Πανεπιστήμιο Κρήτης, Τμήμα Επιστήμης Υπολογιστών
Φθινόπωρο 2006-2007

HY351:
Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων
Information Systems Analysis and Design

Εισαγωγή στη Μοντελοποίηση και στη UML

Γιάννης Τζίτζικας

Διάλεξη : 7
Ημερομηνία : 1-11-2006
Θέμα :

Διάρθρωση

- Μοντελοποίηση
- Πως προέκυψε η UML?
- Επισκόπηση των τεχνικών της UML και των χρήσεων της
- Γιατί να κάνουμε ανάλυση και σχεδίαση με τη UML?
- *Hello World!* σε UML

UML = Unified Modeling Language

Ενοποιημένη Γλώσσα Μοντελοποίησης

Τι είναι μοντελοποίηση?

Τι είναι μοντέλο και γιατί μοντελοποιούμε

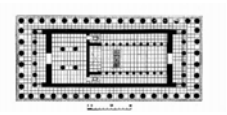
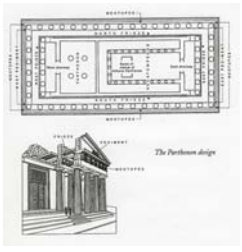
- Μοντέλο: Μια αφαίρεση (απλούστευση) της πραγματικότητας
  - εστιάζει στα σημαντικά, κρύβει τις άσχετες πλευρές και τις δευτερεύουσας σημασίας λεπτομέρειες
- Γιατί μοντελοποιούμε;
  - Ένα μοντέλο μας επιτρέπει την καλύτερη κατανόηση ενός συστήματος
  - Συνήθως φτιάχνουμε μοντέλα σύνθετων συστημάτων τα οποία δεν μπορούμε να κατανοήσουμε στην πληρότητα τους (ένεκα των περιορισμένων μας αντιληπτικών και διανοητικών ικανοτήτων)
  - Μοντελοποιώντας περιορίζουμε το πρόβλημα εστιάζοντας σε επιμέρους πλευρές του συστήματος (διαίρει και βασίλευε) και κλίμακες αφαίρεσης.

Βασικές Αρχές Μοντελοποίησης

- Η επιλογή του τύπου μοντέλου καθορίζει τον τρόπο μελέτης του συστήματος και τη μορφή της λύσης που θα επιτευχθεί.
  - Αντί αρχιτεκτονικού σχεδίου, μαθηματικές φόρμουλες πίεσης στους πυλώνες
- Κάθε μοντέλο μπορεί να παρασταθεί σε διαφορετικά επίπεδα ακρίβειας
- Καλά μοντέλα είναι εκείνα που συνδέονται με την πραγματικότητα
- Κανένα μοντέλο από μόνο του δεν είναι επαρκές. Κάθε μη τετριμμένο σύστημα προσεγγίζεται καλύτερα από ένα (σχετικά μικρό) σύνολο ανεξάρτητων μοντέλων από διαφορετικές σκοπιές.

Βασικές Αρχές Μοντελοποίησης

- Κανένα μοντέλο από μόνο του δεν είναι επαρκές. Κάθε μη τετριμμένο σύστημα προσεγγίζεται καλύτερα από ένα (σχετικά μικρό) σύνολο ανεξάρτητων μοντέλων από διαφορετικές σκοπιές.

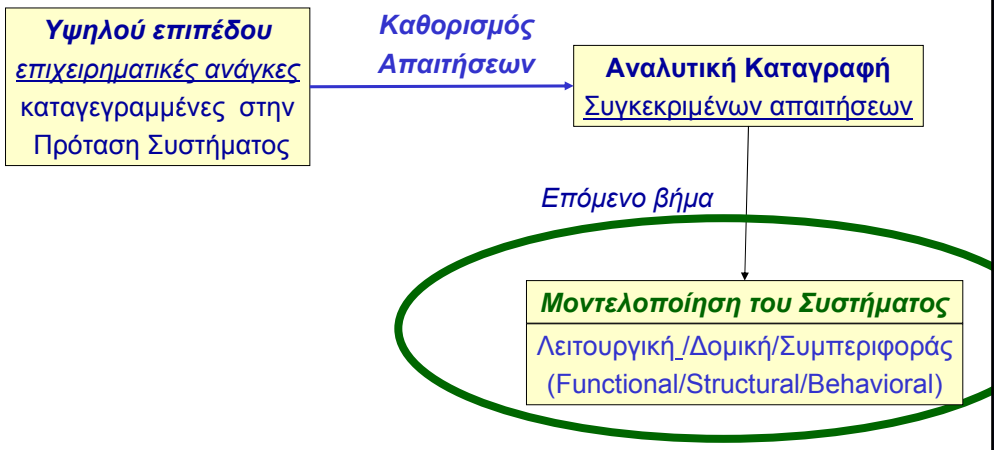




Μοντελοποίηση στην Ανάλυση και Σχεδίαση Πλ. Συστημάτων

Μοντελοποίηση στην Ανάλυση και Σχεδίαση Πλ. Συστ.:

- Βοηθά την **οπτικοποίηση** ενός (υπαρκτού ή προς κατασκευή) συστήματος
- Βοηθά την **προδιαγραφή** της δομής ή συμπεριφοράς ενός συστήματος
- Αποτελεί **οδηγό** για την κατασκευή ενός συστήματος
- **Τεκμηριώνει** τις αποφάσεις που έχουμε πάρει

Η θέση της **μοντελοποίησης** στην Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων



UML = Unified Modeling Language

Ενοποιημένη Γλώσσα Μοντελοποίησης

δηλαδή;

Εισαγωγή στη UML

- Διάδοχος των μεθόδων αντικειμενοστρεφούς ανάλυσης και σχεδίασης (object-oriented analysis and design, OOA&D) που εμφανίστηκαν στα τέλη της δεκαετίας του 80 και αρχές του 90
- Ενοποιεί τις μεθόδους του
  - Booch
  - Rumbaugh (OMT)
  - Jacobson
- Πλέον είναι OMG (Object Management Group) standard

Πως φθάσαμε στη UML;

- 1980: C++
  - Need to adapt the design methods of ('70s-'80s) for the object-oriented world
- 1989-91 "Recursive Design Approach" (Sally Shlaew, Steve Meller)
- P. Coad and Ed. Yourdon (books 1991, 1991b, 1995,1999)
- Responsibility-Driven Design (Wirfs-Brock 90)
- Class-Responsibility-Collaboration (CRC Cards) Beck and Cunnigham
- Grady Booch: work with Rational Software (for Ada systems)
- Jim Rumbaugh: Object-Modeling Technique (OMT)
- The most conceptual of these books: Martin and Odell, 94
- Ivar Jacobson (introduced the concept of Use Cases)

Γενικά δεν υπήρχε διάθεση για ενοποίηση ή τυποποίηση (standardization)

- Κάθε ένας χρησιμοποιούσε τους δικούς του συμβολισμούς και μεθοδολογία

Famous joke:
- What is the difference between a methodologist and a terrorist?
- You can negotiate with a terrorist!

Η γέννηση της UML

- Jim Rumbaugh and G. Booch => Rational Software
- 1996: The 3 amigos (James Rumbaugh, Grady Booch, Ivar Jacobson)
  - **UML Version 1.1 Became OMG standard**
- Τρέχουσα έκδοση: UML Version 2.0, 2003

Ο σκοπός της UML

Να ορίσει ένα κοινό λεξιλόγιο για τον αντικειμενοστρεφισμό και να προσφέρει διαγραμματικές τεχνικές ικανές να μοντελοποιήσουν οποιοδήποτε σύστημα από την ανάλυση έως και την υλοποίησή του.

Ο σκοπός της UML

Να ορίσει ένα κοινό λεξιλόγιο για τον αντικειμενοστρεφισμό και να προσφέρει διαγραμματικές τεχνικές ικανές να μοντελοποιήσουν οποιοδήποτε σύστημα από την ανάλυση έως και την υλοποίησή του.

Για να απαγκιστρωθούμε από την ορολογία και τις λεπτομέρειες της κάθε αντικειμενοστρεφούς γλώσσας (C++, Java, Eiffel, Smalltalk, C#, ή όποιας άλλης προκύψει στο μέλλον). Είναι χρήσιμο να έχουμε μια κοινή γλώσσα επικοινωνίας και σχεδιασμού αντικειμενοστρεφών συστημάτων.

Ο σκοπός της UML

Να ορίσει ένα κοινό λεξιλόγιο για τον αντικειμενοστρεφισμό και να προσφέρει διαγραμματικές τεχνικές ικανές να μοντελοποιήσουν οποιοδήποτε σύστημα από την ανάλυση έως και την υλοποίησή του.

Είναι πολύ βολικό να έχουμε την ίδια γλώσσα από την αρχή έως το τέλος. Θυμηθείτε τα πλεονεκτήματα των εξελικτικών μεθοδολογιών ανάπτυξης λογισμικού.

Ο σκοπός της UML

Να ορίσει ένα κοινό λεξιλόγιο για τον αντικειμενοστρεφισμό και να προσφέρει διαγραμματικές τεχνικές ικανές να μοντελοποιήσουν οποιοδήποτε σύστημα από την ανάλυση έως και την υλοποίησή του.

Τα διαγράμματα βοηθούν πολύ την επικοινωνία.

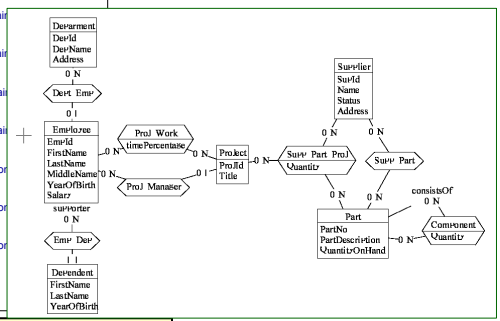
"Logicians may reason about abstractions.
But the great mass of men must have images.
The strong tendency of the multitude in all ages and nations
to idolatry can be explained on no other principle."
- Thomas Macaulay

Γιατί διαγράμματα;

```
create table Component (
    C_P_ID_Par char(10) not null,
    ID_Par char(10) not null,
    Quantity char(1) not null,
    constraint ID_Component primary key
        (C_P_ID_Par, ID_Par));
create table Deparment (
    ID_Dep char(10) not null,
    DepId char(1) not null,
    DepName char(1) not null,
    Address char(1) not null,
    constraint ID primary key (ID_Dep));
create table Dependent (
    FirstName char(1) not null,
    LastName char(1) not null,
    YearOfBirth char(1) not null,
    Supporter char(10) not null);
create table Employee (
    ID_Emp char(10) not null,
    EmpId char(1) not null,
    FirstName char(1) not null,
    LastName char(1) not null,
    MiddleName char(1) not null,
    YearOfBirth char(1) not null,
    Salary char(1) not null,
    ID_Dep char(10),
    constraint ID primary key (ID_Emp));
create table Part (
    ID_Par char(10) not null,
    PartNo char(1) not null,
    PartDescription char(1) not null,
    QuantityOnHand char(1) not null,
    constraint ID primary key (ID_Par));
```

```
create table Project (
    ID_Pro char(10) not null,
    ProjId char(1) not null,
    Title char(1) not null,
    ID_Emp char(10),
    constraint ID primary key (ID_Pro));
create table Proj_Work (
    ID_Emp char(10) not null,
    ID_Pro char(10) not null,
    timePercentage char(1) not null,
    constraint ID_Proj_Work primary key
        (ID_Pro, ID_Emp));
create table Supplier (
    ID_Sup char(10) not null,
    SupId char(1) not null,
    Name char(1) not null,
    Status char(1) not null,
    Address char(1) not null,
    constraint ID primary key (ID_Sup));
create table Supp_Part (
    ID_Par char(10) not null,
    ID_Sup char(10) not null,
    constraint ID_Supp_Part primary key
        (ID_Par, ID_Sup));
create table Supp_Part_Proj (
    ID_Par char(10) not null,
    ID_Pro char(10) not null,
    ID_Sup char(10) not null,
    Quantity char(1) not null,
    constraint ID_Supp_Part_Proj primary key
        (ID_Par, ID_Sup, ID_Pro));
```

```
-- Constraints Section
-- ---------------------

alter table Component add constraint FKconsistsOf
    foreign key (ID_Par)
    references Part;
alter table Component add constraint FKCom_Par
    foreign key (C_P_ID_Par)
    references Part;
alter table Dependent add constraint FKEmp_Dep
    foreign key (Supporter)
    references Employee;
alter table Employee add constraint FKDept_Emp
    foreign key (ID_Dep)
    references Deparment;
alter table Project add constraint FKProj_Manager
    foreign key (ID_Emp)
    references Employee;
alter table Proj_Work add constrai
    foreign key (ID_Pro)
    references Project;
alter table Proj_Work add constrai
    foreign key (ID_Emp)
    references Employee;
alter table Supp_Part add constrai
    foreign key (ID_Sup)
    references Supplier;
alter table Supp_Part add constrai
    foreign key (ID_Par)
    references Part;
alter table Supp_Part_Proj add cor
    foreign key (ID_Sup)
    references Supplier;
alter table Supp_Part_Proj add cor
    foreign key (ID_Pro)
    references Project;
alter table Supp_Part_Proj add cor
    foreign key (ID_Par)
    references Part;
```



Το ίδιο σχήμα εκφρασμένο σε SQL και στη μορφή διαγράμματος ER

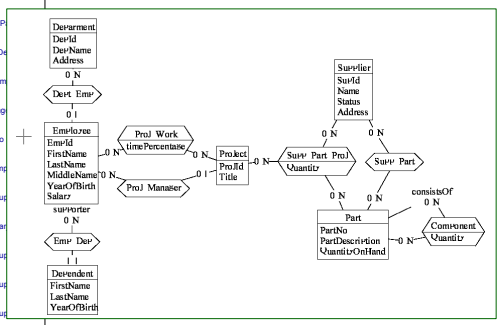
Μια εικόνα αξίζει όσο χίλες λέξεις

Γιατί διαγράμματα;


```
create table Component (
    C_P_ID_Par char(10) not null,
    ID_Par char(10) not null,
    Quantity char(1) not null,
    constraint ID_Component primary key (C_P_ID_Par,
        ID_Par));
create table Deparment (
    ID_Dep char(10) not null,
    DepId char(1) not null,
    DepName char(1) not null,
    Address char(1) not null,
    constraint ID primary key (ID_Dep));
create table Dependent (
    FirstName char(1) not null,
    LastName char(1) not null,
    YearOfBirth char(1) not null,
    Supporter char(10) not null);
create table Employee (
    ID_Emp char(10) not null,
    EmpId char(1) not null,
    FirstName char(1) not null,
    LastName char(1) not null,
    MiddleName char(1) not null,
    YearOfBirth char(1) not null,
    Salary char(1) not null,
    ID_Dep char(10),
    constraint ID primary key (ID_Emp));
create table Part (
    ID_Par char(10) not null,
    PartNo char(1) not null,
    PartDescription char(1) not null,
    QuantityOnHand char(1) not null,
    constraint ID primary key (ID_Par));
```

```
create table Project (
    ID_Pro char(10) not null,
    ProjId char(1) not null,
    Title char(1) not null,
    ID_Emp char(10),
    constraint ID primary key (ID_Pro));
create table Proj_Work (
    ID_Emp char(10) not null,
    ID_Pro char(10) not null,
    timePercentage char(1) not null,
    constraint ID_Proj_Work primary key (ID_Pro, ID_Emp));
create table Supplier (
    ID_Sup char(10) not null,
    SupId char(1) not null,
    Name char(1) not null,
    Status char(1) not null,
    Address char(1) not null,
    constraint ID primary key (ID_Sup));
create table Supp_Part (
    ID_Par char(10) not null,
    ID_Sup char(10) not null,
    constraint ID_Supp_Part primary key (ID_Par, ID_Sup));
create table Supp_Part_Proj (
    ID_Par char(10) not null,
    ID_Pro char(10) not null,
    ID_Sup char(10) not null,
    Quantity char(1) not null,
    constraint ID_Supp_Part_Proj primary key (ID_Par, ID_Sup, ID_Pro));
```

```
-- Constraints Section
-- ---------------------

alter table Component add constraint FKconsistsOf
    foreign key (ID_Par)
    references Part;
alter table Component add constraint FKCom_P
    foreign key (C_P_ID_Par)
    references Part;
alter table Dependent add constraint FKEmp_De
    foreign key (Supporter)
    references Employee;
alter table Employee add constraint FKDept_Em
    foreign key (ID_Dep)
    references Deparment;
alter table Project add constraint FKProj_Manag
    foreign key (ID_Emp)
    references Employee;
alter table Proj_Work add constraint FKPro_Pro
    foreign key (ID_Pro)
    references Project;
alter table Proj_Work add constraint FKPro_Emp
    foreign key (ID_Emp)
    references Employee;
alter table Supp_Part add constraint FKSup_Sup
    foreign key (ID_Sup)
    references Supplier;
alter table Supp_Part add constraint FKSup_Par
    foreign key (ID_Par)
    references Part;
alter table Supp_Part_Proj add constraint FKSup
    foreign key (ID_Sup)
    references Supplier;
alter table Supp_Part_Proj add constraint FKSup
    foreign key (ID_Pro)
    references Project;
alter table Supp_Part_Proj add constraint FKSup
    foreign key (ID_Par)
    references Part;
```

- ποιο μπορεί να κατανοηθεί γρηγορότερα;
- σε ποιο μπορούν να γίνουν αλλαγές πιο γρήγορα;
- με ποιο θα μπορούσαν (>2) άνθρωποι να συμφωνήσουν πιο γρήγορα και πιο εύκολα;
- ποιο μπορεί να κατανοηθεί πιο εύκολα από κάποιον που δεν ξέρει πολλά από πληροφορική;
- ποιο έχει μεγαλύτερη πιθανότητα να είναι κατανοήσιμο και μετά από 10 χρόνια;

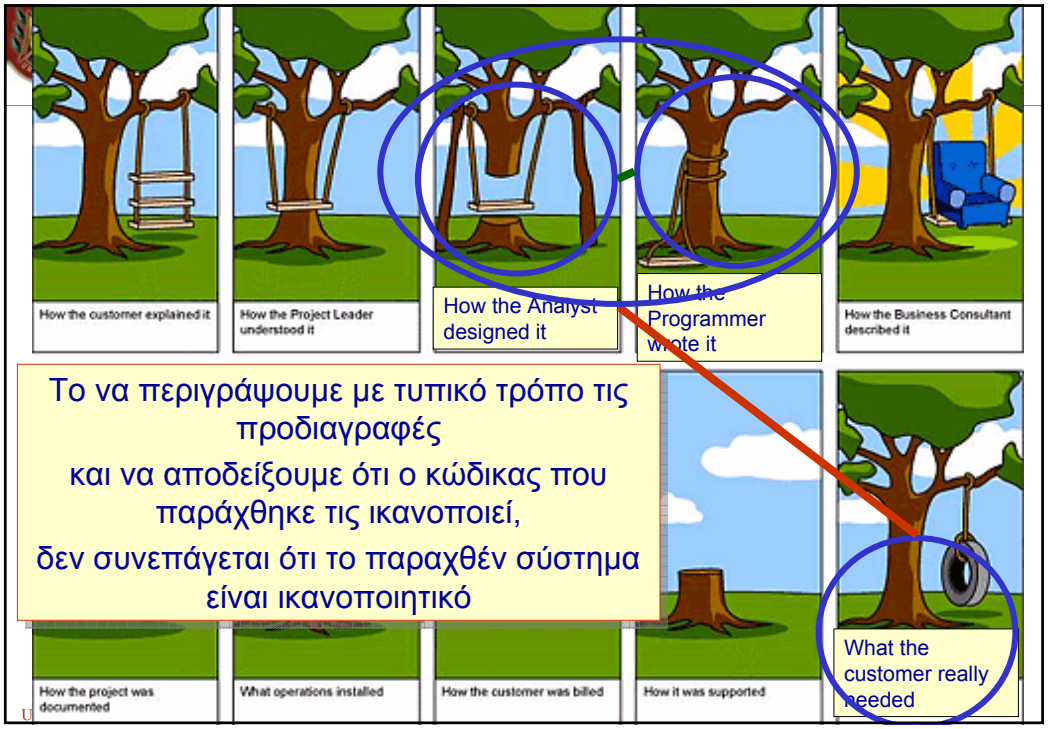
Γιατί όχι φυσική γλώσσα;

Οι περιγραφές σε φυσική γλώσσα συχνά υποφέρουν από ασάφεια.

Η περιγραφή σύνθετων εννοιών δεν είναι ούτε εύκολη ούτε ευέλικτη.

Γιατί όχι τυπικές μεθόδους (formal methods);

Ακόμα και αν αποδείξουμε ότι ένα πρόγραμμα ικανοποιεί μια μαθηματική προδιαγραφή, δεν μπορούμε να αποδείξουμε ότι η μαθηματική προδιαγραφή αντικατοπτρίζει τις πραγματικές απαιτήσεις του συστήματος.



Το να περιγράψουμε με τυπικό τρόπο τις προδιαγραφές

και να αποδείξουμε ότι ο κώδικας που παράχθηκε τις ικανοποιεί,

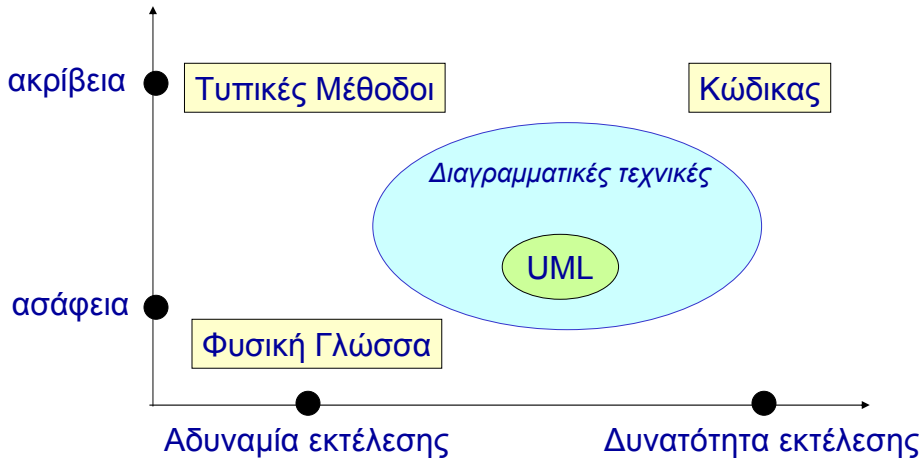
δεν συνεπάγεται ότι το παραχθέν σύστημα είναι ικανοποιητικό

Γιατί όχι τυπικές μεθόδους (formal methods);

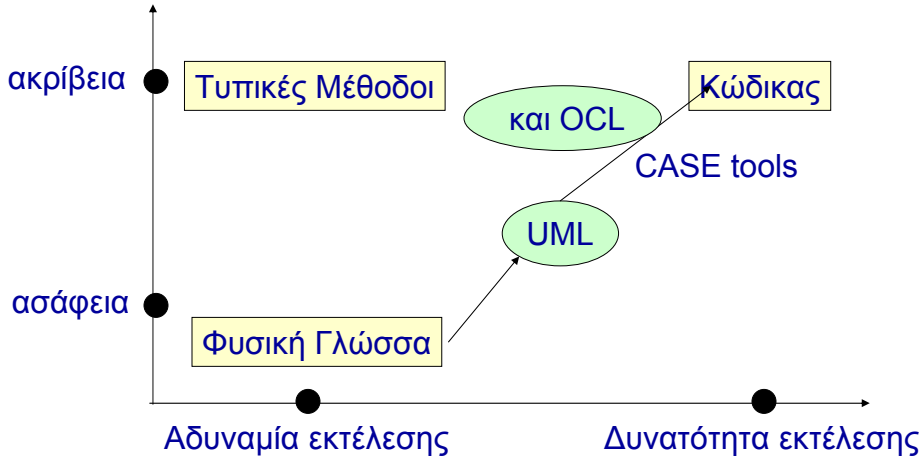
Άλλα προβλήματα των τυπικών μεθόδων:

- Συχνά η ουσία «χάνετε» μέσα σε ήσσονος σημασίας (για το πρόβλημα που μελετάμε) λεπτομέρειες.
- Οι περιγραφές που προκύπτουν κατανοούνται δύσκολα και η διαχείριση τους είναι εξίσου δύσκολη.
  - Η σύνταξη τους είναι συχνά δυσκολότερη απ' ότι ο προγραμματισμός στις γλώσσες προγραμματισμού. **Και δεν είναι ουτε καν εκτελέσιμες**

Η Γενική Εικόνα



Η Γενική Εικόνα



Γιατί να κάνουμε Ανάλυση και Σχεδίαση με τη UML?

- Η ανάπτυξη λογισμικού έχει ως στόχο την παραγωγή εκτελέσιμου κώδικα
  - Κάποιος θα μπορούσε να πει: «τα διαγράμματα είναι .. απλώς διαγράμματα, τίποτα παραπάνω από όμορφες εικόνες»
- Άρα πρέπει να αναρωτηθούμε
  - Γιατί να χρησιμοποιήσουμε UML?
  - Πως θα μας βοηθήσει όταν θα πρέπει να γράψουμε κώδικα ?
- Οι 3 κυριότεροι λόγοι
  - [A] Επικοινωνία
  - [B] Εκμάθηση Αντικειμενοστρεφισμού
  - [C] Επικοινωνία με τους ειδικούς του πεδίου εφαρμογής

Γιατί να κάνουμε Ανάλυση και Σχεδίαση με τη UML? [A] Επικοινωνία

Θεμελιώδης λόγος για χρήση της UML

- Η καλή επικοινωνία μεταξύ των εμπλεκομένων σε ένα έργο αποτελεί καθοριστικό παράγοντα για την έκβαση του.
- Η UML υπερτερεί στο θέμα αυτό από άλλες εναλλακτικές
  - Φυσική Γλώσσα (ασάφειες), Κώδικας (υπερβολικά λεπτομερής, δεν προσφέρει εποπτική εικόνα), Τυπικές Μέθοδοι (δυσχρηστες)
- Υπό αυτήν την έννοια μπορούμε να πούμε ότι χρησιμοποιούμε UML όταν ναι μεν θέλουμε να είμαστε ακριβείς αλλά δεν θέλουμε να χαθούμε στις λεπτομέρειες. Αυτό δεν σημαίνει αποφυγή των λεπτομερειών. Απλά μπορούμε να τονίσουμε τις σημαντικές λεπτομέρειες.
- Επίσης επιτρέπει την επαναληπτική/αυξητική διαδικασία (η ίδια γλώσσα μπορεί να χρησιμοποιηθεί τόσο στο αρχικό και αφηρημένο επίπεδό όσο και στο λεπτομερέστατο επίπεδο του κώδικα.

Γιατί να κάνουμε Ανάλυση και Σχεδίαση με τη UML?
[A] Επικοινωνία: Παραδείγματα

Παραδείγματα

- Εργάζεστε ως σύμβουλος πληροφορικής και θέλετε σε πολύ μικρό χρονικό διάστημα να κατανοήσετε ένα μεγάλο έργο
  - Η UML σας δίνει την συνολική εικόνα του συστήματος
  - Τα διαγράμματα κλάσεων σας λένε τι είδους αφαιρέσεις (abstractions) έχουν γίνει και μπορείτε γρήγορα να εντοπίσετε τα τμήματα που έχουν προβλήματα ή απαιτούν επιπλέον εργασία/βελτίωση.
  - Αν θέλετε μια βαθύτερη εικόνα για το πώς οι κλάσεις συνεργάζονται, τότε μπορείτε να δείτε τα σχετικά διαγράμματα αλληλεπίδρασης (interaction diagrams)
- Εργάζεστε σε έναν οργανισμό ως αναλυτής/σχεδιαστής συστημάτων. Μπορείτε να εκφράσετε την ανάλυση και τη σχεδίαση σε UML και μια άλλη εταιρία να αναλάβει την υλοποίηση του συστήματος.
- Εργάζεστε σε ένα έργο το οποίο πρέπει να βασιστεί σε υπάρχοντα εξαρτήματα (components). Με τη UML μπορείτε να τα κατανοήσετε γρήγορα και εν συνεχεία να εκφράσετε την ανάλυση και τη σχεδίαση του συστήματος λαμβάνοντας υπόψη τη λειτουργικότητα που αυτά παρέχουν.

Γιατί να κάνουμε Ανάλυση και Σχεδίαση με τη UML?
[A] Επικοινωνία: Παραδείγματα

Για τους ίδιους λόγους η UML είναι χρήσιμη και στα πλαίσια μιας ομάδας ανάπτυξης λογισμικού:

- Τα μέλη της ομάδας έχουν μια κοινή εικόνα (σημείο αναφοράς)
- Τα νέα μέλη της ομάδας μπαίνουν γρήγορα στο παιχνίδι
- Μειώνεται το ρίσκο από τις πιθανές αποχωρήσεις μελών από την ομάδα

Γιατί να κάνουμε Ανάλυση και Σχεδίαση με τη UML?

[Β] Εκμάθηση αντικειμενοστρεφισμού (Learning OO)

Η εκμάθηση και καλή χρήση του απαιτεί χρόνο

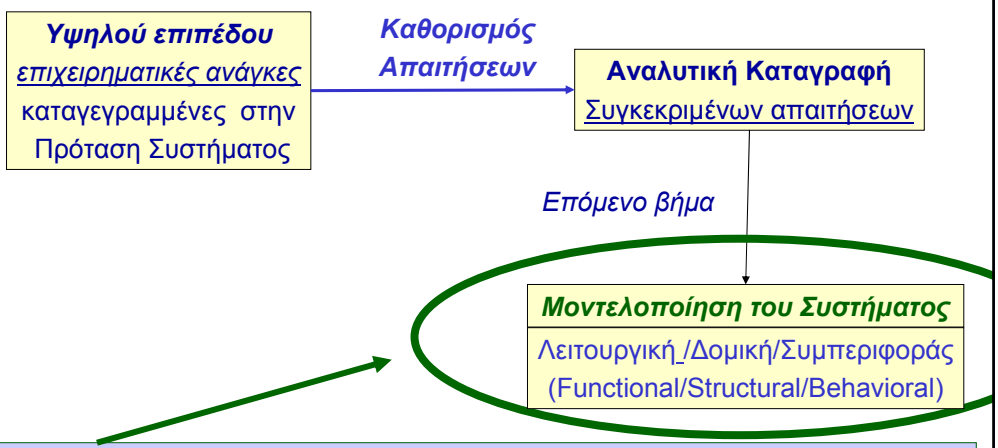
- Διαγράμματα Αλληλεπίδρασης (Interaction diagrams)
  - Κάνουν εμφανή τον τρόπο ανταλλαγής μηνυμάτων και άρα είναι χρήσιμα για τον εντοπισμό των υπερβολικά κεντρικοποιημένων σχεδίων
- Πρότυπα (Patterns):
  - Αποτελούν καλά παραδείγματα σχεδιασμού και προγραμματισμού. Η έκφραση τους σε UML βοηθάει πολύ την κατανόηση και την εκμάθησή τους
- CRC cards
  - αποτελούν χρήσιμη τεχνική για την εκμάθηση του αντικειμενοστρεφικού (δεν είναι τυπικά τμήμα της UML)
- Διαγράμματα Κλάσεων
  - Παρόμοια με τα μοντέλα δεδομένων
  - Κίνδυνος: ανάπτυξη ενός μοντέλου κλάσεων που είναι δεδομενο-κεντρικό (data orinted) αντί για υπευθυνο-κεντρικό (responsibility oriented)

Γιατί να κάνουμε Ανάλυση και Σχεδίαση με τη UML?

[Γ] Επικοινωνία με τους ειδικούς του πεδίου εφαρμογής

- Περιπτώσεις Χρήσης (Use Cases):
  - Περίπτωση Χρήσης: Ένα στιγμιότυπο μιας πλευράς της λειτουργικότητας του συστήματος
  - Το σύνολο όλων των Περιπτώσεων Χρήσης μας δίνει την εξωτερική εικόνα του συστήματος
  - Είναι ένα πολύ καλό εργαλείο για την κατανόηση του τι θέλουν οι χρήστες
- Διαγράμματα Κλάσεων (Class diagrams)
  - Βοηθούν τον εντοπισμό των κυρίαρχων εννοιών. Άρα βοηθούν πολύ την κατανόηση του πεδίου εφαρμογής (πολύ βασικό στην αρχή ενός έργου)
- Διαγράμματα Δραστηριοτήτων (Activity diagrams)
  - Χρήσιμα αν οι ροές εργασιών (workflow processes) είναι σημαντικό κομμάτι του κόσμου των χρηστών
    - Το ότι επιτρέπουν παραλληλία μας βοηθά να αποφύγουμε την καταγραφή περιττών ακολουθιών

Πόσοι τύποι διαγραμμάτων υπάρχουν;



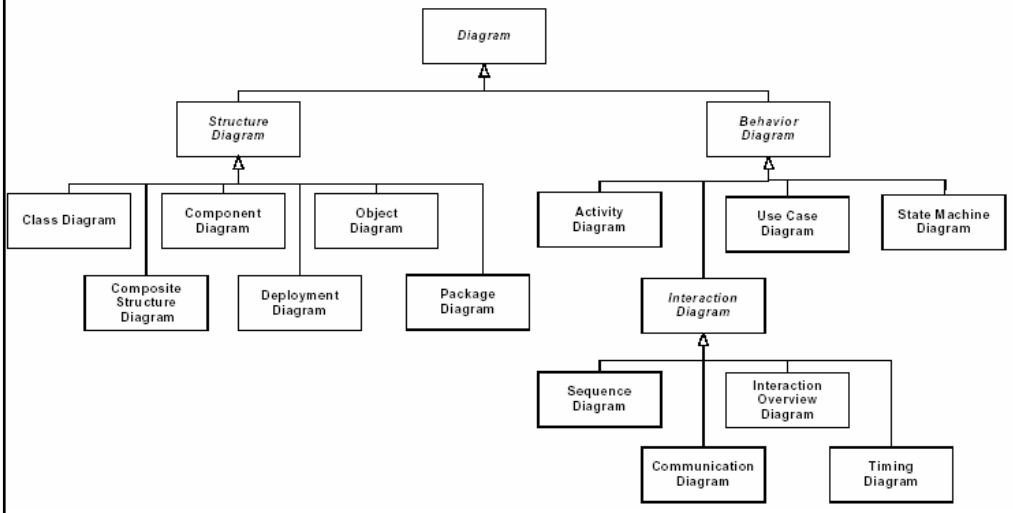
- Η UML 2.0 ορίζει 14 διαγραμματικές τεχνικές για τη μοντελοποίηση ενός συστήματος

Πόσοι τύποι διαγραμμάτων υπάρχουν;

Η UML 2.0 ορίζει 14 διαγραμματικές τεχνικές για τη μοντελοποίηση ενός συστήματος.

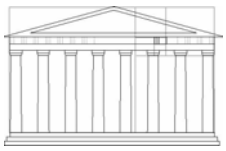
- **Λειτουργική**
  - Περιπτώσεων Χρήσης (Use Case), Δραστηριοτήτων (Activity),
- **Δομή**
  - Κλάσεων (Class) , Αντικειμένων (Object), Πακέτων (Package), Παράταξης (Deployment) , Εξαρτημάτων (Component) , Σύνθετης Δομής (Composite Structure)
- **Συμπεριφορά**
  - Sequence (Αλληλουχίας), Επικοινωνίας (Communication), Χρονισμού (Timing), Καταστάσεων (State), Interaction Overview, Protocol State Machine

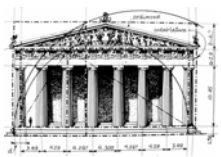
Μια άλλη κατηγοριοποίηση των διαγραμματικών τεχνικών της UML



Πότε (σε ποιες φάσεις) χρησιμοποιούμε ένα διάγραμμα; (when we use what diagram?)

- Διαφορετικές φάσεις του έργου συνήθως συνοδεύονται από διαφορετικού τύπου διαγράμματα
- Μερικοί τύποι διαγραμμάτων μπορούν να χρησιμοποιηθούν σε παραπάνω από μια φάση. Για παράδειγμα μπορεί να έχουμε ένα διάγραμμα που αρχικά είναι πολύ αφηρημένο (και φτιαγμένο από την εννοιολογική σκοπιά) που στη συνέχεια του προσθέσουμε λεπτομέρειες και εν τέλει από αυτό καταλήγουμε στην παραγωγή κώδικα.





Πόσο αυστηρά πρέπει να ακολουθούμε τους κανόνες της γλώσσας μοντελοποίησης;

Εξαρτάται από το σκοπό:

- Αν θέλουμε από τα διαγράμματα να παράγουμε κώδικα αυτομάτως (μέσω ενός εργαλείου CASE), τότε πρέπει να είμαστε ακριβείς και να λάβουμε υπόψη τον τρόπο με τον οποίο το εργαλείο ερμηνεύει τα διαγράμματα της UML και παράγει κώδικα.
- Αν θέλουμε να χρησιμοποιήσουμε τα διαγράμματα μόνο για να επικοινωνήσουμε με άλλους, μπορούμε να είμαστε αρκετά χαλαροί και παρεκκλίνουμε.

Hello World!
σε UML

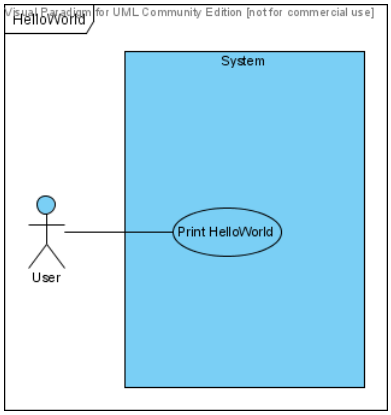
Hello World! σε UML
Use Case and Use Case Diagram

Use Case

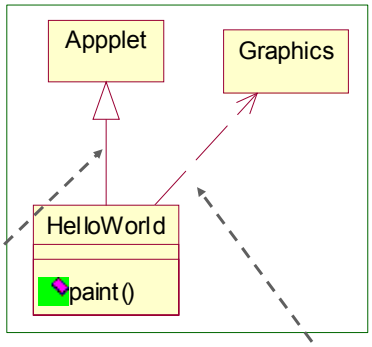
Print HelloWorld

1. Ο χρήστης ξεκινά έναν πλοηγητή του ιστού
2. Πληκτρίζει τη διεύθυνση www.csd.uoc.gr/helloworld
3. Ο πλοηγητής εμφανίζει στην οθόνη τη φράση "HelloWorld!"

Use Case Diagram



Hello World! σε UML
Class Diagram



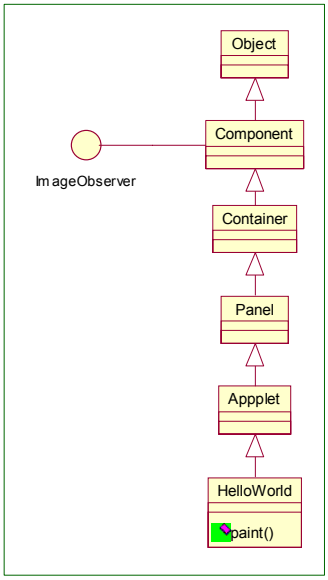

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello world!", 50, 25);
   }
}
```

Γενίκευση/Εξειδίκευση

Εξάρτηση (διότι δέχεται ως παράμετρο ένα αντικείμενο τύπου Graphics)

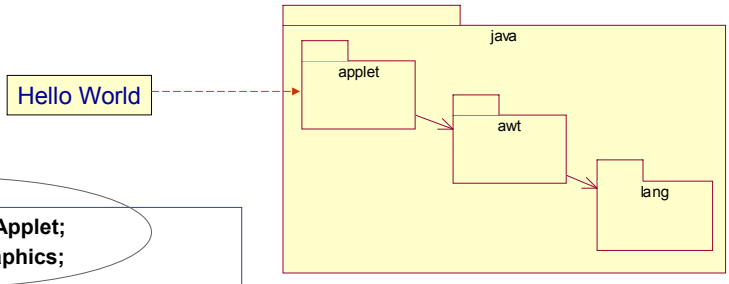
Hello World! σε UML
Class Diagram

Μελετώντας την βιβλιοθήκη της Java, μπορούμε να απεικονίσουμε ολόκληρη την ιεραρχία γενίκευσης της κλάσης Applet



Hello World! σε UML
Package Diagrams

Μπορούμε να απεικονίσουμε τον τρόπο με τον οποίο οι κλάσεις της Java εχουν πακεταριστεί με ένα διάγραμμα συσκευασίας. Το διάγραμμα αυτό απεικονίζει και τις εξαρτήσεις.



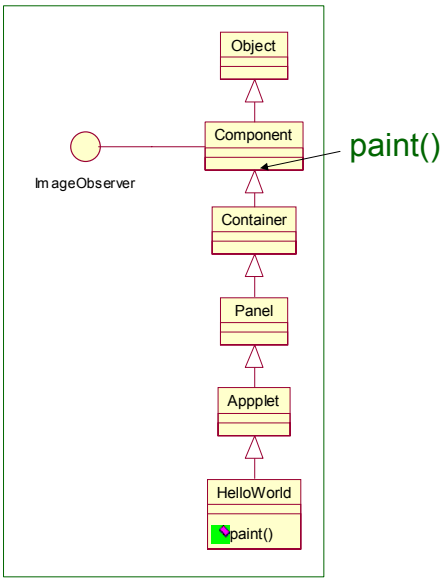

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello world!", 50, 25);
   }
}
```

Hello World! σε UML

Πως συνεργάζονται οι κλάσεις;
Πως η λειτουργία (operation) paint καλείται;

Μελετώντας το διάγραμμα κλάσεων της βιβλιοθήκης μπορούμε να δούμε ότι η paint κληρονομείται από από την κλάση component

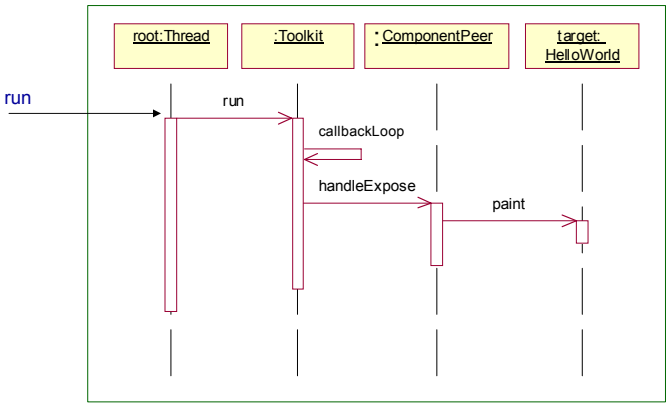
```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello world!", 50, 25);
   }
}
```



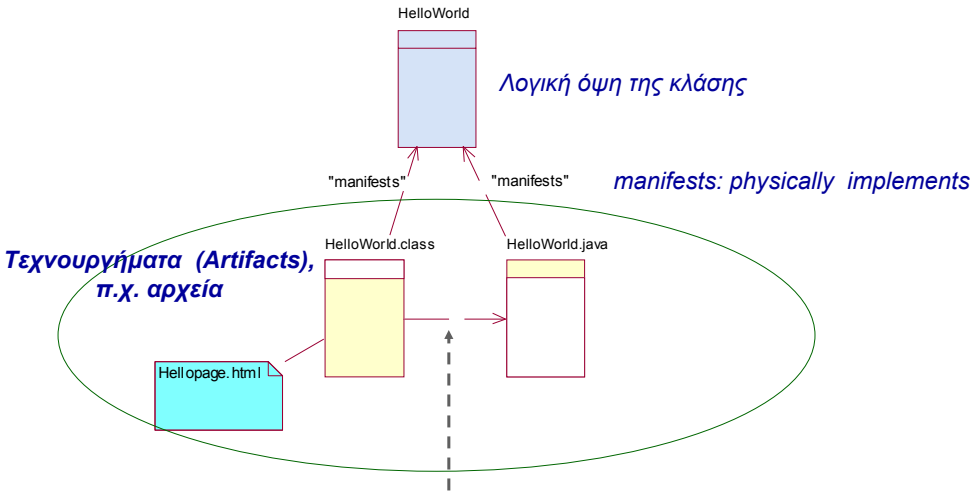
Hello World! σε UML
Interaction Diagram

Μελετώντας τον τρόπο με τον οποίο οι κλάσεις της Java συνεργάζονται, μπορούμε να δούμε ότι η λειτουργία paint καλείται ως εξής:

Καλείται από το thread που εγκλείει το applet

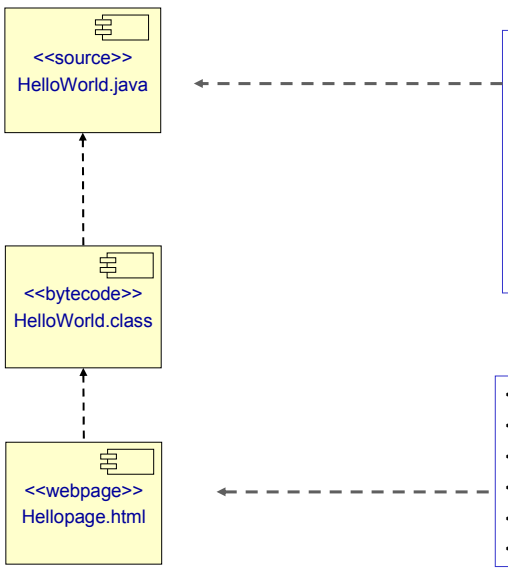


Hello World! σε UML: Η φυσική άποψη



Εξάρτηση (διότι αν αλλάξει το .java πρέπει να ξανακάνουμε compile)

Hello World! σε UML: Η φυσική άποψη ως component diagram



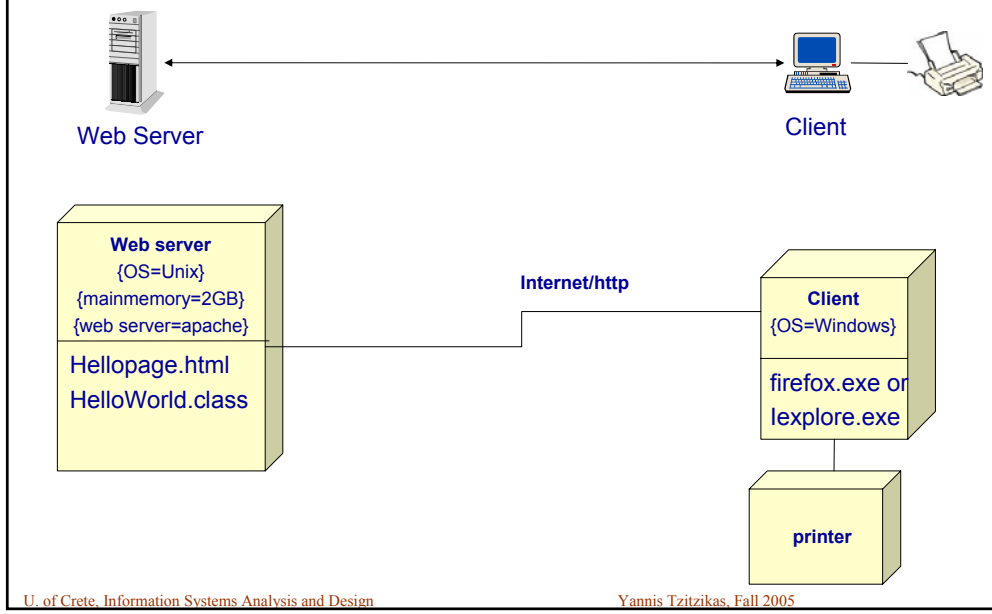
HelloWorld.java

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello world!", 50, 25);
   }
}
```

Hellopage.html

```
<html>
<body>
<APPLET CODE="HelloWorld.class">
</APPLET>
</body>
</html>
```

Hello World! σε UML
Deployment Diagram



Επισκόπηση των διαγραμματικών τεχνικών της UML

Λίστα των κυριότερων διαγραμματικών τεχνικών της UML και των χρήσεων τους

- **Use Case Diagram** (διάγραμμα περιπτώσεων χρήσης)
- **Activity Diagram** (διάγραμμα δραστηριοτήτων)
- **Class Diagram** (διάγραμμα κλάσεων)
- **Interaction Diagram** (διάγραμμα αλληλεπίδρασης)
  - Sequence Diagrams (διαγράμματα αλληλουχίας)
  - Communication Diagrams (διαγράμματα επικοινωνίας)
- **State Diagram** (διάγραμμα καταστάσεων)
- **Component Diagram** (διάγραμμα εξαρτημάτων)
- **Package Diagram** (διάγραμμα πακέτων/συσκευασίας)
- **Deployment Diagram** (διαγράμματα παράταξης)

Περιπτώσεις Χρήσης (Use Cases)

Περίπτωση Χρήσης (Use Case) = ένα σύνολο σεναρίων για την επίτευξη ενός σκοπού του χρήστη

- **Σενάριο =** μια ακολουθία βημάτων που περιγράφουν την αλληλεπίδραση μεταξύ χρήστη και συστήματος

Αγορά Προϊόντος

1. Ο Πελάτης πλοηγείται στον κατάλογο και επιλέγει τα προϊόντα που επιθυμεί να αγοράσει
2. Ο Πελάτης επιλέγει τη λειτουργία «Παραγγελία»
3. Ο Πελάτης συμπληρώνει τα στοιχεία αποστολής (δνση, παράδοση σε 24 ώρες, παράδοση σε 3 ημέρες)
4. Το Σύστημα του παρουσιάζει αναλυτικά την τιμή της παραγγελίας (περιλαμβανομένων των εξόδων αποστολής)
5. Ο Πελάτης συμπληρώνει τα στοιχεία της πιστωτικής του κάρτας
6. Το Σύστημα ελέγχει τα στοιχεία της κάρτας (εξουσιοδότηση, πιστωτικό όριο, ..)
7. Το Σύστημα επιβεβαιώνει την πώληση αμέσως
8. Το Σύστημα στέλνει ένα η-μήνυμα επιβεβαίωσης στον Πελάτη

Εναλλακτική: *Authorization Failure*

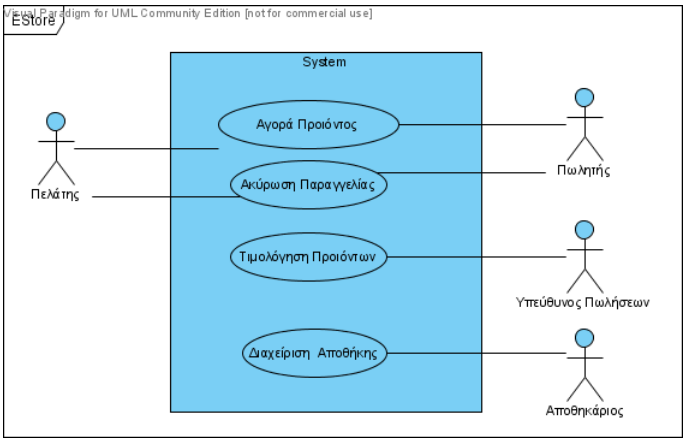
Στο βήμα 6, το Σύστημα αποτυγχάνει να εγκρίνει την αγορά μέσω πιστωτικής.

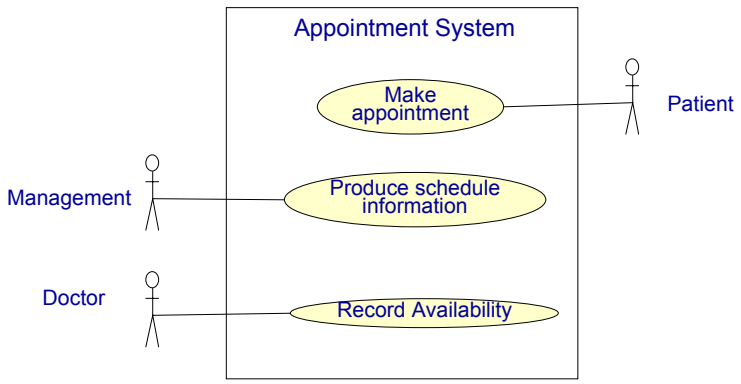
Ο Πελάτης μπορεί να ξαναδώσει τα στοιχεία της πιστωτικής του κάρτας και να ξαναπροσπαθήσει

UML Techniques
Use Case Diagrams (διαγρ. περιπτώσεων χρήσης)

Used for: **Analysis**
Concerns: **Behavior**

Παρουσιάζει του τύπους χρηστών (actors), τις περιπτώσεις χρήσης και τις μεταξύ τους συσχετίσεις.





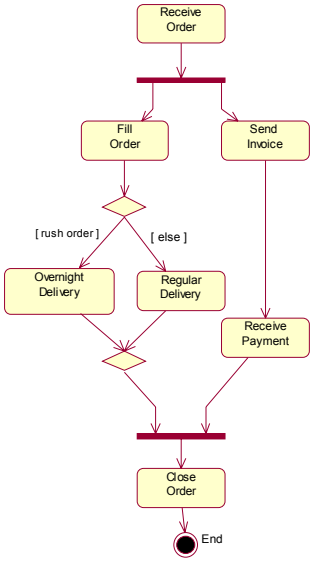
UML Techniques
Activity Diagrams (διαγρ. δραστηριοτήτων)

Used for: **Analysis/Design**
Concerns: **Behavior**

Περιγράφουν τη ροή των εργασιών.

Μπορούν να χρησιμοποιηθούν για να περιγράψουν τη ροή εργασιών:

- σε έναν οργανισμό
- σε μια περίπτωση χρήσης
- σε μια μέθοδο μιας κλάσης

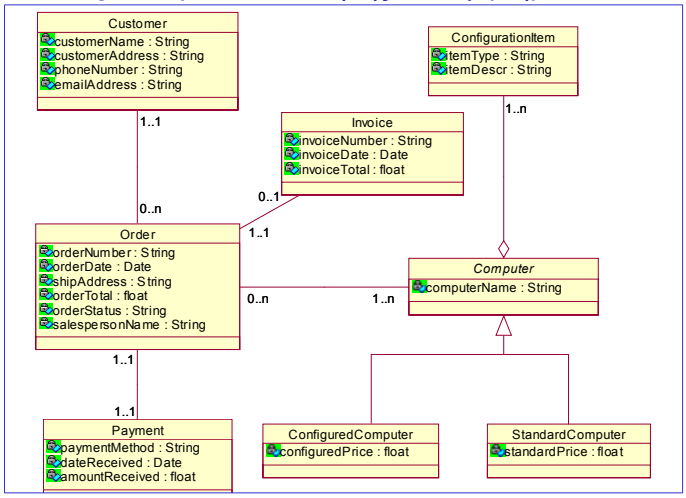


Ο τρόπος διεκπεραίωσης των παραγγελιών ενός οργανισμού

UML Techniques
Class Diagrams (διαγράμματα κλάσεων)

Used for: **Analysis/Design**
Concerns: **Structure**

Παρουσιάζουν τη στατική δομή εννοιών, τύπων και κλάσεων. Οι *έννοιες* δείχνουν πως οι χρήστες βλέπουν τον κόσμο, οι *τύποι* τις διεπαφές των εξαρτημάτων λογισμικού, οι *κλάσεις* των τρόπο υλοποίησης των εξαρτημάτων.

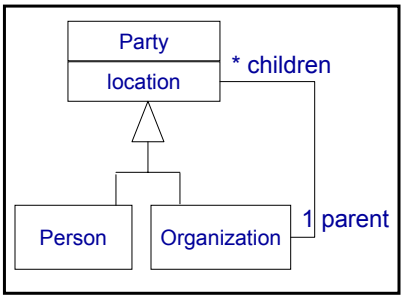


UML Techniques
Object Diagrams (διαγράμματα αντικειμένων)

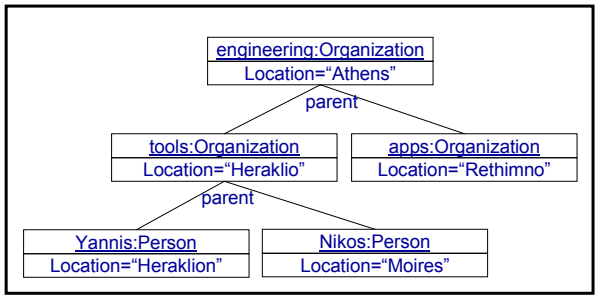
Παρουσιάζουν τη δομή των αντικειμένων στο σύστημα

Used for: **Analysis/Design**
Concerns: **Structure**

Class diagram



Object diagram



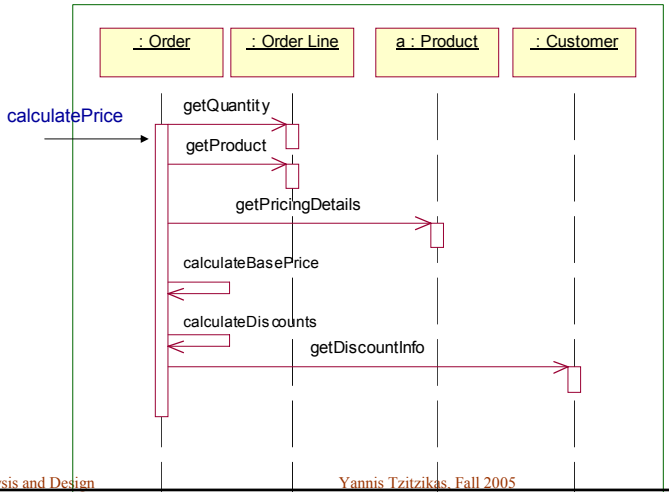
UML Techniques
Interaction Diagrams (διαγρ. αλληλεπίδρασης)

Used for: **Analysis/Design**
Concerns: **Behavior**

Παρουσιάζουν τον τρόπο με τον οποίο πολλά αντικείμενα συνεργάζονται σε μια Περίπτωση Χρήσης

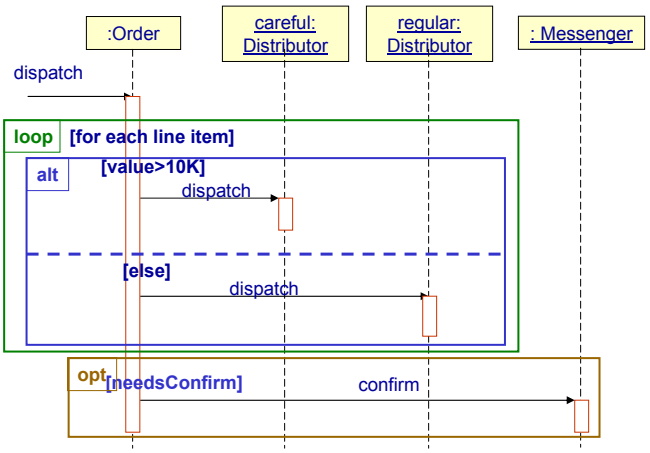
(A) **Sequence** Diagrams (διαγράμματα αλληλουχίας)

Υπολογισμός Τιμής



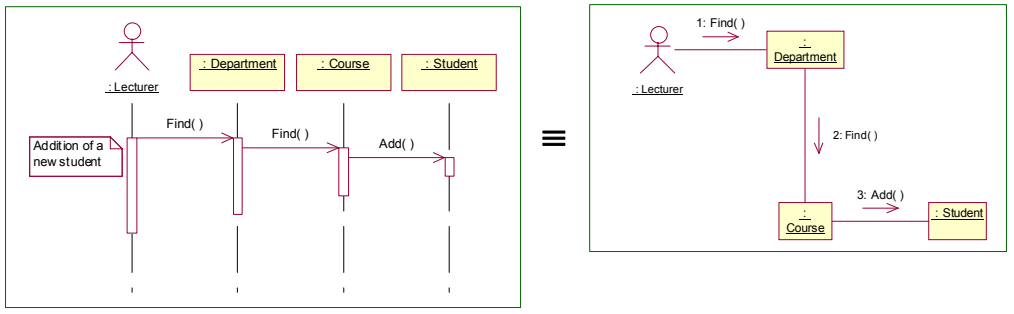
UML Techniques
Interaction Diagrams (διαγρ. αλληλεπίδρασης)

Αποστολή παραγγελίας



UML Techniques
Interaction Diagrams (διαγρ. αλληλεπίδρασης)

Sequence Diagram **Communication** Diagram



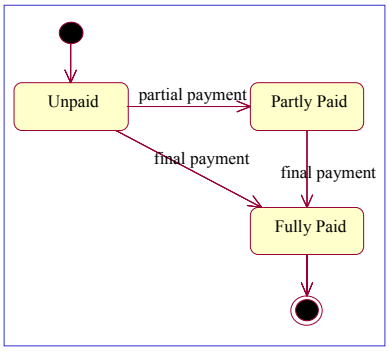
UML Techniques
State Diagrams (διάγρ. καταστάσεων)

Used for: **Analysis/Design**
Concerns: **Behavior**

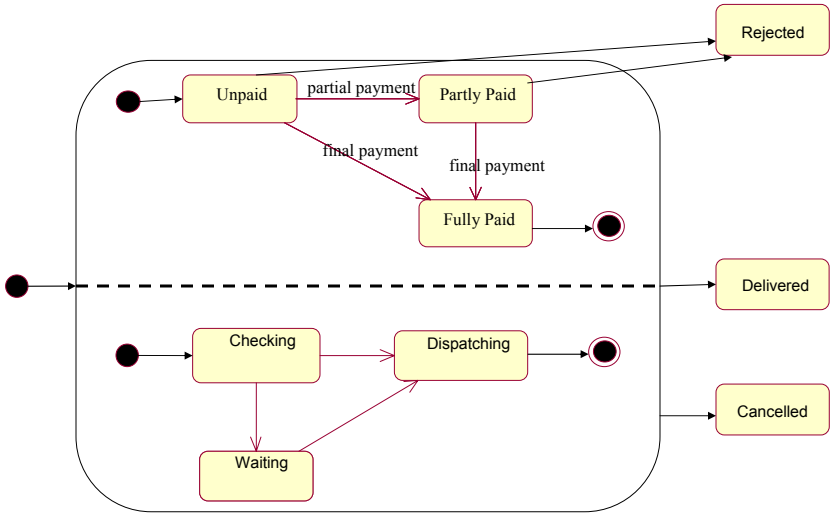
Παρουσιάζει τις καταστάσεις που μπορεί να έχει ένα αντικείμενο (σε όλη τη διάρκεια ζωής του) και πως αυτές αλλάζουν ανάλογα με τα γεγονότα (events) που φθάνουν στο αντικείμενο

- άρα δεν περιοριζόμαστε σε μια Use Case

Οι καταστάσεις μιας Παραγγελίας:



UML Techniques
State Diagrams (διάγρ. καταστάσεων)

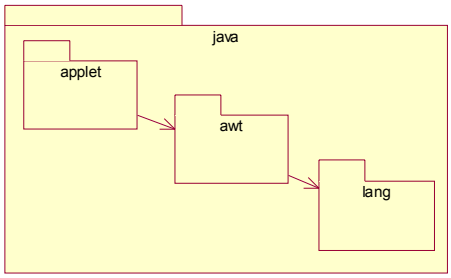


UML Techniques
Package Diagrams (διαγρ. συσκευασίας)

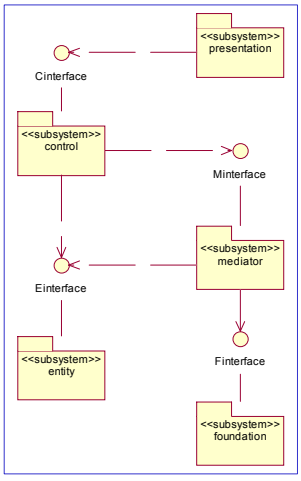
Used for: **Analysis/Design/Implenent**
Concerns: **Structure**

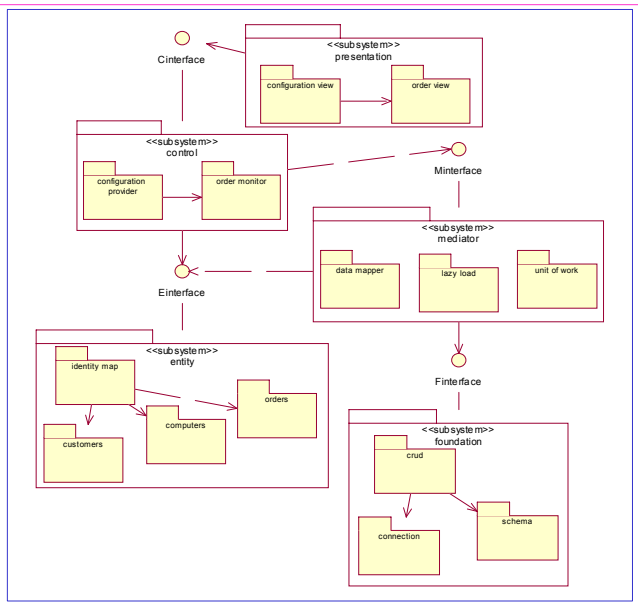
Είναι ένας μηχανισμός ομαδοποίησης. Παρουσιάζει τις ομάδες κλάσεων και τις εξαρτήσεις μεταξύ τους.

(μπορεί επίσης να ομαδοποιήσει και διαγράμματα της UML)



UML Techniques
Package Diagrams

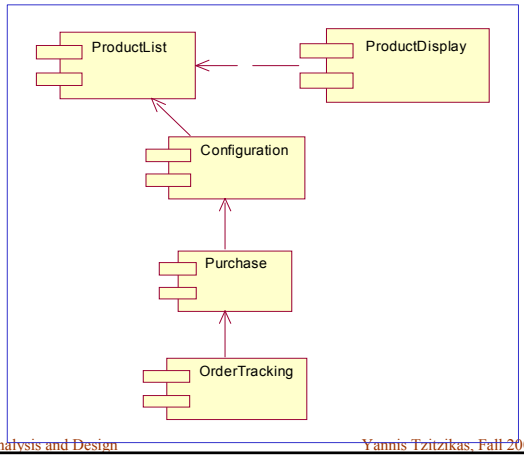




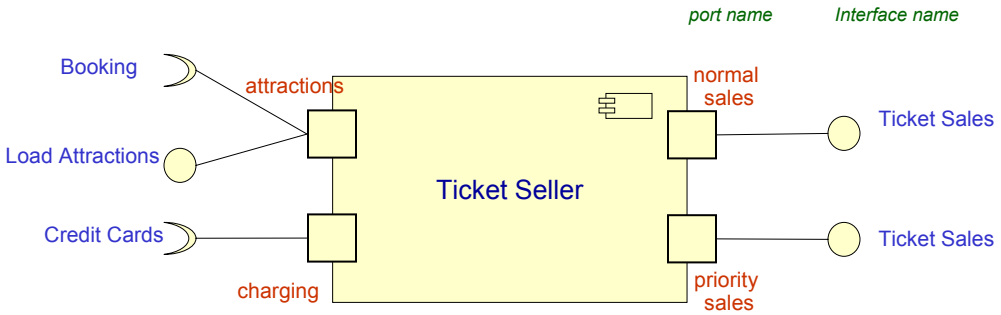
UML Techniques
Component Diagrams (διαγρ. εξαρτημάτω

Used for: **Phys./Design/Implenent**
Concerns: **Structure**

- Εξάρτημα: ένα λογικό και αντικαταστάσιμο τμήμα του συστήματος το οποίο *συμμορφώνεται με* και *πραγματώνει* ένα σύνολο από διεπαφές (interfaces)
  - Component: a logical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.



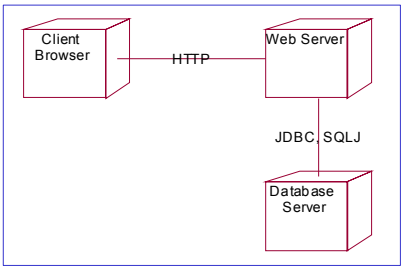
UML Techniques
Component Diagrams (διαγρ. εξαρτημάτων)



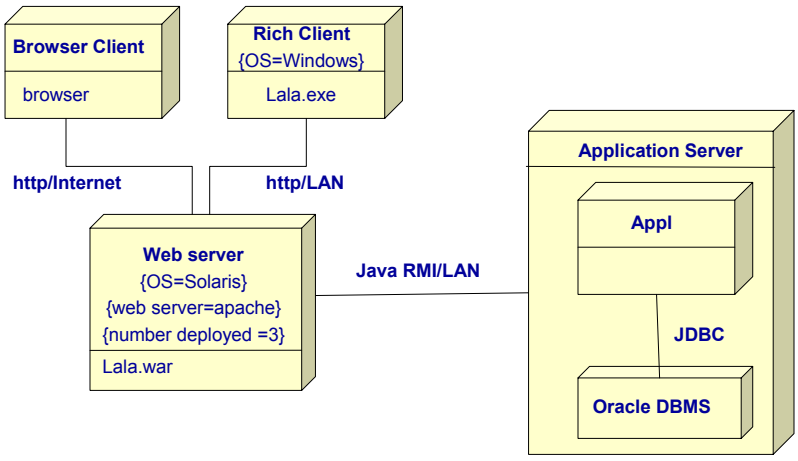
UML Techniques
Deployment Diagrams (διαγρ. παράταξης)

Used for: **Phys./Design/Implenent**
Concerns: **Structure**

Παρουσιάζει την φυσική τοποθέτηση των εξαρτημάτων στους κόμβους υλικού.

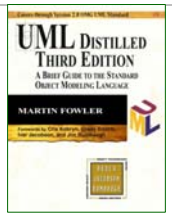


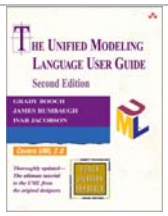
UML Techniques
Deployment Diagrams (διαγρ. παράταξης)



Βιβλία και Πηγές για την UML

- Η σχετική σελίδα της OMG:
  - UML Resource Page: http://www.uml.org/
  - The UML 2.0 Specification: http://www.uml.org/#UML2.0
- Βιβλία
  - **UML Distilled: A Brief Guide to the Standard Object Modeling Language** (3rd Edition) by Martin Fowler, Addison Wesley, 2004.
  - **The Unified Modeling Language User Guide** (3rd edition) by G. Booch, J. Rumbaugh, I. Jacobson, Addison Wesley, 2005





Βιβλία και Πηγές για την UML

Υπάρχουν πολλές εκπαιδευτικές παρουσιάσεις (tutorials) στο διαδίκτυο:

- **http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/index.htm**
- **http://www.sparxsystems.com.au/UML_Tutorial.htm**
- **http://bdn.borland.com/article/0,1410,31863,00.html**
- **http://odl-skopje.etf.ukim.edu.mk/uml-help/**
- **…. Google: UML Tutorial**