ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΠΟΛΥΠΡΑΚΤΟΡΙΚΟ ΠΕΡΙΒΑΛΛΟΝ ΜΕΤΑΤΕΜ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΚΡΟΜΑΣΤΟΡΑ ΠΑΡΑΣΚΕΥΗ

Βόλος 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΠΟΛΥΠΡΑΚΤΟΡΙΚΟ ΠΕΡΙΒΑΛΛΟΝ ΜΕΤΑΤΕΜ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΚΡΟΜΑΣΤΟΡΑ ΠΑΡΑΣΚΕΥΗ

Επιβλέπουσα : Δασκαλοπούλου Ασπασία

Επίκουρος Καθηγήτρια Τ.Μ.Η.Υ.Τ.Δ.

Συνεπιβλέπων : Αντωνόπουλος Χρήστος

Επίκουρος Καθηγητής Τ.Μ.Η.Υ.Τ.Δ.

Βόλος 2011

Περίληψη

Στην παρούσα διπλωματική εργασία θα ασχοληθούμε με την κατασκευή πολυπρακτορικού περιβάλλοντος όπου οι πράκτορες λειτουργούν με βάση τη σύγχρονη γλώσσα MetateM (Fisher 1994).

Πιο συγκεκριμένα, στο *πρώτο κεφάλαιο* θα εισάγουμε τον αναγνώστη στις βασικές έννοιες της γλώσσας MetateM και θα δοθεί μια γενική περιγραφή του αντικειμένου της παρούσας διπλωματικής. Στο *δεύτερο κεφάλαιο* θα γίνει εκτενέστερη μελέτη της γλώσσας MetateM και θα παρουσιάσουμε την υπολογιστική μονάδα ενός πράκτορα στη MetateM και παράδειγμα λειτουργίας πράκτορα. Στη συνέχεια, στο τρίτο κεφάλαιο θα γίνει περιγραφή της εφαρμογής που αναπτύχθηκε για τη συγκεκριμένη διπλωματική εργασία, ενώ το τέταρτο κεφάλαιο αναφέρεται στις επεκτάσεις που θα μπορούσαν να γίνουν στην εφαρμογή. Στο πέμπτο κεφάλαιο παρατίθεται η βιβλιογραφία ενώ στο τέλος παρατίθεται παράρτημα με τον κώδικα της εφαρμογή.

Πίνακας Περιεχομένων

Ευχαριστίες

Ολοκληρώνοντας την παρούσα διπλωματική εργασία που σηματοδοτεί και το τέλος των προπτυχιακών μου σπουδών, θα ήθελα να ευχαριστήσω θερμά την επιβλέπουσα καθηγήτριά μου κ. Ασπασία Δασκαλοπούλου για τη βοήθεια που μου προσέφερε στα πλαίσια της διπλωματικής μου εργασίας αλλά κυρίως για τη βοήθεια, την καθοδήγηση και τη στήριξη που μου προσέφερε τα χρόνια των σπουδών μου. Επίσης ευχαριστώ τον καθηγητή μου κ. Χρήστο Αντωνόπουλο για τη βοήθειά του και εκτός αυτής της διπλωματικής εργασίας.

Ευχαριστώ από καρδιάς τους γονείς μου, Αποστόλη και Χρυσάνθη, γιατί η πίστη τους σε μένα και η στήριξή τους με έκαναν να συνεχίσω και να παρακάμψω όλες τις δυσκολίες. Τους ευχαριστώ που πάντα υπάρχουν εκεί για μένα.

Ευχαριστώ την αδερφή μου Ειρήνη, που είναι από τους σημαντικότερους και πολυτιμότερους ανθρώπους στη ζωή μου.

Τέλος, θέλω να πω ένα μεγάλο ευχαριστώ στους φίλους μου και ιδιαίτερα στα «αδέρφια» που απέκτησα εδώ στο Βόλο, Νεφ Μαρί Ορελί, Λάκη Μόσχο και Φωτεινή Κατσαρού για τα υπέροχα χρόνια στο Βόλο και για την αληθινή και δυνατή φιλία που έχουμε .

Στη Μαρί και την Οικογένειά μου

1 Εισαγωγή

1.1 Εισαγωγή στη σύγχρονη/ συντρέχουσα (concurrent) γλώσσα MetateM (Fisher 1994)

Η παρούσα διπλωματική εργασία ασχολείται με τη σύγχρονη γλώσσα MetateM και συγκεκριμένα με την κατασκευή περιβάλλοντος εκτέλεσής της.

Πολλές «πρακτορο-στραφείς» γλώσσες προγραμματισμού βασίζονται σε λογική όπως αυτή της γλώσσας Prolog, δηλαδή για την κατασκευή των πρακτόρων χρησιμοποιείται η λογική προσέγγιση. Συγκεκριμένα, η εγγραφή ενός προγράμματος Prolog συνίσταται στην περιγραφή των δεδομένων και των σχέσεων που τα διέπουν με τη βοήθεια προτάσεων ενός υποσυνόλου της λογικής πρώτης τάξης.

Η προσέγγιση που ακολουθείται στην παρούσα εργασία είναι διαφορετική. Η βασική υπολογιστική προσέγγιση για την «πρακτορο-στραφή» γλώσσα προγραμματισμού MetateM είναι λογική, όμως η υποκείμενη βάση είναι χρονική. Η σύγχρονη MetateM είναι μια γλώσσα προγραμματισμού που βασίζεται στην έννοια των συγχρονισμένων, επικοινωνούντων αντικειμένων, όπου κάθε αντικείμενο εκτελεί άμεσα ένα τους κανόνες που το διέπουν, οι οποίοι δίδονται σε χρονική λογική, και επικοινωνεί με τα άλλα αντικείμενα χρησιμοποιώντας ασύγχρονη «πανεκπομπή» (broadcast) μηνυμάτων. Έτσι η Ταυτόχρονη MetateM αποτελεί ένα συνδυασμό άμεσης εκτέλεσης χρονικών specifications καθώς και ενός μοντέλου ταυτόχρονου υπολογισμού. Ένα αντικείμενο αποτελείται από ένα σύνολο λογικών κανόνων και η επικοινωνία επιτυγχάνεται με την αξιολόγηση ορισμένων τύπων κατηγορημάτων βάσει των πεποιθήσεών των αντικειμένων, των επιθυμιών και των στόχων τους.

1.2Αντικείμενο Διπλωματικής

Αντικείμενο της παρούσας διπλωματικής είναι η κατασκευή ενός περιβάλλοντος στο οποίο πράκτορες συνεπαγωγικού συλλογισμού δρουν βάσει της σύγχρονης γλώσσας MetateM. Ο κώδικας που έχει γραφεί στοχεύει σε μια εφαρμογή των θεμελιωδών πτυχών της θεωρίας γύρω από τη MetateM . Ιδιαίτερα, σε αυτή την εφαρμογή, έχει γίνει προσπάθεια να δοθεί στους πράκτορες η δυνατότητα της δυναμικής αλληλεπίδρασης μεταξύ των πρακτόρων που συμμετέχουν στην εφαρμογή αλλά και η ανεξάρτητη λειτουργία κατά τη διάρκεια της εκτέλεσης.

2 Η σύγχρονη (concurrent) γλώσσα MetateM (Fisher 1994)

2.1 Τι είναι Πράκτορας

Ο πράκτορας είναι ένα υπολογιστικό σύστημα που βρίσκεται σε κάποιο περιβάλλον, και το οποίο είναι ικανό για αυτόνομη δράση μέσα σε αυτό το περιβάλλον προκειμένου να ικανοποιήσει τους σχεδιαστικούς του στόχους. - Wooldridge & Jennings

An agent is a computer system capable of flexible autonomous action…

Πράκτορας (Agent) είναι μια οντότητα (άτομο, οργανισμός, σύστημα) που λειτουργεί εκ μέρους μίας άλλης οντότητας με αυτόνομο τρόπο. Ένας πράκτορας έχει βασικό του σκοπό να πετύχει τους στόχους που του έχουν ανατεθεί. Για να το καταφέρει αυτό επιλέγει αυτόνομα τις κατάλληλες ενέργειες, σε σχέση με τις συνθήκες του περιβάλλοντός του, βασιζόμενος στις δυνατότητες και τα μέσα που έχει στη διάθεσή του μέχρι να επιτύχει το στόχο του, είτε να αποτύχει, είτε να χρειαστεί εκ νέου αποφάσεις/οδηγίες ή τερματιστεί από το χρήστη στον οποίο ανήκει.

Ανάλογα με τον τομέα εφαρμογής διαφορετικά χαρακτηριστικά θεωρούνται σημαντικά για να χαρακτηρίζεται ένα πρόγραμμα λογισμικού πράκτορας. Γενικά, κάθε πράκτορας πρέπει να μπορεί να λειτουργεί με :

- **Αντιδρασικότητα (ορθολογισμός):** οι νοήμονες πράκτορες μπορούν να αντιλαμβάνονται το περιβάλλον τους και να απαντούν με τον καλύτερο δυνατό τρόπο, εντός λογικού χρόνου σε αλλαγές που συμβαίνουν σε αυτό, με σκοπό να εκπληρώσουν τους στόχους τους.
- **Ενεργητικότητα (αυτονομία):** οι νοήμονες πράκτορες μπορούν να πάρουν πρωτοβουλίες και να ενεργήσουν προς την εκπλήρωση των στόχων τους.
- **Κοινωνικότητα:** οι νοήμονες πράκτορες μπορούν να αλληλεπιδρούν με άλλους πράκτορες (και ανθρώπους) για να εκπληρώσουν τους στόχους τους. Αυτή η αλληλεπίδραση δεν περιορίζεται μόνο στην ανταλλαγή δεδομένων αλλά έχει χαρακτηριστικά που την κάνουν να μοιάζει με ανθρώπινη αλληλεπίδραση π.χ. για τη σύναψη συμφωνιών, τη δημιουργία συνεργασιών, το συντονισμό ενεργειών με άλλους πράκτορες.

Αρχιτεκτονική ενός πράκτορα μπορεί να οριστεί σαν μια μεθοδολογία που καθορίζει πως ένας πράκτορας μπορεί να αποσυντεθεί σε ένα σύνολο μερών και πως αυτά τα μέρη μπορούν να αλληλεπιδρούν μεταξύ τους. Το σύνολο των μερών και οι αλληλεπιδράσεις τους είναι ουσιαστικά υλοποίηση του πώς τα δεδομένα που λαμβάνει ο πράκτορας από το περιβάλλον του και η παρούσα κατάστασή του μπορεί να καθορίσει τη δράση του και σε περιπτώσεις τη μελλοντική εσωτερική του κατάσταση. Βάσει της αρχιτεκτονικής τους οι πράκτορες μπορούν να διακριθούν στους παρακάτω.

- Πράκτορες Συμβολικού Συλλογισμού
- Πράκτορες Παραγωγικού / Επαγωγικού Συλλογισμού

Πράκτορες Συμβολικού Συλλογισμού

Η αρχιτεκτονική ενός πράκτορα Συμβολικού Συλλογισμού καθορίζεται έτσι ώστε ο πράκτορας να έχει ένα αναπαραστατικό, συμβολικό μοντέλο του κόσμου και να λαμβάνει αποφάσεις, για παράδειγμα για το ποιες ενέργειες να κάνει, μέσω Συμβολικού Συλλογισμού.

Ουσιαστικά ο στόχος κατά τη σχεδίαση του πράκτορα Συμβολικού Συλλογισμού ήταν η υλοποίηση ενός πράκτορα που θα ήταν σε θέση να αποδεικνύει θεωρήματα με βάση τις προτάσεις που βρίσκονται στη βάση γνώσης του. Ο πράκτορας σε κάθε βήμα του και δοθείσης μιας κατάστασης ενεργεί όσο καλύτερα μπορεί. Βασικό συστατικό τους είναι η βάση γνώσης τους. Η βάση γνώσης τους περιέχει προτάσεις που περιγράφουν το *περιβάλλον του πράκτορα* και τις *προϋποθέσεις και τα αποτελέσματα των ενεργειών του.*

Η προτάσεις αναπαρίστανται σε μια τυπική γλώσσα αναπαράστασης, δηλαδή κάποιο σύστημα Λογικής. Το σύστημα λογικής για τις προτάσεις ενός πράκτορα επαγωγικού συλλογισμού περιλαμβάνει *σύμβολα της γλώσσας και κανόνες για τη δημιουργία προτάσεων σε αυτή* και *κανόνες εξαγωγής συμπεράσματος, δηλαδή κανόνες δημιουργίας νέων προτάσεων με βάση τις υπάρχουσες.* Ο πράκτορας επεξεργάζεται «συντακτικά» αυτή την αναπαράσταση για να αποφασίσει ποια ενέργεια να εκτελέσει δεδομένων των αντιλήψεών του. [9]

Πράκτορες Επαγωγικού Συλλογισμού

Η βάση γνώσης ενός πράκτορα Επαγωγικού Συλλογισμού αντιστοιχεί στην εσωτερική του κατάσταση. Η πληροφορία αναπαρίσταται σε κατηγορηματική λογική (ή λογική πρώτης τάξης). Τα περιεχόμενα της βάσης γνώσης αντιστοιχούν σε

αυτό που αποκαλούμε «πεποιθήσεις» για τους ανθρώπους, δηλαδή ο πράκτορας θεωρεί ότι οι προτάσεις της βάσης γνώσης του είναι αληθείς.

Όπως συμβαίνει και με τις πεποιθήσεις των ανθρώπων, μπορεί οι πεποιθήσεις ενός πράκτορα να είναι λάθος (π.χ. επειδή υπάρχει σφάλμα σε κάποιον αισθητήρα, επειδή η πληροφορία του πράκτορα δεν είναι ενημερωμένη, επειδή ο συλλογισμός του πράκτορα είναι λανθασμένος κλπ) ή ελλιπής. Πρέπει να υπάρχει λοιπόν προσοχή, ώστε να ισχύει η διάκριση πραγματικού κόσμου και του «μικρο-κόσμου» του πράκτορα .

2.2 Συστήματα Πολλαπλών Πρακτόρων (Multi-Agent System - MAS)

Τα «πρακτορο-στραφή» συστήματα έχουν γνωρίσει μεγάλη άνθηση τα τελευταία χρόνια ως ενός νέου τρόπου αντίληψης, σχεδιασμού και εφαρμογής συστημάτων λογισμικού. Αρχικά η μεγάλη πλειοψηφία των «πρακτορο-στραφών» συστημάτων βασίζονταν σε συστήματα που αποτελούνταν από ένα πράκτορα. Εντούτοις, δεδομένου ότι η τεχνολογία ωριμάζει και εξετάζει τις όλο και περισσότερο σύνθετες εφαρμογές, η ανάγκη για τα συστήματα που αποτελούνται από πολλαπλούς πράκτορες που ενεργούν ισότιμα γίνεται προφανής. Κεντρική ιδέα, δηλαδή για το σχεδιασμό και την αποτελεσματική λειτουργία των συστημάτων αποτελούν τα Συστήματα Πολλαπλών Πρακτόρων (Multi-Agent Systems - MAS).

Ένα Multi-Agent System είναι μια «κοινωνία» που αποτελείται από πολλούς πράκτορες που αλληλεπιδρούν ανταλλάσοντας μηνύματα μέσω ενός δικτύου. Τα χαρακτηριστικά ενός MAS είναι τα παρακάτω :

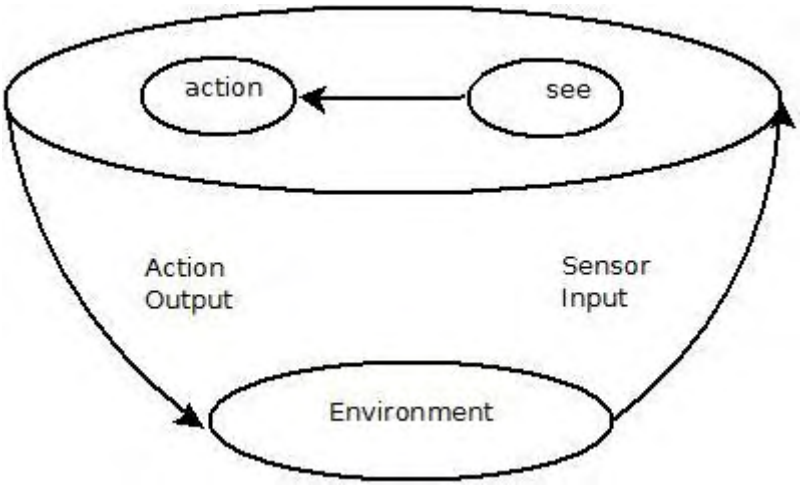
- Κάθε πράκτορας έχει ελλειπή γνώση ή δυνατότητες για να επιλύσει ένα πρόβλημα και ως αποτέλεσμα έχει περιορισμένη άποψη επί του θέματος.
- Δεν υπάρχει καθολικός έλεγχος του συστήματος.
- Τα δεδομένα είναι αποκεντρικοποιημένα.
- Ο υπολογισμός είναι ασύγχρονος.

Στη γενικότερη περίπτωση κάθε πράκτορας εκπροσωπεί διαφορετικά συμφέροντα και για να αλληλεπιδράσουν επιτυχώς οι πράκτορες πρέπει να διαπραγματευτούν, να συνεργαστούν και να συντονιστούν όπως οι άνθρωποι στις κοινωνίες τους. Η αλληλεπίδρασή τους μπορεί να είναι άλλοτε συνεργατική ή εγωιστική. Αυτό σημαίνει ότι οι πράκτορες μπορούν είτε να μοιράζονται έναν κοινό στόχο είτε να λειτουργούν βάσει των δικών τους καθαρά συμφερόντων. [9]

2.3 Πράκτορας Επαγωγικού Συλλογισμού

Βασική ιδέα του πράκτορα επαγωγικού συλλογισμού είναι να δράσει όσο καλύτερα μπορεί δοθείσης μιας κατάστασης. Σκοπός είναι να υλοποιηθεί ένας πράκτορας ο οποίος θα είναι σε θέση να αποδεικνύει θεωρήματα. Σε αυτό το σημείο μπορεί να οριστούν δύο ειδών προβλήματα:

- Το transduction problem είναι το πρόβλημα «μετάφρασης» του πραγματικού κόσμου σε μια καλή συμβολική περιγραφή.
- Το reasoning problem είναι να «χειραγωγείς» τους πράκτορες ώστε να δρουν βάσει αυτής της γνώσης.



Εικόνα 1

Agent as Theorem Provers …

Ο πράκτορας έχει μία βάση δεδομένων Δ που αποτελείται από κατηγορήματα όπως τις παρακάτω:

open (valve221)

dirt (0 , 1)

in (3 , 2)

dir (x , y) ^ in (x , y) → do (clean).

Η βάση δεδομένων είναι πληροφορίες που διαθέτει ο πράκτορας για το περιβάλλον του, και ο ρόλος που παίζει είναι κατά κάποιο τρόπο ανάλογος με το ρόλο που παίζουν οι πεποιθήσεις για τους ανθρώπους. Έτσι κάποιος άνθρωπος μπορεί να πιστεύει ότι η βαλβίδα είναι ανοιχτή (open(valve221)) και ο πράκτορας μπορεί να έχει το αντίστοιχο κατηγόρημα στη βάση δεδομένων του.

Έστω L το σύνολο όλων των προτάσεων της κλασικής λογικής πρώτης τάξης, και έστω D = *p*(L) το σύνολο των βάσεων δεδομένων που προκύπτουν από το L, δηλαδή το σύνολο των συνόλων προτάσεων του L. Η εσωτερική κατάσταση ενός πράκτορα είναι ένα στοιχείοτου συνόλου D. Συμβολίζουμε με Δ, Δ_1,… τα μέλη του D. Η διαδικασία λήψης απόφασης ενός πράκτορα μοντελοποιείται ως ένα σύνολο κανόνων παραγωγής (deduction rules) ***p.*** Αυτοί είναι απλώς οι κανόνες συλλογισμού της λογικής. Γράφουμε $\Delta \vdash_p \phi$ αν η πρόταση φ μπορεί να αποδειχθεί από τη βάση δεδομένων Δ με χρήση μόνο τν κανόνων παραγωγής ***p.*** Η συνάρτηση αντίληψης **see** ενός πράκτορα παραμένει αμετάβλητη :

$$See : S \rightarrow Per.$$

Παρομοίως, η συνάρτηση **next** έχει τη μορφή :

$$Next : D \times Per \rightarrow D.$$

Δηλαδή απεικονίζει μια βάση δεδομένων και μια αντίληψη σε μια νέα βάση δεδομένων. Ωστόσο, η συνάρτηση επιλογής ενέργειας του πράκτορα, η οποία έχει τη μορφή

$$Action : D \rightarrow Ac,$$

ορίζεται με βάση τους κανόνες παραγωγής συμπεράσματος. Ο ορισμός αυτής της συνάρτησης με ψευδοκώδικα φαίνεται παρακάτω.

```
function action(Δ:D) returns an action
begin
  for each a ∈ Actions do
    if Δ →p Do(a) then
      return a
    end-if
  end-for
  for each a ∈ Actions do
    if ¬(Δ →p ¬Do(a)) then
      return a
    end-if
  end-for
  return nil
end
```

2.3.1 Παράδειγμα Πράκτορα Επαγωγικού Συλλογισμού

Vacuum World

Dirt (0,2)	Dirt (1,2)	(2,2)
(0,1)	Robot (1,1)	(2,1)
(0,0)	(1,0)	(2,0)

- In(x,y) : ο πράκτορας βρίσκεται στο x,y.
- Dirt(x,y) : υπάρχει σκόνη στο x,y.
- Facing(d) : ο πράκτορας κοιτάει προς την κατεύθυνση d.

Οι πιθανοί κανόνες που προκύπτουν από τα παραπάνω είναι

In (x,y) ^ Dirt(x,y) → Do(suck)

In (0,0) ^ Facing (north) ^ ¬Dirt (0,0) → Do (forward)

In (0,1) ^ Facing (north) ^ ¬Dirt (0,1) → Do (forward)

In (0,2) ^ Facing (north) ^ ¬Dirt (0,2) → Do (turn)

In(0,2) ^ Facing(east) → Do(forward)

Οι κανόνες για να μαζέψει τη σκόνη το ρομπότ θα μπορούσαν να είναι όπως οι παρακάτω :

In (x,y) ^ Dirt(x,y) → Do(suck)

In (x,y) ^ ¬ Dirt(x,y) ^ ¬ Pebble (x,y) → Do(drop-pebble)

In (x,y) ^ ¬ Dirt(a,b) ^ (a $\neq x \vee b \neq y$) → Do(turn – towards (a, b)) ^ Do (forward)

2.4 Πρακτοροστραφής Προγραμματισμός (Agent Oriented programming)

Η βασική ιδέα του πρακτοροστραφή προγραμματισμού είναι ότι σε αντίθεση με προηγουμένως, πλέον μπορούμε να προγραμματίσουμε απευθείας τους πράκτορες χρησιμοποιώντας νοητικές έννοιες όπως πεποίθηση, επιθυμία, και πρόθεση.

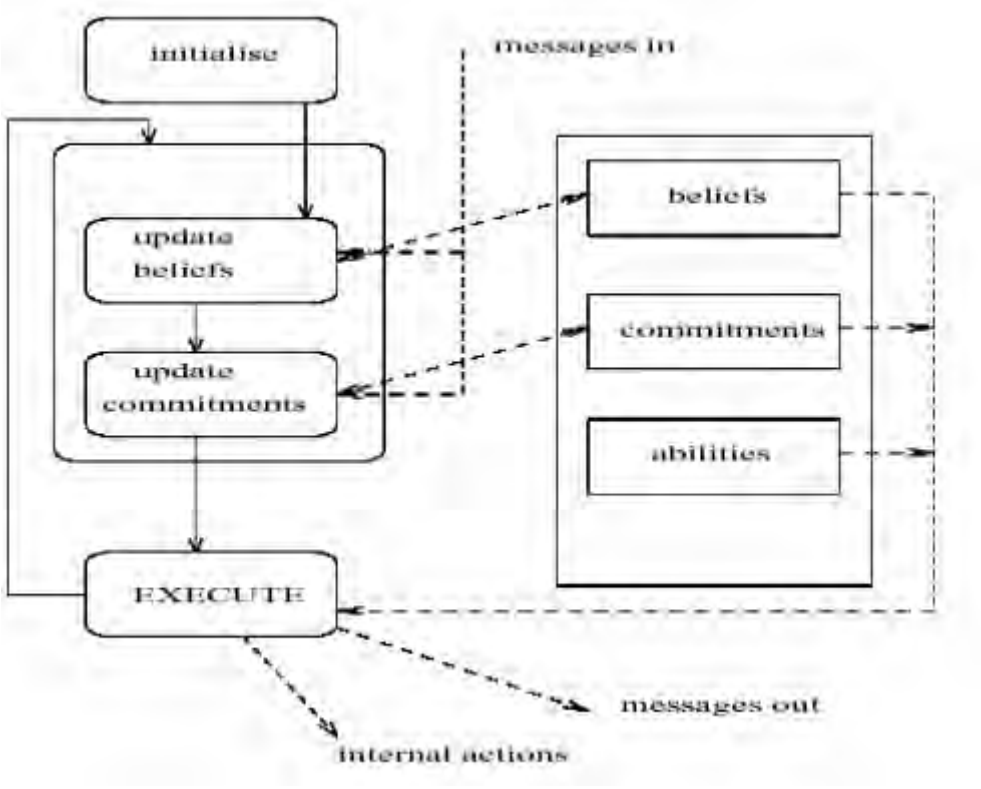
Κατά κάποιο τρόπο, λειτουργούμε με βάση την ιδέα ότι η ίδια λογική που μπορεί να περιγράψει τον άνθρωπο, θα ήταν ίσως χρήσιμη να χρησιμοποιηθεί για να προγραμματιστεί μια μηχανή.

2.4.1 Η πρώτη υλοποίηση του πρακτοροστρεφούς προγραμματισμού – AGENT0

Η πρώτη υλοποίηση του πρακτοροστραφούς προγραμματισμού ήταν η γλώσσα προγραμματισμού AGENT0 [Shoham, 1991] η οποία σήμερα δε χρησιμοποιείται πλέον. Στη γλώσσα αυτή ο πράκτορας ορίζεται από ένα σύνολο **ικανοτήτων** (πράγματα που μπορεί να κάνει), ένα σύνολο **αρχικών πεποιθήσεων**, ένα σύνολο **αρχικών δεσμεύσεων** και ένα **σύνολο κανόνων δέσμευσης**.

Οι κανόνες δέσμευσης περιέχουν μια *συνθήκη μηνύματος* και μια *νοητική συνθήκη* . Ένας πράκτορας για να καθορίσει αν ενεργοποιείται ένας κανόνας δέσμευσης , ελέγχει αν η συνθήκη μηνύματος συμφωνεί με τα μηνύματα που έχει λάβει και αν η νοητική συνθήκη συμφωνεί με τις πεποιθήσεις του. Τα μηνύματα περιορίζονται στις εξής κατηγορίες : «αίτηση» , «μη - αίτηση» και «πληροφορίας».

Το παρακάτω διάγραμμα παρουσιάζει τη λειτουργία ενός πράκτορα στην AGENT0



Εικόνα 2

Αν για παράδειγμα έχω τον παρακάτω κανόνα δέσμευσης

```
COMMIT(
  (agent, REQUEST, DO(time, action)) ;msg condition
  (B,
    [now, Friend agent] AND
    CAN(self, action) AND
    NOT [time, CMT(self, anyact)]) ;metal condition
  self,
  DO(time, action))
```

- αν λάβω ένα μήνυμα από έναν agent που μου αιτείται να κάνω action στη χρονική στιγμή time και πιστεύω ότι
    - ο agent είναι ένας friend
    - μπορώ να κάνω action
    - τη χρονική στιγμή time, δεν υποχρεούμαι να κάνω οποιαδήποτε άλλη action.
- τότε θα κάνω την action τη χρονική στιγμή time.

Η AGENT0 παρέχει υποστήριξη για πολλαπλούς πράκτορες ώστε να συνεργάζονται και να επικοινωνούν. Εντούτοις, η AGENT0 αποτελεί ένα πρωτότυπο, που σχεδιάστηκε για να επεξηγήσει μερικές αρχές, παρά για να είναι μια γλώσσα παραγωγής.

2.5 Η Σύγχρονη MetateM (Concurrent MetateM)

Η Σύγχρονη MetateM (Concurrent Metatem) είναι μια πολυπρακτορική γλώσσα (multi-agent language) στην οποία κάθε πράκτορας προγραμματίζεται βάσει της χρονικής λογικής. Αναπτύχθηκε από τον Michael Fisher και βασίζεται στην απευθείας εκτέλεση λογικών προτάσεων. Ένα σύστημα στη MetateM αποτελείται από ένα πλήθος πρακτόρων που λειτουργούν *συγχρόνως* και επικοινωνούν μεταξύ τους με *ασύγχρονη μετάδοση μηνυμάτων* (asynchronous broadcasting. Ο προσδιορισμός του πράκτορα εκτελείται απευθείας για να παραχθεί η συμπεριφορά του. Η εκτέλεση του προγράμματος ενός πράκτορα αντιστοιχεί στην επαναληπτική κατασκευή ενός λογικού μοντέλου για το χρονικό προσδιορισμό του.

Κάθε πράκτορας στη MetateM έχει δύο κύρια χαρακτηριστικά :

- *Μια διεπαφή (interface),* που ορίζει τον τρόπο με τον οποίο ο πράκτορας μπορεί να αλληλεπιδρά με το περιβάλλον του (δηλαδή με άλλους πράκτορες), και
- *Μια υπολογιστική μονάδα,* που ορίζει τον τρόπο με τον οποίο ενεργεί ο πράκτορας.

2.5.1 Διεπαφή πράκτορα στην MetateM

Διεπαφή Αντικειμένου Οι πράκτορες επικοινωνούν μεταξύ τους με πανεκπομπή (broadcast) μηνυμάτων. Κάθε πράκτορας ενεργεί βάσει των μηνυμάτων που «αναγνωρίζει». Αυτό σημαίνει ότι κάθε πράκτορας πρέπει να είναι σε θέση να φιλτράρει και να λαμβάνει υπόψη του τα μηνύματα αυτά που «αναγνωρίζει» ως χρήσιμα και να αγνοεί όλα τα υπόλοιπα. Το ποια μηνύματα αναγνωρίζει ένας πράκτορας καθώς και ποια μηνύματα ένας πράκτορας μπορεί να παραγάγει καθορίζονται από τη *διεπαφή του πράκτορα (interface definition).* Η διεπαφή ενός πράκτορα αποτελείται από

- ένα *μοναδικό όνομα* που ορίζει την ταυτότητά του
- ένα *σύνολο συμβόλων* που ορίζουν τα μηνύματα *που μπορεί να δεχτεί ο πράκτορας* (είσοδος πράκτορα, προτάσεις περιβάλλοντος)

- ένα *σύνολο συμβόλων* που ορίζουν τα μηνύματα *που μπορεί να στείλει ο πράκτορας* (έξοδος πράκτορα, προτάσεις συστατικού)

Για παράδειγμα η διασύνδεση αντικειμένου για τον πράκτορα 'stack' καθορίζεται με τον παρακάτω τρόπο

stack (pop, push) [popped, full].

Εδώ, το (pop, push) είναι το *σύνολο των συμβόλων που ο πράκτορας αναγνωρίζει* και μπορεί να δεχτεί, ενώ το [popped, full] είναι το *σύνολο των συμβόλων που ένα αντικείμενο μπορεί να παραγάγει.* Σε αυτό το σημείο πρέπει να σημειωθεί ότι τα παραπάνω σύνολα μπορεί να μην είναι ξένα μεταξύ τους. Αυτό σημαίνει ότι ένας πράκτορας μπορεί να «πανεκπέμπει» και τα σύμβολα που αυτός αναγνωρίζει. Σε αυτή την περίπτωση, τα μηνύματα που στέλνονται από τον πράκτορα στον εαυτό του αναγνωρίζονται άμεσα. Επίσης πρέπει να σημειωθεί ότι πολλοί διαφορετικοί πράκτορες μπορεί να στέλνουν και να αναγνωρίζουν τα ίδια σύμβολα.

2.5.2 Υπολογιστική μονάδα πράκτορα στην MetateM - Χρονική Λογική

Η ***Υπολογιστική Μονάδα*** κάθε πράκτορα στη MetateM βασίζεται στην εκτέλεση προτάσεων σε χρονική λογική. Η βασική ιδέα είναι ότι εκτελείται απευθείας ο προσδιορισμός ενός πράκτορα, ο οποίος εκφράζεται ως ένα σύνολο από κανόνες προγράμματος που είναι προτάσεις της χρονικής λογικής με μορφή:

Συνθήκη που αφορά το παρελθόν $\rightarrow$ Επακόλουθο που αφορά το παρόν και το μέλλον.

Μπορούμε να δούμε τη Χρονική Λογική σαν μία επεκταμένη μορφή της κλασικής λογικής. Η Χρονική Λογική που χρησιμοποιείται στη MetateM βασίζεται σε ένα γραμμικό, διακριτό χρονικό μοντέλο. Ο χρόνος μοντελοποιείται σαν μια άπειρη ακολουθία διακριτών καταστάσεων, με ένα συγκεκριμένο αρχικό σημείο. Για τη διατήρηση της ακολουθίας γίνεται χρήση χρονικών τελεστών που καθορίζουν τη διαδοχή των καταστάσεων στην ακολουθία.

Η λογική που χρησιμοποιείται ονομάζεται *Propositional MetateM Logic* (PML) δηλαδή Προτασιακή Λογική MetateM και είναι μία απλή προτασιακή λογική πρώτης τάξης. Ουσιαστικά είναι η κλασική προτασιακή λογική επαυξημένη με ένα σύνολο τροπικών συνδετικών έτσι ώστε να αναφερόμαστε στη χρονική διάταξη των συμβάντων. Βασίζεται σε διακριτά, γραμμικά μοντέλα με πεπερασμένο παρελθόν και άπειρο μέλλον.

Σύνταξη Όπως και στην κλασική λογική, τα «συστατικά» με τα οποία κατασκευάζεται η γλώσσα είναι :

- τα *σταθερά σύμβολα* ***(Lc)***
- ένα *σύνολο από μεταβλητές* ***(Lv)***, και
- ένα σύνολο από *σύμβολα συναρτήσεων* ***(Lf)***

Από τα παραπάνω παράγεται το σύνολο των κανόνων ενός πράκτορα ***(Lt)*** . Στην FML χρησιμοποιούνται επίσης τα παρακάτω σύμβολα :

- το *σύνολο των κατηγορημάτων* ***(Lp)***, που περιλαμβάνει στοιχεία τα οποία είναι συνήθως συμβολοσειρές αλφαριθμητικών .
- τα στοιχεία $\neg, \vee, \wedge, \Rightarrow,$ και $\Leftrightarrow$.
- *μελλοντικούς χρονικούς τελεστές*, που περιλαμβάνουν μοναδιαίους τελεστές όπως οι $\bigcirc, \Diamond$ και $\Box$, και δυαδικούς τελεστές όπως οι *U* και *W*.
- *παρελθοντικούς τελεστές*, που περιλαμβάνουν μοναδιαίους τελεστές όπως οι '$\bullet$','$\bullet$', '$\blacklozenge$' και '$\blacksquare$' , και δυαδικούς τελεστές όπως οι *S* και *Z.*
- *ποσοδείκτες*, ∀ και ∃ .
- '(' και ')' τα οποία χρησιμοποιούνται προς αποφυγή ασαφειών.

Το σύνολο των καλώς διαμορφωμένων τύπων της FML (Well – formed formulae of FML - WFF_f) ορίζεται όπως ακολούθως.

1. Αν $t_1, \ldots, t_n$ ανήκουν στο σύνολο *Lt,* και p είναι κατηγόρημα τάξης n, τότε το $p(t_1, \ldots, t_n)$ ανήκει WFF_f .
2. Αν τα A και B ανήκουν στο WFF_f , τότε ανήκουν στο WFF_f και τα παρακάτω

$$\neg A \quad A \vee B \quad A \wedge B \quad A \Rightarrow B \quad (A)$$
$$\Diamond A \quad \Box A \quad A U B \quad A W B \quad \bigcirc A$$
$$\blacklozenge A \quad \blacksquare A \quad A S B \quad A Z B \quad \bullet A \quad \bullet A$$

3. Αν A ανήκει στο WFF_f και το v ανήκει στο *Lv,* τότε ∃v.A και ∀ v.A ανήκουν και τα δύο στη WFF_f.

Οι χρονικοί τελεστές που χρησιμοποιούνται στη MetateM αφορούν τόσο το παρελθόν όσο και το μέλλον.

Χρονικοί Τελεστές Οι χρονικοί τελεστές που αφορούν το μέλλον είναι οι παρακάτω:

- Ο «*κάποια στιγμή στο μέλλον*» τελεστής '$\Diamond$'.

  $\Diamond\varphi$ είναι αληθής τώρα αν το φ είναι αληθές *κάποια στιγμή* στο μέλλον.

- Ο «*συνέχεια στο μέλλον*» τελεστής '□'.

  □ *φ* είναι αληθής αν η *φ* είναι συνέχεια αληθής στο μέλλον.

- Ο «*μέχρι*» τελεστής '*U*'.

  φ U ψ είναι αληθές αν η *φ* είναι αληθής μέχρι τη στιγμή στο μέλλον που η *ψ* γίνεται αληθής.

Όμοια, υπάρχουν χρονικοί τελεστές που αναφέρονται στο παρελθόν :

- Ο «*από*» τελεστής *'S'*.

  φ S ψ είναι αληθές αν το *φ* είναι από το παρελθόν αληθής από τότε που έγινε αληθής η *ψ.*
- Ο «*κάποια στιγμή στο παρελθόν*» τελεστής '♦' ο οποίος είναι το ανάλογο του τελεστή '◊' για το παρελθόν.
- Ο «*συνέχεια στο παρελθόν*» τελεστής '■', ο οποίος είναι το ανάλογο του '□' για το παρελθόν.
- Ο τελεστής που δηλώνει την αρχή του χρόνου, '*start'.*

  '*start'* είναι αληθής στην αρχή του χρόνου.

- Ο τελεστής που δηλώνει ότι κάτι είναι αληθές στο παρελθόν , '⊙' .

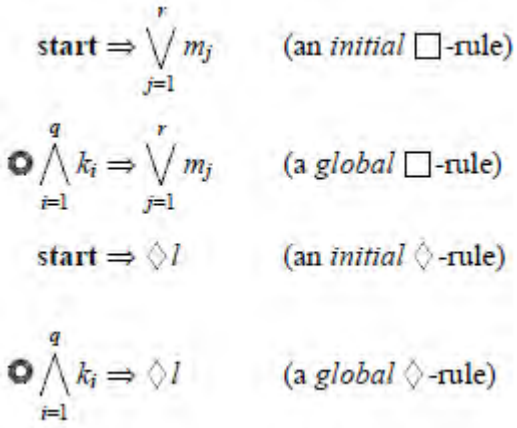
  ⊙*φ* είναι αληθές αν υπήρχε μια τελευταία στιγμή στο χρόνο και αυτή τη στιγμή το *φ* ήταν αληθές.

Με χρήση των παραπάνω τελεστών ορίζονται ένα σύνολο από κανόνες οι οποίοι είναι ουσιαστικά προστάσεις σε χρονική λογική. Η βασική ιδέα είναι ότι εκτελείται απευθείας ο προσδιορισμός ενός πράκτορα, ο οποίος εκφράζεται ως ένα σύνολο από κανόνες προγράμματος που είναι προτάσεις της χρονικής λογικής της μορφής :

Συνθήκη για το **παρελθόν** $\Rightarrow$ επακόλουθο για το **παρόν και το μέλλον**

Η συνθήκη είναι μία πρόταση χρονικής λογικής που αναφέρεται στο παρελθόν, ενώ το επακόλουθο είναι μία πρόταση χρονικής λογικής που αναφέρεται στο παρόν και το μέλλον. Η ερμηνεία ενός τέτοιου κανόνα είναι «με βάση το παρελθόν κατασκεύασε το μέλλον » και αυτό έδωσε στο υπόδειγμα το όνομά του: *δηλωτικό παρελθόν και επιτακτικό μέλλον (Gabbay, 1989)* [1] .

Σε μια πιο γενικευμένη μορφή, οι κανόνες είναι όπως οι παρακάτω:



Πρέπει να σημειώσουμε εδώ ότι το αριστερό μέρος κάθε αρχικού κανόνα (initial rule) αποτελεί περιορισμό μόνο της αρχικής κατάστασης, ενώ το αριστερό μέρος κάθε γενικού κανόνα (global rule) έχει περιορισμό για μια παρελθοντική κατάσταση. Το δεξί μέρος του $\Box$-κανόνα ($\Box$-rule) είναι απλά μια διάζευξη συμβόλων που αναφέρονται στην παρούσα κατάσταση ενώ το δεξί μέρος του $\Diamond$-κανόνα ($\Diamond$-rule) είναι ένα απλό ενδεχόμενο.

Σύμφωνα με τα παραπάνω πρέπει να σημειωθεί ότι χρησιμοποιώντας τη χρονική λογική ως βάση για τους κανόνες μας δίνεται μια επιπλέον δυνατότητα σε σχέση με την κλασική λογική. Ιδιαίτερα, τελεστές όπως οι '$\Diamond$' μας δίνουν τη δυνατότητα να καθορίσουμε την «απροσδιοριστία» του μέλλοντος.

Γιατί χρησιμοποιούμε χρονική λογική

Ένας λόγος είναι ότι μας επιτρέπει τη συνοπτική έκφραση των χρήσιμων δυναμικών ιδιοτήτων των επιμέρους συστατικών. Για παράδειγμα, ο κανόνας

$$\text{Request} \rightarrow \text{reply } U \text{ acknowledgement}$$

χαρακτηρίζει ένα σύστημα όπου όταν λαμβάνεται ένα request, στέλνεται ένα reply μέχρι τη στιγμή που θα ληφθεί ένα acknowledgement. Κατά συνέπεια οι χρονικοί τελεστές που αφορούν το παρελθόν μπορεί να περιγραφούν σε αυτή τη λογική.

Καθώς το χρονικό μοντέλο πάνω στο οποίο βασίζεται η χρονική λογική περιλαμβάνει μια γραμμική σειρά χρονικών στιγμών, μπορούμε να το χρησιμοποιήσουμε για να εκφράσουμε τη σειρά με την οποία λαμβάνουν χώρα οι διάφορες ενέργειες. Παραδείγματος χάριν

$$\text{Hungry} \rightarrow (\text{buy, food} \cap \bigcirc \text{cook\_food} \cap \bigcirc \text{eat}).$$

Παρατηρούμε, λοιπόν, ότι η λογική είναι χρήσιμη για να αναπαραστήσουμε τη δυναμική «δραστηριότητα» μεμονωμένων συμβόλων, και αυτό σχετίζεται πολύ στενά με τις παραδοσιακές εφαρμογές χρονικής λογικής.

2.5.3 Λειτουργία πράκτορα στη MetateM ~ Ένα επαναληπτικό μοντέλο...

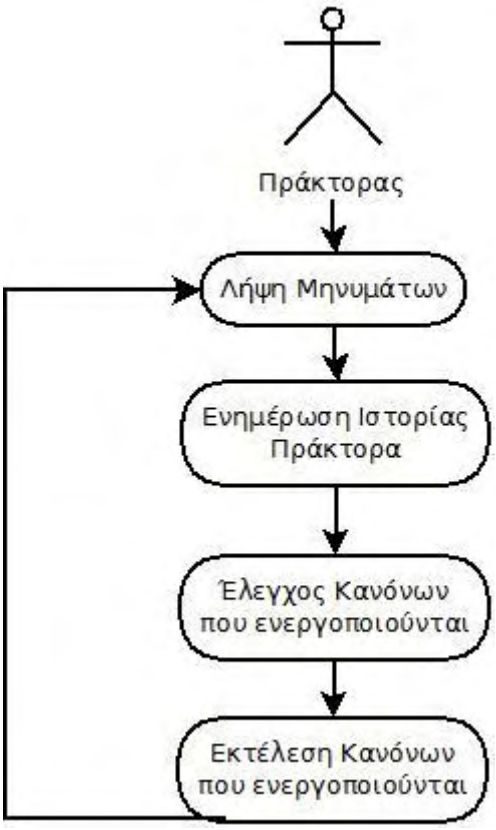
"Declarative past, imperative future" [Gabbay, 1987]

Σε αυτή την παράγραφο θα περιγράψουμε πώς εκτελείται ένα σύνολο κανόνων ενός πράκτορα στην Σύγχρονη MetateM. Η Σύγχρονη MetateM χρησιμοποιεί ένα σύνολο «κανόνων» της παραπάνω μορφής για να περιγράψει τον εσωτερικό ορισμό του πράκτορα. Οι κανόνες εφαρμόζονται κάθε στιγμή στο χρόνο (δηλαδή σε κάθε βήμα της εκτέλεσης) κάνοντας τον πράκτορα να εκτελεί ένα κύκλο.

1 Αρχικά διαβάζει τα μηνύματα που λαμβάνει από το περιβάλλον του (δηλαδή τους άλλους πράκτορες που λαμβάνουν μέρος) και ενημερώνει την ιστορία του με αυτά.
2 Ελέγχει ποιοί κανόνες ενεργοποιούνται, δηλαδή ποιών κανόνων οι συνθήκες παρελθόντος ικανοποιούνται από τα μηνύματα της τρέχουσας ιστορίας του.
3 Εκτελεί όλους τους κανόνες που ενεργοποιούνται μαζί με όποιες δεσμεύσεις είναι ενεργές από προηγούμενους κύκλους. Συγκεκριμένα μαζεύει όλα τα επακόλουθα κανόνων που είναι ενεργοποιημένα στον τωρινό κύκλο και όλα τα επακόλουθα κανόνων που ήταν ενεργοποιημένα σε προηγούμενους κύκλους, αλλά δεν εκτελέστηκαν. Αυτό έχει σαν αποτέλεσμα ο πράκτορας να καλείται να ικανοποιήσει μια λογική πρόταση που έχει τη μορφή διάζευξης. Από τη διάζευξη ο πράκτορας θα επιλέξει μια ενέργεια. Όσες ενέργειες δεν εκτελεστούν στον τρέχοντα κύκλο μεταφέρονται στον επόμενο κύκλο. [9]
4 Ο πράκτορας εκτελεί και πάλι το πρώτο βήμα.

Όταν μια πρόταση γίνει αληθής σε ένα πράκτορα, τότε αυτή συγκρίνεται με τη διεπαφή του. Αν είναι μία από τις προτάσεις συστατικού του πράκτορα, τότε εκπέμπεται ως μήνυμα προς όλους τους άλλους πράκτορες. Κατά τη λήψη ενός μηνύματος, κάθε πράκτορας προσπαθεί να συγκρίνει την πρόταση του μηνύματος

με τις προτάσεις περιβάλλοντος στη διεπαφή του. Αν υπάρχει ταύτιση, τότε προσθέτει την πρόταση του μηνύματος στην ιστορία του.



Εικόνα 3

Σαν ένα παράδειγμα ενός απλού συνόλου κανόνων για έναν πράκτορα, υποθέτουμε τα παρακάτω.

$$\begin{aligned} \textbf{start} &\Rightarrow \text{popped}(a) \\ \bigcirc\text{pop}(X) &\Rightarrow \Diamond\text{popped}(X) \\ \bigcirc\text{push}(Y) &\Rightarrow \text{stack-full}() \vee \text{popped}(Y) \end{aligned}$$

Τα ‘X’ και ‘Y’ εδώ αναπαριστούν καθολικές μεταβλητές. Στο παραπάνω σύνολο κανόνων παρατηρούμε ότι το popped (a) ικανοποιείται είτε στην αρχή του χρόνου είτε όταν το pop (X) ήταν αληθές την προηγούμενη χρονική στιγμή. Παρόμοια, όποτε ισχύει στην προηγούμενη χρονική στιγμή το push (Y) τότε θα πρέπει να ισχύει είτε το stack-full() είτε το popped(Y) .

Στη συνέχεια παρατίθεται η εκτέλεση ενός παραδείγματος

rp(ask1, ask2) [give1, give2]:

- ***ask1*** $\rightarrow \Diamond$ **give1;**
- ***ask2*** $\rightarrow \Diamond$ **give2;**

Start $\rightarrow \Box \neg$(**give1 ^ give2**).

rc1(give1) [ask1]:

Start $\rightarrow$ **ask1 ;**

- ***ask1*** $\rightarrow$ **ask1.**

Rc2(ask1, give2) [ask2]:

- **(*ask1* ^** $\neg$**ask2)** $\rightarrow$ **ask2.**

Η εκτέλεση των παραπάνω κανόνων είναι όπως παρακάτω:

Χρόνος	**Πράκτορας**		
	rp	**rc1**	**rc2**
1.		ask1	
2.	ask1	ask1	ask2
3.	ask1, ask2, give1	ask1	
4.	ask1, give2	ask1,give1	ask2
5.	ask1, ask2, give1	ask1	give2
6.	**…**	**…**	**…**

Γενικά…

- Η MetateM είναι μία γλώσσα προγραμματισμού για πράκτορες που βασίζεται στην εκτέλεση χρονικής λογικής.
- Πολλοί πράκτορες δρουν ταυτόχρονα.
- Η επικοινωνία γίνεται ασύγχρονα με πανεκπομπή μηνυμάτων.
- Η συμπεριφορά ενός πράκτορα παράγεται με άμεση εκτέλεση των προδιαγραφών του (specification)
- Η εκτέλεση αντιστοιχεί σε ένα επαναληπτικό «χτίσιμο» του μοντέλου των προδιαγραφών.

2.5.4 Επεκτάσεις του βασικού συστήματος

Σε αυτή την παράγραφο θα περιγράψουμε κάποιες επεκτάσεις που έχουν ήδη υλοποιηθεί ή είναι υπό τη διαδικασία υλοποίησης.

Αυτόνομα Αντικείμενα

- Δυναμικές Διεπαφές

Ο ορισμός της διεπαφής ενός αντικειμένου καθορίζει, όπως ήδη αναφέραμε, τα είδη των μηνυμάτων που αναγνωρίζει ένας πράκτορας/αντικείμενο. Παρόλα αυτά, ένας πράκτορας μπορεί δυναμικά να αλλάζει το σύνολο των μηνυμάτων που αναγνωρίζει. Ειδικά, ένα αντικείμενο μπορεί είτε να αρχίζει να «ακούει» νέους τύπους μηνυμάτων, είτε να αρχίζει να «αγνοεί» μηνύματα που προηγουμένως αναγνώριζε.

- Δυναμικές Ουρές Μηνυμάτων

Κάθε αντικείμενο μέσα σε ένα σύστημα έχει μία ουρά μηνυμάτων τα οποία τα έχει αναγνωρίσει αλλά δεν τα έχει λάβει ακόμα υπόψη του. Ο αριθμός των μηνυμάτων που ένα αντικείμενο διαβάζει από την ουρά μηνυμάτων του κατά τη διάρκεια της εκτέλεσης δίνεται αρχικά στη διεπαφή του. Η εκτέλεση ενός αντικειμένου βασίζεται σε ένα σύνολο μηνυμάτων που λαμβάνονται από το αντικείμενο από τη στιγμή που το τελευταίο βήμα από την προηγούμενη εκτέλεση έχει ολοκληρωθεί.

Η προκαθορισμένη συμπεριφορά ενός αντικειμένου είναι ότι για κάθε βήμα της εκτέλεσης του να διαβάζει μια σειρά από μηνύματα από την ουρά εισόδου του. Για παράδειγμα, αν η ουρά εισόδου του είναι η

```
pop(a), push(c), pop(d), pop(e), push(c), pop(f), pop(a), ...
```

τότε σε ένα βήμα εκτέλεσης το σύνολο των μηνυμάτων που διαβάζονται να είναι

```
{pop(a), push(c), pop(d), pop(e)}
```

Και η ουρά μηνυμάτων που υπολείπεται να είναι η

```
push(c), pop(f), pop(a), ...
```

Αυτή η συμπεριφορά μπορεί να αλλάξει δυναμικά έτσι ώστε, για παράδειγμα τα αντικείμενα να μπορούν να διαβάζουν μόνο ένα μήνυμα κάθε φορά από την ουρά εισόδου ή περισσότερα από ένα μηνύματα.

Μηχανισμοί Συγχρονισμού

Υπάρχουν δύο προσεγγίσεις για τον συγχρονισμό ασυγχρόνως εκτελούμενων αντικειμένων στη MetateM όπως περιγράφονται παρακάτω:

1. Όταν ένα αντικείμενο είναι στην αναμονή ενός μηνύματος, συνεχίζει να εκτελείται και όταν η απάντηση παραληφθεί από το περιβάλλον πραγματοποιείται η ενέργεια.

2. Όταν ένα αντικείμενο είναι στην περιμένει για ένα μήνυμα, εκτελεί αναμονή, δηλαδή δεν κάνει κάτι μέχρι την άφιξη του μηνύματος.

Point- to- Point μεταφορά μηνυμάτων

Αν και η αποστολή μηνυμάτων με πανεκπομπή (broadcast) έχει περιγραφεί σαν την αρχική υλοποίηση, η point-to- point μεταφορά μηνυμάτων μπορεί να υλοποιηθεί έχοντας ως βάση την πανεκπομπή. Για να παρέχει κανείς point-to-point αποστολή μηνυμάτων, πρέπει κάθε αντικείμενο να αναγνωρίζει μηνύματα όπως το send(me, X) => X . Εδώ το “me” είναι το όνομα ενός μοναδικού αντικειμένου. Έτσι, όποτε ένα αντικείμενο obj1 θελήσει να στείλει ένα μήνυμα ‘p(a)’ σε ένα άλλο αντικείμενο, obj2, πρέπει να εκπέμψει το μήνυμα send (obj2, p(a)).

Ομάδες

Σε αυτή την περίπτωση, τα αντικείμενα είναι μέλη ομάδων. Κάθε αντικείμενο μπορεί να είναι μέλος πολλών ομάδων. Όταν ένα αντικείμενο στέλνει ένα μήνυμα, αυτό το μήνυμα στέλνεται εξ ορισμού με πανεκπομπή σε όλα τα μέλη της ομάδας του, αλλά όχι σε αντικείμενα εκτός αυτής. Διαφορετικά, ένα αντικείμενο μπορεί να επιλέξει να εκπέμψει σε ορισμένες ομάδες (των οποίων είναι μέλος). Αυτός ο μηχανισμός επιτρέπει την ανάπτυξη πολύπλοκων σχημάτων. Για παράδειγμα, αν υποθέσουμε ότι δύο αντικείμενα σε μια ομάδα μπορούν να «δουν» το ένα το άλλο, μπορούμε να προσομοιώσουμε κάτι τέτοιο με τη λήψη ή όχι ενός μηνύματος από τα μέλη της ομάδας. Αν λαμβάνω το μήνυμα σημαίνει ότι μπορώ να «δω» το αντικείμενο. Διαφορετικά σημαίνει ότι «δεν βλέπω» το αντικείμενο και άρα αυτό είναι εκτός της ομάδας που ανήκω εγώ.

3 Περιγραφή Εφαρμογής

Στα πλαίσια της παρούσας διπλωματικής αναπτύχθηκε ένα περιβάλλον εκτέλεσης για την «πρακτορο – στραφή» γλώσσα προγραμματισμού MetateM. Συγκεκριμένα, ο χρήστης έχει τη δυνατότητα να εισάγει σε μορφή αρχείου .txt την υπολογιστική μονάδα, δηλαδή το σύνολο των κανόνων, κάθε πράκτορα που επιθυμεί να πάρει μέρος στο «παιχνίδι» και να δει το πώς εξελίσσεται αυτό στη διάρκεια κάθε κύκλου και μέχρι το τέλος του παιχνιδιού.

Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκε η υψηλού επιπέδου (high-level) αντικειμενοστραφής (object-oriented) γλώσσα προγραμματισμού Java, καθώς επίσης και η βάση δεδομένων της γλώσσας , Java DB, η οποία υποστηρίζει SQL ερωτήματα, τα οποία και χρησιμοποιήθηκαν για την επικοινωνία με τη βάση δεδομένων. Η πλατφόρμα που χρησιμοποιήθηκε είναι η Netbeans IDE 7.0.1.

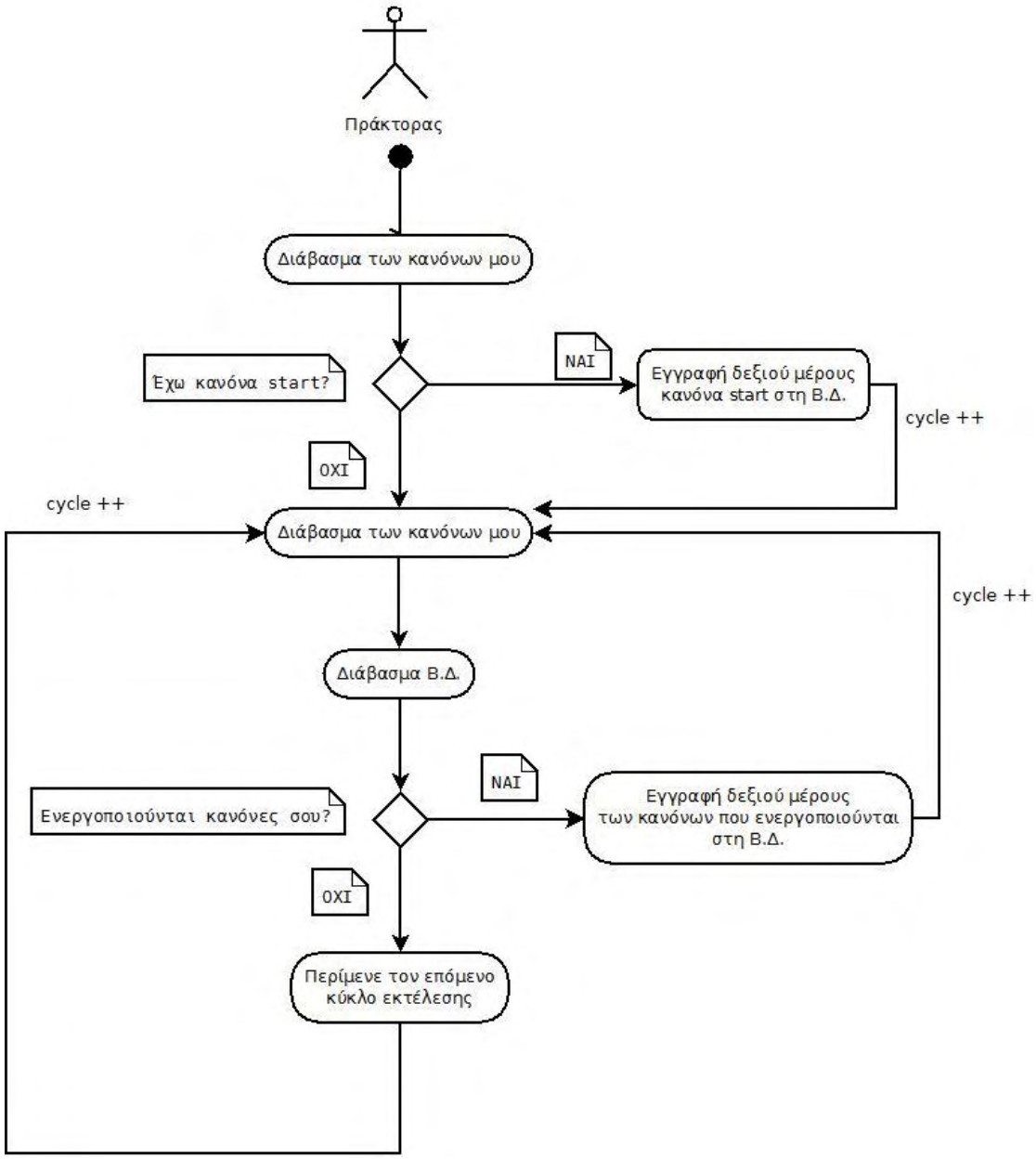
3.1 Βάση και φιλοσοφία της εφαρμογής

Όπως αναφέρθηκε και παραπάνω, κατά την εκτέλεση ενός συνόλου κανόνων ενός πράκτορα στη σύγχρονη MetateM, ο πράκτορας ουσιαστικά εκτελεί κύκλους εκτέλεσης. Αρχικά διαβάζει τα μηνύματα που λαμβάνει από τους άλλους πράκτορες (δηλαδή το περιβάλλον του) και ενημερώνει την ιστορία του με αυτά. Έπειτα ελέγχει το για το ποιοι κανόνες από την υπολογιστική του μονάδα ενεργοποιούνται, δηλαδή ποιών κανόνων οι συνθήκες παρελθόντος ικανοποιούνται από τα μηνύματα της τρέχουσας ιστορίας του και τελικά εκτελεί τους κανόνες που ενεργοποιούνται τη στιγμή εκείνη καθώς και εκείνους που είχαν ενεργοποιηθεί από κάποια παρελθοντική συνθήκη.

Βάσει αυτής της λογικής, η δομή του προγράμματος που γράφτηκε για αυτή τη διπλωματική στηρίχτηκε στο παρακάτω σκεπτικό.

Πράκτορας Αρχικά, κάθε πράκτορας πρέπει να είναι μια ξεχωριστή οντότητα η οποία εκτελείται ανεξάρτητα από τις άλλες οντότητες / πράκτορες . Για αυτό το λόγο υποθέσαμε ότι η εκτέλεση κάθε πράκτορα πρέπει να γίνεται μέσα από ένα ξεχωριστό ***νήμα***.

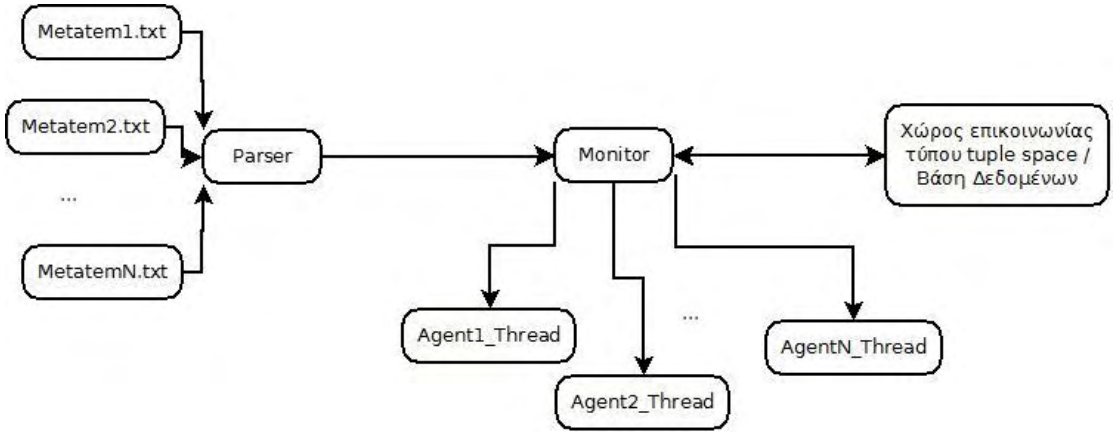
Επικοινωνία Η επικοινωνία των πρακτόρων είναι μία ασύγχρονη διαδικασία κατά την οποία κάθε πράκτορας «πανεκπέμπει» (broadcast) τα μηνύματά του στα **πλαίσια κάθε κύκλου εκτέλεσης** . Αυτό σημαίνει ότι ο πράκτορας πρέπει να δρα με τον περιορισμό ότι σε κάθε κύκλο πρέπει να στέλνει αλλά και να λαμβάνει μηνύματα που αφορούν αυτόν τον κύκλο. Σε κάθε κύκλο εκτέλεσης μας είναι αδιάφορο το με ποια σειρά θα δράσουν οι πράκτορες. Για την υλοποίηση αυτών των περιορισμών, στην εφαρμογή υποθέσαμε ότι η υλοποίηση της ανταλλαγής μηνυμάτων γίνεται μέσα από μία Βάση Δεδομένων η οποία λειτουργεί ***στη λογική ενός Tuple Space***. Συγκεκριμένα, και αρχίζοντας από τον πρώτο κύκλο εκτέλεσης, εγγράφονται στη Βάση Δεδομένων τα δεξιά μέλη των κανόνων *start* . Σε κάθε έναν από τους επόμενους κύκλους εκτέλεσης, κάθε πράκτορας διαβάζει τις εγγραφές της Βάσης Δεδομένων, και εφόσον ικανοποιείται το αριστερό μέρος κάποιου κανόνα του, εγγράφει στη Βάση δεδομένων το δεξί μέρος του κανόνα (είτε την ίδια χρονική στιγμή, είτε σε κάποια από τις επόμενες, ανάλογα με το χρονικό τελεστή). Συγκεκριμένα, εγγράφει σε έναν πίνακα το όνομά του (δηλαδή ποιος είναι ο αποστολέας του μηνύματος - SENDER) και ποιος είναι ο τύπος του μηνύματος (TYPE). Επίσης στον πίνακα εγγράφεται και το σε ποιον κύκλο του παιχνιδιού βρισκόμαστε. Ύστερα από αυτού του τύπου την «πανεκπομπή», οι υπόλοιποι πράκτορες είναι σε θέση να διαβάσουν τα μηνύματά του στον επόμενο κύκλο εκτέλεσης και να δράσουν ανάλογα. Δηλαδή, γενικά, σε κάθε κύκλο παιχνιδιού όλοι οι συμμετέχοντες πράκτορες ελέγχουν την εξέλιξη του παιχνιδιού κοιτώντας τη βάση δεδομένων/ πίνακα και τις εγγραφές του προηγούμενου κύκλου ή των προηγούμενων κύκλων έτσι ώστε να διαπιστώσουν αν ικανοποιείται κάποιος ή κάποιοι κανόνες τους. Εφόσον κάτι τέτοιο ισχύει, ο κάθε πράκτορας εγγράφει το δεξί μέρος κάθε κανόνα που ικανοποιείται στη βάση δεδομένων. Αυτό γίνεται είτε την επόμενη χρονική στιγμή είτε αργότερα, ανάλογα με το χρονικό τελεστή του δεξιού μέρους του κανόνα.



Εικόνα 4

Συντακτική Ανάλυση - Parser Όπως αναφέρθηκε και παραπάνω, ο χρήστης εισάγει τους κανόνες του πράκτορα στην εφαρμογή μέσα από ένα αρχείο .txt. Πριν την εκτέλεση του παιχνιδιού, γίνεται η συντακτική ανάλυση των κανόνων του κάθε πράκτορα για να εξακριβωθεί η ορθότητά τους βάσει της γραμματικής. Συγκεκριμένα, διαβάζεται κάθε κανόνας , η λήξη του οποίου σημειώνεται με το '**;**'. Κάθε κανόνας αναλύεται σε μία σειρά από "tokens",και συγκεκριμένα εδώ σε μια σειρά από λέξεις, για να εξακριβωθεί αν η γραμματική του δομή «σέβεται» τη γραμματική των κανόνων της MetateM.

Η απεικόνιση της γενικής ιδέας πάνω στην οποία στηρίχτηκε η υλοποίηση της εφαρμογής παρουσιάζεται στο παρακάτω σχεδιάγραμμα.



Εικόνα 5

3.2 Επεξήγηση Κώδικα Εφαρμογής

Ο κώδικας της εφαρμογής αποτελείται από οχτώ κλάσεις. Το βασικό σκεπτικό ήταν η υλοποίηση μιας κλάσης στην οποία θα γίνεται ο έλεγχος για τη συντακτική ορθότητα των κανόνων που εισάγει ο χρήστης, καθώς και η υλοποίηση μιας κλάσης – μόνιτορ, μέσω της οποίας γίνεται ο μεταξύ των πρακτόρων συντονισμός καθώς επίσης και η αλληλεπίδραση των πρακτόρων – νημάτων με τη Βάση Δεδομένων της εφαρμογής . Παρακάτω παρατίθενται αναλυτικά οι κλάσεις και οι μέθοδοί τους.

Agent.java

Η κλάση Agent.java αποτελεί επέκταση της Thread. Εδώ υλοποιείται η μέθοδος run() η οποία περιλαμβάνει τις ενέργειες που εκτελεί ένα νήμα από τη στιγμή που δημιουργείται.

private int analyze(String s) Εδώ γίνεται ο έλεγχος για την ορθότητα των κανόνων που εισάγει ο χρήστης . Η μέθοδος επιστρέφει έναν ακέραιο αριθμό ο οποίος κάθε φορά αντιστοιχεί είτε σε ένα συγκεκριμένο τύπο κανόνα, είτε αντιστοιχεί σε εντοπισμό λάθους στη γραμματική του κανόνα.

Συγκεκριμένα επιστρέφεται :

- **0** αν ο κανόνας που αναγνωρίζεται είναι της μορφής
  Start = <Μελλοντικός_Τελεστής > <Σύμβολο>

- **1** αν ο κανόνας που αναγνωρίζεται είναι της μορφής
  <Παρελθοντικός_ Τελεστής> <not> <Σύμβολο> = <Μελλοντικός_ Τελεστής> <Σύμβολο>

- **2** αν ο κανόνας που αναγνωρίζεται είναι της μορφής
  <Παρελθοντικός_ Τελεστής> <Σύμβολο> = <Μελλοντικός_ Τελεστής> <Σύμβολο>

- **3** αν ο κανόνας που αναγνωρίζεται είναι της μορφής
  <Παρελθοντικός_ Τελεστής> <not> <Σύμβολο> <Λογικός_ Τελεστής> <Παρελθοντικός_ Τελεστής> <not> <Σύμβολο> = <Μελλοντικός_ Τελεστής> <Σύμβολο>

- **4** αν ο κανόνας που αναγνωρίζεται είναι της μορφής
  <Παρελθοντικός_ Τελεστής> <Σύμβολο> <Λογικός_ Τελεστής <Παρελθοντικός_ Τελεστής> <not> <Σύμβολο> = <Μελλοντικός_ Τελεστής> <Σύμβολο>

- **5** αν ο κανόνας που αναγνωρίζεται είναι της μορφής
  <Παρελθοντικός_ Τελεστής> <not> <Σύμβολο> <Λογικός_ Τελεστής> <Παρελθοντικός_ Τελεστής> <Σύμβολο> = <Μελλοντικός_ Τελεστής> <Σύμβολο>

- **6** αν ο κανόνας που αναγνωρίζεται είναι της μορφής
  <Παρελθοντικός_ Τελεστής> <Σύμβολο> <Λογικός_ Τελεστής> <Παρελθοντικός_ Τελεστής> <Σύμβολο> = <Μελλοντικός_ Τελεστής> <Σύμβολο>

- **7** αν ο κανόνας που αναγνωρίζεται είναι της μορφής
  <Παρελθοντικός_ Τελεστής> <(> <Σύμβολο> <Λογικός_ Τελεστής> <Σύμβολο> <)> = <Μελλοντικός_ Τελεστής> <Σύμβολο>

- **8** αν ο κανόνας που αναγνωρίζεται είναι της μορφής
  <Παρελθοντικός_ Τελεστής> <(> <not> <Σύμβολο> <Λογικός_ Τελεστής> <Σύμβολο> <)> = <Μελλοντικός_ Τελεστής> <Σύμβολο>

- **9** αν ο κανόνας που αναγνωρίζεται είναι της μορφής
  <Παρελθοντικός_ Τελεστής> <(> <Σύμβολο> <Λογικός_ Τελεστής> <not> <Σύμβολο> <)> = <Μελλοντικός_ Τελεστής> <Σύμβολο>

- **10** αν ο κανόνας που αναγνωρίζεται είναι της μορφής

<Παρελθοντικός_ Τελεστής> <(> <not> <Σύμβολο> <Λογικός_ Τελεστής> <not> <Σύμβολο> <)> = <Μελλοντικός_ Τελεστής> <Σύμβολο>

- **-1** αν ο κανόνας που εισάγεται δεν αναγνωρίζεται

private void fillRules() Στη συνάρτηση αυτή αρχικά καλείται η συνάρτηση analyze() . Ανάλογα με την τιμή που θα επιστραφεί από τη συνάρτηση analyze(), ανάλογα δηλαδή με τη μορφή του κανόνα, τοποθετούνται τα αριστερά (left) και δεξιά (right) μέρη των κανόνων σε μια ArrayList τύπου κλάσης Rule.

private boolean isPast(String s) Ελέγχει αν ο χρονικός τελεστής είναι παρελθοντικός , δηλαδή αν είναι “ALWAYSPAST”, “PAST” ή “YESTERDAY”.

private boolean isFuture(String s) Ελέγχει αν ο χρονικός τελεστής είναι μελλοντικός , δηλαδή αν είναι “TOMORROW”, “FUTURE”, “ALWAYSFUTURE” ή ”NOW”.

private boolean isLog(String s) Ελέγχει αν η συμβολοσειρά που διαβάζεται είναι λογικός τελεστής, δηλαδή αν είναι “OR”, “AND”,”NOTOR” ή “NOTAND”.

private TimeEnum getTimeEnum(String s) Ελέγχει αν η συμβολοσειρά που διαβάζεται είναι χρονικός τελεστής, δηλαδή αν είναι “TOMORROW”, “FUTURE”, “ALWAYSFUTURE”, ”NOW”“ ALWAYSPAST”, “PAST” ή “YESTERDAY” και επιστρέφει την αντίστοιχη εγγραφή από τον enumerator TimeEnum.

private LogicalEnum getLogicalEnum(String s) Ελέγχει αν η συμβολοσειρά που διαβάζεται είναι λογικός τελεστής, δηλαδή αν είναι “OR”, “AND”,”NOTOR” ή “NOTAND” και επιστρέφει την αντίστοιχη εγγραφή από τον enumerator LogicalEnum.

Flag.java

Η κλάση Flag.java ουσιαστικά αποτελεί μία κλάση συντονιστή των νημάτων, μέσω της οποίας γίνεται η συγχρονισμένη επικοινωνία των πρακτόρων/ νημάτων με τη Βάση Δεδομένων.

synchronized void incNumthreads() Μέθοδος που απλά αυξάνει την τιμή μιας μεταβλητής που δρα ως μετρητής των νημάτων.

synchronized void decNumthreads() Μέθοδος που απλά μειώνει την τιμή μιας μεταβλητής που δρα ως μετρητής των νημάτων. Όταν ο αριθμός των νημάτων γίνει μηδέν υπάρχει η δυνατότητα διαγραφής των περιεχομένων της Βάσης Δεδομένων.

synchronized boolean search(TimeEnum time, String type) Μέθοδος η οποία ελέγχει τη Βάση Δεδομένων για την ύπαρξη εγγραφής η οποία, ανάλογα με το αν η μεταβλητή time είναι "YESTERDAY" ή "PAST" , αναφέρεται είτε στον προηγούμενο κύκλο είτε στους προηγούμενους κύκλους αντίστοιχα, και το πεδίο TYPE του πίνακα είναι ίδιο με το πεδίο type του ορίσματος της μεθόδου. Αν υπάρχει τέτοια εγγραφή στον πίνακα, η μέθοδος επιστρέφει *true.* Διαφορετικά *επιστρέφεται false.*

synchronized void writeDB(String apost, String typos) Μέθοδος κατά την κλήση της οποίας εγγράφονται δεδομένα στη Βάση Δεδομένων .

void printDB() Μέθοδος κατά την κλήση της οποίας εκτυπώνονται τα απότελέσματα του SQL ερωτήματος στην οθόνη του υπολογιστή .

private void deleteDB() Μέθοδος κατά την κλήση της οποίας σβήνονται τα περιεχόμενα του πίνακα από τη Βάση Δεδομένων .

synchronized void endOfRules() Μέθοδος κατά την κλήση της οποίας γίνεται συγχρονισμός των πρακτόρων/ νημάτων ανάμεσα στους κύκλους εκτέλεσης .

DoubleStatement.java

Η κλάση DoubleStatement περιέχει απλά δύο constructors βάσει των οποίων δομούνται τα μέρη των κανόνων ανάλογα με το αν αυτά αποτελούνται από ένα ή δύο μέρη.

Οι υπόλοιπες μέθοδοι που υπάρχουν στην κλάση είναι Getter μέθοδοι για τις private μεταβλητές της κλάσεις οι οποίες δημιουργήθηκαν αυτόματα κατά την εκτέλεση του προγράμματος.

LogicalEnum.java

Κλάση που περιέχει έναν enum τύπο για τους λογικούς τελεστές "OR", "AND", "NOTOR" και "NOTAND".

TimeEnum.java

Κλάση που περιέχει έναν enum τύπο για τους χρονικούς τελεστές “ALWAYSPAST”, “PAST”, “YESTERDAY”,”NOW”, “TOMORROW”, “FUTURE” και “ALWAYSFUTURE”.

Rule.java

Κλάση η οποία ορίζει κατά κάποιο τρόπο τη γενική μορφή ενός κανόνα, ότι δηλαδή αποτελείται από αριστερό και δεξί μέρος. Περιέχει επίσης μία μέθοδο η οποία εκτυπώνει τον κανόνα.

ToDoNode.java

Κλάση που χρησιμοποιείται για τον καθορισμό των ενεργειών που έχει να κάνει ο πράκτορας (δημιουργία κυρίως για το χειρισμό λίστας των ενεργειών που έχει ο πράκτορας να κάνει μελλοντικά).

MetatemAgent.java

Είναι η κλάση που περιέχει τη main(). Μέσα στη main() γίνεται η δημιουργία των νημάτων των πρακτόρων, δηλαδή η δημιουργία αντικειμένων τύπου Agent.

3.3 Σενάριο Εκτέλεσης Εφαρμογής

Στην παράγραφο αυτή παρουσιάζεται ένα παράδειγμα τρεξίματος της εφαρμογής . Στο συγκεκριμένο παράδειγμα παίρνουν μέρος τρεις πράκτορες , ο rc1, ο rc2 και ο rp. Για το λόγο αυτό έχουν γραφεί τρία αρχεία .txt που περιέχουν το καθένα το σύνολο των κανόνων κάθε πράκτορα. Παρακάτω παρατίθενται το σύνολο των κανόνων του κάθε πράκτορα.

Rc_1

start = NOW ask1 ;

YESTERDAY ask1 = NOW ask1

Rc_2

YESTERDAY (ask1 AND not ask2) = NOW ask2 ;

PAST not give2 = NOW ask3

Rp

YESTERDAY ask1 = NOW give1 ;

YESTERDAY ask2 = TOMORROW give2 ;

YESTERDAY ask3 = FUTURE give3

Βάσει των παραπάνω κανόνων το παιχνίδι εξελίσσεται όπως παρακάτω :

```
run:
-- end of cycle 0 with 3 threads running
-- end of cycle 1 with 3 threads running
-- end of cycle 2 with 3 threads running
-- end of cycle 3 with 3 threads running
-- end of cycle 4 with 3 threads running
-- end of cycle 5 with 3 threads running
-- end of cycle 6 with 3 threads running
I am thread 3 and i've finished!
I am thread 2 and i've finished!
I am thread 1 and i've finished!
CONTENTS OF DB:
rc1     ask1    0
rc1     ask1    1
rp      give1   1
rc2     ask2    1
rc2     ask3    1
rc2     ask3    2
rp      give1   2
rc1     ask1    2
rp      give2   3
rp      give3   3
rp      give1   3
rc1     ask1    3
rc2     ask2    3
rc2     ask3    3
rp      give3   4
rc1     ask1    4
rp      give1   4
```

```
rp      give2   5
rp      give3   5
rp      give1   5
rc2     ask2    5
rc1     ask1    5
rp      give1   6
rc1     ask1    6
```

ΠΑΡΑΤΗΡΗΣΕΙΣ

- Το παιχνίδι λήγει μετά το πέρας έξι κύκλων εκτέλεσης.
- Η πρώτη στήλη περιέχει τον πράκτορα αποστολέα του μηνύματος, η δεύτερη τον τύπο του μηνύματος και η τρίτη τον κύκλο στον οποίο λαμβάνει χώρα η αποστολή αυτού του μηνύματος.
- Η σειρά με τη οποία λειτουργούν οι πράκτορες δεν είναι σταθερή σε κάθε κύκλο. Αυτό που τηρείται είναι ότι σε κάθε κύκλο παίρνουν μέρος όλοι οι πράκτορες . Μόλις εκτελεστούν όλοι οι πράκτορες περνάμε στον επόμενο κύκλο.

Στη συνέχεια παρουσιάζονται τα περιεχόμενα της Βάσης Δεδομένων μετά το τέλος της εκτέλεσης της εφαρμογής.

#	ID	SENDER	TYPE	CYCLES
1	342	rc1	ask1	0
2	343	rc1	ask1	1
3	344	rp	give1	1
4	345	rc2	ask2	1
5	346	rc2	ask3	1
6	347	rc2	ask3	2
7	348	rp	give1	2
8	349	rc1	ask1	2
9	350	rp	give2	3
10	351	rp	give3	3
11	352	rp	give1	3
12	353	rc1	ask1	3
13	354	rc2	ask2	3
14	355	rc2	ask3	3
15	356	rp	give3	4
16	357	rc1	ask1	4
17	358	rp	give1	4
18	359	rp	give2	5
19	360	rp	give3	5
20	361	rp	give1	5

Τα πεδία του παραπάνω πίνακα είναι :

- ID : είναι το πρωτεύον κλειδί του πίνακα το οποίο αυξάνεται κατά ένα για κάθε νέα εγγραφή του πίνακα.

- SENDER : περιέχει το όνομα του πράκτορα/ αποστολέα.
- TYPE : περιέχει τον τύπο του μηνύματος που στέλνει ο πράκτορας/ αποστολέας.
- CYCLES : περιέχει τον αριθμό του κύκλο στον οποίο γίνεται η αποστολή του μηνύματος.

4 Βιβλιογραφία - Πηγές

1 Michael Wooldridge *«Εισαγωγή στα πολυπρακτορικά συστήματα»* (μετάφραση – επιστημονική επιμέλεια ελληνικής έκδοσης : Ασπασία Δασκαλοπούλου, Επίκουρος Καθηγήτρια Πανεπιστημίου Θεσσαλίας).

2 Michael Fisher , *"Towards a Semantics for Concurrent MetateM".*

3 Michael Fisher, *"A Survey of Concurrent MetateM – The Language and its Applications"*, Department of Computing, Manchester Metropolitan University, United Kingdom.

4 H. Barringer[1], M. Fisher[2], D. Gabbay[3], G. Gough[4] and R. Owens[5], *"Metatem: An Introduction"*

5 Ralf Kuhnel*, "Agent Oriented Programming with Java"*, Institut fur Informatik, Humboldt Universitat Berlin

6 H. Barringer[1], M. Fisher[2], D. Gabbay[3], G. Gough[4] and R. Owens[5], "*MetateM : A Framework for Programming in Temporal Logic*", Department of Computer Science, University of Manchester, Manchester, Department of Computing, Imperial College of Science, Technology and Medicine, London.

7 H. Barringer, A. Hunter, M. Fisher, P. McBrien, M. Reynolds, D. Brough, D. Gabbay, G. Gough and R. Owens, *"Languages, Meta – languages and MetateM, a discussion Paper".*

8 Michael Fisher, Anthony Hepple, "Executing Logical Agent Specifications", Department of Computer Science, University of Liverpool, U.K..

9 Παρουσιάσεις μαθήματος «Συστήματα Πρακτόρων», Ασπασία Δασκαλοπούλου, Επίκουρος Καθηγήτρια Πανεπιστημίου Θεσσαλίας http://inf-server.inf.uth.gr/courses/CE454

10 Skolem normal form , Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Skolem_normal_form

11 Michael Fisher, "MetateM: The Story so Far", Department of Computer Science, University of Liverpool, United Kingdom.

12 Katia P. Sycara, "Multiagent Systems", American Association for Artificial Intelligence, summer 1998.

5 Παράρτημα

Σε αυτό το σημείο παραθέτουμε τον κώδικα της εφαρμογής.

Class MetatemAgent.java

```
/************************* Agent ************************/
package metatemagent;

import java.io.BufferedReader;
import java.io.FileReader;
import java.util.ArrayList;

/**
 *
 * @author Voula
 */
public class Agent extends Thread {

    private int id;
    private String type;
    Flag flag;
    ArrayList<Rule> rules;
    DoubleStatement start;
    ArrayList<ToDoNode> todotomorrow;
    ArrayList<ToDoNode> todofuture;
    String rulefile;

    Agent(int threadId, Flag f, String t, String file) {
```

```
        id = threadId;
        flag = f;
        type = t;
        rules = new ArrayList<Rule>();
        rulefile = file;
        todotomorrow = new ArrayList<ToDoNode>();
        todofuture = new ArrayList<ToDoNode>();
        fillRules();
        this.start();
    }

    private void fillRules() {
        StringBuilder stringBuilder = null;
        try {
            BufferedReader reader = new BufferedReader(new FileReader(rulefile));
            String line = null;
            stringBuilder = new StringBuilder();
            String ls = System.getProperty("line.separator");
            while ((line = reader.readLine()) != null) {
                stringBuilder.append(line);
                stringBuilder.append(ls);
            }

        } catch (Exception e) {
            System.err.println(e.getMessage());
        }

        String res = stringBuilder.toString();
        String[] result = res.split(";");

        int d = result.length;
```

```
        for (int x = 0; x < result.length; x++) {


            int an = analyze(result[x]);

            //System.out.println(type + " -- Result for x = " + x + " : " + an);

            String sn = result[x].trim();

            String[] sent = sn.split("\\s+");

            if (an == 0) {

                start = new DoubleStatement(getTimeEnum(sent[2]), false, sent[3]);

            } else if (an == 1) {

                Rule r = new Rule();

                r.left = new DoubleStatement(getTimeEnum(sent[0]), true, sent[2]);

                r.right = new DoubleStatement(getTimeEnum(sent[4]), false,
sent[5]);

                rules.add(r);

            } else if (an == 2) {

                Rule r = new Rule();

                r.left = new DoubleStatement(getTimeEnum(sent[0]), false, sent[1]);

                r.right = new DoubleStatement(getTimeEnum(sent[3]), false,
sent[4]);

                rules.add(r);

            } else if (an == 3) {

                Rule r = new Rule();

                r.left = new DoubleStatement(getTimeEnum(sent[0]), true, sent[2],
getTimeEnum(sent[4]), true, sent[6], getLogicalEnum(sent[3]));

                r.right = new DoubleStatement(getTimeEnum(sent[8]), false,
sent[9]);

                rules.add(r);

            } else if (an == 4) {

                Rule r = new Rule();

                r.left = new DoubleStatement(getTimeEnum(sent[0]), false, sent[1],
getTimeEnum(sent[3]), true, sent[5], getLogicalEnum(sent[2]));

                r.right = new DoubleStatement(getTimeEnum(sent[7]), false,
sent[8]);

                rules.add(r);
```

```
            } else if (an == 5) {

                Rule r = new Rule();

                r.left = new DoubleStatement(getTimeEnum(sent[0]), true, sent[2],
getTimeEnum(sent[3]), false, sent[5], getLogicalEnum(sent[3]));

                r.right = new DoubleStatement(getTimeEnum(sent[7]), false,
sent[8]);

                rules.add(r);

            } else if (an == 6) {

                Rule r = new Rule();

                r.left = new DoubleStatement(getTimeEnum(sent[0]), false, sent[1],
getTimeEnum(sent[3]), false, sent[4], getLogicalEnum(sent[2]));

                r.right = new DoubleStatement(getTimeEnum(sent[6]), false,
sent[7]);

                rules.add(r);

            } else if (an == 7) {

                Rule r = new Rule();

                r.left = new DoubleStatement(getTimeEnum(sent[0]), false, sent[2],
getTimeEnum(sent[0]), false, sent[4], getLogicalEnum(sent[3]));

                r.right = new DoubleStatement(getTimeEnum(sent[7]), false,
sent[8]);

                rules.add(r);

            } else if (an == 8) {

                Rule r = new Rule();

                r.left = new DoubleStatement(getTimeEnum(sent[0]), true, sent[3],
getTimeEnum(sent[0]), false, sent[5], getLogicalEnum(sent[4]));

                r.right = new DoubleStatement(getTimeEnum(sent[8]), false,
sent[9]);

                rules.add(r);

            } else if (an == 9) {

                Rule r = new Rule();

                r.left = new DoubleStatement(getTimeEnum(sent[0]), false, sent[2],
getTimeEnum(sent[0]), true, sent[5], getLogicalEnum(sent[3]));

                r.right = new DoubleStatement(getTimeEnum(sent[8]), false,
sent[9]);

                rules.add(r);

            } else if (an == 10) {
```

```
                Rule r = new Rule();

                r.left = new DoubleStatement(getTimeEnum(sent[0]), true , sent[3],
getTimeEnum(sent[0]), true, sent[6], getLogicalEnum(sent[4]));

                r.right = new DoubleStatement(getTimeEnum(sent[9]), false,
sent[10]);

                rules.add(r);

            }

        }


        //System.out.println("Teleiwsa thn analysh!");

    }


    @Override

    public void run() {

        if (start != null && start.getTime1() == TimeEnum.NOW) {

            flag.writeDB(type, start.getMsgtype1());


            flag.endOfRules();

        } else {


            flag.nothingToDo();

        }

        for (int i = 0; i < 6; i++) {

            boolean ret1 = false;

            boolean ret2 = false;

            boolean logical = false;


            // do everything from todotomorrow


            for (int j = 0; j < todotomorrow.size(); j++) {


                flag.writeDB(todotomorrow.get(j).getAgenttype(),
todotomorrow.get(j).getMsgtype());
```

```
            }
            todotomorrow.clear();

            // do everything i can from todofuture

            for (int j = 0; j < todofuture.size(); j++) {
                flag.writeDB(todofuture.get(j).getAgenttype(),
todofuture.get(j).getMsgtype());
            }
            todofuture.clear();

            for (Rule r : rules) {
                if (r.left.isHave2()) {
                    // double statement
                    ret1 = flag.search(r.left.getTime1(), r.left.getMsgtype1());
                    ret2 = flag.search(r.left.getTime2(), r.left.getMsgtype2());

                    if (r.left.isIsNegative1()) {
                        ret1 = !ret1;
                    }
                    if (r.left.isIsNegative2()) {
                        ret2 = !ret2;
                    }

                    if (r.left.getLink() == LogicalEnum.AND) {
                        logical = ret1 & ret2;
                    } else if (r.left.getLink() == LogicalEnum.OR) {
                        logical = ret1 | ret2;
                    } else if (r.left.getLink() == LogicalEnum.NOTAND) {
                        logical = !(ret1 & ret2);
                    } else if (r.left.getLink() == LogicalEnum.NOTOR) {
                        logical = !(ret1 | ret2);
```

```
                    }

                    if (logical) {
                        if (r.right.getTime1() == TimeEnum.NOW) {
                            flag.writeDB(type, r.right.getMsgtype1());
                        } else if (r.right.getTime1() == TimeEnum.TOMORROW) {
                            // enqueue in todotomorrow

                            todotomorrow.add(new ToDoNode(type,
r.right.getMsgtype1()));
                        } else if (r.right.getTime1() == TimeEnum.FUTURE) {
                            // enqueue in todofuture
                            todofuture.add(new ToDoNode(type,
r.right.getMsgtype1()));
                        }
                    }


                } else {
                    // simple statement
                    ret1 = flag.search(r.left.getTime1(), r.left.getMsgtype1());

                    if (r.left.isIsNegative1()) {
                        ret1 = !ret1;
                    }
                    if (ret1) {
                        if (r.right.getTime1() == TimeEnum.NOW) {

                            flag.writeDB(type, r.right.getMsgtype1());
                        } else if (r.right.getTime1() == TimeEnum.TOMORROW) {
                            // enqueue in todotomorrow
```

```
                            todotomorrow.add(new ToDoNode(type,
r.right.getMsgtype1()));
                        } else if (r.right.getTime1() == TimeEnum.FUTURE) {
                            // enqueue in todofuture
                            todofuture.add(new ToDoNode(type,
r.right.getMsgtype1()));
                        }
                    }
                }
            }

            flag.endOfRules();
        }
        System.out.println("I am thread " + id + " and i've finished!");
        flag.decNumthreads();
    }
    private int analyze(String s) {
        boolean ok = true;
        String sn = s.trim();
        String[] sent = sn.split("\\s+"); /*"spaei" thn protash se lexeis*/

        /*****************PRWTH PROTASH******************/
        if (sent[0].equals("start")) {
            if (!sent[1].equals("=")) {
                ok = false;
            } else {
                if (!isFuture(sent[2])) {
                    ok = false;
                } else {
                    if (!(sent[3].equals("not") || isPast(sent[3]) ||
isFuture(sent[3]) || isLog(sent[3]) || sent[3].equals(" "))) {

                        return 0;
```

```
                }

            }

        }

        /*********************************************/

        /************ DEUTERH PROTASH + NOT ************/

        /* time not kati = time palikatiallo*/

        if (isPast(sent[0])) {

            if (!sent[1].equals("not")) {

                ok = false;

            } else {

                if (sent[2].equals("not") || isPast(sent[2]) || isFuture(sent[2])
|| isLog(sent[2]) || sent[2].equals(" ")) {

                    ok = false;

                } else {

                    if (!sent[3].equals("=")) {

                        ok = false;

                    } else {

                        if (!isFuture(sent[4])) {

                            ok = false;

                        } else {

                            if (!(sent[5].equals("not") || isPast(sent[5]) ||
isFuture(sent[5]) || isLog(sent[5]) || sent[5].equals(" "))) {

                                return 1;

                            }

                        }

                    }

                }

            }



        }
```

```
        /*********************************************/

        /************ DEUTERH PARAGWGH XWRIS NOT **************/

        /* time kati = time palikatiallo*/

        if (isPast(sent[0])) {

            if (sent[1].equals("not") || isPast(sent[1]) || isFuture(sent[1]) ||
isLog(sent[1]) || sent[1].equals(" ")) {

                ok = false;

            } else {

                if (!sent[2].equals("=")) {

                    ok = false;

                } else {

                    if (!isFuture(sent[3])) {

                        ok = false;

                    } else {

                        if (!(sent[4].equals("not") || isPast(sent[4]) ||
isFuture(sent[4]) || isLog(sent[4]) || sent[4].equals(" "))) {

                            return 2;

                        }

                    }

                }

            }

        }







        /*****************************************************/

        /************** 3 TRITH PARAGWGH ***********************/

        /*  time not kati logikostelesths time not katiallo =  time palikatiallo
*/

        if (isPast(sent[0])) {

            if (!sent[1].equals("not")) {

                ok = false;

            } else {
```

```
                if (sent[2].equals("not") || isPast(sent[2]) || isFuture(sent[2])
|| isLog(sent[2]) || sent[2].equals(" ")) {

                    ok = false;

                } else {

                    if (!isLog(sent[3])) {

                        ok = false;

                    } else {

                        if (!isPast(sent[4])) {

                            ok = false;

                        } else {

                            if (!sent[5].equals("not")) {

                                ok = false;

                            } else {

                                if (sent[6].equals("not") || isPast(sent[6]) ||
isFuture(sent[6]) || isLog(sent[6]) || sent[6].equals(" ")) {

                                    ok = false;

                                } else {

                                    if (!sent[7].equals("=")) {

                                        ok = false;

                                    } else {

                                        if (!isFuture(sent[8])) {

                                            ok = false;

                                        } else {

                                            if (!(sent[9].equals("not") ||
isPast(sent[9]) || isFuture(sent[9]) || isLog(sent[9]) || sent[9].equals(" "))) {

                                                return 3;

                                            }

                                        }

                                    }

                                }

                            }

                        }

                    }
```

```
                }

            }

        }


        /************************************************/

        /************* 4 TRITH PARAGWGH ver.2 ***********************/

        /*  time kati logikostelesths time not katiallo = time palikatiallo    */

        if (isPast(sent[0])) {

            if (sent[1].equals("not") || isPast(sent[1]) || isFuture(sent[1]) ||
isLog(sent[1]) || sent[1].equals(" ")) {

                ok = false;

            } else {

                if (!isLog(sent[2])) {

                    ok = false;

                } else {

                    if (!isPast(sent[3])) {

                        ok = false;

                    } else {

                        if (!sent[4].equals("not")) {

                            ok = false;

                        } else {

                            if (sent[5].equals("not") || isPast(sent[5]) ||
isFuture(sent[5]) || isLog(sent[5]) || sent[5].equals(" ")) {

                                ok = false;

                            } else {

                                if (!sent[6].equals("=")) {

                                    ok = false;

                                } else {

                                    if (!isFuture(sent[7])) {

                                        ok = false;

                                    } else {

                                        if (!(sent[8].equals("not") ||
isPast(sent[8]) || isFuture(sent[8]) || isLog(sent[8]) || sent[8].equals(" "))) {
```

```
                                        return 4;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
        /*************************************************************/
        /************** 5 TRITH PARAGWGH ver.3 ***********************/
        /*  time not kati logikostelesths time  katiallo = time palikatiallo    */
        if (isPast(sent[0])) {
            if (!sent[1].equals("not")) {
                ok = false;
            } else {
                if (sent[2].equals("not") || isPast(sent[2]) || isFuture(sent[2])
|| isLog(sent[2]) || sent[2].equals(" ")) {
                    ok = false;
                } else {
                    if (!isLog(sent[3])) {
                        ok = false;
                    } else {
                        if (!isPast(sent[4])) {
                            ok = false;
                        } else {
                            if (sent[5].equals("not") || isPast(sent[5]) ||
isFuture(sent[5]) || isLog(sent[5]) || sent[5].equals(" ")) {
                                ok = false;
                            } else {
                                if (!sent[6].equals("=")) {
```

```
                                    ok = false;
                                } else {
                                    if (!isFuture(sent[7])) {
                                        ok = false;
                                    } else {
                                        if (!(sent[8].equals("not") ||
isFuture(sent[8]) || isLog(sent[8]) || sent[8].equals(" "))) {
                                            return 5;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }

        /*********************************************************************/
        /************** 6 TRITH PARAGWGH ver.4 ***********************/
        /*  time  kati logikostelesths time katiallo = time palikatiallo    */
        if (isPast(sent[0])) {
            if (sent[1].equals("not") || isPast(sent[1]) || isFuture(sent[1]) ||
isLog(sent[1]) || sent[1].equals(" ")) {
                ok = false;
            } else {
                if (!isLog(sent[2])) {
                    ok = false;
                } else {
                    if (!isPast(sent[3])) {
                        ok = false;
                    } else {
```

```
                        if (sent[4].equals("not") || isPast(sent[4]) ||
isFuture(sent[4]) || isLog(sent[4]) || sent[4].equals(" ")) {

                            ok = false;

                        } else {

                            if (sent[5].equals("=")) {

                                ok = false;

                            } else {

                                if (!isFuture(sent[6])) {

                                    ok = false;

                                } else {

                                    if (!(sent[7].equals("not") ||
isFuture(sent[7]) || isLog(sent[7]) || sent[7].equals(" "))) {

                                        return 6;

                                    }

                                }

                            }

                        }

                    }

                }

            }

        }

        /*************************************************************/

        /************* 7 PEMPTH PARAGWGH ver.1 ***********************/

        /*  time ( kati logikostelesths katiallo ) =  time palikatiallo    */

        if (isPast(sent[0])) {


            if (!sent[2].equals("(")) {

                ok = false;

            } else {

                if (sent[3].equals("not") || isPast(sent[3]) || isFuture(sent[3])
|| isLog(sent[3]) || sent[3].equals(" ")) {

                    ok = false;

                } else {
```

```
                    if (!isLog(sent[4])) {

                        ok = false;

                    } else {

                        if (sent[5].equals("not") || isPast(sent[5]) ||
isFuture(sent[5]) || isLog(sent[5]) || sent[5].equals(" ")) {

                            ok = false;

                        } else {

                            if (!sent[6].equals(")")) {

                                ok = false;

                            } else {

                                if (!isFuture(sent[7])) {

                                    ok = false;

                                } else {

                                    if (!(sent[8].equals("not") || isPast(sent[8])
|| isFuture(sent[8]) || isLog(sent[8]) || sent[8].equals(" "))) {

                                        return 7;

                                    }

                                }

                            }

                        }

                    }

                }

            }

        }


        /*********************************************************************/

        /************** 8 PEMPTH PARAGWGH ver.2 ***********************/

        /*  time ( not kati logikostelesths katiallo ) =  time palikatiallo    */

        if (isPast(sent[0])) {


            if (!sent[1].equals("(")) {

                ok = false;
```

```
            } else {
                if (!sent[2].equals("not")) {
                    ok = false;
                } else {
                    if (sent[3].equals("not") || isPast(sent[3]) ||
isFuture(sent[3]) || isLog(sent[3]) || sent[3].equals(" ")) {
                        ok = false;
                    } else {
                        if (!isLog(sent[4])) {
                            ok = false;
                        } else {
                            if (sent[5].equals("not") || isPast(sent[5]) ||
isFuture(sent[5]) || isLog(sent[5]) || sent[5].equals(" ")) {
                                ok = false;
                            } else {
                                if (!sent[6].equals(")")) {
                                    ok = false;
                                } else {
                                    if (!sent[7].equals("=")) {
                                        ok = false;
                                    } else {
                                        if (!isFuture(sent[8])) {
                                            ok = false;
                                        } else {
                                            if (!(sent[9].equals("not") ||
isPast(sent[9]) || isFuture(sent[9]) || isLog(sent[9]) || sent[9].equals(" "))) {
                                                return 8;
                                            }
                                        }
                                    }
                                }
                            }
                        }
```

```
                }
            }
        }
    }

    /*********************************************************************/
    /************** 9 PEMPTH PARAGWGH ver.3 ***********************/
    /*  time ( kati logikostelesths not katiallo ) = time palikatiallo    */
    if (isPast(sent[0])) {
        if (!sent[1].equals("(")) {
            ok = false;
        } else {

            if (sent[2].equals("not") || isPast(sent[2]) || isFuture(sent[2])
|| isLog(sent[2]) || sent[2].equals(" ")) {
                ok = false;
            } else {
                if (!isLog(sent[3])) {
                    ok = false;
                } else {
                    if (!sent[4].equals("not")) {
                        ok = false;
                    } else {
                        if (sent[5].equals("not") || isPast(sent[5]) ||
isFuture(sent[5]) || isLog(sent[5]) || sent[5].equals(" ")) {
                            ok = false;
                        } else {
                            if (!sent[6].equals(")")) {
                                ok = false;
                            } else {
                                if (!sent[7].equals("=")) {
                                    ok = false;
```

```
                                    } else {
                                        if (!isFuture(sent[8])) {
                                            ok = false;
                                        } else {
                                            if (!(sent[9].equals("not") ||
isPast(sent[9]) || isFuture(sent[9]) || isLog(sent[9]) || sent[9].equals(" "))) {
                                                return 9;
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

        /*****************************************************************/
        /************** 10 PEMPTH PARAGWGH ver.4 ***********************/
        /*  time ( not kati logikostelesths not katiallo ) = time palikatiallo
*/
        if (isPast(sent[0])) {

            if (!sent[1].equals("(")) {
                ok = false;
            } else {
                if (sent[2].equals("not")) {
                    ok = false;
                } else {
                    if (sent[3].equals("not") || isPast(sent[3]) ||
isFuture(sent[3]) || isLog(sent[3]) || sent[3].equals(" ")) {
                        ok = false;
```

```
                    } else {
                        if (!isLog(sent[4])) {
                            ok = false;
                        } else {
                            if (!sent[5].equals("not")) {
                                ok = false;
                            } else {
                                if (sent[6].equals("not") || isPast(sent[6]) ||
isFuture(sent[6]) || isLog(sent[6]) || sent[6].equals(" ")) {
                                    ok = false;
                                } else {
                                    if (!sent[7].equals(")")) {
                                        ok = false;
                                    } else {
                                        if (!sent[8].equals("=")) {
                                            ok = false;
                                        } else {
                                            if (!isFuture(sent[9])) {
                                                ok = false;
                                            } else {
                                                if (!(sent[10].equals("not") ||
isPast(sent[10]) || isFuture(sent[10]) || isLog(sent[10]) || sent[10].equals(" ")))
{
                                                    return 10;
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
```

```
        }
    }

    /*********************************************************************/

    return -1;
}
private boolean isPast(String s) {
    if (s.equals("ALWAYSPAST")) {
        return true;
    } else if (s.equals("PAST")) {
        return true;
    } else if (s.equals("YESTERDAY")) {
        return true;
    } else {
        return false;
    }
}

private boolean isFuture(String s) {
    if (s.equals("TOMORROW")) {
        return true;
    } else if (s.equals("FUTURE")) {
        return true;
    } else if (s.equals("ALWAYSFUTURE")) {
        return true;
    } else if (s.equals("NOW")) {
        return true;
    } else {
        return false;
    }
}
```

```
private boolean isLog(String s) {
    if (s.equals("OR")) {
        return true;
    } else if (s.equals("AND")) {
        return true;
    } else if (s.equals("NOTOR")) {
        return true;
    } else if (s.equals("NOTAND")) {
        return true;
    } else {
        return false;
    }
}

private TimeEnum getTimeEnum(String s) {
    if (s.equals("YESTERDAY")) {
        return TimeEnum.YESTERDAY;
    } else if (s.equals("PAST")) {
        return TimeEnum.PAST;
    } else if (s.equals("ALWAYSPAST")) {
        return TimeEnum.ALWAYSPAST;
    } else if (s.equals("FUTURE")) {
        return TimeEnum.FUTURE;
    } else if (s.equals("NOW")) {
        return TimeEnum.NOW;
    } else if (s.equals("TOMORROW")) {
        return TimeEnum.TOMORROW;
    } else {
        return TimeEnum.ALWAYSFUTURE;
    }
}
```

```
    private LogicalEnum getLogicalEnum(String s) {
        if (s.equals("AND")) {
            return LogicalEnum.AND;
        } else if (s.equals("OR")) {
            return LogicalEnum.OR;
        } else if (s.equals("NOTAND")) {
            return LogicalEnum.NOTAND;
        } else {
            return LogicalEnum.NOTOR;
        }
    }
}
```

Flag.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
/*"Epikoinwnia" me th vash kai ousiastika sugxronismos nhmatwn*/
/*********************************************/
/************* M O N I T O R *****************/
/*********************************************/
package metatemagent;


/**
 *
 * @author Voula
 */
import java.sql.*;
import java.util.Properties;
import java.util.logging.Level;
```

```
import java.util.logging.Logger;


public class Flag {


    int cycle;

    int alldone;

    int numthreads;

    Connection con;

    String strUrl;

    Properties props;

    Statement sta;

    ResultSet res;


    Flag() {

        cycle = 0;

        alldone = 0;

        numthreads = 0;

        strUrl = "jdbc:derby://localhost:1527/dialog DB";

        props = new Properties();

        props.put("user", "voula");

        props.put("password", "voula");

        try {

            con = DriverManager.getConnection(strUrl, props);

            sta = con.createStatement();

        } catch (SQLException ex) {

            Logger.getLogger(Flag.class.getName()).log(Level.SEVERE, null, ex);

        }


    }


    synchronized void incNumthreads() {

        numthreads++;
```

```
    }

    synchronized void decNumthreads() {
        numthreads--;
        if (numthreads == 0) {
            printDB();
            //deleteDB();
        }
    }

    synchronized boolean search(TimeEnum time, String type) {

        if (time == TimeEnum.YESTERDAY) {
            try {
                res = sta.executeQuery("SELECT SENDER FROM APP.MSG WHERE CYCLES = "
+ (cycle - 1) + " AND TYPE LIKE '" + type + "'");
            } catch (SQLException ex) {
                Logger.getLogger(Flag.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        if (time == TimeEnum.PAST) {
            try {
                res = sta.executeQuery("SELECT SENDER FROM APP.MSG WHERE CYCLES < "
+ cycle + " AND TYPE LIKE '" + type + "'");
            } catch (SQLException ex) {
                Logger.getLogger(Flag.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        try {
            if (res.next() ) {
                return true;
            } else {
```

```
                return false;
            }
        } catch (SQLException ex) {
            Logger.getLogger(Flag.class.getName()).log(Level.SEVERE, null, ex);
        }
        return false;
    }

    synchronized void writeDB(String apost, String typos) {

        try {
            int result;
            result = sta.executeUpdate("INSERT INTO APP.MSG (SENDER,TYPE,CYCLES)" +
" VALUES" + "('" + apost + "','" + typos + "'," + cycle + ")");
        } catch (SQLException ex) {
            Logger.getLogger(Flag.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    synchronized void endOfRules()
    {
        alldone++;
        if (alldone < numthreads) {
            try {
                wait();
            } catch (InterruptedException e) {
                System.out.println("wait interrupted: " + e);
            }
        } else {
            System.out.println("-- end of cycle " + cycle + " with " + numthreads +
" threads running");
            alldone = 0;
```

```
            cycle++;
            notifyAll();
        }
    }

    synchronized void nothingToDo() {
        alldone++;

        if (alldone < numthreads) {
            try {
                wait();
            } catch (InterruptedException e) {
                System.out.println("wait interrupted: " + e);
            }
        } else {
            alldone = 0;
            cycle++;
            notifyAll();
        }
    }

    void printDB() {
        try {
            res = sta.executeQuery("SELECT * FROM APP.MSG ");
            System.out.println("CONTENTS OF DB:");
            while (res.next()) {
                System.out.println(res.getString("SENDER") + "\t" +
res.getString("TYPE") + "\t" + res.getString("CYCLES"));
            }
        } catch (SQLException ex) {
            Logger.getLogger(Flag.class.getName()).log(Level.SEVERE, null, ex);
        }
```

```
    }

    private void deleteDB() {
        System.out.println("\n NOW DELETING ALL CONTENTS OF DB \n");
        try {
            int result;
            result = sta.executeUpdate("DELETE FROM APP.MSG");
        } catch (SQLException ex) {
            Logger.getLogger(Flag.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

DoubleStatement.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package metatemagent;
/**
 *
 * @author Voula
 */
public class DoubleStatement {
    private TimeEnum time1;
    private boolean isNegative1;
    private String msgtype1;
    private boolean have2;
    private TimeEnum time2;
    private boolean isNegative2;
```

```
    private String msgtype2;
    LogicalEnum link;

    public DoubleStatement(TimeEnum time1, boolean isNegative1, String msgtype1) {
        this.time1 = time1;
        this.isNegative1 = isNegative1;
        this.msgtype1 = msgtype1;
        this.time2 = null;
        this.isNegative2 = false;
        this.msgtype2 = null;
        this.link = null;
        this.have2 = false;

    }

    public DoubleStatement(TimeEnum time1, boolean isNegative1, String msgtype1,
TimeEnum time2, boolean isNegative2, String msgtype2, LogicalEnum link) {
        this.time1 = time1;
        this.isNegative1 = isNegative1;
        this.msgtype1 = msgtype1;
        this.time2 = time2;
        this.isNegative2 = isNegative2;
        this.msgtype2 = msgtype2;
        this.link = link;
        this.have2 = true;

    }

    public boolean isHave2() {
        return have2;
    }
    public boolean isIsNegative1() {
```

```
        return isNegative1;
    }
    public boolean isIsNegative2() {
        return isNegative2;
    }
    public LogicalEnum getLink() {
        return link;
    }
    public String getMsgtype1() {
        return msgtype1;
    }
    public String getMsgtype2() {
        return msgtype2;
    }
    public TimeEnum getTime1() {
        return time1;
    }
    public TimeEnum getTime2() {
        return time2;
    }
}
```

LogicalEnum.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package metatemagent;
/**
 *
```

```
 * @author Voula
 */
public enum LogicalEnum {
    OR,AND,NOTOR,NOTAND;
}
```

TimeEnum.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package metatemagent;

/**
 *
 * @author Voula
 */
public enum TimeEnum {
    ALWAYSPAST,PAST,YESTERDAY,NOW,TOMORROW,FUTURE,ALWAYSFUTURE;
}
```

MetatemAgent.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package metatemagent;
import java.sql.*;
import java.util.Properties;
/**
 *
```

```
 * @author Voula
 */
public class MetatemAgent {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Flag f = new Flag();
        f.incNumthreads();
        Agent t3 = new
Agent(3,f,"rp","C:\\Users\\Voula\\Desktop\\TELOOOS\\rp.txt");

        f.incNumthreads();
        Agent t1 = new
Agent(1,f,"rc1","C:\\Users\\Voula\\Desktop\\TELOOOS\\rc1.txt");

        f.incNumthreads();
        Agent t2 = new
Agent(2,f,"rc2","C:\\Users\\Voula\\Desktop\\TELOOOS\\rc2.txt");
    }
}
```

Rule.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package metatemagent;

/**
 *
 * @author Voula
 */
```

```
public class Rule {

    public DoubleStatement left;

    public DoubleStatement right;


    public Rule(){

    }

    public void printRule()

    {

        System.out.println(left.getTime1());

        System.out.println(left.isIsNegative1());

        System.out.println(left.getMsgtype1());

        System.out.println(left.getLink());

        System.out.println(left.getTime2());

        System.out.println(left.isIsNegative2());

        System.out.println(left.getMsgtype2());

        System.out.println("-->");

        System.out.println(right.getTime1());

        System.out.println(right.isIsNegative1());

        System.out.println(right.getMsgtype1());


    }

}
```

ToDoNode.java

```
/*

 * To change this template, choose Tools | Templates

 * and open the template in the editor.

 */

package metatemagent;


/**
```

```
 *
 * @author Voula
 */
public class ToDoNode {

    private String agenttype;
    private String msgtype;

    ToDoNode(String at, String mt) {
        agenttype = at;
        msgtype = mt;
    }

    public String getAgenttype() {
        return agenttype;
    }

    public String getMsgtype() {
        return msgtype;
    }
}
```