

Κεφάλαιο Δεκαπέντε

-

Ανάπτυξη Κατανομή και Απόκτηση Συστατικών

15.1 Η Μεθοδολογία - Συστατικοστρεφής Προγραμματισμός

- Ο συστατικοστρεφής προγραμματισμός χειρίζεται τις προοπτικές του προγραμματισμού με συστατικά, για τον οποίο χρησιμοποιείται ο όρος COP (Component Oriented Programming).
- Τα στοιχεία υποστήριξης του είναι:
 - Πολυμορφισμός (ικανότητα αναπλήρωσης).
 - Ενσωμάτωση διαμορφωτών (απόκρυψη πληροφοριών σε υψηλά επίπεδα).
 - Αργή σύνδεση και φόρτωση (ανεξάρτητη ανάπτυξη).
 - Ασφάλεια (τύπων και διαμορφωτών).
- Ιδανική μεθολογία δεν βρέθηκε ακόμη και οι υπάρχουσες λειτουργούν μέσα στο συστατικό. Υπάρχει αδυναμία στις πολύπλοκες αλληλεπιδράσεις με άλλα συστατικά. Η επιλογή αρχιτεκτονικής επιρρεάζει σε ορισμένο βαθμό την ανάπτυξη των συστατικών και των υποδομών τους. Υπάρχουν ακόμη πολλά προβλήματα για επίλυση.

15.1.1 Προβλήματα Ασυγχρονισμού

- Η συναρμολόγηση συστατικών γίνεται με προώθηση συμβάντων. Κάθε προώθηση πυροδοτείται με την αλλαγή κατάστασης ενός συστατικού η οποία μπορεί να ενδιαφέρει άλλα αντικείμενα. Τα συστατικά που είναι εγγεγραμμένα για ένα συμβάν το λαμβάνουν μέσω ενός μηχανισμού διανομής συμβάντων. Τα προβλήματα αυτού του μηχανισμού:
 - Η διανομή συμβάντων γίνεται με μετάδοση σε πολλούς παραλήπτες. Ενώ διεξάγεται η μετάδοση το σύστημα βρίσκεται σε αστάθεια.
 - Οι παραλήπτες συμβάντων μπορούν να στείλουν και οι ίδιοι άλλα συμβάντα χωρίς να έχουν αντιληφθεί την αλλαγή κατάστασης άλλων συστατικών που τους επιρρεάζουν. Υπάρχει δηλαδή πρόβλημα αιτιότητας στη σειρά μετάδοσης και παραλαβής των συμβάντων.
 - Η αποστολή συμβάντων μπορεί να γίνεται σε δυναμικά σύνολα παραληπτών που από την στιγμή της μετάδοσης μέχρι την παραλαβή ίσως και να αλλάξουν.
 - Μερικοί παραλήπτες ίσως να δηλώσουν εξαιρέσεις ενώ χειρίζονται παραληφθέντα αντικείμενα συμβάντων και ενώ η μετάδοση συνεχίζεται.
- Χρειάζεται προσεκτικός ορισμός της συμπεριφοράς του συστήματος για την λύση των πιο πάνω προβλημάτων.

15.1.2 Πολλαπλά Νήματα

- Ο όρος 'πολλαπλά νήματα' (multithreading) περιλαμβάνει την υποστήριξη συγχρονισμού σε περιπτώσεις που μοιράζονται τον ίδιο χώρο κατάστασης. Η πολυπλοκότητα έναντι των σειριακών προγραμμάτων αυξάνει σημαντικά. Υπάρχουν αντιθέσεις σε ταυτόχρονα reads και writes σε μεταβλητές. Για συγχρονισμό χρησιμοποιούνται τεχνικές κλειδώματος αλλά και πάλι υπάρχει πρόβλημα αδιεξόδου όταν το κλείδωμα είναι πολύ συντηρητικό ή με λάθος ακολουθία.
- Στη χρησιμοποίηση πολλαπλών νημάτων υπάρχει καλύτερη κατανομή απόδοσης αν και η ολική απόδοση μεγιστοποιείται με την προτίμηση για εξυπηρέτηση κλήσεων με τον ελάχιστο χρόνο εκτέλεσης. Με συγχρονισμό ακόμα και χωρίς αδιέξοδα η απόδοση μπορεί να μειωθεί.
 - Το παρατεινόμενο κλείδωμα συχνά χρησιμοποιούμενων κοινών πόρων πρέπει να αποφεύγεται.
 - Η προώθηση εξαιρέσεων διά μέσου ορίων νημάτων δημιουργεί πρόβλημα χειρισμού ασύγχρονων εξαιρέσεων.
 - Η διόρθωση λαθών κώδικα είναι εξαιρετικά δύσκολη όταν αυτός χρησιμοποιεί πολλαπλά νήματα και πολύπλοκα σχήματα κλειδώματος.
- Στο πλαίσιο *προγραμματισμού μεταβολών* (transactional programming) μπορεί να παρατηρηθεί ελάττωση της πολυπλοκότητας. Αυτό λειτουργεί στη βάση απόκτησης ικανού αριθμού ασφαλειών από κάθε μεταβολή, που στην αντίθετη περίπτωση αποβάλλει αντί να παραμένει σε αδιέξοδο.

15.1.3 Εκμάθηση από το Σχεδιασμό Κυκλώματος

- Χρησιμοποίηση προτύπων της ηλεκτρονικής: Αντί οι υπολογισμοί να οδηγούνται σαν αποτέλεσμα κλήσεων, οι κλήσεις μπαίνουν σε ουρά και προωθούνται σε σειρά που αποφασίζεται από το σχεδιασμό του συστήματος παρά τη εμφάνιση κλήσεων. Έτσι υπάρχει χρησιμοποίηση διεργασιών και ασύγχρονης επικοινωνίας μεταξύ τους, παρά νήματα με παρενέργειες συγχρονισμού (σύστημα πραγματικού χρόνου με προσέγγιση συγχρονισμού είναι το Esterel).
- Οι υπολογισμοί μπορούν να διαχωρισθούν σε ατομικές εκτελέσεις ή πράξεις (atomic actions), που πυροδοτούνται από ακολουθία συμβάντων του συστήματος. Υπάρχει πάντα μία πράξη που εκτελείται σε κάθε διεργασία. Συνθήκες μπορούν να υπάρχουν για τη διαθεσιμότητα των εκτελέσεων αυτών, και βάση αυτού μπορεί να αποφασισθεί η σειρά τους. Η επικοινωνία τους βασίζεται σε παρενέργειες από καταστάσεις συστατικών.

15.1.4 Αποφυγή Διαδοχής Εφαρμογών

- Η διαδοχή εφαρμογών ανάμεσα σε συστατικά καταλήγει σε προβλήματα και οδηγεί στη χρησιμοποίηση απλής σύνθεσης αντικειμένου και προώθησης μνημάτων, λύσεις που ωστόσο παρουσιάζουν αδυναμίες σε μικρές προσαρμογές. Είναι απλό να δημιουργηθούν υποτάξεις με δεκάδες μεθόδους και να επανεγραφούν μόνο λίγες από αυτές. Αντιθέτως είναι δύσκολο να δημιουργηθούν νέοι υποδοχείς τάξεων που απλά προωθούν μερικές κλήσεις μεθόδων.
- Όταν οι μέθοδοι ενός αντικειμένου σχηματίζουν μέτρια σύνολα διασυνδέσεων, η συγκρότηση τύπου COM βοηθά στην αποφυγή συνεπειών της προώθησης στην απόδοση.
- Μια λύση είναι ο κώδικας για μία τάξη να βασίζεται στις διασυνδέσεις των αντικειμένων που θα προωθηθούν. Το κύριο μειονέκτημα είναι ότι καθώς ο δημιουργηθείς κώδικας πρέπει να εκδοθεί, οι αλλαγές στις ζητούμενες διασυνδέσεις αντικειμένου απαιτούν επαναδημιουργία και επανέκδοση ή διά χειρός διόρθωση του κώδικα που υπάρχει.
- Άλλη λύση είναι η χρήση ενός μηχανισμού *προτύπου* (templates) όπως της C++, που γεννά τον απαιτούμενο κώδικα τη στιγμή της μεταγλώττισης. Αντί να εκδίδεται ο παραγόμενος κώδικας παραμετροποιούνται τα πρότυπα. Ο τελικός κώδικας τότε παράγεται από το μεταγλωττιστή βάση των παραμέτρων του προτύπου. Τα μειονεκτήματα είναι:
 - Δεν μπορεί να γίνει έλεγχος τύπων του προτύπου και ο μεταγλωττιστής μπορεί να αναφέρει λάθη όταν προωθεί ενάρξεις τιμών του προτύπου.
 - Ένα πρότυπο δεν μπορεί να μεταγλωττισθεί ξεχωριστά και αυτό οδηγεί σε εξογκωμένο κώδικα και αδυναμία σύνδεσης εννοιών του προτύπου.

15.1.5 Η Τάξη-Κελύφος

- Η τρίτη λύση είναι η χρησιμοποίηση διαδοχής εφαρμογών, δεδομένου ότι οι τάξεις εκδίδονται με πλήρη μορφή και παραμένουν αναλλοίωτες. Για διασυνδέσεις συστατικών που αναλαμβάνουν την προώθηση αντικειμένων, μπορεί να υπάρξει μία *τάξη-κελύφος* (nutshell-class) που απλοποιεί τον προγραμματισμό τους. Αυτή έχει την ίδια διασύνδεση όπως το αντικείμενο που προωθεί και όλες οι μέθοδοι εφαρμόζονται σαν απλές ωθήσεις στο ζητούμενο αντικείμενο. Και αυτές οι τάξεις είναι ιδεατές, παρόλο που όλες οι μέθοδοι εφαρμόζονται. Με σκοπό την δημιουργία προωθητή που παρεμβάλλεται με κλήσεις μεθόδων, μπορούν να γίνουν υποτάξεις.

15.1.6 Υποστήριξη Γλώσσας

- Η τέταρτη και προτιμότερη θα ήταν η υποστήριξη γλώσσας. Όταν η δημιουργία μίας προωθούμενης γλώσσας υποστηρίζεται απ' ευθείας από την γλώσσα προγραμματισμού, όλα τα μειονεκτήματα που είδαμε ως τώρα μπορούν να αποφευχθούν.
 - Στη C++ ο μηχανισμός εικονικών βασικών τάξεων δεν επιτρέπει το μοίρασμα βασικών τάξεων αντικειμένων σε διαφορετικά αντικείμενα.
 - Επίσης δεν επιτρέπει δυναμικές αλλαγές βασικών τάξεων αντικειμένων, ή ξεχωριστές υποτάξεις εικονικών βασικών τάξεων.
 - Μια γλώσσα που υποστηρίζει δυναμική διαδοχή από ένα αντικείμενο είναι η Objective-C.

15.1.7 Δυναμικά Βασικά Αντικείμενα με Σημασιολογία Προώθησης

- Μία υποθετική επέκταση της Component Pascal θα εισήγαγε τον απαιτούμενο μηχανισμό: εκεί που η Component Pascal ορίζει βασικούς τύπους μίας καταχώρησης, θα μπορούσε να εισαχθεί ένας βασικός δείκτης. Ακολουθεί το παράδειγμα μίας διασύνδεσης view:

TYPE

```
View=POINTER TO ABSTRACT RECORD
  (v: View) Restore, NEW, ABSTRACT;
  (*many more methods*)
END;
```

- Στην Component Pascal, ο τύπος αντικειμένων παρουσίασης αντικειμένων παρουσίασης κειμένου που κληρονομεί διασύνδεση και εφαρμογή από το View θα είναι:

TYPE

```
TextView=POINTER TO RECORD (View)
  (v: TextView) Restore;
  (* implement other View methods *)
  (v: TextView) ThisText(): TextModel;
  (* other text view specific methods *)
END;
```

- Οι μηχανισμοί προώθησης μπορούν να εισαχθούν με τροποποιήσεις των βασικών τύπων. Το ακόλουθο παράδειγμα θεωρεί τους τύπους αντικειμένων που δέχονται τις μεθόδους View, αλλά ανακόπτουν τη μέθοδο Restore.

Δυναμικά Βασικά Αντικείμενα με Σημασιολογία Προώθησης

TYPE

```
Decorator=POINTER TO ABSTRACT RECORD (v:View)
```

```
(d: Decorator) Restore;
```

```
(*other View method invocations are forwarded to base object v *)
```

```
(d: Decorator) GetProperties (...), NEW, ABSTRACT;
```

```
(d: Decorator) SetProperties (...), NEW, ABSTRACT;
```

```
END;
```

```
PROCEDURE (d: Decorator) Restore;
```

```
BEGIN
```

```
d.v.Restore; (*forward to base object: restore it first *)
... (*draw decoration *)
```

```
END Restore;
```

- Το αντικείμενο `Decorator` διακοσμεί την παρουσίαση βάση ιδιοτήτων που εκδίδονται. Καθώς χρησιμοποιεί δυναμικές αναφορές στο αντικείμενο `View` μπορεί να προστεθεί σε οποιοδήποτε αντικείμενο παρουσίασης – και αυτά που έχουν ήδη διακοσμηθεί όπως το `TextView`.
- Συντακτικά η επέκταση εισάγει ένα όνομα πεδίου, που προαιρετικά τίθεται μπροστα από το όνομα του βασικού τύπου. Σημασιολογικά αν μία μέθοδος μείνει ανεφάρμοστη σε περιέχον αντικείμενο, οι κλήσεις προωθούνται στο βασικό αντικείμενο.
- Τα πεδία και οι μέθοδοι που κατέχει το `View` έχουν διαδοχή στο `Decorator`. Επίσης στις υπερκλήσεις από το `Decorator` η κληθείσα μέθοδος ανήκει στο αντικείμενό του και όχι στο `view`. Με αυτή την πρόταση το αντικείμενο `Decorator` μπορεί να περάσει σε ένα πλαίσιο που περιμένει ένα αντικείμενο `View` χωρίς να παρακάμπτει τον διακοσμητή.
- Στη `Component Pascal`, η κατάσταση εξαγωγής ενός πεδίου, μπορεί να οδηγηθεί να είναι ιδιωτική σε διαμορφωτή, εξαχθείσα `read-only` ή πλήρως εξαχθείσα. Οι ίδιοι τρεις τύποι εξαγωγής είναι πιθανοί για το πεδίο βασικού αντικειμένου. Αν το πεδίο δεν εξαχθεί τότε οι πελάτες δεν μπορούν να ξέρουν αν αυτό το αντικείμενο χρησιμοποιεί το βασικό αντικείμενο. Το μόνο που μπορούν να δουν είναι ότι τύπος έχει ένα υπερτύπο. Αν το πεδίο εξαχθεί σαν `read-only`, τότε το βασικό αντικείμενο δίδει απλά πρόσβαση χωρίς δικαίωμα αντικατάστασης. Αυτό μπορεί να χρησιμοποιηθεί για ύπαρξη σταθερών βασικών αντικειμένων που ρυθμίζονται την στιγμή της δημιουργίας τους και μένουν αναλλοίωτα. Τέλος αν τα πεδία γίνουν πλήρως εξαχθέντα τότε υπάρχει δικαίωμα αλλαγής.

Δυναμικά Βασικά Αντικείμενα με Σημασιολογία Προώθησης

- Στο παράδειγμα ένας πελάτης μπορεί να πάρει ένα διακοσμητή έξω από την τρέχουσα παρουσίασή του και να τον εισάξει σε μία νέα παρουσίαση (view):

TYPE

```
DecoratorRewrite=POINTER TO ABSTRACT RECORD (View)
```

```
v:View;
```

```
(d: Decorator) Restore;
```

```
(*Other view method invocations are forwarded to base object v*)
```

```
(d: Decorator) GetProperties (...),NEW, ABSTRACT;
```

```
(d: Decorator) SetProperties (...),NEW, ABSTRACT
```

```
END;
```

```
(*methods explicitly handled are not changed:*)
```

```
PROCEDURE (D: DECORATORREWRITE) Restore;
```

```
BEGIN
```

```
  d.v.Restore; (*forward to base object:restore it first *)
```

```
  ... (* draw decoration *)
```

```
END Restore;
```

```
(*rewrite all View methods that are not overwritten in Decorator:*)
```

```
PROCEDURE (d: DecoratorRewrite) Method (...);
```

```
BEGIN
```

```
  d.v.Method(...) (*forward to base object*)
```

```
END Method;
```

- Ο κανόνας επανεγγραφής (*rewriting*) μπορεί να ορίσει τον μηχανισμό προώθησης.

15.1.8 Η Συμπερίληψη του Καλούντος κώδικα

- Η υποστήριξη από γλώσσα ωφελεί και τους ορισμούς διασυνδέσεων συστατικών.
 - Εσωτερικός κωδικας του συστατικού μπορεί να ενεργοποιεί διεργασίες που εφαρμόζουν μία διασύνδεση (*outgoing interface*). Μόνο η Component Pascal υποστηρίζει πλήρως αυτό το πρότυπο.
 - Εξωτερικός κώδικας προκαλεί τις διεργασίες μίας διασύνδεσης (*incoming interface*).
- Σαν παράδειγμα η βασική τάξη Object εισάγει την `Object.finilize` μέθοδο για συλλογή αχρήστων η οποία είναι προστατευόμενη σαν 'outgoing' και μπορεί να ενεργοποιηθεί με κώδικα από την εισάγουσα τάξη ή μία υποτάξη της ή τα πακέτα που τις περιέχουν. Επιπλέον μία υποτάξη μπορεί να επαναορίσει μία προστατευμένη μέθοδο σαν `public` δίδοντας πρόσβαση σε όλες τις άμεσες και έμμεσες περιπτώσεις υποτάξεων. Άλλα Java αντικείμενα μπορούν απλά να την εφαρμόσουν σαν 'incoming' όπως τα `FileOutputStream` για αποδέσμευση εξωτερικών πηγών στις οποίες ο συλλέκτης δεν έχει πρόσβαση.
- Οι ιδιότητες των προστατευμένων μεθόδων εμποδίζουν τις περισσότερες λανθασμένες κλήσεις, όχι όμως στο βαθμό που να επιτρέπει στο βασικό πακέτο να υποστηρίξει ένα αμετάβλητο κανόνα που να απαγορεύει σε εξωτερικό για το πακέτο κώδικα να ενεργοποιεί τέτοιες μεθόδους. Ο μόνος τρόπος για γλώσσες όπως η Java και C++ είναι να χαρακτηρισθούν αυτές οι μέθοδοι σαν *private*, κάτι που δεν θα επέτρεπε την εξωτερική εφαρμογή της.
- Υπάρχει ανάγκη για *συμπερίληψη* (*encapsulation*) δηλ. καθορισμού ορίων του καλούντος μία μέθοδο κώδικα. Στη Component Pascal μία μέθοδος μπορεί να χαρακτηρισθεί 'implement only' και ακολούθως μπορεί να επανεγγραφεί έξω από τον ορίζοντα διαμορφωτή σαν κανονικά εξαχθείσα μέθοδος. Ο μηχανισμός προστασίας βρίσκεται στο επίπεδο των διαμορφωτών. Οι κλήσεις μπορούν να έρθουν απο 'φιλικές' τάξεις ή διεργασίες του ίδιου διαμορφωτή. Η `Object.finilize` στην Component Pascal ορίζεται σαν

TYPE

ANYPTR=POINTER TO ANYREC;

ANYREC=ABSTRACT RECORD (A: ANYPTR) FINILIZE-, NEW, EMPTY;

END;

Η Συμπερίληψη του Καλούντος Κώδικα

- Ο τύπος ANYREC είναι έμμεσος βασικός όλων των τύπων καταχωριτών στην Component Pascal, παρόμοια με τη τάξη Object σαν έμμεση βασική τάξη όλων των τάξεων στη Java. Οι περιορισμοί στην πρόσβαση γίνονται με *ενδείξεις εξαγωγής* (export marks) που ακολουθούν κάθε νέο ενδείκτη. Υπάρχουν μόνο δύο ενδείκτες εξαγωγής: '*' και '-'. Ένας ενδείκτης με σήμα * εξάγεται με τον ορισμό του, από το διαμορφωτή που τον ορίζει. Ένας ενδείκτης με σήμα - εξάγεται σαν read-only.
- Ο τύπος ANYREC ορίζεται ευρέως, δηλαδή είναι μέρος της γλώσσας και ορισμένος έξω από κάθε κανονικό διαμορφωτή. Έτσι η μέθοδος FINALIZE μπορεί να κληθεί μόνο από το συλλέκτη του συστήματος.
- Η συμπερίληψη του καλούντος τυγχάνει καλής χρήσης σε ορισμένα σημεία της BlackBox. Για παράδειγμα σημαντικές μέθοδοι παρουσίασης (view) μπορούν να κλειθούν μόνο έμμεσα από την υποδομή. Αν η υποδομή έχει πάρει εξαίρεση από προηγούμενη κλήση της μεθόδου στην ίδια παρουσίαση, τότε σταματά να προωθεί τέτοιες κλήσεις. Οι ελαττωματικές παρουσιάσεις έτσι φθίνουν χωρίς να γίνονται τελείως άχρηστες, ή να ενοχλούν το σύστημα. Η BlackBox είναι από τις λίγες υποδομές στις οποίες η παρουσίαση ενσωματώνεται σε σύνθετο κείμενο χωρίς να το θέτει σε ευρύ κίνδυνο. Για παράδειγμα μία παρουσίαση με ελαττωματικό Restore θα επικαλυφθεί από την υποδομή με γκρίζα περιοχή σαρώσεως και οι κλήσεις στο Restore θα σταματήσουν. Εν τούτοις επειδή τα χειριστήρια της παρουσίασης ακόμη δουλεύουν αυτό μπορεί να φυλαχθεί και να επαναφορτισθεί με νέα παρουσίαση, όταν το εφαρμόζον συστατικό της μεθόδου έχει διορθωθεί.
- Ακόμη ένα παράδειγμα συμπερίληψης καλούντος κώδικα από την BlackBox είναι αυτή με τις μεθόδους Externalize και Internalize ενός αντικειμένου για απόκρυψή του απο λανθασμένες κλήσεις.

15.2 Το Περιβάλλον - Επιλογή Υποδομής

- Ένα αντικείμενο συστατικού δεν μπορεί να λειτουργήσει έξω από ένα καθορισμένο περιβάλλον. Εν τούτοις ένα αντικείμενο μπορεί να σχεδιασθεί για λειτουργία σε πολλαπλά περιβάλλοντα ταυτόχρονα. Ανάλογα με την αρχιτεκτονική του συστήματος, οι υποδομές μπορούν να χωρισθούν σύμφωνα με διάφορους ρόλους, όπου κάθε υποδομή μπορεί να αναλαμβάνει ένα συγκεκριμένο μηχανισμό που λειτουργεί μεταξύ των συστατικών. Η υποδομή κατανομής μπορεί να είναι υπεύθυνη για την κατανομή των περιπτώσεων των συστατικών στις μηχανές. Άλλη υποδομή μπορεί να είναι υπεύθυνη για την ολοκλήρωση του σύνθετου κειμένου. Ένα συστατικό μπορεί να έρχεται σε επαφή και με τις δύο υποδομές για εφαρμογή των αντικειμένων που μπορεί και να είναι κατανεμημένα και να λειτουργούν σε σύνθετο κείμενο.
- Η ανάπτυξη υποδομών συστατικών δεν συμβαδίζει με την ολοκλήρωση της αρχιτεκτονικής συστατικών. Η συμβατότητα διαφορετικών υποδομών είναι σχεδόν ανύπαρκτη καθώς σχεδιάζονται απομονωμένα και επιμένουν σε πλήρη έλεγχο.

15.3 Επιλογή Γλώσσας Προγραμματισμού

- Θεωρητικά ο προγραμματισμός συστατικών μπορεί να κάνει χρήση οποιασδήποτε γλώσσας. Προϋποθέσεις:
 - Πολυμορφικός χειρισμός άλλων συστατικών.
 - Αργή σύνδεση (για δυναμικές αλληλεπιδράσεις με άλλα συστατικά).
 - Συμπερίληψη και ασφάλεια τύπων για παραμέτρους ασφάλειας - ασφάλεια διαμορφωτών.
 - Αντικειμενοστρεφής άλλα και συναρτησιακός προγραμματισμός.
- Πολλές γλώσσες δεν υποστηρίζουν συμπερίληψη, πολυμορφισμό, ή ασφάλεια τύπων και διαμορφωτών. Τώρα η δεσπόζουσα συστατικοστρεφής γλώσσα είναι η Java, παρόλο που έχει κάποιες αδυναμίες στην ασφάλεια διαμορφωτών. Υποστηρίζει ασφάλεια πρόσβασης στο επίπεδο πακέτων. Οι νέες τάξεις όμως που εισέρχονται στο πακέτο αποκτούν πρόσβαση στους μηχανισμούς ασφαλείας του. Άλλες συστατικοστρεφείς γλώσσες:
 - Modula-3.
 - Oberon.
 - Component Pascal που υποστηρίζει ασφάλεια πρόσβασης στο επίπεδο διαμορφωτών.
 - Ada 95.

Επιλογή Γλώσσας Προγραμματισμού

- Στο πλαίσιο της δυναμικής απόκτησης αρχείων τάξεων από απόμακρους εξυπηρετητές, υπάρχει περισσότερη πολυπλοκότητα. Αυτά που ανήκουν στο ίδιο πακέτο θα πρέπει να εξακριβωθεί ότι προέρχονται από την ίδια μεταγλώττιση. Για να επιτρέψει περισσότερο ανοικτές ρυθμίσεις, η Java μπορεί να ωφεληθεί από μία κλειστή δομή διαμορφωτών όπου τέτοιοι διαμορφωτές αντιστοιχούν προς ένα μεταγλωττισμένο αρχείο για κατανομή.

15.4 Κατανομή και Απόκτηση Συστατικών

- Εξ' ορισμού τα συστατικά είναι μονάδες ανάπτυξης. Η ύπαρξή τους σχετίζεται με την ολοκλήρωση προϊόντων από ανεξάρτητους πωλητές για κάποιο κοινό στόχο. Οι δύο τεχνικές πλευρές του θέματος είναι η κατανομή και η απόκτηση συστατικών.
- Το πρότυπο του Java applet είναι ένα παράδειγμα μίας εντελώς χαοτικής προσέγγισης. Η κατανομή χρειάζεται υποστήριξη από τεχνικές αγοράς και η απόκτηση καλύτερους τρόπους εύρεσης.
- Η κατάσταση είναι διαφορετική για την κερδοφόρα αγορά χειριστηρίων της Visual Basic και τώρα της ActiveX. Χιλιάδες τέτοια χειριστήρια κατανέμονται επιτυχώς στην αγορά. Η απόκτηση υποστηρίζεται από καταλόγους των προς διάθεση χειριστηρίων αλλά αρκετά ακόμη αποκτούνται με τυχαία εύρεση.
- Τεχνικά αυτό που χρειάζεται είναι οι χαρακτηρισμοί της λειτουργικότητας των συστατικών και των προϋποθέσεων της πλατφόρμας στην οποία θα τρέξουν, να ταξινομηθούν σε καταλόγους. Σ' αυτό το θέμα γίνεται ακόμη έρευνα. Επιπλέον πρέπει να αποφευχθεί η εισαγωγή στην αγορά ελαφρών μικροσκοπικών συστατικών.
- Το επόμενο πρόβλημα βρίσκεται στις υποδομές συστατικών. Η συνένωση συστατικών απαιτεί την προσεκτική επιλογή της κατάλληλης υποδομής. Οι εξειδικευμένες υποδομές θα μπορούσαν να είναι ένα αποτελεσματικό κριτήριο και για την επιλογή των κατάλληλων συστατικών που δουλεύουν με την αντίστοιχη υποδομή. Η τοποθέτηση των συστατικών και των πωλητών τους βασιζόμενοι στις προϋποθέσεις τους είναι επίσης πρόβλημα.
 - Μια προσέγγιση με δυναμική είναι η οντολογική, όπου μία οντότητα καλύπτει την γνώση των οντοτήτων ενός σύμπαντος (βασισμένη σε σχήμα ανταλλαγής γνώσεων). Αυτό έχει γίνει στο Stanford από την επιτροπή ANSI X3T2.
 - Μια οντολογική αποθήκη για αντικειμενοστρεφή συστατικά έχει γίνει από την κοινοπραξία Corinto (από την IBM, Apple και Selfin).
 - Μια μελέτη για αποθήκες συστατικών CORBA έχει γίνει επίσης στο Stanford.
- Άλλο θέμα είναι η τεχνική υποδομή που χρειάζεται για ηλεκτρονικές διόδους κατανομής. Καθώς τα συστατικά μπορούν να μεταφέρονται σαν μέρη συναρμολογήσεων, πρέπει να υπάρξει μηχανισμός αδειών που να διακρίνει διαφορετικές μορφές χρήσης.
- Στην πληρωμή ανά χρήση τα τέλη στις πλήστες των περιπτώσεων είναι χαμηλά. Υπάρχουν επίσης προμηθευτές υπηρεσιών που χρεώνουν μία σταθερή συνδρομή και πληρώνουν οι ίδιοι τέλη χρήσης στους προμηθευτές.