HY352 : ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

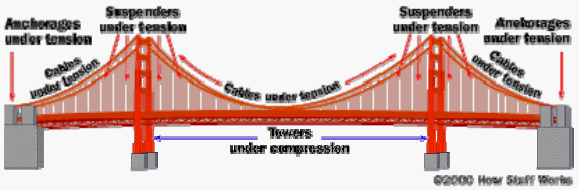
ΔΙΔΑΣΚΩΝ
Αντώνιος Σαββίδης

CSD

ΕΝΟΤΗΤΑ 2

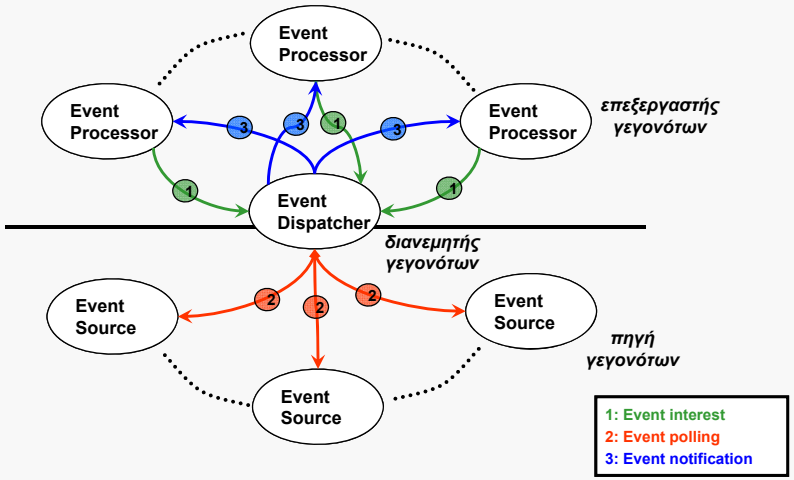
ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΧΕΔΙΑΣΗ ΚΑΙ ΟΙΚΟΓΕΝΕΙΕΣ ΑΡΧΙΤΕΚΤΟΝΙΚΩΝ

Αριθμός διαλέξεων 2



CSD

Event-based architectures (1/8)



CSD

Event-based architectures (2/8)

- Ορισμοί (1/2)
  - ***Event***. Τυποποιημένη πληροφορία που παράγεται δυναμικά, δεν αποθηκεύεται από την πηγή, ενδιαφέρει αρκετούς αποδέκτες οι οποίοι είναι υπεύθυνοι για συγκεκριμένες ενέργειες, και συνήθως δεν αποθηκεύεται ούτε από τους αποδέκτες. Π.χ.
    - **Key press**
      - αρχική πηγή ο device driver του πληκτρολογίου,
      - τελικός αποδέκτης η εκάστοτε εφαρμογή,
      - τύπος δεδομένων: { `unsigned keyCode;` }
    - **Mouse repositiong**
      - Πηγή και αποδέκτης ομοίως με το προηγούμενο
      - Τύπος δεδομένων: { `unsigned x, y;` }
    - **Motion detection**
      - Αρχική πηγή device driver του φωτοκύτταρου, αποδέκτης το σύστημα ασφαλείας
      - Τύπος δεδομένων: {`sensorid_t whichSensor;` }

Event-based architectures (3/8)

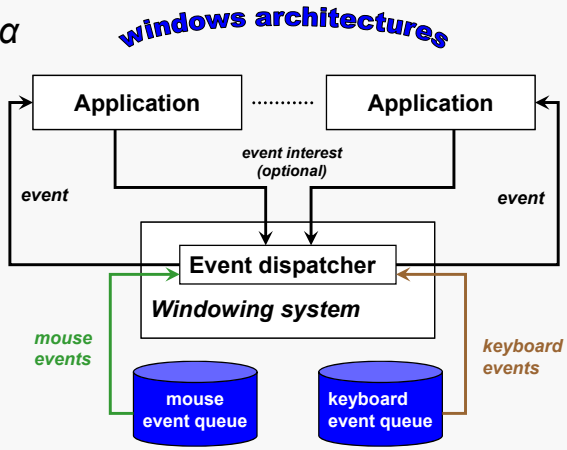
- Ορισμοί (2/2)
  - ***Event history***: Η συσσώρευση των γεγονότων στον διανεμητή (list), με τη σειρά την οποία συμβαίνουν, με επιπλέον προσθήκη ετικετών χρόνου (timestamps).
    - Η κατασκευή του event history δεν πρέπει να επηρεάζει την συμπεριφορά του διανεμητή - τα events συνεχίζουν να διανέμονται κανονικά στους επεξεργαστές.
  - ***Event playback***: Η λειτουργία διανομής events από τον διανεμητή κατά την οποία τα events δεν λαμβάνονται *in real time* από πραγματικές πηγές, αλλά από ένα εκάστοτε εναποθηκευμένο event history
    - με πλήρη εξομοίωση και του χρονισμού παραγωγής - δηλαδή σεβόμαστε τα timestamps
- Η συνήθης χρήση των παραπάνω είναι σε περίπτωση αποτυχιών κυρίως σε κατανεμημένα συστήματα, ή για την αναπαραγωγή ενός execution sessions (όσο ακριβής επιτρέπεται να είναι αυτή)

Event-based architectures (4/8)

- ***Ιδιότητες***
  - Το event model λέγεται και callback model
  - Το τμήμα που δηλώνει ενδιαφέρον για events δεν χρειάζεται να γνωρίζει πως και ποιος τα παράγει
  - Ο κεντρικός διανεμητής των events δεν εξαρτάται από το ποιος θα τα επεξεργαστεί
  - Μπορούν να γίνουν επεκτάσεις τόσο στις κατηγορίες των events όσο και στην επεξεργασία, με την εισαγωγή επιπλέον event processors, χωρίς να επηρεάζονται οι υπόλοιποι
  - Υποστηρίζεται πλήρως επαναχρησιμοποίηση των event processors
  - Μπορούμε να προσθέσουμε event processors on-the-fly (κατά την διάρκεια λειτουργίας του συστήματος – in real time)
  - Η σειρά διανομής ενός event είτε θεωρείται τυχαία, ή ορίζεται ως η σειρά εκδήλωσης ενδιαφέροντος από τους επεξεργαστές
  - → *Πολλές φορές εφαρμόζεται και ως micro-architecture*

Event-based architectures (5/8)

- *Παράδειγμα*



Event-based architectures (6/8)

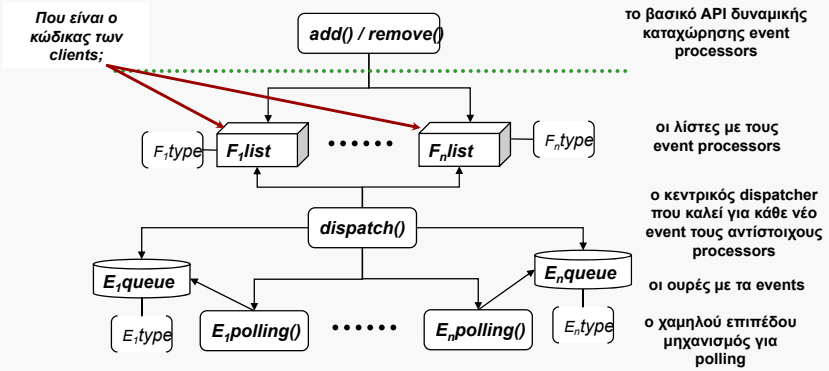
- *Δομή κώδικα (1/2) – C style*

Event data types	***E_1type,…,$E_n$type***
Event-processor function-types	***F_1type,…,$F_n$type***
Event interest control	***add(), remove()***
Event-processor lists	***F_1list,…,$F_n$list***
Event queues	***E_1queue,…,$E_n$queue***
Event polling controller	***E_1polling(),…,$E_n$polling()***
Event dispatcher	***dispatch(), mainloop()***

- Στις event-based αρχιτεκτονικές, αρχικά χτίζεται ο παραπάνω βασικός πυρήνας - *infrastructure*, και στη συνέχεια υλοποιείται το υπόλοιπο λογισμικό χρησιμοποιώντας αυτόν τον βασικό πυρήνα.
  - Δηλ. τους ανεξάρτητους επεξεργαστές των events, οι οποίοι, ανάλογα με την πολυπλοκότητα του συστήματος, μπορεί στο σύνολό τους να αποτελούν πολλαπλάσια ποσότητα κώδικα συγκριτικά με τον ίδιο τον πυρήνα.

Event-based architectures (7/8)

- *Δομή κώδικα (2/2) – micro-architecture*



Event-based architectures (8/8)

- Παράδειγμα – *X Windowing System / Xt*

X Windows Xt – τρόπος διαχείρισης γεγονότων από συσκευές

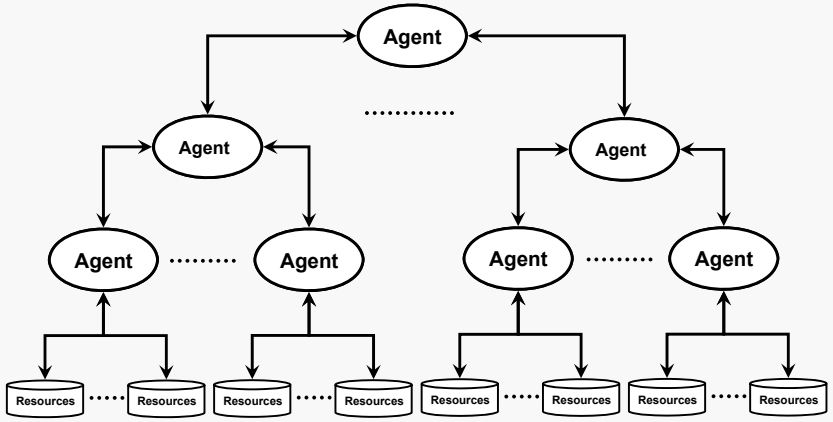
```
typedef void (*XtEventHandler)(Window, XtPointer, XEvent*, Boolean*);
struct XEvent {
...
};
extern XtAddEventHandler (Window, unsigned, Boolean, XtEventHandler, XtPointer);
extern XtRemoveEventHandler (Window, unsigned, Boolean, XtEventHandler, XtPointer);
```

```
void OnMouseMove (Window w, XtPointer unused, XEvent* event, Boolean* dispatchMore) {
}
XtAddEventHandler(widget, PointerMotionMask, false, OnMouseMove, (XtPointer) NULL);
XtRemoveEventHandler(widget, PointerMotionMask, false, OnMouseMove, (XtPointer) NULL);
```

`XtEventHandler`	$F_1type \cup F_ntype$
`XEvent`	$E_1type \cup E_ntype$
`XtAddEventHandler`	add()
`XtRemoveEventHandler`	remove()

Agent-based architectures (1/7)

1. task specialization / εξειδίκευση εργασιών
2. agent control / έλεγχος πρακτόρων
3. dynamic discovery / δυναμική ανακάλυψη
4. dynamic substitution / δυναμική αντικατάσταση
5. failure handling / διαχείριση αποτυχίας



Agent-based architectures (2/7)

- *Ορισμοί*
  - ***Agent / πράκτορας***. Ένα ανεξάρτητο λογισμικό τμήμα με τις εξής ιδιότητες:
    - μπορεί να επιτελεί ένα συγκεκριμένο έργο το οποίο του ανατίθεται ως είσοδος – *agent task*
    - μπορεί να παράγει προαιρετικά αποτελέσματα τα οποία τα τροφοδοτεί στο τμήμα το οποίο του ανέθεσε την διεκπεραίωση – *task results*
    - κρύβει τις λεπτομέρειες υλοποίησης του έργου, ενώ λειτουργεί ανεξάρτητα και παράλληλα για την διεκπεραίωσή του – *independent implementation / parallel execution*
    - προσφέρει δυνατότητες ελέγχου της λειτουργικής συμπεριφοράς του τόσο πριν - *static configuration* - όσο και κατά την εκτέλεση του έργου - *dynamic control*
    - υποστηρίζει δικαιώματα ελέγχου, έτσι ώστε να μπορεί να δέχεται εντολές (και αντίστοιχα να δίνει) μόνο από (σε) συγκεκριμένα τμήματα (τα οποία είναι συνήθως και αυτά agents) – *access privileges*

Agent-based architectures (3/7)

- *Ιδιότητες*
  - *Δυναμική ιεραρχία* από agents, με πολλαπλά επίπεδα ελέγχου - εργασιών
    - Οι agents σε κάθε επίπεδο ιεραρχίας έχουν και διαφορετική εργασία να επιτελέσουν
  - *Δυναμική χρήση* agents. Ανάλογα με τις απαιτούμενες λειτουργίες σε κάθε επίπεδο οι agents που παρέχουν πιο εξειδικευμένες λειτουργίες αναζητούνται και εμπλέκονται δυναμικά (run-time).
  - Υπάρχει επικοινωνία μόνο μεταξύ agents διαδοχικών επιπέδων - «*υπακοή στην ιεραρχία*»
  - *Κατανεμημένος έλεγχος*, τμηματοποίηση και ιεράρχηση, καθώς και υποστήριξη επαναχρησιμοποίησης
  - Συνήθως *στο κατώτερο επίπεδο* διαχειρίζονται συγκεκριμένοι *πόροι* από εξειδικευμένους system agents

Agent-based architectures (4/7)

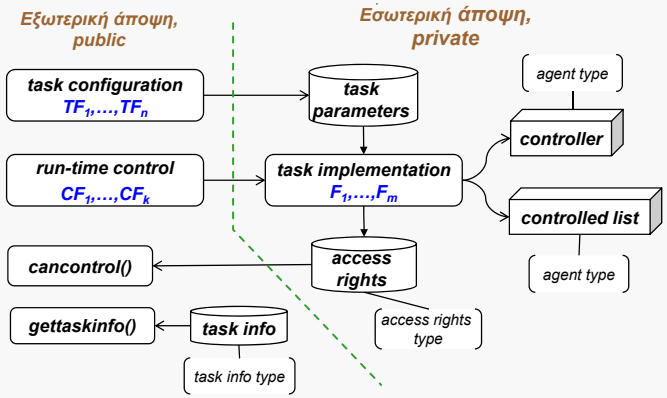
- Δομή κώδικα agent (1/2) – *απλοποίηση C style*

Task configuration functions	$TF_1,\ldots,TF_n$
Agent run-time control functions	$CF_1,\ldots,CF_k$
Task implementation functions	$F_1,\ldots,F_m$
Controller agent	**controller**
Controlled agents	**controlled list**
Task and state query functions	**gettaskinfo(), isbusy(), grant(), release()**
Access privileges	**cancontrol()**

- Ένας agent ***X***, στην υλοποίηση του τρόπου εκτέλεσης του έργου του ***E***, μπορεί:
  - να ψάξει δυναμικά άλλους διαθέσιμους agents ***Y,*** δηλ. ***!isbusy(Y)*** ώστε να εκτελέσουν κομμάτι E_j του έργου ***E***
  - τους οποίους έχει δικαίωμα να ελέγξει, δηλ. ***cancontrol(X,Y) == true***
  - αφού επιβεβαιώσει ότι διεκπεραιώνουν έργα E_j, δηλ. ***gettaskinfo(Y)==***E_j
  - και τελικά τους αποκτήσει μέσω τη κλήσης ***grant(X,Y)***

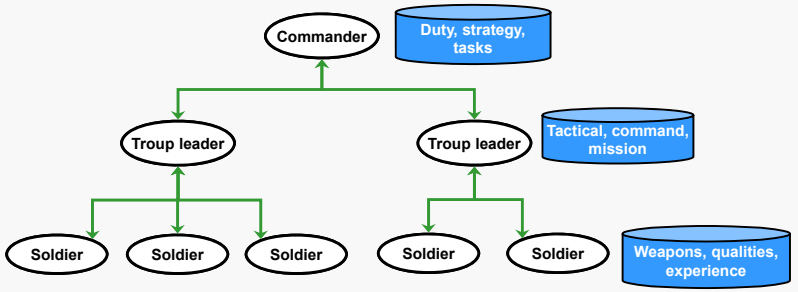
Agent-based architectures (5/7)

- Δομή κώδικα agent (2/2) – *micro-architecture*



Agent-based architectures (6/7)

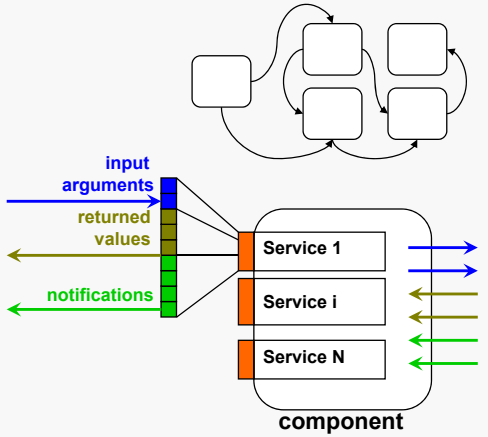
- *Παράδειγμα (1/2) – war simulators*



Agent-based architectures (7/7)

- *Παράδειγμα (2/2)*
  - Οι agents, εκτός από την επικοινωνία με άλλους agents, μπορούν επιπλέον να επικοινωνούν με:
    - ανθρώπους, δηλ. να έχουν και User Interface
    - με το περιβάλλον, δηλ. να έχουν ειδικούς sensors η / και actuators, ακόμη και πιο πολύπλοκο συστήματα αντίληψης (computer vision)
  - Αυτό επιτρέπει την χρήση agent αρχιτεκτονικών για πληθώρα συστημάτων
    - Work-flow management (αυτοματισμό ροής και περάτωσης εργασιών σε μία επιχείρηση)
    - Process monitoring and control (παρακολούθηση και έλεγχος προόδου εργασιών)
    - Automated data collection and analysis (αυτοματοποιημένη συλλογή και ανάλυση πληροφοριών)
    - Surveillance and alert management systems (συστήματα παρακολούθησης και διαχείρισης συναγερμών)

Component architectures (1/4)



Component architectures (2/4)

- *Ιδιότητες*
  - Τα διάφορα components είναι ανεξάρτητα μεταξύ τους και κρύβουν πλήρως τις λεπτομέρειες υλοποίησης
  - Μπορεί να γίνει αντικατάσταση των components on-the-fly, εφόσον ικανοποιούν το ίδιο API
  - Η λεπτομερής αρχιτεκτονική ποικίλει ανάλογα με την τοπολογία / συνδεσμολογία των components
  - Πρακτικά κάθε αρχιτεκτονική μπορεί να περιγραφεί ως μία οργανωμένη δομή από components
    - Μπορούν να χρησιμοποιηθούν components τόσο για την macro-architecture όσο και για τις επιμέρους micro-architectures
  - Υποστηρίζεται η δημιουργία (instantiation) πολλών στιγμιότυπων για κάθε component at run-time, ανάλογα με τις λειτουργικές ανάγκες
  - ➔ *Η αξία αυτού του αρχιτεκτονικού μοντέλου έγκειται στον εντοπισμό μίας γενικής αρχιτεκτονικής οντότητας με την οποία μπορούμε να μοντελοποιήσουμε όλες τις άλλες αρχιτεκτονικές*

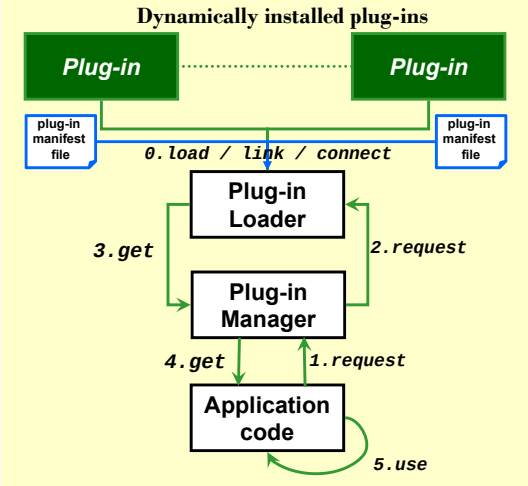
Component architectures (3/4)

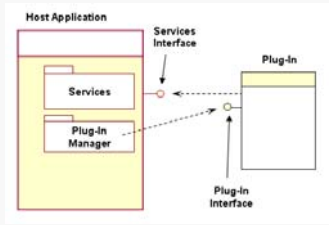
- Η έννοια του component συνεπάγεται συνήθως αρχιτεκτονικές δομές υψηλού επιπέδου
  - Θεωρούμε ότι ένα component αποτελείται από μερικές κλάσεις
  - Μερικές φορές χρησιμοποιούμε τον όρο module ως συνώνυμο του component
    - Ωστόσο το *modular programming* είναι κάτι άλλο από το *component-oriented programming*
  - Πολλές φορές αναφερόμαστε σε ένα package ως μία συλλογή από components – δηλαδή ακόμη υψηλότερου επιπέδου αρχιτεκτονικά τμήματα
  - Οι παραπάνω διαχωρισμοί δεν είναι δογματικοί – μπορεί κάποια components να είναι στην πράξη μία κλάση και κάποια packages να έχουν απλώς ένα component
  - Υπάρχουν τεχνολογίες που υποστηρίζουν τη χρήση components ανεξάρτητα από τη γλώσσα προγραμματισμού – αυτό λέγεται *binary format reuse*

Component architectures (4/4)

- Παραδείγματα τεχνολογιών υλοποίησης που μπορούν να υποστηρίξουν components σε επίπεδο
  - Macro-architecture – *διαχωρισμός σε διαφορετικά προγράμματα, binary level reuse*
    - OMG / CORBA (Common Object Request Broker Architecture)
    - Microsoft DCOM (Distributed Component Object Model), ActiveX
    - Java Beans.
    - RPC (Remote Procedure Calls), RMI (Remote Method Invocation)
    - XML RPC, SOAP (Simple Object Access Protocol)
    - OSGi (mainly Java)
  - Micro-architecture – *διαχωρισμός σε διαφορετικά τμήματα του ίδιου προγράμματος, source or lib level reuse*
    - Σε OOP languages σχεδίαση εξειδικευμένων classes
    - Σε procedural languages υλοποίηση ειδικών APIs

Plug-in architecture (1/9)





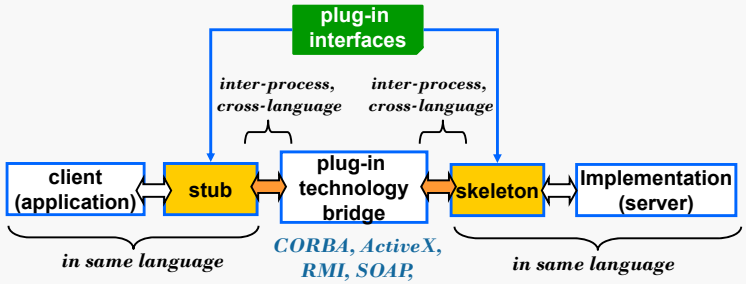
Plug-in architecture (2/9)

- *Ιδιότητες*
  - Τα plug-ins μπορούν να ***ενεργοποιούνται δυναμικά***
  - Τα plug-ins ***δεν*** χρειάζεται να ***γνωρίζονται μεταξύ τους***
  - Μπορούν να οριστούν ***διαφορετικές κατηγορίες plug-ins*** με αντίστοιχα APIs
  - Η βασική ***εφαρμογή*** (host application) ***μικραίνει*** σε μέγεθος κώδικα
  - Οι ***αναβαθμίσεις*** και ***διορθώσεις*** μπορούν να ***διανέμονται*** επιλεκτικά σε επίπεδο plug-in ***μετά την κυκλοφορία*** του προϊόντος
  - Η ***εφαρμογή*** μπορεί να ***επεκτείνεται δυναμικά***
- Είναι περίπτωση component-based αρχιτεκτονικής όπου κάθε plug-in μπορεί να θεωρηθεί ως ανεξάρτητη προσαύξηση
  - αν και υπάρχουν περιπτώσεις όπου plug-ins μπορούν να χτίζονται πάνω από άλλα plug-ins
  - η ***διαλειτουργικότητα*** μεταξύ των plug-ins ***απαιτεί*** ένα προσεκτικά σχεδιασμένο ***interoperation framework*** (communication, events, signals, messages, κλπ) που θα ***παρέχεται από το host application environment***

Plug-in architecture (3/9)

- *Ειδική χρηστικότητα*
  - υποστήριξη επεκτάσεων της εφαρμογής ακόμη και από third-party developers
    - δημοφιλής πρακτική σε *browsers, IDEs, media players (decoders)* αλλά και σε *office systems*
  - υποστήριξη χαρακτηριστικών και λειτουργιών που δεν έχουν ακόμη προβλεφθεί
    - εφόσον ταιριάζουν με τα προβλεπόμενα plug-in APIs
  - μείωση του όγκου της υλοποίησης της βασικής εφαρμογής
    - ανάπτυξη των plug-ins μέσω outsourcing
  - γρήγορη παροχή της βασικής εφαρμογής στους πελάτες
    - με δωρεάν προσφορά των επιπλέον χαρακτηριστικών ως plug-ins

Plug-in architecture (4/9)



Γενική περίπτωση όταν τα plugins είναι σε οποιαδήποτε γλώσσα και πρέπει να μπορούν να διαχειριστούν με ένα συμφωνημένο API

Plug-in architecture (5/9)

Client-side

```
// A_component_stub.h
class A_component {
    public:
    T1  f1 (f1 args here...);
    T2  f2 (f2 args here...);
    ...rest of methods go here..
}

// A_component_stub.cpp
T1 A_component::f1 (f1 args here...) {

    Comp::Values f1_args;
    Marshal (package) every arg in f1_args;

    Comp::Object self;
    Marshal (package) this object in self;

    Comp::Invoke("A::f1", self, f1_args);

    Comp::Value ret;
    Comp::GetResponse(&ret);

    T1 f1_ret;
    Convert from ret to f1_ret;
    return f1_ret;
}
```

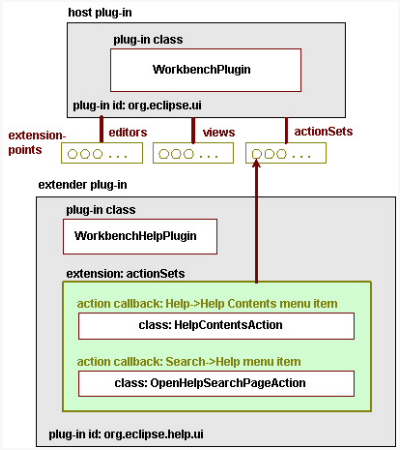
stub: πλήρης κώδικας που παράγεται αυτόματα ώστε ο client να μπορεί να χρησιμοποιήσει ένα remote component

skeleton: ημιτελής κώδικας που παράγεται αυτόματα για την υλοποίηση ενός component στην πλευρά του server

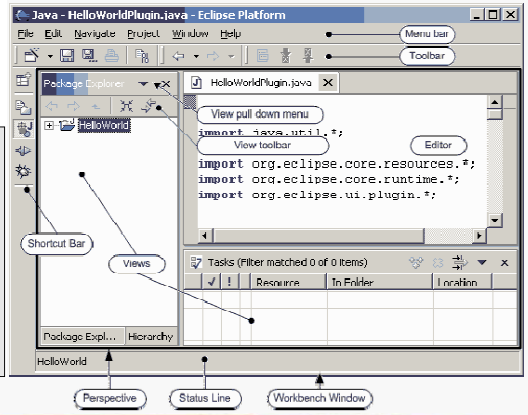
```
// A_component_skeleton.h
class A_component {
    public:
    T1  f1 (f1 args here...);
    T2  f2 (f2 args here...);
    ...rest of methods go here..
}

// A_component_skeleton.cpp
T1 A_component::f1 (f1 args here...) {
    /* Fill in implementation here*/
}
T2 A_component::f2 (f2 args here...) {
    /* Fill in implementation here*/
}
```

Server-side

Plug-in architecture (6/9)



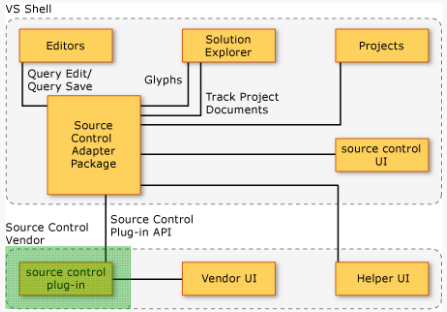
Example: *Eclipse – διαφορετικό interface για κάθε ξεχωριστό τύπος plug-in*



Plug-in architecture (7/9)

- Το παράδειγμα μας δείχνει τον τρόπο επέκτασης μέσω plug-ins της βασικής διεπαφής (εδώ λέγεται workbench)
  - Τα menus του workbench UI μπορούν να επεκταθούν δυναμικά από τα plug-ins μέσω μίας ειδικής πόρτας επέκτασης γνωστής ως ***action sets***
- Το *extender plug-in* στο παράδειγμα μας αφορά το *help UI plug-in* μέσω του οποίου τα συστήματα βοήθειας διαφόρων plug-ins μπορούν να ενσωματώνονται
  - Για να προσφερθούν νέα help functions στον χρήστη, το *help UI plug-in* χρησιμοποιεί την πόρτα επέκτασης των actions sets
  - Ένα help plug-in θα επεκτείνει το workbench UI plug-in με τα κατάλληλα έξτρα menu items όπως «*Help → Help Contents*» και «*Search → Help*»

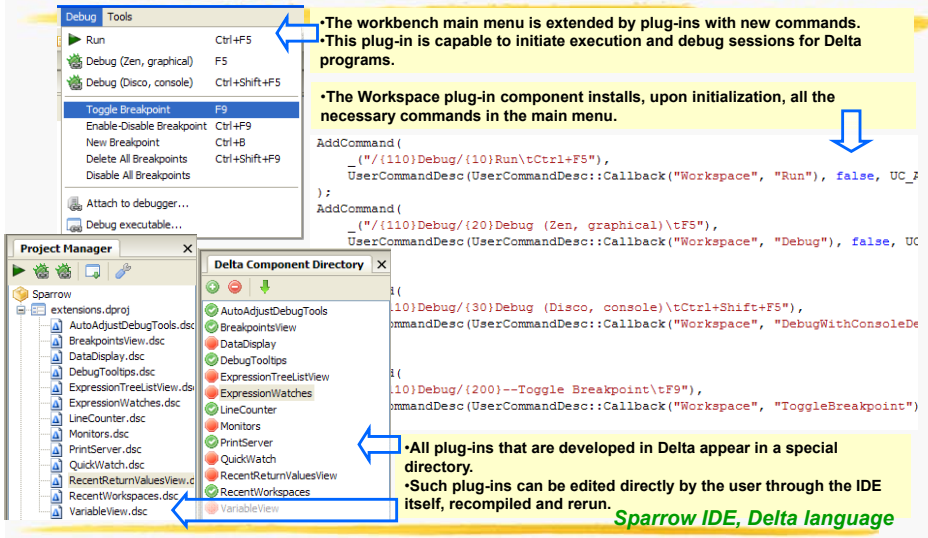
Plug-in architecture (8/9)



Source control (version control) plug-in architecture for Microsoft Visual Studio.

- **To Source Control Adapter Package μεταφράζει μία εντολή του χρήστη για source control σε μία κλήση συνάρτησης που υποστηρίζεται από το source control plug-in.**
- **To Vendor UI είναι η ειδική διεπαφή που μπορεί να παρέχει ο κατασκευαστής του plug-in.**

Plug-in architecture (9/9)



- **The workbench main menu is extended by plug-ins with new commands.**
- **This plug-in is capable to initiate execution and debug sessions for Delta programs.**
- **The Workspace plug-in component installs, upon initialization, all the necessary commands in the main menu.**
- **All plug-ins that are developed in Delta appear in a special directory.**
- **Such plug-ins can be edited directly by the user through the IDE itself, recompiled and rerun.**

Sparrow IDE, Delta language

Περιεχόμενα

- Ορισμός
- Ρόλος στην σχεδίαση
- Γρήγορος προσδιορισμός
- Επίπεδα αρχιτεκτονικής
- Βασικά αρχιτεκτονικά μοντέλα
- *Στοιχεία αρχιτεκτονικής σχεδίασης*

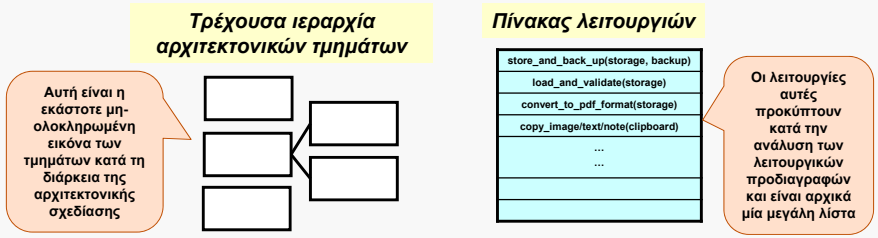
Στοιχεία αρχιτεκτονικής σχεδίασης (1/7)

- Δεν υπάρχει λεπτομερές «συνταγολόγιο» καλής αρχιτεκτονικής σχεδίασης.
- Οι λειτουργίες που συγκεντρώνονται σε κάθε τμήμα πρέπει να σχετίζονται ως προς τα παρακάτω?
  1. functional role – *λειτουργικό ρόλο*
     ΚΥΡΙΟΤΕΡΟΣ ΠΑΡΑΓΩΝ
  2. algorithmic properties – *αλγοριθμικά χαρακτηριστικά*
     ΟΧΙ
  3. data categories accessed – *πρόσβαση δεδομένων*
     ΟΧΙ
  4. resources utilization – *χρησιμοποίηση πηγών*
     ΟΧΙ
  5. performance requirements – *απαιτήσεις δυνατοτήτων*
     ΟΧΙ
  6. implementation language – *γλώσσα υλοποίησης*
     ΑΝ ΕΠΙΒΑΛΛΕΤΑΙ ΛΟΓΩ ΥΛΟΠΟΙΗΣΗΣ
  7. operational criticality – *λειτουργική κρισιμότητα*
     ΕΝΔΕΧΕΤΑΙ

CSD

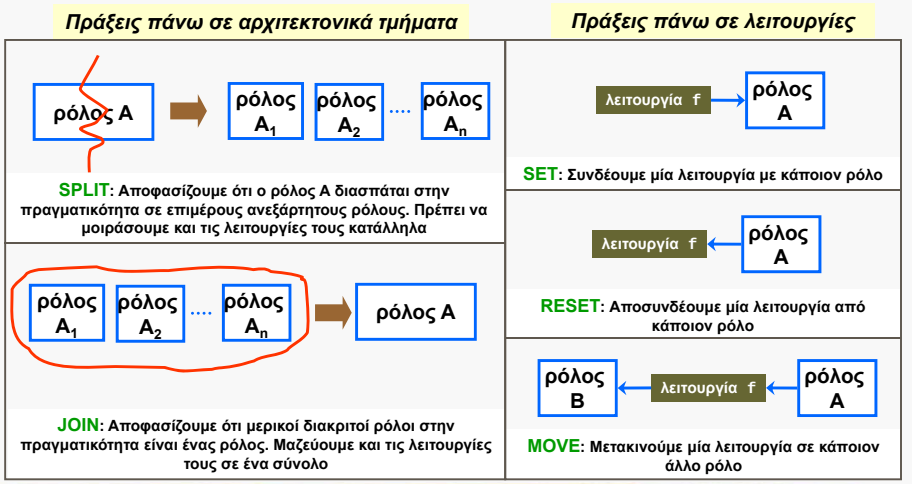
Στοιχεία αρχιτεκτονικής σχεδίασης (2/7)

- *Λειτουργικός ρόλος*
  - Η ευθύνη που φέρει έναν τμήμα για να επιτελέσει κάποιο έργο στο πλαίσιο του συστήματος
  - Όσες λειτουργίες σχετίζονται με τον ίδιο ρόλο εντάσσονται στο ίδιο τμήμα



CSD

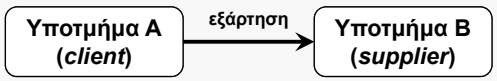
Στοιχεία αρχιτεκτονικής σχεδίασης (3/7)



CSD

Στοιχεία αρχιτεκτονικής σχεδίασης (4/7)

- Στην ***υλοποίηση σε κάποιο αρχιτεκτονικό τμήμα*** είναι σύνηθες να γίνεται χρήση APIs που παρέχονται από άλλα τμήματα:
  - π.χ., κλήση συναρτήσεων, χρήση κλάσεων, τύπων δεδομένων, σταθερών, κλπ.
- Κάθε τέτοια χρήση ορίζει μία ***σχέση εξάρτησης*** του τμήματος που χρησιμοποιεί το API – client - και του τμήματος που το παρέχει - supplier.
- Στόχος μας είναι τα στοιχεία που χρησιμοποιούνται και οδηγούν σε εξαρτήσεις να αντέχουν όσο το δυνατόν περισσότερο σε αλλαγές του supplier. Αυτό επιτυγχάνεται με τεχνικές αφαίρεσης.



CSD

Στοιχεία αρχιτεκτονικής σχεδίασης (5/7)

- Η διαδικασία της αρχιτεκτονικής σχεδίασης βασίζεται πάντοτε σε ***σενάρια λειτουργίας*** τα οποία γεννώνται από τις λειτουργικές απαιτήσεις
  - πρέπει ***να σκεφτείτε όσο περισσότερα σενάρια μπορείτε*** καθώς βάσει αυτών σχεδιάζεται αυξητικά η αρχιτεκτονική
  - ***κάθε σενάριο δοκιμάζει την αντοχή της αρχιτεκτονικής*** προσθέτοντας νέους ρόλους και εξαρτήσεις ή διασπώντας κάποιους ρόλους σε επιμέρους ρόλους
  - όταν τελειώνουμε με ένα σενάριο βελτιστοποιούμε προσεκτικά
    - όλα τα τμήματα αντιπροσωπεύουν όντως διακριτούς ρόλους ή το έχουμε παρακάνει - *υπερβολικά πολλά τμήματα*
    - μήπως κάποιος ρόλος είναι πολύ σύνθετος και πρέπει να αναλυθεί περισσότερο - *υπερβολικά λίγα τμήματα*

CSD

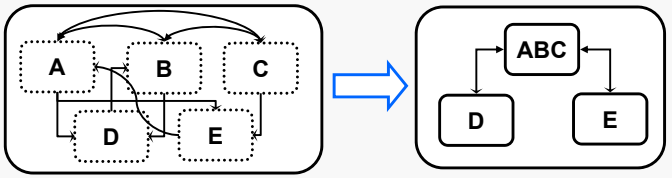
Στοιχεία αρχιτεκτονικής σχεδίασης (6/7)

- Κατά την ανάθεση λειτουργικών ρόλων και την αρχική σχεδίαση των σχέσεων χρήσης μεταξύ των τμημάτων θα πρέπει να εξασφαλίσετε ότι ο *γράφος που προκύπτει με την προηγούμενη σχέση εξάρτησης δεν περιέχει κύκλους*
  - **Εάν έχετε κύκλους, τότε πρέπει να τροποποιήσετε την οργάνωση, τον τεμαχισμό και την λειτουργική θεώρηση των τμημάτων**
- Κατά την διαδικασία υλοποίησης δεν πρέπει να γίνεται παράβαση των σχέσεων εξάρτησης
  - **εάν κάτι τέτοιο είναι αναπόφευκτο, πρέπει να συγκρίνετε το κόστος της αρχιτεκτονικής τροποποίησης με τον κίνδυνο κυκλικών εξαρτήσεων στην υλοποίηση**
    - ***code-change domino effect***
    - **πολύς χρόνος compilation (για large scale systems)**
- ➔ *Προσοχή: ο γράφος εξάρτησης τμημάτων και η αρχιτεκτονική συνδεσμολογία τμημάτων έχουν πολλά κοινά, αλλά δεν είναι το ίδιο πράγμα, ενώ δεν έχουν καμία σχέση με τη συνδεσμολογία επικοινωνίας*

CSD

Στοιχεία αρχιτεκτονικής σχεδίασης (7/7)

- Αντίστροφα, με την επιθεώρηση των σχέσεων κλήσεων εσωτερικά στα τμήματα μπορεί να προκύψει επιπλέον αρχιτεκτονικός τεμαχισμός
  - *Οι ομαδοποιημένες εξαρτήσεις μίας κατεύθυνσης προμηνύουν πιθανά αρχιτεκτονικά σύνορα*



Άτακτη οργάνωση του κώδικα χωρίς τεμαχισμό σε ανεξάρτητα packages / projects

Παγίωση micro-architecture και οργάνωση του κώδικα σε ανεξάρτητα packages / projects