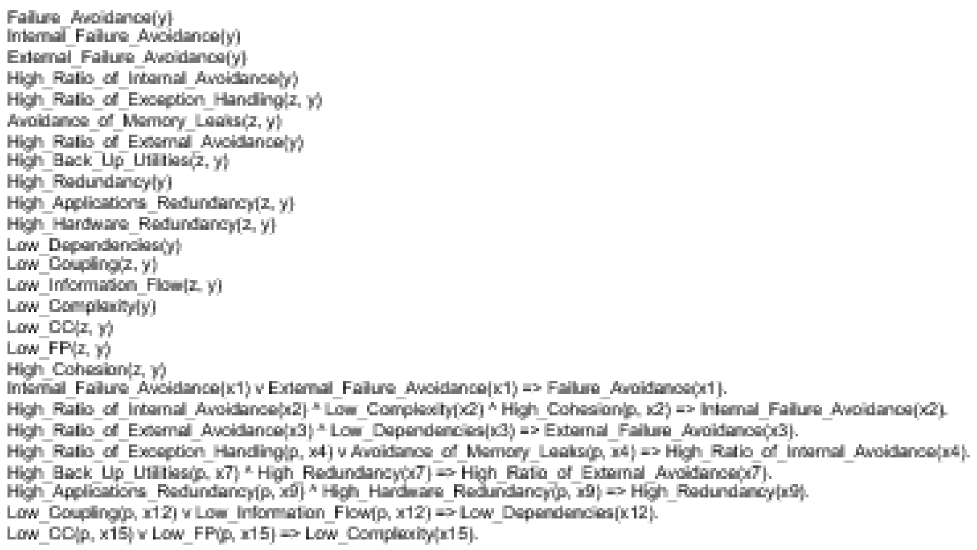
Σχήμα 6.1: Hypothesis Domain XMI for example 1


Failure Avoidance
OR
Internal Failure Avoidance
External Failure Avoidance
AND
OR
High Ratio of Internal Avoidance
Low Complexity
High Cohesion
High Ratio of External Avoidance
Low Dependencies
OR
OR
AND
OR
Low CC
Low FP
High Back Up Utilities
Low Coupling
Low Information Flow
High Ratio of Exception Handling
Avoidance of Memory Leaks
High Redundancy
AND
High Applications Redundancy
High Hardware Redundancy

Σχήμα 6.2: AND/OR Goal Tree of figure 6.1 XMI document

The configuration XMI file, as shown in the previous figure, contains all the thresholds and acceptable values for the features included in the Data Warehouse XMI.

The *entity0.db* file for Module 1 of the Data Warehouse as it appears in the datawarehouse.xml file, is shown in figure 6.6 below.



```
Failure_Avoidance(y)
Internal_Failure_Avoidance(y)
External_Failure_Avoidance(y)
High_Ratio_of_Internal_Avoidance(y)
High_Ratio_of_Exception_Handling(z, y)
Avoidance_of_Memory_Leaks(z, y)
High_Ratio_of_External_Avoidance(y)
High_Back_Up_Utilities(z, y)
High_Redundancy(y)
High_Applications_Redundancy(z, y)
High_Hardware_Redundancy(z, y)
Low_Dependencies(y)
Low_Coupling(z, y)
Low_Information_Flow(z, y)
Low_Complexity(y)
Low_CC(z, y)
Low_FP(z, y)
High_Cohesion(z, y)
Internal_Failure_Avoidance(x1) v External_Failure_Avoidance(x1) => Failure_Avoidance(x1).
High_Ratio_of_Internal_Avoidance(x2) ^ Low_Complexity(x2) ^ High_Cohesion(p, x2) => Internal_Failure_Avoidance(x2).
High_Ratio_of_External_Avoidance(x3) ^ Low_Dependencies(x3) => External_Failure_Avoidance(x3).
High_Ratio_of_Exception_Handling(p, x4) v Avoidance_of_Memory_Leaks(p, x4) => High_Ratio_of_Internal_Avoidance(x4).
High_Back_Up_Utilities(p, x7) ^ High_Redundancy(x7) => High_Ratio_of_External_Avoidance(x7).
High_Applications_Redundancy(p, x9) ^ High_Hardware_Redundancy(p, x9) => High_Redundancy(x9).
Low_Coupling(p, x12) v Low_Information_Flow(p, x12) => Low_Dependencies(x12).
Low_CC(p, x15) v Low_FP(p, x15) => Low_Complexity(x15).
```

Σχήμα 6.3: hypoth.mln file for example 1

The output of the Alchemy tool for Module 1 respectively, is shown in figure 6.7 below.

Thus, the conclusion that can be obtained from the Alchemy tool output in figure 6.7, is that the probability of Failure Avoidance to occur is high and averages to 0.8667 with likelier state of world the path starting from *High Back Up Utilities* leading to the root that has a probability equal to 0.888961. Therefore, through logical deduction, this leads to a probability of 0.12 - 0.15 approximately for Module 1 not to avoid failure.

6.2 Example 2

In this second example presented, the Hypothesis Type to be validated is **Incorrect Operation Avoidance**. After populating the Hypothesis Domain Model using the Eclipse runtime environment, the XMI document which is also compliant to e - core, is shown in figure 6.8 below.

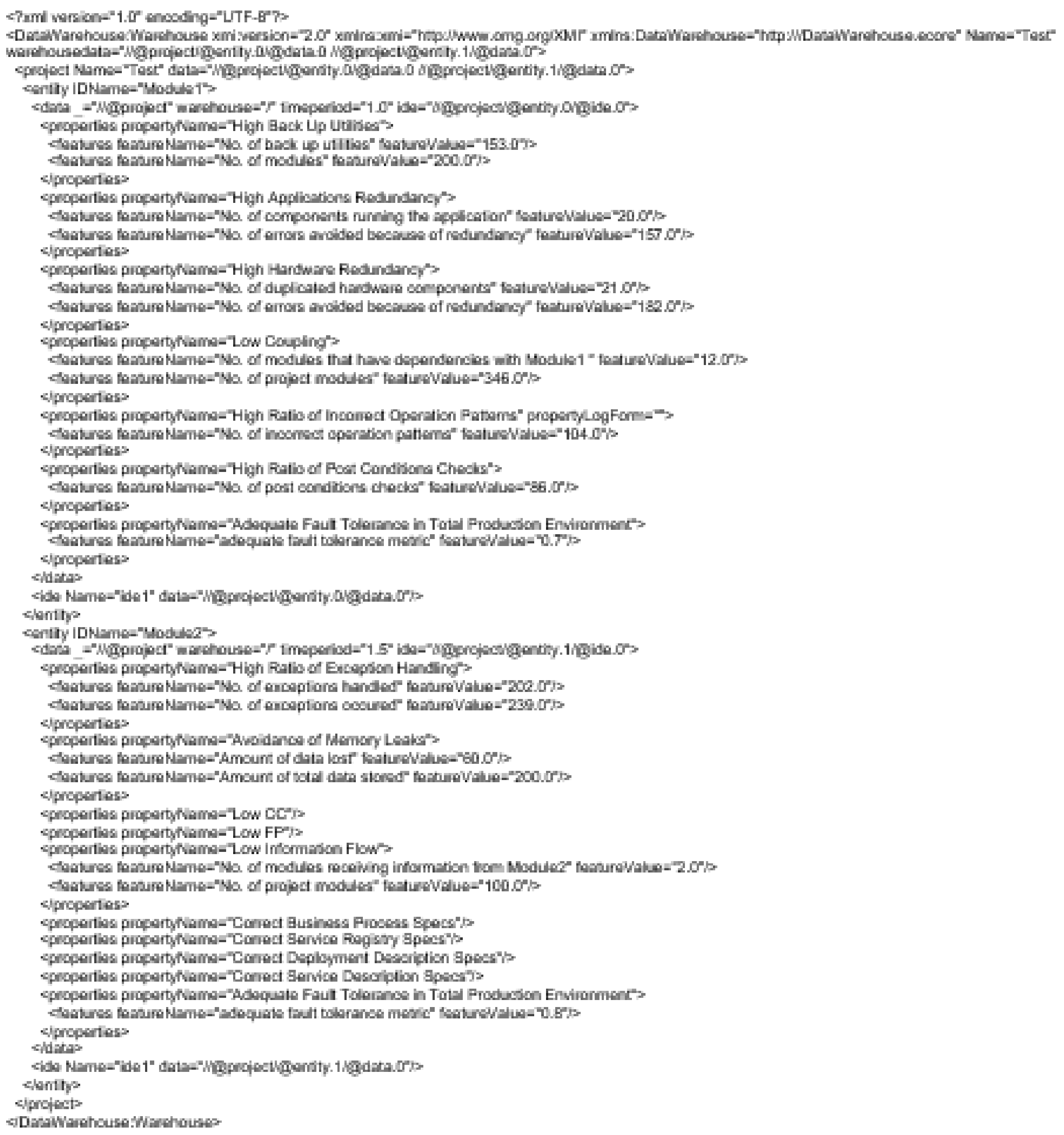
The corresponding AND/OR Goal Tree is illustrated in figure 6.9.

After launching the simulation, the *hypoth.mln* file produced which represents the first - order logic world given as input to Alchemy, can be seen in figure 6.10.

The Data Warehouse XMI file populated for the purposes of this example, can be seen in figure 6.11.

The Configuration XMI file populated for the purposes of this example, can be seen in figure 6.12.

After the simulation launch, the *entity0.db* file for Module 2 produced, including all the predicates that have a true boolean value according to the information extracted by the Data Warehouse and configuration file, can be seen in figure 6.13.



```
<?xml version="1.0" encoding="UTF-8"?>
<DataWarehouse:Warehouse xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:DataWarehouse="http:///DataWarehouse.ecore" Name="Test"
warehousedata="//@project/@entity.0/@data.0 //@project/@entity.1/@data.0">
  <project Name="Test" data="//@project/@entity.0/@data.0 //@project/@entity.1/@data.0">
    <entity IDName="Module1">
      <data _="//@project" warehouse="/" timeperiod="1.0" ide="//@project/@entity.0/@ide.0">
        <properties propertyName="High Back Up Utilities">
          <features featureName="No. of back up utilities" featureValue="153.0"/>
          <features featureName="No. of modules" featureValue="200.0"/>
        </properties>
        <properties propertyName="High Applications Redundancy">
          <features featureName="No. of components running the application" featureValue="20.0"/>
          <features featureName="No. of errors avoided because of redundancy" featureValue="157.0"/>
        </properties>
        <properties propertyName="High Hardware Redundancy">
          <features featureName="No. of duplicated hardware components" featureValue="21.0"/>
          <features featureName="No. of errors avoided because of redundancy" featureValue="182.0"/>
        </properties>
        <properties propertyName="Low Coupling">
          <features featureName="No. of modules that have dependencies with Module1 " featureValue="12.0"/>
          <features featureName="No. of project modules" featureValue="346.0"/>
        </properties>
        <properties propertyName="High Ratio of Incorrect Operation Patterns" propertyLogForm="">
          <features featureName="No. of incorrect operation patterns" featureValue="104.0"/>
        </properties>
        <properties propertyName="High Ratio of Post Conditions Checks">
          <features featureName="No. of post conditions checks" featureValue="86.0"/>
        </properties>
        <properties propertyName="Adequate Fault Tolerance in Total Production Environment">
          <features featureName="adequate fault tolerance metric" featureValue="0.7"/>
        </properties>
      </data>
      <ide Name="ide1" data="//@project/@entity.0/@data.0"/>
    </entity>
    <entity IDName="Module2">
      <data _="//@project" warehouse="/" timeperiod="1.5" ide="//@project/@entity.1/@ide.0">
        <properties propertyName="High Ratio of Exception Handling">
          <features featureName="No. of exceptions handled" featureValue="202.0"/>
          <features featureName="No. of exceptions occured" featureValue="239.0"/>
        </properties>
        <properties propertyName="Avoidance of Memory Leaks">
          <features featureName="Amount of data lost" featureValue="60.0"/>
          <features featureName="Amount of total data stored" featureValue="200.0"/>
        </properties>
        <properties propertyName="Low CC"/>
        <properties propertyName="Low FP"/>
        <properties propertyName="Low Information Flow">
          <features featureName="No. of modules receiving information from Module2" featureValue="2.0"/>
          <features featureName="No. of project modules" featureValue="100.0"/>
        </properties>
        <properties propertyName="Correct Business Process Specs"/>
        <properties propertyName="Correct Service Registry Specs"/>
        <properties propertyName="Correct Deployment Description Specs"/>
        <properties propertyName="Correct Service Description Specs"/>
        <properties propertyName="Adequate Fault Tolerance in Total Production Environment">
          <features featureName="adequate fault tolerance metric" featureValue="0.8"/>
        </properties>
      </data>
      <ide Name="ide1" data="//@project/@entity.1/@data.0"/>
    </entity>
  </project>
</DataWarehouse:Warehouse>
```

Σχήμα 6.4: Warehouse Domain XMI for example 1

Finally, feeding the previous two output files as input to the Alchemy tool, we get the following results as shown in figure 6.14.

From the results of the Alchemy tool, we can logically deduct that *Incorrect Operation Avoidance* has high probability of average 0.85 of occuring, that is the probability of a fault occuring which is connected with the lack of incorrect operations, is around 0.13 - 0.19 approximately.

```
<?xml version="1.0" encoding="UTF-8"?>
<ConfFile:HypoCategory xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:ConfFile="http:///ConfFile.ecore" Name="Fault Tolerance">
  <type Name="Failure Avoidance">
    <property Name="High Ratio of Exception Handling">
      <feature Name="Ratio of Exception Handling" Threshold=" > 0.8"/>
    </property>
    <property Name="Avoidance of Memory Leaks">
      <feature Name="Ratio of Memory Leaks" Threshold="&lt; 0.2"/>
    </property>
    <property Name="Low CC">
      <feature Name="Ratio Low CC" Threshold="&lt; 0.3"/>
    </property>
    <property Name="Low FP">
      <feature Name="Ratio Low FP" Threshold="&lt; 0.3"/>
    </property>
    <property Name="High Cohesion">
      <feature Name="Ratio High Cohesion" Threshold="> 0.7"/>
    </property>
    <property Name="High Back Up Utilities">
      <feature Name="Back Up Utilities Ratio" Threshold="> 0.5"/>
    </property>
    <property Name="High Applications Redundancy">
      <feature Name="Ratio High Apps Redundancy" Threshold="> 0.1"/>
    </property>
    <property Name="High Hardware Redundancy">
      <feature Name="Ratio High Hardware Redundancy" Threshold="> 0.1"/>
    </property>
    <property Name="Low Coupling">
      <feature Name="Ratio Low Coupling" Threshold="&lt; 0.4"/>
    </property>
    <property Name="Low Information Flow">
      <feature Name="Ratio of Information Flow" Threshold="&lt; 0.3"/>
    </property>
  </type>
</ConfFile:HypoCategory>
```

Σχήμα 6.5: Configuration XMI file for example 1

```
High_Back_Up_Utilities(Back_Up_Utilities_Ratio, D1)
High_Applications_Redundancy(Ratio_High_Apps_Redundancy, D2)
High_Hardware_Redundancy(Ratio_High_Hardware_Redundancy, D3)
Low_Coupling(Ratio_Low_Coupling, D4)
```

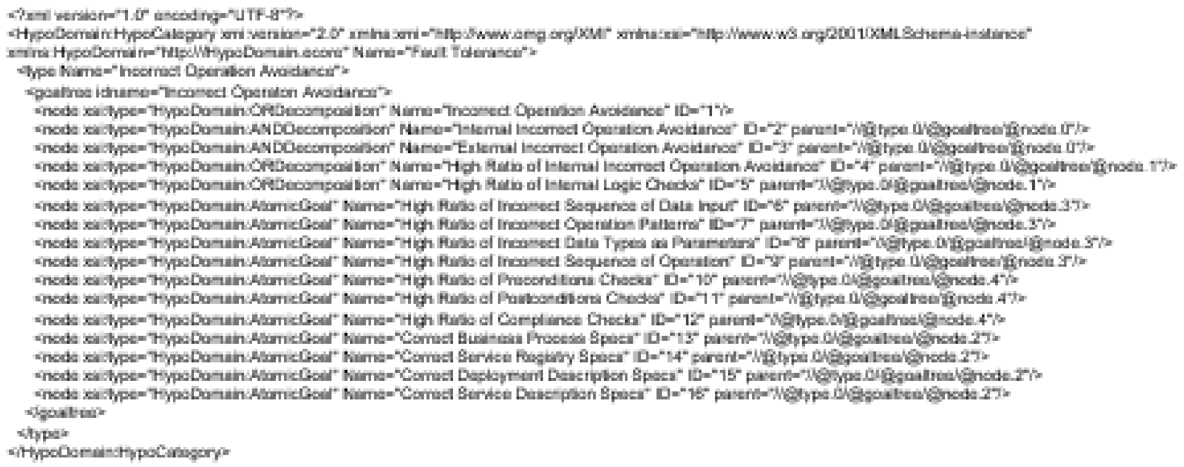
Σχήμα 6.6: entity0.db file for example 1

```
Failure_Avoidance(D1) 0.888961
Failure_Avoidance(D2) 0.864964
Failure_Avoidance(D3) 0.856964
Failure_Avoidance(D4) 0.855964
```

Σχήμα 6.7: Alchemy output for example 1

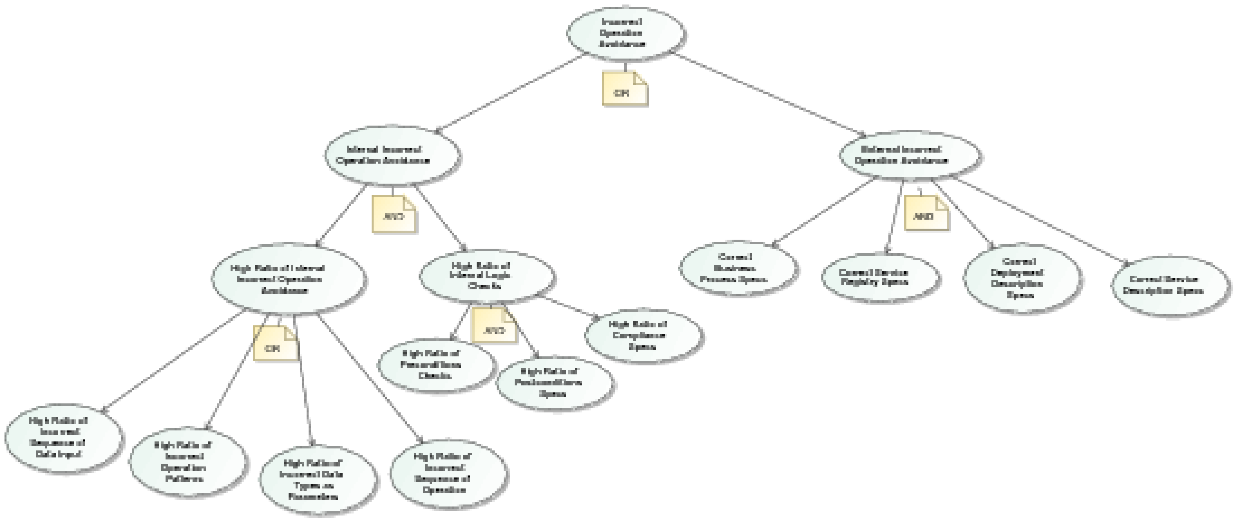
6.3 Example 3

In this third example, the Hypothesis Type to be validated is **Break Down Avoidance**. After populating the Hypothesis Domain Model using the Eclipse runtime environment, the XMI document which is also compliant to e - core, is shown in figure 6.15 below.



```
<?xml version="1.0" encoding="UTF-8"?>
<HypoDomain:HypoCategory xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:HypoDomain="http:///HypoDomain.ecore" Name="Fault Tolerance">
  <type Name="Incorrect Operation Avoidance">
   <goaltree idname="Incorrect Operation Avoidance">
    <node xsi:type="HypoDomain:ORDecomposition" Name="Incorrect Operation Avoidance" ID="1"/>
    <node xsi:type="HypoDomain:ANDDecomposition" Name="Internal Incorrect Operation Avoidance" ID="2" parent="//@type.0/@goaltree/@node.0"/>
    <node xsi:type="HypoDomain:ANDDecomposition" Name="External Incorrect Operation Avoidance" ID="3" parent="//@type.0/@goaltree/@node.0"/>
    <node xsi:type="HypoDomain:ORDecomposition" Name="High Ratio of Internal Incorrect Operation Avoidance" ID="4" parent="//@type.0/@goaltree/@node.1"/>
    <node xsi:type="HypoDomain:ORDecomposition" Name="High Ratio of Internal Logic Checks" ID="5" parent="//@type.0/@goaltree/@node.1"/>
    <node xsi:type="HypoDomain:AtomicGoal" Name="High Ratio of Incorrect Sequence of Data Input" ID="6" parent="//@type.0/@goaltree/@node.3"/>
    <node xsi:type="HypoDomain:AtomicGoal" Name="High Ratio of Incorrect Operation Patterns" ID="7" parent="//@type.0/@goaltree/@node.3"/>
    <node xsi:type="HypoDomain:AtomicGoal" Name="High Ratio of Incorrect Data Types as Parameters" ID="8" parent="//@type.0/@goaltree/@node.3"/>
    <node xsi:type="HypoDomain:AtomicGoal" Name="High Ratio of Incorrect Sequence of Operation" ID="9" parent="//@type.0/@goaltree/@node.3"/>
    <node xsi:type="HypoDomain:AtomicGoal" Name="High Ratio of Preconditions Checks" ID="10" parent="//@type.0/@goaltree/@node.4"/>
    <node xsi:type="HypoDomain:AtomicGoal" Name="High Ratio of Postconditions Checks" ID="11" parent="//@type.0/@goaltree/@node.4"/>
    <node xsi:type="HypoDomain:AtomicGoal" Name="High Ratio of Compliance Checks" ID="12" parent="//@type.0/@goaltree/@node.4"/>
    <node xsi:type="HypoDomain:AtomicGoal" Name="Correct Business Process Specs" ID="13" parent="//@type.0/@goaltree/@node.2"/>
    <node xsi:type="HypoDomain:AtomicGoal" Name="Correct Service Registry Specs" ID="14" parent="//@type.0/@goaltree/@node.2"/>
    <node xsi:type="HypoDomain:AtomicGoal" Name="Correct Deployment Description Specs" ID="15" parent="//@type.0/@goaltree/@node.2"/>
    <node xsi:type="HypoDomain:AtomicGoal" Name="Correct Service Description Specs" ID="16" parent="//@type.0/@goaltree/@node.2"/>
   </goaltree>
  </type>
</HypoDomain:HypoCategory>
```

Σχήμα 6.8: Hypothesis Domain XMI for example 2


Incorrect Operation Avoidance
OR
Internal Incorrect Operation Avoidance
External Incorrect Operation Avoidance
AND
AND
High Ratio of Internal Incorrect Operation Avoidance
High Ratio of Internal Logic Checks
Correct Business Process Specs
Correct Service Registry Specs
Correct Deployment Description Specs
Correct Service Description Specs
OR
AND
High Ratio of Compliance Specs
High Ratio of Preconditions Checks
High Ratio of Postconditions Specs
High Ratio of Incorrect Sequence of Data Input
High Ratio of Incorrect Operation Patterns
High Ratio of Incorrect Data Types as Parameters
High Ratio of Incorrect Sequence of Operation

Σχήμα 6.9: AND/OR Goal Tree of figure 6.8 XMI document

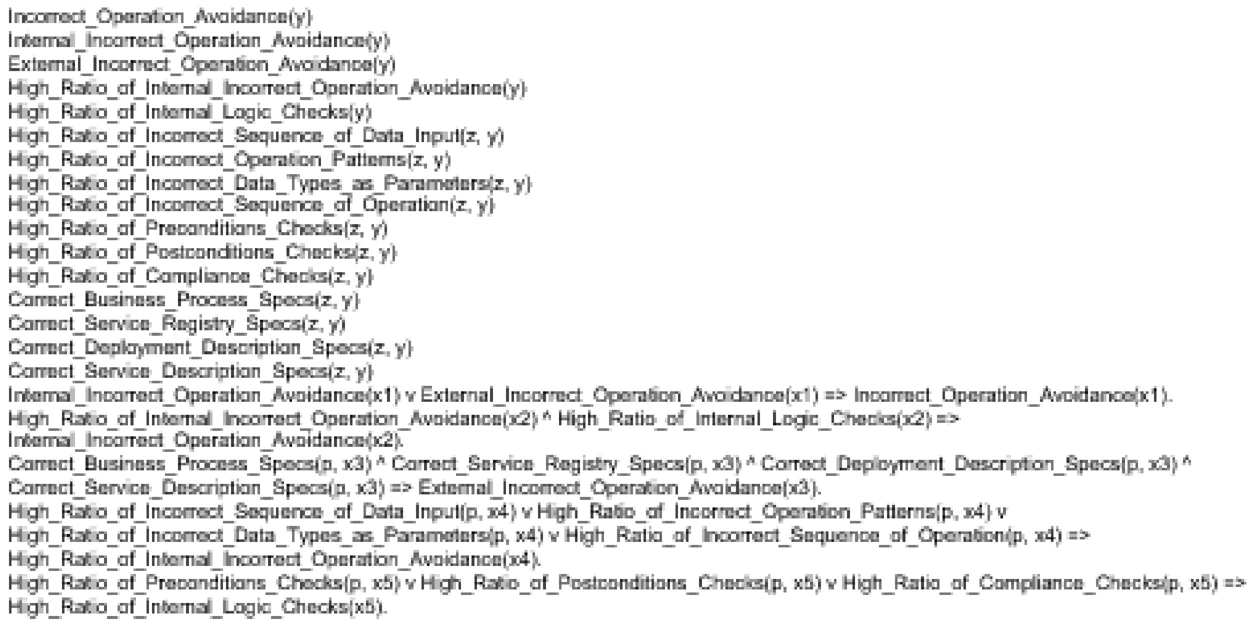
The corresponding AND/OR Goal Tree that is represented, appears in figure 6.16.

After launching the simulation, the *hypoth.mln* file produced which stands for the first - order logic world given as input to Alchemy, can be seen in figure 6.17.

The Data Warehouse XMI file populated again using the Eclipse runtime environment, can be seen in figure 6.18.

The Configuration XMI file populated for the purposes of this example, is shown in figure 6.19.

After the simulation launch, the *entity0.db* file for Module 3 produced including all the predicates that have a true boolean value according to the information extracted by the Data Warehouse and configuration file, can be seen in figure 6.20.

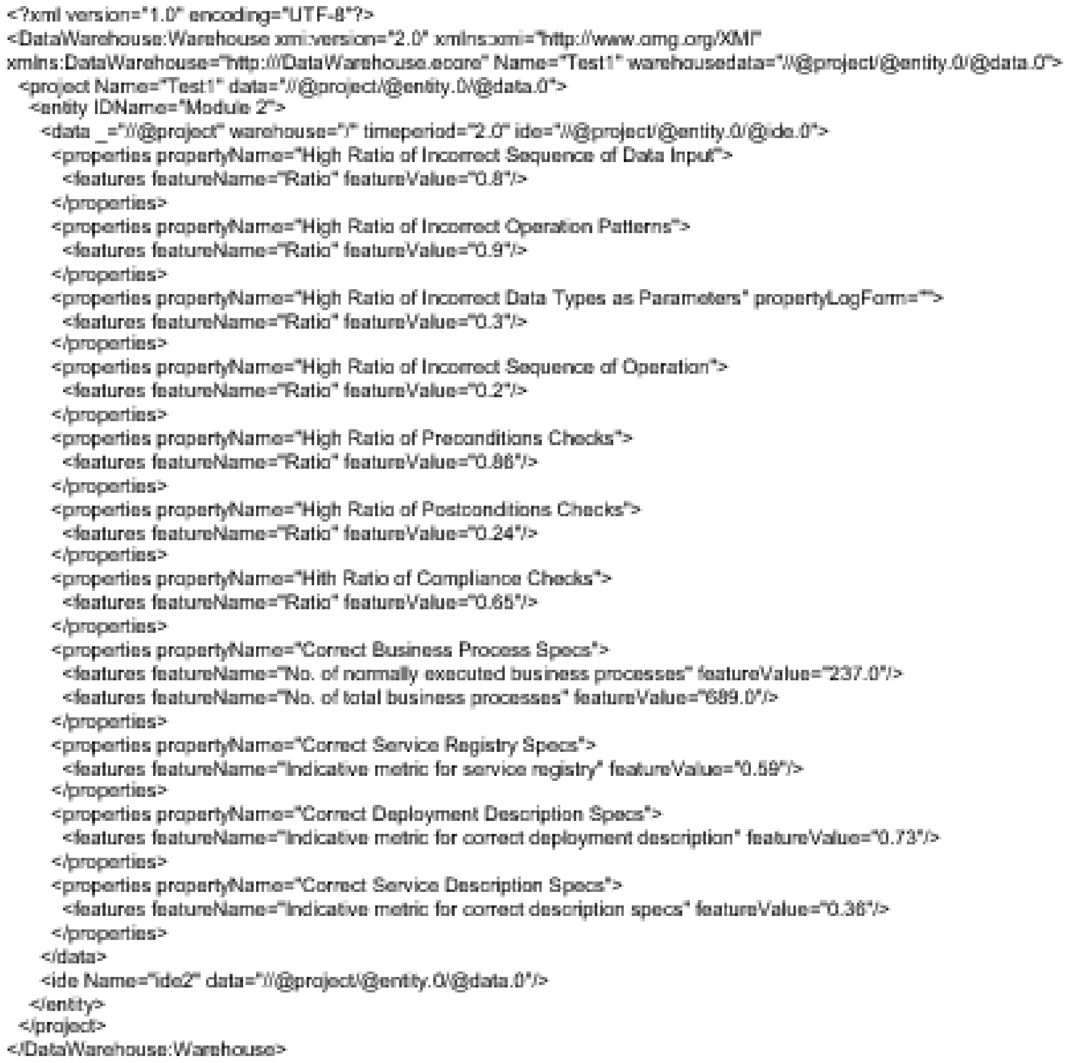


```
Incorrect_Operation_Avoidance(y)
Internal_Incorrect_Operation_Avoidance(y)
External_Incorrect_Operation_Avoidance(y)
High_Ratio_of_Internal_Incorrect_Operation_Avoidance(y)
High_Ratio_of_Internal_Logic_Checks(y)
High_Ratio_of_Incorrect_Sequence_of_Data_Input(z, y)
High_Ratio_of_Incorrect_Operation_Patterns(z, y)
High_Ratio_of_Incorrect_Data_Types_as_Parameters(z, y)
High_Ratio_of_Incorrect_Sequence_of_Operation(z, y)
High_Ratio_of_Preconditions_Checks(z, y)
High_Ratio_of_Postconditions_Checks(z, y)
High_Ratio_of_Compliance_Checks(z, y)
Correct_Business_Process_Specs(z, y)
Correct_Service_Registry_Specs(z, y)
Correct_Deployment_Description_Specs(z, y)
Correct_Service_Description_Specs(z, y)
Internal_Incorrect_Operation_Avoidance(x1) v External_Incorrect_Operation_Avoidance(x1) => Incorrect_Operation_Avoidance(x1).
High_Ratio_of_Internal_Incorrect_Operation_Avoidance(x2) ^ High_Ratio_of_Internal_Logic_Checks(x2) =>
Internal_Incorrect_Operation_Avoidance(x2).
Correct_Business_Process_Specs(p, x3) ^ Correct_Service_Registry_Specs(p, x3) ^ Correct_Deployment_Description_Specs(p, x3) ^
Correct_Service_Description_Specs(p, x3) => External_Incorrect_Operation_Avoidance(x3).
High_Ratio_of_Incorrect_Sequence_of_Data_Input(p, x4) v High_Ratio_of_Incorrect_Operation_Patterns(p, x4) v
High_Ratio_of_Incorrect_Data_Types_as_Parameters(p, x4) v High_Ratio_of_Incorrect_Sequence_of_Operation(p, x4) =>
High_Ratio_of_Internal_Incorrect_Operation_Avoidance(x4).
High_Ratio_of_Preconditions_Checks(p, x5) v High_Ratio_of_Postconditions_Checks(p, x5) v High_Ratio_of_Compliance_Checks(p, x5) =>
High_Ratio_of_Internal_Logic_Checks(x5).
```

Σχήμα 6.10: hypoth.mln file for example 2

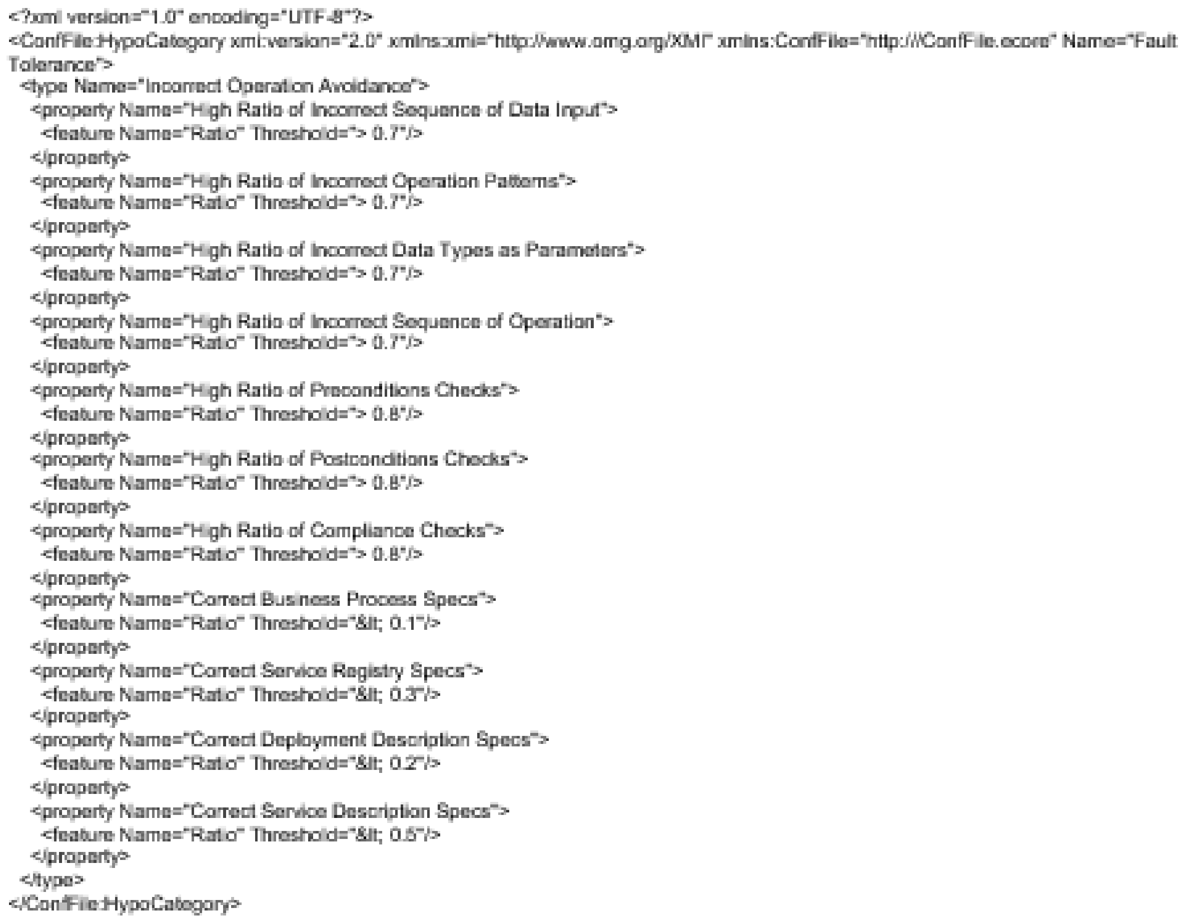
Finally, setting the previous two output files as input to the Alchemy tool, the following results appear as seen in figure 6.21.

From the results of the Alchemy tool, we can logically deduct that *Break Down Avoidance* has a probability of average 0.52 of occuring, that is the probability of a fault occuring which is connected with the project breaking down, is around 0.35 - 0.52 approximately. At this example, we can see that the probably of a fault occuring is a lot higher than at the previous two examples.



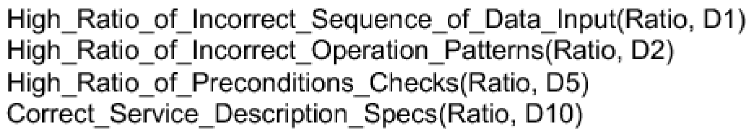
```
<?xml version="1.0" encoding="UTF-8"?>
<DataWarehouse:Warehouse xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:DataWarehouse="http:///DataWarehouse.ecore" Name="Test1" warehousedata="//@project/@entity.0/@data.0">
  <project Name="Test1" data="//@project/@entity.0/@data.0">
    <entity IDName="Module 2">
      <data _="//@project" warehouse="/" timeperiod="2.0" ide="//@project/@entity.0/@ide.0">
        <properties propertyName="High Ratio of Incorrect Sequence of Data Input">
          <features featureName="Ratio" featureValue="0.8"/>
        </properties>
        <properties propertyName="High Ratio of Incorrect Operation Patterns">
          <features featureName="Ratio" featureValue="0.9"/>
        </properties>
        <properties propertyName="High Ratio of Incorrect Data Types as Parameters" propertyLogForm="">
          <features featureName="Ratio" featureValue="0.3"/>
        </properties>
        <properties propertyName="High Ratio of Incorrect Sequence of Operation">
          <features featureName="Ratio" featureValue="0.2"/>
        </properties>
        <properties propertyName="High Ratio of Preconditions Checks">
          <features featureName="Ratio" featureValue="0.86"/>
        </properties>
        <properties propertyName="High Ratio of Postconditions Checks">
          <features featureName="Ratio" featureValue="0.24"/>
        </properties>
        <properties propertyName="Hith Ratio of Compliance Checks">
          <features featureName="Ratio" featureValue="0.65"/>
        </properties>
        <properties propertyName="Correct Business Process Specs">
          <features featureName="No. of normally executed business processes" featureValue="237.0"/>
          <features featureName="No. of total business processes" featureValue="689.0"/>
        </properties>
        <properties propertyName="Correct Service Registry Specs">
          <features featureName="Indicative metric for service registry" featureValue="0.59"/>
        </properties>
        <properties propertyName="Correct Deployment Description Specs">
          <features featureName="Indicative metric for correct deployment description" featureValue="0.73"/>
        </properties>
        <properties propertyName="Correct Service Description Specs">
          <features featureName="Indicative metric for correct description specs" featureValue="0.36"/>
        </properties>
      </data>
      <ide Name="ide2" data="//@project/@entity.0/@data.0"/>
    </entity>
  </project>
</DataWarehouse:Warehouse>
```

Σχήμα 6.11: Warehouse Domain XMI for example 2



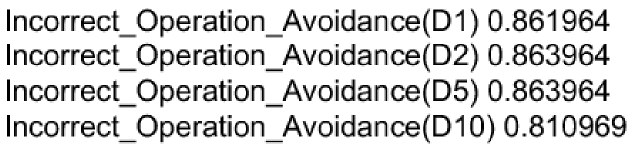
```
<?xml version="1.0" encoding="UTF-8"?>
<ConfFile:HypoCategory xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:ConfFile="http:///ConfFile.ecore" Name="Fault
Tolerance">
  <type Name="Incorrect Operation Avoidance">
    <property Name="High Ratio of Incorrect Sequence of Data Input">
      <feature Name="Ratio" Threshold="> 0.7"/>
    </property>
    <property Name="High Ratio of Incorrect Operation Patterns">
      <feature Name="Ratio" Threshold="> 0.7"/>
    </property>
    <property Name="High Ratio of Incorrect Data Types as Parameters">
      <feature Name="Ratio" Threshold="> 0.7"/>
    </property>
    <property Name="High Ratio of Incorrect Sequence of Operation">
      <feature Name="Ratio" Threshold="> 0.7"/>
    </property>
    <property Name="High Ratio of Preconditions Checks">
      <feature Name="Ratio" Threshold="> 0.8"/>
    </property>
    <property Name="High Ratio of Postconditions Checks">
      <feature Name="Ratio" Threshold="> 0.8"/>
    </property>
    <property Name="High Ratio of Compliance Checks">
      <feature Name="Ratio" Threshold="> 0.8"/>
    </property>
    <property Name="Correct Business Process Specs">
      <feature Name="Ratio" Threshold="&lt; 0.1"/>
    </property>
    <property Name="Correct Service Registry Specs">
      <feature Name="Ratio" Threshold="&lt; 0.3"/>
    </property>
    <property Name="Correct Deployment Description Specs">
      <feature Name="Ratio" Threshold="&lt; 0.2"/>
    </property>
    <property Name="Correct Service Description Specs">
      <feature Name="Ratio" Threshold="&lt; 0.5"/>
    </property>
  </type>
</ConfFile:HypoCategory>
```

Σχήμα 6.12: Configuration XMI file for example 2



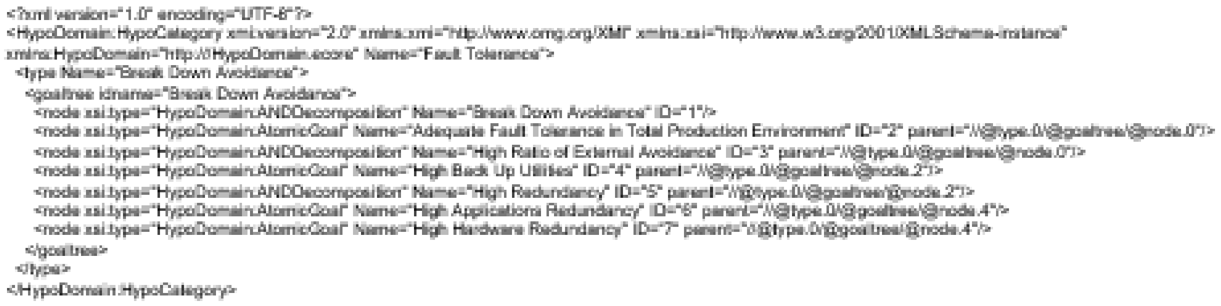
```
High_Ratio_of_Incorrect_Sequence_of_Data_Input(Ratio, D1)
High_Ratio_of_Incorrect_Operation_Patterns(Ratio, D2)
High_Ratio_of_Preconditions_Checks(Ratio, D5)
Correct_Service_Description_Specs(Ratio, D10)
```

Σχήμα 6.13: entity0.db file for example 2



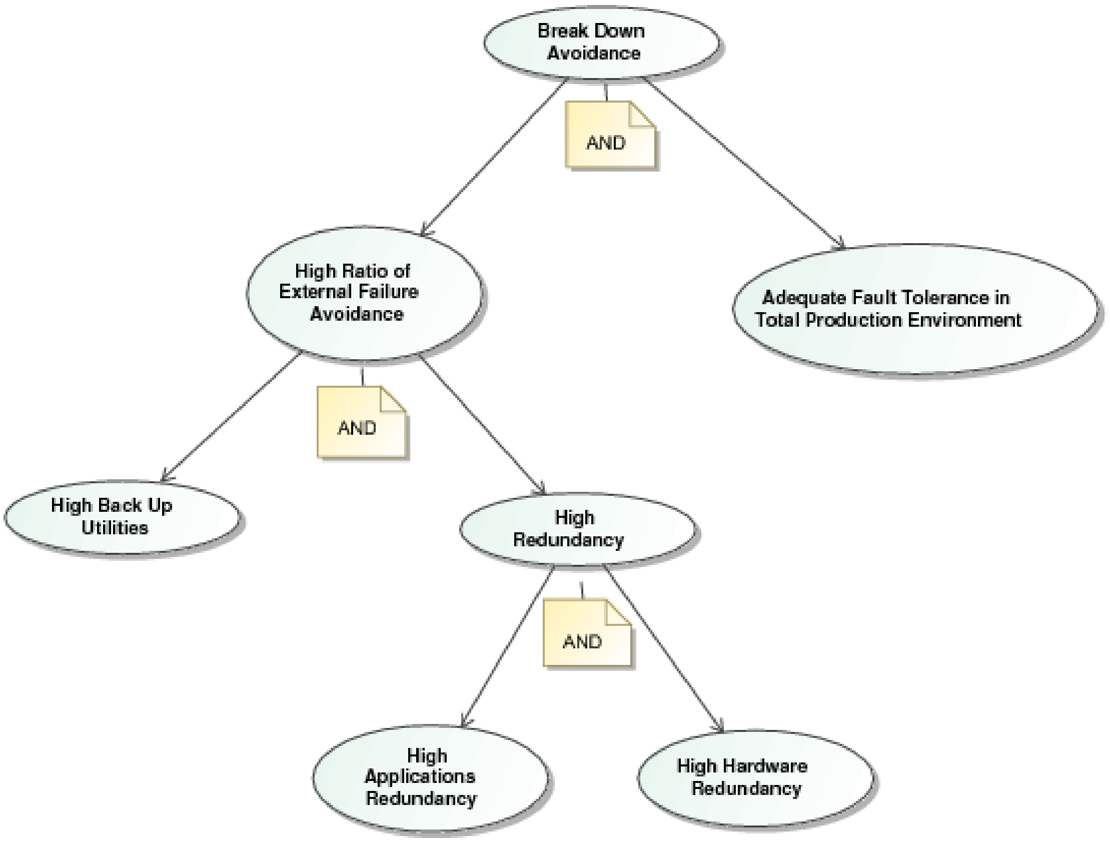
```
Incorrect_Operation_Avoidance(D1) 0.861964
Incorrect_Operation_Avoidance(D2) 0.863964
Incorrect_Operation_Avoidance(D5) 0.863964
Incorrect_Operation_Avoidance(D10) 0.810969
```

Σχήμα 6.14: Alchemy output for example 2

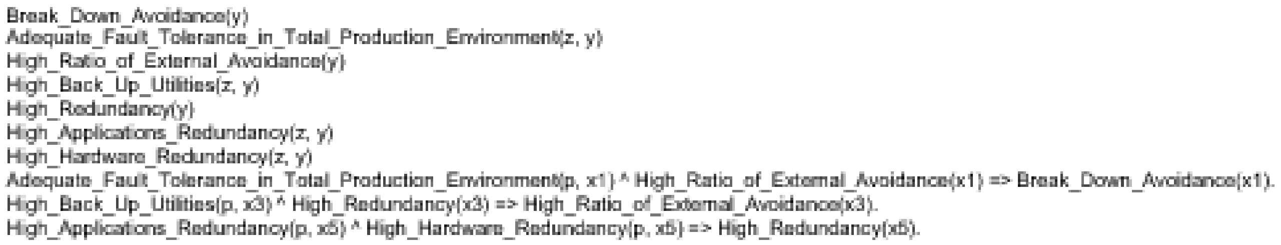


```
<?xml version="1.0" encoding="UTF-8"?>
<HypoDomain:HypoCategory xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:HypoDomain="http:///HypoDomain.ecore" Name="Fault Tolerance">
  <type Name="Break Down Avoidance">
    <goaltree idname="Break Down Avoidance">
      <node xsi:type="HypoDomain:ANDDecomposition" Name="Break Down Avoidance" ID="1"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="Adequate Fault Tolerance in Total Production Environment" ID="2" parent="//@type.0/@goaltree/@node.0"/>
      <node xsi:type="HypoDomain:ANDDecomposition" Name="High Ratio of External Avoidance" ID="3" parent="//@type.0/@goaltree/@node.0"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="High Back Up Utilities" ID="4" parent="//@type.0/@goaltree/@node.2"/>
      <node xsi:type="HypoDomain:ANDDecomposition" Name="High Redundancy" ID="5" parent="//@type.0/@goaltree/@node.2"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="High Applications Redundancy" ID="6" parent="//@type.0/@goaltree/@node.4"/>
      <node xsi:type="HypoDomain:AtomicGoal" Name="High Hardware Redundancy" ID="7" parent="//@type.0/@goaltree/@node.4"/>
    </goaltree>
  </type>
</HypoDomain:HypoCategory>
```

Σχήμα 6.15: Hypothesis Domain XMI for example 3

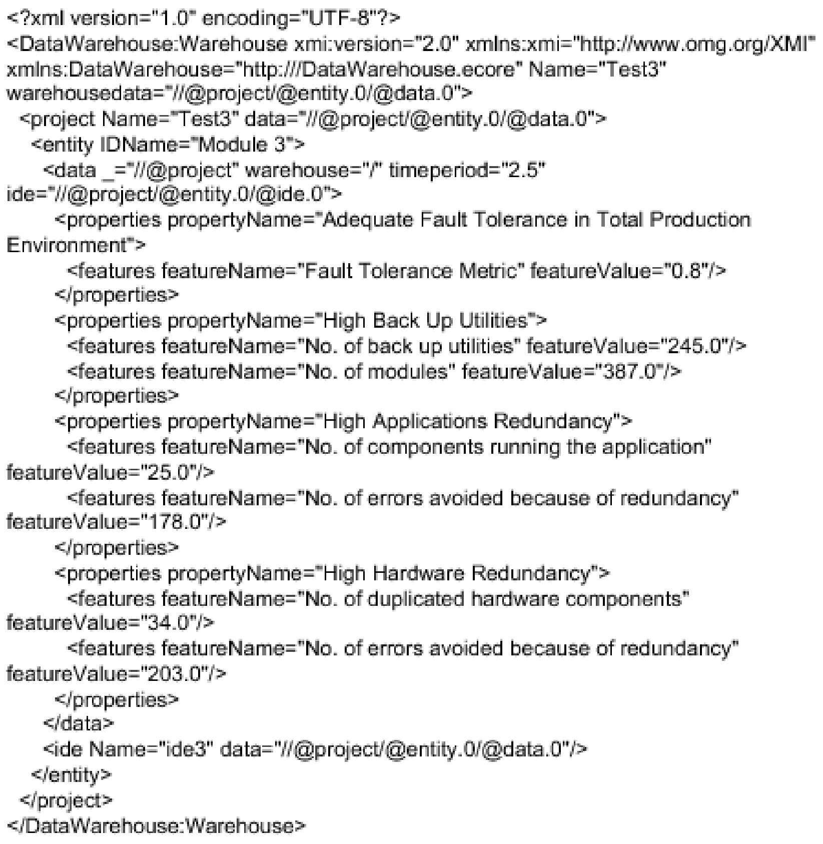

Break Down Avoidance
AND
High Ratio of External Failure Avoidance
Adequate Fault Tolerance in Total Production Environment
AND
High Back Up Utilities
High Redundancy
AND
High Applications Redundancy
High Hardware Redundancy

Σχήμα 6.16: AND/OR Goal Tree of figure 6.16 XMI document



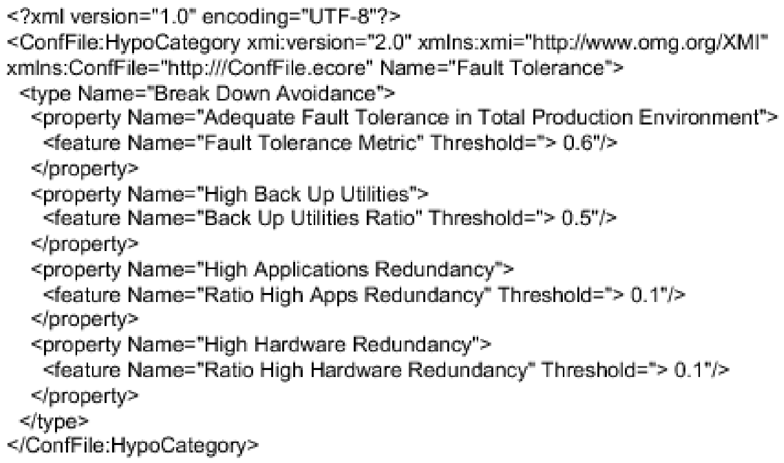
```
Break_Down_Avoidance(y)
Adequate_Fault_Tolerance_in_Total_Production_Environment(z, y)
High_Ratio_of_External_Avoidance(y)
High_Back_Up_Utilities(z, y)
High_Redundancy(y)
High_Applications_Redundancy(z, y)
High_Hardware_Redundancy(z, y)
Adequate_Fault_Tolerance_in_Total_Production_Environment(p, x1) ^ High_Ratio_of_External_Avoidance(x1) => Break_Down_Avoidance(x1).
High_Back_Up_Utilities(p, x3) ^ High_Redundancy(x3) => High_Ratio_of_External_Avoidance(x3).
High_Applications_Redundancy(p, x5) ^ High_Hardware_Redundancy(p, x5) => High_Redundancy(x5).
```

Σχήμα 6.17: hypoth.mln file for example 3



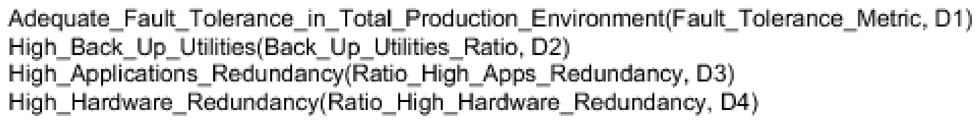
```
<?xml version="1.0" encoding="UTF-8"?>
<DataWarehouse:Warehouse xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:DataWarehouse="http:///DataWarehouse.ecore" Name="Test3"
warehousedata="//@project/@entity.0/@data.0">
  <project Name="Test3" data="//@project/@entity.0/@data.0">
    <entity IDName="Module 3">
      <data _="//@project" warehouse="/" timeperiod="2.5"
ide="//@project/@entity.0/@ide.0">
        <properties propertyName="Adequate Fault Tolerance in Total Production
Environment">
          <features featureName="Fault Tolerance Metric" featureValue="0.8"/>
        </properties>
        <properties propertyName="High Back Up Utilities">
          <features featureName="No. of back up utilities" featureValue="245.0"/>
          <features featureName="No. of modules" featureValue="387.0"/>
        </properties>
        <properties propertyName="High Applications Redundancy">
          <features featureName="No. of components running the application"
featureValue="25.0"/>
          <features featureName="No. of errors avoided because of redundancy"
featureValue="178.0"/>
        </properties>
        <properties propertyName="High Hardware Redundancy">
          <features featureName="No. of duplicated hardware components"
featureValue="34.0"/>
          <features featureName="No. of errors avoided because of redundancy"
featureValue="203.0"/>
        </properties>
      </data>
      <ide Name="ide3" data="//@project/@entity.0/@data.0"/>
    </entity>
  </project>
</DataWarehouse:Warehouse>
```

Σχήμα 6.18: Warehouse Domain XMI for example 3



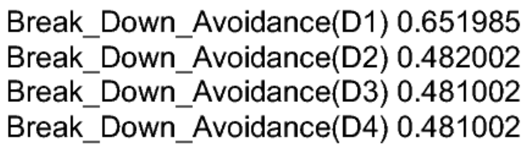
```
<?xml version="1.0" encoding="UTF-8"?>
<ConfFile:HypoCategory xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:ConfFile="http:///ConfFile.ecore" Name="Fault Tolerance">
  <type Name="Break Down Avoidance">
    <property Name="Adequate Fault Tolerance in Total Production Environment">
      <feature Name="Fault Tolerance Metric" Threshold="> 0.6"/>
    </property>
    <property Name="High Back Up Utilities">
      <feature Name="Back Up Utilities Ratio" Threshold="> 0.5"/>
    </property>
    <property Name="High Applications Redundancy">
      <feature Name="Ratio High Apps Redundancy" Threshold="> 0.1"/>
    </property>
    <property Name="High Hardware Redundancy">
      <feature Name="Ratio High Hardware Redundancy" Threshold="> 0.1"/>
    </property>
  </type>
</ConfFile:HypoCategory>
```

Σχήμα 6.19: Configuration XMI file for example 3



```
Adequate_Fault_Tolerance_in_Total_Production_Environment(Fault_Tolerance_Metric, D1)
High_Back_Up_Utilities(Back_Up_Utilities_Ratio, D2)
High_Applications_Redundancy(Ratio_High_Apps_Redundancy, D3)
High_Hardware_Redundancy(Ratio_High_Hardware_Redundancy, D4)
```

Σχήμα 6.20: entity0.db file for example 3



```
Break_Down_Avoidance(D1) 0.651985
Break_Down_Avoidance(D2) 0.482002
Break_Down_Avoidance(D3) 0.481002
Break_Down_Avoidance(D4) 0.481002
```

Σχήμα 6.21: Alchemy output for example 3

Κεφάλαιο 7

Conclusion

This diploma thesis focused on the problem of assisting Software Management operations for large software projects and more specifically to assist the project managers, team leaders and architects to obtain a better view and perspective of the status of their projects and even to identify potential risks or quality problems. An early prediction of possible faults that can lead to failures, can allow the planning of a potential avoidance strategy and, therefore, yield a better quality product.

The objective of this diploma thesis was, first, to investigate ways of modeling information obtained by various IDEs and Project Management tools and, second, to analyze this information so as for project managers to gather valuable information in order to assess the process of the development and make project related decisions. A third objective was to model risk avoidance and quality prediction policies using goal trees and a fourth was to utilize the MLN statistical reasoning framework to verify these risk avoidance and quality prediction policies. Towards this target, an extensible and end - user framework was developed in order to implement the above.

More specifically, data from various IDEs and project management tools are modeled using an extensible MOF compliant schema and stored in a centralized object oriented data repository. The data repository can be accessed by any tool that requires information. A pub/sub notification mechanism allows for different tools also to be notified whenever there is a change in the repository's state. Furthermore, logic pertaining to quality prediction or risk avoidance is encoded in the form of AND/OR Goal Trees. These trees through their different paths denote different ways of satisfying or denying the top goal that is the root node of the corresponding tree. As each Goal Tree node is associated with a First Order Logic expression containing user defined predicates, each path then can be denoted by a Conjunctive Normal Form (CNF) First Order Logic expression. Each predicate in such an expression can be verified or denied by a verifier component. As it was mentioned above, for this thesis we have chosen to utilize a statistical reasoning mechanism based on the Markov Logic Network theory. Markov Logic Networks allow for the evaluation of a probability value denoting the confidence by which a predicate or property holds given a set of constraints and a First Order Logic Knowledge Base. For evaluation purposes we have chosen to encode quality assessment policies that stem from the ISO 9126 software quality standard.

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 was aiming to offer flexibility and extensibility to the system.

In order to accomplish the verification of policies we utilized, it was also necessary to define the domain models introduced in chapter 4. The Data Warehouse Domain Model simulated the data provided originally by the developers, concerning IDEs or other Software Management projects. The Hypothesis Domain Model exploited the flexibility offered by the AND/OR Goal Tree representation structure in order to effectively model the hypotheses the user desires to check for validity. Finally, the Markov Logic Networkds theory was adopted in order to gain softer constraints for the first - order logic world and to calculate probabilities for the possible states of the world.

Through the development of this framework, this diploma thesis focused on handling questions such as whether it is possible to apply Business Intelligence methods into Software Engineering Management and Quality Control in order to accomplish a dynamic quality check based on information gathered from IDEs or if this approach of analyzing and projecting business data can enable the performance of fault and risk prediction with some measure of certainty. In order for this to be accomplished, various fields were investigated such as the first - order logic theory, the Markov Logic Networks theory, the AND/OR Goal Trees theory as well as the ISO International Standards.

7.1 Future Work

As mentioned above, the framework was developed in such a way that extensibility is ensured. Therefore, its actual structure enables various future additions and optimizations. One necessary optimization would be to create a friendly User Interface in order for the framework to be completely end - user and to fulfil its purpose as an independent and autonomic system, apart from the simulation within this diploma thesis.

Another possible extension that should be taken into consideration, is the expansion of the quality assessment features. For the purposes of this diploma thesis and the creation of a pilote prototype, focus was put mostly on the ISO Quality Standards. In the future, a very beneficial expansion would be to enhance the framework with customization option, that is, the user/developer will have the ability to customize the verification process according to his or her demands and criteria. The benefits of such an extension are pointed in [20]. To this direction, the Business Intellegence area of *Profiling* can prove very useful and, thus, should be looked into.

The technique used that is based on the Markov Logic Networks theory also has optimization potential. For the purposes of this diploma thesis, first - order logic constraints that constitute the first - order logic world on which the MLN construction is based, are considered equally probable. This assumption may not always hold. Therefore, a future target would be the creation of an algorithm that will assign weights to each predicate of the first - order logic world. One first estimation of suitable weights for the predicates, would be the computation of the difference of each feature value corresponding to one predicate, and the appropriate acceptance threshold of the same feature. Other verification methods, apart from the Markov Logic Networks, can also be considered and researched.

Finally, the next steps include, to complete development of the framework as stand - alone application integrated to data collection tools and software so as to assist developers to obtain a more comprehensive perspective over the state of their projects and to gain knowledge on probable risks and errors.

Βιβλιογραφία

[1] Jay Ramanathan Rajiv Ramnath Aman Kumar, Preethi Raghavan. 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.

[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] 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.

[4] 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.

[5] 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.

[6] 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.

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

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

[9] JIN Yongqin CHEN Qingzhang LI Jun, LIN Jianming1. Development of the decision support system for software project cost estimation. In *2008 International Symposium on Information Science and Engieering*, pages 299--302, 2008.

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

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

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

[13] 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.

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

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

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

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

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

[19] M. R. Bhashyam M. Ramakrishnan Pankaj Jalote, K. Dinesh. Quantitative quality management through defect prediction and statistical process control. Technical report, 2000.

[20] W. M. Pratt. Experiences in the application of customer - based metrics in impoving software service quality. In *IEEE International Conference on Communications*, volume 3, pages 1459--1462, 1991.

[21] 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.

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

[23] Hamed Ahmadi Ali Kamandi Shahrouz Moaven, Jafar Habibi. A decision support system for software architecture-style selection. In *Sixth International Conference on Software Engineering Research, Management and Applications*, pages 213--220, 2008.

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

[25] Chad Brower Jayavel Shanmugasundaram Sergei Vassilvitskii Erik Vee Ramana Yerneni Steven Euijong Whang, Hector Garcia-Molina. Indexing boolean expressions. *VLDB*, pages 37--48, 2009.

[26] 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.