Τεχνολογικό Εκπαιδευτικό Ίδρυμα Σερρών
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Πληροφορικής & Επικοινωνιών

# Πτυχιακή Εργασία
# **PRO.MA.SI**

Γρηγοριάδης Γρηγόριος 908

**Επιβλέπων καθηγητής**
Πεταλίδης Νικόλαος

**2008-2009**

This document was produced using the LaTeX document typesetting system

## Περίληψη

Σκοπός της πτυχιακής είναι η δημιουργία ενός εκπαιδευτικού παιχνιδιού, όπου ο χρήστης παίρνει το ρόλο ενός διαχειριστή έργου(project manager) μιας εταιρίας. Ο χρήστης καλείται να φέρει εις πέρας μια ιστορία, η οποία αποτελείται από πολλά επίπεδα. Σε κάθε επίπεδο ανατίθεται στο χρήστη ένα έργο(project) το οποίο πρέπει να ολοκληρώσει. Για να φέρει εις πέρας το έργο, ο χρήστης θα πρέπει να διαχειριστεί μια ομάδα ατόμων. Μόλις ολοκληρωθεί το έργο, ο χρήστης βαθμολογείται ανάλογα με τις ενέργειες του. Το παιχνίδι δίνει τη δυνατότητα δημιουργίας καινούργιων ιστοριών και επιπέδων.

Στόχος του παιχνιδιού είναι να εκπαιδεύσει τον χρήστη και να τον προετοιμάσει έτσι ώστε να μπορεί να ανταπεξέλθει στις απαιτήσεις που έχει ο χώρος της διαχείρισης έργων(project management). Η κάθε ιστορία και το κάθε επίπεδο πρέπει να διδάσκουν στο χρήστη και διαφορετικά θέματα, έτσι ώστε ο χρήστης να βελτιώνεται σε κάθε επίπεδο.


***Υπεύθυνη δήλωση :*** *Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Πληροφορικής & Επικοινωνιών του Τ.Ε.Ι Σερρών.*

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους τους φίλους και συγγενείς για την συμπαράσταση που μου έδειξαν, τον καθηγητή Δημήτρη Κοτζίνο καθώς και τους φοιτητές: Αντώνη Χρονάκη, Αλέξη Θεοδωρίδη του Τ.Ε.Ι Σερρών που ασχολούνται με το ProMaSi και θα βοηθήσουν στη συνέχιση του και τον φοιτητή του Τ.Ε.Ι Σερρών Τάσο Παπάζογλου για τη σχεδίαση του http://promasi.gr.

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον Νίκο Πεταλίδη για την πολύτιμη βοήθεια του, που χωρίς αυτήν δεν θα ήταν δυνατή η ολοκλήρωση του ProMaSi .

# Περιεχόμενα


**1 Εισαγωγή** **6**

1.1 Διαχείριση Έργου (Project Management) . . . 7
1.2 Παρόμοια projects . . . 8
1.3 Στόχοι ProMaSi . . . 10
1.4 Δομή εγγράφου . . . 12

**2 System Dynamics** **13**

2.1 Εισαγωγή . . . 13
2.2 System dynamics . . . 14
2.3 Εφαρμογή στη εκπαίδευση διαχείρισης έργων λογισμικού . . . 14
2.4 Συμπεράσματα . . . 17

**3 Περιγραφή του ProMaSi** **19**

3.1 Εισαγωγή . . . 19
3.2 Ο Χρήστης(Project Manager) . . . 19
3.3 Play mode . . . 24
3.3.1 Single player score mode . . . 25
3.3.2 Multi player score mode . . . 25
3.4 Ο Εκπαιδευτής(Game Master) . . . 26

**4 Αρχιτεκτονική ProMaSi** **28**

4.1 Core(PSD) . . . 30
4.1.1 Υπολογισμός τιμών . . . 33
4.1.2 Παράδειγμα PSDμοντέλου . . . 35
4.2 Communication . . . 38
4.3 Model . . . 39
4.4 Shell . . . 40
4.5 UI . . . 42

**5 Δημιουργία σεναρίου** **43**

5.1 X-Path Expressions . . . 48
5.2 Core Designer . . . 50

**6 Περιβάλλον χρήστη(UI)** **58**

6.1 Ξεκινώντας το παιχνίδι . . . 58
6.2 Ανάθεση project . . . 60
6.3 Προετοιμασία . . . 62

6.3.1 Πρόσληψη υπαλλήλων . . . . . 62
6.3.2 Ανάθεση εργασιών . . . . . 63
6.3.3 Ξεκινώντας το project . . . . . 68
6.4 Παίζοντας . . . . . 68
6.5 Τελειώνοντας το project . . . . . 69

**7 Υλοποίηση ProMaSi** **71**
7.1 Υλοποίηση Communication επιπέδου . . . . . 71
7.2 Υλοποίηση Core επιπέδου . . . . . 74
7.3 Υλοποίηση Model επιπέδου . . . . . 82
7.4 Υλοποίηση Shell επιπέδου . . . . . 85

**8 Παρατηρήσεις** **92**

**9 Συμπεράσματα** **94**
9.1 Μέλλον του project . . . . . 94
9.2 Στόχοι . . . . . 94
9.3 Παρατηρήσεις . . . . . 95

**Παράρτημα** **97**

**Α΄ Αποφάσεις για τα εργαλεία του έργου** **97**

**Β΄ Project Outline and Libraries** **99**

**Γ΄ Programming Guidelines** **102**

**Δ΄ Source Control and Build Management** **108**

# 1 Εισαγωγή

Η διαχείριση έργων είναι ένας από τους πιο σημαντικούς και πιο δύσκολους τομείς του management. Οι καθυστερήσεις και οι υπερβάσεις κόστους είναι ο κανόνας παρά η εξαίρεση. Υπερβάσεις κόστους της τάξεως 100% εώς 200% είναι κοινές. Τα έργα καθυστερούν τόσο πολύ που σε αρκετές περιπτώσεις οι συνθήκες αγοράς για τις οποίες σχεδιάστηκαν έχουν αλλάξει. Οι απαιτήσεις του πελάτη αλλάζουν συχνά, δημιουργώντας καθυστέρηση, αύξηση του κόστους και αναστάτωση στην εταιρεία. Έργα που συχνά φαίνεται να πηγαίνουν καλά, εμφανίζουν προβλήματα λίγο πριν το τέλος, δημιουργώντας υπερωρίες, προσλήψεις, καθυστερήσεις, μειώσεις στην ποιότητα και στα χαρακτηριστικά του έργου. Οι δυσκολίες αυτές έχουν επίπτωση στο κύρος, στα κέρδη και στο μερίδιο της αγοράς μιας εταιρείας.

Επιπροσθέτως και η εκπαίδευση των μελλοντικών project managers προκειμένου να αποφύγουν τα προβλήματα που αναφέρθηκαν παραπάνω παρουσιάζει δυσκολίες. Ειδικότερα δε στον τομέα του λογισμικού, η διδασκαλία θεμάτων διαχείρισης έργων θεωρείται από τα πιο δύσκολα αντικείμενα της εκπαίδευσης ενός μηχανικού λογισμικού. Πολλοί συγγραφείς έχουν αναφέρει αυτές τις δυσκολίες [1, 2] και η βιομηχανία λογισμικού διαμαρτυρήθηκε αρκετές φορές για το επίπεδο της εκπαίδευσης των μελλοντικών project managers [3, 4, 5].

Το πρόβλημα της εκπαίδευσης ενός manager έγκειται στο γεγονός ότι οι κανόνες του project management είναι σημαντικοί για την ανάπτυξη μεγάλων έργων τα οποία πρέπει να συντηρηθούν αλλά συνήθως είναι μη παραγωγικοί για την ανάπτυξη μικρών έργων. Από την άλλη μεριά οι φοιτητές δεν έχουν την εμπειρία που χρειάζεται για να καταλάβουν το πόσο σημαντικός είναι ο τομέας της διαχείρισης ενώ τα projects που τους δίνονται ως εργασίες στη διάρκεια ενός εξαμήνου είναι αναγκαστικά πολύ μικρά και μη-ρεαλιστικά. Έτσι οι περισσότεροι φοιτητές επικεντρώνονται στο τεχνικό κομμάτι τους, αδιαφορώντας για τα θέματα διαχείρισής τους.

Μια λύση σε αυτό το πρόβλημα είναι η χρήση εξομοιωτών διαχείρισης έργων κατά τη διδασκαλία. Οι εξομοιωτές επιτρέπουν στους σπουδαστές να διαχειριστούν ένα εικονικό έργο, απαλλάσσοντας τους από την προσήλωση στο τεχνικό κομμάτι, ενώ παράλληλα τους παρουσιάζουν προβλήματα που δε θα προέκυπταν σε ένα τυπικό εξάμηνο. Έτσι βοηθούν τους μαθητές να καταλάβουν πώς αναπτύσσεται ένα project και πώς οι πράξεις τους επηρεάζουν το project, δίνοντας τους μια πιο ρεαλιστική προσέγγιση.

Η πτυχιακή αυτή παρουσιάζει έναν τέτοιο εξομοιωτή που θα μπορούσε, εν καιρώ, να χρησιμοποιηθεί για το σκοπό αυτό.

## 1.1 Διαχείριση Έργου (Project Management)

*Τι είναι το project management και γιατί χρειαζόμαστε project managers;*
Το project management είναι ένα σύνολο από μεθόδους και τεχνικές, οι οποίες ορίζουν το πώς θα γίνεται η διαχείριση μιας ομάδας ατόμων με σκοπό την ολοκλήρωση μιας σειράς εργασιών (έργο) , μέσα σε ένα συγκεκριμένο χρονικό διάστημα και με συγκεκριμένο προϋπολογισμό.

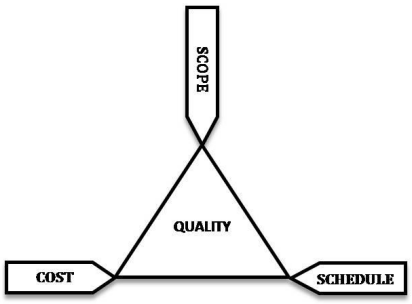
Ο project manager είναι υπεύθυνος για την ποιότητα του προϊόντος (product). Για να έχει το προϊόν υψηλή ποιότητα, ο project manager πρέπει να "ρυθμίσει" το συνολικό κόστος, το χρόνο και τα χαρακτηριστικά του προϊόντος. Οι επιπτώσεις που έχουν αυτοί οι παράγοντες φαίνονται με ένα τρίγωνο γνωστό και ως "Project management triangle" (Σχήμα 1).

**Schedule** είναι ο χρόνος που έχουμε για να ολοκληρώσουμε το έργο.

**Cost** είναι ο προϋπολογισμός που έχουμε διαθέσιμος για το έργο.

**Scope** είναι τα χαρακτηριστικά που πρέπει να έχει το έργο για να ολοκληρωθεί.

Αν αυξήσουμε τα χαρακτηριστικά του έργου (Scope), θα αυξηθεί ο χρόνος που χρειάζεται να ολοκληρωθεί το έργο και ο προϋπολογισμός. Αν μειώσουμε το χρόνο, θα αυξηθεί το κόστος και θα μειωθούν τα χαρακτηριστικά. Αν μειώσουμε τον προϋπολογισμό, θα αυξηθεί ο χρόνος και θα μειωθούν τα χαρακτηριστικά. Ο project manager πρέπει να οργανώσει την ομάδα του, να χρησιμοποιήσει τις κατάλληλες τεχνικές και εργαλεία, έτσι ώστε να μπορέσει να αποφύγει περιπτώσεις όπως:



Σχήμα 1: Project management triangle

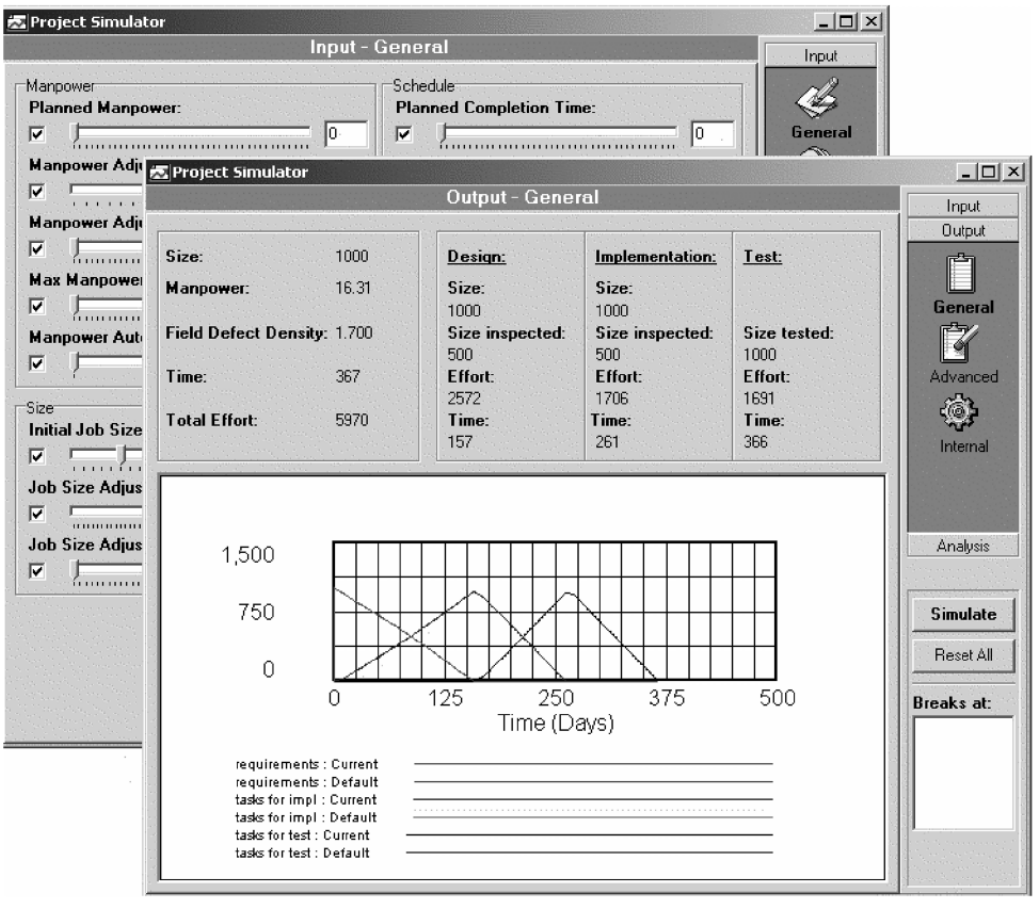
1. Προγράμματα που τελειώνουν αργά, υπερβαίνουν τον προϋπολογισμό ή δεν ικανοποιούν τις προσδοκίες των πελατών.

2. Επιτυχή προγράμματα, που ολοκληρώνονται με υψηλά επίπεδα στρες και υπερωρίες.

Η κατανόηση των σχέσεων αυτών των μεταβλητών είναι πολύ δύσκολη από τους φοιτητές διότι το κόστος δεν υπάρχει σαν παράμετρος στα project που τους δίνονται. Το schedule επίσης, είναι προκαθορισμένο από την αρχή του project και δεν μπορεί να αλλάξει. Έτσι, οι φοιτητές μπορούν να πειράξουν στην ουσία μία μόνο μεταβλητή, το scope. Σε μία εξομοίωση όμως, ο εκπαιδευόμενος είναι υποχρεωμένος να καθορίσει όλες αυτές τις μεταβλητές προκειμένου να πετύχει την υψηλότερη βαθμολογία.

## 1.2 Παρόμοια projects

Μέχρι τώρα λίγοι εξομοιωτές διαχείρισης έργου έχουν αναπτυχθεί. Οι περισσότεροι είναι για πρόβλεψη ή για post-mortem analysis παρά για εκπαιδευτικούς σκοπούς. Μερικοί από αυτούς είναι μοντέλα system dynamics [6] με λίγη διαδραστικότητα. Σε αυτούς, ο μαθητής βλέπει έναν αριθμό επιλογών οπού η αλλαγή της κάθε επιλογής επηρεάζει έναν αριθμό μεταβλητών του μοντέλου. Ο μαθητής βλέπει στο τέλος γραφικές παραστάσεις με αυτές τις αλλαγές (Σχήμα 2).




Σχήμα 2: Εικόνα project simulator [7]

Η παραπάνω προσέγγιση παρουσιάζει τα εξής προβλήματα:

Μειωμένη διαδραστικότητα, εφόσον ο χρήστης δεν μπορεί να αλλάξει τις επιλογές του αφού πάρει το αποτέλεσμα από το project

Ο μαθητής δεν περνάει από τη διαδικασία να ανακαλύπτει τις μεταβλητές που επηρεάζουν το αποτέλεσμα, αφού τις βλέπει όλες μέσω του γραφικού περιβάλλοντος.

Συνήθως το μοντέλο είναι δεμένο με το γραφικό περιβάλλον. Αυτό σημαίνει ότι για να εξομοιώσουμε διαφορετικά σενάρια πρέπει να φτιάξουμε το γραφικό περιβάλλον από την αρχή.

Το γραφικό περιβάλλον αυτών των εξομοιωτών δεν προσελκύει τους μαθητές γεγονός που τους κάνει δυσκολότερα αποδεκτούς από τους μαθητές.

Υπάρχει και ένας μικρός αριθμός εξομοιωτών οι οποίοι αυξάνουν την διαδραστικότητα με το χρήστη και το μοντέλο δεν είναι δεμένο με το γραφικό περιβάλλον. Οι εξομοιωτές που μελετήθηκαν και έχουν τα παραπάνω χαρακτηριστικά είναι:

1. **SESAM:** Είναι ο πρώτος εξομοιωτής που αναπτύχθηκε για εκπαιδευτικούς σκοπούς. Επιτρέπει τον ορισμό ενός στατικού μοντέλου που περιγράφει τα χαρακτηριστικά των ατόμων που δουλεύουν στο project καθώς και όλων των αντικειμένων που σχετίζονται με αυτό. Η συμπεριφορά των ατόμων και των αντικειμένων ορίζεται στο μοντέλο. Κάποιος κατασκευάζει το μοντέλο αυτό με τέτοιο τρόπο έτσι ώστε να αντιπροσωπεύει, έως ένα σημείο, την πραγματικότητα. Ο καθηγητής κατασκευάζει το μοντέλο και ο μαθητής παίζει στο μοντέλο αυτό προσλαμβάνοντας/απολύοντας υπαλλήλους, μιλώντας με τους υπαλλήλους και τον διευθυντή και αναθέτοντας εργασίες στους υπαλλήλους. Το SESAM χρησιμοποιεί τη δικιά του γλώσσα με την οποία μπορούμε να ορίσουμε τα έγγραφα, τους ανθρώπους κ.τ.λ. Μπορούμε να ορίσουμε τη συσχέτιση μεταξύ τους καθώς και να εφαρμόσουμε κάποιους κανόνες οι οποίοι καθορίζουν τις επιτρεπόμενες πράξεις.

2. **SimSE:** Είναι και αυτός ένας εξομοιωτής που χρησιμοποιείται σε μαθήματα του software project management. Το SimSE επιτρέπει τον ορισμό αντικειμένων, ενεργειών και κανόνων. Ένα αντικείμενο μπορεί να είναι ένας υπάλληλος, κάποιο έγγραφο, ένα εργαλείο, project ή κάποιος πελάτης. Οι ενέργειες σε ένα μοντέλο αναπαριστούν μία σειρά δραστηριοτήτων στην οποία μπορούν να συμμετέχουν κάποια αντικείμενα του μοντέλου. Οι κανόνες περιγράφουν τα αποτελέσματα και τις προϋποθέσεις για να γίνει μια ενέργεια. Το SimSE χρησιμοποιεί και αυτό τη δική του γλώσσα η οποία είναι παρόμοια με αυτή του SESAM. Η γλώσσα αυτή επιτρέπει τα αντικείμενα να συσχετιστούν με κάποιο γραφικό και τους κανόνες με την εμφάνιση κάποιου γραφικού. Με αυτόν τον τρόπο, το γραφικό περιβάλλον είναι ανεξάρτητο από το μοντέλο και μπορεί να χρησιμοποιηθεί με διαφορετικά μοντέλα.

Σε όλους τους παραπάνω εξομοιωτές τα βήματα μπορούν να περιγραφούν ως εξής:

1. Κατασκευή μοντέλου που αναπαριστά μια συγκεκριμένη διαδικασία ανάπτυξης λογισμικού.

2. Ανάθεση τιμών στο μοντέλο για την αναπαράσταση συγκεκριμένου σεναρίου.

3. Χρήση του παιχνιδιού (μοντέλου) από ένα μαθητή.

## 1.3 Στόχοι ProMaSi

Σύμφωνα με τα project που είδαμε και τις αρχικές μας ιδέες οι στόχοι που θέσαμε για το ProMaSi είναι οι εξής:

1. **Να είναι πλήρως παραμετροποιήσιμο, έτσι ώστε ο εκπαιδευτής να μπορεί να διδάξει στους εκπαιδευόμενους αυτό που θέλει.**

   Η γλώσσα που θα αναπτυχθεί για το ProMaSi θα πρέπει να είναι απόλυτα παραμετροποιήσιμη, έτσι ώστε να μπορεί ο εκπαιδευτής να φτιάξει ένα απλό μοντέλο έως και ένα μοντέλο όπου οι υπάλληλοι ζητάνε άδειες, αλλάζει η απόδοση τους ανάλογα με το φόρτο εργασίας κ.τ.λ.

2. **Να παρέχει τα κατάλληλα εργαλεία στον εκπαιδευτή, έτσι ώστε να μπορεί να δημιουργήσει εύκολα αυτό που θέλει.**

   Τα εργαλεία που θα παρέχει το ProMaSi θα πρέπει να είναι εύκολα στη χρήση τους και να καλύπτουν όλες τις απαιτήσεις που έχει το ProMaSi . Ο εκπαιδευτής θα πρέπει να μπορεί εύκολα να κατανοεί τη λειτουργία των εργαλείων και να μπορεί να δημιουργεί χωρίς να καταβάλει μεγάλες προσπάθειες ένα μοντέλο. Τα εργαλεία θα πρέπει να παρέχονται στην μορφή IDE και να λειτουργούν άριστα μεταξύ τους.

3. **Να δίνει μια αίσθηση ρεαλισμού στους εκπαιδευόμενους.**

   Ο χρήστης καθώς θα παίζει πρέπει να έχει την αίσθηση ότι δεν παίζει απλά ένα παιχνίδι αλλά ότι δουλεύει όντως σε μια εταιρεία από το σπίτι. Για να επιτευχθεί αυτό θα πρέπει η διεπαφή του χρήστη να μοιάζει με desktop ενός λειτουργικού συστήματος. Θα πρέπει να μπορεί ο χρήστης σε ένα μετέπειτα στάδιο του ProMaSi να μπορεί:

   (α′) Να οργανώνει code reviews με τους developers και να παρακολουθεί το review μέσω βίντεο.

   (β′) Να μιλάει με τους developers μέσω chat προγράμματος.

4. **Οι εκπαιδευόμενοι θα πρέπει να χρησιμοποιούν εργαλεία και μεθόδους που θα χρησιμοποιήσουν και στην πραγματικότητα, έτσι ώστε να είναι καλύτερα προετοιμασμένοι.**

   Για να αυξηθεί ο ρεαλισμός θα πρέπει ο χρήστης να χρησιμοποιεί εργαλεία που χρησιμοποιούν και οι project managers, όπως Gantt charts κτλ. Με αυτόν τον τρόπο δεν

αυξάνεται απλά ο ρεαλισμός του ProMaSi , αλλά ο χρήστης κατανοεί την χρησιμότητα αυτών των εργαλείων.

5. **Να κρατάει το ενδιαφέρον του εκπαιδευόμενου.**

   Στα περισσότερα παιχνίδια εξομοίωσης ο χρήστης κάνει μια ενέργεια και συνήθως περιμένει μέχρι να δει το αποτέλεσμα. Εξομοιωτές διαχείρισης έργου όπως το SimSE δεν αποτελούν εξαίρεση. Αυτό έχει ως αποτέλεσμα ο χρήστης να δαπανά περισσότερο χρόνο περιμένοντας παρά παίζοντας. Επιπλέον, στους εξομοιωτές διαχείρισης έργου ο χρήστης απλά παίζει κάθε ένα σενάριο ανεξάρτητα από τα προηγούμενα και δεν υπάρχει η έννοια της συνέχειας στο παιχνίδι. Κάτι τέτοιο περιορίζει το ενδιαφέρον του χρήστη.

   Το ProMaSi δεν θα αποτελείται από ένα μόνο σενάριο, αλλά από μια σειρά σεναρίων όπου στο τέλος κάθε σεναρίου ο χρήστης, ανάλογα με τη βαθμολογία του, θα μπορεί να προσλάβει πιο καλούς υπαλλήλους, θα έχει μεγαλύτερα budgets και θα αναπτύσσεται η εταιρεία του. Επιπλέον, το ProMaSi θα μπορεί να υποστηρίζει διαδικτυακά σενάρια, όπου οι χρήστες θα ανταγωνίζονται μεταξύ τους και όποιος πάρει τη μεγαλύτερη βαθμολογία θα παίρνει περισσότερα χρήματα, θα μπορεί να προσλάβει καλύτερους υπαλλήλους κ.τ.λ.

6. **Να αναπτυχθεί σαν open source πρόγραμμα.**

   Η ανάπτυξη προγραμμάτων με το μοντέλο του open source έχει αναδειχθεί σαν μία από τις καλύτερες μεθοδολογίες ανάπτυξης. Τα προγράμματα που αναπτύσσονται με αυτό το μοντέλο αναπτύσσονται πάρα πολύ γρήγορα και είναι πολύ καλά ποιοτικά. Μπορεί να συμμετάσχουν σε αυτά όποιοι προγραμματιστές επιθυμούν, ξεπερνώντας έτσι σε αρκετές περιπτώσεις το ανθρώπινο δυναμικό που έχουν μεγάλες εταιρείες. Επιπλέον, η χρήση των προγραμμάτων αυτών είναι δωρεάν. Επιλέχθηκε αυτή η μέθοδος για το ProMaSi γιατί είναι ένα πολύ μεγάλο project και ένα εκπαιδευτικό πρόγραμμα σαν αυτό θα πρέπει να έχει πολύ καλή ποιότητα και να διατίθεται δωρεάν.

7. **Να αναπτυχθεί με δωρεάν open source εργαλεία.**

   Όπως αναφέρθηκε παραπάνω, τα open source εργαλεία αναπτύσσονται γρήγορα και έχουν πολύ καλή ποιότητα. Το ProMaSi πρέπει να αναπτυχθεί με τέτοιου είδους εργαλεία γιατί δεν υποχρεώνονται αυτοί που θέλουν να ασχοληθούν με το project να αγοράσουν κάποιο συγκεκριμένο εργαλείο. Επιπρόσθετα, ένα άλλο χαρακτηριστικό των open source εργαλείων είναι ότι δουλεύουν σχεδόν σε όλα τα λειτουργικά συστήματα.

## 1.4 Δομή εγγράφου

**Κεφάλαιο 2** Αναφορά στα system dynamics, τι προβλήματα λύνουν και πως λειτουργούν.

**Κεφάλαιο 3** Απαιτήσεις του ProMaSi , τι ακριβώς θέλουμε να κάνει και ποιες οι δυνατότητές του.

**Κεφάλαιο 4** Ανάλύση αρχιτεκτονικής του ProMaSi . Πώς έχει χωριστεί το project για να μπορέσει να εξυπηρετήσει τις απαιτήσεις, τι είναι και πως λειτουργεί το κάθε επίπεδο του ProMaSi .

**Κεφάλαιο 5** Μια αναλυτική παρουσίαση για το πώς μπορεί ο εκπαιδευτής να φτιάξει ένα σενάριο έτσι ώστε να μπορούν να παίξουν οι μαθητές.

**Κεφάλαιο 6** Εγχειρίδιο χρήσης του ProMaSi για τους μαθητές. Ανάλυση του περιβάλλοντος χρήστη.

**Κεφάλαιο 7** Λεπτομέρειες για την υλοποίηση του ProMaSi . Παρουσίαση διαγραμμάτων κλάσεων και υλοποίησης των διαφορετικών επιπέδων του ProMaSi , κ.τ.λ.

**Κεφάλαιο 9** Συμπεράσματα που βγάλαμε από το ProMaSi . Η μελλοντική πορεία του project, στόχοι που πρέπει να ολοκληρωθούν.

# 2 System Dynamics

## 2.1 Εισαγωγή

Ο project manager προσπαθεί να λύσει ένα πρόβλημα: πώς με συγκεκριμένους περιορισμούς (προϋπολογισμό, χρόνο) θα μπορέσει να ολοκληρώσει ένα έργο με όλες τις απαιτήσεις και να πετύχει την καλύτερη ποιότητα χωρίς να παραβεί αυτούς τους περιορισμούς. Το πρόβλημα αυτό μπορεί να θεωρηθεί σαν ένα σύνηθες πρόβλημα βελτιστοποίησης. Τις περισσότερες φορές όμως μια τέτοια θεώρηση αποκλείει παραμέτρους οι οποίες σε μια πρώτη μάτια μπορεί να εμφανίζονται ασήμαντες, αλλά να αποδειχτούν στην πράξη σημαντικές. Ακόμα και αν κάποιος προσπαθήσει να θεωρήσει το πρόβλημα στην ολότητά του η πολυπλοκότητα του είναι τέτοια που καθιστά την επίλυσή του αδύνατη. Επιπροσθέτως τις περισσότερες φορές, φαινόμενα ανατροφοδότησης προκαλούν επιπλέον δυσκολία στην λύση του προβλήματος.

Ένα παράδειγμα μπορεί να κάνει το φαινόμενο αυτό πιο κατανοητό.

Μια εταιρεία αναλαμβάνει την εκτέλεση ενός έργου το οποίο σκοπεύει να υλοποιήσει με το υπάρχον έμπειρο προσωπικό της. Την ίδια στιγμή αλλάζει η νομοθεσία που αφορά τις προσλήψεις καθηγητών στη δημόσια εκπαίδευση. Τα δύο γεγονότα μπορεί να είναι ασύνδετα μεταξύ τους αλλά δύο μήνες μετά και ενώ το έργο είναι σε πλήρη εξέλιξη οι περισσότεροι από τους έμπειρους προγραμματιστές της εταιρείας δηλώνουν παραίτηση γιατί προσλαμβάνονται στο δημόσιο και η μείωση του εργατικού δυναμικού καθυστερεί το έργο. Ταυτόχρονα και η αναπλήρωσή του δυναμικού αποτελεί πρόβλημα γιατί η αθρόα εισαγωγή στο δημόσιο δημιουργεί έλλειψη έμπειρων προγραμματιστών. Η εταιρεία προσλαμβάνει άπειρους προγραμματιστές τους οποίους όμως πρέπει να εκπαιδεύσουν οι εναπομείναντες έμπειροι με αποτέλεσμα το έργο να καθυστερεί ακόμα περισσότερο. Η εταιρεία ζητά από το προσωπικό να δουλέψει υπερωρίες μέχρι να ολοκληρωθεί το έργο κάτι το οποίο αρχικά δείχνει να βελτιώνει την πρόοδο του έργου αλλά σύντομα καθώς οι υπάλληλοι αρχίζουν να κουράζονται, χάνουν το ενδιαφέρον τους και κάνουν περισσότερα λάθη στον κώδικα. Αυτό προκαλεί αύξηση της προσπάθειας που απαιτείται για να τελειώσει το έργο, η οποία με τη σειρά της αυξάνει και τις απαιτήσεις από τους προγραμματιστές. Τελικά από την πίεση το προσωπικό παραιτείται ή η παραγωγικότητά του πέφτει κατακόρυφα ενώ το έργο εξαιτίας των καθυστερήσεων ακυρώνεται.

Στο παραπάνω παράδειγμα είναι φανερό το εύρος των παραμέτρων που μπορεί να επηρεάσουν ένα έργο αλλά και το βαθμό στο οποίο η ανατροφοδότηση είναι υπεύθυνη για την αύξηση των προβλημάτων του έργου.

## 2.2 System dynamics

Το 1961 ο Jay Forrester [8] προτείνει μια νέα μέθοδο για τη μελέτη δυναμικών συστημάτων όπως αυτά που παρουσιάστηκαν προηγουμένως με άμεση εφαρμογή στον τομέα της βιομηχανίας.

Τα system dynamics όπως ονομάζονται είναι στην ουσία μια μέθοδος εξομοίωσης για την αναπαράσταση και μελέτη δυναμικών συστημάτων. Με τη λέξη *σύστημα* εδώ εννοούμε ο,τιδήποτε μπορεί να αποσυντεθεί σε ένα σύνολο από μέρη που εργάζονται για ένα κοινό σκοπό.

Στα system dynamics προκειμένου να μελετηθεί ένα σύστημα θεωρείται ότι, σε αντίθεση με την πρακτική που θέλει αυτό να υποδιαιρείται σε μικρότερα και απλούστερα υποσυστήματα, αυτό πρέπει να αντιμετωπιστεί ως *όλον* παίρνοντας υπόψη όλες τις παραμέτρους που το επηρεάζουν. Εξετάζουν δηλαδή συστήματα όπου το *όλον* είναι συνήθως διαφορετικό από το απλό άθροισμα των μερών του.

Ένα τέτοιο σύστημα αντιμετωπίζεται ως ένα σύνολο από μεταβλητές και παραμέτρους. Επιπλέον το σύστημα θεωρείται 'κλειστό' δηλαδή η τιμή μιας μεταβλητής μπορεί να εξαρτάται από τις τιμές που είχε στο παρελθόν η ίδια μεταβλητή, υπάρχει δηλαδή ανατροφοδότηση. Τέλος, θεωρείται συνεχές, δηλαδή οι μεταβλητές στην πάροδο του χρόνου έχουν συνεχείς και όχι διακριτές τιμές.

Τα system dynamics δεν εφαρμόστηκαν αμέσως στη μελέτη προβλημάτων που αφορούσαν έργα λογισμικού. Η πρώτη εφαρμογή τους στην ανάλυση των προβλημάτων έργων λογισμικού έγινε στο [9] όπου μελετήθηκε γενικά το πρόβλημα της στελέχωσης σε ένα έργο λογισμικού. Στη συνέχεια εφαρμόστηκαν σε διάφορα θέματα μελέτης των διαδικασιών ανάπτυξης όπως για παράδειγμα διαδικασίες ανασκόπησης, ελέγχου κτλ προκειμένου να μελετηθεί η βέλτιστη χρονική διάρκεια μιας ανασκόπησης, το καλύτερο δυνατό μέγεθος της ομάδας πιστοποίησης ποιότητας κτλ. Στις περισσότερες περιπτώσεις βεβαίως έχουν χρησιμοποιηθεί για να μελετήσουν είτε τα αίτια επιτυχίας ή αποτυχίας ενός έργου ή για να προβλέψουν την εξέλιξη ενός έργου. Η εφαρμογή τους για την εκπαίδευση μελλοντικών project managers είναι πιο περιορισμένη (για παράδειγμα [7]).

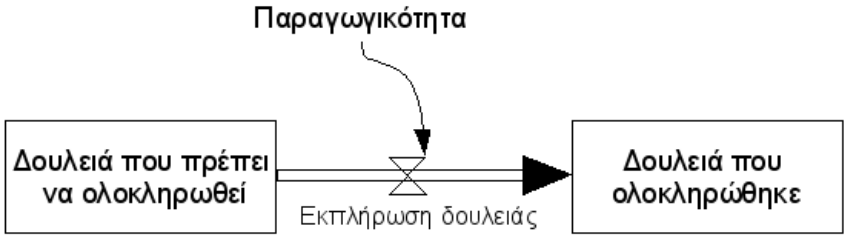
## 2.3 Εφαρμογή στη εκπαίδευση διαχείρισης έργων λογισμικού

Η εξέλιξη ενός έργου μπορεί να περιγράφει από ένα σύνολο από μεταβλητές οι οποίες στην πάροδο του χρόνου αλληλοεξαρτώνται και αλληλεπιδρούν μεταξύ τους. Το σύνολο των μεταβλητών αυτών αποτελεί ένα σύστημα. Κάποιες από τις μεταβλητές αυτές επηρεάζονται άμεσα από τον project manager ή από άλλες παραμέτρους του περιβάλλοντος και αποτελούν την είσοδο στο σύστημα, ενώ οι έξοδοι του συστήματος είναι οι μεταβλητές που μας ενδια-

φέρει να παρακολουθήσουμε όπως η ποιότητα του έργου και το ποσοστό των απαιτήσεων που έχει ολοκληρωθεί. Στο σύστημα υπάρχουν και κάποιες είσοδοι, όπως ο χρόνος και ο προϋπολογισμός, οι οποίες ορίζονται πριν ξεκινήσει το έργο και μπορεί να αλλάξουν στην συνέχεια. Με άλλα λόγια ένα έργο θα μπορούσε να θεωρηθεί ως ένα δυναμικό σύστημα που μπορεί να εξομοιωθεί με τη χρήση system dynamics.

Έχοντας αυτά υπόψη, στα πλαίσια της εκπαίδευσης ενός σπουδαστή θα μπορούσε ο εκπαιδευτής να φτιάχνει ένα μοντέλο του συστήματος ανάλογα με το τι θέλει να διδάξει. Για παράδειγμα, έστω ότι ο εκπαιδευτής θέλει να τονίσει ότι ο αριθμός των προσληφθέντων υπαλλήλων δεν είναι ανάλογος με την άμεση λύση ενός προβλήματος. Στην περίπτωση αυτή, θα συνθέσει ένα σύστημα το οποίο σαν βασική είσοδο θα έχει τον αριθμό των υπαλλήλων. Ανάλογα με τον αριθμό που θα προσλάβει ο χρήστης, αν ξεπερνάει κάποιο όριο θα έχει αρνητικά αποτελέσματα στην ποιότητα και στο χρόνο που θα ολοκληρωθεί το έργο. Ο εκπαιδευτής μπορεί να δημιουργήσει πολλά μοντέλα, το καθένα από τα οποία θα διδάσκει στον χρήστη και κάτι διαφορετικό. Το σύστημα αυτό μπορεί να είναι πολύ απλό, όπως αυτό που αναφέρθηκε παραπάνω, έως και πολύ σύνθετο. Έτσι, ο εκπαιδευτής μπορεί να ξεκινήσει με απλά συστήματα στην αρχή τα οποία, δείχνουν μία βασική συμπεριφορά και όσο προχωράει να συνδυάζει τις συμπεριφορές δημιουργώντας πολυπλοκότερα συστήματα.

Ο ρόλος του χρήστη στο παιχνίδι είναι να δίνει τιμές στις εισόδους του συστήματος, να παρακολουθεί το αποτέλεσμα και να αλλάζει τις τιμές εισόδου προκειμένου να πετύχει το καλύτερο αποτέλεσμα.




Σχήμα 3: Παράδειγμα αναπαράστασης συστήματος σε system dynamics

Στο Σχήμα 3, βλέπουμε την αναπαράσταση ενός πολύ απλού μοντέλου σε system dynamics. Στο παραπάνω σύστημα παρατηρούμε τις εξής οντότητες:

**Δουλειά που πρέπει να ολοκληρωθεί**: Οι γραμμές κώδικα που πρέπει να γραφτούν προκειμένου να τελειώσει το έργο.

**Δουλειά που ολοκληρώθηκε**: Οι γραμμές κώδικα που έχουν γραφτεί.

**Εκπλήρωση δουλειάς**: Οι γραμμές κώδικα που γράφονται ανά μέρα.

**Παραγωγικότητα**: Το σύνολο των γραμμών κώδικα που μπορούν να παράγουν όλοι οι υπάλληλοι ανά μήνα.

Η κάθε οντότητα αναπαριστά μία μεταβλητή του συστήματος. Όλες οι οντότητες αναπαριστώνται από ένα όνομα και από ένα σχήμα. Στο παραπάνω σχήμα, παρατηρούμε τριών ειδών μεταβλητές:

1. **Stock**: Αναπαρίσταται με ένα τετράγωνο (Δουλειά που πρέπει να ολοκληρωθεί, Δουλειά που ολοκληρώθηκε). Το stock αντιπροσωπεύει, μία οντότητα η οποία συσσωρεύεται ή εξαντλείται με την πάροδο του χρόνου. Έχει μία αρχική τιμή η οποία μεταβάλλεται. Για παράδειγμα, στο παραπάνω σύστημα η ΄Δουλειά που πρέπει να ολοκληρωθεί΄ έχει αρχική τιμή 5000 γραμμές κώδικα και με την πάροδο του χρόνου, ανάλογα με την Έκπλήρωση δουλειάς΄, μεταφέρονται στην ΄Δουλειά που ολοκληρώθηκε΄ η οποία έχει αρχική τιμή 0 γραμμές κώδικα.

2. **Flow**: Αναπαρίσταται με ένα βελάκι και μια κλεψύδρα (Εκπλήρωση δουλειάς). Το flow αντιπροσωπεύει τη μεταβολή ενός stock.

3. **Variable**: Αναπαρίσταται με ένα απλό κείμενο (Παραγωγικότητα). Η Variable αντιπροσωπεύει μια γενική μεταβλητή του συστήματος.

Η τιμή κάθε μεταβλητής υπολογίζεται από μια συνάρτηση όπως παρακάτω:

΄Δουλειά που πρέπει να ολοκληρωθεί΄ = -$\int_0^{t'}$Εκπλήρωση δουλειάς΄$dt$ γραμμές

΄Δουλειά που ολοκληρώθηκε΄ = $\int_0^{t'}$Εκπλήρωση δουλειάς΄$dt$ γραμμές

Έκπλήρωση δουλειάς΄ = (Παραγωγικότητα΄/23) * 0.7 (Θεωρούμε ότι οι υπάλληλοι δουλεύουν 23 μέρες το μήνα και παράγουν το 70% των δυνατοτήτων τους)

Παραγωγικότητα΄ = 2000 (γραμμές κώδικα το μήνα)

Αυτές οι τρεις μεταβλητές είναι και τα ουσιαστικά συστατικά στοιχεία όλων των μοντέλων system dynamics. Τα stocks αναπαριστούν τις σημαντικές μεταβλητές του συστήματος, τις τιμές των οποίων θέλουμε να παρακολουθούμε. Είναι μεταβλητές των οποίων η τιμή αλλάζει με την πάροδο του χρόνου συνήθως μετά από κάποια ενέργεια. Ένα flow σε ένα μοντέλο συνήθως αναπαριστά μια ενέργεια η οποία μεταβάλλει την τιμή ενός stock με ένα ρυθμό. Τέλος οι υπόλοιπες μεταβλητές χρησιμοποιούνται κυρίως για να βελτιώσουν την αναγνωσιμότητα του μοντέλου, σα χώρος αποθήκευσης ενδιάμεσων τιμών.

Για την εξομοίωση ενός τέτοιου συστήματος είναι αναγκαστικό να υπολογιστούν οι τιμές των μεταβλητών στις διάφορες χρονικές στιγμές. Αυτό μπορεί να γίνει με διάφορους τρόπους αλλά συνήθως τα βήματα είναι τα ακόλουθα: Το σύστημα υπολογίζει τις τιμές όλων των

μεταβλητών σε κάθε βήμα. Το βήμα στο συγκεκριμένο σύστημα είναι η μέρα. Μπορούμε να πούμε ότι το σύστημα θα τελειώσει την εξομοίωση μόλις η ΄Δουλειά που ολοκληρώθηκε΄ = 5000 γραμμές ή μόλις το βήμα = 90 μέρες (προθεσμία λήξης του έργου). Θα μελετήσουμε τα δύο πρώτα βήματα του συστήματος:

1. Βήμα 1 (1η μέρα): Πρώτα υπολογίζουμε τις μεταβλητές οι οποίες δεν εξαρτώνται από άλλες μεταβλητές, όπως ΄Δουλειά που πρέπει να ολοκληρωθεί΄, ΄Δουλειά που ολοκληρώθηκε΄, Παραγωγικότητα΄. Επειδή τα stocks έχουν αρχική τιμή, ο υπολογισμός της τιμής τους για το επόμενο βήμα γίνεται ένα βήμα πριν. Ξέρουμε από την αρχή ότι στο βήμα 1 ισχύουν τα εξής:

   ΄Δουλειά που πρέπει να ολοκληρωθεί΄=5000

   ΄Δουλειά που ολοκληρώθηκε΄=0 και

   Παραγωγικότητα΄ = 2000 (σταθερή)

   Με τη βοήθεια των εξισώσεων που προαναφέρθηκαν ισχύει ότι:

   Εκπλήρωση δουλειάς΄ = 60,86.

   Στη συνέχεια, υπολογίζουμε τις τιμές των stock για το επόμενο βήμα. Επομένως:

   ΄Δουλειά που πρέπει να ολοκληρωθεί΄ = 4939,14 και

   ΄Δουλειά που ολοκληρώθηκε΄ = 60,86.

2. Βήμα 2(2η μέρα): Ακολουθώντας τη λογική του 1ου βήματος έχουμε:

   Παραγωγικότητα΄=2000

   Εκπλήρωση δουλειάς΄ = 60.86

   ΄Δουλειά που ολοκληρώθηκε΄ = 121,72 και

   ΄Δουλειά που πρέπει να ολοκληρωθεί΄ = 4878,28

Σε αυτό το σύστημα, οι είσοδοι που έχει ο χρήστης είναι η Παραγωγικότητα΄, η οποία εξαρτάται από τους υπαλλήλους που έχει προσλάβει. Ο εκπαιδευτής καθορίζει τη δομή του συστήματος και τους περιορισμούς του. Οι περιορισμοί στο παραπάνω σύστημα είναι ο χρόνος που πρέπει να ολοκληρωθεί το έργο δηλαδή οι 90 ημέρες.

## 2.4 Συμπεράσματα

Δεν θα αναφερθούμε σε περισσότερες λεπτομέρειες για το πως ακριβώς δουλεύουν τα system dynamics γιατί δεν είναι το ουσιαστικό αντικείμενο της εργασίας αλλά θα αναφερθούμε

παρακάτω στο πως υλοποιήσαμε μια εκδοχή αυτών για τις ανάγκες του έργου μας. Επιλέξαμε όμως αυτήν τη μεθοδολογία γιατί είναι πάρα πολύ παραμετροποιήσιμη. Το μειονέκτημά της είναι ότι είναι αρκετά πολύπλοκο να φτιαχτούν μεγάλα μοντέλα. Για την εφαρμογή των system dynamics στον τομέα του project management έχουν γίνει και γίνονται έρευνες, όπως για παράδειγμα στο [10] και γίνονται αρκετές προσπάθειες για την παραγωγή και κατανόηση ρεαλιστικών μοντέλων διαχείρισης έργων λογισμικού.

Για την περαιτέρω κατανόηση και την κατασκευή μοντέλων χρησιμοποιήθηκε το Vensim [11]. Το Vensim είναι ένα πρόγραμμα κατασκευής μοντέλων system dynamics και εξομοιωτής ενώ διατίθεται δωρεάν για εκπαιδευτική χρήση και ακαδημαϊκή ή δημόσια έρευνα. Το Vensim μας έδωσε αρκετές ιδέες για το τι χρειάζεται να κάνουμε για να υλοποιήσουμε τα system dynamics.

Σκοπός όμως του ProMaSi δεν είναι η ακριβής υλοποίηση των system dynamics, για αυτό βασιστήκαμε πάνω στην βασική δομή τους και την αναπτύξαμε έτσι ώστε να ταιριάζει και να εξυπηρετεί τις ανάγκες του ProMaSi . Αυτήν την υλοποίηση την ονομάσαμε ProMaSi system dynamics(PSD). Το τρόπο λειτουργίας του PSD θα τον αναλύσουμε παρακάτω, αλλά η βασική ιδέα παραμένει η ίδια: ένα σύστημα αντιπροσωπεύει ένα έργο, το σύστημα αυτό έχει διάφορες εισόδους και εξόδους, ο χρήστης δίνει την είσοδο, παρακολουθεί την έξοδο και αλλάζει την είσοδο προκειμένου να πετύχει το καλύτερο αποτέλεσμα.

# 3 Περιγραφή του ProMaSi

## 3.1 Εισαγωγή

Το ProMaSi είναι ένα εκπαιδευτικό παιχνίδι που σκοπό έχει τη διδασκαλία βασικών αρχών της διαχείρισης έργων λογισμικού. Στο παιχνίδι αυτό υπάρχουν δύο βασικοί χαρακτήρες: Ο σπουδαστής, ο οποίος στη συνέχεια θα αναφέρεται απλά ως χρήστης ή project manager, και ο εκπαιδευτής ή game master. Ο εκπαιδευτής είναι υπεύθυνος για τη δημιουργία σεναρίων και ο χρήστης είναι υπεύθυνος να φέρει εις πέρας ένα σενάριο. Ένα σενάριο περιέχει, ανάμεσα σε άλλα πράγματα, την περιγραφή ενός έργου λογισμικού προς υλοποίηση, τις μεταβλητές που μπορεί να επηρεάσουν αυτό το έργο καθώς και τον τρόπο που η μία μεταβλητή επηρεάζει την άλλη. Η αναπαράσταση του σεναρίου γίνεται με κάποιο μοντέλο γραμμένο στη γλώσσα του ProMaSi η οποία βασίζεται στα system dynamics και περιγράφεται στα επόμενα κεφάλαια. Οι μεταβλητές που επηρεάζουν την εξέλιξη ενός σεναρίου επιλέγονται από τον εκπαιδευτή με βάση ένα ή περισσότερα θέματα διαχείρισης έργων λογισμικού που θα πρέπει να διδαχθούν οι σπουδαστές. Ο σκοπός του χρήστη είναι να ολοκληρώσει το έργο που περιγράφει το σενάριο, επιλέγοντας κατά την εξέλιξη του σεναρίου τις καλύτερες δυνατές τιμές για τις μεταβλητές αυτές. Πολλές από τις παραμέτρους του παιχνιδιού, δηλ. game play, όπως και ο τρόπος που υπολογίζεται η τελική βαθμολογία του χρήστη καθορίζεται από το play mode. Ένα play mode περιγράφει τους γενικούς κανόνες λειτουργίας, όπως για παράδειγμα το αν ο κάθε χρήστης θα παίζει μόνος του, ή θα συναγωνίζεται άλλους χρήστες που μπορεί να παίζουν παράλληλα.

Στη συνέχεια γίνεται αναλυτικότερη περιγραφή της λειτουργίας του παιχνιδιού.

## 3.2 Ο Χρήστης(Project Manager)

Ο χρήστης έχει το ρόλο του project manager, ο οποίος δουλεύει σε μια εταιρεία και προσπαθεί, αναλαμβάνοντας διάφορα projects, να ανεβάσει το κύρος της εταιρείας. Σκοπός του κάθε project είναι να διδάσκει στον χρήστη και μια διαφορετική πτυχή της διαχείρισης έργων λογισμικού.

Μια εταιρεία στο ProMaSi έχει τα εξής χαρακτηριστικά:

**Όνομα:** Το όνομα της εταιρείας.

**Prestige points:** Οι συνολικοί πόντοι που έχει κερδίσει απ΄ όλα τα projects.

**Περιγραφή:** Η περιγραφή της εταιρείας. Μπορεί να περιλαμβάνει την ιστορία της εταιρείας.

**Υπεύθυνος εταιρείας:** Ο προϊστάμενος του project manager (χρήστη), ο οποίος είναι υπεύθυνος για την ανάθεση έργων.

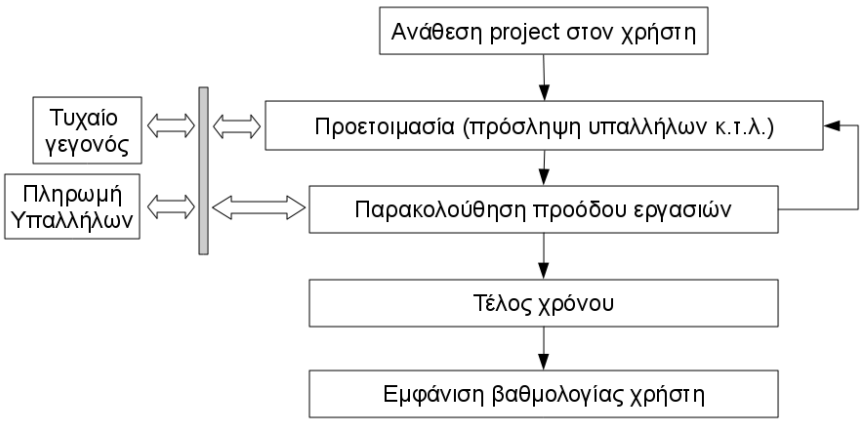
**Λογιστής:** Υπεύθυνος για την πληρωμή των υπαλλήλων.

**Διαχειριστής έργου (Project manager):** Ο ίδιος ο χρήστης, ο οποίος είναι υπεύθυνος για την επιλογή των υπαλλήλων και για το χρονο-προγραμματισμό των εργασιών.

**Υπάλληλοι:** Όλο το δυναμικό της εταιρείας.

**Ωράριο**

Η βασική ροή του παιχνιδιού φαίνεται στο Σχήμα 4. Τα παρακάτω βήματα είναι τα ίδια ανεξαρτήτως του play mode.




Σχήμα 4: Απλή ροή παιχνιδιού

**Ανάθεση project στον χρήστη:** Σε αυτό το βήμα ανατίθεται ένα project στον χρήστη. Ο χρήστης λαμβάνει ένα e-mail με τις λεπτομέρειες του project:

1. **Όνομα:** Το όνομα του project.
2. **Περιγραφή:** Η περιγραφή του project εξαρτάται από τον εκπαιδευτή για το πόσο λεπτομερής θα είναι. Ο εκπαιδευτής μπορεί να την γράψει σαν HTML κείμενο, οπότε η περιγραφή μπορεί να περιλαμβάνει και εικόνες.
3. **Προϋπολογισμός:** Ο προϋπολογισμός του project, το ποσό των χρημάτων που έχει ο χρήστης στην διάθεση του για να φέρει εις πέρας το project. Τα χρήματα πηγαίνουν κυρίως στους μισθούς των υπαλλήλων, αλλά σε μεταγενέστερες εκδόσεις μπορεί να δαπανώνται και για την αγορά εργαλείων, παγίων κτλ.
4. **Ημερομηνία έναρξης:** Η ημερομηνία έναρξης του project. Πριν την έναρξη δίνεται κάποιος χρόνος, ανάλογα με τη δυσκολία του project, για την προετοιμασία του χρήστη.

5. **Προθεσμία:** Η προθεσμία που έχει ο χρήστης μέχρι να τελειώσει το project.
6. **Διαθέσιμες εργασίες:** Το κάθε project έχει κάποιες εργασίες τις οποίες πρέπει να κάνει ο χρήστης. Η κάθε εργασία έχει έναν τίτλο και μία περιγραφή. Παραδείγματα εργασιών:
   - Ανάπτυξη κώδικα.
   - Ανασκόπηση κώδικα(code review).
   - Έλεγχος κώδικα(testing).
   - Επιδιόρθωση bug#312.
   - Ανάπτυξη χαρακτηριστικού #500.

   Οι εργασίες μπορεί να είναι πολύ γενικές (Ανάπτυξη κώδικα) έως και πολύ ειδικές (Επιδιόρθωση bug#312). Μερικές από αυτές μπορεί να είναι προαιρετικές π.χ. ο έλεγχος κώδικα και η ανασκόπηση και το project να χρειάζεται μόνο την ανάπτυξη κώδικα για να τελειώσει. Φυσικά όλες οι εργασίες έχουν κάποια επίπτωση στο project όπως για παράδειγμα στην ποιότητά του.

   Το είδος των εργασιών που επιτρέπεται να αναθέσει ο χρήστης στους υπαλλήλους του εξαρτάται από το σενάριο που έχει αναπτύξει ο εκπαιδευτής. Για παράδειγμα σενάριο που θέλει να αναδείξει την αξία των ανασκοπήσεων έχει περισσότερες εργασίες που σχετίζονται με τη διαδικασία ανασκοπήσεων κώδικα.

**Προετοιμασία (πρόσληψη υπαλλήλων κ.τ.λ.):** Κατά την προετοιμασία ο χρήστης μπορεί να κάνει τις ενέργειες:

- **Εμφάνιση στατιστικών των προηγούμενων έργων**. Μελετώντας τα στατιστικά των προηγούμενων έργων, ο χρήστης μπορεί να μελετήσει τις προηγούμενες μεθόδους που χρησιμοποίησε στα έργα στα οποία έλαβε μεγάλη βαθμολογία. Στα στατιστικά φαίνεται μία γραφική παράσταση από διάφορα μεγέθη (για παράδειγμα μεταβολή του προσωπικού στη διάρκεια του χρόνου) τα οποία θεωρούνται σημαντικά στη διαχείριση έργων. Ο εκπαιδευτής καθορίζει ποια από τα μεγέθη μπορεί να δει ο χρήστης. Ακόμη, ο χρήστης μπορεί να δει και τα στοιχεία του project(περιγραφή, προϋπολογισμός κ.τ.λ.) καθώς και το διάγραμμα Gantt που χρησιμοποίησε.
- **Πρόσληψη-Απόλυση υπαλλήλων.** Ανάλογα με το project, ο χρήστης πρέπει να αποφασίσει αν θα διώξει κάποιους υπαλλήλους ή θα προσλάβει κάποιους άλλους, λαμβάνοντας πάντα υπόψη τον προϋπολογισμό του project.
- **Προγραμματισμός εργασιών.** Ο χρήστης προγραμματίζει τις εργασίες χρησιμοποιώντας το πρόγραμμα Planner. Για κάθε κομμάτι του έργου μπορεί να

δημιουργήσει μία ή περισσότερες προγραμματισμένες εργασίες στις οποίες πρέπει να ορίσει τα εξής:

* Την ημερομηνία έναρξης και τέλους.
* Τους υπαλλήλους που θα ασχοληθούν με την εργασία.
* Τον αριθμός των ωρών αναμένεται να δαπανήσει ο κάθε υπάλληλος για την εργασία.

Ο χρήστης μπορεί να αναθέσει έναν υπάλληλο σε παραπάνω από μία εργασία. Για παράδειγμα μπορεί να αναθέσει έναν υπάλληλο για 3 ώρες στην εργασία 1 και 5 ώρες στην εργασία 2.

**Παρακολούθηση προόδου εργασιών:** Ο χρήστης παρακολουθεί την πρόοδο των εργασιών μέσω e-mails από τους υπαλλήλους. Ένας υπάλληλος στέλνει e-mail στον χρήστη για την πρόοδο της εργασίας που του έχει ανατεθεί. Ανάλογα με την πρόοδο της κάθε εργασίας ο χρήστης μπορεί να κάνει κάποιο βήμα από την προετοιμασία (πρόσληψη, απόλυση, προγραμματισμός εργασιών κ.τ.λ.).

**Τυχαίο γεγονός:** Τα τυχαία γεγονότα συμβαίνουν κατά τη διάρκεια του project και ορίζονται από τον εκπαιδευτή. Τέτοια γεγονότα εξαρτώνται ή επιδρούν σε κάποιες μεταβλητές του μοντέλου. Όταν ενεργοποιηθεί το γεγονός τότε ειδοποιείται ο χρήστης μέσω e-mail. Παραδείγματα γεγονότων αποτελούν η αύξηση απαιτήσεων, η πρόσθεση ενός καινούργιου χαρακτηριστικού στο project κ. α. Στο γεγονός αυτό ο χρήστης μπορεί να προγραμματίσει τις εργασίες του project ή να κάνει κάποια άλλη ενέργεια από την προετοιμασία.

**Πληρωμή Υπαλλήλων:** Η πληρωμή των υπαλλήλων γίνεται κάθε μέρα από τον λογιστή της εταιρείας. Μόλις γίνει στέλνεται ένα e-mail στον χρήστη με το ποσό το οποίο ξοδεύτηκε, το οποίο αφαιρείται από τον προϋπολογισμό του project. Ο χρήστης μπορεί είτε να προσλάβει είτε να απολύσει υπαλλήλους. Αν δε, ο προϋπολογισμός είναι επαρκής συνεχίζει την παρακολούθηση προόδου των εργασιών ή την προετοιμασία.

**Τέλος εργασιών/ Τέλος χρόνου:** Ο ορισμός για το πότε τελειώνει μια εργασία ορίζεται από τον εκπαιδευτή όταν φτιάχνει το σενάριο. Ο χρήστης μία μέρα πριν τη λήξη της προθεσμίας λαμβάνει ένα ειδοποιητικό e-mail.

**Εμφάνιση βαθμολογίας χρήστη:** Η βαθμολογία του χρήστη (prestige points) εξαρτάται από τον εκπαιδευτή. Ο χρήστης βαθμολογείται από το 1–10 και η βαθμολογία του εξαρτάται από το τι θέλει να δείξει ο εκπαιδευτής. Αν ο εκπαιδευτής θέλει να επισημάνει τη σημασία του testing, μπορεί ένας χρήστης ο οποίος τελείωσε το project

νωρίς, να πάρει πολύ χαμηλή βαθμολογία επειδή δεν έδωσε την απαραίτητη σημασία στο testing. Η βαθμολογία αυτή προστίθεται στα prestige points της εταιρείας.

Ο χρήστης παίζει σε εικονικό χρόνο, δηλαδή κάθε σενάριο ξεκινάει από μια συγκεκριμένη ημερομηνία και κάθε λεπτό στο ProMaSi αντιστοιχεί σε ένα δευτερόλεπτο του πραγματικού χρόνου. Ο χρήστης μπορεί να κάνει το χρόνο να πάει πιο γρήγορα ή πιο αργά αλλά δεν μπορεί να τον σταματήσει τελείως. Τα βήματα του χρόνου που μπορεί να επιλέξει ο χρήστης είναι:

1. **Slow:** Το κάθε λεπτό στο ProMaSi αντιστοιχεί σε 3 δευτερόλεπτα.
2. **Normal:** Το κάθε λεπτό αντιστοιχεί σε 1 δευτερόλεπτο.
3. **Fast:** Το κάθε λεπτό αντιστοιχεί σε 100ms.

Συνήθως κατά την προετοιμασία, ο χρήστης έχει το χρόνο στο Slow προκειμένου να προγραμματίσει τις εργασίες, να προσλάβει υπαλλήλους κ.τ.λ. Κατά την παρακολούθηση της προόδου των εργασιών, ο χρήστης έχει το χρόνο στο Fast και μόλις γίνει κάποιο γεγονός επαναφέρει το χρόνο στο Normal ή Slow.

Η διεπαφή του χρήστη μοιάζει με την επιφάνεια εργασίας ενός λειτουργικού συστήματος (OS) η οποία παρέχει στον χρήστη τα κατάλληλα εργαλεία για τον χρονοπρογραμματισμό του έργου και για όλες τις υπόλοιπες εργασίες που πρέπει να κάνει. Όλες οι προηγούμενες ενέργειες που αναφέρθηκαν γίνονται μέσω αυτών των προγραμμάτων. Τα προγράμματα που παρέχονται στο χρήστη είναι τα εξής:

1. **EvolutionBird:** Χρησιμοποιείται για την ανάγνωση των e-mails. Όλα τα e-mails υποστηρίζουν HTML. Το πρόγραμμα αυτό ειδοποιεί το χρήστη όταν έρθει κάποιο καινούργιο e-mail.
2. **Infogate:** Με αυτό το πρόγραμμα ο χρήστης βλέπει πληροφορίες:

   Για την εταιρεία στην οποία δουλεύει (περιγραφή εταιρείας, αριθμός υπαλλήλων που απασχολούνται από την εταιρεία).

   Στοιχεία του project που του έχει ανατεθεί.

   Περιγραφή όλων των υπαλλήλων που απασχολεί η εταιρεία (υπάλληλοι που έχουν προσληφθεί από το χρήστη).
3. **Marketplace:** Με αυτό το πρόγραμμα ο χρήστης βλέπει τους διαθέσιμους υπάλληλους που υπάρχουν στην αγορά καθώς και μία περιγραφή για τον κάθε υπάλληλο. Στην περιγραφή μπορεί ο εκπαιδευτής να γράψει διάφορες πληροφορίες έτσι ώστε να δώσει κάποιο στοιχείο για το οποίο θα φανεί χρήσιμος ο συγκεκριμένος υπάλληλος

στον χρήστη. Η περιγραφή του υπαλλήλου υποστηρίζει HTML format έτσι ώστε να μπορεί ο εκπαιδευτής να βάλει μια εικόνα του υπαλλήλου κ.τ.λ. Ο χρήστης μέσω του marketplace μπορεί να προσλάβει τους υπαλλήλους που επιθυμεί.

4. **Planner:** Με το Planner ο χρήστης προγραμματίζει τις εργασίες του project. Το Planner παρουσιάζει τις προγραμματισμένες εργασίες σαν ένα Gantt διάγραμμα. Το πρόγραμμα αυτό βοηθάει το χρήστη να προγραμματίσει μία εργασία να αρχίσει σε μία συγκεκριμένη ημερομηνία μέσα στα πλαίσια της ημερομηνίας έναρξης και τέλους του project. Μπορεί να αναθέσει έναν ή περισσότερους υπαλλήλους να ασχοληθούν με την συγκεκριμένη εργασία και να διευκρινίσει τις ώρες που θα ασχοληθούν με αυτή.

Η χρήση των προγραμμάτων αυτών καθώς και η ανάλυση της διεπαφής χρήστη γίνονται στο κεφάλαιο 6. Όταν συμβαίνει ένα τυχαίο γεγονός ή όταν τελειώνει μια εργασία ο χρήστης παίρνει την απόφασή του κάνοντας αλλαγές με το Planner (αναπρογραμματισμός εργασιών) ή με πρόσληψη-απόλυση υπαλλήλων.

## 3.3 Play mode

Όπως αναφέρθηκε και στην εισαγωγή το play mode ορίζει διάφορες παραμέτρους για τον τρόπο λειτουργίας του ProMaSi . Η βασική ροή του παιχνιδιού περιγράφηκε παραπάνω και δεν αλλάζει, αλλά επιπλέον ένα play mode ορίζει τα εξής:

1. Τι γίνεται όταν ξεκινάει το παιχνίδι.
2. Πως γίνονται διαθέσιμοι οι υπάλληλοι στην αγορά.
3. Τι γίνεται όταν ανατίθεται ένα project στον χρήστη.
4. Τι γίνεται όταν ξεκινάει το project.
5. Τι γίνεται όταν τελειώνει το project.
6. Τι γίνεται όταν προσλαμβάνεται ένας υπάλληλος.
7. Τι γίνεται σε κάθε τικ του ρολογιού.
8. Τη δομή των αρχείων.

Το κάθε play mode έχει ένα όνομα και μια περιγραφή. Όταν ο χρήστης ξεκινάει το παιχνίδι το πρώτο πράγμα που κάνει είναι να επιλέξει το play mode θέλει να παίξει. Έτσι ξεκινάει το παιχνίδι. Στα πλαίσια αυτής της πτυχιακής αναπτύχθηκε το single player score mode. Το επόμενο play mode που θα αναπτυχθεί είναι το multi player score mode.

### 3.3.1 Single player score mode

Το single player score mode είναι ένα play mode στο οποίο συμμετέχει ένας χρήστης και στόχος του είναι να πάρει τη μεγαλύτερη τοπική βαθμολογία, παίζοντας μία ιστορία. Η κάθε ιστορία ορίζει έναν αριθμό από διαδοχικά project. Η τελική βαθμολογία βγαίνει από τα συνολικά prestige points που έλαβε ο χρήστης στο κάθε project. Στο single player score mode ορίζονται τα εξής:

1. Όταν ξεκινάει το παιχνίδι, ο χρήστης αφού επιλέξει αυτό το play mode, επιλέγει την ιστορία την οποία θέλει να παίξει. Στη συνέχεια δίνει τα στοιχεία του(όνομα,επώνυμο) και έτσι ξεκινάει το παιχνίδι.

2. Οι διαθέσιμοι υπάλληλοι ορίζονται από τον εκπαιδευτή σε ένα xml αρχείο και είναι διαθέσιμοι σε όλη τη διάρκεια του παιχνιδιού.

3. Μόλις ανατίθεται ένα project στον χρήστη, του στέλνεται ένα e-mail με τα χαρακτηριστικά του project.

4. Μόλις ξεκινάει το project το single play mode διαβάζει τις ρυθμίσεις του και ξεκινάει το αντίστοιχο μοντέλο.

5. Όταν τελειώσει το project, εμφανίζεται στον χρήστη μία φόρμα με τα στατιστικά του και γίνεται ανάθεση του επόμενου project της ιστορίας στο χρήστη. Αν η ιστορία δεν έχει άλλο project, ο χρήστης βλέπει τα στατιστικά του και την τελική του βαθμολογία σε σχέση με βαθμολογίες άλλων χρηστών.

6. Κατά την πρόσληψη ενός υπαλλήλου δεν γίνεται τίποτα συγκεκριμένο.

7. Σε κάθε τικ του ρολογιού εκτελείται και ένα βήμα στο μοντέλο του project. Αν έχει περάσει το ωράριο λειτουργίας της εταιρείας γίνεται αλλαγή ημέρας στην και ξεκινάει το ωράριο. Αν έχει περάσει η μέρα πληρώνονται οι υπάλληλοι, τα χρήματα αφαιρούνται από τον προϋπολογισμό του project και στέλνεται ένα e-mail στον χρήστη με το τελικό πόσο.

8. Όλα τα δεδομένα του single player score mode κρατιούνται σε xml αρχεία. Η δομή των δεδομένων θα αναλυθεί στο κεφάλαιο 5.

### 3.3.2 Multi player score mode

Το multi player score mode είναι ένα play mode στο οποίο συμμετέχουν πολλοί παίκτες. Όλοι παίζουν το ίδιο project και μοιράζονται τους διαθέσιμους υπαλλήλους. Σκοπός του κάθε παίκτη είναι να κάνει το υψηλότερο score από τους υπόλοιπους. Σε αυτό το play mode αυτό ορίζονται τα εξής:

1. Όταν ξεκινάει το παιχνίδι, ο χρήστης αφού επιλέξει αυτό το play mode, επιλέγει τον server στον οποίο θέλει να παίξει. Στη συνέχεια δίνει τα στοιχεία του(όνομα,επώνυμο) και περιμένει μέχρι να συμπληρωθεί ο επιθυμητός αριθμός χρηστών για να ξεκινήσει το παιχνίδι.

2. Οι διαθέσιμοι υπάλληλοι είναι ίδιοι για όλους τους παίκτες. Μόλις ένας παίκτης προσλάβει κάποιον υπάλληλο, οι υπόλοιποι παίκτες δεν έχουν το δικαίωμα να τον προσλάβουν.

3. Το project ανατίθεται στον χρήστη αφού συγκεντρωθούν όλοι οι χρήστες και του στέλνεται ένα e-mail με τα χαρακτηριστικά του.

4. Μόλις ξεκινάει το project ξεκινάει και το αντίστοιχο μοντέλο.

5. Όταν τελειώσει το project, εμφανίζεται στον χρήστη μία φόρμα με τα στατιστικά του και περιμένει μέχρι να τελειώσουν και οι υπόλοιποι παίκτες για να δει τις βαθμολογίες.

6. Κατά την πρόσληψη ενός υπαλλήλου στέλνεται μια ειδοποίηση στους υπόλοιπους παίκτες ότι ο υπάλληλος αυτός δεν είναι πλέον διαθέσιμος.

7. Σε κάθε τικ του ρολογιού εκτελείται και ένα βήμα στο μοντέλο του project. Αν έχει περάσει το ωράριο λειτουργίας της εταιρείας γίνεται αλλαγή ημέρας στην και ξεκινάει το ωράριο. Αν έχει περάσει η μέρα πληρώνονται οι υπάλληλοι, τα χρήματα αφαιρούνται από τον προϋπολογισμό του project και στέλνεται ένα e-mail στον χρήστη με το τελικό πόσο.

8. Όλα τα δεδομένα του multi player score mode θα κρατιούνται σε βάση δεδομένων.

Το play mode αυτό δεν έχει αναπτυχθεί ακόμα, απλά αναφέρεται για να δείξουμε πως μπορούν να οριστούν διάφορα play modes.

## 3.4 Ο Εκπαιδευτής(Game Master)

Ο ρόλος του εκπαιδευτή είναι να δημιουργεί σενάρια για το play mode που επιθυμεί. Το κάθε play mode έχει διαφορετικές απαιτήσεις για αυτό και ένα σενάριο του single player score mode δεν θα είναι συμβατό με το multi player score mode.

Κάθε σενάριο είναι ουσιαστικά ένα μοντέλο system dynamics στο οποίο όμως έχουν προστεθεί χαρακτηριστικά απαραίτητα για την εκτέλεσή του σε διαδραστικό περιβάλλον όπως περιγράφτηκε στο προηγούμενο κεφάλαιο.

Η δημιουργία των σεναρίων αποτελεί αυτή τη στιγμή το πιο κρίσιμο και δύσκολο κομμάτι του ProMaSi . Ο εκπαιδευτής πρέπει να αποφασίσει καταρχήν τον εκπαιδευτικό σκοπό του

μοντέλου που θα δημιουργήσει, για παράδειγμα το γεγονός ότι καθώς αυξάνεται ο αριθμός των ατόμων σε ένα έργο, τότε αυξάνεται και ο χρόνος που δαπανάται για συναντήσεις και επικοινωνία των μελών. Στη συνέχεια θα πρέπει να δημιουργήσει ένα μοντέλο στο οποίο λαμβάνονται υπόψη οι ανάγκες επικοινωνίας του προσωπικού κατά την εκτέλεση του έργου και ακολούθως θα πρέπει να ελέγξει την ορθότητα του μοντέλου εξετάζοντας την συμπεριφορά του με διαφορετικές τιμές. Τέλος θα πρέπει να δημιουργήσει σενάρια βασισμένα σε αυτό το μοντέλο τα οποία περιγράφουν συγκεκριμένα έργα.

Επειδή η δημιουργία αυτών των σεναρίων είναι δύσκολη καλό είναι ο εκπαιδευτής έχει στη διάθεση του εργαλεία τα οποία θα τον βοηθήσουν να τα δημιουργήσει. Αυτή τη στιγμή το ProMaSi παρέχει τον core designer, ένα εργαλείο που βοηθάει τον εκπαιδευτή να δημιουργεί μοντέλα system dynamics σε ένα γραφικό περιβάλλον. Τα υπόλοιπα αρχεία που απαιτούνται για ένα σενάριο φτιάχνονται κατευθείαν από τον εκπαιδευτή. Αναλυτική περιγραφή της διαδικασίας δημιουργίας σεναρίων γίνεται στο κεφάλαιο 5.

# 4 Αρχιτεκτονική ProMaSi

Στο προηγούμενο κεφάλαιο, είδαμε πώς λειτουργεί το ProMaSi από τη μεριά του χρήστη. Σε αυτό το κεφάλαιο θα αναλύσουμε πως λειτουργεί το εσωτερικό του ProMaSi .

Το ProMaSi αποτελείται από 5 επίπεδα(layers), το κάθε layer έχει την δική του ευθύνη.

1. **Core:** Είναι υπεύθυνο για την εξομοίωση των system dynamics. Σε αυτό το επίπεδο βρίσκεται η υλοποίηση για το PSD. Η υλοποίηση του Core δεν είναι συγκεκριμένη για το ProMaSi , αλλά μπορεί να χρησιμοποιηθεί και σε άλλα project καθώς η υλοποίηση του δεν περιορίζεται στο project management.

2. **Communication:** Το Communication παρέχει τα κατάλληλα interfaces, έτσι ώστε να μπορεί να επικοινωνεί ένα εξωτερικό επίπεδο με το Core. Το Core μαζί με το Communication μπορούν να χρησιμοποιηθούν για να υλοποιήσουν τη βάση σε οποιοδήποτε παιχνίδι, που σκοπός του είναι η εξομοίωση. Το Communication μπορεί εκτός από το Shell να μεταφέρει και τις τιμές(εισόδους-εξόδους) μέσω δικτύου, έτσι ώστε για παράδειγμα ο εκπαιδευτής να μπορεί να βλέπει τι κάνουν όλοι οι εκπαιδευόμενοι.

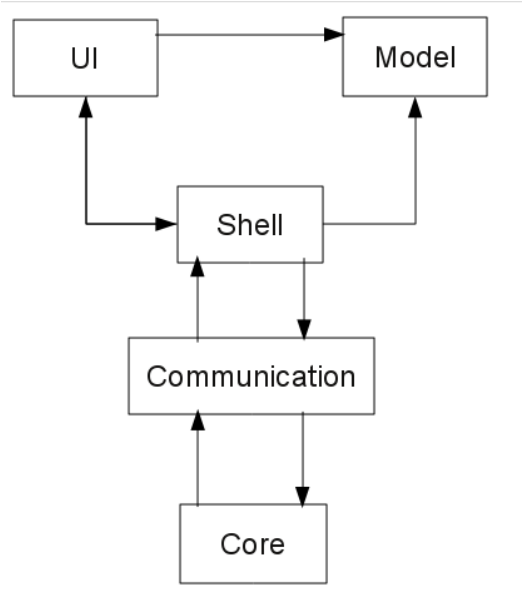
3. **Model:** Εδώ υλοποιούνται όλες οι οντότητες που έχουν σχέση με το project management. Η έννοια του χρόνου ορίζεται επίσης στο Model.

4. **Shell:** Αυτό το επίπεδο επικοινωνεί με το Communication. Από αυτό το επίπεδο παίρνει είσοδο το Core και σε αυτό δίνει την έξοδό του. Ακόμη, παρέχει τα κατάλληλα interfaces για τη δημιουργία ενός καινούργιου play mode - εδώ είναι υλοποιημένο το single player score mode. Επίσης, παρέχει τα κατάλληλα interfaces για τη δημιουργία ενός καινούργιου UI. Το play mode δεν σχετίζεται με κάποιο συγκεκριμένο UI. Έτσι μπορούμε να φτιάξουμε ένα UI παρόμοιο με αυτό του SimSE και όταν ξεκινάει το ProMaSi ο χρήστης να έχει τη δυνατότητα να χρησιμοποιήσει το UI της επιλογής του.

5. **UI:** Το User Interface του ProMaSi . Εδώ είναι υλοποιημένο το desktop UI, το οποίο έχει τη μορφή ενός λειτουργικού συστήματος μαζί με τις εφαρμογές του(mail client, gantt editor κ.τ.λ). Το UI είναι υπεύθυνο για την διαχείριση των δεδομένων του Model.

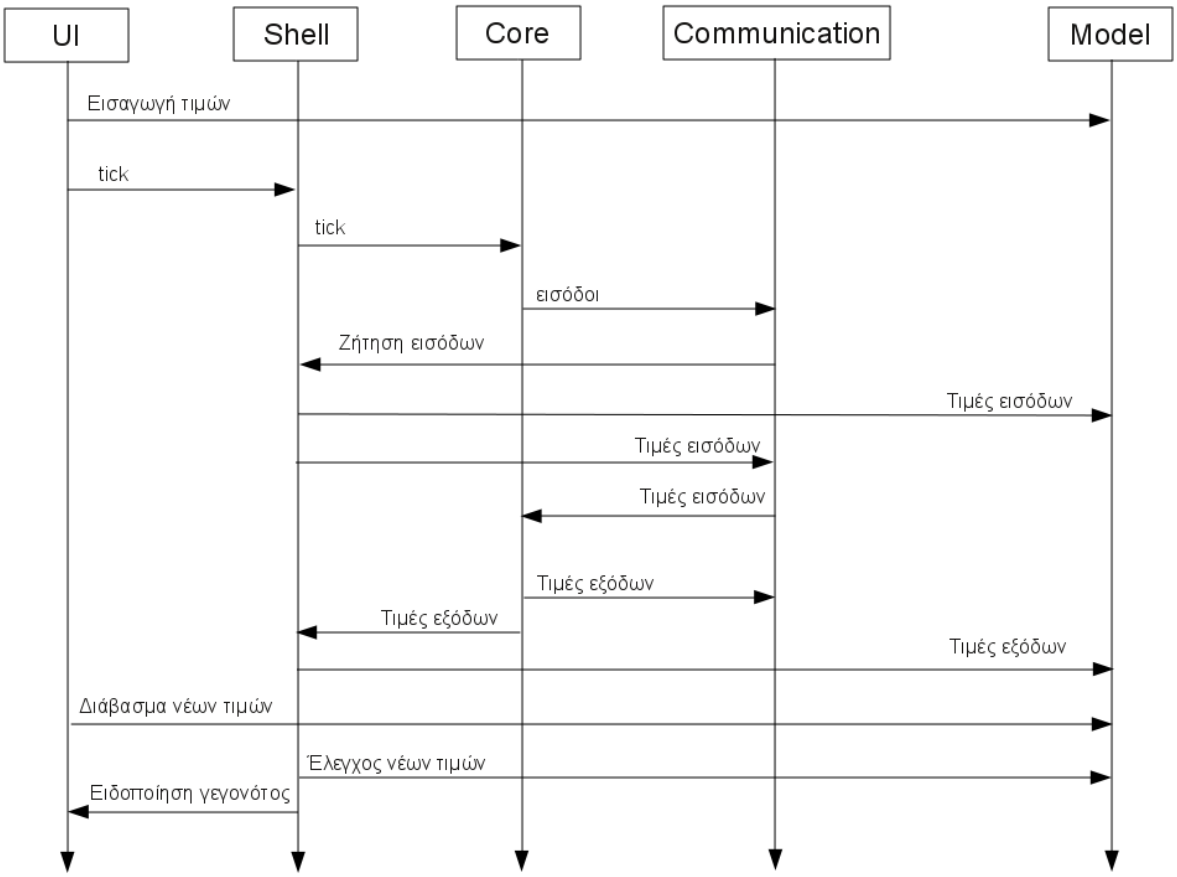
### Επικοινωνία μεταξύ επιπέδων

Στο Σχήμα 5, φαίνεται ο τρόπος με τον οποίο επικοινωνούν τα 5 επίπεδα μεταξύ τους. Ο χρήστης μέσω του UI δίνει τιμές στο Model και βλέπει τιμές από το Model. Για κάθε tick του χρόνου εκτελείται και ένα βήμα του Core(PSD). Το Core μόλις εκτελεί το βήμα, ζητάει τις εισόδους του από το Communication και το Communication με τη σειρά του ειδοποιεί το Shell για το ποιες εισόδους θέλει. Το Shell παίρνει τις τιμές που θέλει από το Model και τις δίνει πίσω. Μόλις τελειώσει το βήμα του Core, στέλνει τις εξόδους του στο Communication, το Communication στέλνει τις εξόδους στο Shell και τις βάζει στο Model. Το UI ενημερώνεται με τις καινούργιες τιμές και το Shell ανάλογα με τις καινούργιες τιμές του Model, μπορεί να ενημερώσει το UI για κάποιο γεγονός(τελείωσε το έργο κ.τ.λ).




Σχήμα 5: Επικοινωνία μεταξύ επιπέδων

Με αυτήν την αρχιτεκτονική μπορούμε να δημιουργήσουμε, για παράδειγμα, ένα UI το οποίο να μοιάζει με τέστ. Ο εκπαιδευτής θα δημιουργεί το κατάλληλο PSD μοντέλο και το UI θα δίνει ερωτήσεις πολλαπλής επιλογής στον χρήστη. Η απάντησή του θα δίνεται σαν είσοδο στο Core, το Core θα δίνει την έξοδο και μόλις τελειώσουν όλες οι ερωτήσεις το UI θα εμφανίζει το τελικό score.




Σχήμα 6: Διάγραμμα ακολουθίας μεταξύ επιπέδων

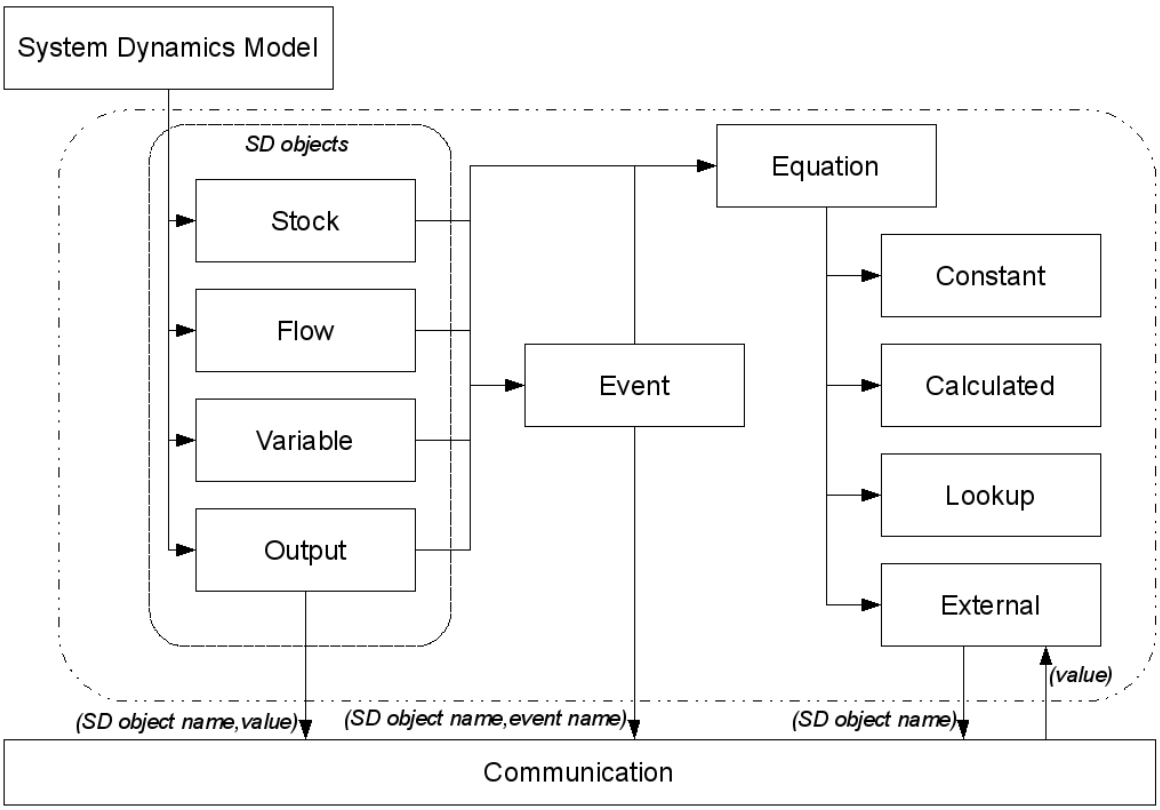
## 4.1 Core(PSD)

Το Core είναι το πιο βασικό επίπεδο του ProMaSi , διότι εδώ βρίσκεται όλος ο μηχανισμός που γίνεται η εξομοίωση. Όπως αναφέραμε και στο Κεφάλαιο 2, το Core δεν υλοποιεί ακριβώς τα system dynamics αλλά μια μορφή που να ταιριάζει στις ανάγκες του ProMaSi , την οποία ονομάσαμε PSD(Promasi System Dynamics). Βασικός στόχος είναι να μπορεί ένα μοντέλο των system dynamics να μεταφραστεί σε PSD. Τα PSD έχουν τα εξής επιπλέον χαρακτηριστικά σε σχέση με τα system dynamics:

1. Παρέχουν και άλλες οντότητες εκτός από τα stock, flow, variable για την αναπαράσταση των εισόδων και εξόδων του μοντέλου.

2. Κάθε οντότητα μπορεί να έχει και διάφορα γεγονότα, τα οποία ενεργοποιούνται όταν ισχύουν συγκεκριμένες συνθήκες.

Για την μετάφραση ενός μοντέλου system dynamics σε PSD πρέπει να αναγνωριστούν οι είσοδοι-έξοδοι του μοντέλου και να προστεθούν κάποια επιπλέον γεγονότα.




Σχήμα 7: Αρχιτεκτονική PSD

Στο Σχήμα 7, φαίνεται η αρχιτεκτονική των PSD. Το μοντέλο αποτελείται από πολλά SD objects(system dynamics objects). Ένα SD object έχει ένα μοναδικό όνομα μέσα στο μοντέλο, προκειμένου να μπορούμε να το αναγνωρίσουμε και μπορεί να έχει έναν από τους ακόλουθους τύπους:

**Stock, Flow, Variable:** Οι τύποι αυτοί είναι ίδιοι με τα system dynamics.

**Output:** Ο τύπος αυτός ορίζει μια έξοδο του συστήματος. Σε κάθε βήμα, όλα τα outputs του μοντέλου ειδοποιούν το Communication επίπεδο με την τιμή τους.

Όλοι οι παραπάνω τύποι έχουν και μία λίστα με τα SD objects, από τα οποία εξαρτάται ο υπολογισμός της τιμής τους. Εκτός από τους παραπάνω τύπους υπάρχουν και οι παράμετροι του συστήματος. Για παράδειγμα η τιμή της παραμέτρου time μας δίνει τη χρονική στιγμή για την οποία επιθυμούμε τον υπολογισμό των τιμών των SD objects του μοντέλου.

Όπως και στα system dynamics από όλα τα SD Objects, τα πιο σημαντικά, είναι τα Stocks. Ο ρόλος των Flows και των Variables είναι και αυτό ο ίδιος όπως στα system dynamics.

Για παράδειγμα οι Variables χρησιμοποιούνται όμως για να αποθηκεύουν ενδιάμεσες τιμές που μπορεί να είναι χρήσιμες στους υπολογισμούς και τα Flows για να ορίσουν το ρυθμό μεταβολής ενός Stock.

Έχοντας υπόψη τα παραπάνω μπορούμε να πούμε ότι ένα μοντέλο στο ProMaSi δεν είναι τίποτα άλλο παρά ένα σύνολο από συζευγμένες μη-γραμμικές διαφορικές εξισώσεις πρώτου βαθμού της μορφής $x'(t) = f(x, p)$ όπου $t$ είναι ο χρόνος, $x'$ το διάνυσμα με όλα τα Stocks, $p$ το σύνολο των παραμέτρων του συστήματος και $f$ μια μη-γραμμική συνάρτηση διανυσμάτων.

Κατά τη διάρκεια της εκτέλεσης του μοντέλου υπολογίζονται τα Flows και ολοκληρώνονται προκειμένου να υπολογιστούν οι τιμές των Stocks. Η ολοκλήρωση γίνεται με μεθόδους αριθμητικής ανάλυσης. Για κάθε Stock υπολογίζεται η τιμή του με βάση τον παρακάτω τύπο: $Stock = Stock_0 + \int_0^t g(Flows, Variables)dt$

Στην πιο απλή του μορφή ο τύπος είναι ο παρακάτω: $Stock = Stock_0 + \int_0^t (inflow - outflow)dt$

Η ολοκλήρωση στο ProMaSi γίνεται με τη μέθοδο του Euler αλλά μπορεί εύκολα να εφαρμοστεί και η μέθοδος Runge-Kutta.

Επιπλέον, για ευκολία στη δόμηση των μοντέλων παρέχονται επίσης και οι ακόλουθοι τύποι συναρτήσεων οι οποίοι μπορούν να χρησιμοποιηθούν κατά τον ορισμό των τιμών ενός SD Object:

**Constant:** Σταθερά.

**Calculated:** Μπορούμε να κάνουμε αριθμητικές πράξεις μεταξύ των SD object του μοντέλου και να καλέσουμε διάφορες συναρτήσεις. Παράδειγμα ((var1+var2)/2)-sin(var3). Οι συναρτήσεις που έχουμε στη διάθεσή μας είναι:

1. **sin(x)**
2. **atanh(x)**
3. **Max(x,y)**
4. **ceil(x)**
5. **cosh(x)**
6. **cos(x)**
7. **sum(x,y,...)**
8. **atan2(x)**
9. **tanh(x)**
10. **pow(x,y)**
11. **asinh(x)**
12. **atan(x)**
13. **log(x)**
14. **exp(x)**
15. **binom(x,y)**
16. **ln(x)**
17. **sinh(x)**
18. **asin(x)**
19. **acos(x)**
20. **if(cond,true,false)**
21. **abs(x)**
22. **str(x)**
23. **floor(x)**
24. **tan(x)**
25. **round(x)**
26. **Zidz(x,y)**
27. **acosh(x)**
28. **rand()**
29. **mod(x,y)**
30. **sqrt(x)**
31. **Min(x,y)**

**Lookup:** Συνάρτηση στην οποία δίνουμε σημεία προκειμένου να δημιουργήσουμε μια γραφική παράσταση. Για παράδειγμα θα μπορούσαμε να δώσουμε τα σημεία (1,1)(2,2)(3,3)(4,4)(5,5) και ένα SD object από το οποίο θα εξαρτάται η τιμή της συνάρτησης π.χ. var1(x). Ανάλογα με την τιμή του var1 και τα στοιχεία που δώσαμε θα πάρουμε και την αντίστοιχη έξοδο(y). Στο παραπάνω παράδειγμα, αν η var1 έχει την τιμή 2.5 η συνάρτηση θα επιστρέψει 2.5. Αν η var1 βρίσκεται εκτός ορίων(<1 ή >5) τότε η συνάρτηση παίρνει την τιμή 0.

**External:** Η συνάρτηση αυτή αντιπροσωπεύει μια είσοδο του συστήματος. Κάθε φορά που θέλουμε να υπολογίσουμε την τιμή της, στέλνει ένα μήνυμα στο Communication με το όνομα της μεταβλητής. Το Communication ειδοποιεί το Shell ότι το συγκεκριμένο SD object θέλει τιμή και το Shell επιστρέφει μια τιμή.

Κάθε SD object έχει κάποια γεγονότα, προκειμένου να ειδοποιήσει το Shell ότι κάτι σημαντικό έχει γίνει. Το κάθε γεγονός έχει ένα όνομα και μία συνάρτηση. Αν η συνάρτηση πάρει τιμή αριθμό >=1, τότε θεωρούμε ότι το γεγονός έχει ενεργοποιηθεί και στέλνει ένα μήνυμα στο Communication με το όνομα του SD object και το όνομα του event. Ένα γεγονός μπορεί να ενεργοποιηθεί μία μόνο φορά.

Εκτός από τα γεγονότα, ένα SD object μπορεί να έχει ένα info και ένα hint. Όσα SD object έχουν ορισμένα αυτά τα δύο πεδία, φαίνονται στα στατιστικά του project. Το info δίνει μια περιγραφή για το SD object και το hint δίνει πληροφορίες για το πως μπορεί ο χρήστης να βελτιώσει τις τιμές αυτού. Έτσι, όταν τελειώσει το project ο χρήστης μπορεί να δει μια γραφική παράσταση με το χρόνο και τις τιμές του SD object για να μπορέσει να δει τα λάθη του.

### 4.1.1 Υπολογισμός τιμών

Μόλις εκτελείται ένα βήμα στο μοντέλο, το Core υπολογίζει τις τιμές όλων των SD objects:

1. Υπολογισμός των variable.
2. Υπολογισμός των flow.
3. Υπολογισμός των output.
4. Υπολογισμός των stock για το επόμενο βήμα.

Για τον υπολογισμό οποιουδήποτε SD object υπολογίζονται:

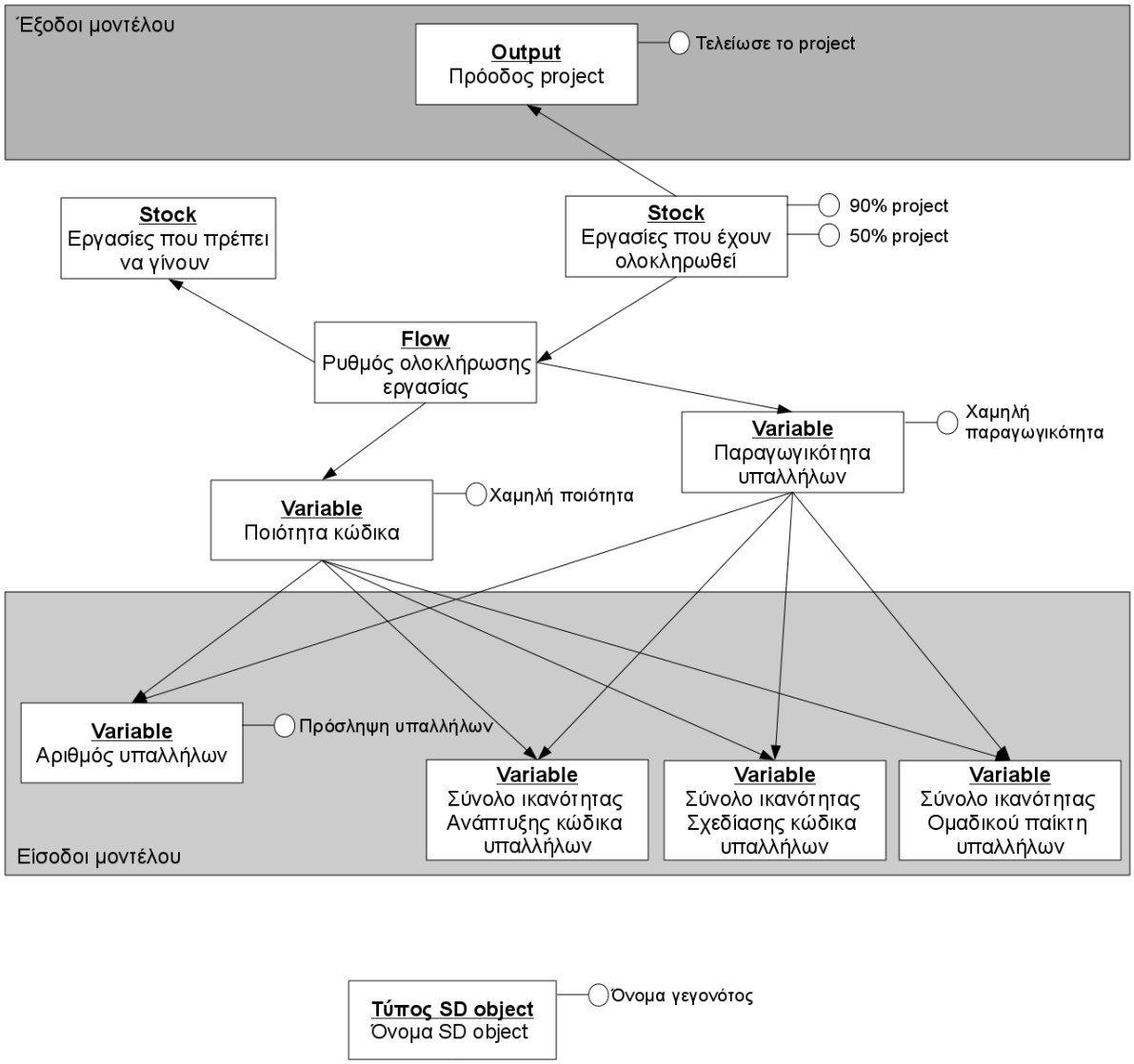
1. Όλα τα SD object από τα οποία εξαρτάται.

2. Η τιμή της συνάρτησής του(constant, calculated, lookup, external). Στην περίπτωση που το SD object έχει external συνάρτηση, στέλνεται ένα μήνυμα στο Communication επίπεδο. Το Communication στέλνει το ερώτημα στο Model επίπεδο, το οποίο επιστρέφει την επιθυμητή τιμή και το Communication την επιστρέφει με τη σειρά του στο Core.

3. Η τιμή της συνάρτησης του κάθε γεγονότος του SD object. Αν η τιμή αυτή είναι >1, στέλνεται ένα μήνυμα στο Communication επίπεδο και το γεγονός απενεργοποιείται.

Σε κάθε βήμα του μοντέλου γνωρίζουμε τις τιμές των stock SD objects, γιατί η τιμή τους υπολογίζεται ένα βήμα πριν, και τις τιμές όλων των SD objects τα οποία έχουν constant και external συναρτήσεις. Αν κάποιο γεγονός χρειάζεται άλλα SD objects του μοντέλου, τότε τα SD objects πρέπει να μπουν σαν εξάρτηση στο SD object που βρίσκεται το γεγονός. Τα SD object που έχουν συναρτήσεις constant και external δεν έχουν εξαρτήσεις, έκτος αν το απαιτεί κάποιο γεγονός - ειδικά στα SD object με constant συνάρτηση δεν υπάρχει κανένας λόγος να μπει κάποιο γεγονός.

### 4.1.2 Παράδειγμα PSD μοντέλου




## Επεξήγηση μεταβλητών μοντέλου

- **Πρόοδος project:** Δείχνει πόσο τοις εκατό έχει ολοκληρωθεί το project. Η μεταβλητή αυτή έχει ένα γεγονός 'Τελείωσε το project', το οποίο ενεργοποιείται όταν η 'Πρόοδος project' έχει την τιμή 1. Ειδοποιεί το Model επίπεδο ότι τελείωσε το project.
- **Εργασίες που πρέπει να γίνουν:** Δείχνει τον αριθμό των εργασιών του project που έχει απομείνει για να ολοκληρωθεί.

- **Εργασίες που έχουν ολοκληρωθεί:** Ο αριθμός των εργασιών που έχει ολοκληρωθεί. Η μεταβλητή αυτή έχει 2 γεγονότα ΄50% project΄ και ΄90% project΄, τα οποία ενεργοποιούνται όταν το ποσοστό ολοκλήρωσης του project είναι μεγαλύτερο ή ίσο του 0.5 και 0.9 αντίστοιχα. Ειδοποιεί το Shell επίπεδο ότι το project έχει ολοκληρωθεί κατά 50% ή 90% αντίστοιχα.
- **Ρυθμός ολοκλήρωσης εργασίας:** Ο ρυθμός με τον οποίο ολοκληρώνεται μια εργασία σε κάθε βήμα(λεπτό).
- **Παραγωγικότητα υπαλλήλων:** Το έργο που παράγουν όλοι οι υπάλληλοι κάθε μέρα. Η μεταβλητή αυτή έχει ένα γεγονός ΄Χαμηλή παραγωγικότητα΄, το οποίο ενεργοποιείται όταν η ΄Παραγωγικότητα υπαλλήλων΄ έχει τιμή μικρότερη του 0.3. Ειδοποιεί το Shell επίπεδο ότι η παραγωγικότητα των υπαλλήλων είναι πολύ χαμηλή.
- **Ποιότητα κώδικα:** Η ποιότητα του κώδικα που παράγουν οι υπάλληλοι. Η μεταβλητή αυτή έχει ένα γεγονός ΄Χαμηλή ποιότητα΄, το οποίο ενεργοποιείται όταν η ΄Ποιότητα κώδικα΄ έχει τιμή μικρότερη του 0.3. Ειδοποιεί το Shell επίπεδο ότι η ποιότητα του κώδικα είναι πολύ χαμηλή.
- **Αριθμός υπαλλήλων:** Ο αριθμός των προσληφθέντων υπαλλήλων. Η μεταβλητή αυτή έχει ένα γεγονός ΄Πρόσληψη υπαλλήλων΄, το οποίο ενεργοποιείται όταν ο ΄Αριθμός υπαλλήλων΄ έχει την τιμή 0. Ειδοποιεί το Shell επίπεδο ότι δεν υπάρχουν υπάλληλοι.
- **Σύνολο ικανότητας Ανάπτυξης κώδικα υπαλλήλων:** Το σύνολο της ικανότητας ΄Ανάπτυξη κώδικα΄ όλων των υπαλλήλων.
- **Σύνολο ικανότητας Σχεδίασης κώδικα υπαλλήλων:** Το σύνολο της ικανότητας ΄Σχεδίαση κώδικα΄ όλων των υπαλλήλων.
- **Σύνολο ικανότητας Ομαδικού παίκτη υπαλλήλων:** Το σύνολο της ικανότητας ΄Ομαδικός παίκτης΄ όλων των υπαλλήλων.

## Τιμές εισόδου

Ο χρήστης έχει προσλάβει 4 υπαλλήλους. Οι υπάλληλοι έχουν τις εξής ικανότητες:

- Υπάλληλος 1(Ανάπτυξη κώδικα=4,Σχεδίαση κώδικα=2,Ομαδικός παίκτης=1)
- Υπάλληλος 2(Ανάπτυξη κώδικα=6,Σχεδίαση κώδικα=1,Ομαδικός παίκτης=5)
- Υπάλληλος 3(Ανάπτυξη κώδικα=8,Σχεδίαση κώδικα=6,Ομαδικός παίκτης=7)
- Υπάλληλος 4(Ανάπτυξη κώδικα=5,Σχεδίαση κώδικα=7,Ομαδικός παίκτης=2)

Οι τιμές αυτές αποθηκεύονται στο Model επίπεδο. Το Core επίπεδο δεν γνωρίζει αυτές τις τιμές και πρέπει κάθε φορά να τις ζητάει από το Model επίπεδο χρησιμοποιώντας την external συνάρτηση. Έτσι, αν ο χρήστης στη μέση του παιχνιδιού απολύσει κάποιον υπάλληλο, η κάθε external συνάρτηση του μοντέλου στο επόμενο βήμα θα ρωτήσει το Model επίπεδο και θα πάρει τις καινούργιες τιμές.

## Υπολογισμός $1^{ου}$ βήματος

Ο ακριβής υπολογισμός δεν θα μας απασχολήσει σε αυτό το παράδειγμα, γιατί έχει αναλυθεί στο Κεφάλαιο 2, για το λόγο αυτό και δεν δίνονται οι συναρτήσεις κάθε μεταβλητής. Θα μας απασχολήσουν τα επιπλέον χαρακτηριστικά του PSD και ο τρόπος επικοινωνίας του με τα άλλα επίπεδα.

Το Core πρώτα θα υπολογίσει τις τιμές από τα SD object τύπου variable και θα επιλέξει ένα στην τύχη, για παράδειγμα την Ποιότητα Κώδικα΄. Η Ποιότητα Κώδικα΄ εξαρτάται από τις μεταβλητές Αριθμός υπαλλήλων΄,Σύνολο ικανότητας Ανάπτυξης κώδικα υπαλλήλων΄,Σύνολο ικανότητας Σχεδίασης κώδικα υπαλλήλων΄, Σύνολο ικανότητας Ομαδικού παίκτη υπαλλήλων΄ οπότε πρώτα θα πρέπει να υπολογίσει τις τιμές αυτών των μεταβλητών. Όλες αυτές οι μεταβλητές έχουν σα συνάρτηση την external, οπότε το Core θα στείλει ένα μήνυμα στο Communication ότι θέλει να πάρει μια τιμή για την μεταβλητή Αριθμός υπαλλήλων΄. Το Communication θα στείλει το μήνυμα στο Shell και ο Αριθμός υπαλλήλων΄ θα πάρει την τιμή 4. Στη συνέχεια, αφού ο Αριθμός υπαλλήλων΄ έχει ένα γεγονός Πρόσληψη Υπαλλήλων΄, θα υπολογίσει την τιμή του γεγονότος. Αφού η τιμή της μεταβλητής είναι 4, το γεγονός δεν θα ενεργοποιηθεί. Με τον ίδιο τρόπο θα υπολογίσει και τις υπόλοιπες μεταβλητές από τις οποίες εξαρτάται η Ποιότητα Κώδικα΄. Έπειτα το Core θα υπολογίσει την τιμή της μεταβλητής Ποιότητα Κώδικα΄ ανάλογα με τον τύπο της συνάρτησης που έχει. Επίσης υπολογίσει και το γεγονός της μεταβλητής αυτής: αν η συνάρτηση του γεγονότος πάρει τιμή αριθμό $>= 1$, τότε το γεγονός θα ενεργοποιηθεί και το Core θα στείλει ένα μήνυμα στο Communication ότι έχει ενεργοποιηθεί το γεγονός Χαμηλή ποιότητα΄ από την μεταβλητή Ποιότητα Κώδικα΄.

Σε αυτό το σημείο έχουμε υπολογίσει τις εξής μεταβλητές: Αριθμός υπαλλήλων΄, Σύνολο ικανότητας Ανάπτυξης κώδικα υπαλλήλων΄, Σύνολο ικανότητας Σχεδίασης κώδικα υπαλλήλων΄, Σύνολο ικανότητας Ομαδικού παίκτη υπαλλήλων΄,Ποιότητα Κώδικα΄. Αν το Core επιλέξει κάποια μεταβλητή η οποία έχει ήδη υπολογιστεί για αυτό το βήμα την προσπερνάει. Η μόνη μεταβλητή τύπου variable που έμεινε είναι η Παραγωγικότητα υπαλλήλων΄, την οποία υπολογίζει με τον ίδιο τρόπο προσπερνώντας όλες τις εξαρτήσεις αφού έχουν ήδη υπολογιστεί.

Στη συνέχεια, υπολογίζει τις μεταβλητές με τύπο flow. Στο μοντέλο αυτό υπάρχουν

μόνο ο Ρυθμός ολοκλήρωσης εργασίας', ο οποίος εξαρτάται από την Ποιότητα κώδικα' και η Παραγωγικότητα υπαλλήλων' οι οποίες όμως έχουν ήδη υπολογιστεί. Αφού έχουμε υπολογίσει όλες τις εξαρτήσεις του, υπολογίζουμε και την τιμή του ανάλογα με τον τύπο συνάρτησής του.

Επόμενο βήμα του Core είναι να υπολογίσει τις τιμές των μεταβλητών που έχουν τύπο output. Πρώτα θα υπολογίσει τις εξαρτήσεις της μεταβλητής Πρόοδος project' οι οποίες είναι η Εργασίες που έχουν ολοκληρωθεί'. Επειδή η Εργασίες που έχουν ολοκληρωθεί' είναι τύπου stock, δεν υπολογίζεται παρά μόνο στο τέλος του βήματος για να πάρει τιμή για το επόμενο βήμα. Επομένως τα stock έχουν ήδη υπολογιστεί στο τέλος του προηγούμενου βήματος, οπότε το Core υπολογίζει την τιμή της Πρόοδος project' ανάλογα με την συνάρτηση που έχει. Έπειτα υπολογίζει το γεγονός Τελείωσε το project', το οποίο δεν ενεργοποιείται. Εφόσον η μεταβλητή αυτή είναι τύπου output το Core στέλνει την τιμή της και το όνομα της στο Communication.

Τέλος, υπολογίζει τις μεταβλητές τύπου stock για το επόμενο βήμα. Από τις μεταβλητές αυτές έχουν υπολογιστεί όλες οι εξαρτήσεις, οπότε το μόνο που έχει να κάνει είναι να υπολογίσει την τιμή της μεταβλητής και μετά τα γεγονότα της όπως με τις υπόλοιπες μεταβλητές.

## 4.2 Communication

Το Communication επίπεδο είναι υπεύθυνο για την επικοινωνία του Core με κάποιο εξωτερικό επίπεδο. Στο επίπεδο αυτό μεταφέρονται μηνύματα από το Core προς το εξωτερικό επίπεδο. Το εξωτερικό επίπεδο δεν μπορεί να επικοινωνήσει με το Core, απλά παίρνει μηνύματα από το Communication και απαντάει. Τα μηνύματα που στέλνει το Communication είναι τα εξής:

- **`Ζήτηση τιμής(όνομα μεταβλητής)`:** Το μήνυμα αυτό στέλνεται από το Communication όταν το Core θέλει να πάρει μια είσοδο. Ως παράμετρο παίρνει το όνομα της μεταβλητής του Core η οποία ζητάει την τιμή από το εξωτερικό επίπεδο και στέλνεται από το Core μέσω της external συνάρτησης. Το εξωτερικό επίπεδο πρέπει να επιστρέψει μια τιμή ανάλογα με το όνομα της μεταβλητής.

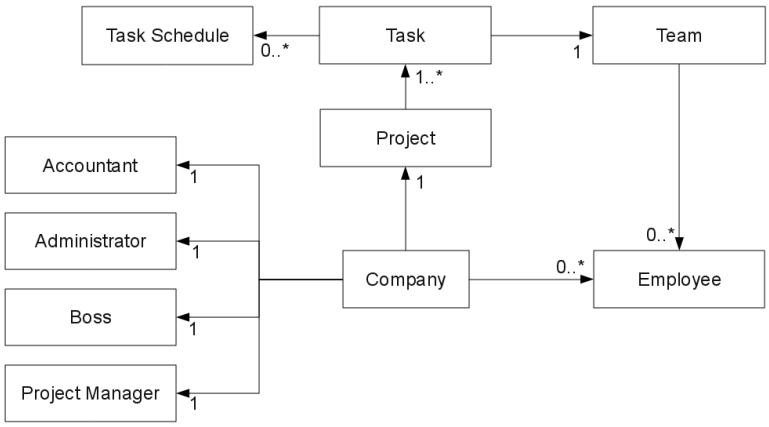
- **`Ενημέρωση τιμής(όνομα μεταβλητής, τιμή μεταβλητής)`:** Το μήνυμα στέλνεται από το Communication όταν το Core στέλνει μια έξοδο. Ως παράμετρο παίρνει το όνομα της μεταβλητής του Core και την τιμή της και στέλνεται από το Core μέσω μιας output μεταβλητής. Το εξωτερικό επίπεδο δεν πρέπει να επιστρέψει τίποτα στο Communication.

- **Γεγονός(όνομα μεταβλητής,όνομα γεγονότος):** Το μήνυμα στέλνεται από το Communication όταν ενεργοποιείται ένα γεγονός στο Core. Ως παράμετρο παίρνει το όνομα της μεταβλητής και το όνομα του γεγονότος. Το εξωτερικό επίπεδο δεν πρέπει να επιστρέψει τίποτα στο Communication, απλά ειδοποιείται ότι έγινε κάποιο γεγονός.

Το Communication δίνει την δυνατότητα τα μηνύματα αυτά εκτός του 'Ζήτηση τιμής' να πηγαίνουν και σε άλλα επίπεδα. Αυτό γίνεται για να μπορούν τα μηνύματα να στέλνονται και μέσω δικτύου σε έναν κεντρικό υπολογιστή ώστε να μπορεί ο εκπαιδευτής να παρακολουθεί την πρόοδο των χρηστών.

## 4.3 Model

Το Model επίπεδο περιέχει τις οντότητες που σχετίζονται με το project management.




Σχήμα 8: Οντότητες Model και συσχετισμός

Κεντρική οντότητα είναι η εταιρεία(company) στην οποία δουλεύει ο χρήστης. Η εταιρεία έχει:

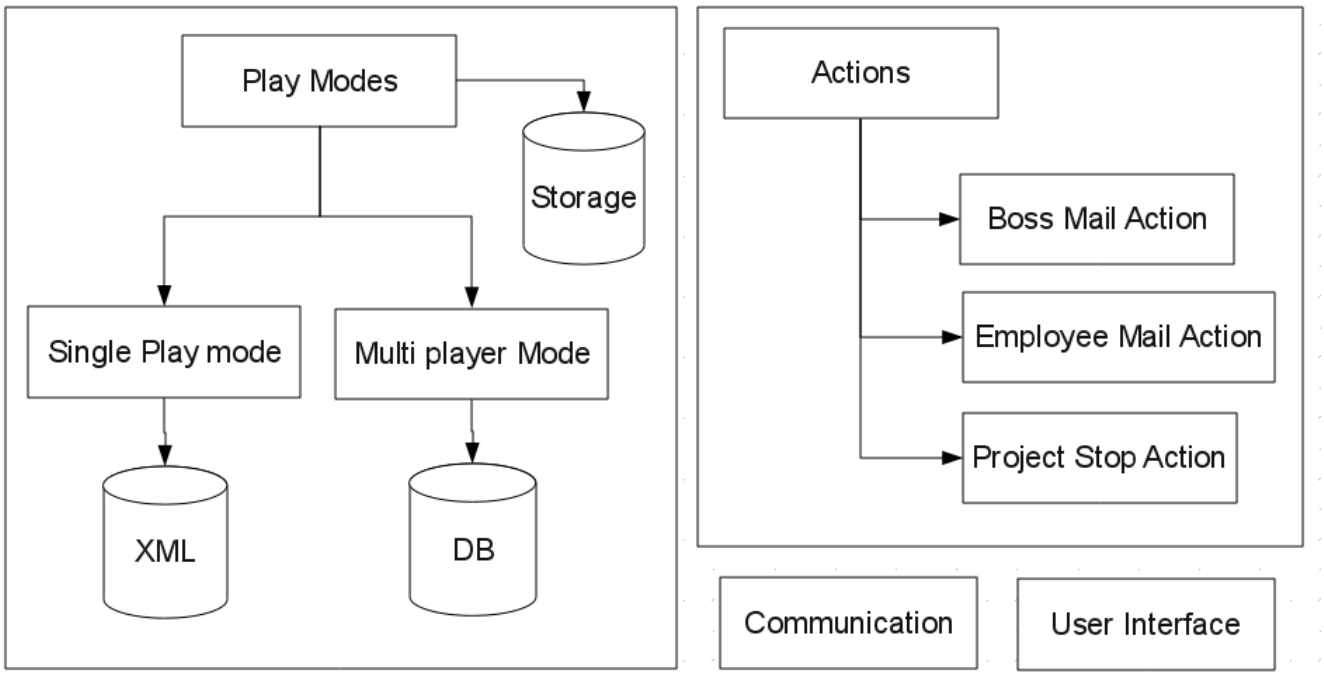
1. **Accountant:** Ο λογιστής της εταιρείας. Υπεύθυνος για τις πληρωμές των υπαλλήλων και την κράτηση λογιστικών δεδομένων για κάθε υπάλληλο.
2. **Administrator:** Υπεύθυνος για το δίκτυο και τους υπολογιστές της εταιρείας.
3. **Boss:** Ο ιδιοκτήτης της εταιρείας. Υπεύθυνος για την ανάθεση των project και άλλων σημαντικών αποφάσεων.
4. **Project Manager:** Ο χρήστης.

5. **Project:** Το τρέχον project με το οποίο ασχολείται η εταιρία.

6. **Employees:** Όλοι οι υπάλληλοι που έχουν προσληφθεί από τον χρήστη και δουλεύουν στην εταιρία.

Η οντότητα project έχει περιγραφτεί στο Κεφάλαιο 3. Οι ικανότητες των υπαλλήλων καθορίζονται από τον εκπαιδευτή. Διάφορα σενάρια απαιτούν οι υπάλληλοι να έχουν διαφορετικές ικανότητες, για παράδειγμα μπορεί για ένα σενάριο να είναι σημαντική η εμπειρία που έχει ο υπάλληλος μέσα στην εταιρία, ή αν ο υπάλληλος έχει οικογένεια κ.τ.λ.

## 4.4 Shell




Σχήμα 9: Shell Frameworks

Το Shell επίπεδο περιλαμβάνει ένα σύνολο από frameworks για την υλοποίηση διάφορων λειτουργιών του ProMaSi . Τα frameworks που περιλαμβάνει είναι:

1. **Play modes:** Εδώ υλοποιούνται τα play modes τα οποία αναλύθηκαν στο Κεφάλαιο 3.

2. **Actions:** Τα actions ενεργοποιούνται όταν ενεργοποιείται ένα γεγονός στο Core. Ο εκπαιδευτής καθορίζει για κάθε γεγονός του Core ποιο action και με τι παραμέτρους θα τρέξει. Το καθένα από αυτά παίρνει και διαφορετικές παραμέτρους. Τα διαθέσιμα actions είναι:

- **Boss Mail Action:** Το action αυτό μόλις εκτελεστεί στέλνει ένα e-mail στο χρήστη από τον ιδιοκτήτη της εταιρείας(boss). Η action παίρνει σαν παραμέτρους τον τίτλο και το κείμενο του e-mail.
- **Employee Mail Action:** Το action αυτό μόλις εκτελεστεί στέλνει ένα e-mail στο χρήστη από έναν υπάλληλο της εταιρείας. Η action παίρνει σαν παραμέτρους τον τίτλο, το κείμενο του e-mail καθώς και έναν κανόνα(X-Path expression) για το ποιος υπάλληλος θα στείλει το e-mail.
- **Project Stop Action:** Το action αυτό μόλις εκτελεστεί σταματάει το project και ο χρήστης βλέπει τα στατιστικά του.

Τα actions θα αναλυθούν εκτενέστερα στο Κεφάλαιο 5.

3. **Communication:** Το Communication είναι η επικοινωνία του Shell με το Core. Εδώ ορίζεται τι κάνει το Shell όταν λάβει ένα μήνυμα από το Core. Το Shell χειρίζεται τα 3 μηνύματα ως εξής:
   - **Ζήτηση τιμής:** Για κάθε μεταβλητή του Core ο εκπαιδευτής ορίζει και ένα X-Path expression, το οποίο όταν εκτελεστεί θα φέρει μια τιμή από το Model επίπεδο.
   - **Ενημέρωση τιμής:** Για κάθε output μεταβλητή του Core ο εκπαιδευτής ορίζει και ένα X-Path expression, το οποίο μόλις εκτελεστεί θα ενημερώσει μια τιμή από μια οντότητα στο Model επίπεδο.
   - **Γεγονός:** Για κάθε γεγονός στο Core ο εκπαιδευτής ορίζει ποιο action θα εκτελεστεί με ποιες παραμέτρους.
4. **User Interface:** Εδώ ορίζονται διάφορα interfaces έτσι ώστε να μπορούμε να χρησιμοποιήσουμε πολλά user interfaces. Για τη δημιουργία ενός user interface πρέπει να οριστούν τα εξής:
   - **Main frame:** Το κεντρικό παράθυρο της εφαρμογής.
   - **Login UI :** Το παράθυρο που χρησιμοποιεί ο χρήστης για να κάνει login στο ProMaSi . Πρέπει για κάθε play mode να γίνει και ένα διαφορετικό Login UI.
   - **Project finished UI:** Το παράθυρο που χρησιμοποιείται για να δείξει τα αποτελέσματα του project όταν τελειώνει(διάφορα στατιστικά) και τη συνολική βαθμολογία σε περίπτωση που έχει τελειώσει το παιχνίδι. Πρέπει για κάθε play mode να γίνει και ένα διαφορετικό project finished UI.

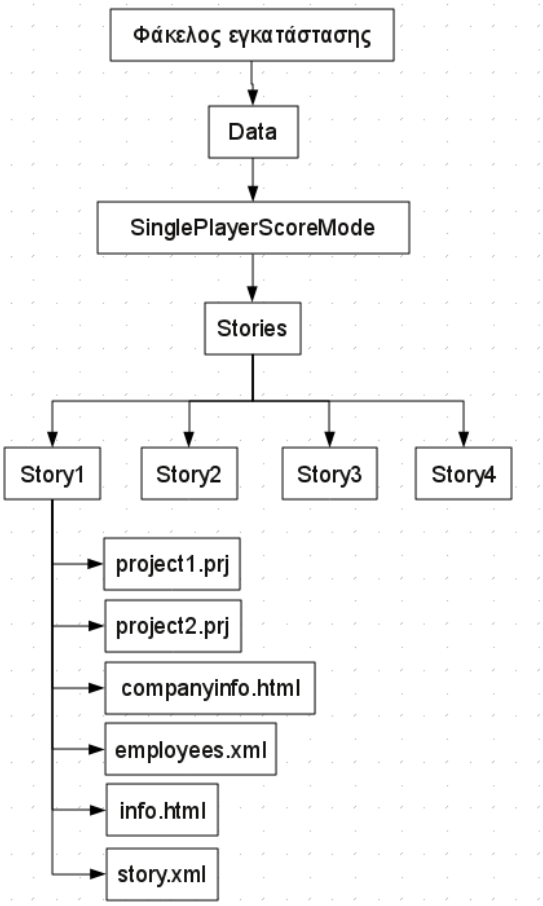
## 4.5 UI

Το UI επίπεδο περιλαμβάνει τις υλοποιήσεις του User interface framework του Shell. Το DesktopUI το οποίο είναι το UI του ProMaSi έχει αναφερθεί στο Κεφάλαιο 3 και θα αναλυθεί στο Κεφάλαιο 6.

# 5 Δημιουργία σεναρίου

Σε αυτό το κεφάλαιο, θα αναλύσουμε τι πρέπει να κάνει ο εκπαιδευτής προκειμένου να δημιουργήσει ένα σενάριο για το single player score mode. Όπως έχουμε αναφέρει, το κάθε play mode έχει διαφορετική δομή αρχείων, για αυτό και η δημιουργία σεναρίου μπορεί να διαφέρει για το καθένα.

Η δομή των αρχείων για το single player score mode είναι:




Κάτω από το φάκελο stories υπάρχουν υποφάκελοι, ο καθένας αντιστοιχεί σε μια ιστορία. Ο κάθε υποφάκελος περιέχει τα εξής αρχεία:

- ***.prj αρχεία:** Είναι ένα zip αρχείο με .prj κατάληξη αντί για .zip. Το αρχείο αυτό περιέχει τον ορισμό ενός project και το όνομα του αρχείου, που πρέπει να είναι το όνομα του project μαζί με την .prj κατάληξη.
- **employees.xml:** Το αρχείο αυτό περιέχει τον ορισμό όλων των υπαλλήλων. Το format του αρχείου είναι ως εξής:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListOfEmployees>
```

```
    <Employee>
        <name>Robert</name>
        <lastName>Marsh</lastName>
        <property name="developer">8</property>
        <property name="designer">9</property>
        <property name="tester">2</property>
        <property name="ubmOtherProductKnowledge">1</property>
        <property name="teamplayer">8</property>
        <salary>1000.0</salary>
        <curriculumVitae></curriculumVitae>
    </Employee>
    <Employee>
        <name>Samuel</name>
        <lastName>Garcia</lastName>
        <property name="developer">5</property>
        <property name="designer">2</property>
        <property name="tester">7</property>
        <property name="teamplayer">5</property>
        <property name="ubmOtherProductKnowledge">1</property>
        <salary>1200.0</salary>
        <curriculumVitae></curriculumVitae>
    </Employee>
    <Employee>
        <name>James</name>
        <lastName>Rowland</lastName>
        <property name="developer">4</property>
        <property name="designer">1</property>
        <property name="tester">10</property>
        <property name="teamplayer">9</property>
        <property name="ubmOtherProductKnowledge">8</property>
        <salary>1500.0</salary>
        <curriculumVitae></curriculumVitae>
    </Employee>
</ListOfEmployees>
```

Για κάθε υπάλληλο δηλώνουμε το όνομα και το επώνυμο του, το μισθό που παίρνει ανά μήνα (23 μέρες), το βιογραφικό του υπαλλήλου και τις ικανότητες του.

- name: Το όνομα του υπαλλήλου.
- lastname: Το επώνυμο του υπαλλήλου.
- property: Μια ικανότητα του υπαλλήλου με συγκεκριμένο όνομα(name) και τιμή.
- salary: Ο μηνιαίος μισθός του υπαλλήλου.
- curriculumVitae: Το βιογραφικό του υπαλλήλου. Υποστηρίζει html format με

εικόνες. Οι εικόνες πρέπει να τοποθετηθούν στον φάκελο της ιστορίας. Επειδή το html format περιλαμβάνει special xml characters (>,<) πρέπει να αντικαταστήσουμε το > με > και το < με <. Το βιογραφικό του υπαλλήλου θα πρέπει να αντιπροσωπεύει τις ικανότητες που έχει ο υπάλληλος.

Όλοι οι υπάλληλοι που δηλώνονται σε αυτό το αρχείο είναι διαθέσιμοι στον χρήστη για πρόσληψη.

- **info.html:** Ένα html αρχείο με την περιγραφή της ιστορίας.
- **story.xml:** Το αρχείο αυτό περιέχει τον ορισμό διάφορων παραμέτρων της ιστορίας. Το format του αρχείου είναι ως εξής:

```
<Story>
        <DifficultyLevel>Novice</DifficultyLevel>
        <StartDate>2008-04-08</StartDate>
        <Projects>
                <Project>Project1</Project>
                <Project>Project2</Project>
        </Projects>
        <Company>
                <Name>UBM</Name>
                <StartTime>09:00</StartTime>
                <EndTime>17:00</EndTime>
                <Description></Description>
        </Company>
        <Persons>
                <Boss>
                        <Name>Chandler</Name>
                        <LastName>Bing</LastName>
                </Boss>
                <Administrator>
                        <Name>Ross</Name>
                        <LastName>Geler</LastName>
                </Administrator>
                <Accountant>
                        <Name>Andrian</Name>
                        <LastName>Green</LastName>
                </Accountant>
        </Persons>
</Story>
```

  - DifficultyLevel: Το επίπεδο δυσκολίας της ιστορίας (Novice, Intermediate, Expert).
  - StartDate: Η ημερομηνία από την οποία θα αρχίσει η εξομοίωση.

- Projects: Τα projects που έχει η ιστορία και με ποια σειρά θα τα παίξει ο χρήστης. Το κάθε project περιέχει το όνομα του project.
- Company: Ο ορισμός της εταιρείας.
  * Name: Το όνομα της εταιρείας.
  * StartTime: Η ώρα που ξεκινάει το ωράριο της εταιρεία.
  * EndTime: Η ώρα που σταματάει το ωράριο της εταιρείας.
  * Description: Η περιγραφή της εταιρείας. Υποστηρίζει html format με εικόνες. Οι εικόνες πρέπει να τοποθετηθούν στον φάκελο της ιστορίας.
- Persons: Αφορά κάθε πρόσωπο της εταιρείας εκτός του χρήστη (Boss, Administrator, Accountant). Δηλώνουμε το όνομα και το επώνυμό του.

Για κάθε project της ιστορίας υπάρχει και ένα .prj αρχείο το οποίο περιέχει τον ορισμό του project. Το αρχείο αυτό είναι ένα zip αρχείο και περιλαμβάνει τα εξής:

- **project.xml:** Το αρχείο αυτό περιλαμβάνει τον ορισμό του project (όνομα, περιγραφή κ.τ.λ.). Το format του αρχείου είναι ως εξής:

```
<Project>
        <name>Bug Fixes(Training)</name>
        <description></description>
        <budget>1000</budget>
        <relativeStartDate>1</relativeStartDate>
        <relativeEndDate>3</relativeEndDate>
        <tasks>
                <Task>
                        <name>Bug 312: Fix</name>
                        <description>Start working on bug 312.</description>
                </Task>
                <Task>
                        <name>Bug 312: Test</name>
                        <description>Test the bug 312</description>
                </Task>
                <Task>
                        <name>Bug 313: Fix</name>
                        <description>Start working on bug 313.</description>
                </Task>
                <Task>
                        <name>Bug 313: Test</name>
                        <description>Test the bug 313</description>
                </Task>
        </tasks>
</Project>
```

- name: Το όνομα του project.
- description: Η περιγραφή του project. Υποστηρίζει html format με εικόνες. Οι εικόνες πρέπει να τοποθετηθούν στον φάκελο της ιστορίας.
- budget: Ο προϋπολογισμός του project.
- relativeStartDate: Η ημερομηνία έναρξης του project. Παίρνει ως όρισμα έναν αριθμό ο οποίος αντιπροσωπεύει πόσες μέρες μετά από την έναρξη της ιστορίας θα ξεκινήσει το project.
- relativeEndDate: Η ημερομηνία που τελειώνει η προθεσμία του project. Παίρνει ως όρισμα έναν αριθμό ο οποίος αντιπροσωπεύει πόσες μέρες μετά από την έναρξη της ιστορίας θα σταματήσει το project.
- tasks: Ο ορισμός των εργασιών του project. Για κάθε εργασία δηλώνουμε όνομα και περιγραφή.

- **sdmodel.xml:** Το αρχείο αυτό περιλαμβάνει τον ορισμό του PSD μοντέλου για το συγκεκριμένο project. Το αρχείο αυτό δημιουργείται με τον Core designer, ο οποίος θα αναλυθεί παρακάτω.

- **coreModelBindings.xml:** Σε αυτό το αρχείο ορίζεται το πως το shell επίπεδο χειρίζεται τα μηνύματα του Core για κάθε μεταβλητή του PSD μοντέλου. Το format του αρχείου είναι ως εξής:

```
<CoreModelBindings>
 <OutputVariables>
  <OutputVariable>
   <sdObjectKey>bug312Progress</sdObjectKey>
   <modelXPath>currentProject/tasks[name='Bug␣312:␣Fix']
       /percentageCompleted</modelXPath>
  </OutputVariable>
  <OutputVariable>
   <sdObjectKey>projectProgress</sdObjectKey>
   <modelXPath>currentProject/percentageCompleted</modelXPath>
  </OutputVariable>
  </OutputVariables>
 <ExternalEquations>
  <ExternalEquation>
   <sdObjectKey>sumDeveloperFix312</sdObjectKey>
   <modelXPath>sum(currentProject/tasks[name='Bug␣312:␣Fix']/currentSchedule
       /team/assignedEmployees/properties[name='developer']/value)</modelXPath>
  </ExternalEquation>
  <ExternalEquation>
   <sdObjectKey>sumTeamplayerFix312</sdObjectKey>
   <modelXPath>sum(currentProject/tasks[name='Bug␣312:␣Fix']/currentSchedule
```

```
        /team/assignedEmployees/properties[name='teamplayer']/value)</modelXPath>
    </ExternalEquation>
  </ExternalEquations>
  <Events>
    <Event>
      <sdObjectKey>bug312Progress</sdObjectKey>
      <eventName>bug312Finished</eventName>
      <actionBinding>
        <actionClassName>org.promasi.shell.model.actions
          .BossMailModelAction</actionClassName>
        <parameter name="message">I seems that you have
          finished with bug 312.</parameter>
        <parameter name="title">Bug 312 finished!</parameter>
      </actionBinding>
    </Event>
    <Event>
      <sdObjectKey>projectProgress</sdObjectKey>
      <eventName>projectFinished</eventName>
      <actionBinding>
        <actionClassName>org.promasi.shell.model.actions.
          ProjectStopModelAction</actionClassName>
      </actionBinding>
    </Event>
  </Events>
</CoreModelBindings>
```

- OutputVariables: Εδώ, για κάθε output μεταβλητή του Core, δηλώνουμε σε ποιό σημείο θα αποθηκευτεί η τιμή της (δηλαδή σε ποιά οντότητα του επιπέδου Model). Στο OutputVariable δηλώνουμε σε ποια μεταβλητή του Core αντιστοιχεί (μέσω του sdObjectKey) και που θα αποθηκευτεί η τιμή της (modelXPath).
- ExternalEquations: Εδώ δηλώνουμε για κάθε μεταβλητή του Core με external συνάρτηση ποια τιμή θα επιστρέψει το Shell από το Model επίπεδο.
- Events: Εδώ δηλώνουμε για κάθε γεγονός του Core ποιο action και με τι παραμέτρους θα εκτελεστεί.

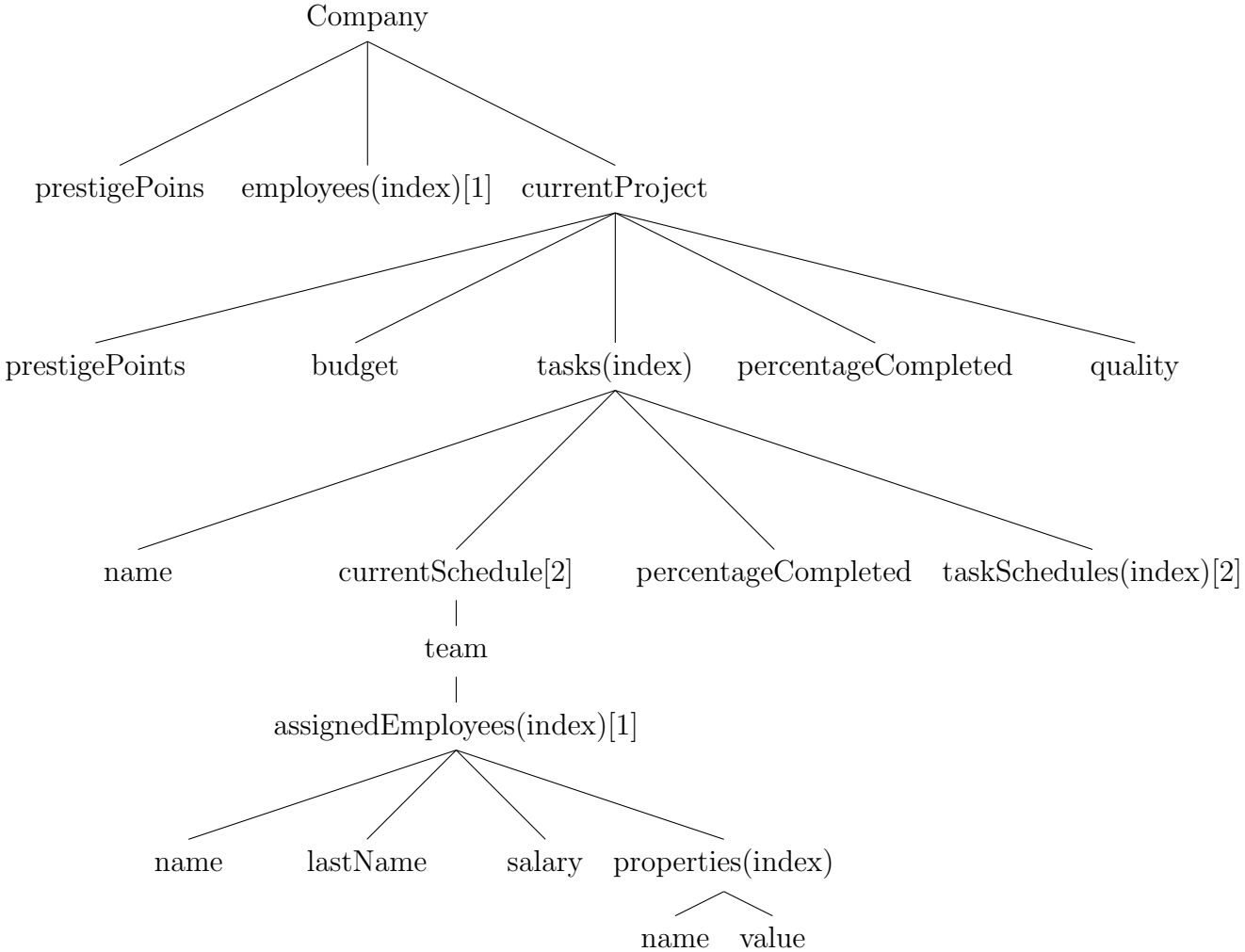
## 5.1 X-Path Expressions

XPath ή XML Path Language, είναι μια query language για την επιλογή των κόμβων από ένα έγγραφο XML. Επιπλέον, μπορεί να χρησιμοποιηθεί για τον υπολογισμό των τιμών από το περιεχόμενο ενός εγγράφου XML. Το XPath ορίστηκε από το World Wide Web Consortium (W3C). Η γλώσσα αυτή βασίζεται στην εκπροσώπηση του XML ως δέντρο, και παρέχει τη δυνατότητα πλοήγησης στους κόμβους του χρησιμοποιώντας διάφορα κριτήρια.

Όπως είδαμε στο παραπάνω παράδειγμα στο coreModelBindings.xml, για κάθε OutputVariable και ExternalEquation ορίζουμε μία τιμή για την ιδιότητα modelXPath. Αυτή είναι ένα X-Path expression που αφορά στο επίπεδο Model. Το X-Path expression δείχνει στο Shell που να βάλει την τιμή και από που να πάρει μια τιμή αντίστοιχα. Το X-Path ξεκινάει με την εταιρεία σαν root αντικείμενο.

Για παράδειγμα, για να πάρουμε τα prestige points του project θα γράψουμε το εξής: currentProject/prestigePoints. Το παραπάνω μεταφράζεται ως εξής: από την εταιρεία πάρε το currentProject(το τρέχων project) και από αυτό πάρε την μεταβλητή prestigePoints. Εφόσον το Company είναι το root αντικείμενο δεν χρειάζεται να το συμπεριλάβουμε στο X-Path.

Οι μεταβλητές που έχουμε στη διάθεσή μας είναι:




Στο παραπάνω διάγραμμα, παρατηρούμε ότι μερικές μεταβλητές χαρακτηρίζονται ως index. Αυτό σημαίνει ότι οι συγκεκριμένες μεταβλητές είναι ένας πίνακας. Για να πάρουμε μια τιμή από μια index μεταβλητή, πρέπει πρώτα να επιλέξουμε ποια τιμή θέλουμε σύμφωνα με κάποιο κριτήριο. Για παράδειγμα, το X-Path currentProject/tasks[name='Bug 313']/percentageCompleted θα μας επιστρέψει την τιμή του percentageCompleted από το

task με το όνομα Bug 313.

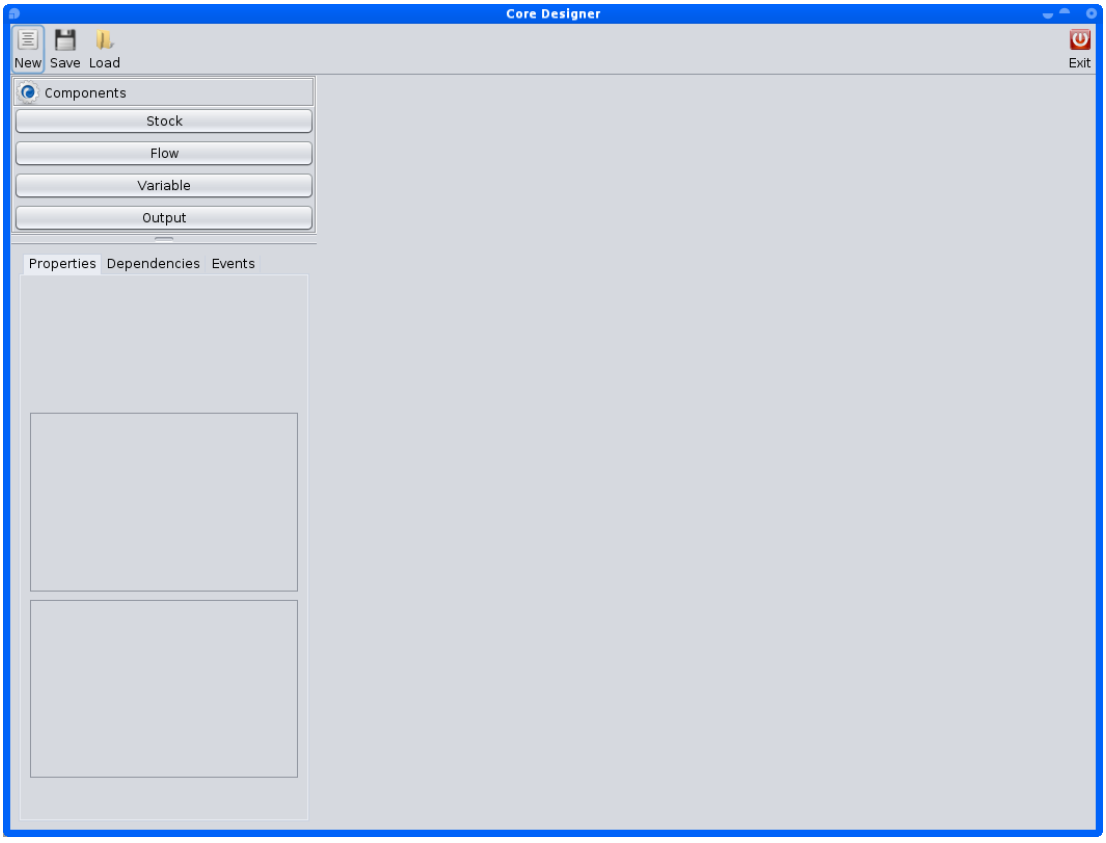
Το X-Path υποστηρίζει και συναρτήσεις όπως:

- **min**
- **avg**
- **count**
- **max**
- **sum**

Για να πάρουμε το σύνολο της 'developer' ιδιότητας όλων των υπαλλήλων που δουλεύουν στο task Bug 313 θα γράψουμε το εξής X-Path: sum(currentProject/tasks[name='Bug 313']/currentSchedule/team/assignedEmployees/properties[name='developer']/value)

## 5.2 Core Designer

Ο Core designer είναι ένα εργαλείο το οποίο βοηθάει τον εκπαιδευτή να φτιάχνει PSD μοντέλα. Όλα τα παραπάνω αρχεία τα φτιάχνει ο εκπαιδευτής μόνος του. Επειδή όμως ένα PSD είναι αρκετά πολύπλοκο για να φτιαχτεί χωρίς κάποιο βοηθητικό εργαλείο, δημιουργήθηκε ο Core designer.




Σχήμα 10: Κεντρική οθόνη του Core designer

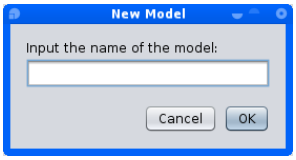
Στην κεντρική οθόνη του Core designer έχουμε τις εξής δυνατότητες:

1. Δημιουργία καινούργιου μοντέλου.

2. Άνοιγμα καινούργιου μοντέλου.

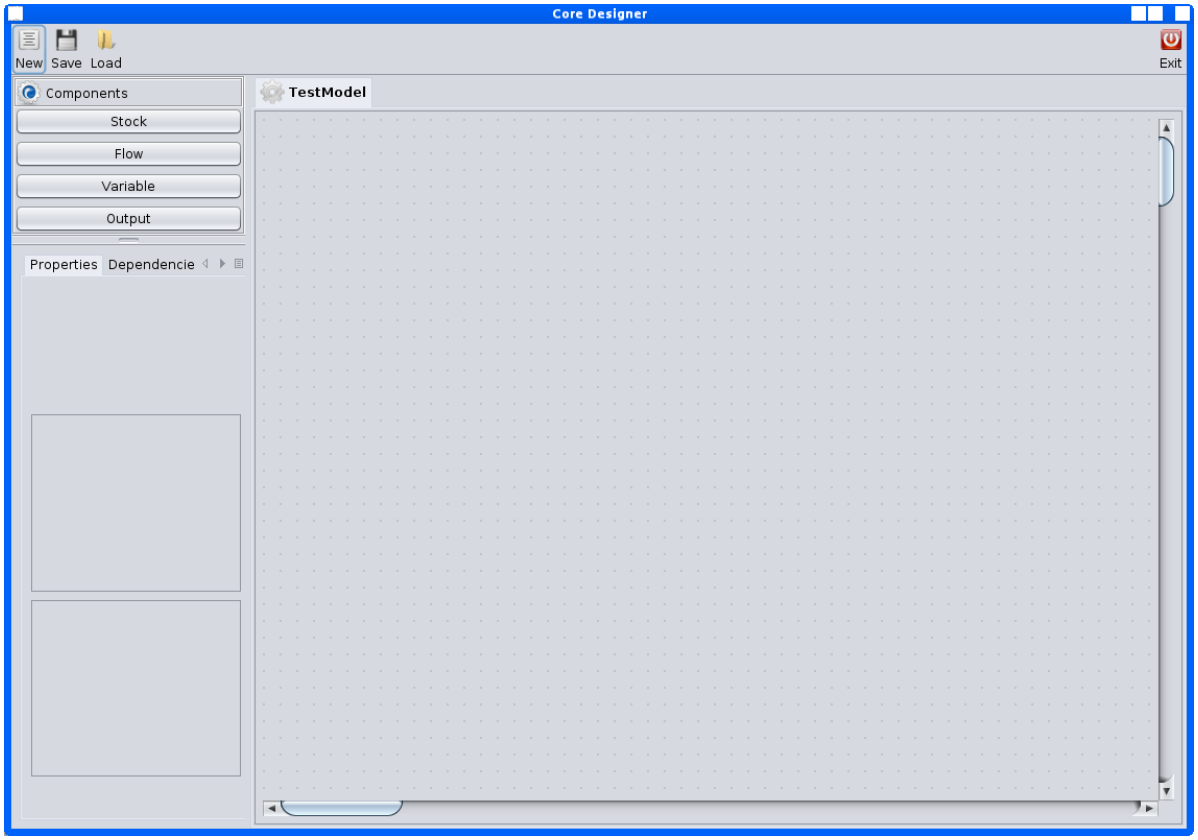
3. Έξοδος απο την εφαρμογή.

Αφού επιλέξουμε να δημιουργήσουμε ένα καινούργιο μοντέλο, εμφανίζεται μια οθόνη η οποία μας ζητάει να δώσουμε το όνομα του μοντέλου.



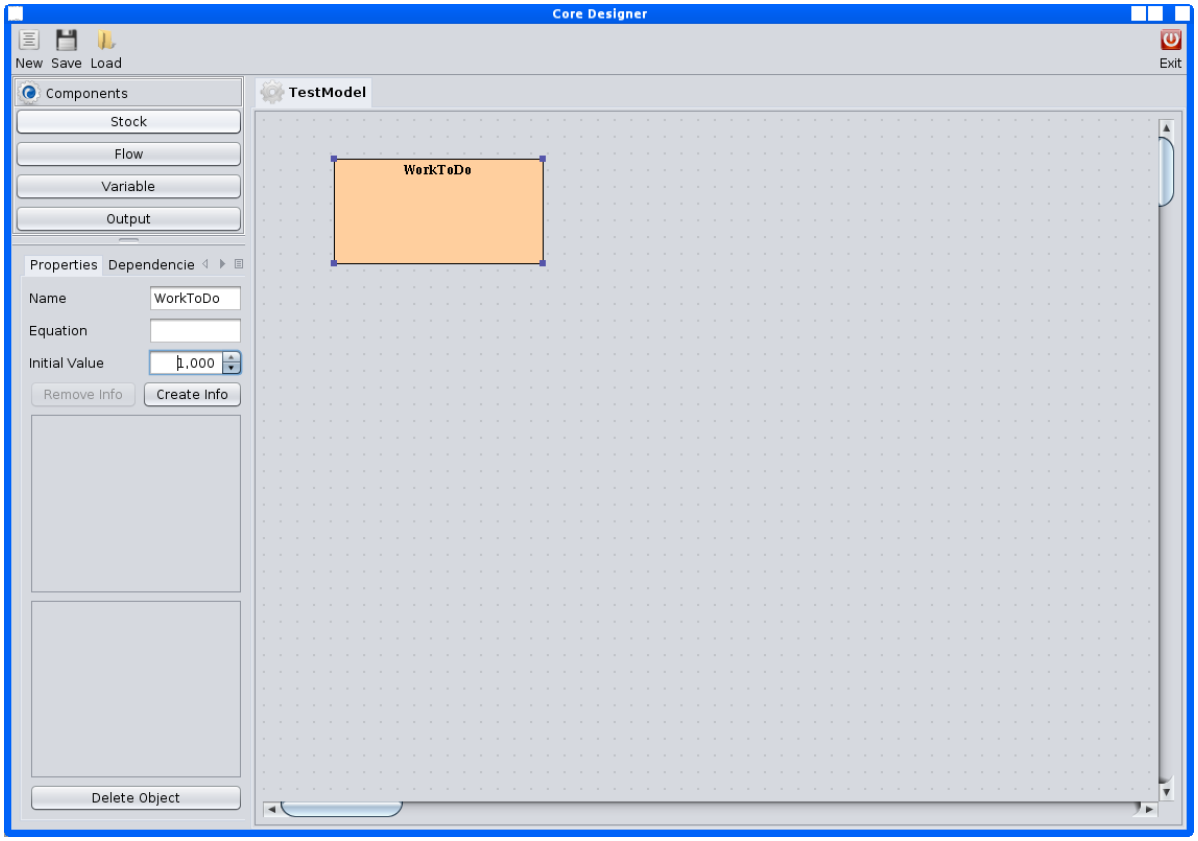

Σχήμα 11: Δημιουργία καινούργιου μοντέλου

Αφού δώσουμε το όνομα του μοντέλου μπορούμε να αρχίσουμε να προσθέτουμε μεταβλητές.



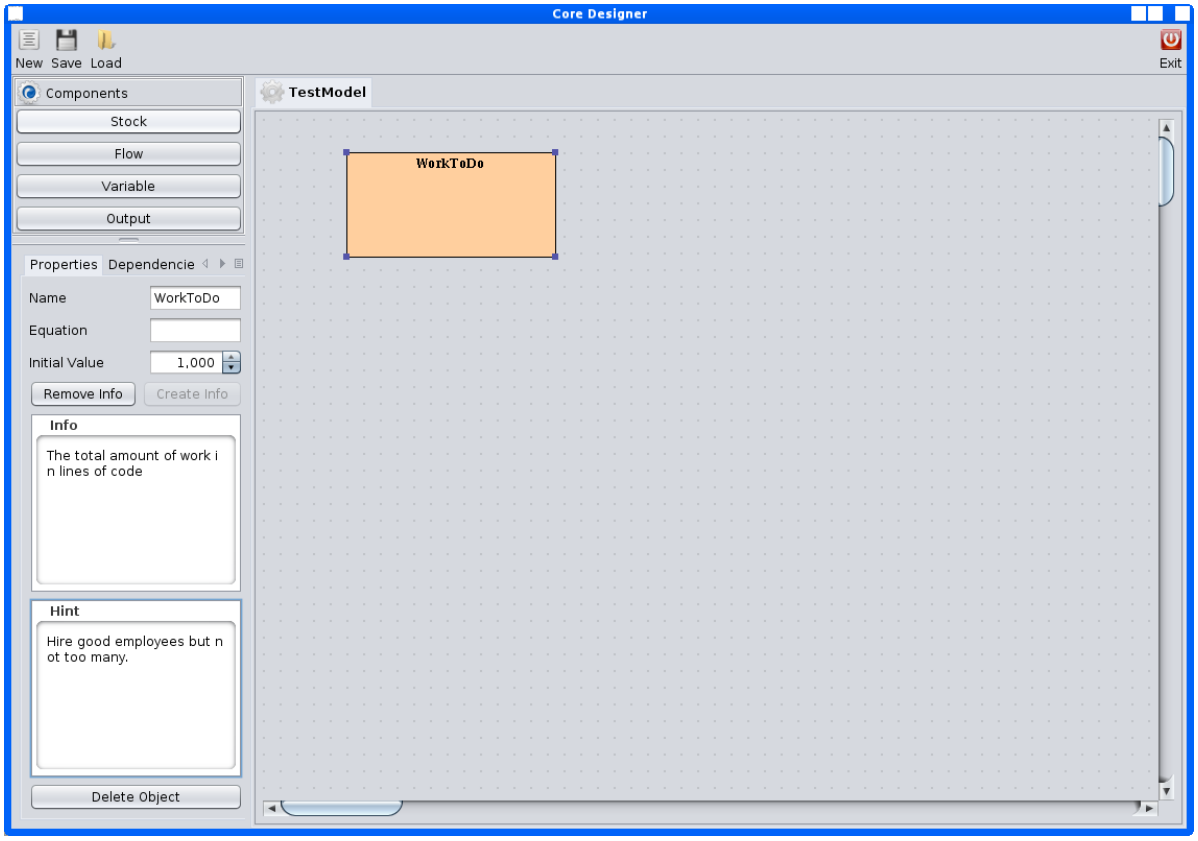

Σχήμα 12: Core designer μετά τη δημιουργία του μοντέλου

Για να προσθέσουμε μια μεταβλητή, πατάμε ένα απ τα κουμπιά Stock, Flow, Variable, Output και το σέρνουμε στην περιοχή σχεδίασης.



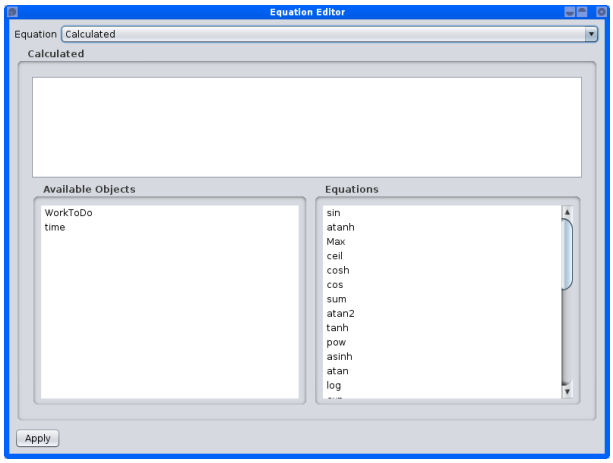

Σχήμα 13: Core designer μετά τη δημιουργία μιας μεταβλητής

Αφού δημιουργηθεί η μεταβλητή μας, μπορούμε μέσα από την καρτέλα properties να βάλουμε το όνομα της μεταβλητής, την εξίσωσή της όπως επίσης info, hint, τα οποία αν δηλωθούν θα φαίνεται η μεταβλητή στα στατιστικά του project. Για να προσθέσουμε info στην μεταβλητή, πατάμε το κουμπί Create info και για να το βγάλουμε πατάμε το κουμπί Remove info. Στις μεταβλητές τύπου Stock στην καρτέλα properties θα εμφανιστεί και μία επιπλέον παράμετρος(initial value), η οποία καθορίζει την αρχική τιμή του stock.




Σχήμα 14: Core designer μετά την πρόσθεση info,hint

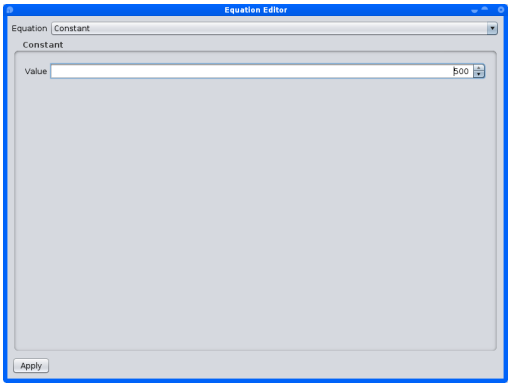
Κάνοντας διπλό κλικ στο πεδίο Equation, εμφανίζεται ο Equation editor ο οποίος μας επιτρέπει να βάλουμε μια εξίσωση στην επιλεγμένη μεταβλητή.



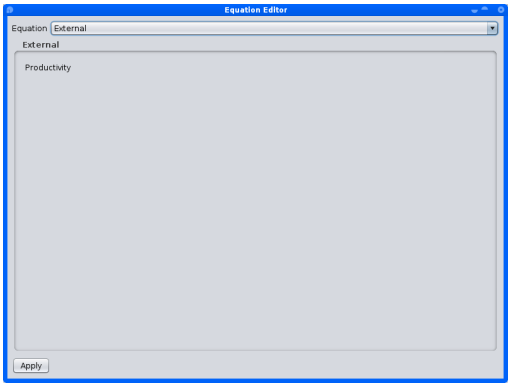

Σχήμα 15: Equation editor για την calculated εξίσωση.

Στο Equation πεδίο διαλέγουμε τον τύπο της συνάρτησης. Για την calculated συνάρτηση

βλέπουμε ότι η οθόνη χωρίζεται σε 3 περιοχές. Στην πάνω περιοχή γράφουμε τη συνάρτηση, κάτω αριστερά βλέπουμε όλες τις μεταβλητές που υπάρχουν στο μοντέλο και έχουν δηλωθεί σαν εξαρτήσεις αυτού του αντικειμένου. Κάτω δεξιά βλέπουμε όλες τις συναρτήσεις που υποστηρίζονται. Κάνοντας διπλό κλικ σε μια συνάρτηση, προστίθεται στην εξίσωση μαζί με τον αριθμό μεταβλητών που παίρνει. Κάνοντας διπλό κλικ σε μια διαθέσιμη μεταβλητή προστίθεται το όνομα της στην εξίσωση.

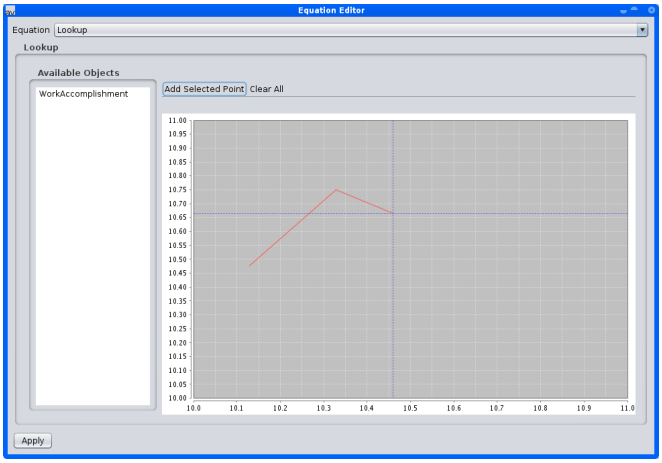



(α′) Equation editor για την constant συνάρτηση




(β′) Equation editor για την external συνάρτηση

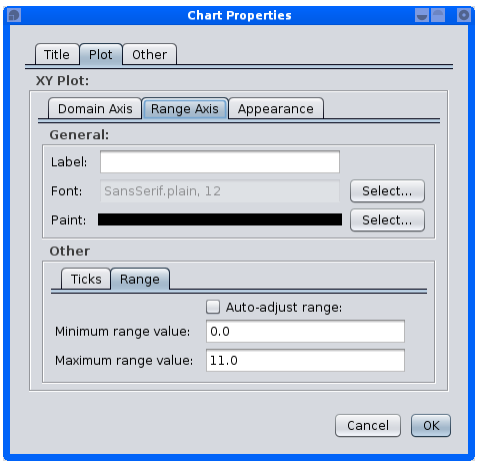
Στην constant συνάρτηση, απλά βάζουμε την τιμή που θέλουμε να έχει η μεταβλητή. Στην external συνάρτηση, απλά μας δείχνει ποιο είναι το όνομα της επιλεγμένης μεταβλητής το οποίο θα στέλνεται στο communication.




Σχήμα 16: Equation editor για την lookup συνάρτηση.

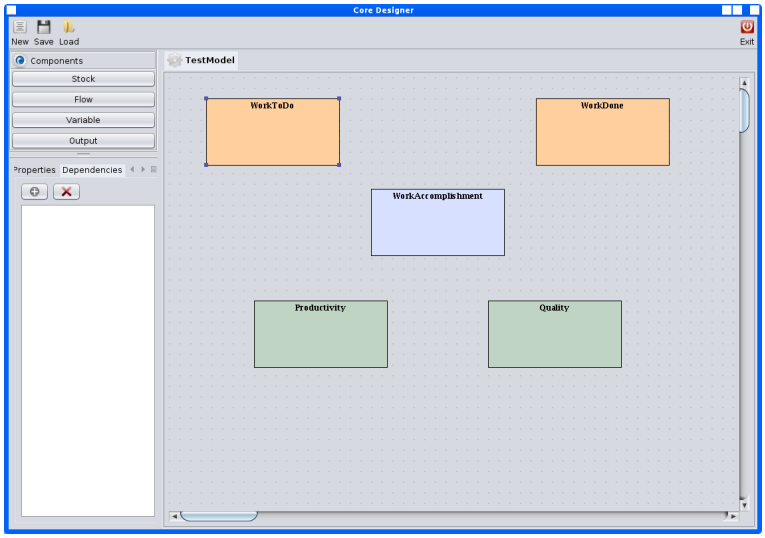
Στην lookup συνάρτηση, σχηματίζουμε μια γραφική παράσταση κάνοντας κλικ σε κάποιο σημείο μέσα στους άξονες και πατώντας το κουμπί Add selected Point προσθέτουμε το σημείο που έχουμε επιλέξει. Μπορούμε έτσι να προσθέσουμε όσα σημεία θέλουμε για να σχηματίσουμε τη γραφική παράσταση. Για να καθαρίσουμε όλα τα σημεία πατάμε το κουμπί

Clear All. Στα αριστερά μας, δίνεται μία λίστα με όλες τις μεταβλητές του μοντέλου που έχουν προστεθεί σαν εξαρτήσεις στην επιλεγμένη μεταβλητή. Επιλέγουμε μια από αυτές τις μεταβλητές. Σε κάθε βήμα, για να υπολογιστεί η τιμή της lookup, θα παίρνει την τιμή της επιλεγμένης μεταβλητής και θα την βάζει σαν x προκειμένου να πάρει το y σύμφωνα με την γραφική παράσταση. Αν η τιμή της μεταβλητής είναι έξω από τα όρια της παράστασης θα επιστραφεί η τιμή 0. Για να αλλάξουμε τα όρια των x-y αξόνων κάνουμε δεξί κλικ στην παράσταση και επιλέγουμε το properties και στην καρτέλα Plot->Range Axis αλλάζουμε το Minimum range value- Maximum range value και πατάμε το κουμπί OK.



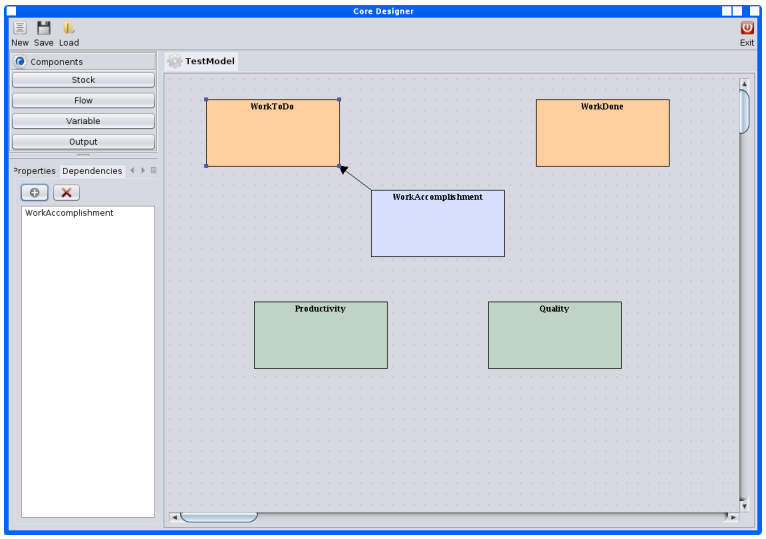

Σχήμα 17: Αλλαγή των ορίων x-y

Για να δηλώσουμε τις εξαρτήσεις μιας μεταβλητής, αφού την επιλέξουμε, πηγαίνουμε στην καρτέλα Dependencies και από εκεί έχουμε 2 επιλογές: να προσθέσουμε μία εξάρτηση ή να αφαιρέσουμε την επιλεγμένη μεταβλητή από τις εξαρτήσεις. Όταν πατήσουμε το κουμπί για την πρόσθεση μιας καινούργιας εξάρτησης, εμφανίζεται ο dependency editor ο οποίος μας δείχνει όλες τις διαθέσιμες μεταβλητές του μοντέλου. Επιλέγουμε την μεταβλητή που θέλουμε να προσθέσουμε σαν εξάρτηση και πατάμε το κουμπί add. Το όνομα της μεταβλητής θα εμφανιστεί στην καρτέλα Dependencies. Για να αφαιρέσουμε μία εξάρτηση, επιλέγουμε την μεταβλητή την οποία θέλουμε να αφαιρέσουμε από την καρτέλα Dependencies και πατάμε το κουμπί delete.



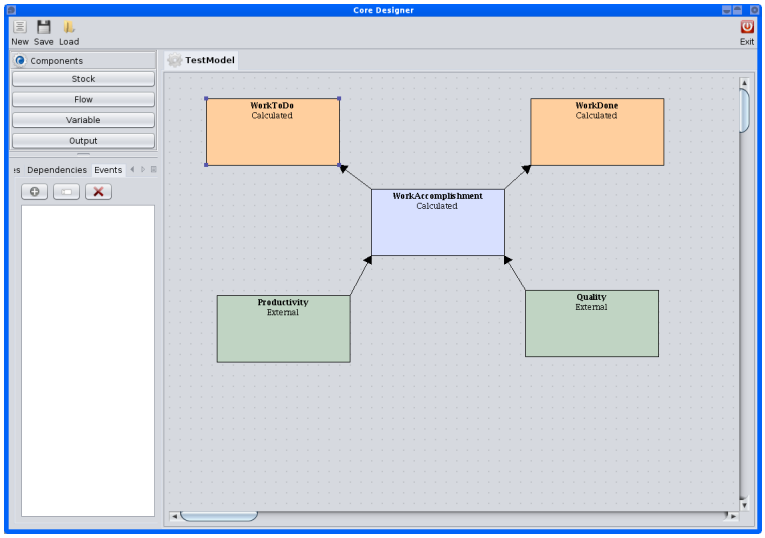

(α′) Καρτέλα dependencies


(β′) Dependency editor

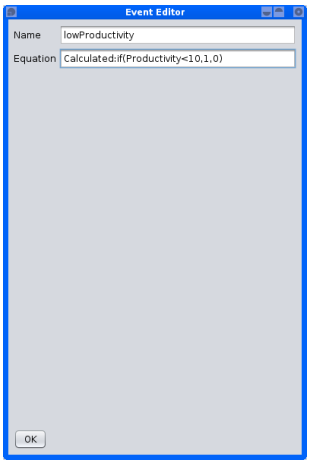



(γ′) Core designer μετά την πρόσθεση μιας εξάρτησης

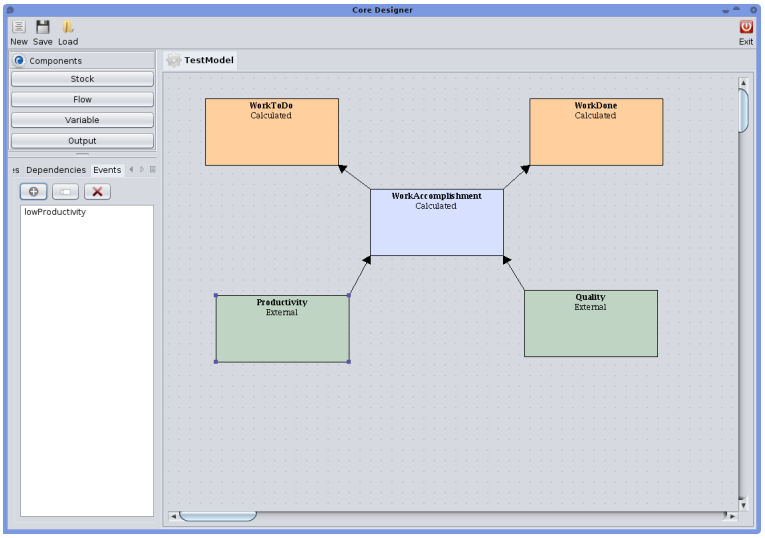
Για να προσθέσουμε ένα γεγονός σε μια μεταβλητή, αφού την επιλέξουμε, πηγαίνουμε στην καρτέλα Events. Σε αυτήν την καρτέλα, βλέπουμε όλα τα γεγονότα της επιλεγμένης μεταβλητής. Μπορούμε να προσθέσουμε, να επεξεργαστούμε και να διαγράψουμε ένα γεγονός. Μόλις πατήσουμε το κουμπί για την πρόσθεση ενός γεγονότος, εμφανίζεται ο Event editor. Εδώ ορίζουμε το όνομα του γεγονότος καθώς και την εξίσωσή του. Μόλις κάνουμε διπλό κλικ στο πεδίο Equation, εμφανίζεται ο Equation editor ο οποίος περιγράφτηκε παραπάνω. Το γεγονός θα ενεργοποιηθεί όταν η συνάρτηση επιστρέψει τιμή μεγαλύτερη του 1. Για να επεξεργαστούμε ένα γεγονός, επιλέγουμε το γεγονός που θέλουμε προς επεξεργασία και πατάμε το κουμπί επεξεργασίας(2ο) για να εμφανιστεί ο Event editor. Για να διαγράψουμε ένα γεγονός, επιλέγουμε το γεγονός και πατάμε το κουμπί διαγραφής(3ο).




(δ′) Καρτέλα events




(ε′) Event editor




(ϛ′) Core designer μετά την πρόσθεση ενός γεγονότος

Τέλος για να σώσουμε το μοντέλο μας, πατάμε το κουμπί Save και επιλέγουμε την τοποθεσία στην οποία θέλουμε να σώσουμε το μοντέλο. Για να φορτώσουμε ένα μοντέλο πατάμε στο κουμπί Load και επιλέγουμε το αρχείο που θέλουμε να φορτώσουμε.

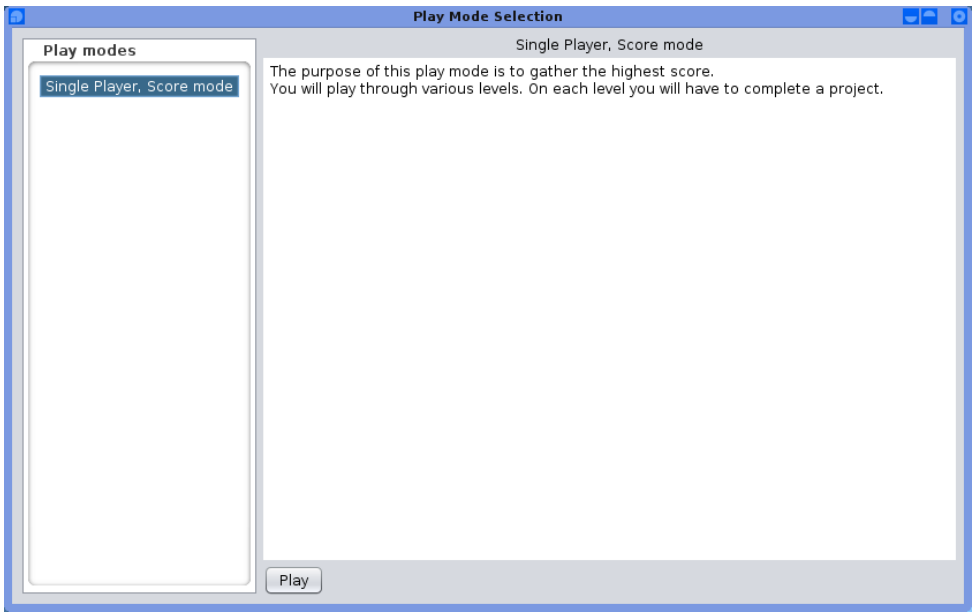
# 6 Περιβάλλον χρήστη(UI)

Το περιβάλλον του χρήστη(Desktop UI) είναι ένα εικονικό λειτουργικό σύστημα, το οποίο παρέχει στο χρήστη διάφορα προγράμματα για να μπορεί να επικοινωνεί με τους υπαλλήλους, να τους αναθέτει εργασίες κ.τ.λ. Συγκεκριμένα, το Desktop UI προσφέρει στον χρήστη τα εξής εργαλεία:

- **Evolution bird:** Ο χρήστης μέσω αυτού του προγράμματος λαμβάνει e-mail για διάφορα γεγονότα.
- **Marketplace:** Χρησιμοποιείται για την πρόσληψη υπαλλήλων.
- **Planner:** Χρησιμοποιείται για την ανάθεση εργασιών στους υπαλλήλους.
- **Infogate:** Ο χρηστής με αυτό το πρόγραμμα μπορεί να δει διάφορες πληροφορίες σχετικά με το project, με την εταιρεία και τους υπαλλήλους της εταιρείας.

## 6.1 Ξεκινώντας το παιχνίδι

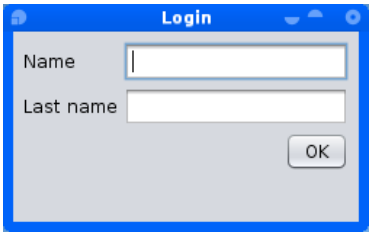
Μόλις ο χρήστης ξεκινήσει το παιχνίδι εμφανίζεται η οθόνη για την επιλογή play mode.




Σχήμα 18: Οθόνη επιλογής play mode

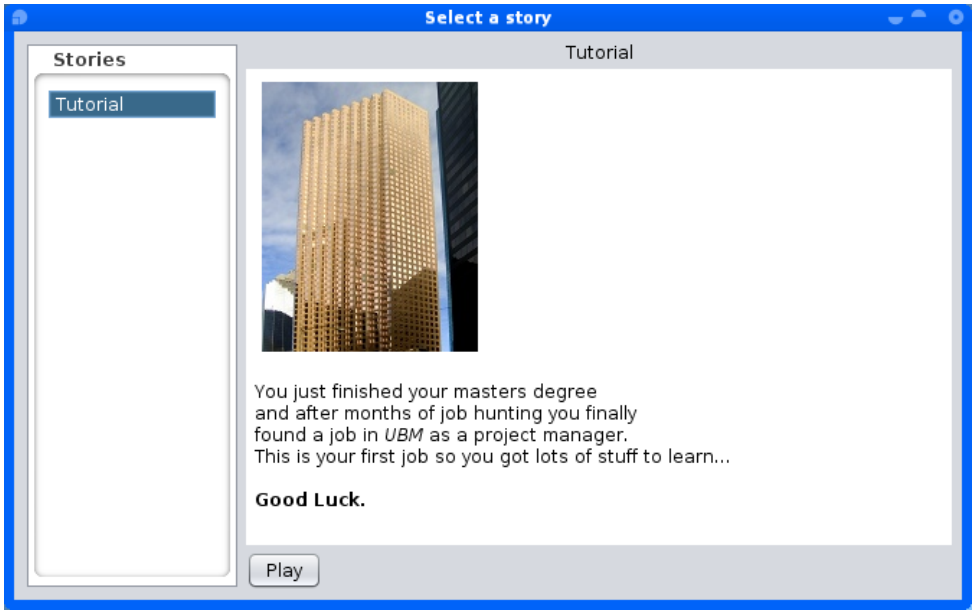
Εδώ ο χρήστης διαλέγει το play mode που θέλει να παίξει(αυτή τη στιγμή είναι διαθέσιμο μόνο το single player score mode). Όταν το επιλέξει εμφανίζεται η περιγραφή του και πατάει Play για να συνεχίσει.

Στη συνέχεια, εμφανίζεται η οθόνη εισαγωγής στοιχείων. Ο χρήστης εισάγει το όνομα και το επώνυμο του και πατάει OK.




Σχήμα 19: Οθόνη εισαγωγής στοιχείων χρήστη

Μετά την εισαγωγή στοιχείων εμφανίζεται η οθόνη για την επιλογή ιστορίας. Εδώ ο χρήστης βλέπει όλες τις διαθέσιμες ιστορίες και επιλέγει ποια ιστορία θέλει να παίξει. Μόλις ο χρήστης επιλέξει μια ιστορία εμφανίζεται η περιγραφή της. Στη συνέχεια πατάει play για να παίξει την επιλεγμένη ιστορία.




Σχήμα 20: Οθόνη επιλογής ιστορίας

Έπειτα εμφανίζεται η κεντρική οθόνη του Desktop UI. Το μέγεθος του παραθύρου πιάνει όλη την οθόνη, έτσι ώστε να δοθεί η ψευδαίσθηση ότι ο χρήστης βρίσκεται σε ένα λειτουργικό σύστημα. Στο πάνω μέρος της οθόνης, βλέπουμε την μπάρα εργαλείων. Αυτή χωρίζεται σε 2 μέρη: Αριστερά έχουμε όλα τα διαθέσιμα προγράμματα (Evolution bird, Marketplace, Planner, Infogate). Όταν πατάμε ένα κουμπί για να ανοίξει ένα πρόγραμμα, αυτό εμφανίζεται στο κέντρο της οθόνης σαν ένα παράθυρο. Αν το πρόγραμμα είναι ήδη ανοιχτό, τότε

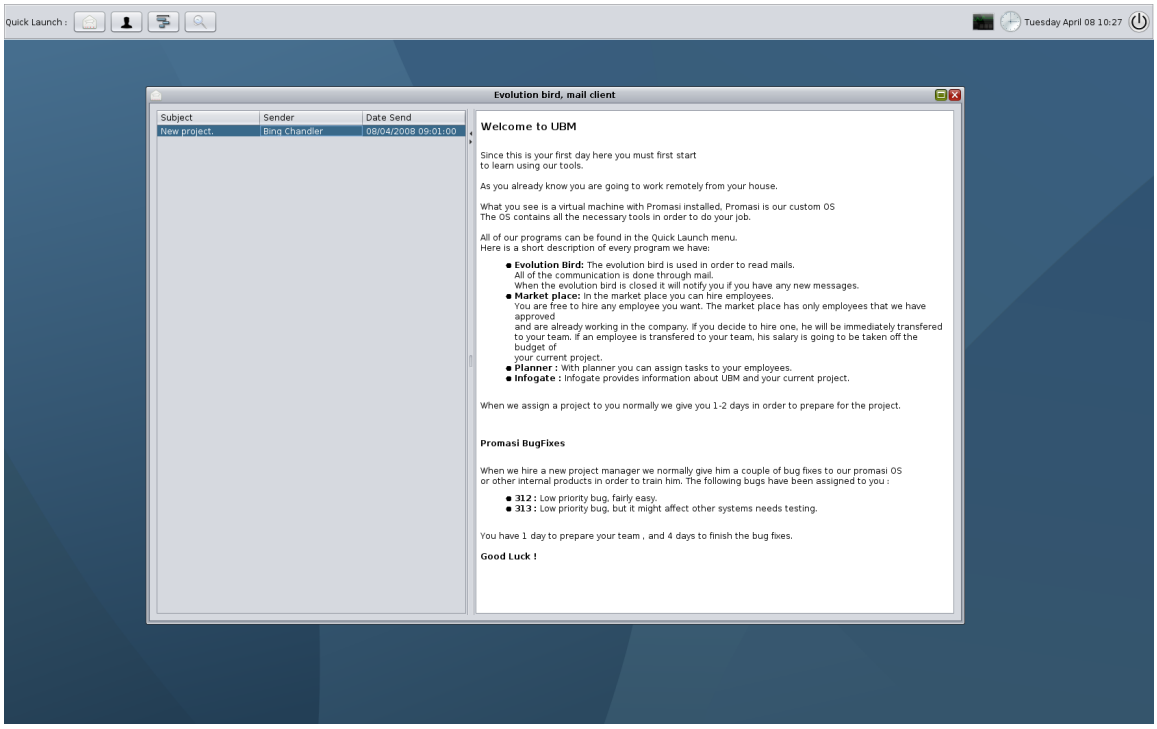
το παράθυρο που αντιστοιχεί στο πρόγραμμα θα έρθει μπροστά από τα υπόλοιπα παράθυρα. Στην δεξιά μεριά, υπάρχουν εργαλεία που αλλάζουν/δείχνουν τη ροή του παιχνιδιού. Ένα από αυτά είναι το ρολόι που δείχνει την ώρα και μπορεί ο χρήστης να αλλάξει την ταχύτητα του χρόνου. Ακόμη υπάρχει ένα εργαλείο που δείχνει πόσες μέρες έμειναν για να ξεκινήσει ή να τελειώσει το project. Μέσα από αυτό, ο χρήστης μπορεί να πάει κατευθείαν στην ημέρα που ξεκινάει το project.


Σχήμα 21: Κεντρική οθόνη του desktop UI

## 6.2 Ανάθεση project

Όταν ξεκινάει το παιχνίδι ο χρήστης βλέπει μια ειδοποίηση ότι έχει ένα καινούργιο μήνυμα. Κάνοντας κλικ πάνω στην ειδοποίηση ανοίγει το Evolution bird.


(α′) Ειδοποίηση από το Evolution bird




(β′) Evolution bird

Το Evolution bird χρησιμοποιείται μόνο για ανάγνωση μηνυμάτων. Το πρόγραμμα χωρίζεται σε 2 μέρη: αριστερά βλέπουμε μία λίστα με όλα τα μηνύματα και δεξιά βλέπουμε τα περιεχόμενα του επιλεγμένου μηνύματος(e-mail). Στη λίστα με τα e-mails βλέπουμε το θέμα του μηνύματος, τον αποστολέα και την ημερομηνία αποστολής. Το πρώτο μήνυμα που

λαμβάνει ο χρήστης είναι η ανάθεση ενός project και περιέχει όλες τις πληροφορίες για το project που θα αναλάβει ο χρήστης.

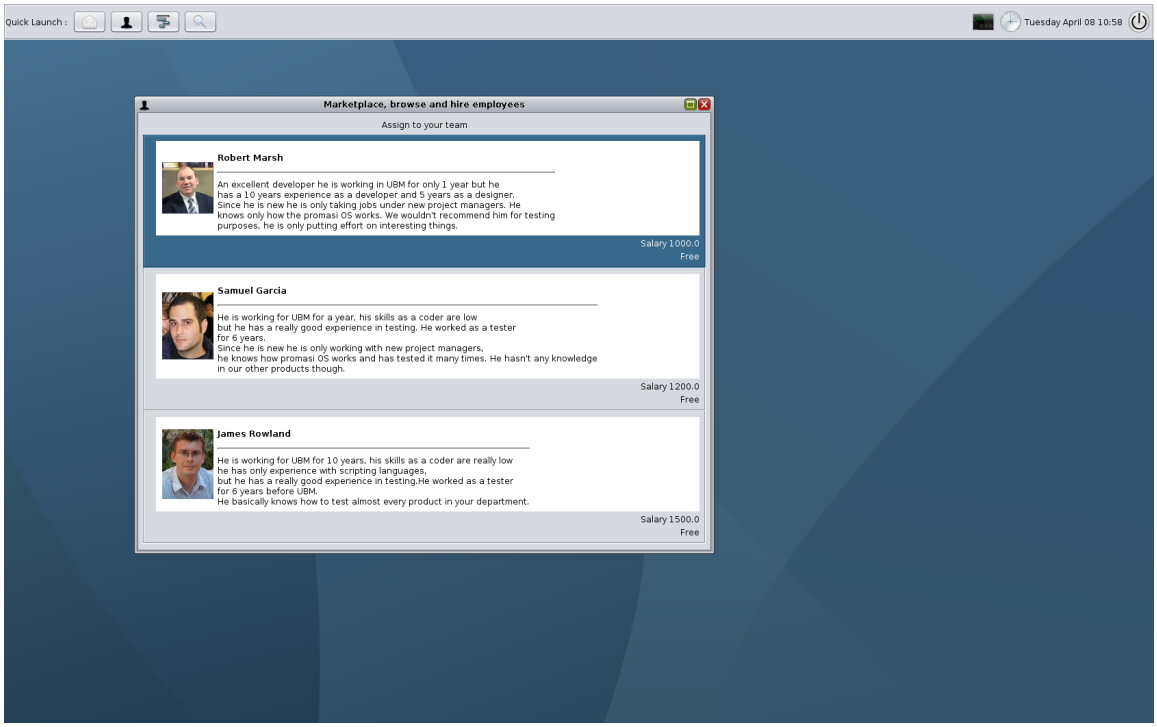
## 6.3 Προετοιμασία

Αφού ανατεθεί το πρώτο project, ο χρήστης πρέπει να προετοιμαστεί πριν ξεκινήσει το project. Συνήθως ο εκπαιδευτής, ειδικά στα πρώτα project, θα πρέπει να δώσει στον χρήστη αρκετό χρόνο για να δει όλους τους υπαλλήλους και να αποφασίσει ποιους θα προσλάβει και να τους αναθέσει εργασίες. Για να μπορεί ο χρήστης να πάρει πιο άνετα τις αποφάσεις του, μπορεί να κάνει κλικ στο ρολόι και να πατήσει το κουμπί slow, προκειμένου ο χρόνος να κυλάει πιο αργά.


Σχήμα 22: Ρολόι του desktop UI

### 6.3.1 Πρόσληψη υπαλλήλων

Το πρώτο πράγμα που πρέπει να κάνει ο χρήστης κατά την προετοιμασία είναι να διαλέξει τους κατάλληλους υπαλλήλους, ειδικά στο πρώτο project που η εταιρεία δεν έχει κανέναν υπάλληλο. Για να προσλάβει υπαλλήλους ο χρήστης πρέπει να ανοίξει το πρόγραμμα Marketplace.

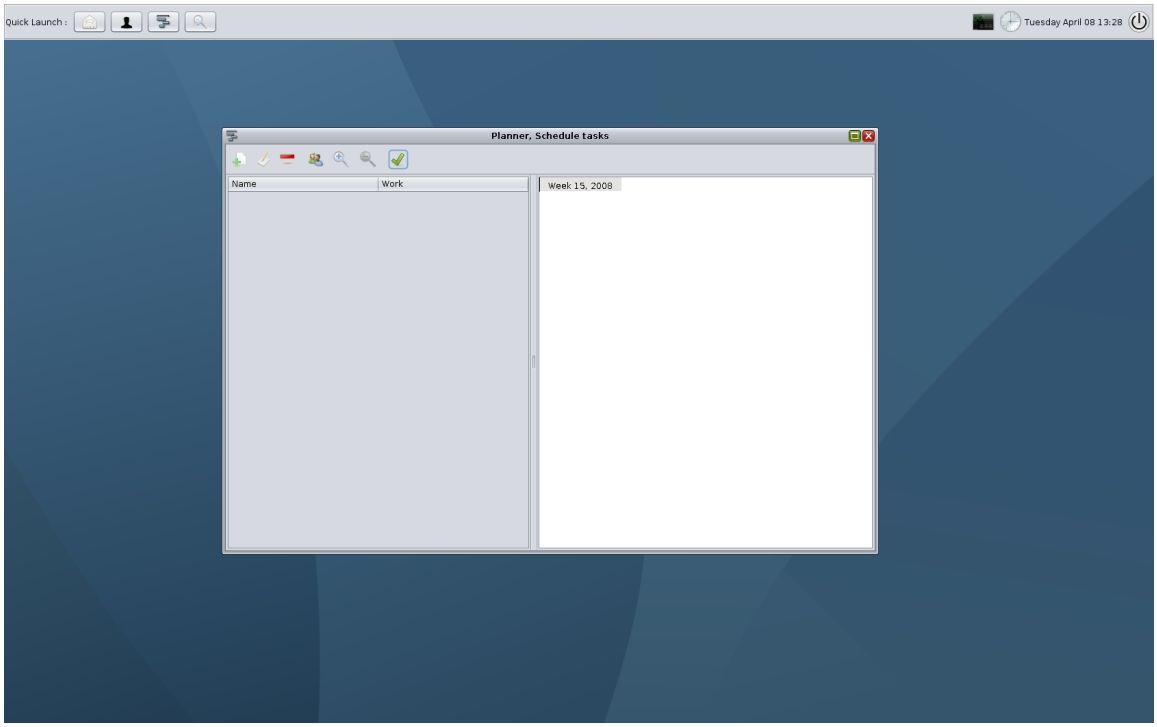



Σχήμα 23: Marketplace

Ο χρήστης βλέπει όλα τα βιογραφικά των διαθέσιμων υπαλλήλων. Συνήθως τα βιογραφικά περιλαμβάνουν κάποιες πληροφορίες, οι οποίες βοηθάνε τον χρήστη να καταλάβει ποιος υπάλληλος είναι ο πιο κατάλληλος για το συγκεκριμένο project. Για να προσλάβει ο χρήστης κάποιον υπάλληλο πρέπει να τον επιλέξει και να πατήσει στο κουμπί Assign to your team.

### 6.3.2 Ανάθεση εργασιών

Επόμενο βήμα μετά την πρόσληψη των υπαλλήλων, είναι η ανάθεση εργασιών. Αυτό γίνεται με το πρόγραμμα το Planner. Ανοίγοντας το Planner ο χρήστης βλέπει στην ουσία ένα Gantt chart.




Σχήμα 24: Planner

Στο Planner έχουμε τα εξής κουμπιά:

 Πρόσθεση εργασίας

 Επεξεργασία εργασίας

 Αφαίρεση επιλεγμένης εργασίας

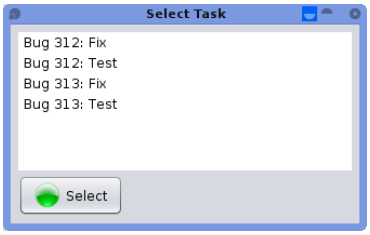
 Εμφάνιση ομάδας

 Μεγέθυνση διαγράμματος

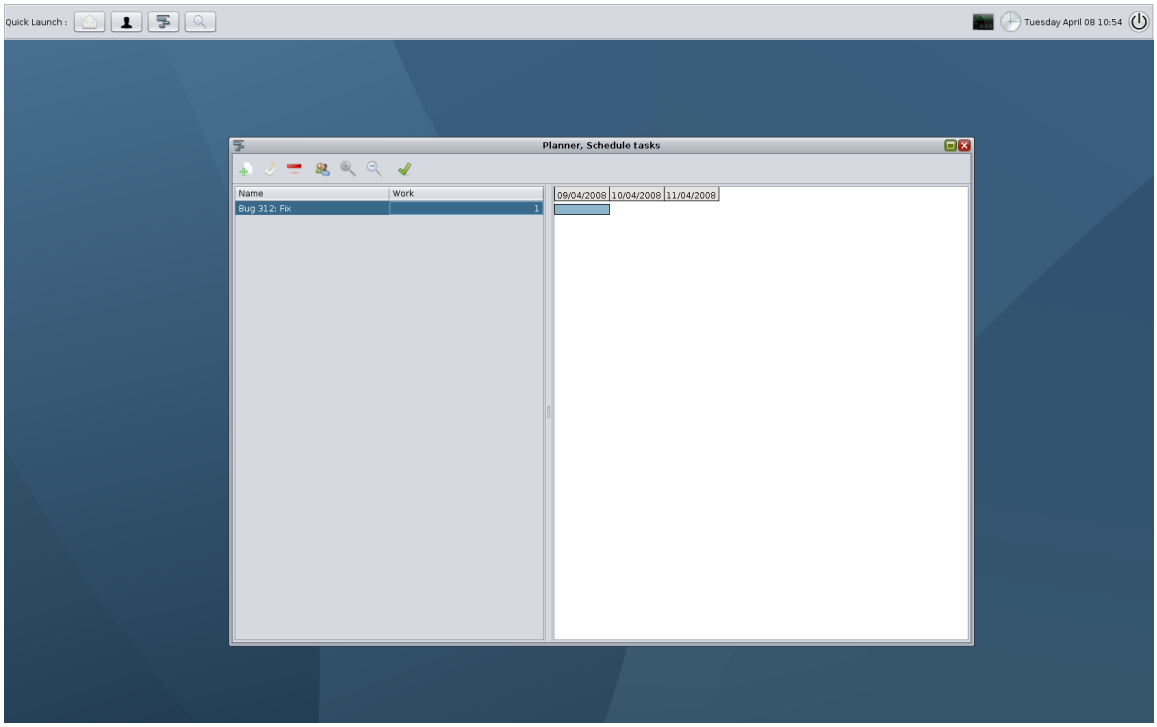
 Σμίκρυνση διαγράμματος

 Αποστολή διαγράμματος στους υπαλλήλους

Για να προσθέσει μία καινούργια προγραμματισμένη εργασία, ο χρήστης πατάει το κουμπί πρόσθεση εργασίας και εμφανίζεται μία οθόνη με τις διαθέσιμες εργασίες του project. Διαλέγει την εργασία που θέλει, πατάει το κουμπί select και εμφανίζεται στο Planner μια καινούργια προγραμματισμένη εργασία.



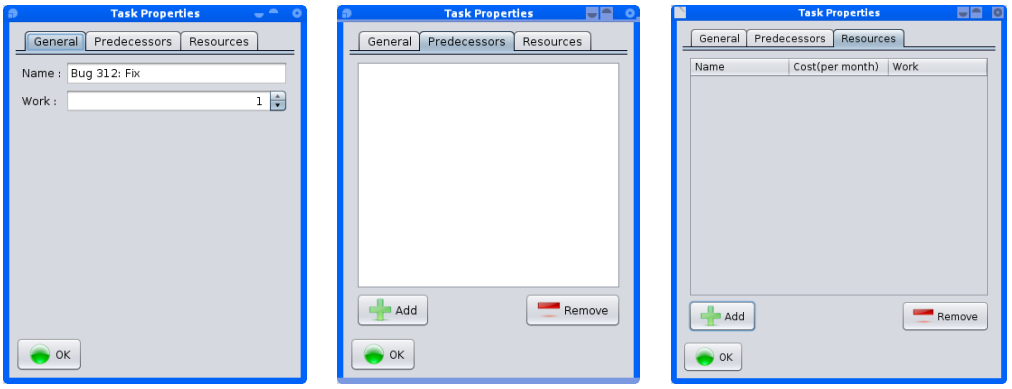

(α′) Επιλογή εργασίας




(β′) Planner μετά τη δημιουργία προγραμματισμένης εργασίας

Κάνοντας διπλό κλικ στο πεδίο work μιας προγραμματισμένες εργασίας αλλάζουμε τη διάρκεια που θα κρατήσει(μέρες). Για να επεξεργαστούμε μια προγραμματισμένη εργασία, πατάμε το κουμπί επεξεργασία εργασίας και εμφανίζεται ο Task schedule editor.

Σχήμα 25: Task schedule editor

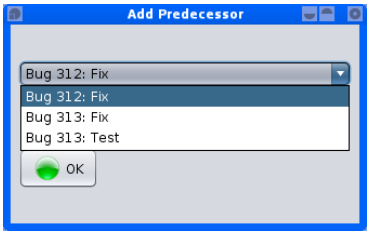



(α′) Καρτέλα General (β′) Καρτέλα Predecessors (γ′) Καρτέλα Resources

Στον Task schedule editor βλέπουμε 3 καρτέλες.

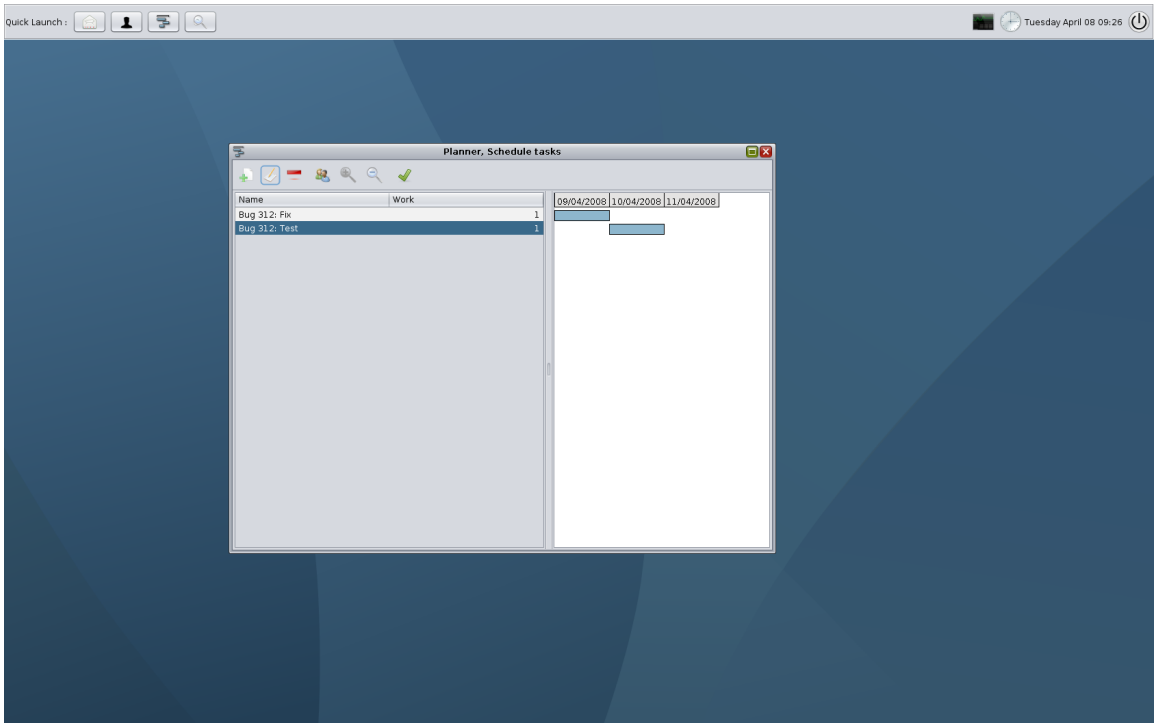
1. **General:** Στην καρτέλα αυτή βλέπουμε το όνομα της εργασίας στην οποία ανήκει η προγραμματισμένη εργασία και τον αριθμό των ημερών που θα κρατήσει. Εάν θέλουμε μπορούμε να αλλάξουμε τον αριθμό ημερών.

2. **Predecessors:** Στην καρτέλα Predecessors ο χρήστης ορίζει τις εργασίες από τις οποίες εξαρτάται η συγκεκριμένη εργασία. Πατώντας το κουμπί add εμφανίζεται μια οθόνη στην οποία ο χρήστης επιλέγει τον predecessor.




Σχήμα 26: Επιλογή predecessor

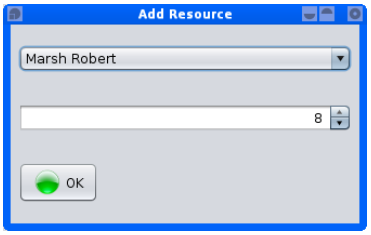
Βάζοντας έναν predecessor αλλάζει η ημερομηνία έναρξης της εργασίας. Η εργασία ξεκινάει την ημερομηνία που τελειώνουν όλοι οι predecessors. Αν αλλάξουμε την διάρκεια ενός predecessor, θα αλλάξει και η ημερομηνία της προγραμματισμένης εργασίας.




Σχήμα 27: Planner μετά την πρόσθεση ενός predecessor

Μπορούμε να αφαιρέσουμε έναν predecessor επιλέγοντας τον και πατώντας το κουμπί remove. Αφαιρώντας έναν predecessor αλλάζει και η ημερομηνία έναρξης της εργασίας. Αν η εργασία δεν έχει κανέναν predecessor, η ημερομηνία της θα είναι ίδια με αυτήν της έναρξης του project.

3. **Resources:** Από αυτήν καρτέλα ο χρήστης ορίζει την ομάδα που θα δουλέψει στην εργασία. Πατώντας το κουμπί add εμφανίζεται η οθόνη επιλογής υπαλλήλου. Εδώ ο χρήστης επιλέγει έναν υπάλληλο και ορίζει πόσες ώρες θέλει να δουλέψει στην συγκεκριμένη εργασία(1-8 ώρες) και τον προσθέτει στην ομάδα. Στην καρτέλα resources εμφανίζονται όλοι οι υπάλληλοι της ομάδας (όνομα, μισθός, ώρες που θα ασχοληθεί με την εργασία).

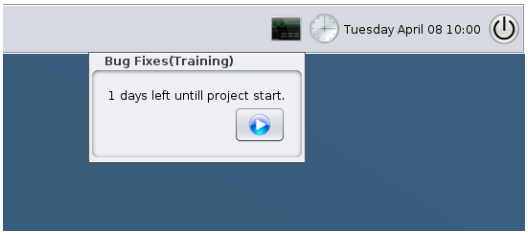



Σχήμα 28: Πρόσθεση υπαλλήλου

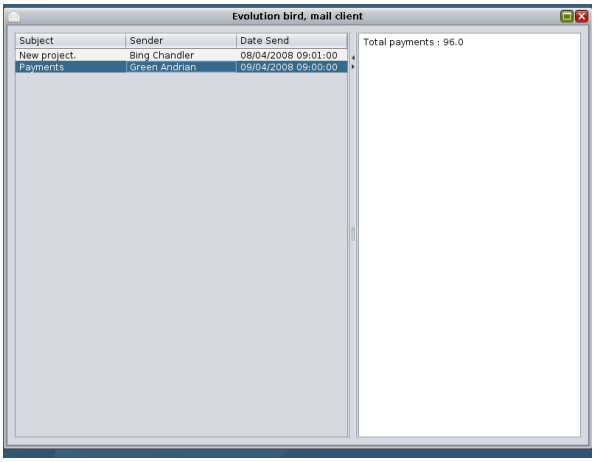
Για να αφαιρέσει ο χρήστης έναν υπάλληλο από την ομάδα, επιλέγει τον υπάλληλο και πατάει το κουμπί remove. Μόλις προσθέσει ο χρήστης όλες τις επιθυμητές εργασίες, πατάει το κουμπί αποστολή διαγράμματος στους υπαλλήλους.

### 6.3.3 Ξεκινώντας το project

Αφού ο χρήστης έχει τελειώσει με την προετοιμασία, έχει προσλάβει τους υπαλλήλους που θέλει και έχει προγραμματίσει τις εργασίες μπορεί να πατήσει το κουμπί αριστερά του ρολογιού και να πατήσει το κουμπί για την έναρξη του project. Η ημερομηνία θα αλλάξει στην ημερομηνία την οποία ξεκινάει το project. Ανάλογα και με το πόσες μέρες έχουν περάσει θα πληρωθούν και οι υπάλληλοι που έχει προσλάβει ο χρήστης. Οι πληρωμές γίνονται κάθε μέρα και ο χρήστης ενημερώνεται με ένα e-mail με θέμα Payments, το οποίο αναφέρει το συνολικό ποσό των πληρωμών.




(α′) Πληροφορίες για την έναρξη του project



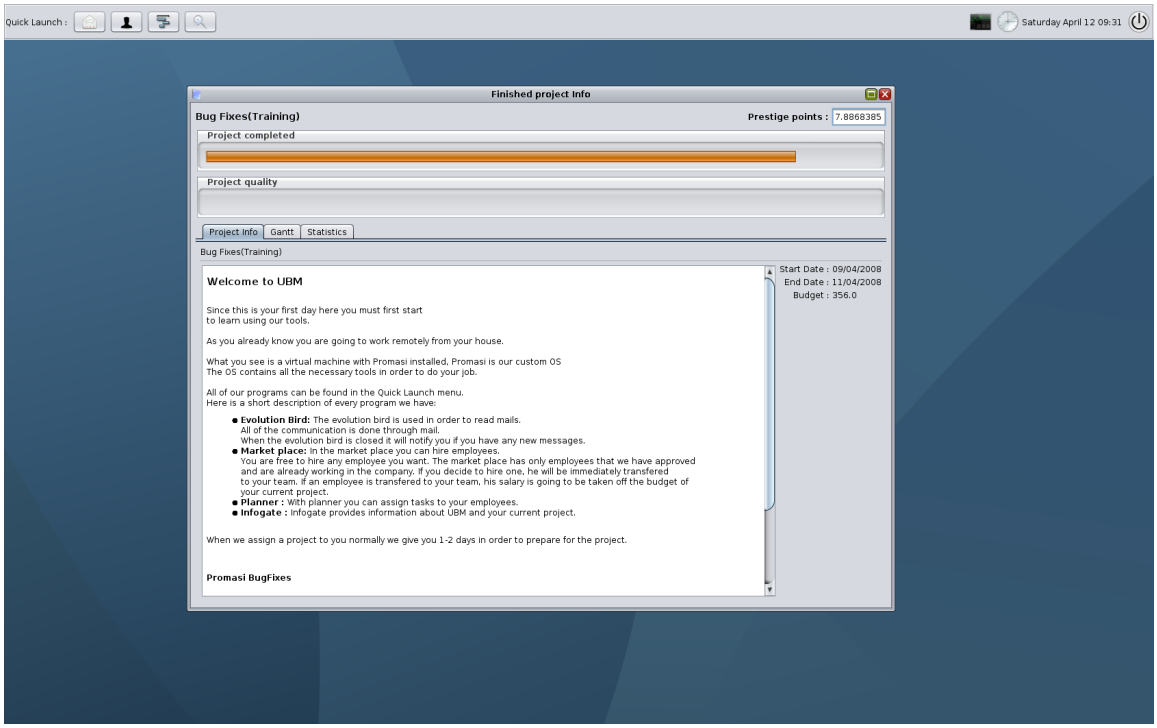

(β′) e-mail για την ενημέρωση πληρωμών

## 6.4 Παίζοντας

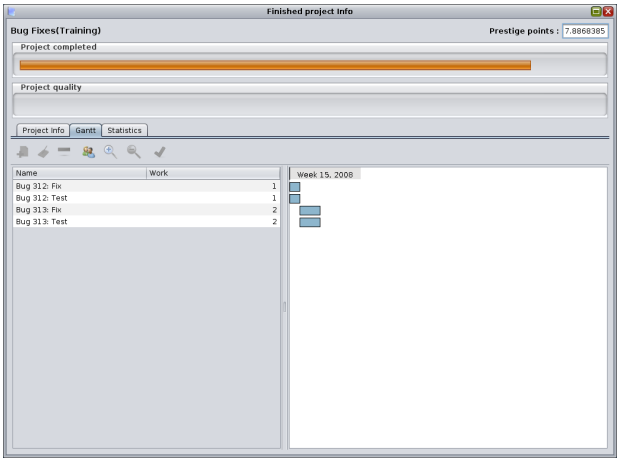
Κατά τη διάρκεια του παιχνιδιού, ο χρήστης θα λαμβάνει διάφορα e-mail σχετικά με την πρόοδο του project ή με κάποια ξαφνικά γεγονότα, όπως η αλλαγή απαιτήσεων. Συνήθως για κάθε τέτοιο e-mail, ο χρήστης θα κάνει κάποια αλλαγή στις προγραμματισμένες εργασίες ή θα προσλαμβάνει και άλλους υπαλλήλους. Ο χρήστης θα βάζει το χρόνο στο fast και όταν λαμβάνει κάποιο e-mail θα αλλάζει την ταχύτητα του χρόνου στο slow προκειμένου να σκεφτεί τι θα κάνει. Για να κάνει κάποια αλλαγή στις προγραμματισμένες εργασίες ανοίγει το Planner, αλλάζει τις ιδιότητες των εργασιών και πατάει το κουμπί αποστολή διαγράμματος στους υπαλλήλους.

## 6.5 Τελειώνοντας το project

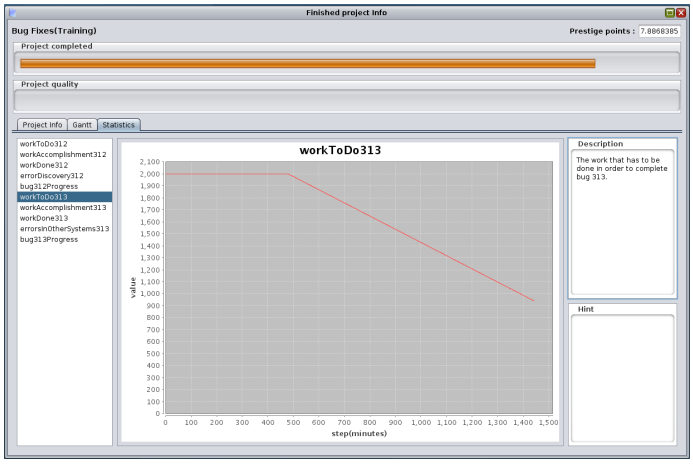
Το project τελειώνει είτε όταν τελειώσει η προθεσμία του είτε όταν τελειώσουν όλες οι εργασίες του. Μόλις τελειώσει, εμφανίζεται στον χρήστη μια οθόνη με τους πόντους που πήρε (prestige points), την ποιότητα που είχε το project και το ποσοστό επί τοις εκατό που ολοκληρώθηκε. Ο χρήστης μπορεί να δει την περιγραφή του project (καρτέλα Project Info), το Gantt διάγραμμα που χρησιμοποίησε (καρτέλα Gantt) καθώς και κάποιες γραφικές παραστάσεις με τις τιμές μεταβλητών του PSD μοντέλου (καρτέλα Statistics). Όταν όλα τα project της ιστορίας τελειώσουν εμφανίζεται μία καινούργια καρτέλα η οποία δείχνει όλη τη βαθμολογία. Σε αυτήν την περίπτωση αν κλείσουμε το παράθυρο κλείνει και η εφαρμογή.



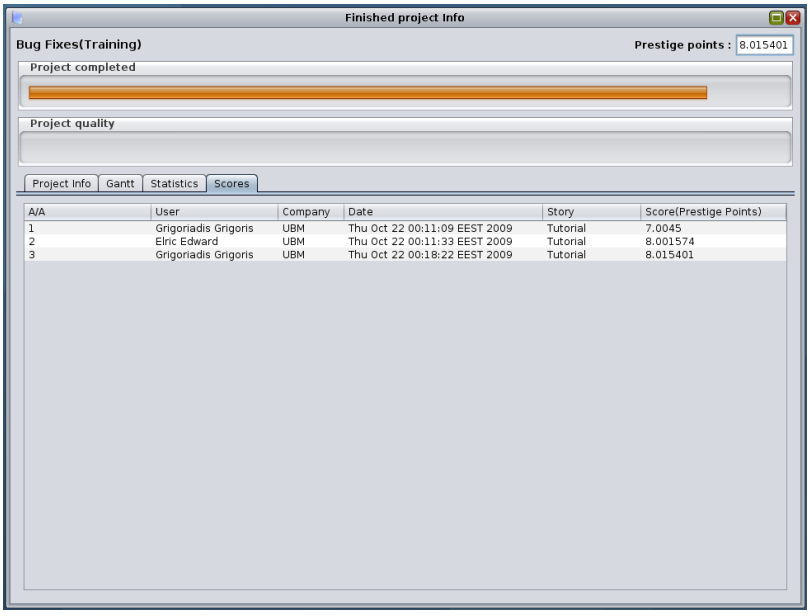

Σχήμα 29: Καρτέλα Project Info




(α′) Καρτέλα Gantt




(β′) Καρτέλα Statistics

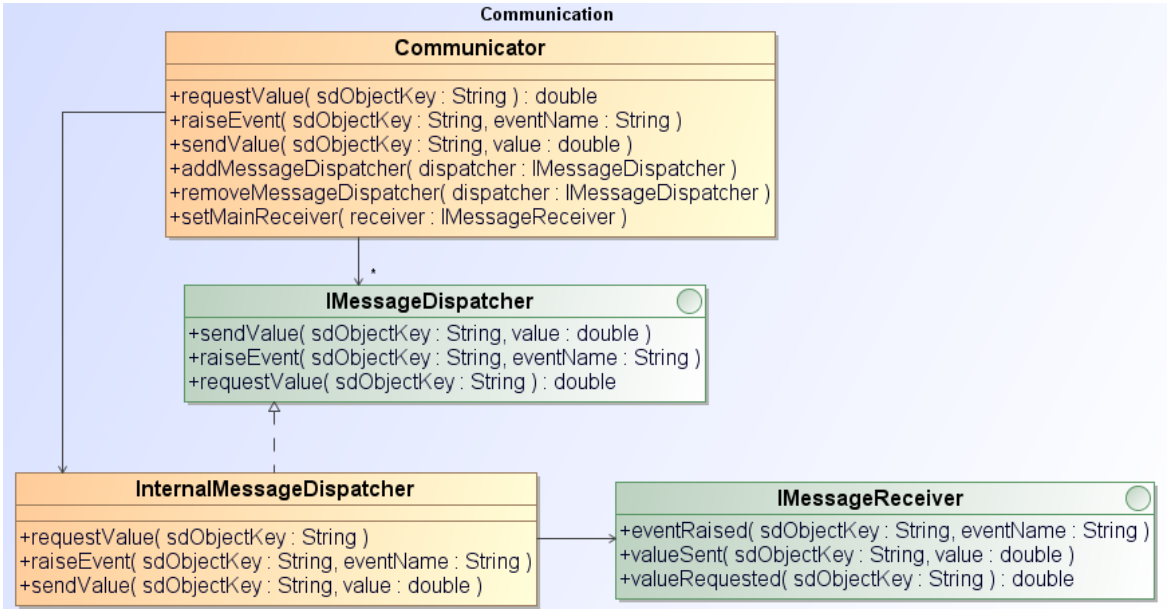



(γ′) Καρτέλα Scores

# 7 Υλοποίηση ProMaSi

Στο κεφάλαιο αυτό παρουσιάζουμε κάποιες λεπτομέρειες της υλοποίησης του ProMaSi .

## 7.1 Υλοποίηση Communication επιπέδου




Σχήμα 30: Communication class diagram

Στο παραπάνω διάγραμμα βλέπουμε την αρχιτεκτονική του Communication επιπέδου σε επίπεδο σχεδίασης κώδικα. Το Communication επίπεδο έχουμε τις εξής κλάσεις:

- **Communicator:** Singleton κλάση η οποία χρησιμοποιείται από το Core επίπεδο, προκειμένου να στείλει τα μηνύματα στο Shell. Ο Communicator υποστηρίζει πολλούς IMessageDispatcher's εκ των οποίων ο ένας είναι ο βασικός. Ο βασικός IMessageDispatcher χρησιμοποιείται για την αποστολή των μηνυμάτων στο Shell, ενώ οι υπόλοιποι χρησιμοποιούνται για την αποστολή των 'Ενημέρωση τιμής' και 'Γεγονός' μηνυμάτων προκειμένου να ενημερώσουν κάποιο άλλο σύστημα.

- **IMessageDispatcher:** Interface το οποίο χρησιμοποιείται από τον Communicator για να στείλει τα γεγονότα σε κάποιο άλλο επίπεδο. Παρέχει 3 μεθόδους μία για κάθε μήνυμα:

  **requestValue(sdPbjectKey:String):double** Στέλνει το 'Ζήτηση τιμής' μήνυμα με το κλειδί του sdObject και επιστρέφει την τιμή για αυτό το sdObject από το άλλο επίπεδο.

**raiseEvent(sdPbjectKey:String,eventName:String)** Στέλνει το Γεγονός΄ μήνυμα με το κλειδί του sdObject που ενεργοποίησε το γεγονός και το όνομα του γεγονότος.

**sendValue(sdPbjectKey:String,value:double)** Στέλνει το Ένημέρωση τιμής΄ μήνυμα με το κλειδί του sdObject και την τιμή του.

Μία υλοποίηση του IMessageDispatcher είναι το InternalMessageDispatcher το οποίο δέχεται ένα IMessageReceiver. Το IMessageReceiver είναι ένα Interface το οποίο χρησιμοποιείται για τη διαχείριση των μηνυμάτων του Core. Ο InternalMessageDispatcher λειτουργεί σαν Strategy στο IMessageReceiver. Το InternalMessageDispatcher είναι ο βασικός IMessageDispatcher στον Communicator.

Μπορούμε να αναθέσουμε έναν IMessageReceiver στον Communicator γράφοντας τον εξής κώδικα:

```
Communicator.getInstance().setMainReceiver( new TestMessageReceiver() );
```

Μπορούμε να προσθέσουμε έναν επιπλέον IMessageDispatcher γράφοντας τον εξής κώδικα:

```
Communicator.getInstance().addAdditionalDispatcher( new TcpMessageDispatcher() );
```

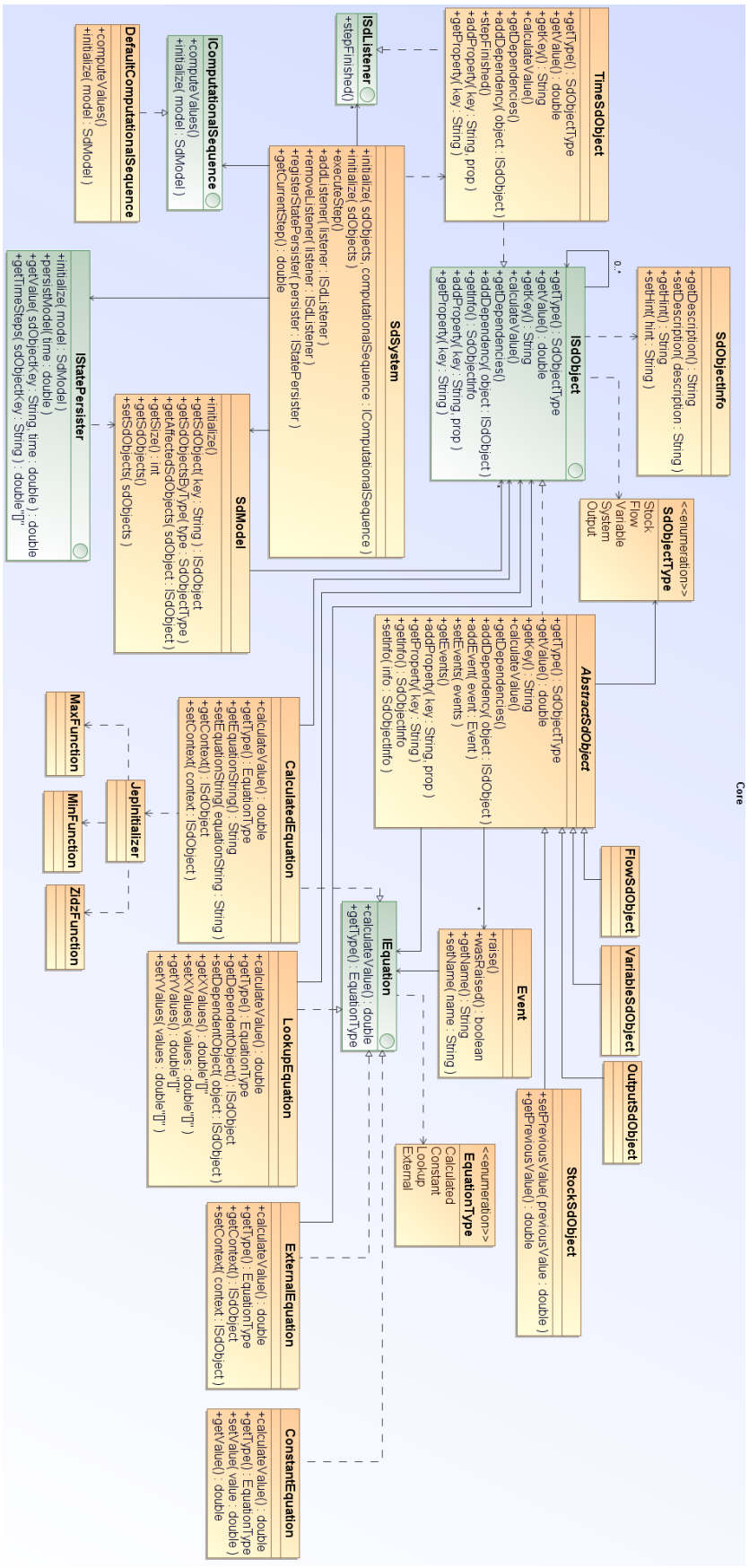
Μπορούμε να καλέσουμε το Ένημέρωση τιμής΄ μήνυμα γράφοντας τον εξής κώδικα:

```
Communicator.getInstance().raiseEvent( "productivity", "lowProductivity" );
```

Η παραπάνω γραμμή κώδικα έχει την εξής ακολουθία:


lowProductivityEvent : Event
Communicator : Communicator
additionalDispatcher : IMessageDispatcher
basicDispatcher : InternalMessageDispatcher
mainReceiver : IMessageReceiver
1: raiseEvent(sdObjectKey="productivity", eventName="lowProductivity")
2: raiseEvent(sdObjectKey="productivity", eventName="lowProductivity")
3: eventRaised(sdObjectKey="productivity", eventName="lowProductivity")
4:
5:
6: ()
For all additional dispatchers
7: raiseEvent(sdObjectKey="productivity", eventName="lowProductivity")
8:
9:

## 7.2 Υλοποίηση Core επιπέδου



Στο παραπάνω διάγραμμα βλέπουμε την αρχιτεκτονική του Core επιπέδου σε επίπεδο σχεδίασης κώδικα. Το Core επίπεδο έχουμε τις εξής κλάσεις:

- **ISdObject:** Interface που αντιπροσωπεύει ένα βασικό sdObject το οποίο έχει τα εξής χαρακτηριστικά:

  1. Ένα κλειδί(όνομα) το οποίο είναι μοναδικό σε όλο το μοντέλο.
  2. Έναν τύπο(SdObjectType).
  3. Μία τιμή η οποία υπολογίζεται με συγκεκριμένο τρόπο(calculateValue()).
  4. Μια λίστα με ISdObject από τα οποία εξαρτάται προκειμένου να υπολογιστεί η τιμή του.
  5. Ένα SdObjectInfo το οποίο περιέχει μια φιλική περιγραφή(info) και ένα hint για το πως μπορεί ο χρήστης να βελτιώσει την τιμή του.

- **IEquation:** Interface το οποίο αντιπροσωπεύει μία συνάρτηση(Constant, Calculated, Lookup, External).

- **Event:** Κλάση η οποία αντιπροσωπεύει ένα γεγονός. Έχει ένα όνομα, ένα IEquation και το ISdObject στο οποίο ανήκει. Η μέθοδος raise() ειδοποιεί το Communication επίπεδο στην περίπτωση που το IEquation.calculateValue() επιστρέψει τιμή >= 1.0. Η raise() υλοποιείται ως εξής:

```
Double value = _equation.calculateValue( );
value = MathUtils.round( value, 2 );
if ( value >= 1.0d )
{
    Communicator.getInstance( ).raiseEvent( _context.getKey( ), _name );
    _wasRaised = true;
}
```

  Μετά τη raise() μπορούμε να φωνάξουμε την μέθοδο wasRaised() προκειμένου να δούμε αν το γεγονός ενεργοποιήθηκε.

- **AbstractSdObject:** Abstract κλάση η οποία αντιπροσωπεύει ένα από τα βασικά sdObjects του Core(Flow,Variable,Stock,Output). Δέχεται ένα η περισσότερα γεγονότα(Events) και ένα IEquation προκειμένου να υπολογιστεί η τιμή του. Ο υπολογισμός της τιμής του γίνεται ως εξής:

```
_value = MathUtils.round( _equation.calculateValue( ), 7 );
if ( _value.isInfinite( ) || _value.isNaN( ) )
{
    _value = 0.0;
}
```

```
Vector<Event> clonedEvents = new Vector<Event>( _events );
for ( Event event: clonedEvents )
{
    event.raise( );
    if ( event.wasRaised( ) )
    {
        _events.remove( event );
    }
}
```

Τα γεγονότα ενεργοποιούνται μόνο μία φορά.

- **FlowSdObject, VariableSdObject:** Δεν έχουν καμία διαφοροποίηση από το AbstractSdObject εκτός από τον τύπο(Flow και Variable αντίστοιχα).
- **StockSdObject:** Κρατάει την προηγούμενη τιμή όταν γίνεται ο υπολογισμός της και σαν καινούργια τιμή προσθέτει την προηγούμενη και την καινούργια τιμή. Η calculateValue() έχει υλοποιηθεί ως εξής:

```
_previousValue = getValue( );
super.calculateValue( );
setValue( _previousValue + getValue( ) );
```

- **OutputSdObject:** Μόλις υπολογιστεί η τιμή του στέλνει μήνυμα στο Communication επίπεδο. Η calculateValue() έχει υλοποιηθεί ως εξής:

```
super.calculateValue( );
Communicator.getInstance( ).sendValue( getKey( ), getValue( ) );
```

- **ConstantEquation:** Αντιπροσωπεύει την Constant συνάρτηση, παίρνει μια τιμή και η calculateValue() επιστρέφει πάντα αυτήν την τιμή.
- **ExternalEquation:** Αντιπροσωπεύει την External συνάρτηση, παίρνει το ISdObject στο οποίο ανήκει και φωνάζει το Communication επίπεδο προκειμένου να πάρει την τιμή της. Η calculateValue() έχει υλοποιηθεί ως εξής:

```
return Communicator.getInstance( ).requestValue( _context.getKey( ) );
```

- **LookupEquation:** Αντιπροσωπεύει την Lookup συνάρτηση, παίρνει μια λίστα από σημεία $x, y$ και ένα ISdObject σύμφωνα με το οποίο θα υπολογιστεί η τιμή της. Η calculateValue() έχει υλοποιηθεί ως εξής:

```
UnivariateRealInterpolator interpolator = new SplineInterpolator( );
UnivariateRealFunction function =
        interpolator.interpolate( _xValues, _yValues );
return function.value( _dependentObject.getValue( ) );
```

- **CalculatedEquation:** Αντιπροσωπεύει την Calculated συνάρτηση, παίρνει ένα string που αντιπροσωπεύει την εξίσωση π.χ. ((var1+var2)/2)-sin(var3) και το ISdObject στο οποίο ανήκει. Η calculateValue() έχει υλοποιηθεί ως εξής:

```
// initialize the JEP object and parse the expression.
JEP jep = JepInitializer.getFullJep( );
jep.parseExpression( _equationString );
// Get a list with all the undeclared variables. Those variable are
// one of the dependency objects of the _context.
SymbolTable symTable = jep.getSymbolTable( );
Collection coll = symTable.values( );
// Get the value of each variable.
for ( Object object : coll )
{
    Variable var = (Variable) object;
    // Make sure that the var is not a JEP constant.
    if ( !var.isConstant( ) )
    {
        ISdObject sdObject = getSdObjectFromContext( var.getName( ) );
        Double value = sdObject.getValue( );
        jep.setVarValue( var.getName( ), value );
    }
}
return jep.getValue( );
```

- **SdModel:** Αντιπροσωπεύει ένα PSD μοντέλο. Το SdModel περιέχει όλα τα sdObjects ενός PSD μοντέλου.

- **IStatePersister:** Interface που χρησιμοποιείται για να σώσει τις τιμές όλων των sdObject του μοντέλου σε κάθε βήμα. Το Interface παρέχει τις εξής μεθόδους:

  **initialize(SdModel model)** Αρχικοποιεί τον StatePersister με το μοντέλο το οποίο θα σώζει.

  **persistModel(double time)** Σώζει τις τιμές όλων των sdObject του μοντέλου με το οποίο αρχικοποιήθηκε.

  **getValue(String sdObjectKey,double time)** Επιστρέφει την τιμή του sdObject με το συγκεκριμένο κλειδί για το συγκεκριμένο χρόνο(βήμα).

  **getTimeSteps(String sdObjectKey)** Παίρνει μια λίστα με όλα τα βήματα τα οποία έχουν σωθεί.

- **IComputationalSequence:** Interface το οποίο παρέχει μεθόδους για τον υπολογισμό των sdObjects ενός SdModel.

**initialize(SdModel model)** Αρχικοποιεί την ComputationalSequence με το μοντέλο για το οποίο θα υπολογίζει τις τιμές των sdObjects.

**computeValues()** Μέθοδος η οποία υπολογίζει τις τιμές όλων των sdObjects του μοντέλου με το οποίο αρχικοποιήθηκε.

- **DefaultComputationalSequence:** Υπολογίζει τις τιμές των sdObjects όπως αναλύεται στο Κεφάλαιο4.1.1. Η computeValues() υλοποιείται ως εξής:

```
public void computeValues ( )
{
  _valuesComputedForCurrentStep.clear( );
  // Compute the values of variable types
  List<ISdObject> variables =
      _model.getSdObjectsByType( SdObjectType.Variable );
  for ( ISdObject variable : variables )
  {
      recursivelyComputeValue( variable );
  }

  // Compute the values of flow types
  List<ISdObject> flows =
      _model.getSdObjectsByType( SdObjectType.Flow );
  for ( ISdObject flow : flows )
  {
      recursivelyComputeValue( flow );
  }

  // Compute the values of output types
  List<ISdObject> outputs =
      _model.getSdObjectsByType( SdObjectType.Output );
  for ( ISdObject output : outputs )
  {
      recursivelyComputeValue( output );
  }

  // Compute the values of stock types for the next step.
  List<ISdObject> stocks =
      _model.getSdObjectsByType( SdObjectType.Stock );
  for ( ISdObject stock : stocks )
  {
      stock.calculateValue( );
  }
}

private void recursivelyComputeValue ( ISdObject sdObject )
```

```
{
    // If the object is in the list its value
    // has been calculated for this step so skip it.
    if ( !_valuesComputedForCurrentStep.contains( sdObject ) )
    {
        // Calculate the values of the dependencies by
        // recursively calling the method.
        List<ISdObject> dependencies = sdObject.getDependencies( );
        for ( ISdObject dependencyObject : dependencies )
        {
            // If the SdObjectType is a Stock then there is
            // no need to calculate its value. Stock values
            // are calculated one step ahead.
            if ( !dependencyObject.getType( )
                    .equals( SdObjectType.Stock ) )
            {
                recursivelyComputeValue( dependencyObject );
            }
        }
        // Calculate the value of the object since
        // all dependencies have been calculated.
        sdObject.calculateValue( );
        // Add it to the list so it won't be calculated
        // again for this step.
        _valuesComputedForCurrentStep.add( sdObject );
    }
}
```

- **ISdListener:** Interface το οποίο παρέχει μια μέθοδο η οποία καλείτε όταν τελειώνει ένα βήμα του μοντέλου.

- **TimeSdObject:** Είναι ένα ISdObject το οποίο υλοποιεί το ISdListener interface. Είναι ένα System sdObject το οποίο κρατάει τα βήματα τα οποία έχουν περάσει από τότε που ξεκίνησε το σύστημα.

- **SdSystem:** Υπεύθυνο για την εκτέλεση ενός SdModel. Έχει τις εξής μεθόδους:

  **initialize(List<ISdObject> sdObjects)** Φωνάζει την initialize(List<ISdObject> sdObjects,IComputationalSequence computationalSequence) με την DefaultComputationalSequence.

  **initialize(List<ISdObject> sdObjects,IComputationalSequence computationalSequence)** Φτιάχνει ένα καινούργιο SdModel και αρχικοποιεί την computationalSequence με το καινούργιο SdModel. Η υλοποίησή της είναι η εξής:

```
public void initialize ( List<ISdObject> sdObjects,
  IComputationalSequence computationalSequence )
{
  normalizeSdObjects( sdObjects );
  _model.setSdObjects( sdObjects );
  _model.initialize( );
  _computationalSequence = computationalSequence;
  _computationalSequence.initialize( _model );
}

private void normalizeSdObjects ( List<ISdObject> sdObjects )
{
    List<ISdObject> copiedSdObjects =
              new Vector<ISdObject>( sdObjects );
    Collections.copy( copiedSdObjects, sdObjects );

    // Add the TimeSdObject and register it as a listener.
    LOGGER.info( "Registering system sdObjects" );
    TimeSdObject time = new TimeSdObject( );
    sdObjects.add( time );
    addListener( time );

    // Check for invalid keys.
    //And add the system variables as dependencies.
    for ( ISdObject sdObject : copiedSdObjects )
    {
        sdObject.addDependency( time );
        if ( sdObject.getKey( ).equals( TimeSdObject.KEY ) )
        {
            sdObjects.remove( sdObject );
        }
    }
}
```

Η normalizeSdObjects προσθέτει όλα τα System sdObjects στο μοντέλο.

**executeStep()** Υπολογίζει τις τιμές των sdObjects, φωνάζει τον IStatePersister και όλους τους ISdListeners. Η υλοποίησή της είναι η εξής:

```
_computationalSequence.computeValues( );
// Persist the model for the current step.
_persister.persistModel( getCurrentStep( ) );
// Call the listeners.
for ( ISdListener listener : _listeners )
{
    listener.stepFinished( );
}
```

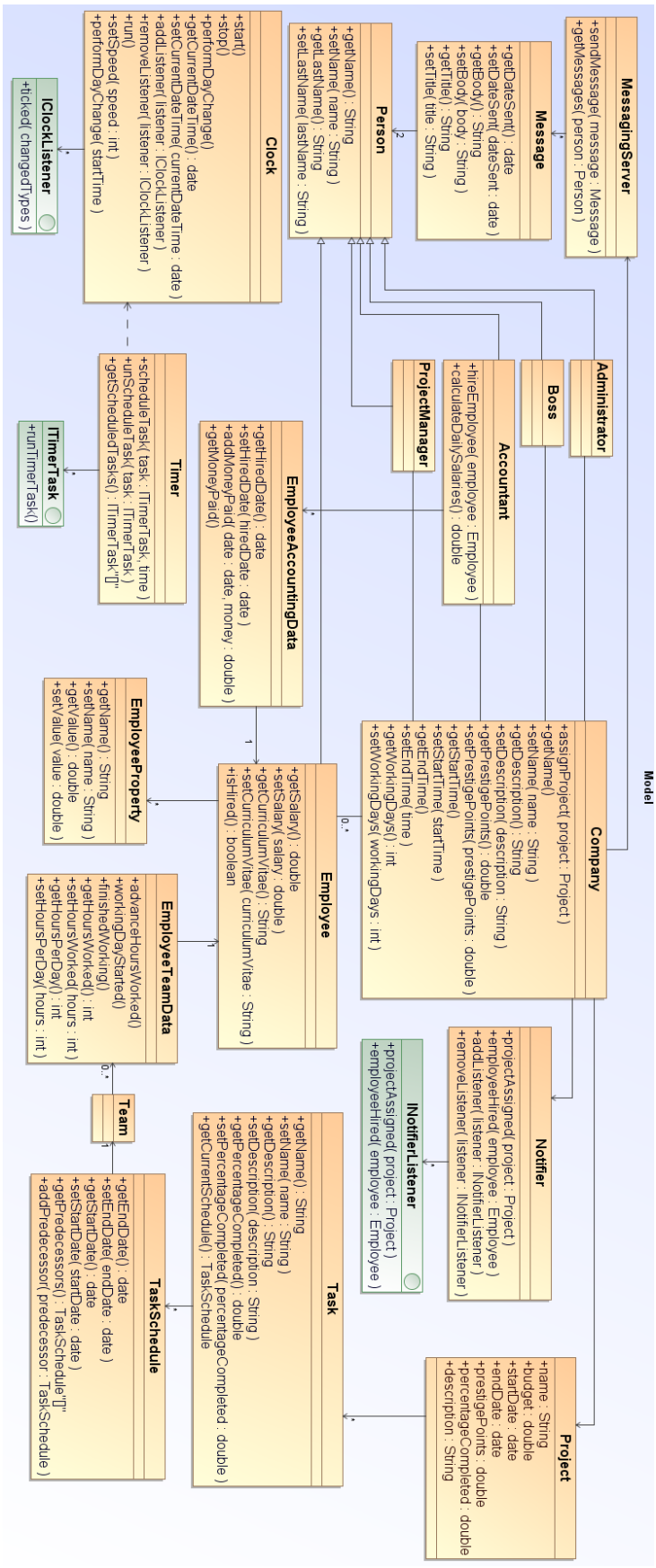
**addListener(ISdListener listener)** Προσθέτει έναν ISdListener.

**removeListener(ISdListener listener)** Αφαιρεί έναν ISdListener.

**registerStatePersister(IStatePersister persister)** Αναθέτει έναν IStatePersister.

**getCurrentStep()** Επιστρέφει την τιμή της TimeSdObject.

## 7.3 Υλοποίηση Model επιπέδου



Στο παραπάνω διάγραμμα βλέπουμε την αρχιτεκτονική του Model επιπέδου σε επίπεδο σχεδίασης κώδικα. Οι περισσότερες κλάσεις αντιπροσωπεύουν οντότητες όπως Company, project κ.τ.λ. Η κλάσεις οι οποίες δεν αντιπροσωπεύουν κάποια οντότητα είναι οι εξής:

- **MessagingServer:** Κλάση η οποία χρησιμοποιείται για να στέλνει μηνύματα (Message). Στον Project Manager μπορούμε να στείλουμε κάποιο μήνυμα γράφοντας τον εξής κώδικα:

  ```
  Message message = new Message( );
  message.setSender( company.getBoss( ) );
  message.setRecipient( company.getProjectManager( ) );
  message.setTitle( "Test message" );
  message.setBody( "Sample message" );
  company.getMessagingServer( ).sendMessage( message );
  ```

  Στη συνέχεια για να πάρουμε το μήνυμα θα γράψουμε τον εξής κώδικα:

  ```
  List<Message> messages = company.getMessagingServer( )
      .getMessages( company.getProjectManager( ) );
  ```

- **Clock:** Singleton κλάση η οποία αντιπροσωπεύει το ρολόι του ProMaSi . Αυτή η κλάση μας δίνει διάφορες μεθόδους για το χειρισμό της ώρας του ProMaSi . Σε κάθε τικ ειδοποιεί όλους τους IClockListener's. Για να αλλάξουμε την ταχύτητα της ώρας θα γράψουμε τον εξής κώδικα:

  ```
  Clock.getInstance( ).setSpeed( 3000 );
  ```

  Αν βάλουμε την ταχύτητα στο 1000 σημαίνει ότι ένα λεπτό θα αντιστοιχεί σε ένα πραγματικό δευτερόλεπτο.

- **IClockListener:** Interface το οποιό καλείται σε κάθε τικ του Clock η μέθοδος ticked παίρνει και μια λίστα με όλους τους τύπους που έχουν αλλάξει, αν άλλαξε η ώρα, μέρα, μήνας, χρόνος κ.τ.λ μπορούμε να ελένξουμε τι έχει αλλάξει γράφοντας τον εξής κώδικα:

  ```
  if ( changedTypes.contains( DurationFieldType.days( ) ) )
  {
    //Do something
  }
  else if ( changedTypes.contains( DurationFieldType.hours( ) ) )
  {
    //Do something
  }
  ```

- **Notifier:** Κλάση η οποία είναι υπεύθυνη να ειδοποιεί όλους τους INotifierListener's για διάφορα γεγονότα. Αυτή τη στιγμή υποστηρίζει 3 γεγονότα:

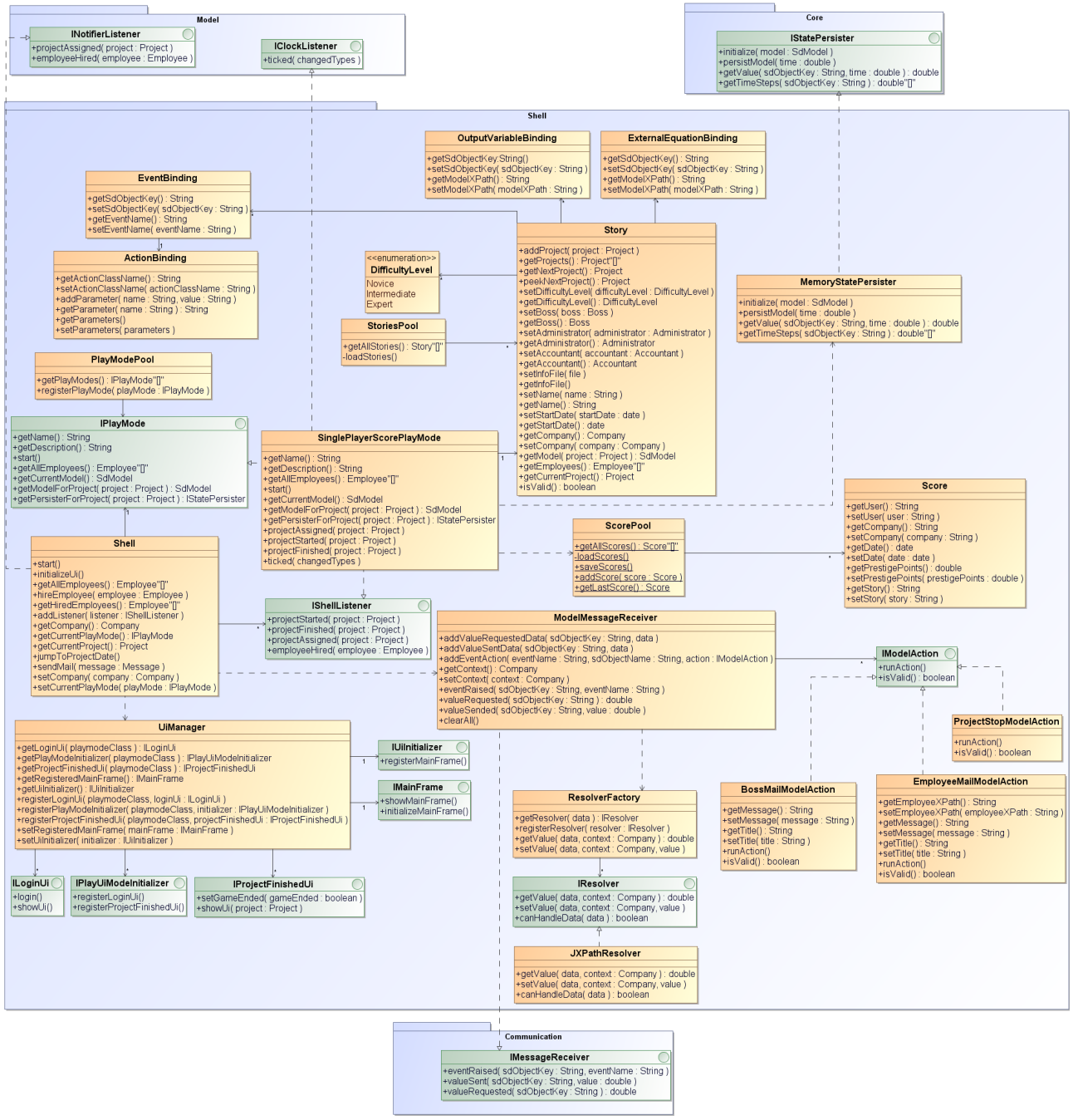
  1. Ανάθεση ενός project.

2. Πρόσληψη ενός υπαλλήλου.

- **Timer:** Singleton κλάση η οποία χρησιμοποιείται για τον προγραμματίσμο ITimerTask σε κάποια συγκεκριμένη ημερομηνία. Μπορούμε να προγραμματίσουμε ένα ITimerTask σε μια συγκεκριμένη ημερομηνία γράφοντας τον εξής κώδικα:

```
Timer.getInstance( ).scheduleTask(
  new ProjectEndTimerTask( project ),
    project.getEndDate( ).toLocalDateTime( ) );
```

## 7.4 Υλοποίηση Shell επιπέδου




Στο παραπάνω διάγραμμα βλέπουμε την αρχιτεκτονική του Shell επιπέδου σε επίπεδο σχεδίασης κώδικα. Το Shell επίπεδο έχουμε τις εξής κλάσεις:

- **IPlayMode:** Interface το οποίο ορίζει ένα play mode. Το play mode έχει ένα όνομα, μια περιγραφή, μια λίστα με όλους τους διαθέσιμους υπαλλήλους στο σύστημα.
- **PlayModePool:** Singleton κλάση η οποία έχει όλα τα διαθέσιμα play modes.

- **UiManager:** Περιέχει όλα τα UI components.
- **IMainFrame:** Interface που αντιπροσωπεύει την κεντρική οθόνη ενός UI.
- **IUiInitializer:** Interface που είναι υπεύθυνο για να καταχωρεί στον UiManager όλα τα components του συγκεκριμένου UI.
- **IProjectFinishedUi:** Interface για το UI που εμφανίζεται όταν τελειώνει ένα project. Υπεύθυνο για να εμφανίζει τα στατιστικά του project.
- **IPlayUiModeInitializer:** Interface που είναι υπεύθυνο για να καταχωρεί στον UiManager όλα τα components του συγκεκριμένου UI για ένα συγκεκριμένο IPlayMode.
- **ILoginUi:** Interface που αντιπροσωπεύει την Login οθόνη ενός IPlayMode για ένα συγκεκριμένο UI. Η οθόνη αυτή είναι υπεύθυνη για την εισαγωγή στοιχείων του χρήστη.
- **IModelAction:** Interface που αντιπροσωπεύει ένα Action το οποίο γίνεται σε κάποιο γεγονός του Core.
- **ProjectStopModelAction:** Αντιπροσωπεύει το Project Stop Action όπως περιγράφτηκε στο Κεφάλαιο 4.4.
- **EmployeeMailModelAction:** Αντιπροσωπεύει το Employee Mail Action όπως περιγράφτηκε στο Κεφάλαιο 4.4.
- **BossMailModelAction:** Αντιπροσωπεύει το Boss Mail Action όπως περιγράφτηκε στο Κεφάλαιο 4.4.
- **IResolver:** Interface το οποίο χρησιμοποιείται για τον χειρισμό των 'Ζήτηση τιμής' και 'Ενημέρωση τιμής' μηνυμάτων. Τα data που δέχεται η κάθε μέθοδος χρησιμοποιούνται για την εύρεση του αντικειμένου από το οποίο θα πάρουμε ή θα βάλουμε την τιμή. Η κάθε υλοποίηση χειρίζεται και data διαφορετικού τύπου.
- **JXPathResolver:** Ένας IResolver ο οποίος χειρίζεται modelXPath σαν data. Η getValue και setValue έχουν υλοποιηθεί ως εξής:

```
public Double getValue ( Object data, Company context )
{
    return (Double) JXPathContext.newContext( context )
        .getValue( data.toString( ) );
}

public void setValue ( Object data, Company context, Object value )
{
```

```
        JXPathContext.newContext( context )
            .setValue( data.toString( ), value );
    }
```

- **ResolverFactory:** Factory pattern στο οποίο καταχωρούμε τους διαθέσιμους IResolver's και φωνάζει αυτόν που μπορεί να διαχειριστεί τα data που του περνάμε στις μεθόδους getValue και setValue.

- **ModelMessageReceiver:** Singleton κλάση η οποία υλοποιεί το IMessageReceiver του Communication επιπέδου. Χρησιμοποιεί το ResolverFactory για την διαχείριση των μηνυμάτων 'Ζήτηση τιμής' και 'Ενημέρωση τιμής' και τα IModelAction για την διαχείριση των γεγονότων του Core.

- **MemoryStatePersister:** Υλοποιεί το IStatePersister του Core επιπέδου. Χρησιμοποιεί Maps για να σώσει τις τιμές των sdObject's σε κάθε βήμα.

- **IShellListener:** Interface το οποίο μπορεί να γίνει register στο Shell και να χειρίζεται τα εξής γεγονότα:
  - Ανάθεση project στον χρήστη.
  - Εκκίνηση project.
  - Τελείωμα project.
  - Πρόσληψη υπαλλήλου.

- **Shell:** Singleton κλάση, η οποία υλοποιεί το INotifierListener του Model επιπέδου. Παρέχει μεθόδους για εύκολη διαχείριση όπως η hireEmployee,jumpToProjectDate κ.τ.λ, είναι υπεύθυνη να ειδοποιεί όλους τους IShellListener's καθώς και για την εκκίνηση του παιχνιδιού εφόσον έχει καθοριστεί το IPlayMode και η εταιρεία. Η start μέθοδος πρέπει να καλείται εφόσον έχουμε βάλει κάποιο IPlayMode, φωνάζοντας την setCurrentPlayMode και την εταιρεία, φωνάζοντας την setCompany. Η υλοποίησή της είναι η εξής:

```
Communicator.getInstance( ).setMainReceiver(
            ModelMessageReceiver.getInstance( ) );
IMainFrame mainFrame = UiManager.getInstance( ).getRegisteredMainFrame( );
mainFrame.initializeMainFrame( );
mainFrame.showMainFrame( );
_currentPlayMode.start( );
```

Οι παρακάτω κλάσεις έχουν αναπτυχθεί για την υλοποίηση του Single player score mode:

- **ActionBinding:** Κλάση η οποία αντιπροσωπεύει τη δομή του actionBinding element από το coreModelBindings.xml όπως περιγράφτηκε στο Κεφάλαιο 5.

```
<actionBinding>
  <actionClassName>org.promasi.shell.model.actions
    .BossMailModelAction</actionClassName>
  <parameter name="message">I seems that you have
    finished with bug 312.</parameter>
  <parameter name="title">Bug 312 finished!</parameter>
</actionBinding>
```

- **EventBinding:** Κλάση η οποία αντιπροσωπεύει τη δομή του Event element από το coreModelBindings.xml όπως περιγράφτηκε στο Κεφάλαιο 5.

```
<Event>
  <sdObjectKey>bug312Progress</sdObjectKey>
  <eventName>bug312Finished</eventName>
  <actionBinding>
  <actionClassName>org.promasi.shell.model.actions
    .BossMailModelAction</actionClassName>
  <parameter name="message">I seems that you have
    finished with bug 312.</parameter>
  <parameter name="title">Bug 312 finished!</parameter>
  </actionBinding>
</Event>
```

- **OutputVariableBinding:** Κλάση η οποία αντιπροσωπεύει τη δομή του OutputVariable element από το coreModelBindings.xml όπως περιγράφτηκε στο Κεφάλαιο 5.

```
<OutputVariable>
  <sdObjectKey>bug312Progress</sdObjectKey>
  <modelXPath>currentProject/tasks[name='Bug␣312:␣Fix']
    /percentageCompleted</modelXPath>
</OutputVariable>
```

- **ExternalEquationBinding:** Κλάση η οποία αντιπροσωπεύει τη δομή του ExternalEquation element από το coreModelBindings.xml όπως περιγράφτηκε στο Κεφάλαιο 5.

```
<ExternalEquation>
  <sdObjectKey>sumTeamplayerFix312</sdObjectKey>
  <modelXPath>sum(currentProject/tasks[name='Bug␣312:␣Fix']/currentSchedule
    /team/assignedEmployees/properties[name='teamplayer']/value)</modelXPath>
</ExternalEquation>
```

- **Score:** Κλάση η οποία κρατάει πληροφορίες για ένα score.

- **ScorePool:** Κλάση υπεύθυνη για το φόρτωμα και την αποθήκευση των Score από ένα αρχείο.

- **Story:** Κλάση η οποία κρατάει πληροφορίες για μια ιστορία. Όπως αναφέρεται στο Κεφάλαιο 5.

- **SinglePlayerScorePlayMode:** Κλάση η οποία υλοποιεί το single player score mode όπως περιγράφεται στο Κεφάλαιο 3.3.1. Η start μέθοδος πρέπει να καλεστεί αφού αναθέσουμε ένα Story. Η μέθοδος αυτή αρχικοποιεί διάφορες κλάσεις του Model επιπέδου και αναθέτει στον χρήστη το πρώτο project της ιστορίας. Η υλοποίηση της start μεθόδου είναι η εξής:

```
Company company = Shell.getInstance( ).getCompany( );
company.setBoss( _currentStory.getBoss( ) );
company.setAccountant( _currentStory.getAccountant( ) );
company.setAdministrator( _currentStory.getAdministrator( ) );
Clock.getInstance( ).addListener( this );
Clock.getInstance( ).setCurrentDateTime( _currentStory.getStartDate( )
      .toDateTime( company.getStartTime( ) ).toMutableDateTime( ) );
Clock.getInstance( ).start( );
company.assignProject( _currentStory.getNextProject( ) );
ScorePool.getAllScores( );
```

  Μόλις γίνει ανάθεσει ενός project στον χρήστη το SinglePlayerScorePlayMode τον ειδοποιεί στέλνοντας του ένα e-mail με την περιγραφή του project. Η μέθοδος που κάνει αυτή τη δουλειά είναι η projectAssigned του IShellListener και η υλοποίησή της είναι η εξής:

```
Company company = _currentStory.getCompany( );
Message message = new Message( );
message.setSender( company.getBoss( ) );
message.setRecipient( company.getProjectManager( ) );
message.setTitle( "New project." );
Project currentProject = company.getCurrentProject( );
message.setBody( currentProject.getDescription( ) );
Shell.getInstance( ).sendMail( message );
```

  Όταν ξεκινάει το project αρχικοποιείται ο ModelMessageReceiver με τα Bindings(Output, External, Event) του συγκεκριμένου project, δημιουργείται ένα καινούργιο SdSystem με το SdModel του project και εκτελείται το πρώτο βήμα του SdSystem. Η μέθοδος που κάνει αυτή τη δουλειά είναι η projectStarted του IShellListener και η υλοποίησή της είναι η εξής:

```
// set up the model message receiver.
ModelMessageReceiver.getInstance( ).clearAll( );
List<OutputVariableBinding> variableBindings = _currentStory
    .getOutputVariableBindings( project );
for ( OutputVariableBinding outputVariableBinding : variableBindings )
{
    ModelMessageReceiver.getInstance( ).addValueSentData(
        outputVariableBinding.getSdObjectKey( ),
```

```
            outputVariableBinding.getModelXPath( ) );
}
List<ExternalEquationBinding> externalBindings = _currentStory
    .getExternalEquationBindings( project );
for ( ExternalEquationBinding externalEquationBinding : externalBindings )
{
    ModelMessageReceiver.getInstance( ).addValueRequestedData(
        externalEquationBinding.getSdObjectKey( ),
            externalEquationBinding.getModelXPath( ) );
}
List<EventBinding> eventBindings = _currentStory.getEventBindings( project );
for ( EventBinding eventBinding : eventBindings )
{
    IModelAction action;
    Class clazz = Class.forName( eventBinding
        .getActionBinding( ).getActionClassName( ) );
    action = (IModelAction) clazz.newInstance( );
    ActionBinding actionBinding = eventBinding.getActionBinding( );
    for ( String paramName : actionBinding.getParameters( ).keySet( ) )
    {
        BeanUtils.setProperty( action, paramName,
            actionBinding.getParameter( paramName ) );
    }
    if ( action.isValid( ) )
    {
        ModelMessageReceiver.getInstance( )
            .addEventAction( eventBinding.getEventName( ),
                eventBinding.getSdObjectKey( ), action );
    }
}
// Set up the SdModel.
SdModel model = _currentStory.getModel( project );
_currentSdSystem = new SdSystem( );
_currentSdSystem.initialize( model.getSdObjects( ) );
// Register the persister.
IStatePersister persister = new MemoryStatePersister( );
_projectPersisters.put( project, persister );
_currentSdSystem.registerStatePersister( persister );
// Execute the first step.
_currentSdSystem.executeStep( );
```

Όταν τελειώνει ένα project προσθέτουμε τα prestige points από το project στα prestige points της εταιρείας, βλέπουμε αν έχει τελειώσει το παιχνίδι και αν έχει τελειώσει δημιουργούμε το Score, εμφανίζουμε το IProjectFinishedUi που έχει γίνει register στον UiManager και αν το παιχνίδι δεν έχει τελειώσει αναθέτουμε το επόμενο project

της ιστορίας στον χρήστη. Η μέθοδος που κάνει αυτή τη δουλειά είναι η projectFinished του IShellListener και η υλοποίησή της είναι η εξής:

```
_currentSdSystem = null;
Company company = Shell.getInstance( ).getCompany( );
Project nextProject = _currentStory.getNextProject( );
company.setPrestigePoints( company.getPrestigePoints( )
  + project.getPrestigePoints( ) );

boolean gameEnded = false;
if ( nextProject == null )
{
    Score score = new Score( company.getProjectManager( ).getLastName( ) +
            "␣" + company.getProjectManager( ).getName( ), company
            .getName( ), Calendar.getInstance( ).getTime( ),
            company.getPrestigePoints( ), _currentStory.getName( ) );
    ScorePool.addScore( score );
    ScorePool.saveScores( );
    gameEnded = true;
}

// Show the ui.
IProjectFinishedUi ui = UiManager.getInstance( ).
          getProjectFinishedUi( SinglePlayerScorePlayMode.class );
ui.setGameEnded( gameEnded );
ui.showUi( project );

if ( nextProject != null )
{
    company.assignProject( nextProject );
}
```

Στο τικ του ρολογιού εκτελείται και ένα βήμα στο μοντέλο του project. Αν έχει περάσει το ωράριο λειτουργίας της εταιρείας γίνεται αλλαγή ημέρας στην και ξεκινάει το ωράριο. Αν έχει περάσει η μέρα πληρώνονται οι υπάλληλοι, τα χρήματα αφαιρούνται από τον προϋπολογισμό του project και στέλνεται ένα e-mail στον χρήστη με το τελικό πόσο. Η μέθοδος που κάνει αυτή τη δουλειά είναι η ticked του IClockListener και η υλοποίησή της είναι η εξής:

```
if ( _currentSdSystem != null )
{
    _currentSdSystem.executeStep( );
}
if ( changedTypes.contains( DurationFieldType.days( ) ) )
{
    Company company = _currentStory.getCompany( );
    double total = company.getAccountant( ).calculateDailySalaries( );
```

```
        LOGGER.info( "Paying employees " + total );
        Project currentProject = currentStory.getCurrentProject( );
        currentProject.setBudget( currentProject.getBudget( ) - total );
        Message message = new Message( );
        message.setSender( company.getAccountant( ) );
        message.setRecipient( company.getProjectManager( ) );
        message.setTitle( "Payments" );
        message.setBody( "Total payments : " + total );
        Shell.getInstance( ).sendMail( message );
    }
    if ( changedTypes.contains( DurationFieldType.hours( ) ) )
    {
        int currentHourOfDay = Clock.getInstance( ).getCurrentDateTime( )
            .getHourOfDay( );
        Company company = Shell.getInstance( ).getCompany( );
        int companyEndHour = company.getEndTime( ).getHourOfDay( );
        if ( currentHourOfDay >= companyEndHour )
        {
            Clock.getInstance( ).performDayChange( company.getStartTime( ) );
        }
    }
```

# 8 Παρατηρήσεις

Το ProMaSi ως ένα μεγάλο project, έχει πολλούς στόχους οι οποίοι δεν μπορούν να εκπληρωθούν στα πλαίσια μίας πτυχιακής εργασίας. Για το λόγο αυτό, το project θα συνεχιστεί και εκτός πτυχιακής. Στόχος είναι μετά το τέλος της να ασχοληθούν και άλλα άτομα, έτσι ώστε να επιτευχθούν όλοι οι επιθυμητοί στόχοι. Για το λόγο αυτό πάρθηκαν μια σειρά από αποφάσεις ώστε να είναι δυνατή η συνέχισή του και στο μέλλον.

Έτσι καταρχήν, το ProMaSi αναπτύχθηκε σαν πρόγραμμα ανοικτού κώδικα, με σκοπό να μπορεί όποιος επιθυμεί να συμμετέχει και να βοηθάει στη βελτίωση του. Επιπροσθέτως τα προγράμματα που χρησιμοποιούνται για την ανάπτυξή του θα πρέπει να έχουν τα εξής βασικά χαρακτηριστικά:

- Να είναι open source ή τουλάχιστον να είναι freeware.
- Να δουλεύουν τουλάχιστον σε Linux και Windows λειτουργικά συστήματα.

Έτσι δεν υποχρεώνουμε τα άτομα που θέλουν να ασχοληθούν με το project να αγοράσουν κάποιο πακέτο λογισμικού. Επιπλέον, τα προγράμματα ανοικτού κώδικα συνήθως είναι καλύτερα από τα προγράμματα κλειστού κώδικα και αναπτύσσονται πολύ πιο γρήγορα.

Τέλος προκειμένου να είναι δυνατή η ανάπτυξη οργανώθηκε ένα συγκεκριμένο πλαίσιο από πολιτικές που διέπουν τον τρόπο συγγραφής του κώδικα, την οργάνωσή του καθώς και τις διαδικασίες συντήρησης οι οποίες παρουσιάζονται συνοπτικά στα παραρτήματα αυτής της πτυχιακής.

# 9 Συμπεράσματα

## 9.1 Μέλλον του project

Στα πλαίσια της πτυχιακής αναπτύξαμε ένα πρωτότυπο το οποίο τεκμηριώνει τις ιδέες μας για το ProMaSi . Αυτό όμως είναι μόνο η αρχή, αρκετοί από τους στόχους που θέσαμε θέλουν αρκετή δουλεία για να ολοκληρωθούν και μπορούν να θεωρηθούν σαν ξεχωριστές πτυχιακές. Μερικές από αυτές είναι:

1. Αυτή τη στιγμή το μόνο εργαλείο που διατίθεται για την δημιουργία ενός σεναρίου είναι ο Core designer ο οποίος χρησιμεύει μόνο στη δημιουργία PSD μοντέλου. Τα υπόλοιπα αρχεία που χρειάζονται για τη δημιουργία του μοντέλου τα δημιουργεί ο εκπαιδευτής. Για να το κάνει αυτό ο εκπαιδευτής πρέπει να έχει μεγάλη γνώση του συστήματος. Αυτό είναι ένα μεγάλο project το οποίο θα βελτιώσει κατά πολύ το ProMaSi .

2. Ένα χαρακτηριστικό του communication επιπέδου είναι ότι μπορεί να στείλει τις τιμές και τα μηνύματα του PSD μοντέλου μέσω δικτύου. Με αυτό το χαρακτηριστικό μπορεί να δημιουργηθεί μια εφαρμογή η οποία θα χρησιμοποιείται από τον εκπαιδευτή για να επιβλέπει τους εκπαιδευόμενους. Ο εκπαιδευτής θα μπορεί να παρακολουθεί κάποιες μεταβλητές από το PSD μοντέλο και να βλέπει ποια γεγονότα έχουν ενεργοποιηθεί για κάθε παίκτη. Αυτό μπορεί να βοηθήσει των εκπαιδευτή στο να δίνει συμβουλές σε κάθε εκπαιδευόμενο και να μπορεί να τονίσει τις λάθος κινήσεις του.

3. Ένα ακόμη project που μπορεί να αναπτυχθεί είναι η δημιουργία του multi player score mode όπως περιγράφεται στο Κεφάλαιο 3.3.2.

## 9.2 Στόχοι

Οι στόχοι που ολοκληρώσαμε είναι οι εξής:

1. **Να είναι πλήρως παραμετροποιήσιμο, έτσι ώστε ο εκπαιδευτής να μπορεί να διδάξει στους εκπαιδευόμενους αυτό που θέλει:** Με τα PSD μοντέλα ο εκπαιδευτής μπορεί να δημιουργήσει events με τα οποία μπορεί με τη βοήθεια των modelXPath να ελέγξει οποιαδήποτε τιμή του Model επιπέδου θέλει και να εκτελέσει ένα action.

2. **Να αναπτυχθεί σαν open source πρόγραμμα, να αναπτυχθεί με δωρεάν open source εργαλεία:** Μπορέσαμε να δημιουργήσουμε ένα μεγάλο project με τη βοήθεια open source εργαλείων μόνο και να δημιουργήσουμε ένα περιβάλλον στο οποίο μπορούν να συμμετέχουν και άλλοι προγραμματιστές.

3. **Να κρατάει το ενδιαφέρον του εκπαιδευόμενου:** Με τη βοήθεια των events του PSD μοντέλου ο εκπαιδευτής μπορεί να δημιουργήσει ένα σενάριο με διάφορα events/actions όπως να φεύγουν οι υπάλληλοι, να προστίθενται καινούργιοι υπάλληλοι στο marketplace, να αλλάζουν οι απαιτήσεις του πελάτη προσθέτοντας δυναμικά καινούργιες εργασίες στο project κ.τ.λ.

4. **Να δίνει μια αίσθηση ρεαλισμού στους εκπαιδευόμενους:** Με το desktop OS ο χρήστης έχει την ψευδαίσθηση ότι δουλεύει σε ένα λειτουργικό σύστημα σε μια εταιρεία.

5. **Οι εκπαιδευόμενοι θα πρέπει να χρησιμοποιούν εργαλεία και μεθόδους που θα χρησιμοποιήσουν και στην πραγματικότητα:** Ο χρήστης στο desktop OS χρησιμοποιεί εργαλεία που θα χρησιμοποιούσε και στην πραγματικότητα(e-mail,gantt κ.τ.λ.).

Ο μόνος στόχος που δεν προλάβαμε να υλοποιήσουμε είναι: *Να παρέχει τα κατάλληλα εργαλεία στον εκπαιδευτή, έτσι ώστε να μπορεί να δημιουργήσει εύκολα αυτό που θέλει*, ο Core designer είναι μέρος αυτού του στόχου άλλα δεν είναι αρκετός γιατί τα περισσότερα αρχεία που χρειάζονται φτιάχνονται με το χέρι. Οι υπόλοιποι στόχοι έχουν υλοποιηθεί άλλα μπορούν να βελτιωθούν.

## 9.3 Παρατηρήσεις

Η ανάπτυξη του ProMaSi πήρε πάρα πολύ καιρό (2 χρόνια) ο χρόνος αυτός ξοδεύτηκε ως εξής:

- 12 μήνες για να αποφασίσουμε τι ακριβώς θέλουμε να κάνει και πως, καθώς και για να βρούμε τις κατάλληλες βιβλιοθήκες.
- 2 μήνες για να στήσουμε το περιβάλλον ανάπτυξης (hosting, programming guidelines, εργαλεία κ.τ.λ).
- 6 μήνες για την ανάπτυξη του κώδικα.
- 4 μήνες για την συγγραφή αυτού του κειμένου.

Το project δεν θα έπαιρνε τόσο χρόνο αν δεν δούλευα σε όλη τη διάρκεια του. Από την άλλη όμως, επειδή η δουλειά που κάνω είναι σχετική (προγραμματιστής) πιστεύω ότι δεν θα είχα την εμπειρία που χρειάζεται για να δημιουργήσω το ProMaSi . Ο περισσότερος χρόνος ξοδεύτηκε στην έρευνα και την αναζήτηση βιβλιοθηκών. Αυτό βοήθησε γιατί χωρίς τη βοήθεια ορισμένων βιβλιοθηκών (JEP,JXPath,Math) η ανάπτυξη του ProMaSi θα ήταν πάρα πολύ δύσκολη. Οι μέρες που πραγματικά ασχολήθηκα με το project είναι οι εξής:

- 4 μήνες για να αποφασίσουμε τι ακριβώς θέλουμε να κάνει και πως, καθώς και για να βρούμε τις κατάλληλες βιβλιοθήκες.
- 1 μήνα για να στήσουμε το περιβάλλον ανάπτυξης (hosting, programming guidelines, εργαλεία κ.τ.λ).
- 2 μήνες για την ανάπτυξη του κώδικα.
- 2 μήνες για την συγγραφή αυτού του κειμένου.

Τέλος όποιος θέλει να ασχοληθεί με το ProMaSi μπορεί να βρει τον κώδικα του στο http://promasi.googlecode.com. Πληροφορίες για το πώς να τον κατεβάσει μπορούν να βρεθούν στο http://promasi.gr. Σε περίπτωση που κάποιος θέλει να ασχοληθεί ενεργά με το project μπορεί να στείλει e-mail στο eddiefullmetal@promasi.gr ή στο nikos@promasi.gr.

# Παράρτημα

## Α΄ Αποφάσεις για τα εργαλεία του έργου

### Γλώσσα προγραμματισμού

Η γλώσσα προγραμματισμού πρέπει να πληρεί τα εξής κριτήρια:

1. Να μπορούν να τρέξουν τουλάχιστον σε Linux και Windows(cross-platform).
2. Να είναι αρκετά ώριμες(να έχουν αρκετούς χρήστες-community,να υπάρχουν αρκετές βιβλιοθήκες κ.τ.λ).

Οι γλώσσες οι οποίες χρησιμοποιούνται περισσότερο και πληρούν τα παραπάνω κριτήρια είναι οι εξής:

- **C++**: Αν και δεν υπάρχει κάποιος compiler για Windows και Linux, ο gcc μπορεί να χρησιμοποιηθεί και στα Windows με το MinGW. Υπάρχουν και αρκετές βιβλιοθήκες οι οποίες είναι cross-platform και open source, όπως η Boost και η QT. Για IDE μπορεί να χρησιμοποιηθεί ο QT Creator, το Eclipse ή το Netbeans τα οποία είναι όλα cross-platform και open source.
- **Java**: Είναι cross-platform και open source γλώσσα προγραμματισμού. Για αυτό το λόγο και όλες οι βιβλιοθήκες που αναπτύσσονται για αυτή την γλώσσα είναι και αυτές cross-platform. Επιπλέον, υπάρχουν πάρα πολλές βιβλιοθήκες, οι οποίες είναι open source. Για IDE μπορεί να χρησιμοποιηθεί το Eclipse ή το Netbeans.
- **C#**: Δεν είναι ακριβώς cross-platform, αλλά το specification της γλώσσας είναι ανοικτό και έχει αναπτυχθεί μια ανεξάρτητη υλοποίηση της γλώσσας, η mono, που είναι για λειτουργικά συστήματα εκτός windows. Η υλοποίηση της mono δεν γίνεται από την εταιρεία που αναπτύσσει τη γλώσσα(Microsoft), αλλά από το community. Το γεγονός αυτό επιδρά και στην ανάπτυξη της γλώσσας, αφού η υλοποίηση από τη Microsoft βρίσκεται στην έκδοση 3.5, ενώ η mono βρίσκεται στην έκδοση 2.0 και υποστηρίζει κάποιες δυνατότητες της έκδοσης 3.0. Όσες βιβλιοθήκες έχουν γίνει από την έκδοση 3.0 και μετά δεν υποστηρίζονται σε άλλα λειτουργικά συστήματα. Για IDE μπορεί να χρησιμοποιηθεί το MonoDevelop, αν και η έκδοση για Windows είναι ακόμα σε δοκιμαστικό στάδιο.

Από τις παραπάνω γλώσσες επιλέχθηκε η Java για την ανάπτυξη του ProMaSi . Η επιλογή αυτής της γλώσσας βασίστηκε στο πολύ μεγάλο community, στην πληθώρα βιβλιοθηκών που μας παρέχει και στο γεγονός ότι είναι μια πραγματικά cross-platform γλώσσα.

## IDE

Υπάρχουν πολλά διαθέσιμα IDE για java, τα καλύτερα εκ των οποίων είναι το Eclipse(http://www.eclipse.org/) και το Netbeans(http://www.netbeans.org/). Η επιλογή του IDE είναι καθαρά θέμα προσωπικής επιλογής, καθώς και τα δύο είναι εξίσου καλά. Το μόνο πλεονέκτημα του Eclipse είναι ότι έχει περισσότερα plug-ins, γι΄ αυτό και επιλέχθηκε.

## Εργαλεία ανάπτυξης εγγράφων

Για την συγγραφή εγγράφων χρησιμοποιήθηκε το OpenDocument format, το οποίο είναι ISO/IEC international standard format και το χρησιμοποιούν πάρα πολλές εφαρμογές. Η εφαρμογή που χρησιμοποιήθηκε για τη συγγραφή OpenDocument αρχείων είναι το Openoffice(http://www.openoffice.org/), το οποίο είναι μια cross-platform - open source εφαρμογή. Επίσης χρησιμοποιήθηκε και το LaTeX για την συγγραφή εγγράφων. Το LaTeX είναι μια γλώσσα, η οποία παρέχει εντολές για την ευκολότερη χρήση του συστήματος στοιχειοθεσίας(typesetting program) TeX και χρησιμοποιείται ευρέως από μαθηματικούς, επιστήμονες, μηχανικούς, φιλόσοφους, οικονομολόγους και άλλους μελετητές στον ακαδημαϊκό και εμπορικό κόσμο, καθώς και από άλλους επαγγελματίες. Το TeX είναι ένα σύστημα στοιχειοθεσίας, το οποίο σχεδιάστηκε και αναπτύχθηκε από τον Donald Knuth, έχοντας 2 στόχους:

1. Να επιτρέπει σε οποιονδήποτε να γράφει υψηλής ποιότητας βιβλία.
2. Να παρέχει ένα σύστημα που θα παράγει το ίδιο αποτέλεσμα σε όλους τους υπολογιστές.

Ένα μεγάλο πλεονέκτημα του LaTeX(και του TeX) είναι ότι χρησιμοποιεί απλά αρχεία κειμένου και έτσι δεν χρειάζεται κάποια ειδική εφαρμογή για την επεξεργασία του. Για να βγει το τελικό έγγραφο, χρειάζεται ένας compiler ο οποίος θα πάρει το αρχείο και θα το μετατρέψει σε PDF(υποστηρίζονται και άλλες μορφές εκτός του PDF όπως HTML).

Για την δημιουργία UML διαγραμμάτων χρησιμοποιήθηκε το MagicDraw (http://www.magicdraw.com/), το οποίο είναι ίσως η καλύτερη εφαρμογή στον τομέα του. Το μειονέκτημα του είναι ότι δεν είναι open source, αλλά έχει community license και μπορεί να χρησιμοποιηθεί δωρεάν για open source projects, όπως το ProMaSi . Αυτή η έκδοση δεν έχει όλες τις δυνατότητες της κανονικής, αλλά αυτές που μας παρέχει είναι αρκετές για το project. Τα αντίστοιχα open source projects που είδαμε δεν είχαν όλες τις δυνατότητες που θέλουμε για την ανάπτυξη του ProMaSi .

# Β′ Project Outline and Libraries

## Δομή project

Το project έχει την εξής δομή:
**PRO.MA.SI**
↪ **Core** *Περιέχει τον κώδικα του Core επιπέδου.*
↪ **Shell** *Περιέχει τον κώδικα του Shell επιπέδου.*
↪ **UI** *Περιέχει τον κώδικα του DesktopOS και του Core Designer.*
↪ **Model** *Περιέχει τον κώδικα του Model επιπέδου.*
↪ **Communication** *Περιέχει τον κώδικα του Communication επιπέδου.*
↪ **Temp** *Περιέχει δοκιμαστίκο κώδικα.*
↪ **Core_Tests** *Περιέχει τα unit tests για το Core επίπεδο.*
↪ **Shell_Tests** *Περιέχει τα unit tests για το Shell επίπεδο.*
↪ **UI_Tests** *Περιέχει τα unit tests για το DesktopOS και για τον Core Designer.*
↪ **Model_Tests** *Περιέχει τα unit tests για το Model επίπεδο.*
↪ **Communication_Tests** *Περιέχει τα unit tests για το Communication επίπεδο.*
↪ **Utilities** *Περιέχει τον κώδικα απο utility κλάσεις οι οποίες είναι κοινές σε όλα τα επίπεδα.*
↪ **Utilities_Tests** *Περιέχει τα unit tests για το Utilities επίπεδο.*
↪ **Data** *Περιέχει τη δομή των αρχείων του ProMaSi , καθώς και τα αρχεία για το tutorial σενάριο.*
↪ **Documents** *Περιέχει όλα τα έγγραφα του ProMaSi .*
↪ **Libraries** *Περιέχει όλες τις εξωτερικές βιβλιοθήκες που χρησιμοποιεί το ProMaSi .*
↪ **Releases** *Περιέχει όλες τις εκδόσεις του ProMaSi .*
↪ **build.xml** *Ένα ant script το οποίο χρησιμοποιείται για παράγει μια εκτελέσιμη έκδοση του κώδικα.*

## Βιβλιοθήκες

Για την ανάπτυξη του ProMaSi χρησιμοποιήθηκαν οι εξής βιβλιοθήκες:

- **joda-time( http://joda-time.sourceforge.net/):** Παρέχει classes για την αντικατάσταση των Java date and time. Ο σχεδιασμός επιτρέπει πολλαπλά συστήματα ημερολόγιων, ενώ εξακολουθεί να παρέχει ένα απλό API. Το προεπιλεγμένο ημερολόγιο είναι το πρότυπο ISO8601 το οποίο χρησιμοποιείται από το ΞΜΛ.

- **JUnit( http://www.junit.org/):** Είναι ένα απλό framework για την ανάπτυξη αυτόματων τεστ(unit tests).

- **Apache commons proper( http://commons.apache.org/components.html):** Παρέχει τα εξής packages:
  - *Attributes:* Runtime API to metadata attributes such as doclet tags.
  - *BeanUtils:* Easy-to-use wrappers around the Java reflection and introspection APIs.
  - *Betwixt:* Services for mapping JavaBeans to XML documents, and vice versa.
  - *Chain:* "Chain of Responsibility" pattern implemention.
  - *CLI:* Command Line arguments parser.
  - *Codec:* General encoding/decoding algorithms (for example phonetic, base64, URL).
  - *Collections:* Extends or augments the Java Collections Framework.
  - *Compress:* Defines an API for working with tar, zip and bzip2 files.
  - *Configuration:* Reading of configuration/preferences files in various formats.
  - *Daemon:* Alternative invocation mechanism for unix-daemon-like java code.
  - *DBCP:* Database connection pooling services.
  - *DbUtils:* JDBC helper library.
  - *Digester:* XML-to-Java-object mapping utility.
  - *Discovery:* Tools for locating resources by mapping service/reference names to resource names.
  - *EL:* Interpreter for the Expression Language defined by the JSP 2.0 specification.
  - *Email:* Library for sending e-mail from Java.
  - *Exec:* API for dealing with external process execution and environment management in Java.
  - *FileUpload:* File upload capability for your servlets and web applications.
  - *IO:* Collection of I/O utilities.
  - *JCI:* Java Compiler Interface
  - *Jelly:* XML based scripting and processing engine.
  - *Jexl:* Expression language which extends the Expression Language of the JSTL.
  - *JXPath:* Utilities for manipulating Java Beans using the XPath syntax.

- *Lang:* Provides extra functionality for classes in java.lang.
- *Launcher:* Cross platform Java application launcher.
- *Logging:* Wrapper around a variety of logging API implementations.
- *Math:* Lightweight, self-contained mathematics and statistics components.
- *Modeler:* Mechanisms to create Model MBeans compatible with JMX specification.
- *Net:* Collection of network utilities and protocol implementations.
- *Pool:* Generic object pooling component.
- *Primitives:* Smaller, faster and easier to work with types supporting Java primitive types.
- *Proxy:* Library for creating dynamic proxies.
- *Sanselan:* A pure-Java image library.
- *SCXML:* An implementation of the State Chart XML specification aimed at creating and maintaining a Java SCXML engine. It is capable of executing a state machine defined using a SCXML document, and abstracts out the environment interfaces.
- *Transaction:* Implementations for multi level locks, transactional collections and transactional file access.
- *Validator:* Framework to define validators and validation rules in an xml file.
- *VFS:* Virtual File System component for treating files, FTP, SMB, ZIP and such like as a single logical file system.

Απο τα παραπάνω πακέτα πολύ σημαντικό ρόλο στην ανάπτυξη του ProMaSi παίξαν τα εξής:

1. Το JXPath, με τη βοήθεια του πακέτου αυτού αναπτύχθηκαν τα modelXPath expressions που περιγράφηκαν στο Κεφάλαιο5.
2. Το Math, με τη βοήθεια του πακέτου αυτού αναπτύχθηκε η lookup equation που περιγράφεται στο Κεφάλαιο4 και ποιο συγκεκριμένα με τη βοήθεια του SplineInterpolator.

- **JEP( http://www.singularsys.com/jep/):** Μπορούμε να δηλώσουμε εξισώσεις π.χ $(x - y) * 2$ σαν κείμενο, να του περάσουμε στην συνέχεια τις παραμέτρους(x,y) και να μας επιστρέψει το αποτέλεσμα. Το JEP χρησιμοποιήθηκε για την υλοποίηση της constant equation που περιγράφτηκε στο Κεφάλαιο4.

- **JFreechart( http://www.jfree.org/jfreechart/):** Είναι μια βιβλιοθήκη για τη δημιουργία γραφικών παραστάσεων. Με τη βοήθεια αυτής της βιβλιοθήκης έγινε ο lookup equation editor και τα στατιστικά των μεταβλητών κατα την ολοκλήρωση του project στο desktop ui.

- **Swing:** Core βιβλιοθήκη της java για τη δημιουργία ui. Χρησιμοποιήθηκε για την ανάπτυξη του desktop ui και τουλάχιστον core designer.

- **MiGLayout( http://www.miglayout.com/):** Ο καλύτερος layout manager ( http://java.sun.com/docs/books/tutorial/uiswing/layout/using.html) για swing. Χρησιμοποιήθηκε σε όλα τα GUI components.

- **GEF( http://gef.tigris.org/):** Βιβλιοθήκη που βοηθάει στην ανάπτυξη προγραμμάτων επεξεργασία διαγραμμάτων(UML κ.τ.λ). Με τη βοήθεια αυτής της βιβλιοθήκης αναπτύχθηκε ο core designer.

- **Log4j( http://logging.apache.org/log4j/):** Χρησιμοποιείται για το logging της εφαρμογής.

- **SwingX( http://www.swinglabs.org/):** Παρέχει extra ui components για το swing toolkit.

- **jide-oss( https://jide-oss.dev.java.net/):** Παρέχει extra ui components για το swing toolkit.

Όλες οι παραπάνω βιβλιοθήκες είναι open-source εκτός απο τη JEP η οποια απο την έκδοση 2.4.1 και μετά έγινε commercial.

# Γ′ Programming Guidelines

Για την συγγραφή κώδικα του ProMaSi χρησιμοποιήθηκαν οι παρακάτω κανόνες.

## Naming Conventions

1. Names representing packages should be in all lower case

   `mypackage, org.myname.applicationname.ui`

2. Names representing types must be nouns and written in pascal case

   `Line, AudioSystem`

3. Names representing abstract classes must be prefixed with Abstract

   `AbstractParser`

4. Names representing interfaces must be prefixed with I

   `IParser`

5. Types extending abstract classes should replace the Abstract prefix with the specific implementation

   `AbstractParser => XmlParser`

6. Types implementing interfaces should replace the I prefix with the specific implementation

   `IParser => XmlParser`

7. Names representing enumerations must be suffixed with type

   `ParserType`

8. Names representing enumeration literals must be written in camel case

   `XmlParser, CsvParser`

9. Names representing variables must be written in camel case

   `line, audioSystem`

10. Names representing constants must be all upper case using underscore to separate words

    MAX_ITERATIONS, COLOR_RED

11. Names representing methods must be verbs written in camel case

    `getName(), computeTotalWidth()`

12. Abbreviations and acronyms should not be upper case when used as a name

    `exportHtmlSource(), openDvdPlayer()`

13. Private class variables should have underscore prefix

```
class Person
{
  private String _name;
  ...
}
```

14. Generic variables should have the same name as their type.

```
setTopic(Topic topic), connect(Database database)
```

15. Variables with large scope should have long names, variables with small scope can have short names.

```
for(int i=0;i<MAX_ITERATIONS;i++)
{
  ...
}
```

16. The name of the object is implicit, and should be avoided in a method name

```
line.getLength() //NOT line.getLineLength()
```

17. The terms get/set must be used when a class variable is accessed directly

```
employee.setName(name), employee.getName()
```

18. Boolean variables must be prefixed with is,can,has and have the appropriate set/get methods

```
isStopped, setStopped(boolean stopped), isStopped()
canStop, setStop(boolean stop), canStop()
hasStopped, setStopped(boolean stopped), hasStopped()
```

19. The term compute can be used in methods where something is computed

```
computeAverage()
```

20. The term find can be used in methods where something is looked up

```
findShortestPath(), findUnusedElements()
```

21. The term initialize can be used where an object or a concept is established

```
initializeFontSet(), initializeGui()
```

22. Plural should be used on names representing a collection of objects

```
Collection<Point> points, int[] values
```

23. numberOf prefix should be used for variables representing a number of objects

```
numberOfPoints, numberOfLines
```

24. Abbreviations in names should be avoided

```
computeAverage() // Not computeAvg()
```

25. Negated boolean variables names must be avoided

```
boolean isFound // Not boolean isNotFound
```

26. Associated Constants should be prefixed by a common type name

```
final int COLOR_RED = 1;
final int COLOR_GREEN = 2;
final int COLOR_BLUE = 3;
```

27.Exception classes should be suffixed with Exception

```
Class AccessException, Class ValidatorException
```

28.Default interface implementations can be prefixed by Default

```
Class DefaultTableCellRenderer implements ITableCellRenderer
```

29.Singleton classes should return their sole instance through method getInstance()

```
Class SwingManager
{
 private static final SwingManager _INSTANCE = new SwingManager();

 private SwingManager()
 {
 }

 public static SwingManager getInstance()
 {
  return _INSTANCE;
 }
}
```

30.All the test classes must have the Tester suffix, and the class name that is tested as a prefix.

`EquationTester , XmlParserTester`

## Files

1. Each line must be 150 characters long.
2. Each file should be 1000 characters long.
3. Each method should be 500 characters long.

## Statements

1. Class and Interface declarations should be organized in the following manner:
   i. Class/Interface documentation.
   ii. class or interface statement.
   iii. Class (static) variables in the order public, protected, package (no access modifier), private.
   iv. Instance variables in the order public, protected, package (no access modifier), private.
   v. Constructors.
   vi. Class Methods.
   vii. Getters/Setters
2. Variables should be initialized where they are declared and they should be declared in the smallest scope possible.
3. Class variables should never be declared public or protected, use protected or public setter/getter methods instead.
4. Variables should be kept alive for as short a time as possible.
5. Only logical expressions must be included in the for/while/if construction.
6. The use of break and continue in loops should be avoided.
7. Complex conditional expressions must be avoided. Introduce temporary boolean variables instead

8. The use of magic numbers in the code should be avoided they can be declared as named constants instead.

## Layout and Comments

1. All comments should be written in English.
2. All classes,methods and class variables should be documented using the Java documentation (javadoc) conventions.
3. All packages should be documented using a package-info.java file.
4. Tricky code should not be commented but rewritten if possible.
5. Getters must have the following javadoc documentation.

```
/**
* @return The {@link _test}.
*/
public String getTest()
```

6. Setters must have the following javadoc documentation

```
/**
* @param test The {@link _test} to set.
*/
public void setTest ( String test )
```

7. Brackets must start in a new line

```
Class XmlParser
{
  ...
}
while(true)
{
  ...
}
```

8. White spaces
   - Operators should be surrounded by a space character.

- Java reserved words should be followed by a white space.
- Commas should be followed by a white space.
- Colons should be surrounded by white space.
- Semicolons in for statements should be followed by a space character.

9. Logical units within a block should be separated by one blank line.

# Δ΄ Source Control and Build Management

## Source Control Management

Για την ανάπτυξη του ProMaSi χρησιμοποιήθηκε το SVN. Το SVN είναι ένα version control ή source code management(SCM) σύστημα. Τα SCM διαχειρίζονται τις αλλαγές σε έγγραφα, προγράμματα και άλλα αρχεία και χρησιμοποιούνται κυρίως για την ανάπτυξη λογισμικού, όπου μια ομάδα ανθρώπων μπορεί να αλλάζει ίδια αρχεία. Συνήθως, τα αρχεία κρατιούνται σε ένα κεντρικό server και όποιος έχει πρόσβαση μπορεί να τα κατεβάσει και να τα επεξεργαστεί. Αφού τα επεξεργαστεί, πρέπει να τα ανεβάσει στο server. Σε κάθε αλλαγή που γίνεται σε ένα αρχείο το SCM αλλάζει την έκδοση του. Οι χρήστες μπορούν να δουν τις διαφορές μεταξύ των εκδόσεων του αρχείου και να δούν τι αλλαγές έκανε ο κάθε χρήστης. Ο κώδικας του ProMaSi φιλοξενείται στο Google-Code( http://promasi.googlecode.com), απο εκεί μπορεί οποιοσδήποτε να κατεβάσει και να δεί τον κώδικα. Περισσότερες λεπτομέρειες για το πως μπόρει κάποιος να κατεβάσει των κώδικα υπάρχουν στο http://promasi.gr/.

## Ant Script

Για τη δημιουργία των εκτελέσιμων αρχείων χρησιμοποιήθηκε το ant( http://ant.apache.org/). Το ant είναι ένα build εργαλείο σαν το make. Αντί να χρησιμοποιεί εντολές χρησιμοποιεί java classes κάνοντάς το έτσι ένα cross-platform build εργαλείο. Το format του αρχείου είναι XML. Το XML έχει διάφορες εργασίες οι οποίες εκτελούνται με τη σειρά που γράφτηκαν στο αρχείο. Υπάρχουν πάρα πολλές εργασίες και μπορούμε να φτίαξουμε και δικές μας δημιουργώντας μια κλάση η οποία υλοποιεί το Task interface.

Το ant του ProMaSi βγάζει ένα zip αρχείο στο φάκελο Releases με όνομα Promasi-v0.1.0_M20091012-1400.zip όπου 0.1.0 η τωρινή έκδοση του ProMaSi και M20091012-1400 είναι η ημερομηνία και ώρα που εκτελέστηκε το ant. Το zip αρχείο έχει την εξής μορφή:
**Data** - περιέχει τα log αρχεία και τα αρχεία του single player score mode.
**docs** - περιέχει τα javadoc αρχεία.
**lib** - περιέχει όλες τις εξωτερικές βιβλιοθήκες που χρησιμοποιεί το ProMaSi . Μέσα σε αυτόν τον φάκελο θα βρούμε και τα jar αρχεία για το Core,Communication,Model,Shell,Utilities.

**src** - περιέχει τον source κώδικα.
**TestReports** - περιέχει τα αποτελέσματα απο τα JUnit tests. **Core-Designer.bat** - αρχείο για να εκτελείτε ο core-designer σε windows λειτουργικό.
**Core-Designer.sh** - αρχείο για να εκτελείτε ο core-designer σε linux λειτουργικό.
**Core-Designer-v0.1.0_M20091022-1400.jar** - jar αρχείο του core-designer
**Promasi-Desktop.bat** - αρχείο για να εκτελείτε ο desktop OS σε windows λειτουργικό.
**Promasi-Desktop.sh** - αρχείο για να εκτελείτε ο desktop OS σε linux λειτουργικό.
**Promasi-Desktop-v0.1.0_M20091022-1400.jar** - jar αρχείο του desktop OS

# Αναφορές


[1] D. Evans, *Teaching Software Engineering Using Lightweight Analysis*. 2001.

[2] D. Deveaux, R. Fleurquin, and P. Frison, "Software engineering teaching: a "Docware" approach," *SIGCSE Bulletin*, vol. 31, no. 3, pp. 163–166, 1999.

[3] M. Jazayeri, "The education of a software engineer," in *Proceedings of the 19th IEEE international conference on automated software engineering*, vol. 0, pp. xviii–xxvii, IEEE Computer Society, 2004.

[4] D. Callahan and B. Pedigo, "Educating experienced IT proffessionals by addressing industry's needs," vol. 19, pp. 57–62, Sep./Oct. 2002.

[5] R. Conn, "Developing software engineers at the C-130J software factory," *IEEE Software*, vol. 19, pp. 25–29, Sep/Oct 2002.

[6] "System dynamics quick introduction." http://www.public.asu.edu/ kirkwood/sysdyn/SDIntro/SDIntro.htm.

[7] D. Rodríguez, M. Ángel Sicilia, J. J. Cuadrado-Gallego, and D. Pfahl, "e-Learning in project management using simulation models: A case study based on the replication of an experiment," *IEEE Transactions on Education*, vol. 49, no. 4, pp. 451–463, 2006.

[8] J. Forrester, *Industrial Dynamics*. MIT Press, 1961.

[9] T. Abdel-Hamid, "The dynamics of software project staffing: A system dynamics based simulation approach," *IEEE Transactions on Software Engineering*, vol. 15, no. 2, pp. 109–119, 1989.

[10] "System dynamics modeling for project management." http://web.mit.edu/jsterman/www/SDG/project.html.

[11] "Vensim." http://www.vensim.com/.

[12] S. Berkun, *The Art of Project Management*. 2005.

[13] F. P. Brooks, *Mythical Man Month*. 1995.

[14] "Wikipedia." http://en.wikipedia.org.

[15] "Project management triangle." http://www.projectsmart.co.uk/introduction-to-project-management.html.

[16] “Simse project.” http://www.ics.uci.edu/ emilyo/SimSE/.

[17] “Xpath.” http://www.w3.org/TR/xpath.

[18] “Open source.” http://www.opensource.org/.

[19] “System dynamics society.” http://www.systemdynamics.org/.

[20] “Mit system dynamics group.” http://scripts.mit.edu/ sdg/.