Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Πανεπιστήμιο Πειραιώς

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**UNIFIED MODELING LANGUAGE ΚΑΙ ΘΕΩΡΙΑ ΠΑΙΓΝΙΩΝ**
Ονοματεπώνυμο Φοιτητή	**ΘΩΜΑΣ ΣΚΟΔΡΑΣ**
Πατρώνυμο	**ΠΑΝΑΓΙΩΤΗΣ**
Αριθμός Μητρώου	**ΜΠΠΛ/ 09050**
Επιβλέπων	**Ευάγγελος Φούντας, Ομότιμος Καθηγητής**

Πανεπιστήμιο Πειραιώς

Ημερομηνία Παράδοσης **Οκτώβριος 2014**

Πανεπιστήμιο Πειραιώς

Πανεπιστήμιο Πειραιώς

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Μαρία Βίρβου Καθηγήτρια	Γεώργιος Τσιχριντζής Καθηγητής	Κωνσταντίνος Πατσάκης Λέκτορας

Περιεχόμενα

Περίληψη **8**

1 Κεφάλαιο U.M.L. **10**

1.1 Εισαγωγή στις Γλώσσες Μοντελοποίησης *10*

1.1.1 Η Ενοποιημένη Γλώσσα Μοντελοποίησης 11

1.2 Επισκόπηση *12*

1.3 Μεθόδοι ανάπτυξης λογισμικού *13*

1.4 Επισκόπηση διαγραμμάτων: *14*

1.4.1 Διαγράμματα Δομής 15

1.4.2 Διαγράμματα Συμπεριφοράς 16

1.4.3 Διαγράμματα Αλληλεπίδρασης 17

1.5 Αναλυτική περιγραφή διαγραμμάτων *17*

1.5.1 Use Case Διαγράμματα 18

1.5.2 Activity Διαγράμματα 25

1.5.3 Class Διαγράμματα 32

1.5.4 Sequence Διαγράμματα 36

1.5.5 Collaboration Διαγράμματα 39

1.5.6 Statechart Διαγράμματα 40

1.5.7 Component Diagrams 43

1.5.8 Deployment Diagrams 45

Πανεπιστήμιο Πειραιώς

2 Θεωρία Παιγνίων 50

2.1 Ιστορική Αναδρομή 51

2.2 Εφαρμογές της θεωρίας παιγνίων 51

2.3 Βασικές έννοιες 52

2.4 Κατηγορίες παιγνίων 55

2.5 Η ισορροπία Nash 57

2.5.1 Προσέγγιση της ισορροπίας Nash 57

2.6 Εξέταση διαφόρων παιγνίων 59

2.6.1 Το δίλημμα του φυλακισμένου “Prisoner’s dilemma” 59

2.6.2 Η μάχη των φύλων “Battle of the Sexes” 63

2.6.3 Το παίγνιο “Chicken Game” 64

2.6.4 Το κλασσικό παιχνίδι κυριαρχίας κινδύνου “Risk Dominance” 65

2.6.5 Το παίγνιο “Matching Pennies” 66

2.7 Αλγόριθμος επίλυσης προβλημάτων Θεωρίας Παιγνίων 67

3 Σχεδιασμός εφαρμογής 69

3.1 Διαδικασία χρήσης UML 70

3.2 Μεθοδολογία Ανάπτυξης Εφαρμογής 76

3.2.1 Συλλογή Απαιτήσεων 76

3.2.2 Use Case διαγράμματα 78

Πανεπιστήμιο Πειραιώς

3.2.3 Activity διαγράμματα 82
3.2.4 Ανάλυση περιβάλλοντος χρήστη (domain model) 86
3.2.5 Class Διάγραμμα 90
3.2.6 Sequence Διάγραμμα 91
3.2.7 Component διάγραμμα εφαρμογής 93
3.2.8 Deployment διάγραμμα εφαρμογής 94
3.2.9 Ανάπτυξη και Έλεγχος 95

4 Συμπεράσματα 96

5 Βιβλιογραφία 97

5.1 Ξένη Βιβλιογραφία 97

5.2 Ελληνική Βιβλιογραφία 98

5.3 Internet Sites (Links) 98

5.4 Notes 99

5.5 Διπλωματικές Εργασίες 99

5.6 Manuals 100

Πανεπιστήμιο Πειραιώς

Περίληψη

Σκοπός της παρούσας εργασίας είναι η μελέτη της χρήσης των διαγραμμάτων της Unified Modeling Language στη θεωρία παιγνίων.

Η Unified Modeling Language (UML) είναι μια γλώσσα μοντελοποίησης γενικού σκοπού στον τομέα του software engineering. Χρησιμοποιείται για τον προσδιορισμό, τη γραφική απεικόνιση και την τεκμηρίωση των στοιχείων ενός συστήματος λογισμικού. Μπορεί να χρησιμοποιηθεί σχεδόν σε όλες τις φάσεις της ανάπτυξης του λογισμικού, από την ανάλυση απαιτήσεων ως τον τελικό έλεγχο του ολοκληρωμένου συστήματος. Ωστόσο η χρήση της UML είναι τόσο ευρεία σε σχέση με το τι μπορεί να περιγραφεί με τη χρήση των διαγραμμάτων της ώστε πολλές φορές μπορεί να θεωρηθεί ότι η φαντασία είναι ο μοναδικός περιορισμός σε σχέση με το τι μπορεί κανείς να περιγράψει με τη χρήση αυτής της γλώσσας.

Η θεωρία παιγνίων αποτελεί μια μεθοδολογία ανάλυσης καταστάσεων μεταξύ μιας ομάδας λογικών ατόμων η οποία ανταγωνίζεται με σκοπό την απόκτηση του μεγαλύτερου δυνατού οφέλους από τον κάθε ένα. Σκοπός της είναι να μας βοηθήσει να καταλάβουμε διάφορες καταστάσεις αλληλεπίδρασης μεταξύ δύο ή περισσοτέρων οντοτήτων, κάθε μία από τις οποίες συμπεριφέρεται με στρατηγικό τρόπο προσπαθώντας να πάρει κάποιες αποφάσεις οι οποίες θα μεγιστοποιήσουν το συμφέρον της. Η μεμονωμένες οντότητες στην συγκεκριμένη περίπτωση ονομάζονται παίκτες, και είναι αυτοί που παίρνουν αποφάσεις. Σκοπός των παικτών είναι η μεγιστοποίηση του κέρδους τους, το οποίο μετράται σε μια κλίμακα ωφέλειας. Επομένως το παίγνιο όπως αναφέρεται στην θεωρία παιγνίων αναπαριστά την κατάσταση κατά την οποία δύο ή περισσότεροι παίκτες επιλέγουν τρόπους ενέργειας, δημιουργώντας καταστάσεις αλληλεξάρτησης.

Σκοπός της παρούσας εργασίας είναι η μελέτη της χρήσης των ιδιοτήτων της UML στην επίλυση προβλημάτων θεωρίας παιγνίων και το κατά πόσο βοηθά η εναλλακτική απεικόνιση του προβλήματος με χρήση των διαγραμμάτων της UML σε σχέση με την κλασσική απεικόνιση με πίνακες.

Η εργασία χωρίζεται σε τρία βασικά μέρη. Στο πρώτο γίνεται παρουσίαση της γλώσσας UML και δίδονται παραδείγματα για το κάθε διάγραμμα βασισμένα στην εφαρμογή επίλυσης προβλημάτων θεωρίας παιγνίων. Στο δεύτερο μέρος γίνεται παρουσίαση της θεωρίας παιγνίων μέσα από μερικά χαρακτηριστικά παραδείγματα τα οποία συνοδεύονται από εξειδικευμένα διαγράμματα που αφορούν το κάθε πρόβλημα. Τέλος γίνεται παρουσίαση του σχεδιασμού της εφαρμογής επίλυσης προβλημάτων θεωρίας παιγνίων.

Πανεπιστήμιο Πειραιώς

Abstract

The purpose of this study is to investigate the use of diagrams of Unified Modeling Language in Game Theory.

The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system. It can be used in many different phases of development, from requirements analysis to the control of integrated systems. However, the use of UML is so broad in terms of what can be described with the use of diagrams that can often be seen that the imagination is the only limitation in terms of what can be described using this language.

Game theory is a methodology for analyzing situations among a group of reasonable people that compete each other with the purpose of each one to get the most benefit. Its purpose is to help us understand the various situations in which two or more interacting entities, each of which behaves strategically and trying to get some decisions. The individual entity in this case is named player, and is the one who makes decisions. The aim of each player is to maximize profit, which is measured on a scale of interest. So the game mentioned in game theory represents the situation in which two or more players choose courses of action that create situations of interdependence.

The purpose of this study is to evaluate the use of the properties of UML in game theory problem solving and whether it helps the alternative representation of the problem with the use of UML diagrams in relation to classical imaging tables.

The study is divided into three main parts. The first presents the UML language and examples are given for each block based on the application of problem-solving game theory. The second part is a presentation of game theory through some examples accompanied by specialized diagrams concerning each problem. Finally we present the design of the game theory problem solving application.

Πανεπιστήμιο Πατρών

1 Κεφάλαιο U.M.L.

1.1 ΕΙΣΑΓΩΓΗ ΣΤΙΣ ΓΛΩΣΣΕΣ ΜΟΝΤΕΛΟΠΟΙΗΣΗΣ

Οι γλώσσες μοντελοποίησης είναι τεχνητές γλώσσες με σκοπό την χρησιμοποίησή τους για την έκφραση πληροφοριών και γνώσεων σε δομές που ορίζονται από ένα συνεκτικό σύνολο κανόνων. Οι κανόνες χρησιμοποιούνται για την ερμηνεία του νοήματος των μερών του συστήματος.

Οι γλώσσες μοντελοποίησης μπορεί να είναι γραφικές ή λεκτικές.

- Οι γραφικές γλώσσες μοντελοποίησης χρησιμοποιούν διαγράμματα με καλά ορισμένα σύμβολα που αντιστοιχούν στις έννοιες , γραμμές και βέλη που συνδέουν τα σύμβολα και δείχνουν τις σχέσεις μεταξύ των συμβόλων και διάφορα άλλα γραφικά σύμβολα που εκπροσωπούν τους περιορισμούς.

- Οι λεκτικές γλώσσες μοντελοποίησης χρησιμοποιούν συγκεκριμένες αυστηρά ορισμένες και τυποποιημένες λέξεις-κλειδιά που συνοδεύονται από παραμέτρους ή άλλους γλωσσικούς όρους και φράσεις προκειμένου να δημιουργήσουν εκφράσεις ερμηνεύσιμες σε υπολογιστικό σύστημα (πχ γλώσσα EXPRESS).

Μερικές γλώσσες μοντελοποίησης, αν και είναι εκτελέσιμες, η χρήση τους δεν σημαίνει απαραίτητα ότι ο ρόλος των προγραμματιστών δεν είναι εξίσου σημαντικός. Αντιθέτως, οι εκτελέσιμες γλώσσες μοντελοποίησης χρησιμοποιούνται επικουρικά στην κατασκευή και δημιουργία λογισμικού ενισχύοντας την παραγωγικότητα των προγραμματιστών, ώστε πιο δύσκολα προβλήματα, όπως το parallel computing ή τα distributed systems να μπορούν να αντιμετωπιστούν με μεγαλύτερη ευκολία και αποδοτικότητα.

Στη βιβλιογραφία απαντάται ένας μεγάλος αριθμός γλωσσών μοντελοποίησης.

Για τις ανάγκες της παρούσας εργασίας θα επικεντρωθούμε στη μελέτη της Ενοποιημένης Γλώσσας Μοντελοποίησης ή Unified Modeling Language (UML).

Όπως αναφέραμε παραπάνω η UML είναι μία πρότυπη γλώσσα μοντελοποίησης με καλά καθορισμένους τρόπους επικοινωνίας μεταξύ όλων των εμπλεκόμενων στη δημιουργία του συστήματος μερών (προγραμματιστές, σχεδιαστές, πελάτες κλπ). Ο οργανισμός OMG, ο οποίος ανέπτυξε και καθιέρωσε την UML σαν γενικό πρότυπο, αναφέρει την UML σαν...

> *«...μια γραφική γλώσσα η οποία συμβάλλει στην οπτικοποίηση (visualizing), προσδιορισμό (specifying), κατασκευή (constructing) και τεκμηρίωση (documenting) συστημάτων λογισμικού...»*

Πανεπιστήμιο Πειραιώς

Η UML αποτελεί μια αυστηρή, καλά ορισμένη και σαφή «γλώσσα» για τον προσδιορισμό, την απεικόνιση, την κατασκευή και την τεκμηρίωση όλων των μερών των πληροφοριακών συστημάτων που αναπτύσσονται σε ένα οργανισμό καθώς επίσης και του σχεδιασμού του επιχειρηματικού μοντέλου (business model) της εφαρμογής. Η UML αποτελεί το σύνολο των καλύτερων μερών των μεθοδολογιών για Αντικειμενοστραφή ανάλυση και σχεδιασμό που αναπτύχθηκαν κατά τις δεκαετίες του 80 και 90 και συγκεντρώνει τα καλύτερα χαρακτηριστικά τους ούτως ώστε να καθίσταται εύκολη η μοντελοποίηση και κατασκευή σύνθετων και μεγάλων υπολογιστικών συστημάτων.
Συγκεκριμένα η UML σαν πρότυπο καθιερώθηκε τον Νοέμβριο του 1997 από το Object Management Group (OMG), ένα ανοιχτό consortium εταιρειών πληροφορικής όπου αποτελεί συνένωση των μεθοδολογιών Object Modeling Technique (OMT) του Jim Rumbaugh, Booch του Grady Booch και της μεθοδολογίας Objectory του Ivar Jacobson (γνωστοί στην διεθνή βιβλιογραφία ως «the Three Amigos»). Σκοπός του οργανισμού OMG ήταν η συνένωση των καλύτερων στοιχείων των παραπάνω μεθοδολογιών και η παρουσίαση ενός γενικού κοινώς αποδεκτού προτύπου σχεδιασμού και ανάπτυξης εφαρμογών. Η UML αναπτύχθηκε από τους Grady Booch, Ivar Jacobson και James Rumbaugh στην Rational Software το 1990. Το 2000 η UML έγινε αποδεκτή από τον International Organization for Standardization (ISO) ως το βιομηχανικό στάνταρ για τη μοντελοποίηση software συστημάτων. Η παρούσα έκδοση της UML είναι η 2.4.1 και εκδόθηκε από τον OMG τον Αύγουστο του 2011.
Η UML αποτελεί ένα σύνολο γραφικών σχημάτων που προσφέρει την δυνατότητα δημιουργίας των βασικών σχεδίων (blueprints) ενός συστήματος. Η UML παρέχει στους εμπλεκομένους στην ανάπτυξη συστημάτων ένα ευρύ σύνολο από γραφικά στοιχεία, σχετικά με την τεχνολογία της πληροφορικής, όπως για παράδειγμα κλάσεις (classes), συστατικά (components), αντικείμενα (objects), κόμβους (nodes), δραστηριότητες (activities), use cases, καταστάσεις (states) και τις συσχετίσεις (relationships, associations) για τον ολοκληρωμένο σχεδιασμό ενός συστήματος.
Το συνηθέστερο παράδειγμα που χρησιμοποιείται για να περιγράψει την φύση της UML (και γενικά για την χρήση ενός μοντέλου για μια αφηρημένη παρουσίαση της εφαρμογής) είναι αυτό της κατασκευής ενός οικοδομήματος. Ο μηχανικός (ο σχεδιαστής) χρησιμοποιεί το αρχιτεκτονικό σχέδιο του οικοδομήματος για να μπορέσει να επικοινωνήσει με τον εργολάβο (προγραμματιστή) που αναλαμβάνει την ανέγερση του οικοδομήματος. Χωρίς τη χρήση του σχεδίου η επικοινωνία μεταξύ του μηχανικού και του εργολάβου θα ήταν από δύσκολη έως αδύνατη. Έτσι όλα τα απαιτούμενα στοιχεία που θα πρέπει να χρησιμοποιηθούν για την υλοποίηση του σχεδίου της εφαρμογής, παρέχονται από την UML προκειμένου ο μηχανικός (σχεδιαστής) να μπορέσει να περιγράψει στον εργολάβο (προγραμματιστή) με ακρίβεια το σχέδιο της εφαρμογής.
Η UML παρέχει στους χρήστες τρόπους περιγραφής της κάθε πλευράς του συστήματος μέσα από ένα πλήθος διαγραμμάτων. Οι διαφορετικές αυτές πλευρές του συστήματος ονομάζονται όψεις (views). Οι όψεις αυτές παρουσιάζουν το σύστημα από διαφορετικές οπτικές γωνίες όπως θα περιγράψουμε αναλυτικότερα παρακάτω: την στατική, την λειτουργική και την δυναμική όψη του συστήματος.

1.1.1 Η Ενοποιημένη Γλώσσα Μοντελοποίησης

Η Unified Modeling Language (UML) αποτελεί μια γενικού σκοπού γλώσσα μοντελοποίησης στο πεδίο του software engineering. Χρησιμοποιείται για τη γραφική απεικόνιση, προσδιορισμό, κατασκευή και τεκμηρίωση των στοιχείων ενός συστήματος λογισμικού. Μπορεί να χρησιμοποιηθεί σχεδόν σε όλες τις φάσεις ανάπτυξης, από την ανάλυση απαιτήσεων ως τον έλεγχο ενός ολοκληρωμένου συστήματος. Αποτελείται από ένα σύνολο προσυμφωνημένων, αυστηρώς ορισμένων όρων, συμβόλων και διαγραμμάτων που επιτρέπουν:

Πανεπιστήμιο Πειραιώς

- την εμφάνιση των βασικών λειτουργιών και των ορίων ενός συστήματος, χρησιμοποιώντας «περιπτώσεις χρήσης» (use-cases) και «actors»
- την περιγραφή της πραγματοποίησης των περιπτώσεων χρήσης με «διαγράμματα αλληλεπίδρασης»
- την περιγραφή της στατικής δομής ενός συστήματος χρησιμοποιώντας «διαγράμματα κλάσεων»
- τη μοντελοποίηση της συμπεριφοράς και της αλληλεπίδρασης των αντικειμένων με «διαγράμματα καταστάσεων»
- την περιγραφή της υλοποίησης του σχεδίου της αρχιτεκτονικής με «διαγράμματα συστατικών» και «ανάπτυξης»
- την επέκταση της λειτουργικότητας με «στερεότυπα».

1.2 ΕΠΙΣΚΟΠΗΣΗ

Η UML συνδυάζει τεχνικές μοντελοποίησης δεδομένων (διαγράμματα σχέσης οντοτήτων), επιχειρήσεων (ροές εργασίας), αντικειμένων, και των επιμέρους τμημάτων. Χρησιμοποιείται με όλες τις διαδικασίες, σε όλες τις φάσεις ανάπτυξης του κύκλου ζωής του λογισμικού, και μεταξύ των διαφόρων τεχνολογιών υλοποίησης.

Η UML, προσφέρει ένα πρότυπο τρόπο για την απεικόνιση αρχιτεκτονικών σχεδίων ενός συστήματος, συμπεριλαμβανομένων κάποιων στοιχείων όπως:

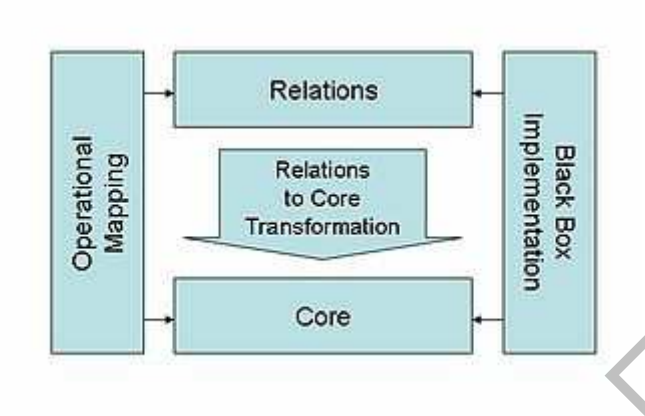
- δραστηριότητες (activities)
- φορείς (actors)
- επιχειρηματικές διαδικασίες (business processes)
- βάση δεδομένων (database schemas)
- λογικά στοιχεία (logical components)
- γλώσσα προγραμματισμού δηλώσεις (programming language statements)
- επαναχρησιμοποιήσιμα στοιχεία λογισμικού (reusable software components).

Η UML στηρίζεται στη μέθοδο Booch, στην Object Modeling Technique (OMT) και στην Object oriented Software Engineering (OOSE) και έχει δημιουργήσει μια ενιαία, κοινή και ευρέως χρησιμοποιήσιμη γλώσσα μοντελοποίησης. Στόχος της UML είναι να αποτελέσει την καλύτερη

Πανεπιστήμιο Πειραιώς

πρότυπη γλώσσα μοντελοποίησης που να μπορεί να μοντελοποιήσει παράλληλα και κατανεμημένα συστήματα (distributed systems).

Οι εφαρμογές QVT (Querty/View/Transformation) εξειδικεύονται στο να μετατρέπουν ένα μοντέλο UML σε άλλες γλώσσες προγραμματισμού όπως για παράδειγμα σε Java. Η UML είναι επεκτάσιμη, με δύο μηχανισμούς για την προσαρμογή: τα προφίλ (profiles) και τα στερεότυπα (stereotypes)



Διάγραμμα 1: Μετατροπή UML μέσω QVT (*πηγή en.wikipedia.org***)**

1.3 ΜΕΘΟΔΟΙ ΑΝΑΠΤΥΞΗΣ ΛΟΓΙΣΜΙΚΟΥ

Η UML δεν αποτελεί από μόνη της μέθοδο ανάπτυξης. Ωστόσο, ο σχεδιασμός της της επιτρέπει να είναι συμβατή με όλες τις κορυφαίες object-oriented μεθόδους ανάπτυξης λογισμικού. Η UML εξελίσσεται συνεχώς και κάποιες από τις μεθόδους έχουν αναδιατυπωθεί για να επωφεληθούν από τις νέες παραστάσεις (π.χ. OMT), και νέες μέθοδοι έχουν δημιουργηθεί με βάση την UML, όπως η IBM Rational Unified Process (RUP).

Μοντελοποίηση

Είναι σημαντικό να γίνει ξεκάθαρο ότι υπάρχει σαφής διάκριση μεταξύ του μοντέλου UML και του συνόλου των διαγραμμάτων του συστήματος. Τα διαγράμματα αποτελούν μερικές γραφικές αναπαραστάσεις των μοντέλων ενός συστήματος. Από την άλλη τα μοντέλα έχουν ως αναπόσπαστο κομμάτι και την τεκμηρίωση που επεξηγεί και ερμηνεύει τα στοιχεία των μοντέλων και τα διαγράμματα.

Τα διαγράμματα UML αντιπροσωπεύουν τις δύο κύριες διαφορετικές όψεις του μοντέλου - συστήματος:

- Στατική (ή δομική) άποψη. Η άποψη αυτή δίνει έμφαση στη στατική δομή του συστήματος και χρησιμοποιεί τα αντικείμενα, τις ιδιότητες, τις λειτουργίες και τις σχέσεις. Η στατική άποψη περιλαμβάνει τα διαγράμματα κλάσεων και τα διαγράμματα δομής .

- Δυναμική (ή συμπεριφορική) άποψη. Η άποψη αυτή δίνει έμφαση στη δυναμική συμπεριφορά του συστήματος, δείχνοντας συνεργασίες και την αλληλεξάρτηση μεταξύ των αντικειμένων και τις αλλαγές στις εσωτερικές καταστάσεις των αντικειμένων. Η

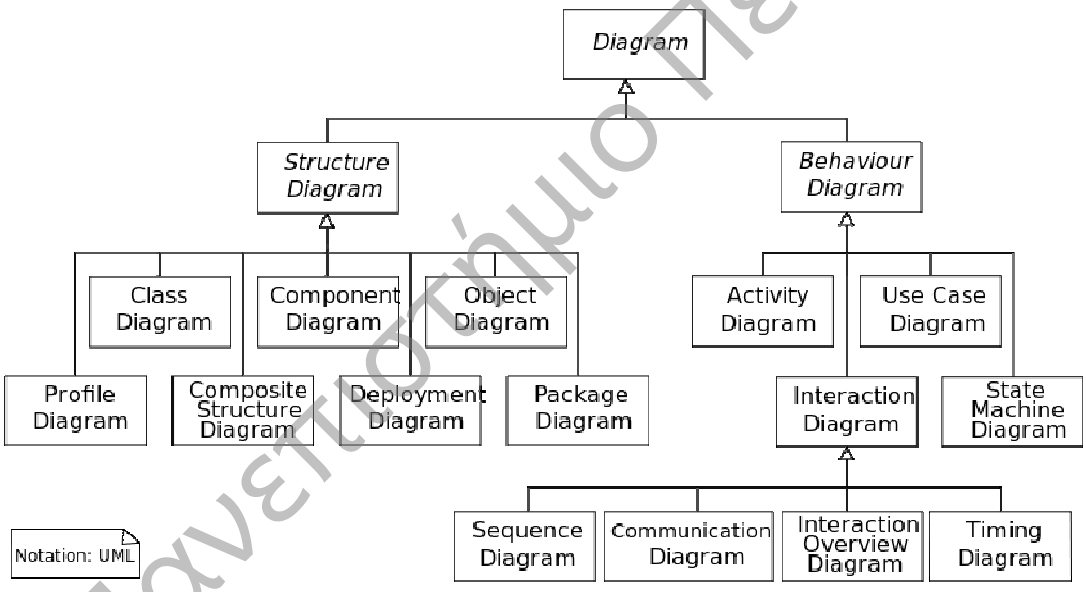
Πανεπιστήμιο Πειραιώς

άποψη αυτή περιλαμβάνει διαγράμματα ακολουθίας, διαγράμματα δραστηριότητας και διαγράμματα μηχανής.

Τα UML μοντέλα χρησιμοποιούν XMI format ούτος ώστε να μπορούν να ανταλλάσσονται μεταξύ των UML εργαλείων.

1.4 ΕΠΙΣΚΟΠΗΣΗ ΔΙΑΓΡΑΜΜΑΤΩΝ:

Η UML 2.4.1 (Αύγουστος 2011) αποτελείται από 14 τύπους διαγραμμάτων που χωρίζονται σε δύο κατηγορίες. Τα δομικά στοιχεία αντιπροσωπεύονται από επτά τύπους διαγραμμάτων, και οι υπόλοιποι επτά αντιπροσωπεύουν γενικούς τύπους συμπεριφοράς, μεταξύ των οποίων τέσσερις που αντιπροσωπεύουν διαφορετικές πτυχές των αλληλεπιδράσεων. Αυτά τα διαγράμματα ταξινομούνται ιεραρχικά όπως φαίνεται στο ακόλουθο διάγραμμα κλάσης.



Διάγραμμα 2: Ομαδοποίηση διαγραμμάτων UML (*πηγή en.wikipedia.org*)

Όψεις (Views) της UML

Μία όψη της UML είναι ένα σύνολο διαγραμμάτων της UML όπου παρουσιάζει μια πλευρά του συστήματος. Ένα σύστημα έχει πολλές πλευρές, όπως για παράδειγμα την πλευρά της λειτουργικότητας (functionality) του συστήματος (τι ακριβώς κάνει το σύστημα), την πλευρά της

Πανεπιστήμιο Πειραιώς

στατικής (static) δομής του και την πλευρά των δυναμικών (dynamic) αλληλεπιδράσεων των στοιχείων του. Από την παρατήρηση όλων των διαφορετικών πλευρών του συστήματος μέσα από τις όψεις της UML θα μπορέσουμε τελικά να αποκτήσουμε μια ολοκληρωμένη εικόνα του συστήματος.
Οι βασικές όψεις της UML είναι:

- Static (Στατική): δείχνει την στατική εικόνα του συστήματος παρουσιάζοντας όλα τα συστατικά στοιχεία (components) που χρησιμοποιούνται στην δημιουργία του συστήματος.
- Functional (Λειτουργική): αναπαριστά τις βασικότερες λειτουργίες του συστήματος και τον τρόπο με τον οποίο υλοποιούνται αυτές.
- Dynamic (Δυναμική): αναπαριστά τον τρόπο αλληλεπιδράσεων (επικοινωνία) μεταξύ των συστατικών στοιχείων του συστήματος.

Η UML δεν περιορίζει τους τύπους στοιχείων σε κάποιον ορισμένο τύπο διαγράμματος. Γενικά κάθε στοιχείο UML μπορεί να εμφανιστεί σχεδόν σε κάθε τύπο διαγραμμάτων. Αυτή η ευελιξία έχει εν μέρει περιοριστεί στις τελευταίες εκδόσεις της UML 2.X. Τα προφίλ UML δίνουν τη δυνατότητα να ορίζουν πρόσθετους τύπους διαγράμματος ή να επεκτείνουν τα ήδη υπάρχοντα διαγράμματα με επιπρόσθετες σημειώσεις. Παραδοσιακά, ένα σχόλιο ή μια σημείωση που εξηγεί τη χρήση, περιορισμό, ή πρόθεση επιτρέπεται σε ένα διάγραμμα UML.

Στη συνέχεια θα παρουσιάσουμε αναλυτικά τις όψεις της UML.

1.4.1 Διαγράμματα Δομής

Τα Διαγράμματα Δομής χρησιμοποιούνται για να περιγράψουν τα πράγματα που πρέπει να περιέχονται στο σύστημα που σχεδιάζεται. Από τη στιγμή που τα διαγράμματα δομής περιγράφουν τη δομή του συστήματος, χρησιμοποιούνται εκτεταμένα στην τεκμηρίωση της αρχιτεκτονικής λογισμικού των συστημάτων λογισμικού. Η όψη αυτή δείχνει μια στατική εικόνα του συστήματος παρουσιάζοντας τα βασικά συστατικά στοιχεία που θα χρησιμοποιηθούν στην δημιουργία του συστήματος. Η στατική εικόνα του συστήματος αφορά στην απλή παρουσίαση των στοιχείων του συστήματος (δηλαδή τις κλάσεις, τα δομικά στοιχεία του συστήματος – τα αρχεία, τον κώδικας κλπ.) καθώς και στον τρόπο με τον οποίο συσχετίζονται μεταξύ τους τα στοιχεία αυτά. Στην στατική όψη δεν παρουσιάζεται η συμπεριφορά των στοιχείων αυτών (οι λειτουργίες που επιτελούν και ο τρόπος δυναμικής επικοινωνίας) αλλά η όψη αυτή χρησιμοποιείται απλώς σαν ένα στατικό σχέδιο (blueprint) της εφαρμογής. Για το λόγο αυτό αναφέρεται σαν στατική όψη και περιλαμβάνει τα παρακάτω διαγράμματα της UML.

- Class Diagram (Διάγραμμα κλάσεων): περιγράφει τη δομή ενός συστήματος, δείχνει τις τάξεις του συστήματος, τα χαρακτηριστικά των τάξεων αυτών, καθώς και τις σχέσεις μεταξύ των τάξεων, δηλαδή παρουσιάζει τις κλάσεις του συστήματος, τα βασικά αντικείμενα από τα οποία αποτελείται το σύστημα.
- Component Diagram (Διάγραμμα μερών): περιγράφει τον τρόπο με τον οποίο ένα σύστημα λογισμικού χωρίζεται σε διάφορα μέρη - συνιστώσες και δείχνει τις εξαρτήσεις μεταξύ αυτών των στοιχείων. Παρουσιάζει τα δομικά στοιχεία του κώδικα του συστήματος, όπως ο εκτελέσιμος κώδικας, αρχεία κλπ.
- Composite Structure Diagram (Διάγραμμα σύνθετης δομής): περιγράφει την εσωτερική δομή μιας κλάσης και τις συνεργασίες που η κλάση αυτή καθιστά δυνατές.

Πανεπιστήμιο Πειραιώς

- Deployment Diagram (Διάγραμμα Ανάπτυξης): περιγράφει το υλικό που χρησιμοποιείται από τις εφαρμογές του συστήματος, το περιβάλλον εκτέλεσης, τα αντικείμενα hardware και παρουσιάζει τα μέσα που χρησιμοποιούνται για την φυσική υλοποίηση του συστήματος όπως οι υπολογιστές, δρομολογητές, εκτυπωτές, μέσα αποθήκευσης και λοιπά μηχανήματα.
- Object Diagram (Διάγραμμα αντικειμένων): δείχνει μια μερική ή πλήρη θέα της δομής ενός παραδείγματος μοντελοποιημένου συστήματος σε συγκεκριμένο - δεδομένο χρόνο.
- Package Diagram (Διάγραμμα Πακέτων): περιγράφει τον τρόπο με τον οποίο ένα σύστημα χωρίζεται σε λογικές ομάδες, και περιγράφει τις εξαρτήσεις μεταξύ αυτών των ομάδων.
- Profile Diagram (Διάγραμμα Προφίλ): λειτουργεί σε επίπεδο μεταμοντέλου για να περιγράψει τα στερεότυπα, ως κλάσεις με το <<στερεότυπο>> στερεότυπο, και τα προφίλ, σαν πακέτα με το στερεότυπο <<προφίλ>>. Η σχέση επέκτασης (συνεχής γραμμή με κλειστό, γεμάτο βέλος) δείχνει ποιο δεδομένο στερεότυπο ενός στοιχείου μεταμοντέλου επεκτείνεται.

1.4.2 Διαγράμματα Συμπεριφοράς

Τα Διαγράμματα Συμπεριφοράς τονίζουν τον τρόπο με τον οποίο πρέπει να συμπεριφέρεται ένα σύστημα - μοντέλο. Δεδομένου ότι τα διαγράμματα συμπεριφοράς αναπαριστούν τον τρόπο συμπεριφοράς ενός συστήματος, χρησιμοποιούνται κατά κόρον για να περιγράψουν τη λειτουργικότητα των συστημάτων λογισμικού. Η όψη αυτή παρουσιάζει την λειτουργική πλευρά του συστήματος δηλαδή δείχνει το πώς λειτουργεί το σύστημα (ποιες είναι οι βασικές λειτουργίες του συστήματος). Η λειτουργικότητα του συστήματος βοηθά στην καταγραφή των απαιτήσεων του συστήματος αφού η όψη αυτή μας βοηθά να κατανοήσουμε τι θέλουμε να κάνει το σύστημα. Τα διαγράμματα της UML που ανήκουν σε αυτή την όψη είναι:

- Activity Diagram (Διάγραμμα Δραστηριοτήτων): περιγράφει την βήμα-προς-βήμα ροή των συστατικών ενός συστήματος. Τα διαγράμματα δραστηριότητας δείχνουν τη συνολική ροή του ελέγχου και παρουσιάζουν τον τρόπο υλοποίησης των βασικών λειτουργιών του συστήματος.
- UML State machine Diagram (UML διάγραμμα κατάστασης μηχανής): περιγράφει τις δυνατές καταστάσεις και τις εναλλαγές των καταστάσεων του συστήματος.
- Use Case Diagram (Διάγραμμα περιπτώσεων χρήσης): περιγράφει τις λειτουργίες που ένα σύστημα μπορεί να παρέχει στους χρήστες από την άποψη των φορέων, τους στόχους τους ως περιπτώσεις χρήσης, και τις εξαρτήσεις μεταξύ των περιπτώσεων χρήσης δηλαδή παρουσιάζει τις βασικές λειτουργίες του συστήματος (τι θέλουμε να

Πανεπιστήμιο Πειραιώς

κάνει το σύστημα). Χρησιμοποιούνται για την καταγραφή των απαιτήσεων του συστήματος.

1.4.3 Διαγράμματα Αλληλεπίδρασης

Τα Διαγράμματα αλληλεπίδρασης, αποτελούν ένα υποσύνολο των διαγραμμάτων συμπεριφοράς. Τονίζουν τη ροή του ελέγχου και τη ροή των δεδομένων μεταξύ των αντικειμένων στο σύστημα. Η όψη αυτή παρουσιάζει τις συμπεριφορές των βασικών συστατικών στοιχείων του συστήματος, δηλαδή οι κλάσεις του συστήματος. Με τον όρο συμπεριφορά εννοούμε τον τρόπο με τον οποίο επικοινωνούν και αλληλεπιδρούν οι κλάσεις προκειμένου να πραγματοποιηθούν οι λειτουργίες του συστήματος. Τα διαγράμματα της UML που ανήκουν στην όψη αλληλεπίδρασης είναι:

- Communication Diagram (Διάγραμμα επικοινωνίας): δείχνει τις αλληλεπιδράσεις μεταξύ των αντικειμένων και των μερών αναφορικά με την αλληλουχία των μηνυμάτων. Αντιπροσωπεύουν ένα συνδυασμό πληροφοριών που λαμβάνονται από τα διαγράμματα κλάσεων, ακολουθιών και περιπτώσεων χρήσης και περιγράφουν τόσο τη στατική δομή όσο και τη δυναμική συμπεριφορά του συστήματος.
- Interaction Overview Diagram (Διάγραμμα επισκόπησης αλληλεπίδρασης): παρέχει μια επισκόπηση του συστήματος στην οποία ο κάθε κόμβος αναπαριστά διαγράμματα επικοινωνίας και παρουσιάζεται η αλληλεπίδραση των αντικειμένων μέσω της ανταλλαγής μηνυμάτων σε αύξουσα σειρά.
- Sequence Diagram (Διάγραμμα ακολουθίας): δείχνει τον τρόπο επικοινωνίας μεταξύ των αντικειμένων από την άποψη της ακολουθίας των μηνυμάτων και επιπλέον την διάρκεια ζωής των αντικειμένων σε σχέση με αυτά τα μηνύματα, δηλαδή παρουσιάζει την αλληλεπίδραση των αντικειμένων μέσω της ανταλλαγής μηνυμάτων στην διάρκεια του χρόνου
- Timing Diagrams (Διαγράμματα χρονισμού): αποτελούν έναν ειδικό τύπο των διαγραμμάτων αλληλεπίδρασης όπου εστιάζει στους χρονικούς περιορισμούς και παρουσιάζει τις διάφορες καταστάσεις, που έχουν οι βασικές κλάσεις της εφαρμογής όταν εκτελείται μια καθορισμένη λειτουργία.

Οι όψεις της UML όπως παρουσιάστηκαν παραπάνω δεν είναι ανεξάρτητες η μια από την άλλη. Προκειμένου να σχηματίσουμε μια ολοκληρωμένη εικόνα για το σύστημα πρέπει να μελετήσουμε και να αναλύσουμε όλες τις όψεις του συστήματος και να τις χρησιμοποιήσουμε ταυτόχρονα για να έχουμε μια ολοκληρωμένη περιγραφή του συστήματος από όλες τις διαφορετικές οπτικές γωνίες.

1.5 ΑΝΑΛΥΤΙΚΗ ΠΕΡΙΓΡΑΦΗ ΔΙΑΓΡΑΜΜΑΤΩΝ

Όπως ήδη αναφέραμε παραπάνω οι όψεις της UML συνδυάζονται μεταξύ τους για να παρουσιάσουν όσο το δυνατόν πιο ολοκληρωμένα όλες τις πλευρές του συστήματος. Η χρήση κάθε όψης απαιτεί την χρησιμοποίηση κάποιων διαγραμμάτων της UML που ανήκουν στην συγκεκριμένη όψη. Αυτό σημαίνει ότι προκειμένου να ολοκληρώσουμε τις όψεις της UML

Πανεπιστήμιο Πειραιώς

πρέπει να δημιουργήσουμε διάφορα διαγράμματα τα οποία θα χρησιμοποιήσουμε σε συνδυασμό για να δημιουργήσουμε μια ολοκληρωμένη εικόνα του συστήματος.

Διαγράμματα της UML

Τα βασικά διαγράμματα που θα χρησιμοποιήσουμε από την UML είναι τα εξής:

- Use case
- Activity
- Class
- Sequence
- Collaboration
- Statechart
- Component
- Deployment

Κάθε εμπλεκόμενος στην ανάπτυξη του συστήματος χρησιμοποιεί διαγράμματα της UML. Κάθε ένα από τα διαγράμματα, κατ' αντιστοιχία με τις όψεις της UML, παρουσιάζει ένα συγκεκριμένο χαρακτηριστικό του συστήματος (δομή συστήματος, συμπεριφορά στοιχείων που ανήκουν στο σύστημα και επικοινωνία μεταξύ τους).

Ακολουθεί αναλυτική παρουσίαση των κυριότερων διαγραμμάτων της UML.

1.5.1 Use Case Διαγράμματα

Τα Use Case διαγράμματα είναι από τα πιο σημαντικά διαγράμματα της UML και χρησιμοποιούνται πάντα κατά την διαδικασία της ανάλυσης του συστήματος. Το Use Case διάγραμμα είναι ένα διάγραμμα που αναπαριστά μια γενική όψη του συστήματος από την πλευρά των χρηστών, δείχνει δηλαδή όλες τις λειτουργίες του συστήματος όπως τις αντιλαμβάνεται ο εξωτερικός παρατηρητής του συστήματος. Με τα Use Case διαγράμματα παρουσιάζονται οι απαιτήσεις (requirements) που έχουν οι χρήστες από το σύστημα και αναπαριστάται η λειτουργικότητα του συστήματος (functionality).

Τα βασικά στοιχεία (elements) του Use Case διαγράμματος είναι:

· **Actors**

Οι actors αποτελούν τους χρήστες του συστήματος που είναι έξω από αυτό και επικοινωνούν με το σύστημα. Οι actors αφού είναι έξω από το σύστημα διαμορφώνουν το εξωτερικό περιβάλλον ή την περιφέρεια του συστήματος και ορίζουν την εμβέλειά (scope) του. Οι actors ορίζουν το εξωτερικό περιβάλλον και διαχωρίζουν το σύστημα που σχεδιάζουμε από αυτό. Οι actors αναπαρίστανται σαν φιγούρες όπως φαίνεται παρακάτω (stick figures).

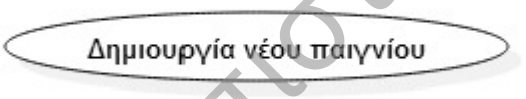
Πανεπιστήμιο Πειραιώς

· Use Case

Το use case διάγραμμα χρησιμοποιείται για να αναπαραστήσει μερικά ή ολικά τμήματα της λειτουργικότητας που προσφέρεται από το σύστημα προς τον κάθε χρήστη του συστήματος. Δείχνουν τις βασικές λειτουργίες (key features) που επιτελεί το σύστημα. Ένας εύκολος τρόπος για να εντοπίσουμε τα use cases είναι για τον κάθε χρήστη να σκεφτούμε τι θέλει να κάνει με το σύστημα. Συνήθως τα use cases εντοπίζονται μέσα από συνεντεύξεις και συνομιλίες με τους χρήστες του συστήματος, οι οποίοι περιγράφουν όσο το δυνατόν πιο αναλυτικά τι θέλουν από το σύστημα χωρίς να τους ενδιαφέρει ο τρόπος υλοποίησης των λειτουργιών που θα προσφέρονται από αυτό.

Με ένα use case δεν περιγράφουμε τον τρόπο λειτουργίας του συστήματος αλλά το τι μπορεί να κάνει το σύστημα. Χρησιμοποιείται κυρίως για να καθορίσει την συμπεριφορά του συστήματος χωρίς να δίνει πληροφορίες για την εσωτερική δομή του. Το κάθε use case του διαγράμματος δείχνει τις λειτουργίες που είναι σε θέση να εκτελεί το σύστημα καθώς αλληλεπιδρά με τους χρήστες (actors) της.

Τα use cases τα αναπαριστούμε με ένα οβάλ σχήμα που περιέχει μια μικρή περιγραφή της λειτουργίας του use case αυτού.



Τα use cases χρησιμοποιούνται στην ανάλυση του συστήματος και είναι πολύ σημαντικά για την σωστή και ολοκληρωμένη καταγραφή των απαιτήσεων του συστήματος και στη συνέχεια για το σωστό σχεδιασμό του αφού για να ξεκινήσουμε χρειαζόμαστε μια ολοκληρωμένη εικόνα για το τι θέλουμε και τι πρέπει να κάνει το σύστημα.

Έτσι στα πρώτα βήματα του σχεδιασμού του συστήματος προσπαθούμε να καταγράψουμε τις περισσότερες από τις λειτουργίες με την χρήση των use cases. Φυσικά καθώς η ανάπτυξη προχωρά μπορεί να εμφανιστούν νέες λειτουργίες και να καταγραφούν στα Use Case διαγράμματα, που έχουν ήδη δημιουργηθεί.

· Use case σενάρια

Με ένα Use Case διάγραμμα συνήθως δημιουργείται και ένας αριθμός από use case σενάρια που αναπαριστούν αν όχι όλες τότε τις περισσότερες περιπτώσεις (σενάρια) οι οποίες είναι πιθανό να συμβούν κατά τη διάρκεια εκτέλεσης της εφαρμογής. Με αυτό τον τρόπο καταγράφουμε την συμπεριφορά του συστήματος σε διαφορετικές περιπτώσεις προκειμένου να λάβουμε τα απαραίτητα συμπεράσματα σχετικά με την πορεία του σχεδιασμού του συστήματος. Τα use case σενάρια αποτελούν τη βάση για την δημιουργία άλλων διαγραμμάτων της UML (για

Πανεπιστήμιο Πειραιώς

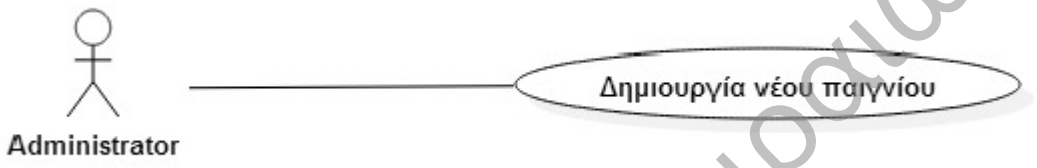
παράδειγμα Activity διαγράμματα) καθώς επίσης χρησιμοποιούνται και για τον τελικό έλεγχο του συστήματος αναφορικά με την σωστή συμπεριφορά του στις περιπτώσεις, που αναφέρονται στα σενάρια αυτά.

Οι συσχετίσεις (relationships) που χρησιμοποιούνται στα Use Case διαγράμματα είναι οι παρακάτω:

· *Association relationship*

Ο ρόλος του είναι να δείξει ότι ένας actor επικοινωνεί – χρησιμοποιεί ένα use case.

Αναπαρίσταται με μια γραμμή από τον actor προς το use case με το οποίο συνδέεται.

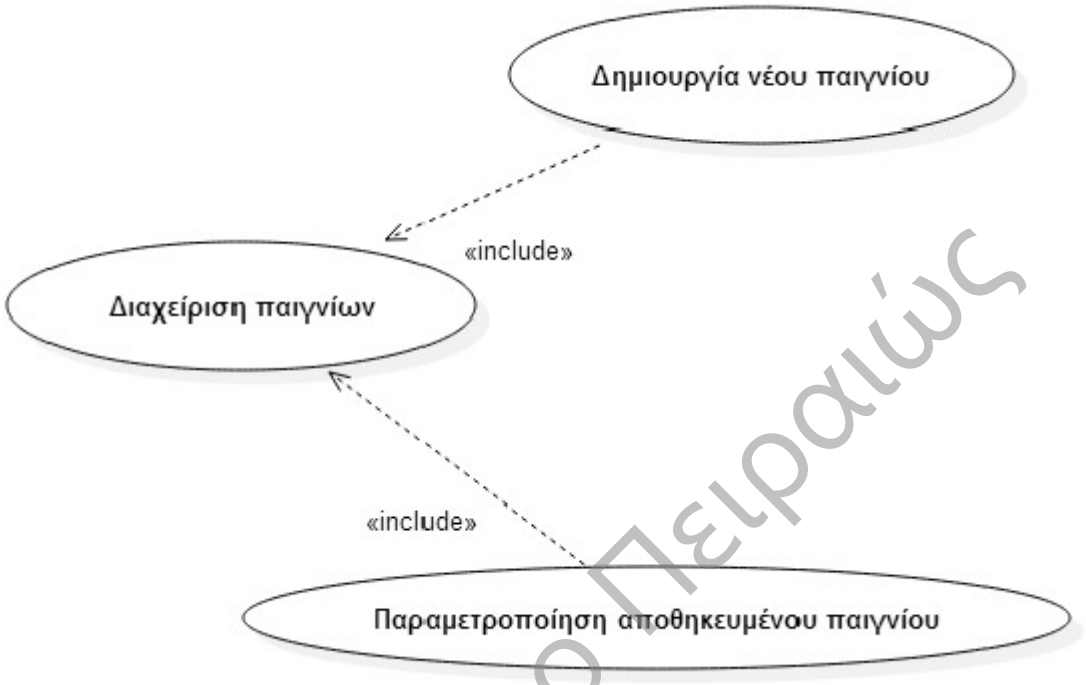


· *Include relationship*

Πολλές φορές κάποιο use case περιλαμβάνει κάποια βήματα, τα οποία είναι κοινά και σε άλλα use cases. Προκειμένου να εξαλειφθεί αυτή η επανάληψη, τα επαναλαμβανόμενα βήματα τα αναπαριστούμε με ένα μοναδικό use case και τα υπόλοιπα use cases συσχετίζονται με αυτό. Η συσχέτιση include χρησιμοποιείται προκειμένου να δείξει ότι κάποια use cases χρησιμοποιούν κάποια λειτουργία (functionality) ενός άλλου use case. Αποτυπώνεται με μια διακεκομμένη γραμμή όπως φαίνεται παρακάτω με την λέξη <<include>> και την μορφή ανοικτού βέλους, με φορά από τα use cases που χρησιμοποιούν την κοινή λειτουργία προς το use case που περιέχει την κοινή λειτουργία.

Για παράδειγμα, έχουμε μια λειτουργία του συστήματος που ονομάζεται Δημιουργία νέου παιγνίου και χρησιμοποιείται για την δημιουργία νέου παιγνίου στο σύστημα. Επίσης έχουμε μια λειτουργία που ονομάζεται Παραμετροποίηση αποθηκευμένου παιγνίου, η οποία χρησιμοποιείται για να παραμετροποιήσουμε ένα αποθηκευμένο παίγνιο. Οι δύο αυτές λειτουργίες για να λειτουργήσουν πρέπει πρώτα να βρουν το σωστό παίγνιο, δηλαδή πρέπει να χρησιμοποιήσουν την λειτουργία του συστήματος Διαχείριση παιγνίων. Σε μορφή κώδικα αυτή η συσχέτιση παρουσιάζεται με την κλήση της συνάρτησης Διαχείριση παιγνίων από δύο σημεία (από το σημείο της δημιουργίας και από το σημείο της παραμετροποίησης). Το χαρακτηριστικό της συσχέτισης αυτής είναι ότι η λειτουργία Παραμετροποίηση παιγνίων θα χρησιμοποιεί πάντα την λειτουργία Διαχείριση παιγνίων, που κάνει include.

Πανεπιστήμιο Πειραιώς

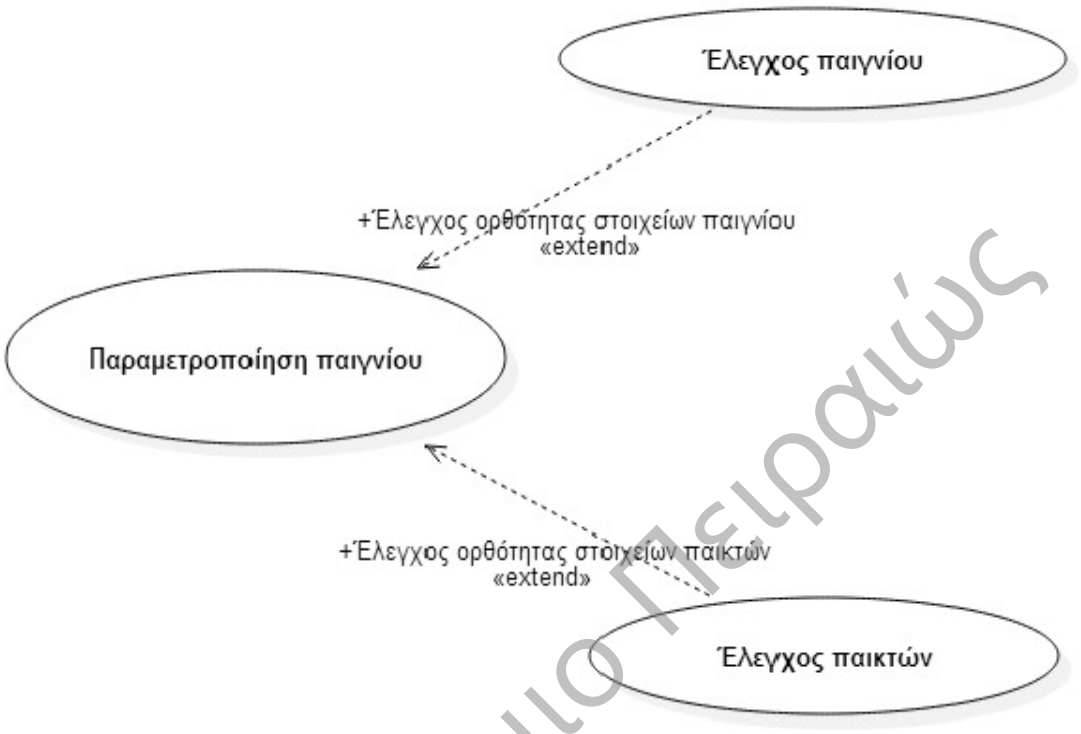


· *Extend relationship*

Η συσχέτιση extend δείχνει ότι ένα use case χρησιμοποιεί τις λειτουργίες κάποιου άλλου use case με το να προσθέτει (ή καλύτερα να επεκτείνει – extend) τις λειτουργίες του. Η συσχέτιση μεταξύ των δυο use cases δημιουργείται κάτω από ορισμένες συνθήκες, δηλαδή με την προϋπόθεση ότι ικανοποιείται κάποιο κριτήριο, το ένα use case χρησιμοποιεί τις λειτουργίες του άλλου use case. Αναπαρίσταται με μία διακεκομμένη γραμμή με ανοιχτό βέλος, με φορά από το use case που παρέχει την λειτουργία προς το use case που δέχεται την λειτουργία, με την λέξη <<extend>>.

Για παράδειγμα, έχουμε την λειτουργία Παραμετροποίηση παιγνίου, η οποία ενημερώνει με καινούργια στοιχεία ένα αποθηκευμένο παίγνιο. Η παραμετροποίηση του παιγνίου εκτός των άλλων παραμετροποιεί στοιχεία για ένα παίγνιο και έναν παίκτη. Η λειτουργία στην περίπτωση που αλλάζει στοιχεία του παιγνίου, για να εξασφαλιστεί η ακεραιότητα των στοιχείων, θα πρέπει να χρησιμοποιήσει τις λειτουργίες Έλεγχος στοιχείων παιγνίου και Έλεγχος στοιχείων παικτών για τον έλεγχο την ορθότητας τους προτού κάνει την ενημέρωση του παιγνίου.

Πανεπιστήμιο Πειραιώς

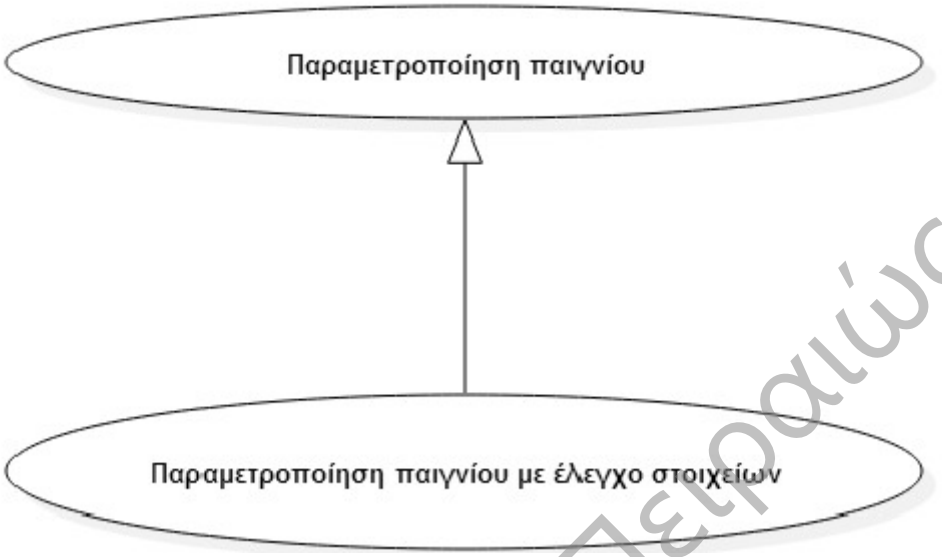


· *Generalization relationship*

Χρησιμοποιούμε την Generalization Relationship για να αναπαραστήσουμε την κληρονομικότητα (inheritance) μεταξύ των use cases. Κληρονομικότητα μεταξύ use cases σημαίνει ότι ορισμένα use cases κληρονομούν κάποιες από τις λειτουργίες τους από ένα use case-γονέα. Αναπαρίσταται με μια ευθεία γραμμή με ένα κλειστό βέλος, που δείχνει πάντα προς το use case-πατέρα.

Για παράδειγμα, έχουμε το use case Παραμετροποίηση παιγνίου και το use case Παραμετροποίηση παιγνίου με έλεγχο στοιχείων, όπως παρουσιάστηκε παραπάνω. Το δεύτερο use case κληρονομεί όλα τα στοιχεία του πρώτου και προσθέτει επιπλέον λειτουργίες (έλεγχο στοιχείων παιγνίου και παικτών).

Πανεπιστήμιο Πειραιώς



Επίσης generalization relationship (κληρονομικότητα) μπορεί να υπάρξει και μεταξύ των actors.

Χρήση των Use Case διαγραμμάτων

Τα Use Case διαγράμματα χρησιμοποιούνται για την ανάλυση, σχεδίαση και ανάπτυξη του συστήματος. Ένα Use Case διάγραμμα όλες τις βασικές λειτουργίες του συστήματος όπως φαίνονται σε έναν εξωτερικό παρατηρητή. Με τα Use Case διαγράμματα παρουσιάζονται οι απαιτήσεις που έχουν οι χρήστες από το σύστημα.

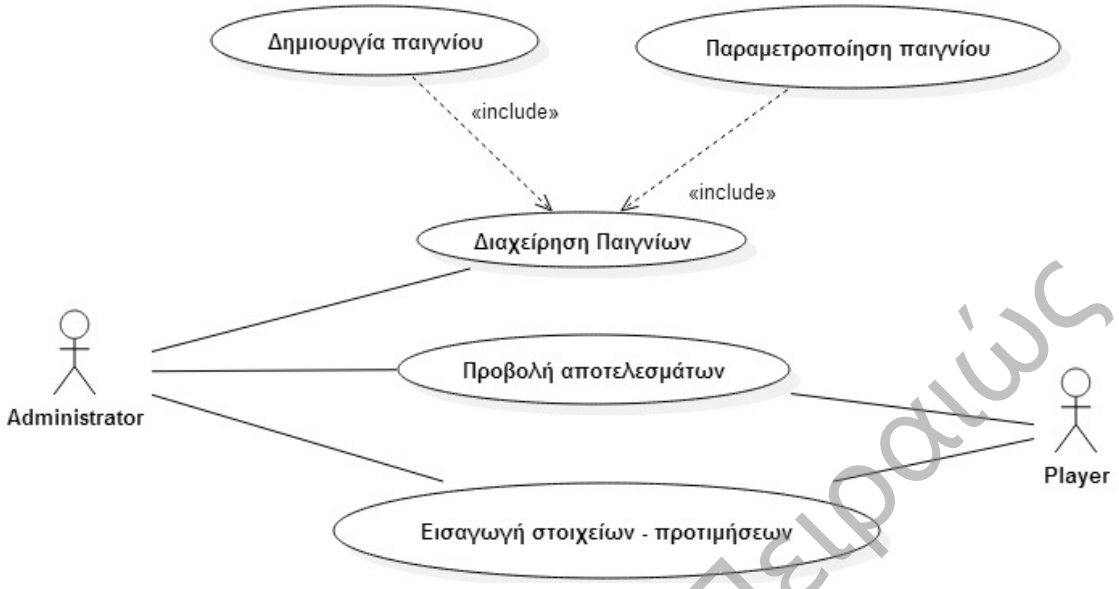
Η προβολή των συσχετίσεων μεταξύ των actors και του συστήματος μπορεί να βοηθήσει στην κατασκευή της εφαρμογής και στον έλεγχο των λειτουργιών της με την χρήση διαφόρων use case σεναρίων (τα οποία θα αναπαρασταθούν στη συνέχεια με την βοήθεια των Sequence και των Activity διαγραμμάτων).

Τέλος, μέσα από τα Use Case διαγράμματα προκύπτει η τεκμηρίωση (documentation) του συστήματος αφού ουσιαστικά φαίνεται ποιοί χρήστες και με ποιόν τρόπο θα χρησιμοποιήσουν το σύστημα.

Παράδειγμα Use Case διαγράμματος

Στη συνέχεια παρουσιάζεται ένα παράδειγμα Use Case διαγράμματος, που παρουσιάζει ένα σύστημα διαχείρισης ενός παιγνίου.

Πανεπιστήμιο Πειραιώς



Παράδειγμα use case σεναρίου

Για να περιγράψουμε την λειτουργία ενός use case χρησιμοποιούμε ένα σενάριο που περιλαμβάνει μια σειρά από βήματα τα οποία ακολουθούμε από την αρχή ως το τέλος της λειτουργίας του. Στα βήματα αυτά περιλαμβάνονται όλες οι δυνατές περιπτώσεις που μπορεί να συμβούν κατά την διάρκεια της εκτέλεσης της λειτουργίας αυτής. Κάθε διαφορετικό μονοπάτι από την αρχή ως το τέλος αποτελεί ένα διαφορετικό σενάριο. Στη συνέχεια παρουσιάζεται ένα σενάριο για την λειτουργία του συστήματος διαχείρισης παιγνίου.

Use Case: Διαχείριση παιγνίου (Βασικό Σενάριο)

1. Εισαγωγή Administrator στο σύστημα
2. Δημιουργία νέου παιγνίου
   2.1 Όνομα παιγνίου
   2.2 Ορισμός αριθμού παικτών
   2.3 Ορισμός τύπου παιγνίου
   2.4 Ορισμός παικτών
3. Παραμετροποίηση παιγνίου
   3.1 Επιλογή παιγνίου
   3.2 Τροποποίηση ονόματος παιγνίου
   3.3 Τροποποίηση αριθμού παικτών
   3.4 Τροποποίηση τύπου παιγνίου

Πανεπιστήμιο Πειραιώς

3.5 Τροποποίηση ορισμού παικτών

4. Έξοδος από το σύστημα

Όπως βλέπουμε από το παραπάνω βασικό σενάριο μπορούμε να έχουμε παραπάνω από ένα μονοπάτια από την αρχή ως το τέλος της λειτουργίας. Κάθε ένα από τα μονοπάτια αυτά αποτελεί και ένα εναλλακτικό σενάριο εκτέλεσης της λειτουργίας.

1.5.2 Activity Διαγράμματα

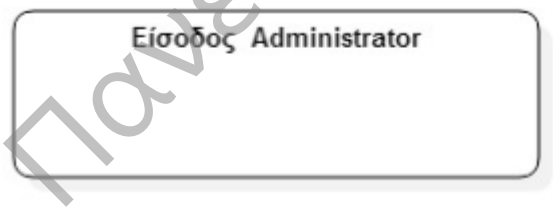
Όπως τα Use Case διαγράμματα δείχνουν τον σκοπό (goal) μιας λειτουργίας του συστήματος, δηλαδή περιγράφουν τι κάνει το σύστημα, τα Activity διαγράμματα δείχνουν τον τρόπο με τον οποίο επιτυγχάνεται η λειτουργία αυτή, δηλαδή πως θα γίνει η εκτέλεσή της. Με τα Activity διαγράμματα μπορούμε να αναπαραστήσουμε τον τρόπο υλοποίησης των διάφορων workflows στο σύστημα (η ροή δηλαδή της εργασίας στην κάθε λειτουργία), ποιες αποφάσεις (decision paths) πρέπει να ακολουθηθούν, από τις διάφορες περιπτώσεις που υπάρχουν για να ολοκληρωθεί η λειτουργία που περιγράφεται και πως ολοκληρώνεται το workflow.

Συνήθως τα Activity διαγράμματα χρησιμοποιούνται για να αναπαραστήσουν κάποιο use case σενάριο εκτέλεσης μιας λειτουργίας του συστήματος, που έχει δημιουργηθεί από κάποιο Use Case διάγραμμα.

Τα βασικά στοιχεία (elements) ενός Activity διαγράμματος είναι:

· **Activities (Δραστηριότητες)**

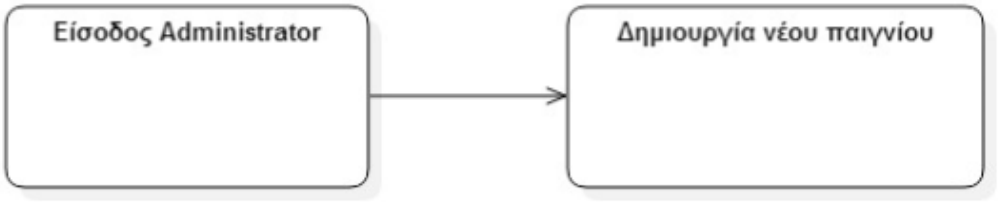
Activity (δραστηριότητα) ονομάζεται το κάθε βήμα της συνολικής διαδικασίας στο οποίο πρέπει να εκτελεστεί κάποια λειτουργία προκειμένου να προχωρήσουμε σε ένα άλλο activity. Τα activitys αναπαρίστανται με ένα ορθογώνιο με στρογγυλεμένες άκρες το οποίο περιέχει το όνομα της δραστηριότητας.



· **Transition (Μετάβαση)**

Transition (μετάβαση) ονομάζεται η μεταφορά στο επόμενο βήμα, η μετάβαση δηλαδή σε ένα άλλο activity. Αναπαρίσταται με ένα ανοιχτό βέλος, που δείχνει προς την επόμενη δραστηριότητα της διαδικασίας.

Πανεπιστήμιο Πειραιώς



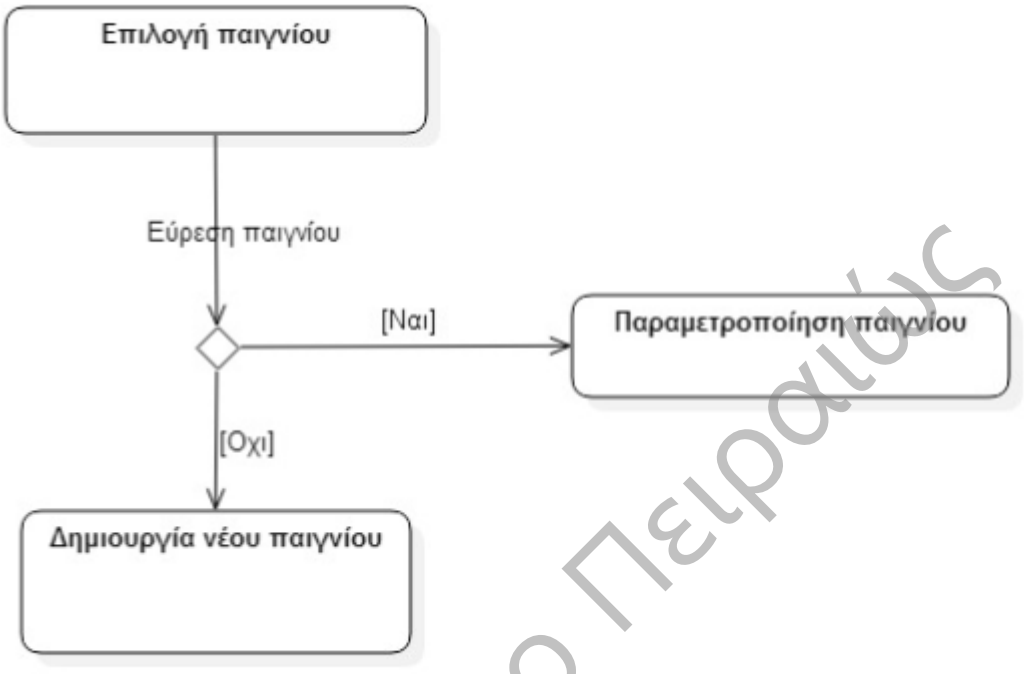
· **Guard Condition (Συνθήκες Ελέγχου)**

Για να μεταβούμε από το ένα activity στο άλλο πολλές φορές πρέπει να είναι αληθής κάποια συγκεκριμένη συνθήκη. Η guard condition (συνθήκη ελέγχου) αναπαρίσταται με το κείμενο της συνθήκης μέσα σε αγκύλες ([..]). Αυτό τοποθετείται πάνω από το βέλος της μετάβασης (transition).

· **Decision (Σημεία Απόφασης)**

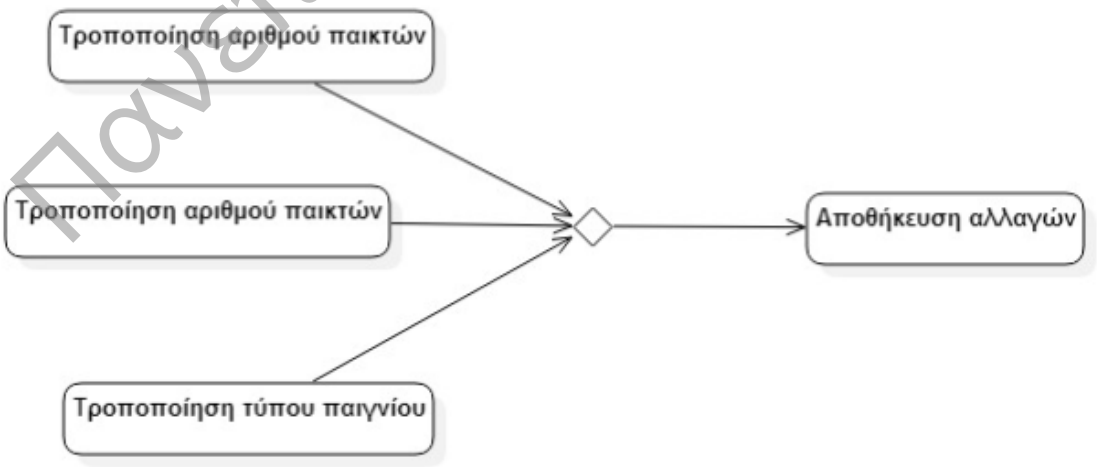
Decision ονομάζεται το σημείο απόφασης στο οποίο καταλήγουν ορισμένα transitions και πρέπει να ακολουθήσουν αναλόγως της συνθήκης ένα συγκεκριμένο μονοπάτι. Η κάθε δυνατή έξοδος του decision χαρακτηρίζεται από μια guard condition, δηλαδή μια συνθήκη που πρέπει να είναι αληθής ούτως ώστε να επιλεγεί η συγκεκριμένη έξοδος. Οι συνθήκες στις εξόδους του decision πρέπει να είναι αμοιβαίως αποκλειόμενες, δηλαδή η αληθής τιμή της μιας να σημαίνει ψευδής τιμή των υπολοίπων εξόδων προκειμένου να υπάρχει κάθε φορά αποκλειστικά μια έξοδος από το σημείο απόφασης. Το decision αναπαρίσταται με ρόμβο.

Πανεπιστήμιο Πειραιώς



· **Merge Point (Σημεία Συγχώνευσης)**

Merge point είναι το σημείο στο οποίο καταλήγουν περισσότερα από ένα μονοπάτια και από το οποίο ξεκινά ένα transition προς κάποιο activity. Τα μονοπάτια τα οποία οδηγούν στο merge point πρέπει απαραιτήτως να είναι αμοιβαίως αποκλειόμενα, δηλαδή να είναι ανεξάρτητα το ένα από το άλλο. Το merge point αναπαρίσταται με ρόμβο.



· **Start and End (Αρχή – Τέλος διαγράμματος)**

Μια μαύρη κουκκίδα δείχνει την έναρξη του activity diagram ενώ μια μικρή μαύρη μικρότερη κουκκίδα μέσα σε να κύκλο αναπαριστά το τέλος του activity diagram.

Πανεπιστήμιο Πειραιώς

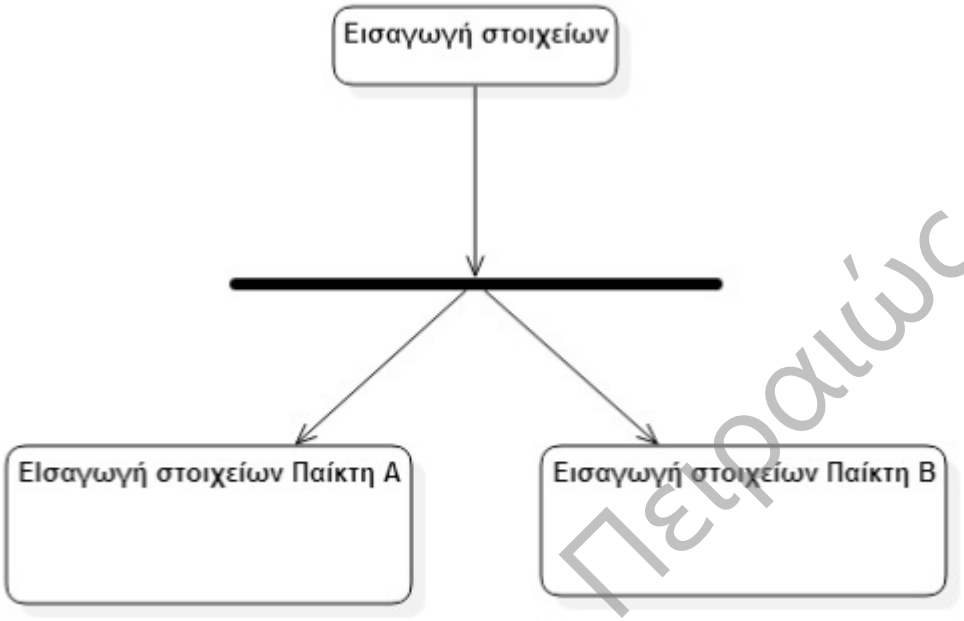
Σε κάθε activity diagram υπάρχει πάντα μια αρχή και τουλάχιστον ένα τέλος.

· **Concurrency (Ταυτόχρονη Επεξεργασία)**

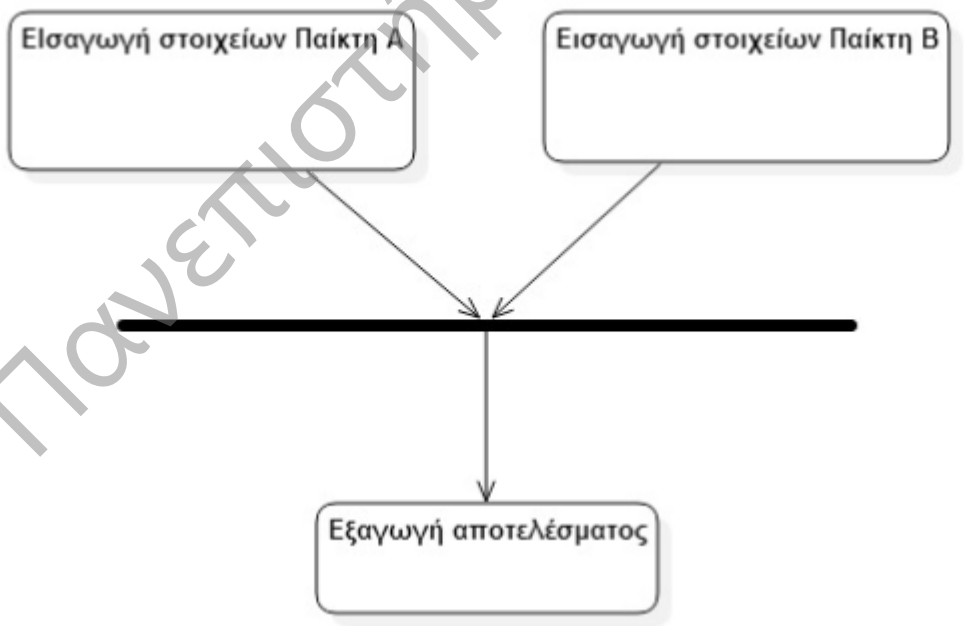
Η ταυτόχρονη επεξεργασία (concurrency) αναπαρίσταται με μια παχιά μαύρη γραμμή. Ταυτόχρονη επεξεργασία υπάρχει όταν εκτελούνται ταυτόχρονα διαφορετικές λειτουργίες οι οποίες καταλήγουν σε κάποιο κοινό σημείο. Η διαφορά σε σχέση με το σημείο συγχώνευσης (merge point) που είδαμε παραπάνω είναι ότι εδώ δεν είναι υποχρεωτικό να έχουμε αμοιβαίως αποκλειόμενες λειτουργίες που καταλήγουν σε ένα κοινό σημείο αλλά έχουμε ένα συγχρονισμό λειτουργιών που εκτελούνται παράλληλα. Υπάρχουν δυο τύποι ταυτόχρονης επεξεργασίας:

Fork: ταυτόχρονη επεξεργασία όπου μια δραστηριότητα ξεκινάει πολλές διαφορετικές παράλληλες δραστηριότητες ταυτόχρονα (αναπαράσταση fork)

Πανεπιστήμιο Πειραιώς



Synchronization: ταυτόχρονη δραστηριότητα πολλών δραστηριοτήτων που εκτελούνται παράλληλα και συγχρονίζονται σε μια (αναπαράσταση synchronization)



Πανεπιστήμιο Πειραιώς

Χρήση Activity διαγραμμάτων

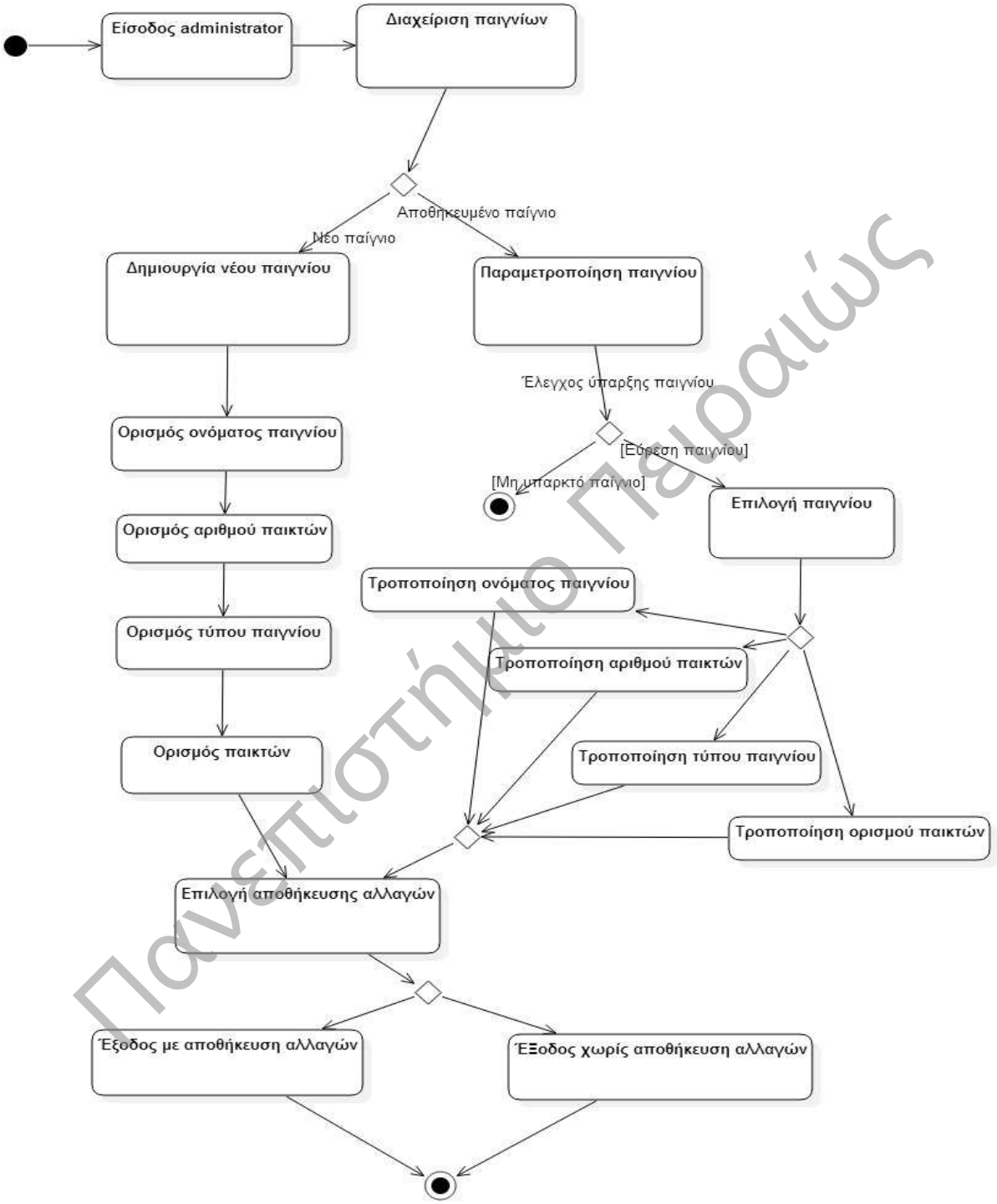
Τα Activity διαγράμματα χρησιμοποιούνται για να αναπαραστήσουν τον τρόπο εκτέλεσης μιας λειτουργίας του συστήματος, και παρουσιάζονται τα βήματα που απαιτούνται για την εκτέλεση

της συγκεκριμένης λειτουργίας. Τα Activity διαγράμματα βοηθούν τον χρήστη να μπορέσει να δει την ροή της εργασίας (workflow) γεγονός που τον βοηθάει να ελέγξει όλες τις πιθανές συνθήκες που πρέπει να συμπεριλάβει στην εφαρμογή.

Παράδειγμα Activity διαγράμματος

Στη συνέχεια παρουσιάζεται ένα Activity διάγραμμα, που περιγράφει το σενάριο που παρουσιάστηκε παραπάνω για την λειτουργία της διαχείρισης των παιγνίων.

Πανεπιστήμιο Πειραιώς



Πανεπιστήμιο Πειραιώς

1.5.3 Class Διαγράμματα

Τα Class Διαγράμματα αποτελεί τον βασικό κορμό κάθε μοντέλου που περιλαμβάνει η αντικειμενοστραφής ανάλυση. Ένα Class διάγραμμα περιγράφει στατικά τη δομή της εφαρμογής και περιλαμβάνει όλες τις οντότητες (τα αντικείμενα – κλάσεις) που θα χρησιμοποιηθούν σε αυτή. Συνήθως η δημιουργία ενός Class διαγράμματος ακολουθεί τη δημιουργία των use case και των activity διαγραμμάτων και έχει προέλθει μέσα από συνεντεύξεις με τους πελάτες της εφαρμογής προκειμένου να ανακαλυφθούν οι κύριες οντότητες της εφαρμογής.

Από τα Use Case διαγράμματα προκύπτουν οι απαιτήσεις σε αντικείμενα (objects) που χρειάζονται για να επιτευχθούν οι στόχοι (goals) του συστήματος. Μέσα από την ανάλυση των διαγραμμάτων αυτών προσπαθούμε να ανακαλύψουμε τις κύριες οντότητες της εφαρμογής. Τα Activity διαγράμματα χρησιμεύουν στον καθορισμό της συμπεριφοράς (behavior) των αντικειμένων δηλαδή καθορίζουν το πως θα λειτουργούν τα αντικείμενα και ποια θα είναι η ροή της εργασία καθ' όλη τη διάρκεια της εφαρμογής προκειμένου να επιτευχθεί ο σκοπός της εφαρμογής.

Τα Class διαγράμματα περιγράφουν την κατασκευαστική δομή (structural view) του συστήματος, δηλαδή τους τύπους όλων των αντικειμένων που θα χρησιμοποιηθούν στο σύστημα, παρουσιάζοντας την στατική δομή των αντικειμένων αυτών και περιγράφει το είδος των συσχετίσεων που υπάρχουν μεταξύ των αντικειμένων αυτών. Με την ανάλυση του Class διαγράμματος βλέπουμε τις κλάσεις που δημιουργούνται για την υλοποίηση της εφαρμογής. Τα βασικότερα δομικά στοιχεία των Class διαγραμμάτων είναι:

- Οι κλάσεις (δομή και συμπεριφορά τους – μέθοδοι κλάσεων).
- Οι συσχετίσεις μεταξύ των κλάσεων.
- Η πολλαπλότητα (multiplicity) και navigation (ροή μέσα στις κλάσεις).

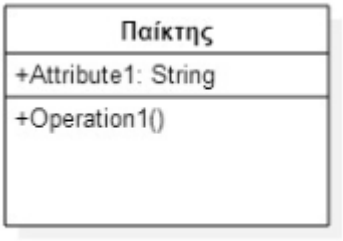
Class (Κλάση)

Η κλάση αντιπροσωπεύει ένα σύνολο αντικειμένων με παρόμοια δομή, συμπεριφορά και συσχετίσεις. Είναι το πρότυπο (pattern) με το οποίο τα διάφορα αντικείμενα παράγονται κατά την διάρκεια εκτέλεσης της εφαρμογής. Τα αντικείμενα που παράγονται από μια κλάση θεωρούνται στιγμιότυπα της κλάση αυτής. Για παράδειγμα μπορούμε να έχουμε την κλάση Παίγνιο και από την κλάση αυτή να προκύπτουν τα στιγμιότυπα Παίγνιο μηδενικού αθροίσματος, Συνεργατικό παίγνιο, Παίγνιο κανονικής μορφής, Παίγνιο πεπερασμένης μορφή κλπ.

Η κλάση αποτελεί το βασικό δομικό στοιχείο του αντικειμενοστραφούς σχεδιασμού και ανάλυσης. Οι προγραμματιστές επικεντρώνονται στην προσπάθεια να δημιουργήσουν το μοντέλο του συστήματος αποτελούμενο από κλάσεις, οι οποίες περιέχουν τα χαρακτηριστικά (attributes) και τις μεθόδους (operations) αντίστοιχα.

Πανεπιστήμιο Πειραιώς

Όπως φαίνεται παρακάτω στην UML η αναπαράσταση μιας κλάσης γίνεται με ένα ορθογώνιο, το οποίο αποτελείται από 3 τμήματα.



Τα βασικά τμήματα της κλάσης είναι:

- **Τμήμα Ονόματος Κλάσης**: στο τμήμα αυτό γράφεται το όνομα της κλάσης.
- **Τμήμα Χαρακτηριστικών Κλάσης**: στο τμήμα αυτό δηλώνονται τα χαρακτηριστικά της κλάσης, οι μεταβλητές δηλαδή που ανήκουν στην κλάση. Κάθε χαρακτηριστικό ορίζεται με το όνομα και τον τύπο του (π.χ. Integer, String, Date κλπ.).

Προαιρετικά για κάθε χαρακτηριστικό ορίζονται και τα εξής:

- **Visibility**: ορίζεται η ορατότητα του χαρακτηριστικού της κλάσης σε σχέση με τα υπόλοιπα στοιχεία και σε ποια σημεία του προγράμματος είναι διαθέσιμο το συγκεκριμένο χαρακτηριστικό. Ορίζει δηλαδή ποιες άλλες κλάσεις μπορούν να «δουν» το χαρακτηριστικό αυτό. Οι διαθέσιμες επιλογές που έχει ο χρήστης για να ορίσει το visibility του χαρακτηριστικού είναι:

Συμβολισμός	**Ονομασία**	**Περιγραφή**
+	Public:	Ορατό από άλλες κλάσεις σε ολόκληρο το πρόγραμμα
-	Private:	Ορατό μόνο μέσα στην ίδια την κλάση
#	Protected:	Ορατό μόνο από υπο-κλάσεις (κλάσεις παιδιά) της κλάσης
~	Package:	Ορατό από άλλες κλάσεις του ίδιου package της κλάσης

Αρχική Τιμή: δηλώνεται η αρχική τιμή του χαρακτηριστικού της κλάσης.

Η γενική μορφή δήλωσης ενός attribute είναι:

Ορατότητα Όνομα_Χαρακτηριστικού: Τύπος = Αρχική_Τιμή

Για παράδειγμα: + myGender: String = Male

· **Τμήμα Λειτουργιών Κλάσης**: στο τμήμα αυτό δηλώνονται οι λειτουργίες της κλάσης, δηλαδή οι συναρτήσεις της κλάσης. Όπως και με τα χαρακτηριστικά έτσι και στο τμήμα αυτό δηλώνεται το όνομα της μεθόδου, ο επιστρεφόμενος τύπος της, τυχόν ορίσματα (parameter list) που έχει η μέθοδος καθώς και η ορατότητα.

Η γενική μορφή δήλωσης μιας μεθόδου είναι:

Ορατότητα Όνομα_Μεθόδου(Όνομα_Ορίσματος: Τύπος_Ορίσματος, …): Τύπος

Για παράδειγμα: + CalculateStrategy (apodosi: integer): integer

Πανεπιστήμιο Πειραιώς

Relationships (Συσχετίσεις)

Κάθε κλάση μπορεί να επικοινωνεί και να συσχετίζεται με τις υπόλοιπες κλάσεις. Υπάρχουν διάφοροι τύποι συσχετίσεων μεταξύ των κλάσεων, όπως παρουσιάζονται παρακάτω:

· **Association**

Δείχνει τη συσχέτιση μιας κλάσης με μία άλλη. Η συσχέτιση αυτή μπορεί να είναι μιας κατεύθυνσης (δηλαδή μόνο η μία κλάση «γνωρίζει» την σχέση της με την άλλη κλάση) ή δυο κατευθύνσεων (σχέση ανάμεσα και στις δύο κλάσεις). Στην περίπτωση μονής κατεύθυνσης η γραμμή που συνδέει τις κλάσεις έχει ένα βέλος, που υποδεικνύει την κλάση που δεν γνωρίζει την σχέση.

Ένα παράδειγμα μιας μονής σχέσης μεταξύ δύο κλάσεων είναι η σχέση μεταξύ των κλάσεων Administrator και Παίγνιο. Η σχέση μεταξύ των δύο αυτών κλάσεων είναι ότι η κλάση Administrator διαχειρίζεται την κλάση Παίγνιο (όχι όμως και το αντίστροφο).

Τα χαρακτηριστικά που ορίζονται για κάθε association είναι:

- **Όνομα συσχέτισης:** χρησιμοποιείται μια φράση με ρήμα η οποία δηλώνει με ποιο τρόπο η κλάση σχετίζεται με μια άλλη. Για παράδειγμα ο Administrator διαχειρίζεται το Παίγνιο.
- **Πολλαπλότητα (Multiplicity) συσχέτισης:** Πολλαπλότητα είναι ο αριθμός των εμφανίσεων (instance) μιας κλάσης που συσχετίζεται με μια άλλη κλάση. Η πολλαπλότητα είναι μια ένδειξη για το πόσα αντικείμενα συμμετέχουν σε κάθε συσχέτιση μεταξύ των κλάσεων. Για παράδειγμα ένας Administrator μπορεί να διαχειριστεί από 1..* Παίγνια (ένα ή περισσότερα παίγνια).

Οι περιπτώσεις πολλαπλής συσχέτισης μεταξύ των κλάσεων δίνεται από τον παρακάτω πίνακα:

Πολλαπλότητα	**Επεξήγηση**
0..1	Καμία ή μία εμφάνιση
0..* ή *	Κανένα όριο στον αριθμό των εμφανίσεων
1	Ακριβώς μια εμφάνιση

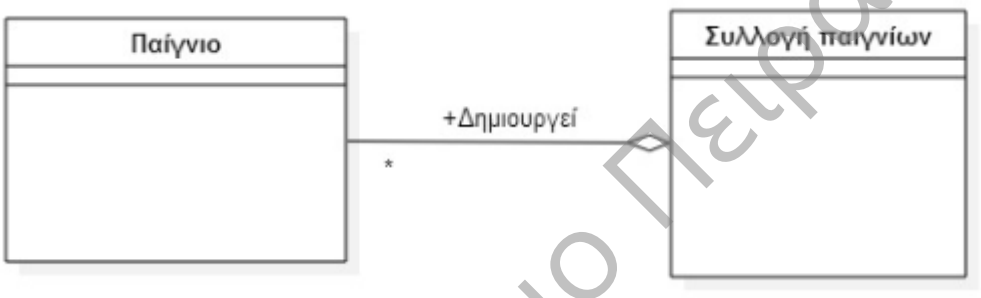
Πανεπιστήμιο Πειραιώς

1..*	Τουλάχιστον μία συσχέτιση

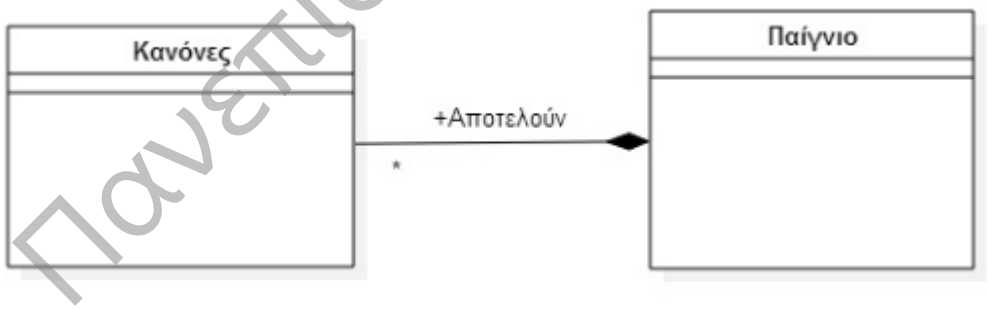
· Aggregation

Είναι μια ειδική μορφή συσχέτισης και δείχνει ότι οι κλάσεις σχετίζονται μεταξύ με τέτοιο τρόπο ώστε να ενώνονται μεταξύ τους για να δημιουργήσουν μια νέα κλάση (για παράδειγμα μια κλάση Παίγνιο σχετίζεται με την κλάση Συλλογή Παιγνίων με την έννοια ότι πολλά Παίγνια δημιουργούν μια Συλλογή Παιγνίων. Υπάρχουν 2 ειδών aggregations:

- **Basic Aggregation**: δείχνει ότι μια κλάση είναι υφιστάμενη μιας άλλης κλάσης αλλά η κλάση-μέρος μπορεί να υπάρξει και μόνη της (π.χ. η κλάση Παίγνιο μπορεί να υπάρξει και μόνη της και μπορεί να δημιουργήσει την κλάση Συλλογή Παιγνίων). Αναπαρίσταται με μια γραμμή με ένα άσπρο ρόμβο προς την πλευρά της κλάσης-σύνολο.



- **Composition Aggregation:** δείχνει την συσχέτιση μέρος-σύνολο αλλά σημαίνει ότι η κλάση-μέρος δεν μπορεί να υπάρξει μόνη της (π.χ. η κλάση Παίγνιο αποτελείται από πολλές κλάσεις Κανόνες, αλλά η κλάση Κανόνας από μόνη της δεν μπορεί να υπάρχει). Αναπαρίσταται με μια γραμμή με ένα μαύρο ρόμβο προς την πλευρά της κλάσης σύνολο.

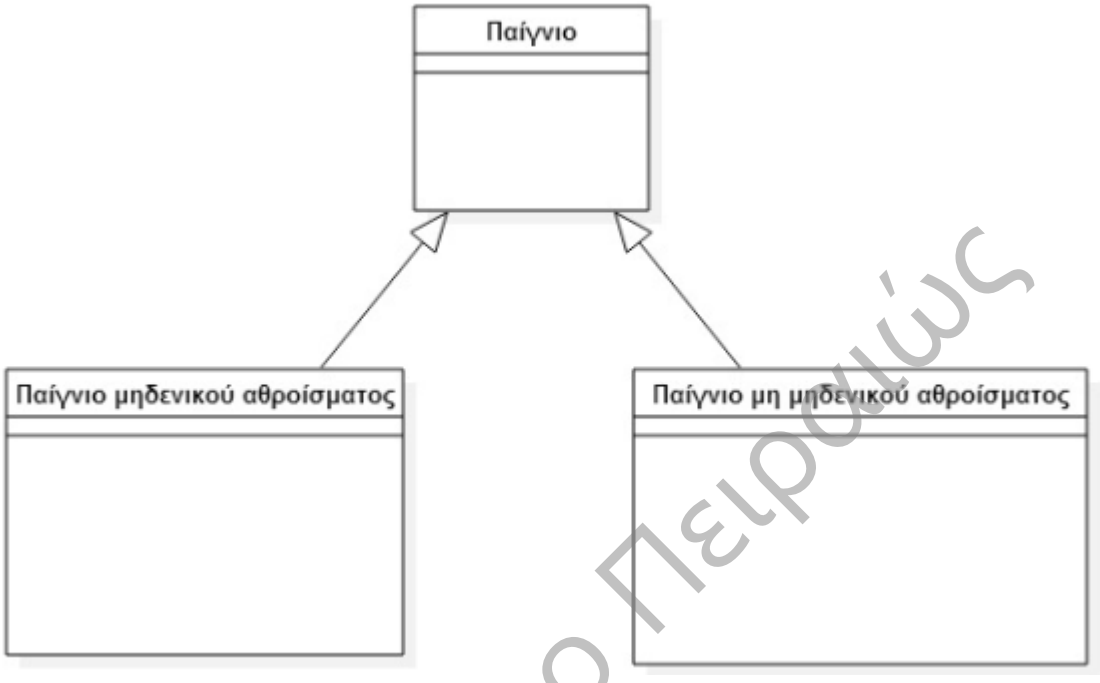


Generalization

Είναι ο τρόπος με τον οποίο αναπαριστούμε την κληρονομικότητα (inheritance) μεταξύ των κλάσεων (συσχέτιση πατέρα-παιδιού μεταξύ των κλάσεων). Στην συσχέτιση αυτή ο πατέρας κληρονομεί τα χαρακτηριστικά και τις μεθόδους του στα παιδιά.

Η αναπαράσταση της κληρονομικότητας γίνεται με μια γραμμή με ένα κλειστό βέλος, που δείχνει από την κλάση-παιδί προς την κλάση-πατέρας. Για παράδειγμα έχουμε την κλάση Παίγνιο που κληρονομείται από τις κλάσεις Παίγνιο μηδενικού αθροίσματος και Παίγνιο μη μηδενικού αθροίσματος.

Πανεπιστήμιο Πειραιώς



Χρήση των Class διαγραμμάτων

Τα Class διαγράμματα είναι η ραχοκοκαλιά της ανάλυσης του συστήματος και χρησιμοποιούνται για να παραστήσουν την στατική δομή της εφαρμογής. Παρουσιάζουν τα αντικείμενα, από τα οποία αποτελείται η εφαρμογή καθώς και τις συσχετίσεις μεταξύ αυτών. Μέσα από τα Class διαγράμματα οι προγραμματιστές της εφαρμογής θα μπορέσουν να δουν ποια αντικείμενα θα χρησιμοποιηθούν και έτσι θα αρχίσουν να δημιουργούν τις κλάσεις του συστήματος.

1.5.4 Sequence Διαγράμματα

Τα Sequence διαγράμματα χρησιμοποιούνται για να αναπαραστήσουν την αλληλεπίδραση των αντικειμένων ενός συστήματος στο πέρασμα του χρόνου για μια συγκεκριμένη δραστηριότητα του συστήματος. Συνήθως χρησιμοποιούμε Sequence διαγράμματα για να περιγράψουμε μια λειτουργία του συστήματος που έχει ήδη περιγραφεί με ένα Use Case διάγραμμα. Το Sequence διάγραμμα δείχνει τον τρόπο συμμετοχής των αντικειμένων στην περιγραφόμενη λειτουργία και την σειρά με την οποία ανταλλάσσονται τα μηνύματα που χρησιμοποιούνται στην επικοινωνία των αντικειμένων. Τα αντικείμενα που παρουσιάζονται σε ένα Sequence διάγραμμα συνήθως προέρχονται από την ανάλυση των Class διαγραμμάτων.

Με την χρήση των Sequence διαγραμμάτων βλέπουμε την διάρκεια ζωής των διαφόρων αντικειμένων σε μία συγκεκριμένη λειτουργία του συστήματος και την επικοινωνία μεταξύ τους.

Πανεπιστήμιο Πειραιώς

Με τον όρο διάρκεια ζωής εννοούμε την χρονική διάρκεια κατά την οποία τα αντικείμενα είναι ενεργά και ανταλλάζουν μηνύματα. Η έννοια του χρόνου αναπαρίσταται με την σειρά με την οποία ανταλλάσσονται τα μηνύματα, η οποία καθορίζεται διαβάζοντας το διάγραμμα από πάνω προς τα κάτω.

Ένα Sequence διάγραμμα έχει δύο διαστάσεις με την κάθετη διάσταση να αντιπροσωπεύει τον χρόνο ζωής του κάθε αντικειμένου και την οριζόντια διάσταση να περιέχει τα αντικείμενα που συμμετέχουν στην λειτουργία που περιγράφεται με το διάγραμμα.

Τα βασικά στοιχεία (elements) που χρησιμοποιούνται στα Sequence διαγράμματα είναι:

· **Objects (Αντικείμενα)**

Είναι τα αντικείμενα τα οποία συμμετέχουν στην λειτουργία που περιγράφεται στο διάγραμμα και που επικοινωνούν μεταξύ τους. Τα αντικείμενα αναπαρίστανται με ένα τετράγωνο κουτί στο άνω μέρος του διαγράμματος με το όνομα του αντικειμένου.

· **Objects Lifelines (Γραμμές Ζωής Αντικειμένων)**

Είναι κάθετες γραμμές, που αναπαριστούν τον χρόνο ζωής των αντικειμένων. Το διακεκομμένο κάθετο τμήμα της γραμμής δείχνει ότι το αντικείμενο είναι απενεργοποιημένο (αλλά όχι διαγραμμένο, συμμετέχει δηλαδή ακόμα στην λειτουργία απλώς δεν έχει διαγραφεί) και περιμένει να δεχθεί κάποιο μήνυμα προκειμένου να ενεργοποιηθεί.

Ένα κάθετο στενό ορθογώνιο δείχνει ότι το αντικείμενο είναι ενεργοποιημένο και έχει στείλει κάποιο μήνυμα σε κάποιο άλλο αντικείμενο. Όταν ένα αντικείμενο έχει καταστραφεί (διαγραφεί) η γραμμή αυτή σταματά και τοποθετείται ένα μεγάλο Χ πάνω στη γραμμή, που επισημαίνει ότι το αντικείμενο αυτό παύει να συμμετέχει στη λειτουργία που περιγράφεται (αντίστοιχα με την εντολή destroy που χρησιμοποιείται για να διαγραφεί ένα αντικείμενο).

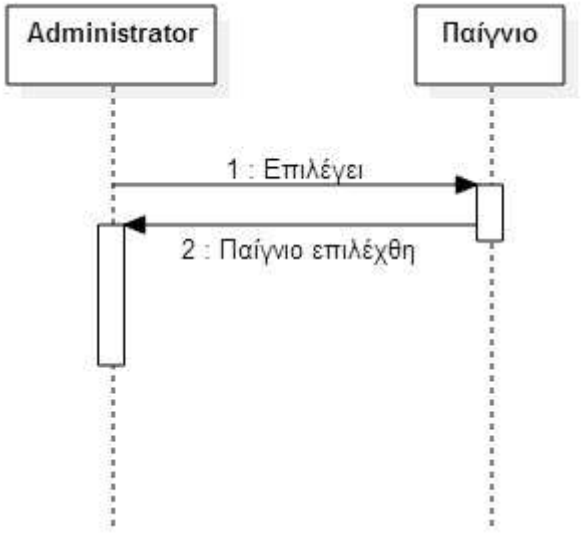
· **Messages (Μηνύματα)**

Είναι τα διάφορα μηνύματα που ανταλλάσσονται μεταξύ των διαφόρων αντικειμένων του διαγράμματος. Ένα μήνυμα αναπαρίσταται με ένα βέλος με κατεύθυνση προς το αντικείμενο παραλήπτη και είναι αυτά τα οποία προκαλούν την ενεργοποίηση κάποιας λειτουργίας στο αντικείμενο στο οποίο απευθύνονται. Τα μηνύματα ουσιαστικά αναπαριστούν τις συναρτήσεις που αποστέλλονται ανάμεσα στα αντικείμενα της εφαρμογής. Στα μηνύματα ορίζεται το όνομα, τα ορίσματα (arguments) που τυχόν έχουν καθώς και τον τύπο (return type).

Τα μηνύματα σε ένα Sequence διάγραμμα είναι δυο ειδών:

- **Synchronous:** μηνύματα που αποστέλλονται σε ένα αντικείμενο και απαιτούν απάντηση (response) ορθής παράδοσης. Η απάντηση στο μήνυμα αναπαρίσταται στο διάγραμμα με μια διακεκομμένη γραμμή από τον παραλήπτη προς τον αποστολέα.

Πανεπιστήμιο Πειραιώς



- **Asynchronous**: μηνύματα που δεν απαιτούν απάντηση, απλώς αποτελούν ένα απλό σήμα (signal) από ένα αντικείμενο που ενεργοποιεί ένα άλλο αντικείμενο.

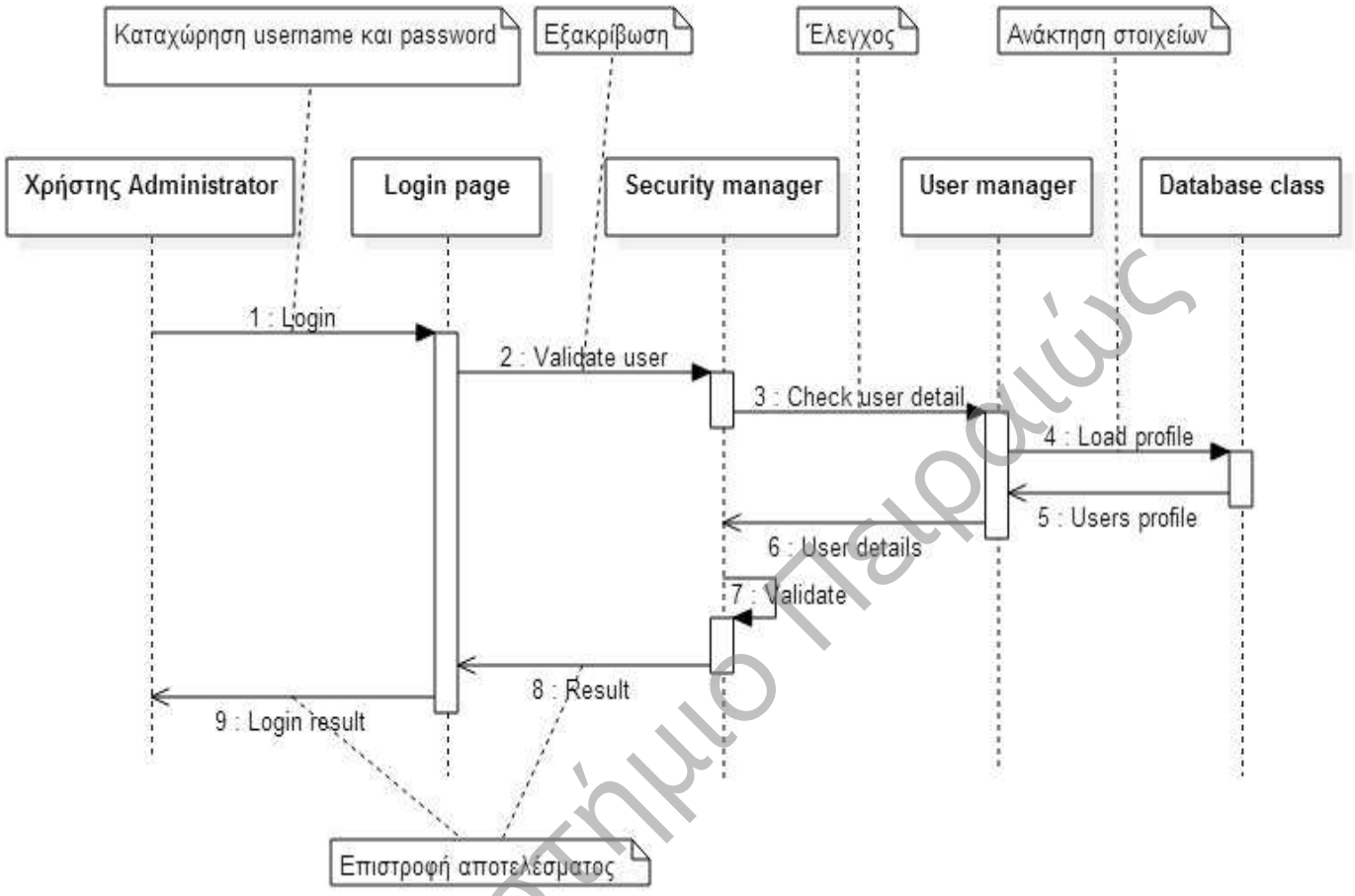
Χρήση των Sequence διαγραμμάτων

Τα Sequence διαγράμματα χρησιμοποιούνται για να περιγράψουν μια λειτουργία του συστήματος στο πέρασμα του χρόνου. Η περιγραφή αυτή γίνεται μέσα από την παρουσίαση των αντικειμένων που χρησιμοποιούνται στην λειτουργία αυτή, την παρουσίαση των μηνυμάτων που ανταλλάσσονται μεταξύ των αντικειμένων καθώς και την παρουσίαση της διάρκειας ζωής του κάθε αντικείμενου, που αναπαριστάται από την σειρά των μηνυμάτων από πάνω προς τα κάτω.

Παράδειγμα Sequence διαγράμματος

Στη συνέχεια παρουσιάζουμε ένα παράδειγμα Sequence διαγράμματος που χρησιμοποιείται για τον έλεγχο των στοιχείων των χρηστών, όταν αυτοί συνδέονται σε μια login οθόνη για να εισέλθουν σε μια εφαρμογή.

Πανεπιστήμιο Πειραιώς



1.5.5 Collaboration Διαγράμματα

Τα Collaboration διαγράμματα είναι μια παραλλαγή των Sequence διαγραμμάτων στα οποία παρουσιάζονται για μια συγκεκριμένη λειτουργία του συστήματος τα αντικείμενα και ο τρόπος επικοινωνίας μεταξύ τους. Η διαφορά είναι ότι εδώ δεν εμφανίζονται τα μηνύματα που ανταλλάσσονται μεταξύ των αντικειμένων στην διάρκεια του χρόνου (από πάνω προς τα κάτω) αλλά εμφανίζονται με αύξουσα αρίθμηση ανάλογα με τον χρόνο της αποστολής τους.

Τα αντικείμενα που χρησιμοποιούνται στα διαγράμματα αυτά είναι τα ίδια με τα αντικείμενα που χρησιμοποιήθηκαν στα Sequence διαγράμματα και αναπαρίστανται με ένα ορθογώνιο, που περιέχει το όνομα του αντικειμένου.

Τα μηνύματα αναπαρίστανται ως εξής:

 # αριθμός μηνύματος : [συνθήκη] όνομα_μηνύματος (ορίσματα)

όπου ο αύξων αριθμός του μηνύματος δείχνει την σειρά αποστολής του.

Όπως στα Sequence διαγράμματα η σειρά αποστολής των μηνυμάτων φαινόταν από τη σειρά εμφάνισής τους από πάνω προς τα κάτω, στα Collaboration διαγράμματα η σειρά αποστολής των μηνυμάτων φαίνεται από τον αριθμό που υπάρχει στον τίτλο του κάθε μηνύματος.

Πανεπιστήμιο Πειραιώς

Η μετάδοση των μηνυμάτων αναπαρίσταται με βέλη ακριβώς όπως στην περίπτωση των Sequence διαγραμμάτων.

Χρήστη των Collaboration διαγραμμάτων

Τα Collaboration διαγράμματα έχουν την ίδια ακριβώς χρησιμότητα με τα Sequence διαγράμματα δηλαδή δείχνουν την λειτουργία ενός τμήματος του συστήματος παριστάνοντας τα διάφορα αντικείμενα και τα μηνύματα που ανταλλάσσονται μεταξύ τους στη διάρκεια του χρόνου.

Η κύρια διαφορά με τα Sequence διαγράμματα είναι ότι η παρουσίαση του χρόνου δεν γίνεται παρουσιάζοντας τα μηνύματα από πάνω προς τα κάτω αλλά η παρουσίαση της χρονικής διάρκειας παρουσιάζεται με ένα αύξοντα αριθμό που χαρακτηρίζει κάθε μήνυμα.

1.5.6 Statechart Διαγράμματα

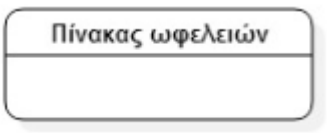
Τα Statechart διαγράμματα χρησιμοποιούνται για να αναπαραστήσουν τον κύκλο ζωής των αντικειμένων (objects) του συστήματος όπως αυτά έχουν προκύψει από τα Use Case, τα Sequence και τα Class διαγράμματα. Ο κύκλος ζωής των αντικειμένων αφορά τις διαφορετικές καταστάσεις (states) στις οποίες εμφανίζονται αυτά τα αντικείμενα καθώς και στα γεγονότα (events) τα οποία ενεργοποιούν αυτές τις καταστάσεις. Περιγράφουν όλες τις πιθανές καταστάσεις στις οποίες μπορεί να βρεθεί ένα αντικείμενο όταν συμβούν αντίστοιχα γεγονότα, που τα ενεργοποιούν.

Μέσα από το διάγραμμα φαίνονται οι διάφορες λειτουργίες (activities) τις οποίες εκτελεί ένα αντικείμενο όταν βρίσκεται σε μια συγκεκριμένη κατάσταση. Τα Statechart διαγράμματα δείχνουν την συμπεριφορά κάποιων αντικειμένων, που χρησιμοποιούνται σε διάφορα use cases που έχουν δημιουργηθεί. Χρησιμοποιούνται κυρίως για αντικείμενα των οποίων την συμπεριφορά είναι απαραίτητο να κατανοήσουμε πλήρως μέσα σε ολόκληρο το σύστημα. Τα Statechart διαγράμματα δεν χρησιμοποιούνται για την περιγραφή κάθε αντικειμένου του συστήματος, αλλά μόνο εκείνα τα οποία θεωρούνται σημαντικά και πρέπει να παρατηρείται η συμπεριφορά τους και συνδυάζονται με άλλα διαγράμματα όπως για παράδειγμα τα Activity διαγράμματα προκειμένου να δείξουν την ροή της εργασία σε κάποιο συγκεκριμένο σημείο της εφαρμογής καθώς και την συμπεριφορά των διαφόρων αντικειμένων που χρησιμοποιούνται.

Τα βασικά στοιχεία των Statechart διαγραμμάτων παρουσιάζονται στη συνέχεια:

· **State (Κατάσταση)**

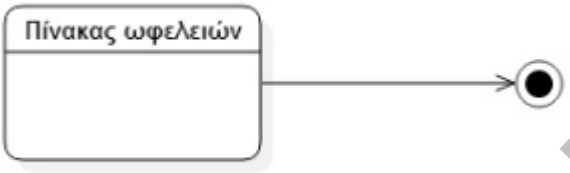
Η κατάσταση (state) στην οποία μπαίνει κάποιο αντικείμενο απεικονίζεται με ένα ορθογώνιο με στρογγυλές άκρες το οποίο περιέχει το όνομα της κατάστασης.



Πανεπιστήμιο Πειραιώς

Η αρχική κατάσταση του αντικειμένου απεικονίζεται με ένα μαύρο κύκλο.

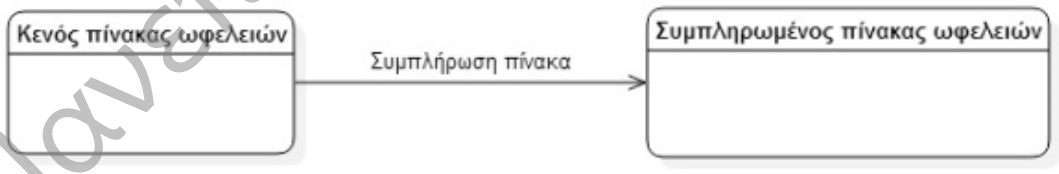
Η τελική κατάσταση, στην οποία καταλήγει το αντικείμενο απεικονίζεται με ένα κύκλο που περιέχει ένα μικρότερο μαύρο κύκλο. Οι τελικές καταστάσεις σε ένα Statechart διάγραμμα μπορούν να είναι περισσότερες από μια.



· **Events (Γεγονότα)**

Event ονομάζεται το γεγονός, το οποίο προκαλεί την μετάβαση από την μια κατάσταση σε μια άλλη. Η μετάβαση (transition) από την μια στην άλλη κατάσταση απεικονίζεται με βέλος το οποίο ονομάζεται από τον event που προκαλεί την κατάσταση. Επίσης στο transition παρουσιάζεται όταν είναι απαραίτητο μια συνθήκη ελέγχου (guard condition) η οποία πρέπει να είναι αληθής προκειμένου να πραγματοποιηθεί η μετάβαση.

Για παράδειγμα, το αντικείμενο Πίνακας ωφελειών αρχικά είναι κενό. Όταν συμπληρωθεί από τον παίκτη ή τον Administrator μετατρέπεται σε Συμπληρωμένο πίνακα ωφελειών.



Υπάρχουν 3 τύποι events:

- **Call event:** κλήση μιας συγκεκριμένης λειτουργίας από ένα αντικείμενο
- **Change event:** αλλαγή κατάστασης κάτω από μια συγκεκριμένη συνθήκη
- **Time Event:** εκτέλεση λειτουργίας μετά την έλευση συγκεκριμένου χρόνου

· **Actions (Λειτουργίες)**

Action είναι η λειτουργία η οποία προκαλεί ένα γεγονός (events) και είναι στην πραγματικότητα η αιτία που προκαλεί την μετάβαση στην επόμενη κατάσταση. Η αναπαράσταση της λειτουργίας (action) απεικονίζεται με το όνομα του γεγονότος (event), που προκαλεί αυτή την μετάβαση, το

Πανεπιστήμιο Πειραιώς

σύμβολο «/» και το όνομα του γεγονότος (action), που εκτελείται για να πραγματοποιηθεί η μετάβαση.

Στο προηγούμενο παράδειγμα, είχαμε ένα call event, την κλήση δηλαδή μιας λειτουργίας (action) και συγκεκριμένα της λειτουργίας Συμπλήρωση πίνακα.

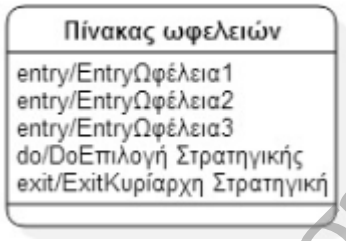
· Internal Actions (Εσωτερικές Λειτουργίες)

Internal Actions είναι οι εσωτερικές λειτουργίες οι οποίες εκτελούνται από ένα αντικείμενο όταν βρίσκονται σε κάποια κατάσταση. Οι εσωτερικές λειτουργίες αναπαρίστανται εντός του ορθογωνίου της κατάστασης. Οι τύποι των εσωτερικών λειτουργιών ενός αντικειμένου είναι οι παρακάτω:

Entry: εσωτερικές λειτουργίες που εκτελούνται όταν το αντικείμενο έρθει στην συγκεκριμένη κατάσταση. Αναπαρίσταται με τη λέξη Entry, το «/» και το όνομα της εσωτερικής λειτουργίας.

Do: εσωτερικές λειτουργίες που εκτελούνται μέσα στην ίδια κατάσταση. Αναπαρίστανται με τη λέξη Do, το «/» και το όνομα της εσωτερικής λειτουργίας που θα εκτελεστεί μέσα στο αντικείμενο.

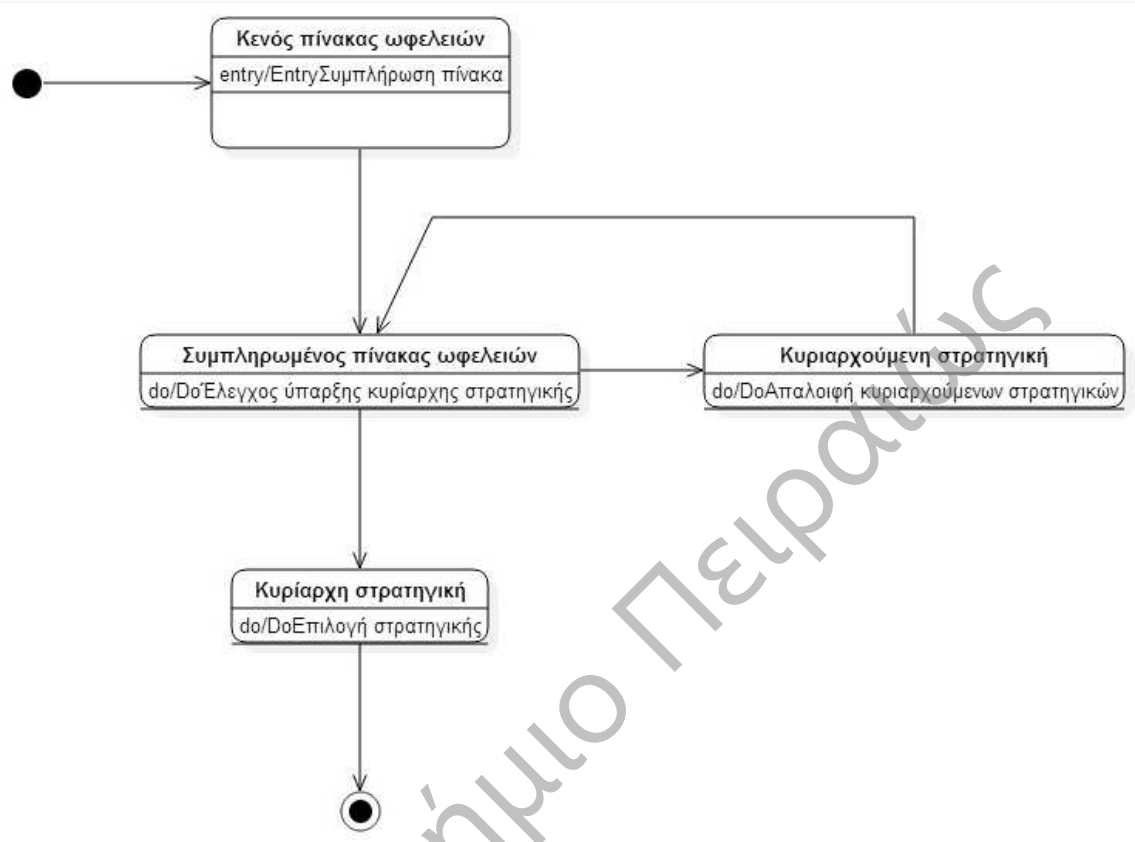
Exit: εσωτερικές λειτουργίες που θα εκτελεστούν όταν το αντικείμενο σταματήσει να έχει την συγκεκριμένη κατάσταση. Αναπαρίστανται με τη λέξη Exit, το «/» και το όνομα της εσωτερικής λειτουργίας.



Παράδειγμα Statechart διαγράμματος

Στη συνέχεια παρουσιάζεται ένα παράδειγμα Statechart διαγράμματος για την περίπτωση της εφαρμογής συμπλήρωσης πίνακα ωφελειών. Στο παράδειγμα παρουσιάζονται οι καταστάσεις του αντικειμένου Πίνακας ωφελειών και τα διάφορα γεγονότα που αλλάζουν τις καταστάσεις του.

Πανεπιστήμιο Πειραιώς



1.5.7 Component Diagrams

Το Component διάγραμμα χρησιμοποιείται για την περιγραφή της φυσικής υλοποίησης των τμημάτων λογισμικού του συστήματος. Όπως και τα Class διαγράμματα χρησιμοποιούνται για να περιγράψουν τον λογικό σχεδιασμό της εφαρμογής (logical design) με την παρουσίαση των κλάσεων που αποτελούν την εφαρμογή έτσι και τα Component διαγράμματα χρησιμοποιούνται για να περιγράψουν την φυσική υλοποίηση (physical implementation) της εφαρμογής παρουσιάζοντας όλα τα τμήματα του πηγαίου κώδικα (source code) καθώς και τα εκτελέσιμα αρχεία (executable files) που χρησιμοποιούνται από την εφαρμογή.

Σκοπός του Component διαγράμματος είναι η παρουσίαση των τμημάτων του κώδικα της εφαρμογής και των συσχετίσεων μεταξύ τους. Τα components (συστατικά μέρη) που μπορεί να υπάρξουν σε ένα σύστημα του ομαδοποιούνται ως εξής:

Deployment components: συστατικά μέρη τα οποία χρησιμεύουν για να τρέξει η εφαρμογή.

Work product components: στοιχεία που περιλαμβάνουν μοντέλα, πηγαίο κώδικα και αρχεία δεδομένων.

Execution components: στοιχεία που δημιουργούνται κατά την διάρκεια εκτέλεσης της εφαρμογής.

Για μια επιχείρηση τα κομμάτια του λογισμικού μπορεί επίσης να αντιπροσωπεύουν επιχειρηματικές διαδικασίες και έγγραφα. Τα συστατικά μέρη (components) της εφαρμογής μπορεί να δημιουργηθούν βάσει των κλάσεων του συστήματος, που έχουν ήδη οριστεί στα

Πανεπιστήμιο Πειραιώς

αντίστοιχα Class διαγράμματα, ή και να σχεδιαστούν από την αρχή από τους προγραμματιστές της εφαρμογής.

Τα βασικά στοιχεία (elements) των Component διαγραμμάτων είναι τα παρακάτω:

· **Component**

Το component είναι ένα δομικό στοιχείο της εφαρμογής μας. Γραφικά αναπαρίσταται με ένα ορθογώνιο με δυο μικρά ορθογώνια στην αριστερή πλευρά του και μπορεί να είναι τμήμα κώδικα, κάποιο εκτελέσιμο αρχείο, ένα αρχείο βιβλιοθήκης εφαρμογών του οργανισμού (π.χ. ένα αρχείο .dll) ή οτιδήποτε άλλο ανήκει και αποτελεί τμήμα της εφαρμογής .

· **Component Stereotypes**

Stereotypes είναι δηλώσεις που δείχνουν τον τύπο του component και δηλώνονται πάνω από το όνομα του. Τα διαθέσιμα stereotypes που μπορούν να χρησιμοποιηθούν σε ένα component είναι τα εξής:

<<executable>>: εκτελέσιμο αρχείο.

<<library>>: βιβλιοθήκη που χρησιμοποιείται από ένα εκτελέσιμο αρχείο κατά την εκτέλεση της εφαρμογής.

<<table>>: πίνακας ή άλλο στοιχείο Βάσης Δεδομένων.

<<file>>: δήλωση αρχείου δεδομένων ή αρχείου με κώδικα.

<<document>>: δήλωση κάποιου αρχείου (π.χ. έγγραφο που χρησιμοποιείται στην εφαρμογή).

· **Component Dependencies**

Dependencies είναι οι συσχετίσεις που υπάρχουν μεταξύ των components. Οι εξαρτήσεις μεταξύ των components δείχνουν τον τρόπο που αυτά σχετίζονται μεταξύ τους και πως οι αλλαγές σε ένα component μπορεί να επηρεάσουν άλλα components του συστήματος. Οι σχέσεις αυτές μεταξύ των components αναπαρίστανται με μια διακεκομμένη γραμμή ανοιχτού βέλους όπως φαίνεται παρακάτω.

Πανεπιστήμιο Πειραιώς

1.5.8 Deployment Diagrams

Τα Deployment διαγράμματα χρησιμοποιούνται για να αναπαραστήσουν την αρχιτεκτονική του hardware στο οποίο θα εγκατασταθεί το σύστημα καθώς επίσης και τον προσδιορισμό των τμημάτων του συστήματος όπου θα γίνονται οι διάφορες εργασίες, όπως για παράδειγμα υπολογιστές, servers, εκτυπωτές, αποθηκευτικοί χώροι κλπ. Σκοπός των διαγραμμάτων αυτών είναι η παρουσίαση της φυσικής υλοποίησης του συστήματος και συνήθως χρησιμοποιούνται σε συνεργασία με τα Component διαγράμματα για να δώσουν μια ολοκληρωμένη εικόνα του συστήματος. Παρουσιάζεται ο φυσικός σχεδιασμός της εφαρμογής και τα μηχανήματα στα οποία τρέχουν τα στοιχεία αυτά.

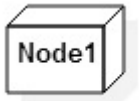
Παρακάτω παρουσιάζονται τα βασικά στοιχεία των deployment diagrams:

· **Nodes (Κόμβοι)**

Κάθε κόμβος (node) είναι μια ανεξάρτητη μονάδα του συστήματος στο οποίο γίνονται οι διάφορες εργασίες κατά τη διάρκεια της εκτέλεσης της εφαρμογής. Οι κόμβοι περιλαμβάνουν συσκευές επεξεργασίας (υπολογιστές, εκτυπωτές και άλλες μηχανές επεξεργασίας) αλλά επίσης και ανθρώπινους (human resources) ή μηχανικούς πόρους επεξεργασίας (mechanical process resources).

Μέσα σε έναν κόμβο μπορούν να οριστούν επιπλέον και components (κώδικας) κάποιου συγκεκριμένου υπολογιστή που χρησιμοποιούνται για την εκτέλεση του προγράμματος προκειμένου να υπάρχει καλύτερη ομαδοποίηση τόσο του λογισμικού όσο και του υλικού της εφαρμογής.

Το Deployment διάγραμμα υποδεικνύει τον τρόπο με τον οποίο τα components από τα οποία αποτελείται το σύστημα κατανέμονται στις φυσικές μονάδες του συστήματος. Τα Deployment διαγράμματα χρησιμοποιούνται παράλληλα με τα Component διαγράμματα για να αναπαραστήσουν τα δομικά στοιχεία της εφαρμογής μαζί με τα απαραίτητα μηχανήματα στα οποία βρίσκονται τα κομμάτια του λογισμικού. Οι κόμβοι αναπαρίστανται με ένα τρισδιάστατο ορθογώνιο, το οποίο μπορεί να αποτελεί οποιοδήποτε hardware (πχ υπολογιστής)



· **Association**

Η επικοινωνία (association) μεταξύ των κόμβων αναπαρίσταται με μια γραμμή που ενώνει τους δύο κόμβους. Στην γραμμή επικοινωνίας μπορεί να χρησιμοποιηθεί και μια περιγραφή που να αναπαριστά τον τρόπο επικοινωνίας μεταξύ των υλικών του συστήματος. Για παράδειγμα με τη η λέξη <<TCP/IP>> υποδηλώνεται ότι η επικοινωνία γίνεται μέσω ενός τοπικού δικτύου (intranet) ή η λέξη <<parallel>> υποδηλώνει τον τρόπο επικοινωνίας ενός Η/Υ με έναν εκτυπωτή.

Πανεπιστήμιο Πειραιώς

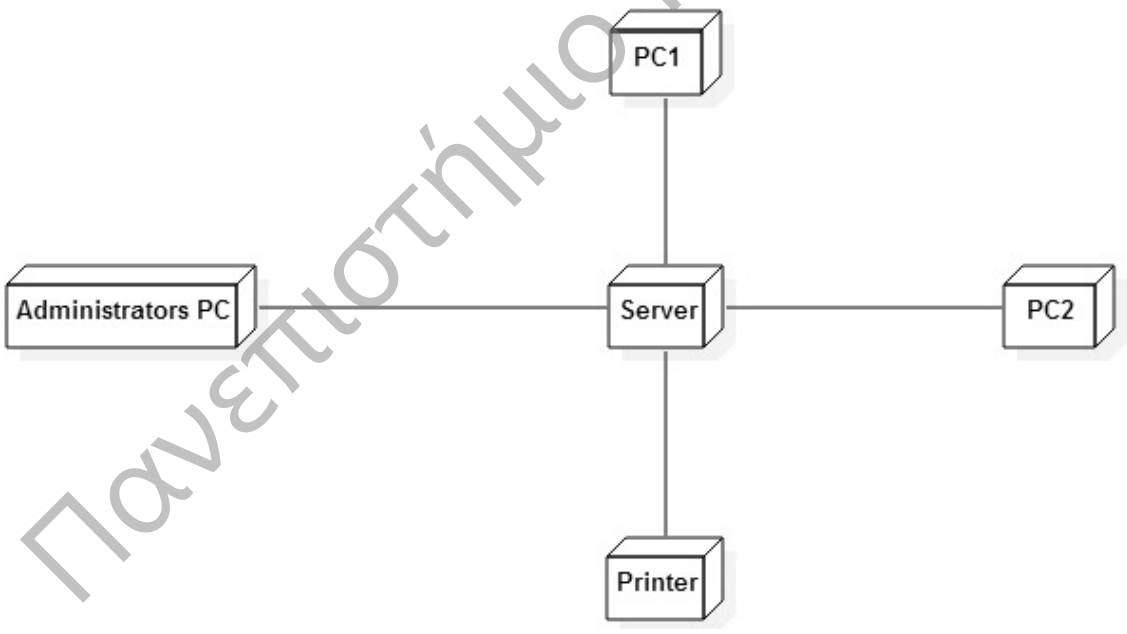
Χρήση των Deployment και των Component διαγραμμάτων

Ο συνδυασμός των Deployment και των Component διαγραμμάτων, παρέχει μια υψηλού επιπέδου φυσική περιγραφή του ολοκληρωμένου συστήματος παρουσιάζοντας τον τρόπο με τον οποίο τα δομικά στοιχεία του συστήματος (components, nodes) κατανέμονται, επικοινωνούν και συνδυάζονται.

Μέσα από τα Component διαγράμματα μπορούμε να δούμε την φυσική περιγραφή του λογισμικού του συστήματος, ενώ μέσα από τα Deployment διαγράμματα μπορούμε να δούμε την διάταξη του hardware που θα λειτουργήσει το σύστημα ενώ πολλές φορές είναι πολύ χρήσιμος ο συνδυασμός και των δυο διαγραμμάτων προκειμένου να έχουμε μια ολοκληρωμένη εικόνα του συστήματος.

Παράδειγμα Deployment διαγράμματος

Στη συνέχεια παρουσιάζεται ένα παράδειγμα χρήσης ενός Deployment διαγράμματος, όπου παρουσιάζεται η αρχιτεκτονική μιας τυπικής εφαρμογής επίλυσης προβλημάτων θεωρίας παιγνίων. Συγκεκριμένα φαίνονται οι υπολογιστές, που περιέχουν την εφαρμογή, τα άλλα μηχανήματα που χρησιμοποιούνται καθώς και οι συσχετίσεις μεταξύ του hardware.



Πανεπιστήμιο Πειραιώς

Στον παρακάτω πίνακα, παρουσιάζονται συνοπτικά τα διαγράμματα της UML, που παρουσιάσαμε καθώς και οι χρήσεις τους κατά την διαδικασία της ανάπτυξης της εφαρμογής

Διάγραμμα	**Χρήση**
Use Case	✓ Περιγραφή βασικών λειτουργιών ✓ Απαιτήσεις χρηστών ✓ Τεκμηρίωση συστήματος ✓ Χρήση σεναρίων για έλεγχο λειτουργιών
Activity	✓ Αναπαράσταση εκτέλεσης λειτουργιών ✓ Υλοποίηση use case σεναρίων ✓ Προβολή workflow σε διάφορα σημεία της εφαρμογής
Class	✓ Δημιουργία μετά την ανάλυση των use case (εύρεση αντικειμένων) και των activity (εύρεση συσχετίσεων) διαγραμμάτων ✓ Παρουσίαση στατικής δομής εφαρμογής ✓ Παρουσίαση αντικειμένων της εφαρμογής και συσχετίσεων μεταξύ τους ✓ Χρησιμοποιείται από τους προγραμματιστές της εφαρμογής για την υλοποίηση των κλάσεων της εφαρμογής
Sequence	✓ Προβολή αλληλεπιδράσεων μεταξύ των αντικειμένων στη διάρκεια του χρόνου ✓ Προβολή ανταλλαγής μηνυμάτων μεταξύ των αντικειμένων ✓ Περιγραφή συγκεκριμένων λειτουργιών του συστήματος ✓ Τα αντικείμενα εντοπίζονται από τα class διαγράμματα
Collaboration	✓ Παρόμοια χρήση με τα Sequence διαγράμματα ✓ Η σειρά ανταλλαγής μηνυμάτων αναπαρίσταται με αύξουσα αρίθμηση
Statechart	✓ Αναπαριστούν την διάρκεια ζωής των αντικειμένων ✓ Παρουσιάζονται οι καταστάσεις των βασικών αντικειμένων της εφαρμογής ✓ Παρουσιάζονται τα γεγονότα που αλλάζουν μια κατάσταση ✓ Παρουσιάζουν τις εσωτερικές λειτουργίες ενός αντικειμένου σε μία κατάσταση ✓ Τα αντικείμενα προκύπτουν από τα use case, sequence και class διαγράμματα
Component	✓ Αναπαριστούν την φυσική δομή των τμημάτων λογισμικού της εφαρμογής ✓ Παρουσιάζονται οι συσχετίσεις μεταξύ τους
Deployment	✓ Αναπαριστούν την φυσική μορφή του hardware που θα χρησιμοποιείται από την εφαρμογή ✓ Σε συνδυασμό με τα component διαγράμματα δημιουργούν μια ολοκληρωμένη εικόνα της εφαρμογής

Πανεπιστήμιο Πειραιώς

Παρακάτω παρουσιάζεται η σειρά με την οποία πρέπει να χρησιμοποιούνται τα διαγράμματα της UML.

Το πρώτο βήμα αφορά στη δημιουργία των Use Case διαγραμμάτων. Στη συνέχεια δημιουργούνται τα Activity διαγράμματα, τα οποία μαζί με τα Use Case διαγράμματα βοηθούν στην δημιουργία των Class διαγραμμάτων. Στη συνέχεια τα Use Case διαγράμματα μαζί με τα Class διαγράμματα χρησιμοποιούνται στην δημιουργία των Sequence διαγραμμάτων, ενώ τα Class διαγράμματα μαζί με τα Use Case και τα Sequence διαγράμματα αποτελούν τη βάση για την δημιουργία των Statechart διαγραμμάτων. Επαναλαμβάνουμε ότι τα Collaboration διαγράμματα είναι παρόμοια με τα Sequence διαγράμματα, επομένως μπορούμε να πούμε ότι δημιουργούνται με τον ίδιο τρόπο.

Τέλος, τα Deployment και τα Component διαγράμματα είναι κατά κάποιον τρόπο αυτόνομα των υπολοίπων με την έννοια ότι δεν εξαρτώνται άμεσα από τα υπόλοιπα διαγράμματα αλλά συνδυάζονται μεταξύ τους.

Στόχοι και πλεονεκτήματα της UML

Η UML έχει σχεδιαστεί με σκοπό να εκπληρώσει συγκεκριμένους στόχους προκειμένου να μπορέσει να γίνει ένα πρότυπο εξυπηρέτησης των πρακτικών αναγκών της διαδικασίας ανάπτυξης εφαρμογών. Οι βασικοί στόχοι που προσπαθεί να εξυπηρετήσει η UML, όπως τους έχουν θέσει οι δημιουργοί της, είναι οι εξής:

- Παροχή στους σχεδιαστές μίας έτοιμης-προς-χρήση, εκφραστικής και οπτικής γλώσσας μοντελοποίησης για την παραγωγή και ανταλλαγή μοντέλων εφαρμογών.
- Υποστήριξη των ανεξάρτητων προδιαγραφών που ορίζουν συγκεκριμένες γλώσσες προγραμματισμού και διαδικασίες ανάπτυξης εφαρμογών.
- Παροχή μιας τυποποιημένης καλά ορισμένης βάσης για την κατανόηση της γλώσσας μοντελοποίησης.
- Ενθάρρυνση της αγοράς εργαλείων μοντελοποίησης. Η UML αποτελεί ένα καθιερωμένο πλέον πρότυπο και ενθαρρύνεται η ανάπτυξη εργαλείων μοντελοποίησης που υποστηρίζουν την UML ενώ παράλληλα παρέχουν επιπλέον δυνατότητες, όπως για παράδειγμα ολοκλήρωση με κώδικα, έλεγχο σύνταξης, διαχείριση βάσεων δεδομένων κλπ.
- Υποστήριξη εννοιών υψηλότερου επιπέδου όπως τα συστατικά στοιχεία των εφαρμογών, συνεργασίες μεταξύ συστατικών στοιχείων κλπ.

Η UML προσφέρει πολλά πλεονεκτήματα κατά τη διάρκεια ανάπτυξης εφαρμογών:

- Παρέχει μια εύκολη στην κατανόηση γλώσσα για οπτική μοντελοποίηση και ανάπτυξη του συστήματος.

Πανεπιστήμιο Πειραιώς

- Παρέχει την δυνατότητα ενοποίησης των ήδη υπαρχόντων μεθοδολογιών ανάπτυξης αντικειμενοστραφών εφαρμογών.
- Ενσωματώνει τους κανόνες, τις στρατηγικές και τις πρακτικές της επιχείρησης στο σύστημα.
- Επικεντρώνεται σε σύγχρονα θέματα ανάπτυξης εφαρμογών.
- Υποστηρίζει προδιαγραφές που είναι ανεξάρτητες από συγκεκριμένες γλώσσες προγραμματισμού ή διαδικασίες ανάπτυξης.

Πανεπιστήμιο Πειραιώς

2 Θεωρία Παιγνίων

Η θεωρία παιγνίων είναι μια μεθοδολογία ανάλυσης καταστάσεων μεταξύ μιας ομάδας λογικών (παραδοχή) ατόμων που λειτουργούν ως ανταγωνιστές και έχουν σκοπό τη μεγαλύτερη δυνατή απόκτηση οφέλους. Σκοπός της είναι να μας βοηθήσει να αναλύσουμε διάφορες καταστάσεις στις οποίες αλληλεπιδρούν δύο ή περισσότεροι παίκτες, κάθε ένας από τους οποίους συμπεριφέρεται με στρατηγικό τρόπο και προσπαθεί να πάρει την καλύτερη δυνατή απόφαση σε σχέση πάντα με την απόφαση του αντιπάλου. Σκοπός του κάθε παίκτη είναι να μεγιστοποιήσει το κέρδος του, σύμφωνα με την διαθέσιμη κλίμακα ωφέλειας του.

Σύμφωνα με το σχετικό άρθρο της ηλεκτρονικής εγκυκλοπαίδειας Wikipedia:

> *"Τα παίγνια είναι μία μέθοδος ανάλυσης προβλημάτων που έχουν σχέση με τον τρόπο λήψης αποφάσεων σε καταστάσεις σύγκρουσης και συνεργασίας. Παίκτης μπορεί να είναι ένα πρόσωπο, μία οργάνωση, ένα κράτος ή ένας συνασπισμός. Ως αντικείμενο έρευνας μπορούν να θεωρηθούν διάφορα προβλήματα πολιτικής, ψυχολογικής, κοινωνικής, οικονομικής μορφής."*

Απαραιτήτως για τη λύση των προβλημάτων αυτών πρέπει να προηγηθεί η ανάλυση καταστάσεων, όπου δύο ή περισσότεροι δρώντες (παίκτες) βρίσκονται αντιμέτωποι και ακολουθούν συνεργατικές στρατηγικές. Κάθε παίκτης χρησιμοποιεί όλα τα διαθέσιμα μέσα, προκειμένου να εμποδίσει τον αντίπαλό του να αποκτήσει πλεονεκτήματα που θα περιορίσουν τα κέρδη του. Επομένως, οι ενέργειές του καθενός εξαρτώνται άμεσα από τις επιλογές (στρατηγικές) που έχει διαθέσιμες ο αντίπαλος.

Χαρακτηρισμός ενός παιγνίου

Στην αρχική μας υπόθεση υπάρχει μία κατάσταση, στην οποία ορισμένοι δρώντες (παίκτες) παίρνουν αποφάσεις, οι οποίες οδηγούν σε ορισμένα αποτελέσματα (consequence). Ο αριθμός των παικτών μπορεί να ποικίλει από δύο και πάνω. Στην πρώτη περίπτωση έχουμε τα "δύο προσώπων παίγνια" (two-person-games), και στη δεύτερη περίπτωση τα "παίγνια ν-προσώπων" (n-person-games). Οι συμμετέχοντες σε παίγνιο περισσότερων των δύο προσώπων μπορούν να σχηματίσουν κατά τη διάρκεια του παιγνίου "συμμαχίες" μικρής ή μεγάλης διαρκείας, οπότε το παίγνιο μετατρέπεται αυτομάτως σε "παίγνιο δύο προσώπων". Θα πρέπει να αναφερθεί ότι υπάρχει μεγάλη διαφορά ενός παιγνίου σε σχέση με μία πραγματική κατάσταση απλού ανταγωνισμού ή σύγκρουσης στο ότι οι συνθήκες είναι αυστηρά ορισμένες και σύμφωνα με ορισμένους κανόνες. Το χαρακτηριστικό του ανταγωνισμού μεταξύ των παικτών περιέχεται σε όλα τα παίγνια και το αποτελέσμά του οδηγεί σε "κέρδη" ή "απώλειες".

Πανεπιστήμιο Πειραιώς

2.1 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Η πρώτη αναφορά στη Θεωρία Παιγνίων όπως την γνωρίζουμε σήμερα έγινε τον 18ο αιώνα (1838) από τον Γάλλο οικονομολόγο Augustin Cournot ο οποίος επεδίωξε να αναλύσει καταστάσεις ολιγοπωλίων με τρόπο παρόμοιο με τις σύγχρονες μεθόδους της θεωρίας παιγνίων.

Η ουσιαστική ανάπτυξη της θεωρίας παιγνίων αποδίδεται στον Ούγγρο φυσικό και μαθηματικό, John von Neumann, ο οποίος απέδειξε ότι τα παιχνίδια μηδενικού αθροίσματος έχουν πάντα λύση και ότι η απώλεια ενός παίκτη είναι ίση με το κέρδος του δεύτερου.

Ένα βήμα παραπέρα έκανε ο Αμερικανός μαθηματικός και οικονομολόγος John Nash ο οποίος εισήγαγε την έννοια της ισορροπίας για παιχνίδια μη-μηδενικού αθροίσματος, γνωστή σαν ισορροπία Nash. Πρόκειται για μια κατάσταση, από την οποία κανέναν παίκτη δεν τον συμφέρει να απομακρυνθεί, δεδομένων των επιλογών των αντιπάλων του.

Από εκείνο το σημείο και μετά η θεωρία παιγνίων γνώρισε μεγάλη ανάπτυξη και αποδοχή και άρχισε να εφαρμόζεται σε όλους τους τομείς, ενώ πληθώρα ερευνητικών πειραμάτων ξεκίνησαν προσπαθώντας να βρουν λύση σε όλο και περισσότερα προβλήματα.

Ο Reinhard Selten μελέτησε τα δυναμικά παίγνια εισάγοντας την έννοια της ισορροπίας στα υποπαίγνια (subgame perfect equilibrium) και της ισορροπίας τρεμάμενου χεριού (trembling hand perfect equilibrium), ενώ ο John Harsanyi γενίκευσε τις ιδέες του John Nash και μελέτησε παίγνια ατελούς πληροφόρησης.

Αργότερα άρχισε να εφαρμόζεται και σε άλλους κλάδους όπως αυτός της βιολογίας, σαν αποτέλεσμα της εργασίας του John Maynard Smith σχετικά με την έννοια της “εξελικτικά σταθερής στρατηγικής” (evolutionary stable strategy). Στη συνέχεια η θεωρία παιγνίων εφαρμόστηκε ακόμα και στον σχεδιασμό δημοπρασιών. Πολλοί επιστήμονες απασχολήθηκαν στην κατανομή δικαιωμάτων χρήσης του ηλεκτρομαγνητικού φάσματος στη βιομηχανία των κινητών τηλεπικοινωνιών με βάση την θεωρία παιγνίων. Σήμερα η θεωρία παιγνίων χρησιμοποιείται κατά κόρον στις Οικονομικές επιστήμες κυρίως σε θέματα ανταγωνισμού και της συνεργασίας μέσω της παιγνιοθεωρητικής ανάλυσης

2.2 ΕΦΑΡΜΟΓΕΣ ΤΗΣ ΘΕΩΡΙΑΣ ΠΑΙΓΝΙΩΝ

Η θεωρία παιγνίων έχει μεγάλη γκάμα εφαρμογών. Θα λέγαμε πως η γκάμα των θεμάτων που καλύπτει η θεωρία παιγνίων καλύπτει πολλούς τομείς και έχει εφαρμογές στην οικονομία, στις επιχειρήσεις, στην πληροφορική, στις τηλεπικοινωνίες, στην πολιτική, στην κοινωνιολογία, στη βιολογία και φυσικά στην καθημερινότητα. Αποτελεί μια σύγχρονη μαθηματική θεωρία όπου κάθε είδος αναμέτρησης μπορεί να αναλυθεί, από την ντάμα και το σκάκι μέχρι τον τζόγο ή έναν πυρηνικό πόλεμο, και να προβλεφθεί ο νικητής.

Η θεωρία παιγνίων χρησιμοποιείται από τους οικονομολόγους κατά κόρον τα τελευταία χρόνια για να αναλύσει διάφορους κλάδους όπως για παράδειγμα τη βιομηχανική οργάνωση (industrial organization), το σχεδιασμό μηχανισμών (mechanism design), τις δημοπρασίες, τις συμφωνίες, τα ολιγοπώλια, τα μονοπώλια, (βλ το πρώτο μοντέλο δυοπωλίου που ανέπτυξε ο Γάλλος

Πανεπιστήμιο Πειραιώς

μαθηματικός Κουρνό το 1838) τα συστήματα για να μπορεί κάποιος να ψηφίσει και πολλά άλλα. Οι έρευνες αυτές εστιάζουν στην ισορροπία που μπορεί να επιτευχθεί στα παιχνίδια, την οποία θα σχολιάσουμε περεταίρω παρακάτω. Επιπρόσθετα μπορεί να χρησιμοποιηθεί στην παγκόσμια διπλωματία και στις πολεμικές στρατηγικές, επηρεάζοντας τη μοίρα των διακρατικών σχέσεων ακόμη και αν αυτό δεν είναι άμεσα ορατό.

Χρησιμοποιείται όμως και για την περιγραφή σχέσεων συνεργασίας κυρίως σε κλάδους όπως της Πολιτικής Οικονομίας και ειδικά στη θεωρία της συλλογικής δράσης (Collective action). Αυτό σχετίζεται άμεσα με τον ρόλο του κράτους και των θεσμών σε θέματα συνεργασίας. Ένα χαρακτηριστικό παράδειγμα είναι η προσπάθεια για δίκαιη παροχή δημόσιων αγαθών και η αποτελεσματική φορολογία.

Η θεωρία παιγνίων έχει χρησιμοποιηθεί ακόμα και στη βιολογία για να βοηθήσει στην κατανόηση διαφόρων φαινομένων. Η πρώτη φορά που χρησιμοποιήθηκε για τέτοιο σκοπό ήταν για να εξηγήσει την εξέλιξη (και την σταθερότητα) της αναλογίας 1 προς 1 στα φύλα. Σύμφωνα με τον Ronald Fisher (1930) αυτή η αναλογία είναι αποτέλεσμα εξελικτικών δυνάμεων που δρώντας μεμονωμένα, προσπαθούν να μεγιστοποιήσουν τον αριθμό των απογόνων του κάθε είδους. Επιπλέον επιστήμονες έχουν προσπαθήσει να εξηγήσουν την εμφάνιση της επικοινωνίας στα ζώα, ενώ έχουν αναλύσει και την επιθετική συμπεριφορά τους.

Είναι εύκολα αντιληπτό ότι μπορούν να αναφερθούν πρακτικά άπειρες εφαρμογές της θεωρίας παιγνίων σε διάφορους τομείς ακόμη και στην καθημερινότητα μας, από πολύ πολύπλοκα προβλήματα έως τα πιο απλά όπως για παράδειγμα ποια καριέρα θα ακολουθήσουμε ποιο κινητό να αγοράσουμε ή και αν θα πάρουμε ομπρέλα φεύγοντας από το σπίτι.

2.3 ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ

Τα βασικά χαρακτηριστικά του παιγνίου αποτελούν Θεμέλιο λίθο στην θεωρία παιγνίων. Στοιχεία του παιγνίου αποτελούν το σύνολο των παικτών, το σύνολο των πιθανών ενεργειών που έχουν διαθέσιμες οι παίκτες (οι στρατηγικές τους), οι πληροφορίες που υπάρχουν κατά τη διάρκεια του παιχνιδιού, τα αποτελέσματα (ωφέλεια – ζημία) που μπορεί να αποκομίσει ο παίκτης για κάθε ενέργεια του, καθώς και οι προτιμήσεις των παικτών βάσει των αποτελεσμάτων. Το όφελος που μπορεί να αποκομίσει ο παίκτης (outcome), εξαρτάται από τις στρατηγικές που θα ακολουθήσει ο ίδιος και ο αντίπαλός του και από τις αποδόσεις που μπορεί να λάβει. Η απόδοση (payoff), αποτελεί την αριθμητική αποτίμηση των στόχων του, την χρησιμότητα με άλλα λόγια που θα αποκτήσει όταν το παιχνίδι θα τελειώσει.

Στρατηγική ορίζουμε το σύνολο των κανόνων σχετικά με την επιλογή που πρέπει να ακολουθήσει ο παίκτης, στο κάθε παίγνιο ξεχωριστά, λαμβάνοντας όμως πάντα υπόψη του και όλες τις δυνατές κινήσεις του αντιπάλου.

Πανεπιστήμιο Πειραιώς

Οι στρατηγικές μπορούν να διακριθούν σε αμιγείς “pure” και σε μεικτές “mixed” στρατηγικές. Σε μια αμιγή (καθαρή) στρατηγική κάθε μία από τις δυνατές επιλογές που έχει ο παίκτης επιλέγεται στο ακέραιο. Αντίθετα σε μια μεικτή στρατηγική περιλαμβάνονται συνδυασμοί επιλογών, από τις οποίες τουλάχιστον μία επιλέγεται με μη ακέραιες τιμές. Οι μεικτές στρατηγικές δηλαδή υποδεικνύουν ότι η στρατηγική που θα επιλέξει ο παίκτης θα γίνει τυχαία μεταξύ του συνόλου των καθαρών στρατηγικών που έχει, με κάποια πιθανότητα. Επομένως η μεικτή στρατηγική αποτελεί μια κατανομή πιθανοτήτων πάνω στις καθαρές στρατηγικές (διαθέσιμες επιλογές) που έχει ο κάθε παίκτης.

Μία ακόμα διάκριση αποτελεί η δυνατότητα των παικτών να παίζουν ταυτόχρονα. Η μορφή μπορεί να απεικονιστεί ως “κανονική” (normal) ή “στρατηγική” (strategic) χρησιμοποιώντας έναν πίνακα ο οποίος αντιστοιχεί τις διαθέσιμες στρατηγικές των παικτών με τις αποδόσεις που θα έχουν στην κάθε περίπτωση.

Στρατηγικό παιχνίδι ονομάζεται το μοντέλο στο οποίο έχουμε Ν παίκτες, καθένας από τους οποίους διαλέγει αποκλειστικά και μόνο μία στρατηγική, η οποία δεν μπορεί να αλλάξει. Σε ένα στρατηγικό παιχνίδι έχουν παρατηρηθεί πολλές διαφορετικές συμπεριφορές παικτών ανάλογα με τα υπόλοιπα χαρακτηριστικά του:

- Το παιχνίδι παίζεται μόνο μία φορά.
- Κάθε παίκτης έχει πλήρη γνώση του παιχνιδιού (κάθε παίκτης γνωρίζει όλες τις κινήσεις και τις αποδόσεις που αφορούν τον ίδιο και τον αντίπαλό του).
- Οι παίκτες δρουν ορθολογικά. Ένας ορθολογικός παίκτης είναι ο παίκτης που παίζει εγωιστικά, και έχει σκοπό να μεγιστοποιήσει το κέρδος του στο παιχνίδι, γνωρίζοντας πως και οι αντίπαλοι του είναι ορθολογιστές και έχουν τα ίδια ακριβώς κίνητρα.
- Όλοι οι παίκτες διαλέγουν τις κινήσεις τους την ίδια στιγμή χωρίς όμως να γνωρίζουν τις επιλογές των υπολοίπων παικτών.

Για την καλύτερη κατανόηση της κανονικής μορφής των παιγνίων, θα παραθέτουμε ένα παράδειγμα για να εξηγήσουμε περισσότερο τα στρατηγικά παίγνια.

A/B	b1	b2
a1	10,10	-200,8
a2	0,2	0,0

Πίνακας 2.1 Παίγνιο κυριαρχίας κινδύνου “Risk Dominance”

Το συγκεκριμένο παίγνιο είναι δύο παικτών και έτσι έχουμε δύο γραμμές και δύο στήλες αντίστοιχα. Οι δύο παίκτες θα ονομαστούν Α και Β. Ο παίκτης Α είναι ο “παίκτης γραμμής”, ενώ ο παίκτης Β “παίκτης στήλης”. Οι επικεφαλίδες των στηλών και των γραμμών αποτελούν τις στρατηγικές του κάθε παίκτη. Η πρώτη διαθέσιμη στρατηγική επιλογή του Α παίκτη είναι η πρώτη γραμμή, και ονομάζεται a1, ενώ η δεύτερη διαθέσιμη στρατηγική του είναι η a2.

Πανεπιστήμιο Πειραιώς

Ομοίως για τον παίκτη Β η πρώτη διαθέσιμη στρατηγική επιλογή του είναι η πρώτη στήλη, δηλαδή η b1, ενώ η δεύτερη διαθέσιμη στρατηγική του είναι η δεύτερη στήλη, η b2. Στα κελιά του κάθε πίνακα παρουσιάζονται ανά ζεύγος αριθμοί που αναπαριστούν το κέρδος (όφελος, payoff) του κάθε παίκτη για τον κάθε συνδυασμό στρατηγικών. Το πρώτο νούμερο του ζεύγους σε κάθε κελί αντιστοιχεί στο κέρδος του παίκτη γραμμής, ενώ το δεύτερο αντιστοιχεί στο κέρδος του παίκτη στήλης.

Ξεκινώντας το παιχνίδι οι παίκτες πρέπει να διαλέξουν ταυτόχρονα μία στρατηγική. Το κέρδος που αντιστοιχεί στους παίκτες αναπαρίσταται το σημείο τομής των δύο επιλογών των παικτών. Για παράδειγμα, αν ο Α παίκτης διαλέξει την πρώτη στρατηγική επιλογή (a1) μεταξύ των a1 και a2 και ο Β επίσης την πρώτη (b1) μεταξύ των b1 και b2 τότε το κέρδος τους θα είναι από 10 μονάδες για τον καθένα.

Οι παίκτες πριν αποφασίσουν και διαλέξουν ποια στρατηγική θα ακολουθήσουν, προσπαθούν να καταλάβουν ποια στρατηγική τους ωφελεί περισσότερο, άσχετα αν είναι δυνατόν από την επιλογή του αντιπάλου τους. Σε αυτό το σημείο η επιλογή θα γίνει με βάση την ύπαρξη κυρίαρχης στρατηγικής για τον παίκτη.

Κυρίαρχη ή “dominant” στρατηγική ονομάζεται η στρατηγική η οποία εάν για όλους τους συνδυασμούς στρατηγικών των άλλων παικτών έχει το μεγαλύτερο δυνατό όφελος σε σχέση με τις υπόλοιπες διαθέσιμες. Είναι πάντα η καλύτερη μεταξύ των διαθέσιμων ανεξάρτητα από την κίνηση που θα κάνει ο άλλος παίκτης αφού έχει το μεγαλύτερο δυνατό κέρδος σε σχέση με τις υπόλοιπες εναλλακτικές επιλογές του. Αντίθετα κυριαρχούμενη ή “dominated” στρατηγική ονομάζεται η στρατηγική για την οποία υπάρχει κάποια άλλη στρατηγική που είναι πάντα καλύτερη ανεξάρτητα από την κίνηση του άλλου παίκτη.

Στο παράδειγμα που παρουσιάσαμε παραπάνω παρατηρούμε οτι για τον Β παίκτη η στρατηγική b1 κυριαρχεί της στρατηγικής b2, αφού (10>8) και (2>0), δηλαδή είτε ο Α παίκτης διαλέξει την a1 στρατηγική είτε την a2, ο Β θα επιλέξει πάντα την b1. Επομένως η καλύτερη δυνατή επιλογή του είναι η b1 στρατηγική.

Δεν ισχύει όμως το ίδιο για τις στρατηγικές του παίκτη Α. Ο Α γνωρίζει πως αν ο Β επιλέξει την b1 στρατηγική, τον συμφέρει να διαλέξει την στρατηγική a1, αφού (10>0) εάν όμως ο Β διαλέξει την b2, ο Α έχει συμφέρον να επιλέξει την a2 και όχι την a1 αφού (-200<0). Επομένως για τον Α παίκτη δεν υφίσταται κυρίαρχη στρατηγική.

Αν για κάποιον παίκτη υπάρχει κυρίαρχη στρατηγική τότε αυτός την ακολουθεί ανεξάρτητα από την κίνηση του άλλου και τότε το παιχνίδι έχει λύση κυρίαρχης στρατηγικής. Όμως είναι πολύ πιθανό να μην υπάρχουν κυρίαρχες στρατηγικές αλλά μόνο να υπάρχουν ασθενείς κυριαρχίες.

Μια στρατηγική χαρακτηρίζεται ασθενώς κυρίαρχη “weakly dominant” εάν για κάθε μία από τις εναλλακτικές στρατηγικές που μπορεί να ακολουθήσει ο παίκτης υπάρχει τουλάχιστον μία

Πανεπιστήμιο Πειραιώς

στρατηγική με ίση απολαβή για όλους τους συνδυασμούς στρατηγικών των υπολοίπων παικτών και καλύτερη απολαβή για τουλάχιστον έναν συνδυασμό στρατηγικών των άλλων παικτών. Όλες οι υπόλοιπες εναλλακτικές στρατηγικές ονομάζονται ασθενώς κυριαρχούμενες “weakly dominated strategy”. Στο παραπάνω παίγνιο η στρατηγική a1 είναι ασθενώς κυρίαρχη της a2 αφού (10>-200) και (0=0).

Ο συνδυασμός και των δύο στρατηγικών που επιλέχθηκαν από τον κάθε παίκτη μας δίνει την έννοια της ισορροπίας “equilibrium”. Η ισορροπία στο παίγνιο προκύπτει από τις καλύτερες στρατηγικές του κάθε παίκτη στο παιχνίδι. Στο παραπάνω παράδειγμα η ισορροπία βρίσκεται στο κελί (a1, b1) δηλαδή στη λύση (10, 10) αφού η καλύτερη στρατηγική για τον Α παίκτη είναι η a1, για τον Β παίκτη η b1 και η τομή τους αποτελεί το κελί (a1, b1).

Για να εντοπίσουμε αυτήν την ισορροπία εντοπίζουμε εάν υπάρχει κυρίαρχη στρατηγική για κάποιον παίκτη και τότε επιλέγεται, όπως αναφέραμε και παραπάνω. Στην περίπτωση όμως που κυρίαρχη στρατηγική δεν υπάρχει, τότε ο σταδιακός περιορισμός των κυριαρχούμενων στρατηγικών “dominated” μπορεί να οδηγήσει στη δημιουργία νέων κυριαρχούμενων στρατηγικών, οι οποίες κατά τον ίδιο τρόπο θα απαλειφθούν. Ξεκινώντας το παιχνίδι ο κάθε παίκτης διαγράφει μία μια τις ασθενώς κυριαρχούμενες στρατηγικές από τις επιλογές του και αυτό συνεχίζεται μέχρι να βρεθεί για τον κάθε παίκτη μόνο μία στρατηγική.

Αυτή η διαδικασία εύρεσης σημείου ισορροπίας ονομάζεται απαλοιφή κυριαρχούμενων στρατηγικών “Iterated Elimination of Dominated Strategies, IEDS”. Η διαδικασία αυτή βασίζεται στην απαραίτητη παραδοχή πως οι παίκτες είναι ορθολογικοί και γνωρίζουν ότι και οι αντίπαλοι τους είναι ορθολογικοί γεγονός που δείχνει ότι κανένας από αυτούς δεν θα έχει συμφέρον να επιλέξει μια στρατηγική η οποία θα είναι ασθενώς κυριαρχούμενη. Αν απαλειφθούν μόνο κυριαρχούμενες στρατηγικές, το αποτέλεσμα δεν θα επηρεαστεί από τη σειρά της απαλοιφής. Ο κίνδυνος να καταλήξουμε σε λάθος αποτέλεσμα υπάρχει μόνο αν η απαλοιφή γίνει με λάθος σειρά επί των ασθενώς κυριαρχούμενων στρατηγικών. Σωστή σειρά αποτελεί η ταυτόχρονη απαλοιφή των κυριαρχούμενων στρατηγικών για όλους τους παίκτες σε κάθε γύρο.

Στη συνέχεια θα επανέλθουμε και θα αναλύσουμε την σημαντικότερη έννοια ισορροπίας στη θεωρία παιγνίων την ισορροπία Nash.

Πανεπιστήμιο Πειραιώς

2.4 ΚΑΤΗΓΟΡΙΕΣ ΠΑΙΓΝΙΩΝ

Η ταξινόμηση των παιγνίων σε διάφορες κατηγορίες μπορεί να γίνει με βάση διάφορα είδη κριτηρίων. Εδώ θα γίνει μια προσπάθεια να τα κατηγοριοποιήσουμε με βάση τα κυριότερα χαρακτηριστικά τους. Έτσι ένα παίγνιο μπορεί να κατηγοριοποιηθεί:

- Σύμφωνα με τον αριθμό των παικτών που παίρνουν μέρος. Αν το πλήθος των παικτών είναι δύο τότε ονομάζονται “παίγνια δύο παικτών”, ενώ αν οι παίκτες είναι περισσότεροι (έστω n), τότε έχουμε “παίγνια n παικτών”. Στην περίπτωση που υπάρχει μόνο ένας παίκτης έχοντας σαν αντίπαλο του “τη φύση”, όπως για παράδειγμα ισχύει στην πασιέντζα θεωρούνται πως ανήκουν στην πρώτη κατηγορία των παιγνίων με δύο παίκτες.

- Σύμφωνα με τη δυνατότητα συνεργασίας. Εάν οι παίκτες (δύο ή περισσότεροι) πριν την έναρξη του παιγνίου έχουν τη δυνατότητα να συνεργαστούν και να συμφωνήσουν μεταξύ τους για τις στρατηγικές που θα ακολουθήσουν τα παίγνια ονομάζονται “συνεργατικά παίγνια” (cooperative games) σε αντίθεση με τα παίγνια όπου ο κάθε παίκτης αποφασίζει χωρίς να έχει προηγηθεί συνεννόηση με τους υπόλοιπους, και τα οποία ονομάζονται “μη συνεργατικά παίγνια” (non cooperative games).

- Σύμφωνα με τα χαρακτηριστικά των αποδοχών τους. Όταν το κέρδος ενός παίκτη ισούται με την απώλεια του αντιπάλου του, το παίγνιο θεωρείται “παίγνιο μηδενικού αθροίσματος” (zero-sum games). Σε αυτή την κατηγορία παιγνίων το άθροισμα των αμοιβών είναι ίσο με μηδέν με προφανές αποτέλεσμα η συνεργασία για τους παίκτες να είναι ανούσια. Αντίστοιχα όταν το άθροισμα των αμοιβών είναι διάφορο του μηδενός έχουμε “παίγνια μη-μηδενικού αθροίσματος” (non zero-sum games) στα οποία το κέρδος κάποιου δεν αποτελεί τη ζημιά κάποιου ανταγωνιστή, και οι δύο μπορεί να κερδίσουν ή και να χάσουν αντίστοιχα.

- Σύμφωνα με τη σειρά που παίρνονται οι αποφάσεις. Αν οι αντίπαλοι επιλέγουν ταυτόχρονα μια στρατηγική στην αρχή του παιχνιδιού, χωρίς να γνωρίζει ο καθένας την κίνηση του άλλου, τότε μιλάμε για “στατικό παίγνιο” ή “στρατηγικό παίγνιο” ή “παίγνιο σε κανονική μορφή”. Στην αντίθεση περίπτωση έχουμε τα “δυναμικά παίγνια” ή “παίγνια σε εκτεταμένη μορφή” όπου οι παίκτες έχουν αποκτήσει κάποια γνώση από τις προηγούμενες ενέργειες και έτσι η σειρά με την οποία λαμβάνονται οι αποφάσεις έχει σημασία. Στα παίγνια αυτά γίνεται η χρήση δέντρου για την αναπαράσταση των κινήσεων.

- Σύμφωνα με τον αριθμό των στρατηγικών. Τα παίγνια κατηγοριοποιούνται σε “πεπερασμένα” και σε “μη πεπερασμένα”. Τα πεπερασμένα παίγνια έχουν ένα μετρήσιμο αριθμό κινήσεων, σε αντίθεση με τα “μη πεπερασμένα” τα οποία μπορούν να έχουν άπειρες κινήσεις και ο νικητής γίνεται γνωστός αφού όλες αυτές οι κινήσεις τελειώσουν.

- Τέλος σύμφωνα με την πληροφόρηση που παρέχουν. Τα παίγνια χαρακτηρίζονται “παίγνια πλήρους πληροφόρησης” όταν όλοι οι παίκτες είναι πλήρως ενημερωμένοι για τις κινήσεις των αντιπάλων. Παίγνια πλήρους πληροφόρησης μπορούν να είναι μόνο τα δυναμικά παίγνια, αφού στα στατικά οι παίκτες δεν είναι ενημερωμένοι. Όταν οι παίκτες είναι μερικώς ενημερωμένοι τότε λέμε ότι έχουμε “παίγνια ατελούς πληροφόρησης”.

Πανεπιστήμιο Πειραιώς

2.5 Η ΙΣΟΡΡΟΠΙΑ NASH

Στους βασικούς θεμελιωτές της θεωρίας παιγνίων ανήκει ο John Nash ο οποίος εισήγαγε για πρώτη φορά την ιδέα της ισορροπίας η οποία χρησιμοποιείται πλέον ευρέως σε όλους τους κλάδους της σύγχρονης επιστήμης. Ο John Nash ασχολήθηκε αρχικά με "προβλήματα συμφωνιών", δηλαδή προβλήματα στα οποία οι παίκτες μοιράζονται κάποια κοινά συμφέροντα. Ασχολήθηκε με την ισορροπία στη θεωρία παιγνίων και από εκείνον προέρχεται η έννοια της "Nash ισορροπίας" στην οποία αποδεικνύεται η ύπαρξη λύσεων σε παίγνια ν παικτών. Ο John Nash ανακάλυψε μια γενική "λύση" για όλα τα πεπερασμένα παίγνια και απέδειξε ότι κάθε τέτοιο παίγνιο διαθέτει τουλάχιστον μια λύση.

2.5.1 Προσέγγιση της ισορροπίας Nash

Το θεώρημα που διατύπωσε ο Nash και έγινε παγκοσμίως γνωστό αναφέρει πως κάθε παίγνιο με συγκεκριμένο πλήθος παικτών και ενεργειών έχει τουλάχιστον ένα σημείο ισορροπίας, στο οποίο όλοι οι παίκτες επιλέγουν τις πιο συμφέρουσες για αυτούς ενέργειες, λαμβάνοντας υπόψη τους και τις επιλογές των αντιπάλων τους. Οι παίκτες υπολογίζουν τις κινήσεις του αντιπάλου τους, προσπαθούν να καταλάβουν τη συμπεριφορά τους και σύμφωνα με αυτό κάνουν την επιλογή της στρατηγικής. Δηλαδή η στρατηγική του κάθε παίκτη αποτελεί την καλύτερη δυνατή αντίδραση (απόκριση) στην στρατηγική του άλλου παίκτη. Αυτός ο συνδυασμός στρατηγικών αποτελεί την ισορροπία Nash.

Ο παίκτης επιλέγει από τις δικές του στρατηγικές εκείνη, η οποία είναι η καλύτερη δυνατή απάντηση στην στρατηγική που υπολογίζει ότι θα επιλέξει ο άλλος παίκτης. Επομένως δεν υπάρχει κίνητρο για κανέναν παίκτης να φύγει μονομερώς από αυτήν την ισορροπία που έχει δημιουργηθεί. Οι παίκτες αναγνωρίζουν πως βρίσκονται σε ισορροπία αν οποιαδήποτε αλλαγή στις στρατηγικές από οποιονδήποτε από αυτούς, οδηγήσει σε χαμηλότερη ωφέλεια από αυτή που θα είχαν αν παρέμεναν στη προηγούμενη στρατηγική τους. Δεδομένων των διαθέσιμων επιλογών των αντιπάλων, ο κάθε παίκτης δεν έχει κάτι περισσότερο να κερδίσει και για αυτό δεν αλλάζει στρατηγική.

Η θεωρία για την ισορροπία Nash, βασίζεται σε δύο συνιστώσες:

Κατ αρχήν κάθε παίκτης επιλέγει βασιζόμενος στην ορθολογική απόφαση που προέρχεται από τις πεποιθήσεις του για το τι θα πράξει ο αντίπαλος και δεύτερον κάθε πεποίθηση του παίκτη για την επιλογή του αντιπάλου του είναι ορθά υπολογισμένη.

Για να κατανοήσουμε περισσότερο την έννοια της ισορροπίας Nash, θα χρησιμοποιήσουμε πάλι το ανωτέρω παίγνιο το οποίο παραθέτουμε πάλι για ευκολία.

A/B	b1	b2
a1	10,10	-200,8
a2	0,2	0,0

Πίνακας 2.2 Παίγνιο κυριαρχίας κινδύνου "Risk Dominance"

Πανεπιστήμιο Πειραιώς

Ξεκινώντας με τον Α παίκτη βρίσκουμε την στρατηγική που θα επιλέξει σε κάθε δυνατή στρατηγική του αντιπάλου. Έστω ότι ο Α παίκτης πιστεύει ότι ο Β θα επιλέξει την b1 στρατηγική. Τότε προφανώς θα επιλέξει από τις δύο δικές του στρατηγικές εκείνη που θα του δώσει το μεγαλύτερο δυνατό όφελος. Η a1 θα του δώσει 10 μονάδες ωφέλειας, ενώ η a2 θα του δώσει 0 (όπως αναφέραμε και παραπάνω οι πρώτοι αριθμοί του κάθε κελιού αντιστοιχούν στον παίκτη γραμμής, δηλαδή στον Α). Άρα θα επιλέξει την a1 στρατηγική με κέρδος 10. Αυτό το νούμερο το κρατάμε. Αν ο Α παίκτης από την άλλη πιστεύει πως ο Β θα διαλέξει την b2 στρατηγική αυτός λογικά σκεπτόμενος θα προτιμήσει την a2 αφού το κέρδος του θα είναι μεγαλύτερο (-200<0), άσχετα αν πρόκειται για 0 μονάδες. Τα νούμερα που επιλέξαμε τα χρωματίζουμε με κόκκινο χρώμα.

Ύστερα από τις επιλογές του παίκτη Α, ο πίνακας παρουσιάζεται ως εξής:

A/B	b1	b2
a1	10,10	-200,8
a2	0,2	0,0

Πίνακας 2.3 Πρώτο στάδιο του παιγνίου

Ομοίως προχωρούμε και για τον παίκτη Β. Αν αυτός πιστεύει ότι ο Α θα επιλέξει την a1 στρατηγική, τότε θα προτιμήσει την b1 στρατηγική που θα του δώσει κέρδος 10 μονάδες και όχι 8 μονάδες (οι δεύτεροι αριθμοί σε κάθε κελί αναφέρονται στον παίκτη στήλης, δηλαδή στον Β). Αν ο Β πιστεύει για τον Α πως θα επιλέξει την a2 στρατηγική, τότε και πάλι θα επιλέξει την b1 αφού θα έχει κέρδος 2 μονάδες αντί για 0 μονάδες. Αυτά τα νούμερα τα χρωματίζουμε με μπλε χρώμα.

Ύστερα και από τις επιλογές του Β παίκτη ο πίνακας έχει ως εξής:

A/B	b1	b2
a1	10,10	-200,8
a2	0,2	0,0

Πίνακας 2.4 Δεύτερο στάδιο του παιγνίου

Η ισορροπία Nash επιτυγχάνεται όταν η καλύτερη επιλογή του παίκτη Α είναι ίδια με την καλύτερη επιλογή του παίκτη Β, όταν δηλαδή σε ένα κελί υπάρχουν οι επιλογές και των δύο παικτών. Αυτό αποτελεί και το σημείο ισορροπίας. Στο παράδειγμα μας ισορροπία έχουμε στο κελί (a1, b1)=(10, 10).

Πρέπει αν αναφερθεί ότι σε μερικά παιχνίδια παρατηρούνται περισσότερες από μία ισορροπίες Nash, ενώ υπάρχουν και παιχνίδια στα οποία δεν μπορεί να υπάρχει κανένα σημείο ισορροπίας Nash.

Πανεπιστήμιο Πειραιώς

Έχουμε αναφέρει πως οι στρατηγικές μπορεί να είναι καθαρές ή μικτές. Η επιλογή μικτής στρατηγικής ισοδυναμεί με το να επιλέξει ο παίκτης με βάση κάποια πιθανότητα τυχαία μεταξύ συγκεκριμένων καθαρών στρατηγικών. Για παράδειγμα μπορεί σε κάποιο σενάριο να πούμε πως ο παίκτης Α θα επιλέξει την a1 στρατηγική με πιθανότητα p ή την a2 με πιθανότητα 1-p. Ο παίκτης δηλαδή που χρησιμοποιεί κάποια μικτή στρατηγική επιλέγει τις πιθανότητες της καθεμιάς από τις διαθέσιμες καθαρές στρατηγικές που εμπεριέχονται στην συγκεκριμένη μικτή στρατηγική, αφήνοντας τα υπόλοιπα στην τύχη. Στην καθημερινή ζωή η χρήση μικτών στρατηγικών, όσο και αν φαίνεται παράξενο αποτελεί πολύ συχνό φαινόμενο μεταξύ των παικτών.

Ο Nash επίσης απέδειξε πως κάθε πεπερασμένο παίγνιο εμπεριέχει τουλάχιστον ένα σύνολο μικτών στρατηγικών (μία ανά παίκτη) που αποτελεί ισορροπία Nash σε μικτές στρατηγικές (ΙΝΜΣ) Όταν υπάρχουν περισσότερες της μίας ισορροπίες Nash (με καθαρές στρατηγικές), η λύση βρίσκεται στην ισορροπία Nash με μικτές στρατηγικές. Ακόμη δε και αν δεν υπάρχει ισορροπία σε καθαρές στρατηγικές, υπάρχει πάντα μία μοναδική ισορροπία σε μικτές στρατηγικές. Βέβαια αν και η ισορροπία σε καθαρές στρατηγικές φαίνεται πιο ελκυστική πρόταση από ότι η ισορροπία στις μικτές, αφού δεν βασίζεται στις τυχαίες επιλογές των παικτών, από τη στιγμή που δεν υπάρχει σε κάθε παιχνίδι ισορροπία σε καθαρές στρατηγικές, η ισορροπία σε μικτές στρατηγικές αποκτάει μεγαλύτερη αξία αφού δίνει για κάθε παιχνίδι τουλάχιστον μία ισορροπία.

2.6 ΕΞΕΤΑΣΗ ΔΙΑΦΟΡΩΝ ΠΑΙΓΝΙΩΝ

Ένα από τα παράδοξα της ισορροπίας Nash που μπορεί να θεωρηθεί και σαν αδυναμία της είναι ότι υπάρχει περίπτωση σε κάποια παίγνια οι παίκτες έχουν αποκτούν μεγαλύτερο όφελος αν δεν επιλέξουν την ισορροπία Nash αλλά επιλέξουν κάποια άλλη στρατηγική. Ενώ η ισορροπία Nash δίνει την ελκυστικότερη λύση για όλους τους παίκτες συνολικά, οδηγώντας στο σημείο ισορροπίας, υπάρχουν κάποια διάσημα παίγνια που αποτελούν την εξαίρεση στον κανόνα. Ενδεικτικά θα παρουσιάσουμε κάποια από αυτά τα παίγνια και θα αναλυθούν στη συνέχεια.

2.6.1 Το δίλημμα του φυλακισμένου “Prisoner’s dilemma”

Το πιο διάσημο παίγνιο στην ιστορία της θεωρίας παιγνίων είναι το παίγνιο του διλήμματος του φυλακισμένου (Prisoner’s dilemma).

Το συγκεκριμένο παίγνιο επινοήθηκε τον Ιανουάριο του 1950 από τους Melvin Dresher και Merrill Flood και χρησιμοποιήθηκε σαν παράδειγμα στο RAND Corporation. Αργότερα όταν αυτό το παράδειγμα παρουσιάστηκε σε ένα σεμινάριο στο Stanford University, ο Albert W. Tucker σκαρφίστηκε μία ιστορία πάνω στην οποία βάσισε όλη του την διάλεξη. Το παίγνιο αυτό έμεινε στην ιστορία από τότε κάνοντας την θεωρία παιγνίων γνωστή σε όλες τις κοινωνικές

Πανεπιστήμιο Πειραιώς

επιστήμες, ενώ και παράλληλα πολλοί μελετητές έχουν ασχοληθεί με αυτό γράφοντας διάφορα βιβλία.

Η ιστορία του Tucker έχει ως εξής:

Η αστυνομία συλλαμβάνει δύο υπόπτους για ένα έγκλημα και τους κρατά σε διαφορετικά κελιά, ώστε να μην έχουν μεταξύ τους επικοινωνία. Οι αστυνομικοί είναι βέβαιοι για την ενοχή τους αλλά λόγω έλλειψης αποδεικτικών στοιχείων τους προσφέρουν μια συμφωνία: αν και οι δύο ομολογήσουν ότι διέπραξαν το έγκλημα θα καταδικαστούν σε τρία χρόνια φυλάκισης. Αν μόνο ο ένας ομολογήσει και ο άλλος αρνηθεί, αυτός που ομολόγησε θα αφεθεί ελεύθερος ενώ ο άλλος που θα φυλακιστεί για πέντε χρόνια. Τέλος, αν κανείς δεν ομολογήσει, θα περάσουνε και οι δύο έναν χρόνο στη φυλακή.

Το παραπάνω πρόβλημα μπορεί να παρουσιαστεί στον επόμενο πίνακα

A/B	b1 ομολογεί	b2 δεν ομολογεί
a1 ομολογεί	3 χρόνια φυλακή	ελευθερία , 5 χρόνια
a2 δεν ομολογεί	5 χρόνια , ελευθερία	1 χρόνος φυλακή

Πίνακας 2.5 Το δίλημμα του φυλακισμένου(αρχική μορφή)

Το δίλημμα αυτό μπορεί να απεικονιστεί εναλλακτικά με τη μορφή του παρακάτω παιγνίου, όπου τα νούμερα είναι η ωφέλεια που αποκομίζει ο παίκτης.

A/B	b1 ομολογεί	b2 δεν ομολογεί
a1 ομολογεί	1,1	5,0
a2 δεν ομολογεί	0,5	3,3

Πίνακας 2.6 Το δίλημμα του φυλακισμένου (τελική μορφή)

Το δίλημμα εμφανίζεται με την υπόθεση ότι και οι δύο φυλακισμένοι νοιάζονται μόνο για να ελαχιστοποιήσουν την ποινή τους. Κάθε παίκτης έχει δύο εναλλακτικές επιλογές : είτε να ομολογήσει και να συνεργαστεί με την αστυνομία (confess), είτε να παραμείνει σιωπηλός (not confess). Για παράδειγμα το καλύτερο σενάριο για τον παίκτη Α είναι να ομολογήσει και ο παίκτης Β να μην ομολογήσει. Το επόμενο καλύτερο αποτέλεσμα για τον Α είναι να μην ομολογήσει κανένας από τους δύο, ενώ το χειρότερο δυνατό σενάριο είναι να ομολογήσει ο Β ενώ ο Α να μην ομολογήσει. Το ίδιο ισχύει και για τον παίκτη Β. Όπως γίνεται αντιληπτό οτιδήποτε και να σκοπεύει να κάνει ο Β, ο παίκτης Α έχει συμφέρον να επιλέξει την πρώτη στρατηγική (να ομολογήσει δηλαδή), αφού έτσι θα έχει καλύτερα αποτελέσματα για τον ίδιο. Το ίδιο ισχύει και για τον Β παίκτη ο οποίος θα προτιμήσει και αυτός να ομολογήσει. Σε αυτό το

Πανεπιστήμιο Πειραιώς

σημείο υπάρχει το δίλημμα αφού βάσει του πίνακα φαίνεται πως οι παίκτες θα αποκομίσουν μεγαλύτερο όφελος αν και οι δύο επιλέξουν να μην ομολογήσουν από το να τα ομολογήσουν όλα. Έτσι η καλύτερη στρατηγική ξεχωριστά για τον καθένα, παράγει ένα αποτέλεσμα που δεν είναι το καλύτερο δυνατό για την ομάδα, κάνοντας τα ατομικά κίνητρα να υπονομεύουν το κοινό συμφέρον.

Πρόκειται για ένα παιχνίδι όπου τα κέρδη βελτιστοποιούνται από τη συνεργασία. Το καλύτερο δυνατό αποτέλεσμα και για τους δύο παίκτες μαζί είναι να μην ομολογήσουν στους αστυνομικούς. Παρόλα αυτά, το κίνητρο για κάθε παίκτη να γίνει προδότης είναι μεγάλο. Οτιδήποτε και αν επιλέξει να κάνει ο ένας παίκτης, ο αντίπαλος έχει πάντα κίνητρο να ομολογήσει. Έτσι το παίγνιο αυτό έχει μία και μοναδική Nash ισορροπία, μία κυρίαρχη στρατηγική για τον καθένα, η οποία είναι η λύση (A1,B1)=(1,1), η από κοινού ομολογία.

Το παράδοξο του αποτελέσματος μπορεί εν μέρει να εξηγηθεί από το γεγονός ότι οι φυλακισμένοι βρίσκονται απομονωμένοι σε ξεχωριστά κελιά και δεν μπορούν να έχουν επικοινωνία μεταξύ τους προκειμένου να αποφασίσουν από κοινού τι θα κάνουν. Αν μπορούσαν να έρθουν σε επαφή και να το συζητήσουν και να εμπιστευθούν ο ένας τον άλλο ίσως να έβλεπαν πως η καλύτερη λύση είναι να μη μιλήσει κανένας τους.

Ακόμη και με μια προφορική συμφωνία είναι πολύ πιθανό οι φυλακισμένοι να προσπαθήσουν να προδώσουν τον υποτιθέμενο αντίπαλο τους, προλαβαίνοντας τον από μια αντίστοιχη πιθανή προδοσία. Εδώ επέρχεται ο παράγοντας της αξιοπιστίας: υπάρχει μια έφεση προς συνεργασία μόνο με εκείνους που πιστεύουμε ότι έχουν αντίστοιχη έφεση να συνεργαστούν. Σε πραγματικές συνθήκες πάντως έχει παρατηρηθεί ότι δύσκολα κάποιος από τους δύο θα επιλέξει να προδώσει τον άλλο, αφού η σιωπή αποτελεί ύψιστη τιμή σε τέτοιες κοινωνικές ομάδες.

Μια άλλη περίπτωση είναι οι δύο ύποπτοι να επιλέξουν να μην ομολογήσουν, αν έχουν ξαναπεράσει από παρόμοιες καταστάσεις και γνωρίζουν πως δεν πρόκειται να προδοθούν. Αυτή η ισορροπία λέγεται “υπό-παιγνιακή τέλεια ισορροπία Nash”. Οι φυλακισμένοι έχουν μάθει ότι το να μην καρφώνουν ο ένας τον άλλον έχει τα καλύτερα αποτελέσματα και για τους δύο.

Μια παραλλαγή του διλήμματος του φυλακισμένου είναι όταν αφορά πάνω από δύο πρόσωπα και ονομάζεται free rider problem (το πρόβλημα των τζαμπατζήδων). Έχει παρόμοια δομή με το δίλημμα του φυλακισμένου αφού για μια ακόμα φορά η κυρίαρχη ατομική στρατηγική υπερέχει της κοινής λογικής. Αφορά κυρίως τις περιπτώσεις δημοσίων αγαθών (όλοι τα χρησιμοποιούν άσχετα αν έχουν πληρώσει γι’αυτά, όπως για παράδειγμα το οδικό δίκτυο) όπου η πρόσβαση δεν μπορεί να περιοριστεί σε αυτούς που έχουν συνεισφέρει και στους άλλους, τους free riders, οι οποίοι δεν συνεισφέρουν αλλά ωφελούνται από τη χρήση τους.

Το πιο γνωστό παιχνίδι στην ιστορία της θεωρίας παιγνίων μελετήθηκε σε μεγάλο βαθμό από πάρα πολλούς ανθρώπους, ανάμεσα τους ο John Nash (που αναφέρθηκε παραπάνω) και ο Robert Axelrod. Στα τέλη της δεκαετίας του 70 ο Axelrod προσπάθησε να προσεγγίσει το πρόβλημα όταν γίνονται διαδοχικές επαναλήψεις, αφού έτσι γίνεται πιο περίπλοκο και δεν είναι απόλυτα σαφές ποια στρατηγική είναι βέλτιστη. Έτσι λοιπόν οργάνωσε ένα πρωτάθλημα όπου κάλεσε θεωρητικούς των παιγνίων να δημιουργήσουν αλγορίθμους που να περιέχουν από μία στρατηγική και τους έβαλε να διαγωνιστούν για έναν καθορισμένο αριθμό γύρων. Οι “άπληστες” στρατηγικές έτειναν να έχουν άσχημη έκβαση, σε αντίθεση με τις πιο αλτρουιστικές που τα

Πανεπιστήμιο Πειραιώς

πήγαν καλύτερα. Νικητής αναδείχτηκε ο Anatol Rapoport που δημιούργησε τον πιο απλό αλγόριθμο, τον Tit for Tat, δηλαδή “μία σου και μία μου”.

Πρόκειται για μία στρατηγική δεσμευμένης συνεργασίας όπου ο παίκτης σαν κίνηση καλής θέλησης ξεκινάει με συνεργασία, και στη συνέχεια αντιγράφει την στρατηγική που επέλεξε ο αντίπαλος στον προηγούμενο γύρο. Το πείραμα επαναλήφθηκε και για την περίπτωση όπου η ακολουθία των αγώνων μεταξύ των δύο παικτών θα τερματιζόταν τυχαία με νικητή πάλι τον ίδιο αλγόριθμο. Το πλεονέκτημα αυτής της στρατηγικής έχει να κάνει με τον συνδυασμό αυστηρότητας απέναντι στους αποστάτες αφού τους τιμωρείς άμεσα αλλά και ηπιότητας αφού μέσα σε έναν γύρο μπορείς να τον συγχωρήσεις. Τελικά φαίνεται πως αυτός που δεν συμπεριφέρεται εγωιστικά, είναι αυτός που κερδίζει.

Το δίλημμα του φυλακισμένου αν και δεν φαίνεται άμεσα σχετικό με την καθημερινότητα του ανθρώπου, μπορούμε να το διακρίνουμε σε όλα τα κοινωνικά φαινόμενα.

Το παίγνιο αυτό αναλύεται σε μία τεράστια βιβλιογραφία και μάλιστα πιστεύεται πως αποτελεί τον κεντρικό πυρήνα της κοινωνικής ζωής. Οι εφαρμογές του λοιπόν στην καθημερινή ζωή ποικίλλουν από την οικονομία, την πολιτική και την κοινωνιολογία έως την ανθρωπολογία και την εξελικτική βιολογία.

Στην εξωτερική πολιτική για παράδειγμα αυτό το παίγνιο χρησιμοποιείται για να περιγράψει το πρόβλημα που έχουν δύο κράτη σχετικά με την απόκτηση όπλων. Τα κράτη έχουν δύο στρατηγικές επιλογές: μπορούν είτε να αυξήσουν την στρατιωτική τους δύναμη αγοράζοντας καινούριο εξοπλισμό, είτε μπορούν να κάνουν μια συμφωνία έτσι ώστε να μειώσουν την χρήση όπλων. Δεν υπάρχει βεβαιότητα για κανένα κράτος ότι το άλλο θα κρατήσει την υπόσχεση του και επομένως και τα δύο έχουν κίνητρο να αγοράσουν τελικά τα όπλα. Χαρακτηριστικό παράδειγμα για αυτήν την περίπτωση αποτελεί η διαμάχη μεταξύ της Αμερικής και της Ρωσίας τη δεκαετία του 50 (όταν πρωτομελετήθηκε το συγκεκριμένο παίγνιο) για την απόκτηση πυρηνικών όπλων.

Στον αθλητισμό πολλοί παλαιστές καταφεύγουν στο προσωρινό χάσιμο πολλών κιλών με σκοπό να διαγωνιστούν με ελαφρύτερους (πιο εύκολους) αντιπάλους, πηγαίνοντας στην μικρότερη κατηγορία. Σε αυτό το τρικ μπορεί να καταφύγουν πολλοί διαγωνιζόμενοι με αποτέλεσμα να υποβαθμίζεται ο συναγωνισμός. Ακόμη όμως και στην περίπτωση που κάποιος διαγωνιζόμενος παραμείνει στο αρχικό του βάρος, υπάρχει η πιθανότητα να συναγωνιστεί κάποιον που έχει χάσει βάρος.

Είναι φανερό πως το δίλημμα του φυλακισμένου υπάρχει σε κάθε συναλλαγή ή σύγκρουση ατομικών συμφερόντων. Τα παραδείγματα είναι πολλά από τα πολιτικά παιχνίδια και τους πλειστηριασμούς έως την συμπεριφορά των αθλητών σε αγώνες και την επιλογή που έχουν δύο αντιμαχόμενα μέρη για το αν θα χρησιμοποιήσουν δικηγόρους και θα καταφύγουν στα δικαστήρια για να λύσουν τις διαφορές τους ή θα συμβιβαστούν μεταξύ τους. Το κοινό στοιχείο

Πανεπιστήμιο Πειραιώς

των παραπάνω παραδειγμάτων είναι ότι αν ο καθένας δράσει συλλογικά συνεργατικά θα υπάρξει το καλύτερο συνολικό αποτέλεσμα. Δυστυχώς η πλειοψηφία προτιμά να διαλέξει το προσωπικό συμφέρον, με αποτέλεσμα να οδηγούνται σε μη βέλτιστα αποτελέσματα.

2.6.2 Η μάχη των φύλων “Battle of the Sexes”

Το παίγνιο “battle of the sexes” (η μάχη των φύλων) αποτελεί ένα ακόμα κλασσικό παιχνίδι στη θεωρία παιγνίων. Στην παραδοσιακή μορφή του παιχνιδιού, το οποίο χρονολογείται από τη δεκαετία του `50, ένας άντρας και μια γυναίκα θέλουν να αποφασίσουν πως θα περάσουν το βράδυ τους. Ο άντρας προτιμά να κάτσουν σπίτι και να δουν τον αγώνα που έχει στην τηλεόραση. Η γυναίκα προτιμά να πάνε στην όπερα. Και για τους δύο όμως είναι σημαντικό να κάνουν κάτι μαζί και όχι να μείνουν χώρια.

Στον παρακάτω πίνακα αναλύουμε τις επιλογές που έχουν ως στρατηγικές, όπου οι γραμμές αντιστοιχούν στις επιλογές του άντρα και οι στήλες στις επιλογές της γυναίκας .

A/B	b1 sports	b2 opera
a1 sports	2,1	0,0
a2 opera	0,0	1,2

Πίνακας 2.7 Η μάχη των φύλων

Η μάχη των φύλων παρουσιάζει ένα παίγνιο κατά το οποίο το ζευγάρι πρέπει να συνεργαστεί, άσχετα από το αν έχουν διαφορετικές προτιμήσεις, αφού και οι δύο βγαίνουν χαμένοι αν μείνουν χώρια. Πρόκειται για κλασική μορφή συνεργατικού παιγνίου. Εδώ μας ωφελεί να γνωρίζει ο αντίπαλος τη στρατηγική που πρόκειται να ακολουθήσουμε, γιατί μπορεί να τη χρησιμοποιήσει για κοινό μας όφελος.

Αν και το παιχνίδι ανήκει στην κατηγορία των παιχνιδιών που οι παίκτες πρέπει να παίξουν ταυτόχρονα, δεν είναι αναγκαίο ότι στην πραγματικότητα οι παίκτες θα δράσουν έτσι. Το μόνο που απαιτείται είναι ο καθένας από τους δύο να δράσει χωρίς να γνωρίζει το τι θα επιλέξει ο άλλος.

Αυτό μπορεί να επιτευχθεί αν οι παίκτες πάρουν την απόφαση τους χωρίς προηγουμένως να έχουν έρθει σε συνεννόηση. Είναι μη ρεαλιστικό να υποθέσουμε όμως πως το ζευγάρι δεν θα συζητήσει και δεν θα επαναληφθεί ίδιο «έργο» πολλές φορές. Αν κάθε μέρα έχουν να πάρουν μια παρόμοια απόφαση (επαναλαμβανόμενο παίγνιο) τότε σίγουρα ο καθένας θα μπορεί να προβλέψει τις κινήσεις του άλλου.

Το σημαντικότερο ρόλο σε αυτό το παιχνίδι τον έχει αυτός που θα παίξει πρώτος και θα ανακοινώσει την απόφαση του στο ταίρι του. Αν για παράδειγμα η γυναίκα ανακοινώσει πως έχει αγοράσει τα εισιτήρια για την όπερα προτού ο άντρας πεί ότι θέλει να παρακολουθήσει τον αγώνα, είναι πολύ πιθανό ο άντρας να πεισθεί και να επιλέξει από την αρχή να πάνε στην όπερα. Σε πάρα πολλές περιπτώσεις (αλλά όχι σε όλες) αυτός που κινείται πρώτος αποκτά και το πλεονέκτημα.

Πανεπιστήμιο Πειραιώς

Φαίνεται αμέσως πως σε αυτό το παίγνιο δεν υπάρχει κάποια κυρίαρχη στρατηγική για κανέναν από τους παίκτες. Εντοπίζουμε όμως δύο σημεία ισορροπίας Nash στο συγκεκριμένο παίγνιο, τη λύση (A1,B1)=(2,1) και τη λύση (A2,B2)=(1,2). Αν και οι δύο επιλέξουν να κάτσουν σπίτι και να δούνε αγώνα ο άντρας έχει όφελος 2 μονάδες και η γυναίκα 1 μονάδα, ενώ αν επιλέξουν και οι δύο να πάνε στην όπερα η γυναίκα έχει όφελος 2 μονάδες και ο άντρας 1. Και στις δύο στρατηγικές κανένας δεν έχει κίνητρο να επιλέξει κάτι άλλο.

2.6.3 Το παίγνιο “Chicken Game”

Ένα άλλο πολύ γνωστό παίγνιο είναι το Chicken Game. Το παιγνίδι αυτό είναι γνωστό από τη δεκαετία του `50 και μετά στην Αμερική και έχει γίνει ευρέως γνωστό από την ταινία «Επαναστάτης χωρίς αιτία» (Rebel without a cause,1955) με πρωταγωνιστή τον James Dean. Σε αυτό το παιχνίδι δύο οδηγοί οδηγούν με ταχύτητα προς τον γκρεμό. Όποιος αλλάξει πρώτος την πορεία του αυτοκινήτου του για να μην πέσει είναι το «κοτόπουλο» (chicken) και χάνει. Αν κανένας παίκτης δεν αποφασίσει να αλλάξει πορεία, τότε και τα δύο αυτοκίνητα θα πέσουν από τον γκρεμό και όλοι θα χάσουν.

Η παραπάνω κατάσταση μπορεί να περιγραφεί με τον ακόλουθο πίνακα:

A/B	b1 driving straight	b2 swerving
a1 driving straight	0,0	3,1
a2 swerving	1,3	2,2

Πίνακας 2.8 Chicken Game

Για κάθε παίκτης υπάρχουν δύο στρατηγικές επιλογές. Ο παίκτης μπορεί είτε να συνεχίσει να οδηγεί προς τον γκρεμό (πρώτη στρατηγική), είτε να αποκλίνει από την πορεία του (δεύτερη στρατηγική). Αν και οι δύο αποκλίνουν παραμένουν ζωντανοί. Το τι θα επιλέξουν εξαρτάται από το τι πιστεύει ο καθένας για τον άλλο. Αν ο ένας παίκτης πιστεύει πως ο άλλος παίκτης είναι πιο γενναίος από αυτόν, τότε θα προτιμήσει να αλλάξει πρώτος πορεία. Αντίθετα αν νομίζει πως ο ίδιος είναι ο πιο γενναίος από τους δύο, τότε θα συνεχίσει να οδηγεί προς τον γκρεμό. Σε περίπτωση όμως που κάποιος από τους δύο υποτιμήσει τον αντίπαλο του θα πεθάνουν και οι δύο.

Σε αυτό το μοντέλο γίνεται η υπόθεση ότι ο κάθε παίκτης διαλέγει προκαταβολικά την στρατηγική που θα ακολουθήσει και δεν την αλλάζει (άσχετα αν πρόκειται για μη ρεαλιστικό σενάριο, αφού αν κάποιος παίκτης στρίψει ο άλλος θα συνεχίσει για να κερδίσει). Επίσης το μοντέλο βασίζεται σε μία ακόμα παραδοχή, ότι αν και οι δύο οδηγοί στρίψουν, θα είναι προς την αντίθετη κατεύθυνση.

Πανεπιστήμιο Πειραιώς

Αυτό το μοντέλο δεν έχει κυρίαρχη στρατηγική για κάποιον παίκτη.

Υπάρχουν δύο ισορροπίες Nash σε αμιγείς στρατηγικές όπως φαίνεται από τον πίνακα: οι λύσεις (A1,B2)=(3,1) και (A2,B1)=(1,3). Άρα η καλύτερη στρατηγική για κάθε παίκτη είναι να κάνει το αντίθετο του αντιπάλου του. Αν ο Α πεισθεί πως ο Β δεν θα συνεχίσει να οδηγεί, η καλύτερη λύση είναι να μην αλλάξει πορεία και το ανάποδο. Φυσικά αν και οι δύο δεν αλλάξουν πορεία και συνεχίσουν θα πεθάνουν.

Το πρόβλημα αυτό συναντάται επίσης στον κόσμο της βιολογίας σαν πρόβλημα Hawk-Dove (γεράκι-περιστέρι). Πρόκειται για μια βασική ιδέα η οποία έχει πολλές εφαρμογές στην καθημερινότητα. Δύο ζώα μάχονται για την ίδια πηγή τροφής. Κάθε ένα μπορεί να συμπεριφερθεί επιθετικά ή αμυντικά δηλαδή σαν γεράκι ή σαν περιστέρι. Αν και τα δύο επιλέξουν την επιθετική συμπεριφορά τότε κανένα δεν θα μπορέσει να πάρει το φαγητό και θα βγουν και τα δύο χαμένα (η λύση A1,B1=0,0 του παραπάνω πίνακα). Αν από την άλλη επιλέξουν την ήρεμη συμπεριφορά θα μοιραστούν το φαγητό χωρίς προβλήματα αν και το κέρδος τους δεν θα είναι το ίδιο. Το κέρδος θα είναι χαμηλότερο από το κέρδος που θα είχε αν ακολουθούσε την επιθετική συμπεριφορά (η λύση A2,B2=2,2 στον πίνακα). Κάθε ζώο προτιμά να δράσει σαν γεράκι (επιθετικά) αν υπολογίζει πως ο αντίπαλος θα δράσει σαν περιστέρι (ήπια). Αυτές είναι οι δύο ισορροπίες Nash σε αμιγείς στρατηγικές, δηλαδή το ένα ζώο να παίξει σαν γεράκι και το άλλο σαν περιστέρι. Και τα δύο θα επωφεληθούν αν κατορθώσουν να αποφύγουν την σύγκρουση (δηλαδή την ταυτόχρονη υιοθέτηση της γερακίσιας συμπεριφοράς) όχι όμως στον μέγιστο βαθμό για το καθένα ατομικά. Έτσι το κίνητρο να δράσει κανείς επιθετικά και να αποκτήσει όλο το ποσό είναι ισχυρό, γι' αυτό και οδηγούμαστε ορισμένες φορές σε σύγκρουση.

Το chicken game έχει αρκετές ομοιότητες με το battle of the sexes, αν και εδώ υπάρχει το χειρότερο σενάριο, αυτό όπου αν και οι δύο παίκτες συνεχίσουν θα χάσουν. Και στα δύο παραπάνω παίγνια ο παίκτης πρέπει να αποφασίσει ανάμεσα σε δύο σχετικά λογικές στρατηγικές που αποτελούν ισορροπία Nash. Μια διαφορά που μπορούμε να εντοπίσουμε ανάμεσα στα δύο παραπάνω παίγνια είναι ότι οι δύο ισορροπίες Nash βρίσκονται για τη μεν μάχη των φύλων διαγώνια επάνω αριστερά στην κάτω δεξιά, ενώ για το chicken game ανάποδα (κάτω αριστερά και πάνω δεξιά). Παρόλα αυτά οι δύο ισορροπίες σε κάθε παιχνίδι εντοπίζονται στις ίδιες και όχι σε αντικρουόμενες επιλογές).

Πανεπιστήμιο Πειραιώς

2.6.4 Το κλασσικό παιχνίδι κυριαρχίας κινδύνου "Risk Dominance"

Το παίγνιο που χρησιμοποιήθηκε παραπάνω στην εργασία και αναλύθηκε σε βάθος προκειμένου να γίνει καλύτερη η κατανόηση διαφόρων ορισμών της θεωρίας παιγνίων, αποτελεί ένα κλασσικό παιχνίδι κυριαρχίας κινδύνου (risk dominance). Το παράδοξο για τον μη μυημένο είναι ότι αν και φαίνεται καθαρά πως η λύση (a1, b1)=(5, 5) αποτελεί το σημείο ισορροπίας, εντούτοις η ύπαρξη μεγάλης ζημίας στο κελί (a1, b2)= (-200, 8) προκαλεί φόβο στον εκάστοτε παίκτη Α ο οποίος ενδεχομένως προτιμάει να διαλέξει την λιγότερο επικερδή στρατηγική ώστε να μην υπάρχει καμιά περίπτωση να πέσει πάνω σε αρνητικό κέρδος. Ούτε όμως τον παίκτη Β συμφέρει να ακολουθήσει την b2 στρατηγική αφού το κέρδος του θα είναι μικρότερο ανεξάρτητα από την στρατηγική που θα επιλέξει ο Α παίκτης. Κύριος σκοπός του παιγνίου αυτού είναι το να

παρατηρηθεί αν οι παίκτες θα σκεφτούν να ρισκάρουν διαλέγοντας τη σωστή στρατηγική, ή θα φοβηθούν και θα επιλέξουν με λάθος κριτήρια και θα συμβιβαστούν με τα “λίγα”.

2.6.5 Το παίγνιο “Matching Pennies”

Το τελευταίο παίγνιο που παρουσιάζουμε ονομάζεται matching pennies και μελετήθηκε για πρώτη φορά από τον von Neumann. Υπάρχουν δύο παίκτες καθένας εκ των οποίων έχει από ένα κέρμα.

Πρέπει και οι δύο ταυτόχρονα να επιλέξουν κορώνα (head) ή γράμμα (tail) γνωρίζοντας ότι αν και τα δύο νομίσματα δείχνουν το ίδιο (δείχνουν δηλαδή και τα δύο ή κορώνα ή γράμμα), ο παίκτης Α κερδίζει ένα νόμισμα από τον παίκτη Β. Αν τα νομίσματα δεν δείχνουν το ίδιο, τότε ο Β παίκτης κερδίζει και παίρνει από τον Α ένα νόμισμα. Δηλαδή ότι νόμισμα κερδίζει ο ένας παίκτης, ο άλλος το χάνει.

Ο πίνακας παρακάτω μας δίνει τη στρατηγική μορφή του παιχνιδιού.

A/B	b1 haid	b2 tail
a1 head	1,-1	-1,1
a2 tail	-1,1	1,-1

Πίνακας 2.9 Matching Pennies

Το παίγνιο αυτό αποτελεί παίγνιο μηδενικού αθροίσματος (zero sum game) αφού το κέρδος του ενός αποτελεί με τη ζημιά του άλλου. Από τον πίνακα γίνεται αμέσως αντιληπτό πως δεν υπάρχει ισορροπία σε αμιγείς στρατηγικές. Ο Α παίκτης θα προτιμήσει να παίξει γράμμα αν και ο Β παίξει γράμμα, ενώ θα προτιμήσει να παίξει κορώνα αν ο παίκτης Β παίξει κορώνα. Η μοναδική ισορροπία Nash που εντοπίζεται σε αυτό το παιχνίδι είναι σε μεικτές στρατηγικές. Οι δύο στρατηγικές του κάθε παίκτη δημιουργούνται από συνθήκες τυχαιότητας (ανάμεσα σε κορώνα ή γράμμα), δίνοντας την ίδια πιθανότητα και στις δύο και χωρίς να υπάρχει κίνητρο να δοκιμάσει κάποια άλλη στρατηγική. Επομένως η ισορροπία Nash σε μεικτές στρατηγικές είναι η (½,½),(½,½). Αν ο παίκτης Β διαλέξει είτε κορώνα είτε γράμμα με πιθανότητα ½ και για τις δύο στρατηγικές, τα αποτελέσματα για τον παίκτη Α θα είναι ½*1+½*(-1)=0 όταν θα παίξει κορώνα και ½*(-1) +½*1=0 όταν θα παίξει γράμμα. Για τον παίκτη Α επομένως είναι τελείως αδιάφορο το τι θα επιλέξει ο Β με αποτέλεσμα να παίξει και αυτός τυχαία.

Το κύριο δηλαδή χαρακτηριστικό αυτού του παιγνίου είναι ότι ο κάθε παίκτης προσπαθεί να μαντέψει την κίνηση που θα κάνει ο άλλος. Σε κάθε τέτοιου είδους παιχνίδια, που ο καθένας προσπαθεί να μαντέψει την κίνηση του άλλου, δεν υφίσταται ισορροπία Nash αφού η λύση απαραιτήτως προϋποθέτει αβεβαιότητα σχετικά με το τι θα πράξουν οι παίκτες.

Πανεπιστήμιο Πειραιώς

Αν όμως το παιχνίδι επαναληφθεί για αρκετές φορές είναι πιθανό κάποιος παίκτης να καταφέρει να ψυχολογήσει τον αντίπαλο του και να προβλέψει την κίνηση του και έτσι να δράσει με τρόπο ανάλογο με το πώς κάποιος παίζει το γνωστό παιχνίδι «Πέτρα, ψαλίδι, μολύβι, χαρτί». Παραλλαγές του μπορούμε να δούμε σε διάφορα παιχνίδια όπως στο πόκερ, στο baseball, στο σκάκι και σε πολλά άλλα παραδείγματα. Παρόμοιο είναι το παιχνίδι που παίζουν τα παιδιά και ονομάζεται «μονά ή ζυγά» (odds and evens). Δύο παίκτες φανερώνουν ταυτόχρονα ένα ή δύο δάχτυλα, και ο νικητής ανακηρύσσεται όπως παραπάνω από το αν ο αριθμός των δακτύλων είναι ίδιος ή όχι.

2.7 ΑΛΓΟΡΙΘΜΟΣ ΕΠΙΛΥΣΗΣ ΠΡΟΒΛΗΜΑΤΩΝ ΘΕΩΡΙΑΣ ΠΑΙΓΝΙΩΝ

Με βάση τα όσα έχουμε αναφέρει παραπάνω μπορούμε να καταλλήξουμε σε έναν αλγόριθμο γενικής επίλυσης προβλημάτων θεωρίας παιγνίων. Συνοψίζουμε στα εξής:

Γενικά

- Εάν υπάρχει κυρίαρχη στρατηγική για κάποιον παίκτη, τότε αυτή επιλέγεται πάντα.
- Εάν δεν υπάρχει κυρίαρχη στρατηγική τότε
    - Εάν υπάρχουν κυριαρχούμενες στρατηγικές, τότε αυτές αγνοούνται.
- Η επιλογή θα γίνει μεταξύ των μη-κυριαρχούμενων στρατηγικών.
- Πάντα υπάρχει τουλάχιστον μία μη-κυριαρχούμενη στρατηγική

Εάν δεν υπάρχει κυρίαρχη στρατηγική, τότε απαλείφουμε τις κυριαρχούμενες στρατηγικές. Η απαλοιφή κυριαρχούμενων στρατηγικών είναι πιθανό να οδηγήσει στη δημιουργία νέων κυριαρχούμενων στρατηγικών, οι οποίες ομοίως με τη σειρά τους θα απαλειφθούν και αυτές. Η διαδικασία αυτή ονομάζεται επαναλαμβανόμενη απαλοιφή κυριαρχούμενων στρατηγικών ή Iterated Elimination of Dominated Strategies, IEDS

Η μέθοδος της επαναλαμβανόμενης απαλοιφής κυριαρχούμενων στρατηγικών βασίζεται στις εξής λογικές και απαραίτητες παραδοχές:

- Κάθε παίκτης είναι λογικός και παίζει με βάση το προσωπικό του συμφέρον και άρα δεν θα επιλέξει μια κυριαρχούμενη στρατηγική.
- Κάθε παίκτης γνωρίζει ότι και οι υπόλοιποι παίκτες είναι λογικοί και επιδιώκουν και εκείνοι την μεγιστοποίηση του δικού τους συμφέροντος και άρα δεν θα προτιμήσουν να παίξουν τις δικές τους κυριαρχούμενες στρατηγικές.
- Κάθε παίκτης γνωρίζει ότι οι υπόλοιποι παίκτες γνωρίζουν ότι ο ίδιος είναι ορθολογικός.
- Κάθε παίκτης γνωρίζει ότι οι υπόλοιποι παίκτες γνωρίζουν ότι αυτός γνωρίζει ότι οι υπόλοιποι παίκτες είναι ορθολογικοί

Πανεπιστήμιο Πειραιώς

Δεν καταλήγουμε πάντα σε μία και μοναδική λύση με τη μέθοδο IEDS. Κάποια προβλήματα δεν έχουν καθόλου κυριαρχούμενες στρατηγικές (π.χ. η μάχη των φύλων). Άλλα προβλήματα έχουν μερικές μόνο κυριαρχούμενες στρατηγικές, μετά την απαλοιφή των οποίων καταλήγουν σε προβλήματα χωρίς κυριαρχούμενες στρατηγικές.

Εάν υπάρχει λύση κυρίαρχων στρατηγικών, τότε η λύση αυτή είναι η μοναδική λύση IEDS και είναι το μοναδικό σημείο ισορροπίας Nash

Εάν ένας παίκτης έχει κυρίαρχη στρατηγική, τότε κάθε λύση IEDS και κάθε σημείο ισορροπίας Nash περιλαμβάνει αυτή τη στρατηγική στη λύση για τον συγκεκριμένο παίκτη. Κάθε λύση IEDS είναι και σημείο ισορροπίας Nash όμως σημεία ισορροπίας Nash, δεν είναι απαραίτητα λύσεις IEDS.

Πανεπιστήμιο Πειραιώς

3 Σχεδιασμός εφαρμογής

Για την ανάπτυξη του σχεδίου χρησιμοποιήθηκε το εργαλείο StarUML.

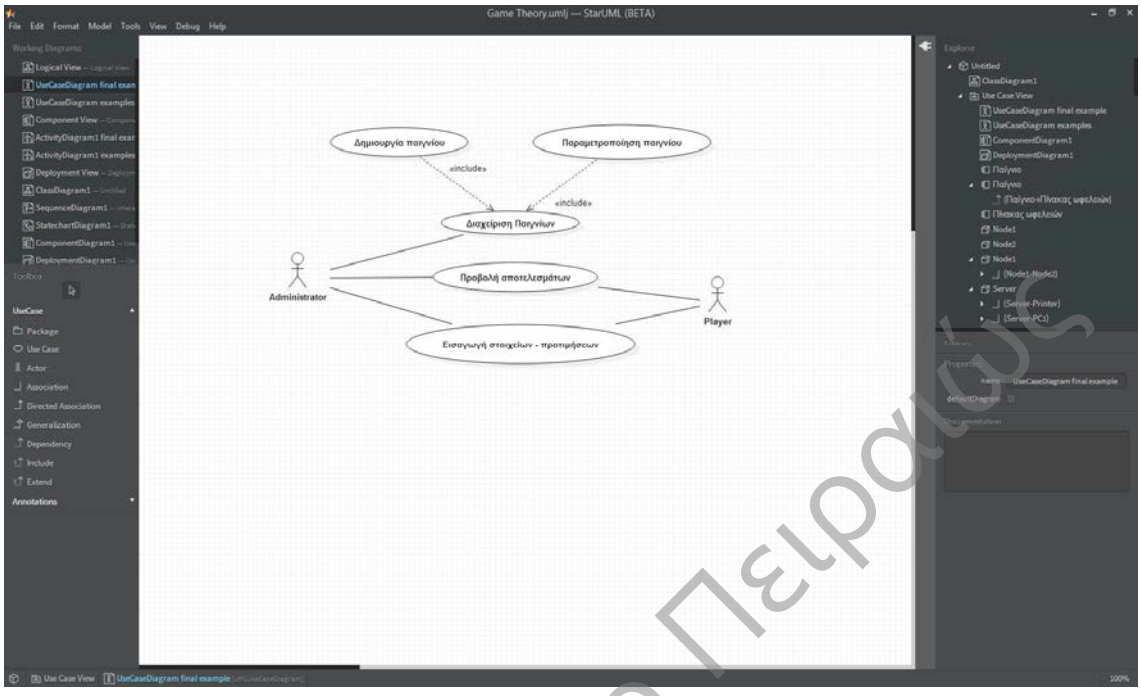
Το StarUML είναι ένα λογισμικό ανοιχτού κώδικα. Κύριο χαρακτηριστικό του είναι η γρήγορη και εύκολη ανάπτυξη μοντέλων UML τα οποία ωστόσο είναι ευέλικτα και επεκτάσιμα. Υποστηρίζει πολλές μεταβλητές που μπορούν να παραμετροποιηθούν όπως UML profile, Approach, Model Framework, NX (notation extension), MDA.

Το StarUML διαθέτει δυνατότητες επέκτασης στην αρχιτεκτονική του, έτσι ώστε ο καθένας να μπορεί να προσθέσει τις δικές του επεκτάσεις στις συμβατές γλώσσες (C++, Delphi, C#, VB, κτλ). Το StarUml ως λογισμικό ανοιχτού κώδικα έχει γραφτεί κυρίως σε Delphi.

Το StarUML χρησιμοποιήθηκε για το σχεδιασμό όλων των UML διαγραμμάτων που περιλαμβάνονται στην παρούσα εργασία. Το λογισμικό αυτό επιλέχτηκε μετά από έρευνα στα διαθέσιμα εργαλεία σχεδίασης. Οι λόγοι που υποστηρίζουν αυτή την επιλογή είναι η ευχρηστία η λειτουργικότητα και η δωρεάν διάθεση.

Το StarUML βασίζεται στην έκδοση 2.4.1 της UML και υποστηρίζει όλους τους τύπους διαγραμμάτων. Επιπλέον υποστηρίζει και ελεύθερη σχεδίαση διαγραμμάτων. Ένα ακόμα σημαντικό χαρακτηριστικό του, είναι η συγχώνευση διαγραμμάτων, κάτι το οποίο επιτρέπει την παράλληλη ανάπτυξη μοντέλων από διαφορετικές ομάδες. Τέλος, σημαντική είναι η δυνατότητα κωδικοποίησης του μοντέλου. Το StarUML υποστηρίζει τη μετατροπή ενός μοντέλου UML στις γλώσσες: Java, C++ και CORBA. Υποστηρίζεται επίσης και η αντιστροφή διαδικασία από C# σε UML.

Πανεπιστήμιο Πειραιώς



Διάγραμμα 3: Περιβάλλον χρήσης StarUML

3.1 ΔΙΑΔΙΚΑΣΙΑ ΧΡΗΣΗΣ UML

Η UML όπως αναλύθηκε και στο πρώτο κεφάλαιο παρέχει συγκεκριμένους συμβολισμούς και κανόνες στους εμπλεκόμενους στην ανάπτυξη εφαρμογών προκειμένου να σχεδιάσουν αντικειμενοστραφή μοντέλα. Παρόλα αυτά η UML δεν περιγράφει τον τρόπο εκτέλεσης αυτής της εργασίας, τη μεθοδολογία δηλαδή που θα πρέπει να χρησιμοποιηθεί για την ανάπτυξη της εφαρμογής. Η UML έχει σχεδιαστεί για να χρησιμοποιείται από πολλές διαφορετικές μεθοδολογίες ανάπτυξης εφαρμογών και αποτελεί ένα γενικό πρότυπο ανεξάρτητο από συγκεκριμένες διαδικασίες.
Ωστόσο, για να χρησιμοποιηθεί επιτυχώς η UML πρέπει να γίνει χρήση κάποιας συγκεκριμένης διαδικασίας κυρίως όταν στον σχεδιασμό και την ανάπτυξη της εφαρμογής συμμετέχει μια μεγάλη ομάδα ανθρώπων προκειμένου να υπάρξει ένας κοινός τρόπος επικοινωνίας και μια δυνατότητα μέτρησης της προόδου και πορείας της εργασίας.
Μια διαδικασία περιγράφει βασικά τα εξής: τι πρέπει να γίνει, πώς θα γίνει, πότε θα γίνει και γιατί πρέπει να γίνει με ένα συγκεκριμένο τρόπο. Περιγράφει έναν τις δραστηριότητες που πρέπει να γίνουν με καθορισμένη σειρά προκειμένου να επιτευχθεί ο συγκεκριμένος στόχος. Η ορθή ολοκλήρωση όλων των επιμέρους δραστηριοτήτων της κάθε διαδικασίας δηλώνει ότι έχει επιτευχθεί ο στόχος της ο οποίος στην προκειμένη περίπτωση είναι η ορθή ανάπτυξη της εφαρμογής.

Πανεπιστήμιο Πειραιώς

Κάθε διαδικασία ανάπτυξης μιας (αντικειμενοστραφούς) εφαρμογής με την χρήση της γλώσσας UML πρέπει να έχει συγκεκριμένα χαρακτηριστικά. Η διαδικασία καθοδηγείται από τα Use Cases (use-case-driven), επικεντρώνεται σε μια αρχιτεκτονική (architecturecentric), είναι επαναληπτική (iterative) και αυξητική (incremental). Οι όροι αυτοί αναλύονται παρακάτω.

Χαρακτηριστικά Διαδικασίας χρήσης της UML

· Καθοδήγηση από Use Cases

Τα Use Cases χρησιμοποιούνται προκειμένου να περιγραφούν οι λειτουργικές απαιτήσεις του συστήματος, τι πρέπει δηλαδή να κάνει το σύστημα. Τα Use Cases «οδηγούν» στην δημιουργία των υπολοίπων διαγραμμάτων και εγγράφων. Τα Use Cases χρησιμοποιούνται σε κάθε φάση της ανάπτυξης της εφαρμογής προκειμένου να επιβεβαιωθεί ότι οι απαιτήσεις που έχουν καταγραφεί υλοποιούνται στο σύστημα και επιπλέον για περαιτέρω έλεγχο. Το πρώτο και πολύ βασικό χαρακτηριστικό που πρέπει να προσέχουμε στην διαδικασία που περιγράφουμε είναι ο έλεγχος σε κάθε βήμα της διαδικασίας ότι υλοποιούνται σωστά και ολοκληρωμένα όλες οι απαιτήσεις που έχουν καταγραφεί με την μορφή των Use Cases.

· Επικέντρωση σε Αρχιτεκτονική

Μια διαδικασία που είναι επικεντρωμένη σε μια συγκεκριμένη αρχιτεκτονική σημαίνει ότι από την αρχή της ανάπτυξης της εφαρμογής πρέπει να ληφθεί υπόψη όλων των εμπλεκόμενων μερών μια σαφώς ορισμένη αρχιτεκτονική του συστήματος. Η UML παρέχει στον αναγνώστη των διαγραμμάτων διάφορες εναλλακτικές όψεις του συστήματος αναλόγως με τα διαγράμματα που χρησιμοποιούνται. Έτσι η βασική αρχιτεκτονική του συστήματος πρέπει να προσδιοριστεί από την αρχή της διαδικασίας και σε κάθε βήμα η αρχιτεκτονική αυτή να βελτιώνεται μέχρι την ολοκλήρωση της. Η αρχιτεκτονική αυτή αποτελεί τον βασικό χάρτη που παρουσιάζει τα μέρη του συστήματος, τις συσχετίσεις τους και τους μηχανισμούς επικοινωνίας μεταξύ τους.

Ένα είδος αρχιτεκτονικής για παράδειγμα σύμφωνα με την οποία θα σχεδιαστεί το σύστημα είναι αυτή της Αρχιτεκτονικής N-tier. Σύμφωνα με αυτή την αρχιτεκτονική η εφαρμογή χωρίζεται σε διάφορα νοητά επίπεδα κάθε ένα από αυτά εκτελεί μια συγκεκριμένη λειτουργία και είναι ανεξάρτητο από τα υπόλοιπα. Οι διάφορες όψεις της UML όπως αυτές θα παρουσιαστούν με τα εκάστοτε διαγράμματα δείχνουν στον προγραμματιστή τον τρόπο με τον οποίο έχουν υλοποιηθεί λογικά και φυσικά τα διάφορα επίπεδα της Αρχιτεκτονικής N-tier.

Επαναληπτική

Σε αντίθεση με τις ακολουθιακές παραδοσιακές μεθοδολογίες ανάπτυξης όπως αυτή του μοντέλου του Καταρράκτη, όπου το κάθε βήμα είναι αυτοτελές και προηγείται κάθε επόμενου χωρίς τη δυνατότητα περεταίρω επεξεργασίας στο μέλλον, οι διαδικασίες ανάπτυξης εφαρμογών που χρησιμοποιούν την UML είναι επαναληπτικές. Αυτό σημαίνει ότι κάθε βήμα της διαδικασίας δεν είναι αυτόνομο από τα προηγούμενα βήματα αλλά επιτρέπεται η επιστροφή σε αυτό κάποια στιγμής το μέλλον για βελτίωση κάποιων αδυναμιών, τροποποίηση κάποιων χαρακτηριστικών και διόρθωση ή ακόμα και προσθήκη κάποιων διαγραμμάτων της UML, όπου βέβαια αυτό απαιτείται. Με τον τρόπο αυτό καθ' όλη τη διάρκεια της διαδικασίας υπάρχει συνεχής επανατροφοδότηση (feedback) σε όλα τα στάδια της διαδικασίας με σκοπό την βελτίωση της διαδικασίας και κατ επέκταση της ίδιας της εφαρμογής.

Πανεπιστήμιο Πειραιώς

Η επαναληπτική διαδικασία αποτελεί ένα σύνολο από επαναλαμβανόμενα βήματα κάθε ένα από τα οποία εξαρτάται από το σε ποιο σημείο βρισκόμαστε και τι πρόοδο έχουμε επιτύχει. Για παράδειγμα, οι απαιτήσεις των χρηστών του συστήματος καταγράφονται στα πρώτα βήματα της διαδικασίας. Όσο προχωράει η διαδικασία οι αλλαγές που γίνονται είναι όλο και λιγότερες. Εάν δεν έχει γίνει σωστή καταγραφή των απαιτήσεων στην αρχή της διαδικασίας θα δημιουργηθεί μεγάλο πρόβλημα και θα πρέπει να αλλάξουν όταν ολοκληρώνεται το σύστημα. Βέβαια η επαναληπτική διαδικασία στην δημιουργία της αρχιτεκτονικής του συστήματος δεν μας απαγορεύει να κάνουμε αλλαγές στις απαιτήσεις των χρηστών αλλά άσχετα από αυτό πρέπει να λαμβάνεται και υπόψη το κόστος το οποίο πρέπει να είναι το μικρότερο δυνατό.

Αυξητική

Κάθε επανάληψη που γίνεται στην διάρκεια της ανάπτυξης της εφαρμογής έχει ως σκοπό να παράγει ένα αποτέλεσμα. Φυσικά θα πρέπει να ελεγχθεί για να διαπιστωθεί η εξέλιξη του συστήματος. Αυξητική είναι μια διαδικασία όταν σε κάθε επανάληψη φροντίζουμε να βελτιώνονται όσες αδυναμίες εντοπίστηκαν στον προηγούμενο κύκλο και έτσι παράγεται μια καινούργια έκδοση (version) του συστήματος. Βέβαια η κάθε βελτιωμένη έκδοση του συστήματος δεν σημαίνει ότι το προϊόν είναι έτοιμο προς παράδοση αλλά σημαίνει ότι το κάθε βήμα έχει ολοκληρωθεί και έχει ελεγχθεί. Εδώ εισάγεται και η έννοια των μηχανισμών ελέγχου που θα πρέπει να υπάρχουν και να τηρούνται από τους υπεύθυνους του σχεδιασμού στο τέλος του κάθε βήματος.

Συνήθως όταν αναπτύσσεται ένα αντικειμενοστραφές σύστημα αρχικά το βάρος πέφτει στην υλοποίηση των απαιτήσεων του συστήματος (τι πρέπει να κάνει το σύστημα) και στην υλοποίηση των κανόνων και της λογικής της επιχείρησης και το user interface της εφαρμογής (το Presentation layer της αρχιτεκτονικής) περνά σε δεύτερη μοίρα. Αφού γίνουν οι απαραίτητοι έλεγχοι στην ορθότητα των απαιτήσεων η διαδικασία συνεχίζεται στην βελτίωση του user interface της εφαρμογής, προσθέτοντας και την ανάπτυξη των διαφόρων οθονών και μηνυμάτων.

Έτσι κάθε βήμα της διαδικασίας χτίζει σιγά-σιγά την εφαρμογή, βελτιώνοντας και τροποποιώντας εάν κριθεί αναγκαίο τα αντίστοιχα διαγράμματα της UML όπου αυτά χρησιμοποιούνται.

Μοντέλο Διαδικασίας

Στην συνέχεια θα παρουσιάσουμε το μοντέλο μιας αντικειμενοστραφούς διαδικασίας ανάπτυξης εφαρμογών. Η διαδικασία, που παρουσιάζουμε περιλαμβάνει τα εξής βήματα:

- Συλλογή Απαιτήσεων
- Ανάλυση

Πανεπιστήμιο Πειραιώς

- Σχεδιασμός
- Ανάπτυξη
- Έλεγχος

Κάθε βήμα της διαδικασίας προσθέτει και ένα συγκεκριμένο αποτέλεσμα, το οποίο περιγράφεται από έναν αριθμό διαγραμμάτων UML. Στόχος του παραδείγματος της διαδικασίας είναι να παρουσιαστεί η χρήση της UML και πιο συγκεκριμένα ποια διαγράμματα παίρνουν μέρος σε κάθε βήμα της διαδικασίας. Η διαδικασία που θα περιγράψουμε είναι επαναληπτική και αυξητική πράγμα κάτι που σημαίνει ότι είναι εφικτή η επιστροφή σε ένα προηγούμενο βήμα προκειμένου να γίνουν οι απαραίτητες αλλαγές και διορθώσεις σε κάποια από τα διαγράμματα της UML ή σε οτιδήποτε άλλο έχει καταγραφεί. Στη συνέχεια θα παρουσιαστούν αναλυτικά τα βήματα της διαδικασίας χρήσης της UML.

Συλλογή Απαιτήσεων

Η συλλογή απαιτήσεων αποτελεί το πρώτο και ίσως πιο σημαντικό βήμα σε ολόκληρη τη διαδικασία. Είναι πολύ σημαντικό να κατανοήσουμε σε βάθος τι θέλουν οι χρήστες από το σύστημα. Η συλλογή απαιτήσεων είναι κατά κάποιον τρόπο ένα είδος «συμφωνίας» μεταξύ των τελικών χρηστών και των προγραμματιστών του συστήματος.

Όταν είναι αδύνατη η καταγραφή λεπτομερειών σχετικά με τις λειτουργίες του συστήματος τότε απλά καταγράφεται μια ιδέα και μια απλή αναφορά για το τι θα προσφέρει το σύστημα.

Το αποτέλεσμα της συλλογής των απαιτήσεων είναι το πρώτο μοντέλο της εφαρμογής, στο οποίο θα πρέπει να παρουσιάζονται:

- · **Όραμα (Vision) του συστήματος**: με τον όρο αυτό εννοούμε μια απλή αναφορά στις βασικές λειτουργίες του συστήματος. Η παρουσίαση του οράματος του συστήματος γίνεται γραπτώς, ύστερα από συζητήσεις και συνεντεύξεις με τους πελάτες του συστήματος.
- · **Απαιτήσεις του συστήματος:** με την καταγραφή των απαιτήσεων του συστήματος, εντοπίζουμε τι θέλουμε να κάνει το σύστημα. Οι απαιτήσεις του συστήματος αναπαρίστανται κατά κανόνα με τα Use Case διαγράμματα, τα οποία δείχνουν τις βασικές λειτουργίες του συστήματος, όπως τις αντιλαμβάνονται οι εξωτερικοί χρήστες (actors).
- · **Παρουσίαση Λειτουργιών:** οι βασικές λειτουργίες της εφαρμογής προκύπτουν και καταγράφονται από την ανάλυση των Use Case διαγραμμάτων και τη χρήση κάποιων use case σεναρίων. Η καταγραφή αυτή επιτυγχάνεται από την χρήση Activity διαγραμμάτων που περιγράφουν την υλοποίηση των use cases.
- · **Ανάλυση του περιβάλλοντος του πελάτη (domain model**): μέσα από την επαφή με τον πελάτη (συνεντεύξεις και συζητήσεις) αναλύεται το περιβάλλον (domain) του πελάτη προκειμένου να καταγραφούν οι βασικές οντότητες (αντικείμενα) του συστήματος. Η καταγραφή των οντοτήτων αυτών γίνεται σε ένα γενικό διάγραμμα κλάσεων χωρίς μεγάλη λεπτομέρεια αναφορικά με τις συσχετίσεις μεταξύ των αντικειμένων αυτών.

Πανεπιστήμιο Πειραιώς

Στο βήμα αυτό δεν υπάρχει απαίτηση για μεγάλη λεπτομέρεια σχετικά με το πως θα υλοποιηθούν όλα όσα καταγράφθηκαν στην συλλογή απαιτήσεων αλλά σκοπός του περισσότερο είναι το πώς παρουσιάζεται εξωτερικά σύστημα, πως δηλαδή το βλέπουν οι εξωτερικοί χρήστες χωρίς πολλές τεχνικές λεπτομέρειες.

Ανάλυση

Το επόμενο βήμα είναι αυτό της ανάλυσης. Στο βήμα αυτό τα αποτελέσματα της καταγραφής των απαιτήσεων, αναλύονται περισσότερο και βελτιώνονται. Στόχος εδώ είναι να αναλυθεί το περιβάλλον του προβλήματος ούτως ώστε να εντοπιστούν οι κύριες κλάσεις του «φυσικού - πραγματικού κόσμου». Οι κλάσεις αυτές και οι συσχετίσεις τους θα χρησιμοποιηθούν στην εφαρμογή. Συγκεκριμένα, οι λειτουργίες που εκτελούνται σε αυτό το βήμα είναι οι παρακάτω:

- · **Ανάλυση των Use Cases και των Αντικειμένων:** τα Use Case και τα αντικείμενα του προηγούμενου βήματος αναλύονται για τον εντοπισμό των κλάσεων της εφαρμογής και την πρόσθεση διαφόρων χρήσιμων λεπτομερειών σε αυτές (π.χ. χαρακτηριστικά και μέθοδοι στις κλάσεις, συσχετίσεις, κληρονομικότητα κλπ.). Η καταγραφή των αντικειμένων και των συσχετίσεων γίνεται στα Class διαγράμματα, τα οποία δείχνουν την στατική επικοινωνία των κλάσεων.
- · **Εντοπισμός νέων κλάσεων:** εκτός από την βελτίωση των ήδη υπαρχόντων κλάσεων μπορεί να ανακαλυφθούν νέες κλάσεις και να προστεθούν στο μοντέλο.
- · **Καταγραφή της επικοινωνίας μεταξύ των κλάσεων:** καταγράφεται η επικοινωνία μεταξύ των κλάσεων. Για κάποιες σημαντικές λειτουργίες της εφαρμογής καταγράφεται η επικοινωνία μεταξύ των αντικειμένων με την χρήση των αντίστοιχων Sequence και Collaboration διαγραμμάτων.
- · **Ανάλυση των καταστάσεων βασικών αντικειμένων:** με την χρήση των Statechart διαγραμμάτων παρουσιάζεται η αλλαγή των καταστάσεων των βασικών αντικειμένων της εφαρμογής.

Το αποτέλεσμα της παραπάνω ανάλυσης είναι ένα μοντέλο που περιγράφει σε ικανοποιητικό βαθμό το πρόβλημα το οποίο προσπαθεί να διαχειριστεί το σύστημα. Επίσης στο μοντέλο παρουσιάζονται όλες οι βασικές κλάσεις του συστήματος καθώς και φυσικά οι σχέσεις ανάμεσα τους.

Σχεδιασμός

Στο τρίτο βήμα του σχεδιασμού οι επιβλέποντες της ανάπτυξης της εφαρμογής χρησιμοποιούν τα αποτελέσματα της ανάλυσης προσπαθώντας να σχεδιάσουν την εφαρμογή. Οι εμπλεκόμενοι έχοντας ως σημείο αναφοράς το μοντέλο της ανάλυσης κάνουν τις απαραίτητες αλλαγές και

Πανεπιστήμιο Πειραιώς

βελτιώσεις όπου κρίνεται απαραίτητο. Πολύ συχνά τα βήματα της Ανάλυσης και του Σχεδιασμού δεν συνδυάζονται και θεωρούνται ένα ενιαίο βήμα.

Οι βασικότερες λειτουργίες που γίνονται στο βήμα αυτό παρουσιάζονται παρακάτω:

- · **Βελτίωση των κλάσεων**: οι κλάσεις της εφαρμογής βελτιώνονται ακόμα περισσότερο σύμφωνα με την λειτουργικότητα τους (επικοινωνία με τις διάφορες οθόνες της εφαρμογής, επικοινωνία με το χρήστη, με άλλα συστήματα, με την Βάση Δεδομένων κλπ).
- · **Σχεδιασμός user interface**: σχεδιάζονται οι οθόνες παρουσίασης της εφαρμογής, οι αναφορές και η επικοινωνία με άλλα συστήματα. Οι οθόνες σχεδιάζονται με γνώμονα τα δεδομένα από τα Use Case και Sequence διαγράμματα, που έχουν δημιουργηθεί στα προηγούμενα βήματα της διαδικασίας. Το αποτέλεσμα αυτού του βήματος είναι κάποια πρότυπα σχέδια του interface που παρουσιάζονται στους πελάτες αλλά και τους προγραμματιστές της εφαρμογής προκειμένου να συζητηθεί η μορφή τους έτσι ώστε να γίνουν απαραίτητες αλλαγές.
- · **Σχεδιασμός βιβλιοθηκών:** σχεδιάζονται οι απαραίτητες βιβλιοθήκες των κλάσεων και των οθονών, που θα υποστηρίξουν την εφαρμογή.
- · **Σχεδιασμός αρχιτεκτονικής:** παράγονται τα σχέδια εκείνων που θα βοηθήσουν σε τεχνικά θέματα όπως ασφάλεια, επικοινωνία με άλλες εφαρμογές και επικοινωνία με την Βάση Δεδομένων. Σχεδιάζεται η φυσική επικοινωνία των συστατικών μερών της εφαρμογής καθώς και όλα τα θέματα ασφάλειας. Τα αποτελέσματα του σχεδιασμού αρχιτεκτονικής απεικονίζονται με τα Component και τα Deployment διαγράμματα.

Ένα πολύ σημαντικό θέμα είναι αυτό της τεκμηρίωσης των αποτελεσμάτων (διαγραμμάτων, οθονών κλπ.) από τους υπεύθυνους. Τα διαγράμματα μεν που έχουν δημιουργηθεί θα αποτελέσουν την βάση για τα επόμενα βήματα της ανάπτυξης και του ελέγχου αλλά και η τεκμηρίωση δε είναι ένας εξίσου σημαντικός παράγοντας προκειμένου να υποστηριχθεί η εργασία των προγραμματιστών που θα αναλάβουν την δημιουργία του κώδικα της εφαρμογής.

Ανάπτυξη

Το επόμενο βήμα της ανάπτυξης περιλαμβάνει την συγγραφή του κώδικα της εφαρμογής. Εάν τα προηγούμενα βήματα έχουν ολοκληρωθεί σωστά η συγγραφή του κώδικα δεν θα είναι δύσκολη διαδικασία. Οι εργασίες που γίνονται σε αυτό το βήμα είναι οι εξής:

- · **Συγγραφή κώδικα:** με την χρήση των Class, των Activity και των Component διαγραμμάτων συνδυαστικά οι προγραμματιστές ξεκινούν την συγγραφή του κώδικα της εφαρμογής.
- · **Δημιουργία οθονών**: στο βήμα αυτό ολοκληρώνεται η δημιουργία της διεπαφής (user interface) της εφαρμογής. Επίσης ελέγχεται η επικοινωνία των οθονών με τις αντίστοιχες κλάσεις.
- · **Ολοκλήρωση τεκμηρίωσης**: ολοκληρώνεται η τεκμηρίωση των αποτελεσμάτων των βημάτων της διαδικασίας.

Έλεγχος

Πανεπιστήμιο Πειραιώς

Σκοπός του τελευταίου βήματος της μεθοδολογίας είναι να ανακαλυφθούν τυχόν λάθη στον κώδικα. Παίρνοντας σαν βάση ορισμένα Use Case διαγράμματα στα οποία έχει καταγραφεί η λειτουργικότητα του συστήματος, ελέγχουμε την ορθή λειτουργία του. Σε αυτό βοηθούν και τα υπόλοιπα διαγράμματα της UML τα οποία δείχνουν τις διάφορες όψεις της εφαρμογής.

Στόχος του τελευταίου βήματος είναι ο εντοπισμός και διόρθωση λαθών. Να τονίσουμε στο σημείο αυτό ότι η εύρεση λαθών δεν αποτελεί αποτυχία αλλά επιτυχία δεδομένου ότι θα καταφέρουμε να κάνουμε εγκαίρως τις απαιτούμενες διορθώσεις.

3.2 ΜΕΘΟΔΟΛΟΓΙΑ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΗΣ

Η μεθοδολογία που θα ακολουθήσουμε δείχνει τον τρόπο χρήσης της UML στην διαδικασία ανάπτυξης αντικειμενοστραφών μοντέλων. Η μεθοδολογία χρήσης της UML περιλαμβάνει τα εξής βήματα:

- Συλλογή απαιτήσεων
- Ανάλυση
- Σχεδιασμός
- Ανάπτυξη
- Έλεγχος

Σκοπός της παρουσίασης της μεθοδολογίας είναι να επισημάνουμε τον τρόπο με τον οποίο χρησιμοποιούμε τα διάφορα διαγράμματα της UML για τον σχεδιασμό, ανάπτυξη και τεκμηρίωση της εφαρμογής. Στην συνέχεια παρουσιάζονται αναλυτικά τα βήματα της μεθοδολογίας.

3.2.1 Συλλογή Απαιτήσεων

Όραμα του συστήματος

Με βάση αυτά που έχουμε αναφέρει ως τώρα σχετικά με την U.M.L. και τη θεωρία παιγνίων θα σχεδιάσουμε μία εφαρμογή η οποία θα βοηθά στην επίλυση τέτοιου είδους προβλημάτων.

Σε πρώτη φάση θα περιγράψουμε λεκτικά όλες τις λειτουργίες που θέλουμε να εκτελεί το πρόγραμμα. Στη συνέχεια θα ακολουθήσει ο επιμέρους σχεδιασμός των διαγραμμάτων

Πανεπιστήμιο Πειραιώς

δημιουργώντας όλες τις απαραίτητες όψεις της εφαρμογής, με παράλληλη επεξήγηση του κάθε βήματος μέχρι την ολοκλήρωση του σχεδιασμού της.

Βασικός σκοπός της εφαρμογής είναι η οργάνωση και όταν είναι δυνατόν η επίλυση ενός προβλήματος θεωρίας παιγνίων. Οι βασικές οντότητες που θα χρησιμοποιηθούν στην εφαρμογή είναι οι πίνακες απόδοσης, οι στρατηγικές και τα αποτελέσματα και θα παρουσιαστούν στη συνέχεια. Οι οντότητες αυτές σχετίζονται με τον κάθε χρήστη που συνδέεται στην εφαρμογή βάσει συγκεκριμένων δικαιωμάτων που έχουν οριστεί για τους ρόλους στους οποίους ανήκει ο χρήστης και αφορούν μια πληθώρα πληροφοριών που σχετίζονται με την συνολική λειτουργία της εφαρμογής.

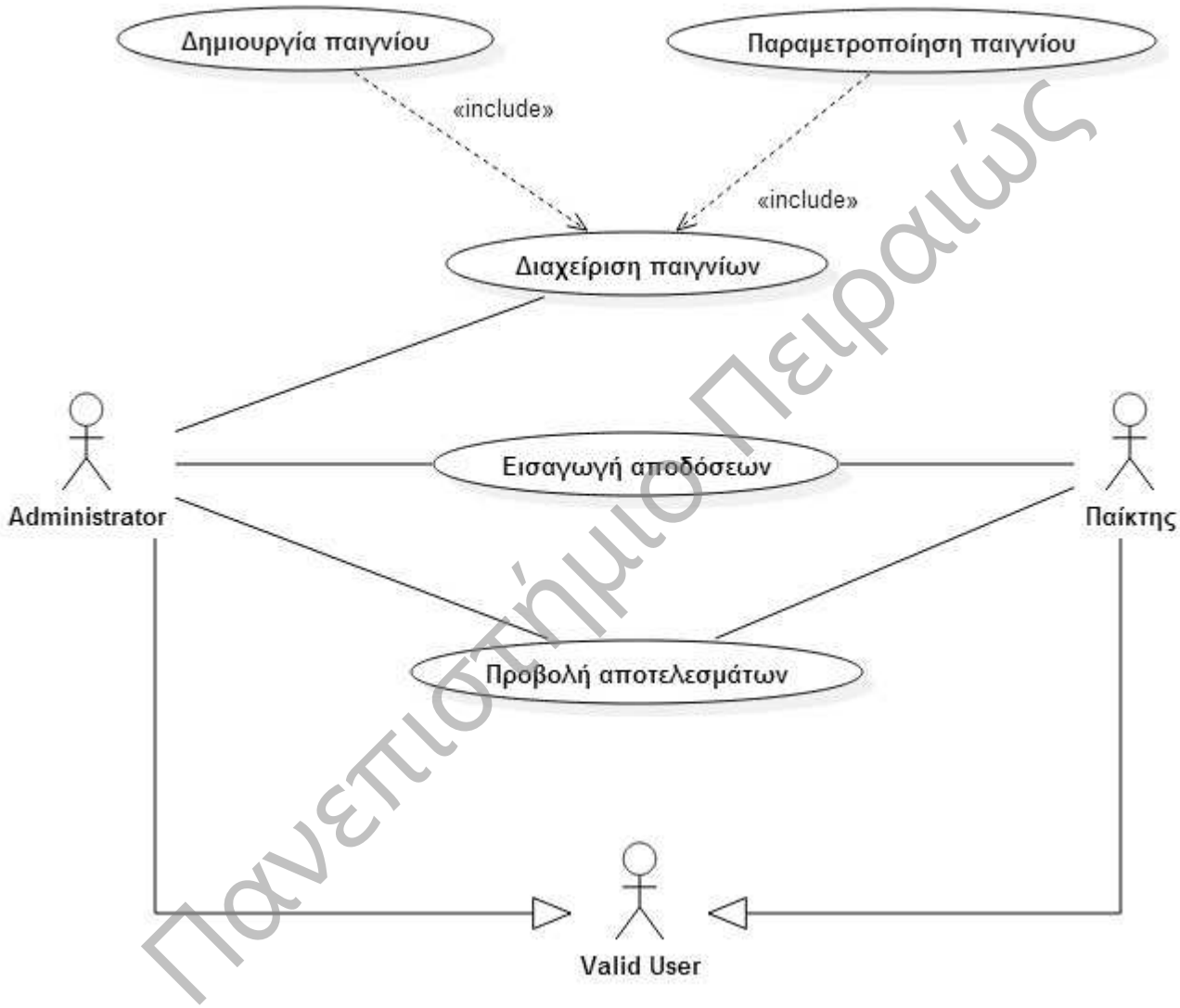
Απαιτήσεις συστήματος

Οι απαιτήσεις του συστήματος εντοπίζονται με την παρουσίαση των βασικών λειτουργιών του συστήματος όπως αυτές φαίνονται στους εξωτερικούς χρήστες του συστήματος. Η παρουσίαση των βασικών λειτουργιών του συστήματος δεν δείχνει λεπτομέρειες σχετικά με τον τρόπο υλοποίησης των λειτουργιών αυτών αλλά απλώς παρουσιάζονται όπως θα τις έβλεπε κάποιος εξωτερικός χρήστης του συστήματος.

Για την παρουσίαση των βασικών λειτουργιών του συστήματος θα χρησιμοποιήσουμε τα Use Case διαγράμματα.

Πανεπιστήμιο Πειραιώς

3.2.2 Use Case διαγράμματα



Πανεπιστήμιο Πειραιώς

Διάγραμμα 4: Main Use Case Διάγραμμα

Οι Actors δηλαδή τα πρόσωπα που αλληλεπιδρούν με το σύστημα και βρίσκονται εξωτερικά από αυτό στο περιβάλλον είναι:

- Valid User: ένας χρήστης που έχει συνδεθεί με επιτυχία στο σύστημα. Ο actor αυτός δεν εκτελεί κάποια συγκεκριμένη λειτουργία αλλά χρησιμοποιείται για να κληρονομήσουν την ιδιότητα του αυτή (σύνδεση στο σύστημα) οι υπόλοιποι actors.

- Administrator: ο διαχειριστής του συστήματος ο οποίος εκτελεί συγκεκριμένες λειτουργίες δημιουργίας διαφόρων οντοτήτων του συστήματος προκειμένου να χρησιμοποιηθούν από τους άλλους χρήστες.

- Παίκτης: ένα ειδικός χρήστης του συστήματος ο οποίος εκτελεί συγκεκριμένες λειτουργίες ενημέρωσης στοιχείων με δεδομένα.

Οι actors Administrator και Παικτης κληρονομούν την ιδιότητα του Valid User της σύνδεσης στο σύστημα δηλαδή για να εκτελέσουν τις λειτουργίες τους όλοι οι actors πρέπει πρώτα να έχουν συνδεθεί με επιτυχία στο σύστημα.

Οι βασικές λειτουργίες του συστήματος, τα βασικά use cases δηλαδή είναι:

- Διαχείριση παιγνίων
- Εισαγωγή αποδόσεων
- Προβολή αποτελεσμάτων

Οι χρήστες της εφαρμογής χωρίζονται σε δύο κατηγορίες. Ο Administrator θα είναι υπεύθυνος να «στήσει» ένα σενάριο και να ορίζει όλα τα χαρακτηριστικά του παιγνίου. Οι παίκτες αναλόγως του παιγνίου παίζουν είτε καθοριστικό ρόλο επιλέγοντας τη μία στρατηγική έναντι μιας άλλης είτε έχουν ρόλο παρατηρητή αναλόγως του παιγνίου.

Ο Administrator θα μπορεί να διενεργεί τις εξής λειτουργίες: 1. Διαχείριση παιγνίων, 2. Εισαγωγή στοιχείων – προτιμήσεων – αποδόσεων και 3. Προβολή αποτελεσμάτων.

Οι παίκτες θα μπορούν να διενεργήσουν τις εξής λειτουργίες: 1. Εισαγωγή στοιχείων – προτιμήσεων – αποδόσεων (υπό προυποθέσεις) και 2. Προβολή αποτελεσμάτων.

Από τα παραπάνω use cases υπάρχει κάποιο το οποίο τα θεωρούμε σημαντικό και πρέπει να δείξουμε περισσότερες λεπτομέρειες για να καταλάβουμε τον τρόπο λειτουργίας του.

Το use case που θεωρούμε σημαντικό είναι το «Διαχείριση Παιγνίων». Για αυτό το use case θα χρησιμοποιήσουμε επιπλέον Use Case διαγράμματα.

Παρουσίαση λειτουργιών

Αφού έχουμε παρουσιάσει τα Use Case διαγράμματα για να περιγράψουμε τις βασικές λειτουργίες του συστήματος, θα προχωρήσουμε στην ανάλυση μερικών από τις λειτουργίες αυτές. Σκοπός είναι να παρουσιάσουμε με λεκτικό τρόπο τον τρόπο λειτουργίας των use cases αυτών. Επίσης σε αυτό το βήμα δημιουργούμε κάποια use case σενάρια που περιγράφουν συγκεκριμένες λειτουργίες οι οποίες χρήζουν μεγαλύτερης ανάλυσης.

Πανεπιστήμιο Πειραιώς

Οι λειτουργίες που θα αναλύσουμε στην συνέχεια είναι οι παρακάτω:

- Διαχείριση παιγνίων
- Εισαγωγή αποδόσεων
- Προβολή αποτελεσμάτων

Use Case: Διαχείριση παιγνίων (Βασικό Σενάριο)

1. Εισαγωγή Administrator στο σύστημα
2. Δημιουργία νέου παιγνίου
   - 2.1 Όνομα παιγνίου
   - 2.2 Ορισμός αριθμού παικτών
   - 2.3 Ορισμός τύπου παιγνίου
     - 2.3.1 Ορισμός δυνατότητας συνεργασίας
     - 2.3.2 Ορισμός χαρακτηριστικών των αποδόσεων
     - 2.3.3 Ορισμός σειράς αποφάσεων
     - 2.3.4 Ορισμός αριθμού στρατηγικών
     - 2.3.5 Ορισμός παρεχόμενης πληροφόρησης
   - 2.4 Ορισμός παικτών
3. Παραμετροποίηση παιγνίου
   - 3.1 Επιλογή παιγνίου
   - 3.2 Τροποποίηση ονόματος παιγνίου
   - 3.3 Τροποποίηση αριθμού παικτών
   - 3.4 Τροποποίηση τύπου παιγνίου
     - 3.4.1 Τροποποίηση δυνατότητας συνεργασίας
     - 3.4.2 Τροποποίηση χαρακτηριστικών των αποδόσεων
     - 3.4.3 Τροποποίηση σειράς αποφάσεων
     - 3.4.4 Τροποποίηση αριθμού στρατηγικών
     - 3.4.5 Ορισμός παρεχόμενης πληροφόρησης
   - 3.5 Τροποποίηση ορισμού παικτών
4. Έξοδος από το σύστημα

Πανεπιστήμιο Πειραιώς

Το παραπάνω σενάριο δείχνει δύο διαφορετικές επιλογές που έχει ο χρήστης για να διαχειριστεί ένα παίγνιο. Μπορεί να επιλέξει είτε Δημιουργία νέου παιγνίου είτε Παραμετροποίηση αποθηκευμένου παιγνίου. Η διαδικασία της επιλογής διαχείρισης θα γίνει περισσότερο σαφής στη συνέχεια που θα παρουσιαστεί το αντίστοιχο Activity διάγραμμα του συγκεκριμένου σεναρίου.

Use Case: Εισαγωγή αποδόσεων

1. Εισαγωγή Administrator/Παίκτη στο σύστημα
2. Εισαγωγή απόδοσης για κάθε στρατηγική
3. Επαλλήθευση ορθότητας αποδόσεων (σε σχέση με τους κανόνες που έχει ορίσει ο Administrator)
4. Έξοδος απο το σύστημα

Use Case: Προβολή αποτελεσμάτων

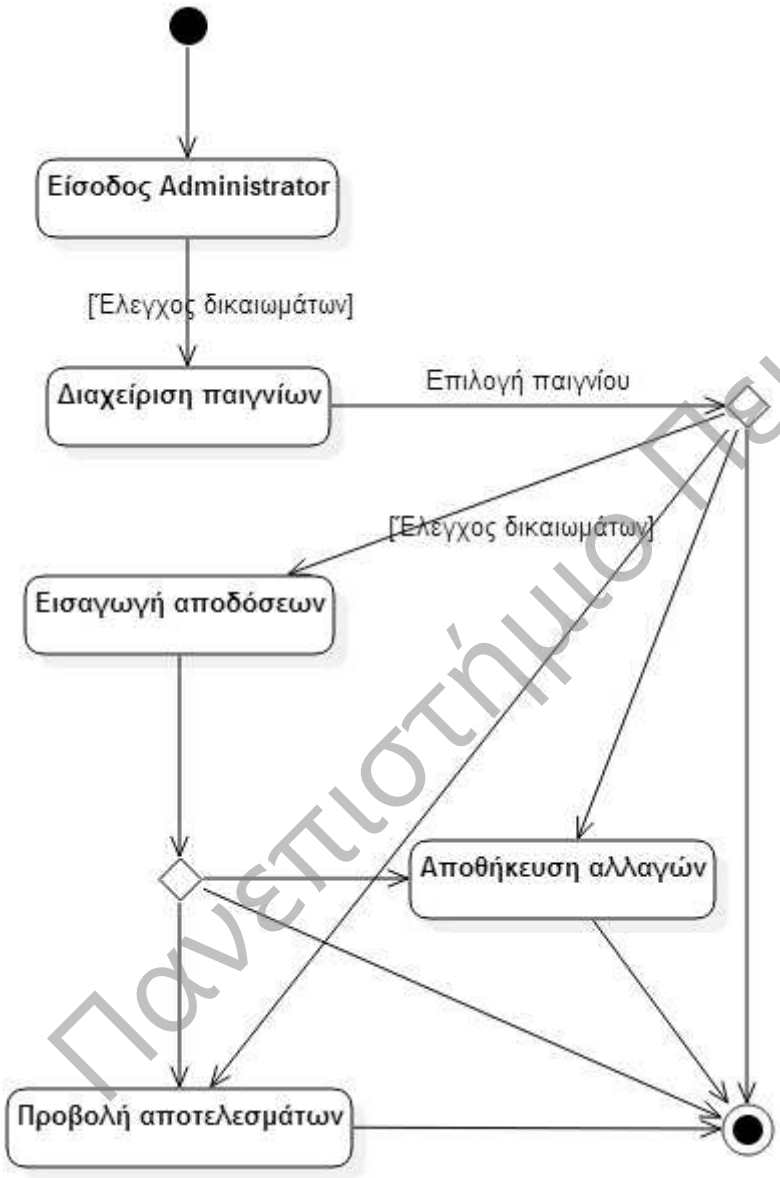
1. Εισαγωγή Administrator/Παίκτη στο σύστημα
2. Επιλογή στρατηγικής (υπό προυποθέσεις)
3. Προβολή αποτελεσμάτων παιγνίου
4. Έξοδος απο το σύστημα

Στη συνέχεια για κάθε use case σενάριο που έχουμε δημιουργήσει θα κατασκευάσουμε ένα αντίστοιχο Activity διάγραμμα. Σκοπός κάθε Activity διαγράμματος είναι η οπτικοποίηση του αντίστοιχου use case σεναρίου στο οποίο αναφέρεται.

Πανεπιστήμιο Πειραιώς

3.2.3 Activity διαγράμματα

Use case Βασικό σενάριο για Administrator

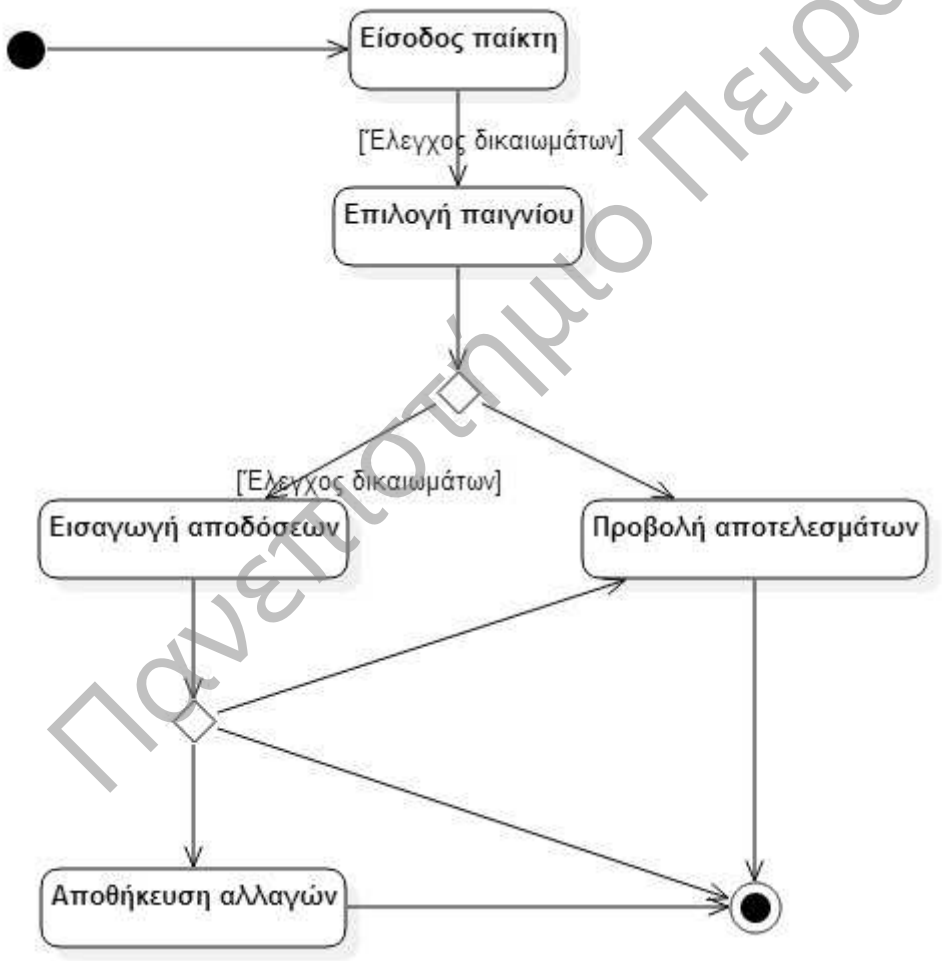


Διάγραμμα 5: Activity Διάγραμμα Administrator

Πανεπιστήμιο Πειραιώς

Στο παραπάνω διάγραμμα παρουσιάζονται οι κύριες επιλογές του Administrator. Ο Administrator αφού κάνει login στο σύστημα μπορεί να επιλέξει να διαχειριστεί ένα παίγνιο (ακολουθεί περεταίρω ανάλυση), να εισάγει αποδόσεις για ένα παίγνιο που έχει επιλέξει ή να προβάλει τα αποτελέσματα ενός παιγνίου. Όπως φαίνεται και από το διάγραμμα προκειμένου να χρησιμοποιήσει κάποιες από τις δυνατότητες που του παρέχονται θα πρέπει να προηγηθούν κάποιες επιλογές (πχ προκειμένου να δει τα αποτελέσματα ενός παιγνίου θα πρέπει πρώτα να επιλέξει ένα παίγνιο)

Use Case Βασικό σενάριο για Παίκτες



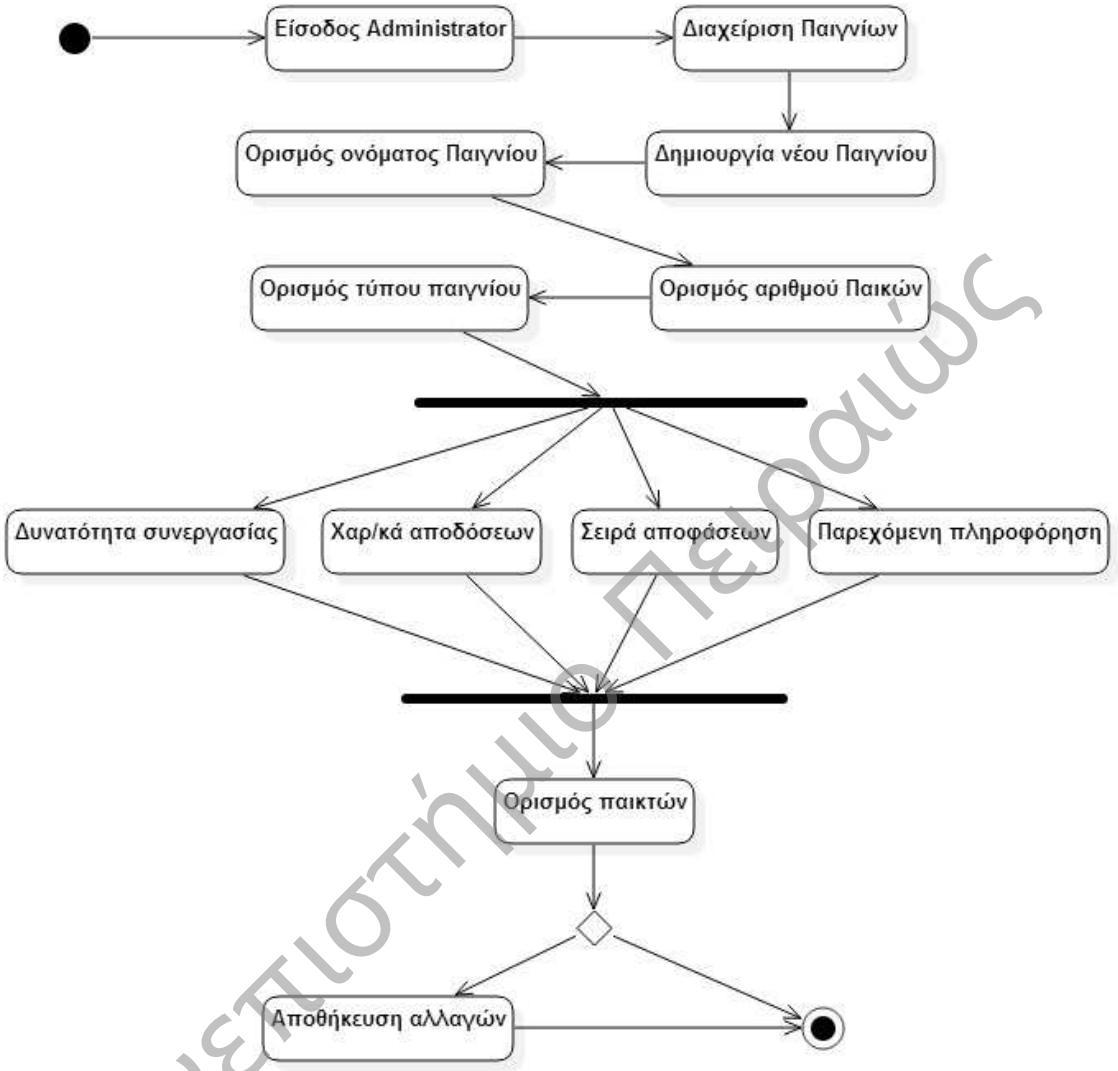
Διάγραμμα 6: Activity Διάγραμμα Παικτών

Πανεπιστήμιο Πειραιώς

Στο παραπάνω διάγραμμα παρουσιάζονται οι κύριες επιλογές των Παικτών. Ο παίκτης αφού κάνει login στο σύστημα μπορεί να επιλέξει να εισάγει αποδόσεις για ένα παίγνιο που έχει επιλέξει εφόσον έχει το δικαίωμα ή να προβάλει τα αποτελέσματα ενός παιγνίου. Όπως φαίνεται και από το διάγραμμα προκειμένου να χρησιμοποιήσει κάποιες από τις δυνατότητες που του παρέχονται θα πρέπει να προηγηθούν κάποιες επιλογές (πχ προκειμένου να δει τα αποτελέσματα ενός παιγνίου θα πρέπει πρώτα να επιλέξει ένα παίγνιο)

Use Case Σενάριο Δημιουργίας Παιγνίου για Administrator

Πανεπιστήμιο Πειραιώς

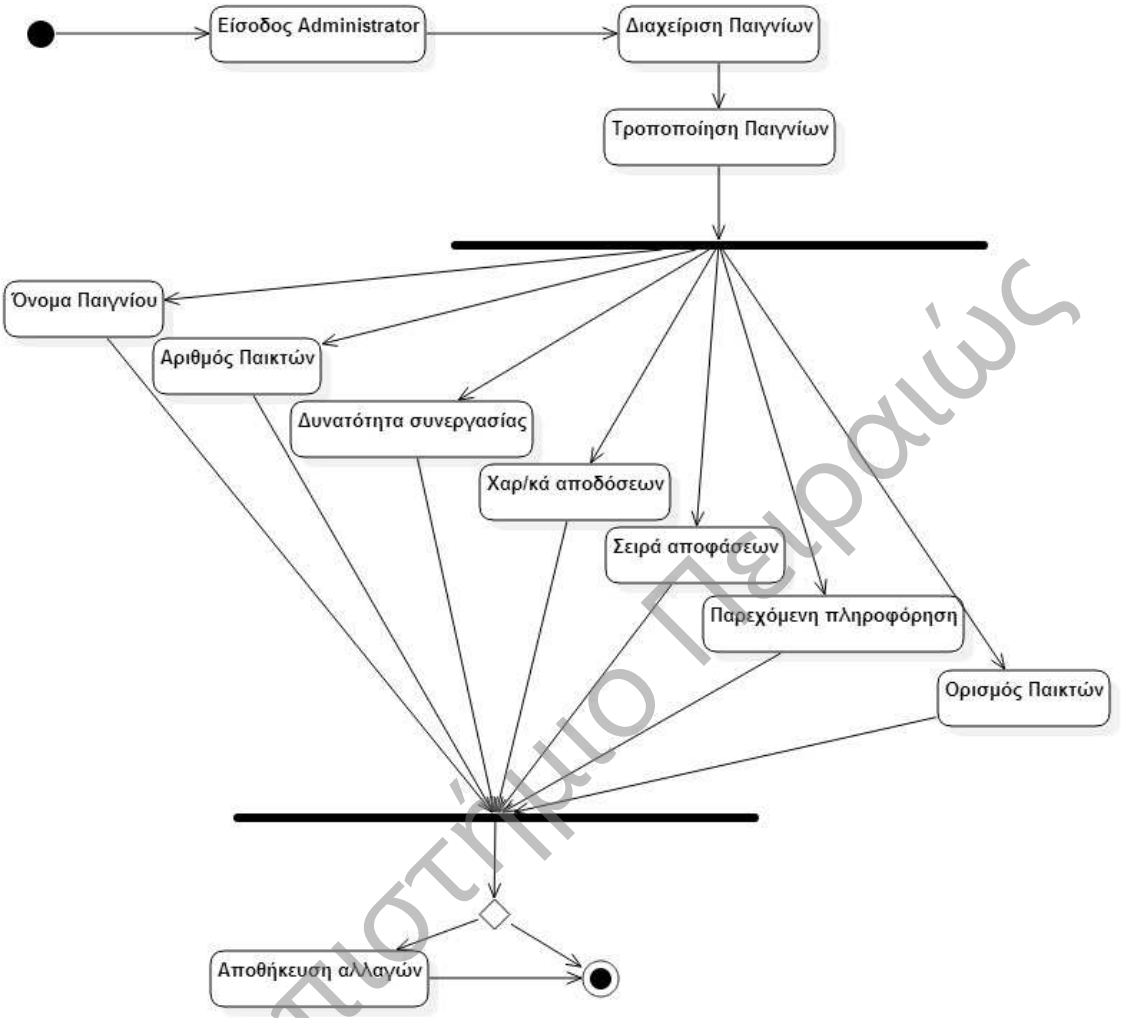


Διάγραμμα 7: Activity Διάγραμμα Παικτών

Πανεπιστήμιο Πειραιώς

Στο παραπάνω διάγραμμα παρουσιάζεται το use case Δημιουργία νέου παιγνίου που αφορά τον Administrator.
Ο Administrator αφού επιλέξει τη δημιουργία νέου παιγνίου θα πρέπει με μία σειρά επιλογών να δημιουργήσει ένα παίγνιο τα χαρακτηριστικά του οποίου θα χρησιμοποιηθούν στη συνέχεια στην εύρεση σημείου ισορροπίας. Επιπλέον σε αυτό το στάδιο ο Administrator θα ορίσει και τα δικαιώματα των παικτών πάνω στο παίγνιο.

Use Case Σενάριο Παραμετροποίησης Παιγνίου για Administrator



Διάγραμμα 8: Activity Διάγραμμα Τροποποίησης Παιγνίου

Στο παραπάνω διάγραμμα παρουσιάζεται το use case Παραμετροποίηση παιγνίου που αφορά τον Administrator.
Ο Administrator αφού επιλέξει το παίγνιο που θα παραμετροποιήσει, με μία σειρά επιλογών μπορεί να αλλάξει οποιοδήποτε από τα χαρακτηριστικά του παιγνίου.

Πανεπιστήμιο Πειραιώς

3.2.4 Ανάλυση περιβάλλοντος χρήστη (domain model)

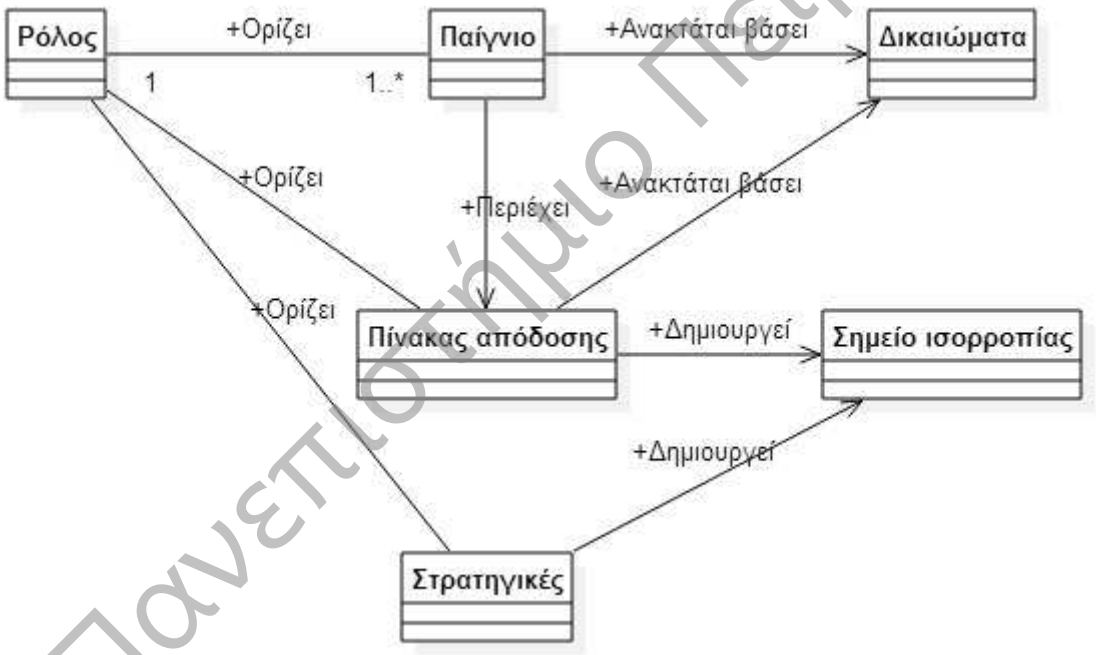
Το επόμενο βήμα σχετίζεται με την περιγραφή του περιβάλλοντος χρήστη του συστήματος (domain model). Το domain model παρουσιάζει τις βασικές οντότητες που θα χρησιμοποιηθούν στην δημιουργία της εφαρμογής και γίνεται προσπάθεια καταγραφής τους για περαιτέρω

ανάλυση τους. Μέσω της ανάλυσης των use case διαγραμμάτων, των use case σεναρίων και των αντίστοιχων activity διαγραμμάτων έχουμε επιχειρήσει να εντοπίσουμε κάποια αντικείμενα (οντότητες), που είναι πολύ πιθανό να συμμετέχουν στο σύστημα. Τα αντικείμενα αυτά είναι τα παρακάτω:

- Πίνακες αποδόσεων
- Ρόλοι
- Παίγνια
- Αποτελέσματα
- Δικαιώματα
-

Χρήση Class Διαγράμματος

Η ανάλυση της κάθε οντότητας θα παρουσιαστεί αναλυτικότερα στις επόμενες σελίδες. Η καταγραφή των βασικών οντοτήτων του συστήματος γίνεται σε ένα Class διάγραμμα όπως παρουσιάζεται στη συνέχεια



Διάγραμμα 9: Βασικό Class Διάγραμμα

Πανεπιστήμιο Πειραιώς

Παρουσίαση αντικειμένων του περιβάλλοντος της εφαρμογής
Τα βασικά αντικείμενα που έχουν εντοπιστεί μέχρι στιγμής και θα χρησιμοποιηθούν στην εφαρμογή παρουσιάζονται στον επόμενο πίνακα:

Ονομασία	**Περιγραφή**
Ρόλος	Η οντότητα του Ρόλου διαχειρίζεται τον ρόλο που έχει κάθε χρήστης που συνδέεται στην εφαρμογή. Βάσει του ρόλου ορίζεται τι έχει δικαίωμα να βλέπει ο χρήστης. Ένας ρόλος μπορεί να είναι μια συγκεκριμένη οντότητα-ρόλος, π.χ. Administrator, Παίκτης κλπ.
Δικαιώματα	Η οντότητα των Δικαιωμάτων διαχειρίζεται τα στοιχεία που έχει δικαίωμα να βλέπει ο χρήστης βάσει του Ρόλου στον οποίο ανήκει. Τα στοιχεία, που έχει δικαίωμα να βλέπει ο χρήστης είναι τα παίγνια και οι πίνακες απόδοσης.
Παίγνια	Η οντότητα των παιγνίων διαχειρίζεται τους κανόνες και τα χαρακτηριστικά που αφορούν το εκάστοτε παίγνιο.
Πίνακας απόδοσης	Η οντότητα των πινάκων απόδοσης διαχειρίζεται τις αποδόσεις που αφορούν τον κάθε παίκτη και βοηθά στον εντοπισμό κυρίαρχων στρατηγικών που θα οδηγήσουν στην επίλυση του προβλήματος και την εύρεση σημείου ισορροπίας.
Στρατηγικές	Η οντότητα των στρατηγικών διαχειρίζεται τη σειρά προτίμησης των πιθανών αποτελεσμάτων των πινάκων αποδόσεων. Προκύπτει από τους πίνακες απόδοσης στην περίπτωση όπου υπάρχουν κυρίαρχες στρατηγικές ή από τους παίκτες στην περίπτωση όπου υπάρχουν μη κυριαρχούμενες στρατηγικές.
Σημείο ισορροπίας	Η οντότητα του σημείου ισορροπίας αφορά το σημείο ή τα σημεία εκείνα του πίνακα τα οποία αποτελούν τα σημεία ισορροπίας του παιγνίου.

Πίνακας 4.1: Αντικείμενα περιβάλλοντος της εφαρμογής.

Ανάλυση
Μετά από το βήμα της Συλλογής Απαιτήσεων προχωράμε στο επόμενο βήμα της Ανάλυσης. Στο σημείο αυτό αφού έχουμε καταγράψει αναλυτικά τις απαιτήσεις των χρηστών, δηλαδή τι θέλουν να κάνει το σύστημα που σχεδιάζουμε και έχουμε καταλήξει στις βασικές οντότητες, που θα χρησιμοποιηθούν στην εφαρμογή πρέπει να προχωρήσουμε στην ανάλυση των αντικειμένων. Το βήμα αυτό είναι η «ραχοκοκαλιά» της διαδικασίας γιατί αναλύουμε όλα τα αντικείμενα που αποτελούν την δομή της εφαρμογής (μιας και μιλάμε για αντικειμενοστραφή εφαρμογή).
Έτσι θα αναλύσουμε τα αντικείμενα, θα προσπαθήσουμε να βρούμε καινούργια αντικείμενα και θα καταγράψουμε τον τρόπο, με τον οποίο τα αντικείμενα αυτά θα επικοινωνούν μεταξύ τους.

Ανάλυση Use Cases και Αντικειμένων
Μέσα από την ανάλυση των Use Case διαγραμμάτων, των use case σεναρίων και των αντίστοιχων Activity διαγραμμάτων που προηγήθηκε έχουμε εντοπίσει τα βασικά αντικείμενα που θα χρησιμοποιηθούν στην εφαρμογή. Αυτά τα αντικείμενα θέλουμε να τα ορίσουμε σε κλάσεις, δηλαδή σε εκείνες τις οντότητες του αντικειμενοστραφούς προγραμματισμού που θα χρησιμοποιηθούν για την ανάπτυξη της εφαρμογής. Επομένως για κάθε οντότητα θα δημιουργηθεί η αντίστοιχη κλάση.
Στις κλάσεις που θα δημιουργήσουμε πρέπει να προσθέσουμε κατάλληλα χαρακτηριστικά (μεταβλητές) και κατάλληλες μεθόδους (συναρτήσεις) αντίστοιχα. Τα χαρακτηριστικά και οι

Πανεπιστήμιο Πειραιώς

μέθοδοι της κάθε κλάσης ανακαλύπτονται μέσα από την λεπτομερή ανάλυση της λειτουργίας της κάθε κλάσης. Για παράδειγμα, για την κλάση που διαχειρίζεται τα Παίγνια πρέπει να χρησιμοποιήσουμε τα παρακάτω Χαρακτηριστικά, που σχετίζονται με την κλάση αυτή:

Χαρακτηριστικό	**Χρήση**
Name	Όνομα της κλάσης
Code	Κωδικός της κλάσης

Επίσης για την διαχείριση, ανάκτηση δεδομένων για μια Κλάση και την παρουσίαση της στην εφαρμογή χρειαζόμαστε τις παρακάτω Μεθόδους:

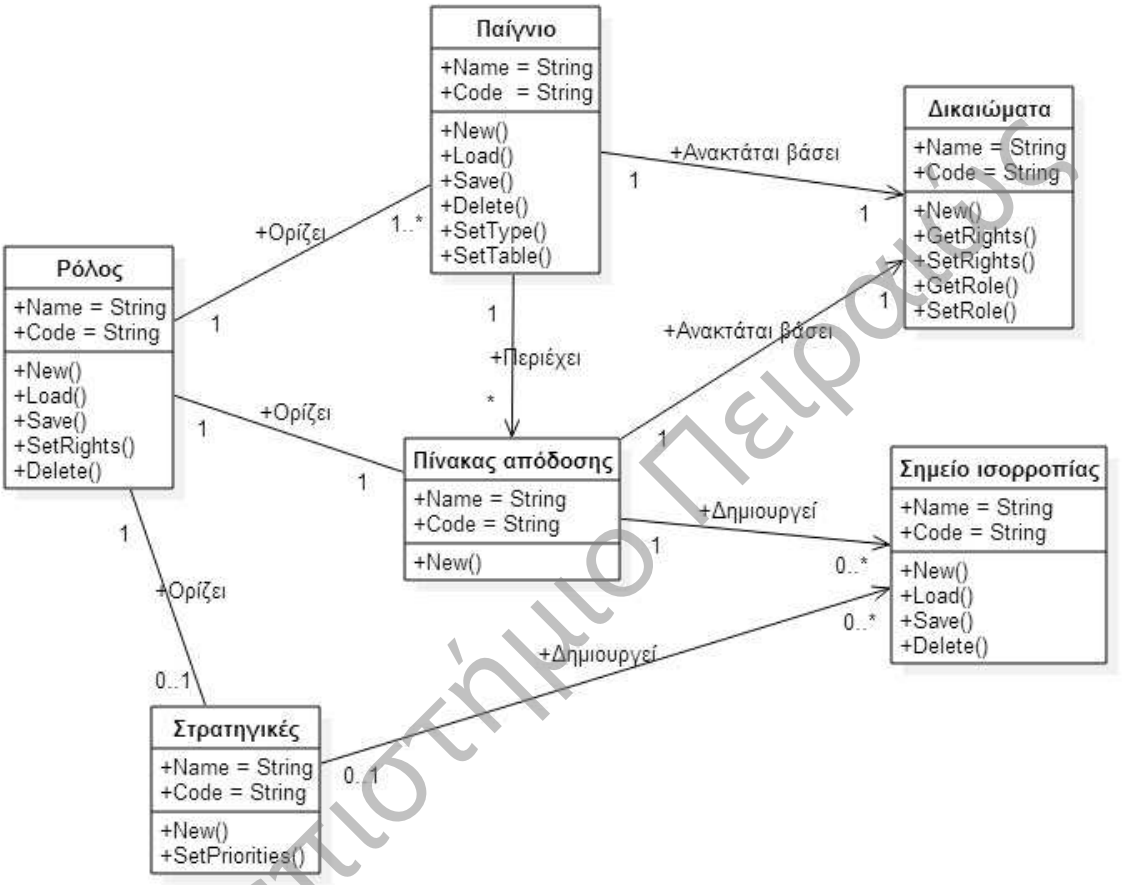
Μέθοδος	**Χρήση**
New	Δημιουργία κλάσης
Load	Φόρτωση δεδομένων
Save	Αποθήκευση δεδομένων
Delete	Διαγραφή δεδομένων
Set type	Ορισμός τύπου παιγνίου
Set table	Ορισμός πίνακα αποδόσεων

Για την γραφική αναπαράσταση της κλάσης Παίγνια θα χρησιμοποιήσουμε ένα ορθογώνιο που θα περιέχει τα Χαρακτηριστικά και τις Μεθόδους όπως φαίνεται σε διάγραμμα παρακάτω.

Την ανάλυση που κάναμε για την κλάση Παίγνια θα την επαναλάβουμε για όλες τις κλάσεις του συστήματος. Τις κλάσεις αυτές μαζί με τις συσχετίσεις τους θα τις παραστήσουμε σε ένα Class διάγραμμα:

Πανεπιστήμιο Πειραιώς

3.2.5 Class Διάγραμμα



Διάγραμμα 10: Αναλυτικό Class Διάγραμμα

Καταγραφή επικοινωνίας μεταξύ των Αντικειμένων

Αφού έχουμε εντοπίσει τα αντικείμενα, το επόμενο βήμα της διαδικασίας σχετίζεται με την καταγραφή του τρόπου επικοινωνίας μεταξύ τους. Στα προηγούμενα βήματα της διαδικασίας έχουμε παρουσιάσει τον στατικό τρόπο συσχέτισης των αντικειμένων. Η δυναμική επικοινωνία των αντικειμένων σχετίζεται με την ανταλλαγή μηνυμάτων μεταξύ τους στο πέρασμα του χρόνου.

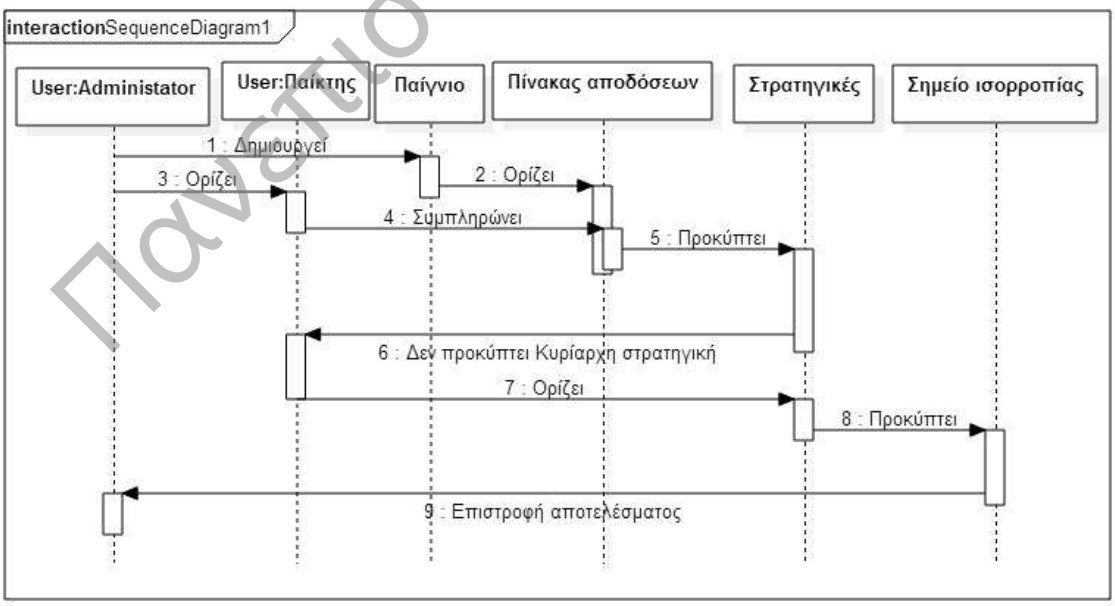
Ανταλλαγή μηνυμάτων μεταξύ των αντικειμένων ονομάζεται η κλήση μεθόδων μεταξύ των κλάσεων και αποτυπώνεται με τα Sequence διαγράμματα. Εναλλακτικά, εάν θέλουμε να παραστήσουμε την ανταλλαγή μηνυμάτων σε αύξουσα σειρά μπορούμε να χρησιμοποιήσουμε

Πανεπιστήμιο Πειραιώς

Collaboration διαγράμματα. Για λόγους ευκολίας της παρουσίασης θα χρησιμοποιήσουμε μόνο Sequence διαγράμματα.
Η κατασκευή των Sequence διαγραμμάτων βασίζεται στην ανάλυση των Use Case και των Class διαγραμμάτων καθώς και των use case σεναρίων που έχουν δημιουργηθεί στα βήματα που έχουν προηγηθεί της διαδικασίας. Στην συνέχεια παρουσιάζουμε το Sequence διάγραμμα για τη λειτουργία «Εύρεση σημείου ισορροπίας».

3.2.6 Sequence Διάγραμμα

Sequence διάγραμμα: Εύρεση σημείου ισορροπίας
Ο Administrator συνδέεται στην οθόνη διαχείρισης παιγνίων. Κατά το άνοιγμα της οθόνης φορτώνονται οι διαθέσιμες επιλογές που έχει. Ο Administrator μπορεί να επιλέξει ανάμεσα στο να δημιουργήσει ένα νέο παίγνιο ή να επιλέξει να φορτώσει ένα παίγνιο που έχει ήδη δημιουργηθεί.
Το είδος και οι προδιαγραφές του παιγνίου καθορίζουν τη μορφή του πίνακα αποδόσεων.
Ο Administrator ορίζει τους παίκτες οι οποίοι θα συμμετέχουν στο παίγνιο. Αναλόγως του σεναρίου ο εκάστοτε παίκτης μπορεί να συμπληρώσει τον πίνακα απόδοσής του μόνος του ή ο πίνακας να είναι προσυμπληρωμένος από τον Administrator.
Στη συνέχεια από τον πίνακα απόδοσης προκύπτει εάν υπάρχουν κυρίαρχες στρατηγικές προκειμένου να επιλεχθούν. Εάν όχι απορρίπτονται οι κυριαρχούμενες στρατηγικές και εάν οι μη κυριαρχούμενες στρατηγικές είναι περισσότερες από μια για τον κάθε παίκτη θα πρέπει οι παίκτες να ορίσουν τις προτιμήσεις τους ούτως ώστε να προκύψει σειρά κατάταξης των προτιμήσεών τους.
Μετά τα παραπάνω βήματα έχει προκύψει τουλάχιστον ένα σημείο ισορροπίας το οποίο επιστρέφεται στον Administrator.



Διάγραμμα 11: Sequence Διάγραμμα

Πανεπιστήμιο Πειραιώς

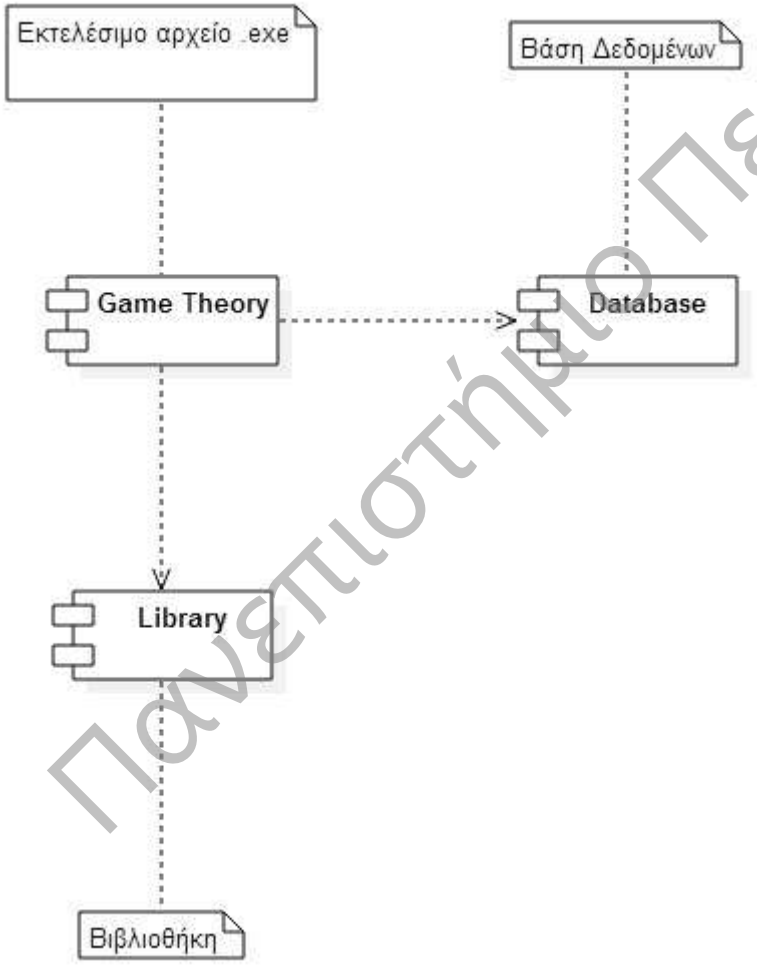
Σχεδιασμός
Μετά την ανάλυση των αντικειμένων της εφαρμογής ακολουθεί το επόμενο βήμα του Σχεδιασμού. Στο βήμα αυτό σχεδιάζουμε τα διάφορα αντικείμενα ούτως ώστε στην συνέχεια οι προγραμματιστές της εφαρμογής να μπορούν να τα υλοποιήσουν στην γλώσσα προγραμματισμού που έχουμε επιλέξει. Στο βήμα αυτό επίσης ασχολούμαστε με κάποια τεχνικά θέματα που προκύπτουν όπως ο σχεδιασμός των βιβλιοθηκών κώδικα και των οθονών που θα χρησιμοποιηθούν στην εφαρμογή καθώς επίσης και με τον σχεδιασμό της αρχιτεκτονικής βάσει της οποίας θα υλοποιηθεί η εφαρμογή. Το στάδιο αυτό στην παρούσα εργασία παραλείπεται καθώς υπερβαίνει τους σκοπούς της.

Χρήση Component και Deployment Διαγραμμάτων
Στην συνέχεια παρουσιάζουμε τα Deployment και τα Component διαγράμματα που αναπαριστούν την αρχιτεκτονική της εφαρμογής.

Πανεπιστήμιο Πειραιώς

3.2.7 Component διάγραμμα εφαρμογής

Το Component διάγραμμα χρησιμοποιείται για την περιγραφή της φυσικής υλοποίησης των τμημάτων λογισμικού του συστήματος. Όπως βλέπουμε από το παρακάτω διάγραμμα το εκτελέσιμο αρχείο της εφαρμογής μας συνδέεται με την βάση δεδομένων στην οποία αποθηκεύονται όλα τα στοιχεία (από τους λογαριασμούς των χρηστών της εφαρμογής μέχρι και τα παίγνια, οι πίνακες απόδοσης και τα αποτελέσματά τους) ενώ επιπλέον συνδέεται με την βιβλιοθήκη η οποία περιέχει όλα τα απαραίτητα αρχεία τύπου .lib προκειμένου να τρέξει η εφαρμογή

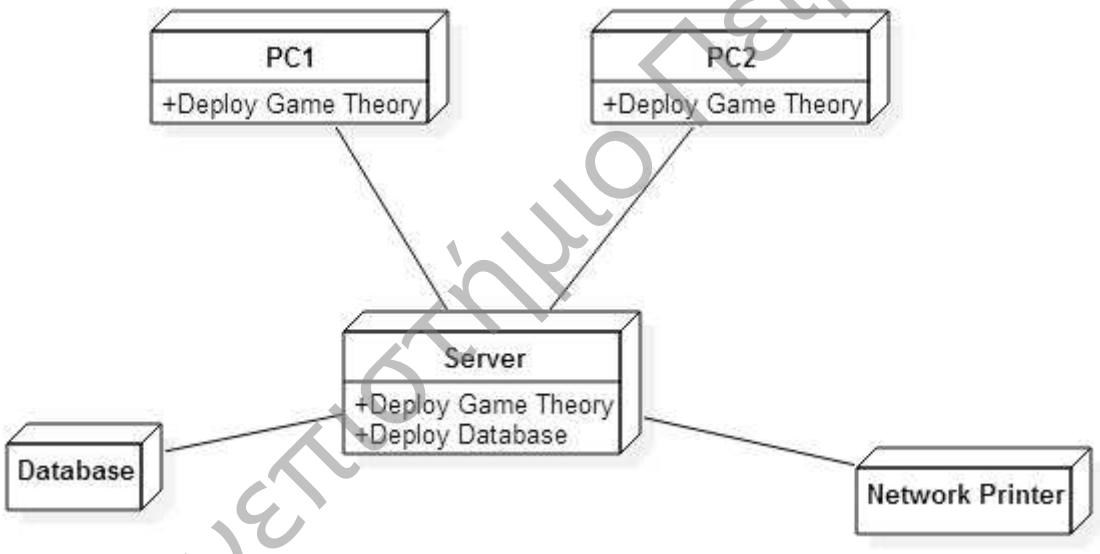


Διάγραμμα 12: Component Διάγραμμα

Πανεπιστήμιο Πειραιώς

3.2.8 Deployment διάγραμμα εφαρμογής

Τα Deployment διαγράμματα χρησιμοποιούνται για να αναπαραστήσουν την αρχιτεκτονική του hardware στο οποίο θα εγκατασταθεί το σύστημα καθώς επίσης και τον προσδιορισμό των τμημάτων του συστήματος όπου θα γίνονται οι διάφορες εργασίες, όπως για παράδειγμα υπολογιστές, servers, εκτυπωτές, αποθηκευτικοί χώροι κλπ.
Στην εφαρμογή που σχεδιάσαμε θέλουμε η εφαρμογή μας να είναι αποθηκευμένη σε έναν κεντρικό server καθώς επίσης και σε έναν αριθμό προσωπικών υπολογιστών (PC) τα οποία θα μπορούν να συνδέονται και ο κάθε χρήστης αναλόγως του ρόλου του να μπορεί να εκτελέσει τις εργασίες που επιθυμεί. Για τυπικούς λόγους αναπαράστασης ο server συνδέεται με ένα εξωτερικό σύστημα αποθήκευσης για λόγους ασφαλείας καθώς επίσης και με έναν εκτυπωτή για να είναι δυνατή η εκτύπωση όποιων στοιχείων επιθυμεί ο κάθε χρήστης (αποτελέσματα παιγνίων κλπ)



Διάγραμμα 13: Deployment Διάγραμμα

Πανεπιστήμιο Πειραιώς

3.2.9 Ανάπτυξη και Έλεγχος

Τα δυο τελευταία βήματα της αντικειμενοστραφούς διαδικασίας είναι αυτά της Ανάπτυξης και του Ελέγχου τα οποία δεν θα παρουσιάσουμε αναλυτικά καθώς ξεφεύγουν από τη θεματολογία της παρούσας εργασίας, ωστόσο κρίνεται σκόπιμο να αναφέρουμε. Στο βήμα της Ανάπτυξης οι προγραμματιστές της εφαρμογής προχωράνε στην συγγραφή κώδικα καθώς και στην δημιουργία όλων των οθονών που θα χρησιμοποιηθούν στην εφαρμογή.
Τέλος στο τελευταίο βήμα του Ελέγχου με την βοήθεια διαφόρων διαγραμμάτων της UML τα οποία παρουσιάζουν τις διάφορες όψεις της εφαρμογής προσπαθούμε να ελέγξουμε εάν οι λειτουργίες της εφαρμογής είναι σύμφωνες με τις απαιτήσεις των χρηστών. Επίσης γίνεται έλεγχος του κώδικα της εφαρμογής για τυχόν λάθη στις απαιτήσεις του συστήματος.

Πανεπιστήμιο Πειραιώς

4 Συμπεράσματα

Σκοπός της παρούσας εργασίας ήταν η μελέτη της χρήσης των ιδιοτήτων της Unified Modeling Language στην επίλυση προβλημάτων θεωρίας παιγνίων και το κατα πόσο βοηθά η εναλλακτική απεικόνιση του προβλήματος με χρήση των διαγραμμάτων της UML σε σχέση με την κλασσική απεικόνιση με πίνακες. Προσπαθήσαμε να αναφέρουμε κάποια από τα θέματα που απασχολούν τους μελετητές προβλημάτων θεωρίας παιγνίων και πιο συγκεκριμένα να επιχειρήσουμε να εντοπίσουμε το σημείο ισορροπίας ενός παιγνίου μέσω κάποιου τύπου διαγράμματος που χρησιμοποιείται στην UML. Στη συνέχεια επιχειρήσαμε να σχεδιάσουμε μέσω της UML μία εφαρμογή η οποία θα μπορούσε να επιλύει προβλήματα θεωρίας παιγνίων.
Στο Κεφάλαιο 1 παρουσιάσαμε ορισμένα εισαγωγικά θέματα σχετικά με τις γλώσσες μοντελοποίησης και πιο συγκεκριμένα προχωρήσαμε στην παρουσίαση της UML. Παρουσιάσαμε τα βασικά διαγράμματα της UML καθώς επίσης και τον τρόπο με τον οποίο χρησιμοποιούνται τα διαγράμματα αυτά για τον τελικό αποτέλεσμα που είναι η ανάλυση, σχεδιασμός και ανάπτυξη των εφαρμογών
Στο Κεφάλαιο 2 παρουσιάσαμε επιγραμματικά τις βασικές αρχές της θεωρίας παιγνίων. Παρουσιάσαμε τα βασικότερα παραδείγματα προβλημάτων προκειμένου να αντιληφθούμε τον τρόπο επίλυσης και τον τρόπο αντιμετώπισης και προσέγγισης της εύρεσης του σημείου ισορροπίας καθώς και να εντοπίσουμε τον βασικό αλγόριθμο επίλυσης τέτοιων προβλημάτων.
Τέλος στο Κεφάλαιο 3 σχεδιάσαμε μία εφαρμογή επίλυσης προβλημάτων θεωρίας παιγνίων με χρήση διαγραμμάτων UML.
Συνοπτικά μπορούμε να πούμε, ότι η χρήση διαγραμμάτων UML από μόνη της δεν βοηθά περισσότερο από τους κλασικούς πίνακες αποτελεσμάτων στην εύρεση του σημείου ισορροπίας. Η γραφική αναπαράσταση όπου κρίνεται απαραίτητο καλύπτεται πλήρως από τη χρήση δένδρων. Ωστόσο ως γλώσσα μοντελοποίησης μπορεί να χρησιμοποιηθεί με τον καλύτερο δυνατό τρόπο για τη γραφική απεικόνιση, προσδιορισμό, κατασκευή και τεκμηρίωση των στοιχείων ενός συστήματος λογισμικού.

Πανεπιστήμιο Πειραιώς

5 Βιβλιογραφία

5.1 ΞΕΝΗ ΒΙΒΛΙΟΓΡΑΦΙΑ

- Boggs Wendy, Boggs Michael (2002) “Mastering UML with Rational Rose 2002”, Sybex
- Eriksson H.,Penker M.,Lyons B.,Fado D. (OMG) (2004) “UML 2 Toolkit”, Wiley Publishing, Inc
- Fowler Martin, (2004) “UML Distilled. A brief guide to the standard Object Modeling Language”, 3rd Edition, Addison-Wesley
- Pender Tom (2002) “UML Weekend Crash Course”, Wiley Publishing, Inc
- Pender Tom (2003) “UML Bible”, Wiley
- Quatrani Terry (1999) “Visual Modeling with Rational Rose and UML”, Addison Wesley
- Quatrani Terry (2003) “Introduction to the Unified Modeling Language”, IBM, Rational Software, A technical discussion of UML
- Schmuller Joseph (2004) “Teach Yourself UML in 24 Hours”, SAMS
- Siman Si Alhir (1998) “The True Value of the Unified Modeling Language (UML)”, DistributedComputing.com
- Siman Si Alhir (2000) “Understanding Use Case Modeling”
- Object Management Group (OMG) (2003) “Unified Modeling Language Specification” έκδοση 1.5
- Booch G., Rumbaugh J. and Jacobson I., The Unified Modeling Language User Guide, Addison Wesley, Boston, MA, 1999
- Eriksson E., Penker M., UML Toolkit, John Wiley
- Booch G., Rumbaugh J., Jacobson I. (2005): Unified Modelling Language User Guide. Addison, Wesley Professional, 2 Edition
- Larman, C. (2005): Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd Edition, Prentice Hall.
- Arlow,J. Neustadt, I. (2006). UML 2 And The Unified Process: Practical Object-Oriented Analysis And Design. 2nd Edition. Addison-Wesley
- Bergstrom, S. Raberg, L. (2003). Adopting The Rational Unified Process: Success With RUP. Addison-Wesley
- Kental, S. (2001). The Unified Process Explained. Addison-Wesley
- Kroll, P. Kruchten, P. (2003). The Rational Unified Process Made Easy: A Practicioner’s Guide To The RUP. Addison-Weslay
- Larman, C (2005). Applying UML And Patterns: An Introduction To Object- Oriented Analysis And Design And The Unified Process. 3rd edition. NJ: Prentice Hall PTR
- Nisan, Roughgarden, Tardos, Vazirani. Algorithmic Game Theory, 2007
- Siegfried Tom (2006), A beautiful Math: John Nash, game theory and the Modern quest for a code of nature, Washington,D.C.,Joseph Henry Press
- Hargreaves Heap P.Shaun and Varoufakis Yanis (1995), Game theory: A critical Introduction, London, Routledge
- Rasmusen Eric (2001), Games and Information:An introduction to Game Theory,fourth edition

Πανεπιστήμιο Πειραιώς

- Fisher Len Ph.D. (2008), Rock, Paper, Scissors: Game Theory in every day life, New York, Basic Books
- Osborne J.Martin (2002), An introduction to game theory, Oxford University Press
- Osborne J.Martin and Rubinstein Ariel (1998), A Course in Game Theory, London, The MIT Press Cambridge
- Daskalakis Constantinos (2008), The Complexity of Nash Equilibria, Electrical Engineering and Computer Sciences University of California at Berkeley
- Prajit Dutta (1999) Strategies and Games , Theory and Practice
- Fowler, M. (2004): Εισαγωγή στη UML Συνοπτικός οδηγός της πρότυπης γλώσσας μοντελοποίησης αντικειμένων, 3η έκδοση, Pearson Education Inc.,
- Dixit and Nalebuff (2001), Πώς να σκέπτεστε στρατηγικά-η εφαρμογή της στρατηγικής στην πολιτική, στις επιχειρήσεις και στην καθημερινή ζωή, Αθήνα: Καστανιώτης

5.2 ΕΛΛΗΝΙΚΗ ΒΙΒΛΙΟΓΡΑΦΙΑ

- Γερογιάννης Β., Κακαρόντζας Γ., Καμέας Α., Σταμέλος Γ., Φιτσιλής Π., Αντικειμενοστρεφής Ανάπτυξη Λογισμικού με τη UML, Κλειδάριθμος 2006
- Βαρουφάκης Γιάνης, Θεωρία Παιγνίων: Η θεωρία που φιλοδοξεί να ενοποιήσει τις κοινωνικές επιστήμες,Gutenberg
- Martin J. Osborne , (2010) Εισαγωγή στην Θεωρία Παιγνίων, Κλειδάριθμος
- Κοτταρίδη και Σιουρούνης (επ) (2002), Αφιέρωμα στον John Nash: Θεωρία παιγνίων, Αθήνα: Εκδόσεις Ευρασία
- Αξελροντ, Ρ (2000), Η εξέλιξη της συνεργασίας, Αθήνα: Καστανιώτης [1984]
- Τσεμπελής, Γ. (2004), «Εμφωλευμένα Παίγνια: Η χρήση Ορθολογική Επιλογή στη Συγκριτική Πολιτική», Αθήνα: Παπαζήσης

5.3 INTERNET SITES (LINKS)

- http://el.wikipedia.org/wiki/%CE%98%CE%B5%CF%89%CF%81%CE%AF%CE%B1_%CF%80%CE%B1%CE%B9%CE%B3%CE%BD%CE%AF%CF%89%CE%BD (Άρθρο με τίτλο «Θεωρία παιγνίων»)
- http://en.wikipedia.org/wiki/Game_theory (Άρθρο με τίτλο «Game Theory»)
- http://el.wikipedia.org/wiki/Unified_Modeling_Language (Άρθρο με τίτλο «Unified Modeling Language»)
- http://en.wikipedia.org/wiki/Unified_Modeling_Language (Άρθρο με τίτλο «Unified Modeling Language»)

Πανεπιστήμιο Πειραιώς

- http://www.omg.org/spec/UML/ (Επίσημο site του οργανισμού Object Managmenet Group)
- http://www.omg.org/gettingstarted/what_is_uml.htm (Άρθρο με τίτλο «Introduction to OMG's Unified Modeling Language»)
- http://www.sparxsystems.com/uml-tutorial.html (Tutorial για τη χρήση της γλώσσας UML)
- http://staruml.sourceforge.net/en/ (Επίσημο site της εφαρμογής StarUML)
- http://levine.sscnet.ucla.edu/general/whatis.htm (Άρθρο με τίτλο «What is Game Theory»)
- http://gerasimos-politis.blogspot.gr/2011/12/nash.html (Άρθρο με τίτλο «Θεωρία παιγνίων - Ισορροπία Nash στην καθημερινή ζωή»)

5.4 NOTES

- Laurie Williams (2004) An Introduction to the Unified Modeling Language
- Βραχνός Ευριπίδης , Αθανάσιος Τζιμογιάννης (2006) Θέματα επιστημών και τεχνολογίας στην εκπαίδευση Τόμος 2, Τεύχος 3 «Εκπαιδευτικά περιβάλλοντα οπτικοποίησης αλγορίθμων»
- Σημειώσεις μαθήματος «Τεχνολογία Λογισμικού» Βίρβου Μαρία , Πανεπιστήμιο Πειραιά
- Σημειώσεις μαθήματος «Πληροφοριακά Συστήματα» Μεταξιώτης Κωνσταντίνος , Πανεπιστήμιο Πειραιά
- Σημειώσεις μαθήματος «Θεωρία Παιγνίων» Γιάννης Ρεφανίδης , Πανεπιστήμιο Μακεδονίας
- Σημειώσεις μαθήματος «Θεωρία Παιγνίων» Ν. Τσάντας , Πανεπιστήμιο Μακεδονίας
- Σημειώσεις μαθήματος «Θεωρία Παιγνίων» Εμμανουήλ Πετράκης , Πανεπιστήμιο Κρήτης
- Σημειώσεις μαθήματος «Αλγοριθμική Θεωρία Παιγνίων» Δ. Φωτάκης , Εθνικό Μετσόβιο Πολυτεχνείο
- Thomas Ferguson “Game Theory Part II Two-Person Zero-Sum Games”
- Σημειώσεις μαθήματος «Θεωρία Παιγνίων για Πολιτικούς Επιστήμονες» Άρης Αλεξόπουλος , Πανεπιστήμιο Κρήτης
- Σημειώσεις μαθήματος «Τεχνολογία Λογισμικού» Μανώλης Γιακουμάκης , Οικονομικό Πανεπιστήμιο Αθηνών
- Σημειώσεις μαθήματος «Τεχνολογία Λογισμικού» Γ. Χαραλαμπίδης , Πανεπιστήμιο Αιγαίου

5.5 ΔΙΠΛΩΜΑΤΙΚΕΣ ΕΡΓΑΣΙΕΣ

- Παπαοικονόμου Εμμανουήλ (2006) «Εφοδιασμός και Διακίνηση Προιόντων»
- Δρακάτος Στυλιανός (2012) «Το πρόβλημα της ισορροπίας Nash σε κοινοβουλευτικές συμμαχίες»
- Κοντόπουλος Μιχάλης , Νακούλας Άγγελος (2006) «Οργάνωση Γραμματείας Μεταπτυχιακών Σπουδών»

Πανεπιστήμιο Πειραιώς

- Αρετάκη Αικατερίνη (2009) «Η UML στην ανάπτυξη ενσωματωμένων συστημάτων»
- Nouri Najjar (2009) “Game Theory Portfolio, A Computational Exploration of the Nash Equilibrium”
- Βλαχοπούλου Αναστασία (2010) «Εμπειρική Προσέγγιση της Nash Ισορροπίας»
- Ψωρομήτας Χρήστος (2009) «Σχεδιασμός και Ανάπτυξη Σημασιολογικού Πληροφοριακού Συστήματος με Χρήση Οντολογίας και UML»
- Μούρκα Αναστασία (2011) «Εξαγωγή απαιτήσεων λογισμικού από εικόνες UML διαγραμμάτων σεναρίων χρήσης»
- Κουνάδη Ιωάννα (2010) «Εκπαιδευτικό λογισμικό σε UML»
- Μιχαλάκης Αντώνιος (2011) «Οι γλώσσες UML και OCL στη μοντελοποίηση των τοπολογικ’ών σχέσεων: η περίπτωση της πράξης χαρακτηρισμού έκτασης από τις δασικές υπηρεσίες.»
- Οικονόμου Όλγα (2010) «Εκπαιδευτικό υλικό πολυμέσων για το αντικείμενο της σχεδίασης και ανάπτυξης προγραμμάτων αξιοποιώντας τη γλώσσα UML»
- Παπαοικονόμου Εμμανουήλ (2006) «Αντικειμενοστραφής ανάλυση και σχεδιασμός συστήματος ελέγχου διοίκησης με την χρήση της UML»
- Νάσος Αθανασίου (2012) «Αλγοριθμική και Εξελικτική Θεωρία Παιγνίων»

5.6 MANUALS

- StarUML The Open Source UML/MDA platform (http://staruml.sourceforge.net/en/)
- Εισαγωγή στη UML «Γλώσσες Προδιαγραφής» (http://www.uml.org/)

Πανεπιστήμιο Πειραιώς