Πανεπιστήμιο Μακεδονίας
Πρόγραμμα Μεταπτυχιακών Σπουδών Ειδίκευσης
στην Εφαρμοσμένη Πληροφορική

ΑΝΑΠΤΥΞΗ ΕΥΦΥΟΥΣ ΣΥΣΤΗΜΑΤΟΣ ΠΡΟΣΩΠΙΚΟΥ ΣΧΕΔΙΑΣΜΟΥ ΜΕ ΚΙΝΗΤΗ ΔΙΑΣΥΝΔΕΣΗ

Διπλωματική εργασία του φοιτητή:
Ανδρέα Χριστοφόρου 56/04

Επίβλεψη:
Ρεφανίδης Ιωάννης

Θεσσαλονίκη, 2006

Αφιέρωση

Η συγγραφή της παρούσα εργασίας σίγουρα δεν ήταν ένα εύκολο και πάντα ευχάριστο έργο. Για να ολοκληρωθεί και να φτάσει στην τελική της μορφή χρειάστηκαν αρκετές ώρες έρευνας και πρακτικής εφαρμογής της θεωρίας, αλλά και συνάμα μεράκι και αγάπη για το συγκεκριμένο αντικείμενο. Για τον λόγο αυτό θα ήθελα να ευχαριστήσω πρώτα τους γονείς μου για την αμέριστη ψυχική και υλική συμπαράσταση, χωρίς την οποία δεν θα μπορούσα να φτάσω εδώ που βρίσκομαι. Έπειτα θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Γιάννη Ρεφανίδη, ό οποίος με χρήσιμες συμβουλές και με την επιστημονική του γνώση βοήθησε τα μέγιστα στην περάτωση της εργασίας αυτής. Τέλος θα ήθελα να ευχαριστήσω όλους τους καθηγητές του Προγράμματος Μεταπτυχιακών Σπουδών του Τμήματος Εφαρμοσμένης Πληροφορικής του Πανεπιστημίου Μακεδονίας που συνεισέφεραν δύο χρόνια τώρα με επιμονή και υπομονή στην διαμόρφωση της επιστημονικής μου κατάρτισης.

1 ΕΙΣΑΓΩΓΗ

1.1 ΠΕΡΙΓΡΑΦΗ ΠΡΟΒΛΗΜΑΤΟΣ

Η σύγχρονη εποχή με τον γρήγορο ρυθμό της ανάγκασε πολλούς ανθρώπους να χρησιμοποιούν ατζέντες, μικρά σημειωματάρια, για την καταγραφή και υπενθύμιση των εργασιών που πρέπει να εκτελέσουν. Με την εισχώρηση των υπολογιστών και μικροσυσκευών στην ζωή μας εμφανίστηκαν ηλεκτρονικές ατζέντες που πλεονεκτούν των πατροπαράδοτων ατζέντων κυρίως λόγω της ευκολίας χρήσης, ταχύτητας διαχείρισης και μεγάλου όγκου πληροφορίας που μπορούν να αποθηκεύουν. Ειδικότερα οι ατζέντες που βρίσκονται σε μικροσυσκευές, και καθώς οι μικροσυσκευές αποκτούν όλο και περισσότερη επεξεργαστική ισχύ, είναι η φυσική συνέχεια των πατροπαράδοτων ατζέντων.

Μια ηλεκτρονική ατζέντα εκτός από τις κλασσικές λειτουργίες, όπως εισαγωγή εργασιών και εμφάνιση εργασιών με διάφορους τρόπους στον χρήστη, μπορούν επίσης να κάνουν προτάσεις για επίλυση συγκρούσεων μεταξύ εργασιών και να υπενθυμίσουν τον χρήστη για επερχόμενες εργασίες με ηχητικές ή/και γραφικές υπενθυμίσεις. Παρόλα όμως τα πλεονεκτήματα των ηλεκτρονικών ατζέντων, δεν έχουν ακόμα εκμεταλλευτεί πλήρως τις δυνατότητες των μικροσυσκευών και των εξελίξεων στην τεχνητή νοημοσύνη.

Μια ατζέντα του μέλλοντος θα πρέπει εκτός από τις κλασσικές λειτουργίες της να μπορεί να εκτελέσει κάποιες από τις εργασίες ή υποεργασίες κάποιας εργασίας, όπως αποστολή email για υπενθύμιση συναντήσεων, αγορές από το διαδίκτυο κ.α. Επίσης εκτός από την ενημέρωση του χρήστη για συγκρούσεις πρέπει να είναι σε θέση να παρουσιάσει εναλλακτικές προτάσεις που αφορούν όχι μόνο εργασίες που συγκρούονται αλλά και προτάσεις για μετακίνηση εργασιών που βελτιώνουν το πρόγραμμα του χρήστη.

Για παράδειγμα θα μπορούσε να προτείνει στον χρήστη την μετακίνηση κάποιας εργασίας σε άλλη μέρα και ώρα για να εκτελεστεί μαζί με κάποια άλλη εργασία που ο χρήστης θα εκτελέσει σε μια τοποθεσία που είναι κοντά στην τοποθεσία της αρχικής εργασίας, μειώνοντας έτσι τον χρόνο μετακινήσεων. Μία ατζέντα επίσης θα μπορούσε να ενσωματώσει τεχνικές μηχανικής μάθησης, μαθαίνοντας έτσι την συμπεριφορά του χρήστη σε βάθος χρόνου, ούτως ώστε να προσαρμόσει τις προτάσεις της στις προτιμήσεις του χρήστη.

Μερικά από τα πλεονεκτήματα τέτοιων συστημάτων είναι:

- Η ελάφρυνση των προγραμμάτων των χρηστών δημιουργώντας βέλτιστα πλάνα.
- Η αυτόματη εκτέλεση εκ μέρους του συστήματος κάποιων ενεργειών (π.χ. αγορά αεροπορικών εισιτηρίων).
- Η οικονομία χρημάτων εκ μέρους του χρήστη εκμεταλλευόμενο πληροφορίες από το διαδίκτυο (π.χ. σύγκριση τιμών).
- Ο συντονισμός ενεργειών μεταξύ χρηστών.

Στην παρούσα εργασία θα παρουσιάσουμε ένα πρότυπο σύστημα παροχής υπηρεσιών σχεδιασμού και χρονοπρογραμματισμού για τις καθημερινές εργασίες κάποιου χρήστη. Σε γενικές γραμμές το σύστημα θα προσπαθεί να βρει της εργασίες που πρέπει ο χρήστης να εκτελέσει για να πετύχει κάποιους στόχους. Παρόλο που το πρότυπο σύστημα που θα παρουσιάσουμε είναι περιορισμένων δυνατοτήτων μπορεί να αποτελέσει την βάση για ένα σύστημα-βοηθό για τον άνθρωπο.

Οι απαιτήσεις του συστήματος που υλοποιήσαμε ήταν οι εξής:

- Η διεπαφή χρήστη θα βρίσκεται σε μικροσυσκευή.
- Ο χρήστης θα πρέπει να εισάγει στην μικροσυσκευή του τις καθημερινές του εργασίες μαζί με τυχόν περιορισμούς στους χρόνους εκτέλεσης, χρησιμοποιώντας την διεπαφή χρήστη.
- Οι εργασίες θα αποστέλλονται στον εξυπηρετητή όπου και το πλάνο του χρήστη θα δημιουργείται.
- Το τελικό πλάνο θα επιστρέφει στην μικροσυσκευή του χρήστη όπου και θα εμφανίζεται χρήστη.
- Οι εργασίες του χρηστή θα επιλέγονται από μια οντολογία εργασιών.
- Το σύστημα πρέπει να είναι ανεξάρτητο από το λειτουργικό εκτέλεσης του.

1.2 ΥΠΑΡΧΟΝΤΑ ΣΥΣΤΗΜΑΤΑ ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Στην βιβλιογραφία υπάρχουν πολλά συστήματα που σκοπό έχουν την λύση προβλημάτων χρονοπρογραμματισμού εργασιών, που αφορούν τον χώρο της υγείας, την εξερεύνηση του διαστήματος, την υποστήριξη στρατιωτικών επιχειρήσεων, την οργάνωση ομάδων ανθρώπων ή/και ατόμων, την κατασκευαστική αλυσίδα κ.α. Παρόλο που οι υλοποιήσεις αυτές χρησιμοποιούν διαφορετικές μεθόδους αναπαράστασης και επίλυσης αποτελούν όλες απλά διαφορετικές προσεγγίσεις στο ίδιο πρόβλημα, το πρόβλημα του σχεδιασμού και χρονοπρογραμματισμού.

Ένα σύστημα που επιλύει το πρόβλημα της οργάνωσης ομάδων και ατόμων είναι το Electric Elves [3]. Το σύστημα Electric Elves αποτελείται από ευφυείς πράκτορες που εργάζονται σε μια ανθρώπινη επιχείρηση βοηθώντας στις εργασίες του οργανισμού. Το σύστημα βοηθά τους ανθρώπους σε εργασίες ρουτίνας, όπως ο προγραμματισμός συναντήσεων, η επιλογή ομιλητών σε κάποια συνάντηση, η οργάνωση γευμάτων εργασίας κλπ. Επιπλέον οι πράκτορες μπορούν να αλληλεπιδρούν με ανθρώπους που δουλεύουν στον οργανισμό καθώς και με ανθρώπους εκτός οργανισμού μέσω email, fax κλπ.

Οι πράκτορες φροντίζουν ώστε οι εργασίες του οργανισμού να εκτελούνται σωστά. Φροντίζουν επίσης κατά την δημιουργία του προγράμματος των εργασιών να μεγιστοποιούν την παραγωγικότητα των εργαζομένων. Το σύστημα Electric Elves προέρχεται από το University of Southern California. Χρηματοδοτείται από το πρόγραμμα Control of Agent-based Systems (COABS) του DARPA.

Στον χώρο της υγείας έχουμε το σύστημα Autominder [19]. Το Autominder βοηθά ηλικιωμένους ανθρώπους με προβλήματα μνήμης να εκτελούν τις καθημερινές τους εργασίες χρησιμοποιώντας ειδοποιήσεις. Το σύστημα χρησιμοποιώντας τεχνικές μηχανικής μάθησης μαθαίνει από την συμπεριφορά του πελάτη. Έτσι οι ειδοποιήσεις του είναι χρονικά και συντακτικά προσεγμένες. Αρχικά ο γιατρός εισάγει την περιγραφή των ενεργειών καθώς και περιορισμούς που αφορούν τον χρόνο και τρόπο εκτέλεσης, στο Autominder. Κατόπιν το σύστημα χρησιμοποιώντας δυναμικό planning παρακολουθεί τον χρήστη και παράγει υπενθυμίσεις για τις ενέργειες που πρέπει να εκτελεστούν. Το Autominder είναι ένα διαπανεπιστημιακό πρόγραμμα μεταξύ των University of Pittsburgh, Carnegie Mellon University και του University of Michigan.

Στην υποστήριξη στρατιωτικών επιχειρήσεων το DEOS [7, 8] προσπαθεί να δημιουργήσει στρατηγικές μάχης με ελάχιστες ζημιές και μέγιστα αποτελέσματα. Το DEOS προσπαθεί να δημιουργήσει στρατηγικές μάχης που περιλαμβάνουν το ναυτικό, την αεροπορία και τον στρατό ούτως ώστε να πετύχουν μια δυναμική λίστα στόχων. Κάθε

στόχος στην λίστα έχει μια βαθμολογία που περιγράφει την σημαντικότητα του και καθώς η μάχη εξελίσσεται το DEOS προσαρμόζει την στρατηγική του λαμβάνοντας υπόψη ευκαιρίες επίτευξης σημαντικών στόχων, πιθανότητες επιτυχίας ή αποτυχίας αποστολών και επάρκειας υλικού. Για παράδειγμα το σύστημα μπορεί να οργανώσει τον βομβαρδισμό κάποιο στόχου επιλέγοντας τα μέσα (βομβαρδιστικά, μαχητικά) που θα λάβουν μέρος στην επιχείρηση, μειώνοντας έτσι τις μετακινήσεις των αεροσκαφών, τον οπλισμό τους, μεγιστοποιώντας την πιθανότητα επιτυχίας, και άλλες σημαντικές παραμέτρους.

Στον χώρο της αγοράς υπάρχει το σύστημα Impulse[27] που αποτελείται από πράκτορες αγοραστών και πωλητών που βρίσκονται στην προσωπική μικροσυσκευή κάποιου ατόμου και τον βοηθούν στην λήψη αποφάσεων αγοράς κάνοντας έρευνα αγοράς για τα προϊόντα που ζητά καθώς και παρουσιάζοντας του εναλλακτικές προτάσεις. Ο χρήστης διατηρεί μια λίστα με αντικείμενα που επιθυμεί να αγοράσει καθώς και προτιμήσεις που αφορούν τα αντικείμενα, όπως μέγιστη τιμή, χρόνο και ποιότητα εγγύησης. Καθώς ο χρήστης κινείται σε μια περιοχή κοντινά καταστήματα που πωλούν κάποια από τα αντικείμενα που αναζητεί ο χρήστης επικοινωνούν με την μικροσυσκευή του χρήστη και προσπαθούν να έρθουν σε συμφωνία για την αγοραπωλησία των αντικειμένων, ενημερώνοντας τον χρήστη για τις επιτυχείς συμφωνίες. Το σύστημα Impulse προέρχεται από το Massachusetts Institute of Technology (MIT) Media Laboratory.

1.3 ΠΕΡΙΓΡΑΦΗ ΚΕΦΑΛΑΙΩΝ

Λόγω της σημαντικότητας του σχεδιασμού ενεργειών και του χρονο-προγραμματισμού στο σύστημα που θα υλοποιήσουμε κρίναμε ορθό να παρουσιάσουμε στα αρχικά κεφάλαια, βασικές έννοιες και μεθόδους που χρησιμοποιούν αυτές οι τεχνολογίες. Στο Κεφάλαιο 2 θα παρουσιάσουμε μεθόδους αναπαράστασης και επίλυσης προβλημάτων σχεδιασμού ενεργειών μιας και ο σχεδιασμός ενεργειών είναι μια από της κυρίες απαιτήσεις του συστήματος που υλοποιήσαμε.

Κύρια απαίτηση του συστήματος είναι και η δυνατότητα χρονοπρογραμματισμού έτσι στο Κεφάλαιο 3 θα παρουσιάσουμε μεθόδους αναπαράστασης και επίλυσης προβλημάτων ικανοποίησης περιορισμών που μια ειδική τους μορφή είναι και τα προβλήματα χρονοπρογραμματισμού. Στο Κεφάλαιο 4 θα αναφερθούμε στον χρονικό σχεδιασμό ενεργειών και σε μεθόδους αναπαράστασης και επίλυσης χρονικών προβλημάτων σχεδιασμού. Στο Κεφάλαιο 5 περιγράφουμε την υλοποίηση του συστήματος παροχής υπηρεσιών σχεδιασμού και χρονοπρογραμματισμού και των υποσυστημάτων του. Τέλος στο Κεφάλαιο 6 θα παρουσιάσουμε τα συμπεράσματα που καταλήξαμε από την ενασχόληση μας με αυτό το αντικείμενο καθώς και ιδέες για μελλοντικές επεκτάσεις του συστήματος.

2 ΣΧΕΔΙΑΣΜΟΣ ΕΝΕΡΓΕΙΩΝ

2.1 ΕΙΣΑΓΩΓΗ

Κατά την διάρκεια μιας συνηθισμένης μέρας ένας άνθρωπος κάνει πολλά πράγματα. Για παράδειγμα μια συνηθισμένη ενέργεια που κάνουμε όταν ξυπνήσουμε το πρωί, είναι να φτιάξουμε καφέ. Η διαδικασία είναι απλή: Ζεσταίνουμε νερό, βάζουμε το ζεστό νερό σε μια κούπα, προσθέτουμε γάλα, προσθέτουμε ζάχαρη, προσθέτουμε καφέ και ανακατεύουμε. Σαν άνθρωποι δεν χρειάζεται να σκεφτούμε πως θα φτιάξουμε ένα καφέ. Η διαδικασία είναι ήδη γνωστή σε εμάς και το μόνο που κάνουμε είναι να ακολουθήσουμε τα βήματα.

Στον χαοτικό κόσμο που ζούμε δεν είναι δυνατό να έχουμε έτοιμες διαδικασίες για όλα τα πράγματα που θέλουμε να κάνουμε. Ακόμα και εμείς οι άνθρωποι χρειάζεται κάποιες φορές να σκεφτούμε τις ενέργειες που πρέπει να εκτελέσουμε για να πετύχουμε κάποιο στόχο. Αυτή η διαδικασία εύρεσης των ενεργειών που απαιτούνται ονομάζεται «Σχεδιασμός ενεργειών».

Πιο συγκεκριμένα, *ο σχεδιασμός ενεργειών είναι η διαδικασία εύρεσης μιας ακολουθίας ενεργειών που σκοπό έχουν την επίτευξη κάποιων προκαθορισμένων στόχων. Πλάνο ονομάζουμε την ακολουθία ενεργειών που προκύπτει από την πιο πάνω διαδικασία.*

Στις ενότητες που ακολουθούν θα παρουσιάσουμε το γενικό μοντέλο του σχεδιασμού ενεργειών, μεθόδους αναπαράστασης προβλημάτων σχεδιασμού ενεργειών καθώς και μία μορφή σχεδιασμού ενεργειών που αναφέρεται στην βιβλιογραφία σαν κλασσικός σχεδιασμός ενεργειών (classical planning). Επίσης θα παρουσιαστούν μέθοδοι επίλυσης προβλημάτων σχεδιασμού και δημιουργίας ευρετικών συναρτήσεων.

2.2 ΤΟ ΜΟΝΤΕΛΟ ΣΧΕΔΙΑΣΜΟΥ ΕΝΕΡΓΕΙΩΝ

Το γενικό μοντέλο αναπαράστασης αυτής της διαδικασίας είναι ένα σύστημα εναλλαγής καταστάσεων [5] που ορίζεται από την τετράδα Σ = (S,A,E,γ), όπού:

- S = $\{s_1,s_2, \ldots\}$ το πεπερασμένο σύνολο καταστάσεων.
- A = $\{a_1,a_2, \ldots\}$ το πεπερασμένο σύνολο ενεργειών.
- E = $\{e_1,e_2, \ldots\}$ το πεπερασμένο σύνολο γεγονότων.
- γ: S x A x E → 2^S η συνάρτηση εναλλαγής κατάστασης.

Το σύστημα εναλλαγής καταστάσεων αναπαρίσταται με ένα κατευθυνόμενο γράφο με κόμβους το σύνολο S. Κάθε $s' \in \gamma(s,u)$, όπου u είναι το ζεύγος (α,e), $a \in A$ και $e \in E$, αναπαρίσταται από ένα τόξο από το s στο s΄ με ετικέτα το u.

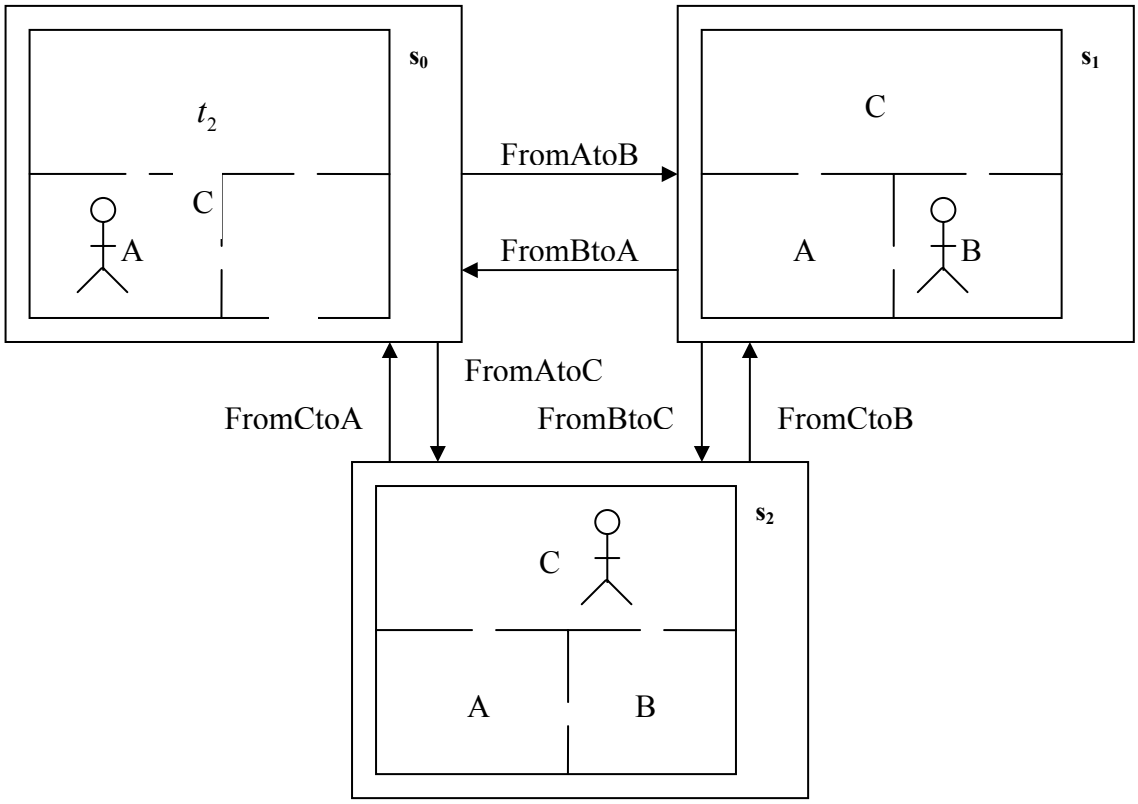
Παράδειγμα 2-1

Στην Διάγραμμα 2-1 εμφανίζεται ένα σύστημα εναλλαγής καταστάσεων που περιγράφει ένα ρομπότ και τρία δωμάτια. Τα τρία δωμάτια συνδέονται μεταξύ τους και το ρομπότ μπορεί να μετακινείται από κάποιο δωμάτιο σε κάποιο άλλο. Το σύνολο των καταστάσεων S είναι το $\{s_0,s_1,s_2\}$, το σύνολο ενεργειών A το {FromBtoA, FromCtoA, FromAtoB, FromCtoB, FromAtoC, FromBtoC}.

Δεδομένου του συστήματος εναλλαγής καταστάσεων ένα πρόβλημα σχεδιασμού ενεργειών ορίζεται ως $P = (\Sigma, s_0, g)$, όπου

- Σ το σύστημα εναλλαγής καταστάσεων.
- s_0 η αρχική κατάσταση.
- g το σύνολο των καταστάσεων στόχων.

Η λύση σε ένα τέτοιο πρόβλημα είναι μια ακολουθία ενεργειών $(\alpha_1,\alpha_2,\ldots,\alpha_\kappa)$ τέτοια ώστε $\gamma(\ldots\gamma(\gamma(s_0,\alpha_1),\alpha_2),\ldots,\alpha_\kappa) = s_\kappa$ και $s_\kappa \in g$.



Διάγραμμα 2-1 : Ένα σύστημα εναλλαγής καταστάσεων

2.3 ΚΛΑΣΣΙΚΟΣ ΣΧΕΔΙΑΣΜΟΣ ΕΝΕΡΓΕΙΩΝ

Ο κλασικός σχεδιασμός ενεργειών ονομάζεται ο σχεδιασμός ενεργειών για συστήματα εναλλαγής κατάστασης στα οποία ισχύουν οι εξής περιοριστικές υποθέσεις:

- Το σύστημα Σ έχει πεπερασμένο σύνολο καταστάσεων.
- Το σύστημα Σ είναι πλήρως παρατηρήσιμο. Έχουμε πλήρη γνώση της κατάστασης του Σ ανά πάσα στιγμή.
- Το σύστημα Σ είναι αιτιοκρατικό. Η εφαρμογή μιας ενέργειας α σε μια κατάσταση s οδηγεί σε μία και μόνο κατάσταση s'.
- Το σύστημα Σ είναι στατικό. Η κατάσταση του συστήματος δεν αλλάζει παρά μόνο όταν εφαρμόζεται μια ενέργεια.
- Οι στόχοι του συστήματος είναι είτε μια κατάσταση είτε ένα σύνολο καταστάσεων. Δεν επιτρέπονται στόχοι όπως, καταστάσεις που πρέπει να αποφύγουμε ή καταστάσεις που πρέπει περιέχονται στο τελικό πλάνο.
- Το πλάνο λύση πρέπει να είναι σειριακό. Το πλάνο πρέπει να είναι μια γραμμικά ταξινομημένη πεπερασμένη ακολουθία ενεργειών.
- Οι ενέργειες και τα γεγονότα του συστήματος δεν έχουν χρονική διάρκεια. Οι αλλαγές που προκαλούν στην κατάσταση του συστήματος είναι ακαριαίες.
- Ο σχεδιασμός ενεργειών γίνεται πριν από την εκτέλεση του πλάνου.

Ένα τέτοιο σύστημα ορίζεται από την τριάδα $\Sigma = (S,A,\gamma)$ και από την συνάρτηση εναλλαγής $\gamma(s,u)$ προκύπτει μία και μόνο κατάσταση s΄.

2.4 ΜΕΘΟΔΟΙ ΑΝΑΠΑΡΑΣΤΑΣΗΣ ΠΡΟΒΛΗΜΑΤΩΝ

Στην παρούσα ενότητα θα παρουσιάσουμε δύο μεθόδους αναπαράστασης προβλημάτων σχεδιασμού ενεργειών, την αναπαράσταση που χρησιμοποιείται στον κλασσικό σχεδιασμό ενεργειών και την αναπαράσταση με την χρήση μεταβλητών κατάστασης.

2.4.1 Αναπαράσταση κλασσικών προβλημάτων

Η κλασσική αναπαράσταση χρησιμοποιεί σημειογραφία που βασίζεται στην λογική πρώτης τάξης. Για την περιγραφή του κόσμου του προβλήματος, των καταστάσεων και των ενεργειών, χρησιμοποιούνται σύνολα από λογικές προτάσεις.

Πιο συγκεκριμένα, για την περιγραφή μιας κατάστασης χρησιμοποιείται ένα σύνολο από σταθερές λογικές προτάσεις, δηλαδή προτάσεις που δεν περιέχουν μεταβλητές, και που ισχύουν για την συγκεκριμένη κατάσταση. Λογικές προτάσεις που δεν περιέχονται στο σύνολο θεωρούνται ως ψευδής.

Παράδειγμα 2-2

Η κατάσταση s0 του συστήματος εναλλαγής καταστάσεων που φαίνεται στο Διάγραμμα 2-1 μπορεί να αναπαρασταθεί ως εξής:

$$s_0 = \{room(a), room(b), room(c), robot(r), adj(a,b), adj(a,c), adj(b,c), in(a,r), empty(b), empty(c)\}$$

- room(α): Υπάρχει το δωμάτιο α.
- room(b): Υπάρχει το δωμάτιο b.
- room(c): Υπάρχει το δωμάτιο c.
- robot(r): Υπάρχει το ρομπότ r.
- adj(α,b): Το δωμάτιο α συγκοινωνεί με το δωμάτιο b.
- adj(α,c): Το δωμάτιο α συγκοινωνεί με το δωμάτιο c.
- adj(b,c): Το δωμάτιο b συγκοινωνεί με το δωμάτιο c.
- in(α,r): Το ρομπότ r βρίσκεται στο δωμάτιο α.
- empty(b): Το δωμάτιο b είναι άδειο.
- empty(c): Το δωμάτιο c είναι άδειο.

Η πιο πάνω κατάσταση θα μπορούσε να αναπαρασταθεί πιο σύντομα χρησιμοποιώντας μόνο τις λογικές προτάσεις που αλλάζουν τιμή από κατάσταση σε κατάσταση, δημιουργώντας ταυτόχρονα ένα σύνολο που περιέχει τις λογικές προτάσεις που έχουν σταθερή τιμή ανεξάρτητα από την κατάσταση την οποία εξετάζουμε.

Η πιο πάνω κατάσταση θα αναπαρασταθεί ως $s_0 = \{in(a,r), empty(b), empty(c)\}$ και το σύνολο των λογικών προτάσεων με σταθερή τιμή ως $c = \{room(a), room(b), room(c), robot(r), adj(a,b), adj(a,c), adj(b,c)\}$.

Μια ενέργεια αναπαρίσταται με την τριάδα: o = (preconditions(o), add(o), delete(o))

- preconditions(o) το σύνολο των λογικών προτάσεων που πρέπει να περιέχει μια κατάσταση s ούτως ώστε η ενεργεία o να είναι εφαρμόσιμη σε αυτήν.
- add(o) το σύνολο των λογικών προτάσεων που η ενέργεια o θα προσθέσει στην κατάσταση s.
- delete(o) το σύνολο των λογικών προτάσεων που η ενέργεια o θα διαγράψει από την κατάσταση s.

Πιο συγκεκριμένα, μια ενέργεια *o* είναι εφαρμόσιμη σε μια κατάσταση *s* όταν το σύνολο preconditions(*o*) είναι υποσύνολο της κατάστασης *s* (preconditions(o) $\subseteq$ s).

Το αποτέλεσμα εφαρμογής μιας ενέργειας *o* σε μια κατάσταση *s* έχει ως αποτέλεσμα

$\gamma(s,o) = (s\text{-delete}(o)) \cup \text{add}(o)$

Παράδειγμα 2-3

Η ενέργεια FromAtoB του συστήματος εναλλαγής κατάστασης που φαίνεται στο Διάγραμμα 2-1 αναπαρίσταται ως εξής:

preconditions = {in(a,r}, empty(b)}

add = {in(b,r), empty(a)}

delet = {in(a,r), empty(b)}

και η εφαρμογή της στην κατάσταση s_0 θα είχε το εξής αποτέλεσμα:

$$\gamma(\text{FromAtoB},s_0) = (\{in(a,r), empty(b), empty(c)\} - \{in(a,r), empty(b)\}) \cup \{in(b,r), empty(a)\} = \{in(b,r), empty(a), empty(c)\}$$

Λόγω του πλήθους των ενεργειών που ένα πρόβλημα σχεδιασμού ενεργειών μπορεί να περιέχει, συνηθίζεται οι ομοειδείς ενέργειες να ομαδοποιούνται σε σχήματα ενεργειών. Τα σχήματα ενεργειών είναι γενικές περιγραφές των ενεργειών χρησιμοποιώντας λογικές προτάσεις που περιέχουν μεταβλητές.

Το σχήμα ενέργειας αναπαρίσταται ως, o = (name(o), preconditions(o), add(o), delete(o))

- name(*o*) το όνομα της ενέργειας. Έχει την μορφή $n(x_1,\ldots,x_k)$, όπου n ένα αλφαριθμητικό και x_1 έως x_k η μεταβλητές που εμφανίζονται στο *o*.
- preconditions(*o*) το σύνολο των λογικών προτάσεων που πρέπει να περιέχει μια κατάσταση *s* ούτως ώστε η ενεργεία *o* να είναι εφαρμόσιμη σε αυτήν.
- add(*o*)[1] το σύνολο των λογικών προτάσεων που η ενέργεια *o* θα προσθέσει στην κατάσταση *s*.
- delete(o)[1] το σύνολο των λογικών προτάσεων που η ενέργεια *o* θα διαγράψει από την κατάσταση *s*.

Παράδειγμα 2-4

Αν ομαδοποιήσουμε τις ενέργειες μετακίνησης που εμφανίζονται στο Διάγραμμα 2-1 σε ένα σχήμα ενεργείας με το όνομα Move θα έχουμε το εξής:

Move(R,X,Y)

preconditions = {in(X,R}, empty(Y)}

add = {in(Y,R), empty(X)}

delete = {in(X,R), empty(Y)}

Ένα στιγμιότυπο *α* κάποιου σχήματος ενέργειας *o* δημιουργείται με την απόδοση τιμών στις μεταβλητές του σχήματος ενέργειας. Εάν αποδοθούν τιμές σε όλες τις μεταβλητές του σχήματος ενέργειας η ενέργεια *α* ονομάζεται συγκεκριμένη (grounded). Εάν τουλάχιστον μια μεταβλητή δεν πάρει τιμή η ενέργεια ονομάζεται μη-συγκεκριμένη (lifted).

[1] Συνηθίζεται τα σύνολα add και delete να περιγράφονται με ένα σύνολο που περιέχει τα στοιχεία του συνόλου add και τις αρνήσεις των στοιχείων του συνόλου delete.

Χρησιμοποιώντας τα πιο πάνω ένα πρόβλημα σχεδιασμού ενεργειών $P = (\Sigma, s_0, g)$ ορίζεται ως η τριάδα συνόλων $P = (O, s_0, g)$ όπου:

- *O* το σύνολο των ενεργειών του προβλήματος.
- s_0 το σύνολο της αρχικής κατάστασης.
- g το σύνολο των λογικών προτάσεων στόχων.

Κλείνοντας αυτή την ενότητα θα πρέπει να αναφερθούμε στην γλώσσα σχεδιασμού ενεργειών PDDL [13]. Η γλώσσα PDDL και οι επεκτάσεις αυτής, χρησιμοποιούνται κατά κόρον στον ορισμό κλασσικών προβλημάτων. Για την περιγραφή κάποιου προβλήματος σχεδιασμού ενεργειών χρησιμοποιούνται δύο αρχεία κειμένου. Στο πρώτο περιγράφεται ο κόσμος του προβλήματος χρησιμοποιώντας γενικές λογικές προτάσεις (προτάσεις που περιέχουν μόνο μεταβλητές), και στο δεύτερο περιγράφεται το πρόβλημα χρησιμοποιώντας σταθερές λογικές προτάσεις.

Παράδειγμα 2-5

Ο κόσμος του προβλήματος που φαίνεται στο Διάγραμμα 2-1 περιγράφεται με την γλώσσα PDDL ως εξής:

```
(define (domain moveRobot)
    (:requirements :strips :typing)
    (:types room robot)
    (:predicates
        (in ?l – room ?r – robot)
        (empty ?l – room)
        (adj ?l1 – room ?l2 – room))
    (:action move
        :parameters (?r – robot ?from – room ?to – room)
        :preconditions ((adj ?from ?to) (in ?from ?r) (free ?to))
        :effect ((in ?to ?r) (free ?from) (not(in ?from ?to) (free ?to))))
```

Διάγραμμα 2-2 : Ο ορισμός του κόσμου του προβλήματος

Κάποιο πιθανό πρόβλημα με αρχική κατάσταση το ρομπότ στο δωμάτιο A και τελική το ρομπότ στο δωμάτιο C περιγράφεται ως εξής:

```
(define (problem moveRobot-1)
    (:domain moveRobot)
    (:objects A B C – room R – robot)
    (:init (in A R) (empty B) (empty C))
    (:goal (in C R)))
```

Διάγραμμα 2-3 : Ο ορισμός του προβλήματος

Η κλασσική αναπαράσταση χρησιμοποιείται από την πλειοψηφία των σχεδιαστών ενεργειών που αναφέρονται στην βιβλιογραφία. Τα πλεονεκτήματα της βρίσκονται κυρίως στην φυσικότητα και την απλότητα του τρόπου περιγραφής του κόσμου του προβλήματος, των καταστάσεων και των ενεργειών.

2.4.2 Αναπαράσταση με την χρήση μεταβλητών κατάστασης

Η αναπαράσταση με χρήση μεταβλητών κατάστασης (state variables) [10], σε αντίθεση με την κλασσική αναπαράσταση, προσπαθεί να περιγράψει ένα πρόβλημα σχεδιασμού ενεργειών χρησιμοποιώντας συναρτήσεις.

Ας θεωρήσουμε την λογική πρόταση in(α,r) από το Παράδειγμα 2-2 και δύο σύνολα το Robots και Rooms, που ονομάζονται τύποι αντικειμένων, όπου $a \in \text{Rooms}$ και $r \in \text{Robots}$. Η λογική αυτή πρόταση αναπαριστά μια σχέση μεταξύ του δωματίου α και του ρομπότ r, ότι το ρομπότ βρίσκεται μέσα στο δωμάτιο. Επίσης η τιμή αυτής της λογικής πρότασης αλλάζει από κατάσταση σε κατάσταση. Επειδή υπάρχει μόνο και μόνο ένα δωμάτιο που περιέχει κάποιο ρομπότ ανά πάσα στιγμή, αυτή η σχέση μπορεί να θεωρηθεί ως μονοσήμαντη αντιστοιχία του συνόλου Robots στο σύνολο Rooms και να αναπαρασταθεί ως συνάρτηση.

Χρησιμοποιώντας σημειογραφία συναρτήσεων η πιο πάνω σχέση γράφετε ως $rloc_r : S \to Rooms$ και $rloc_r(s_0)$ ισούται με το δωμάτιο που βρίσκεται το ρομπότ στην κατάσταση s_0. Το αλφαριθμητικό $rloc_r$ ονομάζεται σύμβολο μεταβλητής κατάστασης και αναφέρεται σε μια συνάρτηση μεταβλητής κατάστασης. Η συνάρτηση αυτή μπορεί να γενικοποιηθεί για όλα τα ρομπότ και να γραφεί ως $rloc : Robots \times S \to Rooms$ και $rloc(r, s_0) = a$.

Η συνάρτηση μπορεί επίσης να γραφεί χρησιμοποιώντας αντί αντικειμένων, όπως r και α, μεταβλητών που ονομάζονται μεταβλητές αντικειμένων και έχουν ως πεδίο τιμών κάποιο τύπο αντικειμένων, όπως Robots, ή την ένωση κάποιον τύπων αντικειμένων, όπως $Robots \cup Rooms$. Αν κάποια συνάρτηση μεταβλητής κατάστασης περιέχει μόνο αντικείμενα ονομάζεται συγκεκριμένη, αν περιέχει τουλάχιστον μια μεταβλητή αντικειμένων ονομάζεται ελεύθερη ή μη συγκεκριμένη συνάρτηση μεταβλητής κατάστασης.

Για την περιγραφή σχέσεων που δεν αλλάζουν από κατάσταση σε κατάσταση του προβλήματος χρησιμοποιείται ένα σύνολο σταθερών σχέσεων (rigid relations). Μια σταθερή σχέση ορίζεται ως η ν-άδα $r(\nu_1,\nu_2,\ldots,\nu_\kappa)$ οπού

- r το σύμβολο της σταθερής σχέσης.
- $\nu_i \in D$, όπου D κάποιος τύπος αντικειμένων ή η ένωση κάποιων τύπων αντικειμένων.
- $r \subseteq D_1^r \times \ldots \times D_k^r$

Παράδειγμα 2-6

Η κατάσταση s0 του συστήματος εναλλαγής καταστάσεων που φαίνεται στο Διάγραμμα 2-1 μπορεί να αναπαρασταθεί ως εξής:

$$s_0 = \{rloc(r) = a, rcontains(a) = r, rcontains(b) = empty, rcontains(c)\}$$

και το σύνολο σταθερών σχέσεων c:

$$c = \{room(a), room(b), room(c), robot(r), adj(a,b), adj(a,c), adj(b,c)\}$$

- $rloc : Robots \times S \to Rooms$
- $rcontains : Rooms \times S \to Robots \cup \{empty\}$
- $Rooms = \{a,b,c\}$, $Robots = \{r\}$
- $room \subseteq Rooms$, $robot \subseteq Robots$
- $adj \subseteq Rooms \times Rooms$

Χρησιμοποιώντας την αναπαράσταση μεταβλητών κατάστασης ένα σχήμα ενέργειας αναπαρίσταται με την τριάδα:

o = (name(o), preconditions(o), add(o), delete(o))

- name(*o*) το όνομα της ενέργειας. Έχει την μορφή $n(x_1,\ldots,x_k)$, όπου n ένα αλφαριθμητικό και x_1 έως x_k οι μεταβλητές που χρησιμοποιούνται στο *o*.
- precond(*o*) σύνολο των σταθερών σχέσεων και ελεύθερων συναρτήσεων μεταβλητής κατάστασης.
- effects(*o*) το σύνολο των αποδόσεων τιμών σε μεταβλητές κατάστασης της μορφής $x(t_1,...,t_k) \leftarrow t_{k+1}$ όπου t_i κάποια μεταβλητή αντικειμένου.

Αντικαθιστώντας όλες της μεταβλητές αντικειμένων του σχήματος ενέργειας *o* δημιουργείται η ενέργεια *α*. Για να είναι εφαρμόσιμη μια ενέργεια *α* σε κάποια κατάσταση *s* πρέπει οι τιμές των συναρτήσεων μεταβλητών κατάστασης της κατάστασης *s* να ισούνται με τις τιμές των συναρτήσεων μεταβλητών κατάστασης που περιέχονται στο σύνολο precond(*α*)

Παράδειγμα 2-7

Το σχήμα ενέργειας Move αναπαρίσταται ως εξής:

Move(R, X, Y)

preconditions = { rloc(R)=X, rcontains(X)=R, rcontains(R)=empty, adj(X,Y) }

effects = { rloc(R) ← Y, rcontains(X) ← empty, rcontains(Y) ← R }

Χρησιμοποιώντας τα πιο πάνω ένα πρόβλημα σχεδιασμού ενεργειών $P = (\Sigma, s_0, g)$ ορίζεται ως η τετράδα συνόλων $P = (O, R, s_0, g)$ όπου:

- *O* το σύνολο των ενεργειών του προβλήματος.
- *R* το σύνολο των σταθερών σχέσεων του προβλήματος.
- s_0 η αρχική κατάσταση.
- g το σύνολο των στόχων.

Η αναπαράσταση με την χρήση μεταβλητών κατάστασης προσπαθεί να αναπαραστήσει τον κόσμο χρησιμοποιώντας περισσότερο μαθηματικούς όρους πάρα όρους λογικής. Χρησιμοποιείται σε αρκετούς σχεδιαστές ενεργειών με κυριότερο τον IxTeT [9].

2.5 ΜΕΘΟΔΟΙ ΕΠΙΛΥΣΗΣ ΠΡΟΒΛΗΜΑΤΩΝ ΣΧΕΔΙΑΣΜΟΥ

Στην παρούσα ενότητα θα παρουσιάσουμε τρεις μεθόδους επίλυσης προβλημάτων σχεδιασμού. Στην ενότητα 2.5.1 την αναζήτηση στον χώρο των καταστάσεων, στην ενότητα 2.5.2 την αναζήτηση στον χώρο των πλάνων και στην ενότητα 2.5.3 τον σχεδιασμό ενεργειών με την χρήση του γράφου σχεδιασμού.

2.5.1 Σχεδιασμός ενεργειών ως αναζήτηση στο χώρο των καταστάσεων

Ο χώρος των καταστάσεων συνηθίζεται να παρουσιάζεται ως ένας γράφος με κόμβους τις καταστάσεις. Μεταξύ δύο καταστάσεων s και s΄ αν υπάρχει ενέργεια α τέτοια ώστε γ(α,s)= s΄ τότε υπάρχει το τόξο (s, s΄) με ετικέτα το *α*. Χρησιμοποιώντας αυτή την αναπαράσταση, το πρόβλημα του σχεδιασμού ενεργειών μετατρέπεται σε πρόβλημα αναζήτησης και μπορούμε να χρησιμοποιήσουμε αλγόριθμους τυφλής ή ευρετικής αναζήτησης για την λύση του.

Μπορούμε επίσης να αναζητήσουμε την λύση είτε ξεκινώντας από την αρχική κατάσταση (προς τα εμπρός) είτε από τους στόχους (προς τα πίσω). Η λύση είναι η

ακολουθία των ενεργειών που βρίσκονται στο μονοπάτι που συνδέει την αρχική κατάσταση με μια τελική κατάσταση που ικανοποιεί τους στόχους.

Αναζήτηση προς τα εμπρός

Στην αναζήτηση προς τα εμπρός ξεκινούμε από την αρχική κατάσταση s_0 και εφαρμόζοντας μια ενεργεία α παίρνουμε την κατάσταση s΄. Αν η κατάσταση s΄ ικανοποιεί του στόχους, σταματάμε την αναζήτηση, αλλιώς επαναλαμβάνουμε το προηγούμενο βήμα μέχρι να φτάσουμε σε μια κατάσταση που ικανοποιεί τους στόχους.

Στο Διάγραμμα 2-4 φαίνεται μια υλοποίηση του αλγόριθμου της προς τα εμπρός αναζήτησης. Αρκετοί σχεδιαστές ενεργειών χρησιμοποιούν αναζήτηση προς τα εμπρός, είτε για τον υπολογισμό της ευρετικής τους συνάρτησης είτε για την εύρεση λύσης, με γνωστότερους τους FF [11], GRT [22] και ASP/HSP [2].

```
Forward(O, s_0, π, g)
  s ← s_0
  π_f ← π
  if g ⊆ s return π_f
  applicable ← {α | α is a ground instance of an operator in O,
                    and precond(a) ⊆ s}
  if applicable = ∅ then return failure
  nondeterministically choose an action a ∈ applicable
  s ← γ(s,a), π_f ← π_f.a
  return Forward(O, s, π_f, g)
```

Διάγραμμα 2-4 : Ο αλγόριθμος της προς τα εμπρός αναζήτησης

Αναζήτηση προς τα πίσω

Στην αναζήτηση προς τα πίσω ξεκινούμε από τους στόχους g και εφαρμόζοντας κάποια αντίστροφη ενέργεια α παίρνουμε την κατάσταση s΄ όπου $s' = \gamma^{-1}(s,\alpha)$. Αν η κατάσταση s΄ είναι υποσύνολο της κατάστασης s_0 τότε σταματάμε την αναζήτηση, αλλιώς επαναλαμβάνουμε το προηγούμενο βήμα μέχρι να φτάσουμε σε μια κατάσταση που είναι υποσύνολο της κατάστασης s_0.

Στο Διάγραμμα 2-5 φαίνεται μια υλοποίηση του αλγόριθμου της προς τα πίσω αναζήτησης, ως αναδρομικού αλγόριθμου. Αρκετοί σχεδιαστές ενεργειών χρησιμοποιούν αναζήτηση προς τα πίσω, είτε για τον υπολογισμό της ευρετικής τους συνάρτησης είτε για την εύρεση λύσης, με γνωστότερους τους GRT[22] και HSPr[2].

```
Backward(O, s_0, π, g)
  π_f ← π
  if g ⊆ s_0 return π
  applicable ← {α | α is a ground instance of an operator in O,
                    and a relevant for g}
  if applicable = ∅ then return failure
  nondeterministically choose an action a ∈ applicable
  g ← γ^-1(g,a), π_f ← a.π_f
  return Backward(O, s_0, π_f, g)
```

Διάγραμμα 2-5 : Ο αλγόριθμος της προς τα πίσω αναζήτησης

Αναζήτηση πρώτα στο καλύτερο

Στην αναζήτηση πρώτα στο καλύτερο ξεκινούμε από την αρχική κατάσταση s_0 και εφαρμόζοντας όλες τις ενέργειες που είναι εφαρμόσιμες σε αυτήν δημιουργούμε το σύνολο καταστάσεων $S' = \gamma(s, a_k),\ \forall a_k : pre(a_k) \subseteq s$. Βαθμολογούμε τις καταστάσεις με βάση τον ευρετικό μηχανισμό μας και επιλέγουμε ως επόμενη κατάσταση την καλύτερη, μέχρι να φτάσουμε σε μια κατάσταση που ικανοποιεί τους στόχους.

Στο Διάγραμμα 2-6 φαίνεται μια υλοποίηση του αλγόριθμου αναζήτησης πρώτα στο καλύτερο.

```
Bfs(O, s_0, π, g)
  s ← s_0
  π_f ← π
  if g ⊆ s return π_f
  applicable ← {α | α is a ground instance of an operator in O,
                    and precond(a) ⊆ s}
  if applicable = ∅ then return failure
  S ← γ(s, a),  ∀a ∈ applicable
  hS ← h(S)
  s ← max(hS)
  π_f ← π_f.a
  return Bfs(O, s, π_f, g)
```

Διάγραμμα 2-6 : Ο αλγόριθμος πρώτα στο καλύτερο

Αναζήτηση A*

Στην αναζήτηση A* ξεκινούμε από την αρχική κατάσταση s_0 και εφαρμόζοντας όλες τις ενέργειες που είναι εφαρμόσιμες σε αυτήν δημιουργούμε το σύνολο καταστάσεων $S' = \gamma(s, a_k),\ \forall a_k : pre(a_k) \subseteq s$. Βαθμολογούμε τις καταστάσεις με βάση τον ευρετικό μηχανισμό μας και προσθέτουμε στην βαθμολογία κάθε κατάστασης την βαθμολογία της προηγούμενης κατάστασης. Η αρχική κατάσταση βαθμολογείται με 0.

Στο Διάγραμμα 2-7 φαίνεται μια υλοποίηση του αλγόριθμου της προς τα πίσω αναζήτησης, ως αναδρομικού αλγόριθμου.

```
Bfs(O, s_0, π, g)
  s ← s_0
  π_f ← π
  if g ⊆ s return π_f
  applicable ← {α | α is a ground instance of an operator in O,
                    and precond(a) ⊆ s}
  if applicable = ∅ then return failure
  S ← γ(s, a),  ∀a ∈ applicable
  ∀s' ∈ S,  h(S) ← h(S) ∪ (h(s') + h(s))
  s ← max(hS)
  π_f ← π_f.a
  return Bfs(O, s, π_f, g)
```

Διάγραμμα 2-7 : Ο αλγόριθμος A*

2.5.2 Σχεδιασμός ενεργειών ως αναζήτηση στο χώρο των πλάνων

Παρόλο που ο χώρος των πλάνων διαφέρει από τον χώρο των καταστάσεων η βασική ιδέα στην αναζήτηση στον χώρο των πλάνων βασίζεται, όπως και η αναζήτηση προς τα πίσω στον χώρο των καταστάσεων, στην μέθοδο της *ανάλυσης μέσων-στόχων*. Η ιδέα πίσω από αυτή την μέθοδο είναι ότι οι ενέργειες (μέσα) επιλέγονται με βάση τους στόχους.

Ο χώρος των πλάνων έχει ως κόμβους ημιτελή πλάνα, δηλαδή σύνολα από ενέργειες που μπορεί να είναι μη συγκεκριμένες και μη διατεταγμένες. Τα τόξα μεταξύ των κόμβων είναι ενέργειες διόρθωσης των ελαττωμάτων του ημιτελούς πλάνου. Για τις ενέργειες διόρθωσης και τα ελαττώματα κάποιου ημιτελούς πλάνου θα αναφερθούμε στις επόμενες ενότητες. Επίσης τα πλάνα λύσεις που παράγονται από αυτήν την διαδικασία δεν είναι ακολουθίες ενεργειών αλλά σύνολα ενεργειών με περιορισμούς διάταξης των ενεργειών.

Ελαττώματα ημιτελών πλάνων

Σε ένα ημιτελές πλάνο μπορούν να υπάρξουν δύο ειδών ελαττώματα. Στόχοι που δεν έχει ακόμα ικανοποιηθεί και απειλές μεταξύ ενεργειών. Μια ενέργεια α απειλεί μια ενέργεια β όταν διαγράφει κάτι από το πλάνο, το οποίο το χρειάζεται η ενέργεια β ($del(\alpha) \cap precond(\beta) \neq \varnothing$). Στην περίπτωση κλασσικής αναπαράστασης έχουμε απειλή όταν μια ενέργεια διαγράφει μια λογική πρόταση που χρειάζεται κάποια άλλη ενέργεια και στην αναπαράσταση με μεταβλητές κατάστασης όταν μια ενέργεια έχει ως προαπαιτούμενο μια τιμή κάποιας μεταβλητής κατάστασης και κάποια άλλη ενέργεια αλλάζει αυτή την τιμή.

Αιτιολογικοί σύνδεσμοι και περιορισμοί διάταξης

Για να αναπαρασταθεί στο ημιτελές πλάνο αυτή η ανάγκη, κάποιας ενέργειας σε μια τιμή ή λογική πρόταση χρησιμοποιούμε αιτιολογικούς συνδέσμους. Ένας αιτιολογικός σύνδεσμος συμβολίζεται με $a_i \xrightarrow{p} a_j$, όπου α_i και α_j είναι ενέργειες του ημιτελούς πλάνου και p μια λογική πρόταση ή απόδοση τιμής σε μεταβλητή κατάστασης για την οποία ισχύουν $p \in effects(a_i)$ και $p \in precond(a_j)$.

Ένας περιορισμός διάταξης μεταξύ δύο ενεργειών του ημιτελούς πλάνου συμβολίζεται με $(a_i \prec a_j)$, όπου α_i και α_j είναι ενέργειες του ημιτελούς πλάνου και η ενέργεια α_i πρέπει να εκτελεστεί πριν από την ενέργεια α_j.

Ενέργειες διόρθωσης

Ανοικτοί στόχοι, δηλαδή στόχοι που δεν έχουν ακόμα ικανοποιηθεί, διορθώνονται με την εισαγωγή κάποιας ενέργειας ή με την χρήση κάποιας ενέργειας που υπάρχει ήδη στο ημιτελές πλάνο. Οι απειλές διορθώνονται με την εισαγωγή περιορισμών διάταξης έτσι ώστε η ενέργεια α που απειλεί μια ενέργεια β να εκτελεστεί πριν την ενέργεια β ($\alpha \prec \beta$) ή μετά την ενέργεια β ($\beta \prec \alpha$).

Ημιτελή πλάνα

Ένα ημιτελές πλάνο ορίζεται ως η τριάδα[2]: $\pi = (A, O, L)$

- $A = \{a_1, ..., a_k\}$ το σύνολο των ενεργειών.
- O το σύνολο των περιορισμών διάταξης.
- L το σύνολο των αιτιολογικών συνδέσμων.

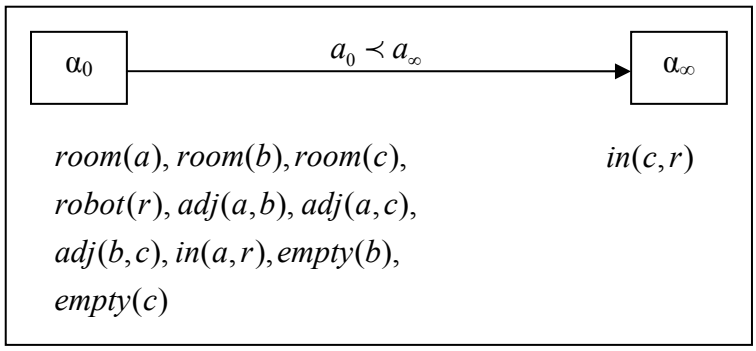
[2] Εάν το ημιτελές πλάνο περιέχει μη-συγκεκριμένες ενέργειες τότε χρησιμοποιείται ένα επιπλέον σύνολο B που περιέχει τους περιορισμούς απόδοσης τιμής στις μεταβλητές των ενεργειών.

Εδώ πρέπει να σημειωθεί ότι το αρχικό ημιτελές πλάνο περιέχει ήδη δυο ενέργειες, την ενέργεια a_0 που έχει ως αποτελέσματα την αρχική κατάσταση s_0 κάποιου προβλήματος P ($effects(a_0) = s_0$) και την ενέργεια α_∞ που έχει ως προαπαιτούμενα τους στόχους g του προβλήματος P ($precond(a_\infty) = g$). Επίσης περιέχει και τον περιορισμό διάταξης $(\alpha_0 \prec a_\infty)$

Η εισαγωγή κάποιας ενέργειας α_κ στο πλάνο γίνεται με την εισαγωγή της στο σύνολο A, την εισαγωγή των απαιτούμενων περιορισμών διάταξης στο σύνολο O και την εισαγωγή αιτιολογικών συνδέσμων στο σύνολο L.

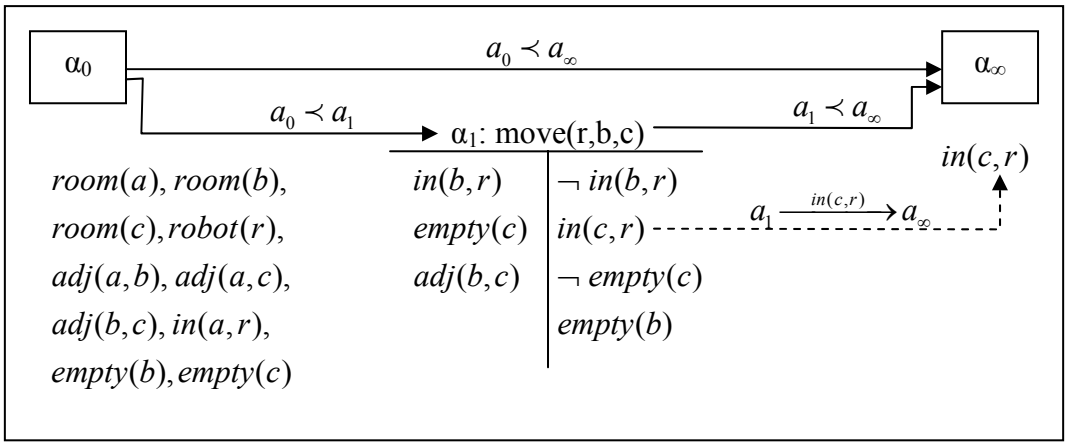
Παράδειγμα 2-8

Το αρχικό πλάνο π_0 του προβλήματος που ορίζεται στο Διάγραμμα 2-2 και στο Διάγραμμα 2-3, χρησιμοποιώντας κλασσική αναπαράσταση εμφανίζεται παρακάτω. Η αρχική κατάσταση είναι η $s_0 = \{room(a), room(b), room(c), robot(r), adj(a,b),$ $, adj(a,c), adj(b,c), in(a,r), empty(b), empty(c)\}$ και $g = \{in(c,r)\}$.



Διάγραμμα 2-8 : Το αρχικό ημιτελές πλάνο

Εισάγοντας την ενέργεια move(r,b,c) στο πλάνο το πλάνο μας θα είναι



Διάγραμμα 2-9 : Το ημιτελές πλάνο μετά την εισαγωγή μιας ενέργειας

Ο αλγόριθμος POP

Ο αλγόριθμος POP (partial order planning) χρησιμοποιείται από πολλούς σχεδιαστές ενεργειών για την λύση προβλημάτων, με πρώτο τον NOAH [23] και ως ευρετικός μηχανισμός από τον UnPOP [14]. Χρησιμοποιεί ως παραμέτρους το ημιτελές πλάνο π και το σύνολο Agenda που περιέχει ζεύγη της μορφής *(α,p)*, όπου *α* κάποια ενέργεια του πλάνου και *p* κάποιο προαπαιτούμενο της ενέργειας *α* που δεν έχει ακόμα ικανοποιηθεί. Ο αλγόριθμος ξεκινά με *π* το π_0 και την Agenda να περιέχει την ενέργεια α_∞ με τα προαπαιτούμενα της.

Pop (π, *agenda*) //$\pi = (A, O, L)$
  if *agenda* $= \varnothing$ then return π
  select a pair $(a_j, p) \in$ *agenda*
  agenda $\leftarrow$ *agenda* $- (a_j, p)$
  relevant $\leftarrow$ select all actions a_k from A and from new instances
      of operators for which $p \in$ *effect*(a_k)
  if *relevant* $= \varnothing$ return failure
  non-deterministically select an action $a_i \in$ *relevant*
  $L \leftarrow L \cup \{a_i \xrightarrow{p} a_j\}$
  if $a_i \notin A$ then do
    $A \leftarrow A \cup \{a_i\}$
    $O \leftarrow O \cup \{(a_i \prec a_j), (a_0 \prec a_i \prec a_\infty)\}$
    agenda $\leftarrow$ *agenda* $\cup$ *precond*(a_i)
  if $a_i \in A$ then do
    if $(a_i \prec a_j) \notin O$
      $O \leftarrow O \cup \{(a_i \prec a_j)\}$
  for each thread on $a_i \xrightarrow{p} a_j$ or due to a_i do
    resolvers $\leftarrow$ set of resolvers of thread
    if *resolvers* $= \varnothing$ return failure
    non-deterministically select a *resolver* $\in$ *resolvers*
    $O \leftarrow O \cup$ *resolver*
  return(Pop (π, *agenda*))

Διάγραμμα 2-10 : Ο αλγόριθμος POP

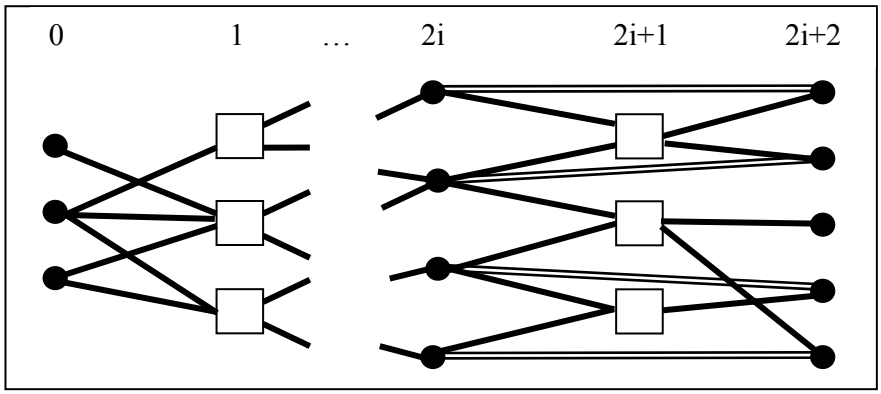
2.5.3 Σχεδιασμός ενεργειών με την χρήση του γράφου σχεδιασμού

Ο σχεδιασμός ενεργειών με την χρήση γράφων σχεδιασμού είναι σχετικά μια νέα μέθοδος, εμφανίστηκε το 1995, και την υλοποιούσε ο σχεδιαστής Graphplan [1]. Παρόλο που ξεκίνησε ως αλγόριθμος επίλυσης προβλημάτων, χρησιμοποιείται ως ευρετικός μηχανισμός από πολλούς σχεδιαστές, όπως ο FF [11] λόγω της απλότητας και της ταχύτητας του.

Ως χώρος για την εύρεση λύσης χρησιμοποιείται ένας γράφος, ο γράφος σχεδιασμού, όπως φαίνεται στο Διάγραμμα 2-1, που αποτελείται από εναλλασσόμενα επίπεδα κόμβων. Υπάρχουν δύο είδη κόμβων, οι κόμβοι γεγονότων και οι κόμβοι συγκεκριμένων ενεργειών. Ένα επίπεδο περιέχει μόνο κόμβους του ίδιου τύπου και τα δύο είδη επιπέδων εναλλάσσονται μεταξύ τους, με πρώτο επίπεδο ένα επίπεδο κόμβων γεγονότων. Επίσης τα πλάνα λύσεις που παράγονται από αυτήν την μέθοδο είναι δυνατό να περιέχουν ενέργειες που εκτελούνται παράλληλα.

Η εκτέλεση του αλγόριθμου χωρίζεται σε δύο φάσεις λειτουργίας, την επέκταση του γράφου και την εξαγωγή λύσης, οι οποίες εναλλάσσονται μέχρι να βρεθεί κάποια λύση. Στην φάση επέκτασης του γράφου, επεκτείνεται ο γράφος με διαδοχικές προσθήκες επιπέδων ενεργειών και επιπέδων γεγονότων και ελέγχεται μια αναγκαία, αλλά όχι ικανή συνθήκη για την ύπαρξη λύσης. Εάν η συνθήκη ύπαρξης λύσης

ικανοποιείται τότε ο αλγόριθμος περνάει στην φάση εξαγωγής λύσης όπου εκτελείται προς τα πίσω αναζήτηση στον γράφο για την εύρεση λύσης. Εάν δεν υπάρχει λύση τότε επεκτείνεται ο γράφος κατά δύο επίπεδα και επανελέγχεται για λύση. Ο αλγόριθμος τερματίζει όταν βρεθεί λύση ή όταν μετά την επέκταση του γράφου δημιουργηθεί επίπεδο γεγονότων 2*i*+2 που περιέχει τα ίδια γεγονότα και σχέσεις αμοιβαίου αποκλεισμού με το επίπεδο 2*i*.



Διάγραμμα 2-11 : Ο γράφος σχεδιασμού

Φάση επέκτασης γράφου

Ο αρχικός γράφος περιέχει μόνο ένα επίπεδο γεγονότων με τα γεγονότα της αρχικής κατάστασης του προβλήματος. Ο γράφος επεκτείνεται με ένα επίπεδο ενεργειών που περιέχει τις ενέργειες των οποίων όλα τα προαπαιτούμενα γεγονότα υπάρχουν στο προηγούμενο επίπεδο και δεν είναι μεταξύ τους ασύμβατα, και με ένα επίπεδο γεγονότων που περιέχει τα γεγονότα που δημιουργούν οι ενέργειες που προστεθήκαν στο προηγούμενο επίπεδο. Στο Διάγραμμα 2-11 η σχέση μεταξύ προαπαιτούμενου γεγονότος και ενέργειας εμφανίζεται με μονή γραμμή, όπως και η σχέση μεταξύ ενέργειας και γεγονότων που δημιουργεί η ενέργεια. Εάν ένα γεγονός ισχύει σε ένα επίπεδο ισχύει και σε όλα τα επόμενα επίπεδα και αυτό εμφανίζεται με διπλή γραμμή και ονομάζεται ενέργεια διατήρησης (no-op).

Επειδή στα επίπεδα ενεργειών εμφανίζονται περισσότερες από μια ενέργειες, υπονοείται ότι οι ενέργειες του ιδίου επιπέδου θα εκτελεστούν παράλληλα. Αυτό όμως δεν είναι πάντα δυνατό, γιατί είναι πιθανό σε κάποιο επίπεδο ενεργειών να υπάρχουν ενέργειες που έχουν ασύμβατα αποτελέσματα ή/και προαπαιτούμενα μεταξύ τους.

Για να αναπαρασταθεί αυτή η ασυμβατότητα μεταξύ ενεργειών ή γεγονότων χρησιμοποιούνται σχέσεις *αμοιβαίου αποκλεισμού* (mutual exclusion relations) και συμβολίζονται στο Διάγραμμα 2-12 με διακεκομμένες γραμμές μεταξύ ενεργειών ιδίου επιπέδου και διακεκομμένες γραμμές μεταξύ γεγονότων ιδίου επιπέδου.

Δύο ενέργειες a_i και a_j στο επίπεδο 2*i*+1 είναι αμοιβαία αποκλειόμενες εάν:

- $effects^+(a_i) \cap effects^-(a_j) \neq \varnothing \vee effects^-(a_i) \cap effects^+(a_j) \neq \varnothing$.

  Η ενέργεια a_i προσθέτει κάποιο γεγονός που διαγράφει η ενέργεια a_j , ή το αντίθετο (ασύμβατα αποτελέσματα).

- $pre(a_i) \cap effects^-(a_j) \neq \varnothing \vee effects^-(a_i) \cap pre(a_j) \neq \varnothing$.

  Η ενέργεια a_i έχει ως προαπαιτούμενο κάποιο γεγονός που το διαγράφει η ενέργεια a_j, ή το αντίθετο (παρέμβαση).

- Υπάρχουν δύο γεγονότα p και q τέτοια ώστε $p \in pre(a_i)$ και $q \in pre(a_j)$ τα οποία είναι μεταξύ τους ασύμβατα (ανταγωνιστικές απαιτήσεις).

Δύο γεγονότα *p* και *q* στο επίπεδο 2*i* είναι μεταξύ τους ασύμβατα όταν:

- Όταν όλες οι ενέργειες στο επίπεδο 2*i*-1, συμπεριλαμβανομένων και τον ενεργειών διατήρησης, που επιτυγχάνουν τα δύο γεγονότα είναι μεταξύ τους ασύμβατες (ασύμβατη υποστήριξη).

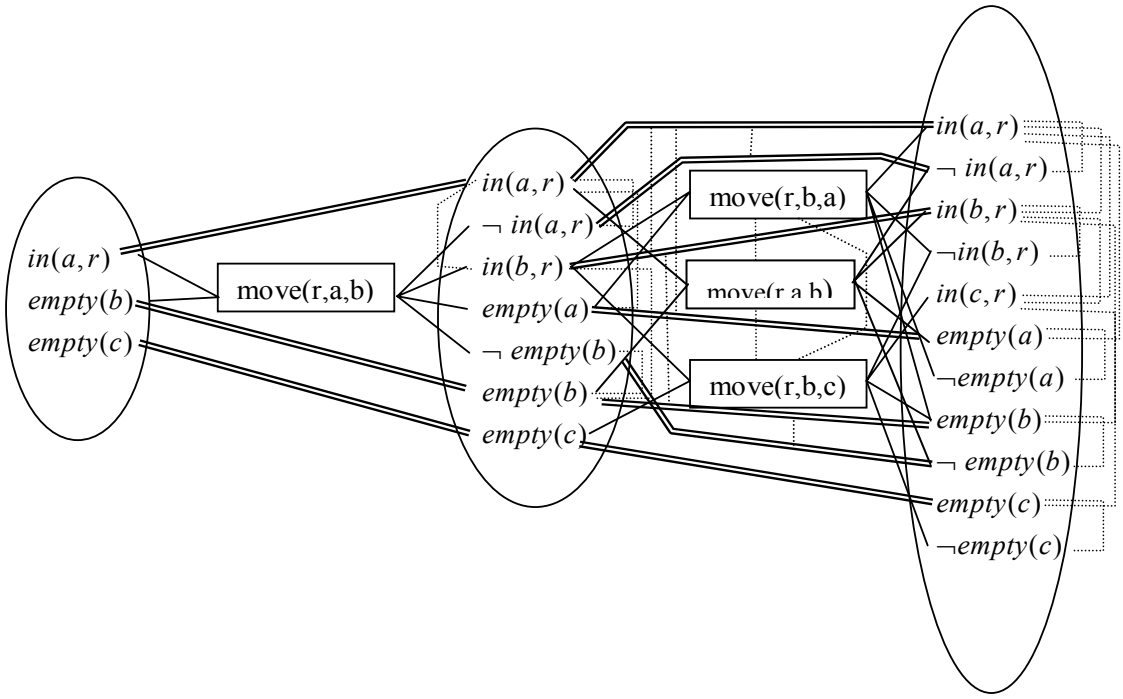
Φάση εξαγωγής λύσης

Ο αλγόριθμος περνάει στην φάση εξαγωγής λύσης όταν σε κάποιο επίπεδο 2*i* εμφανιστούν όλοι οι στόχοι χωρίς να είναι αμοιβαία αποκλειόμενοι μεταξύ τους. Σε αυτή την φάση επιλέγεται μια ενέργεια από το προηγούμενο επίπεδο για κάθε στόχο, φροντίζοντας να μην είναι ασύμβατες μεταξύ τους. Σε περίπτωση που περισσότερες από μια ενέργειες πετυχαίνουν κάποιο στόχο, επιλέγεται μη-αιτιοκρατικά μια από αυτές. Σε περίπτωση αποτυχίας εξαγωγής πλάνου τότε επιλέγεται κάποια άλλη ενεργεία από αυτές που πετυχαίνουν το γεγονός και η αναζήτηση συνεχίζεται. Στην συνέχεια τα προαπαιτούμενα γεγονότα γίνονται οι νέοι στόχοι και επαναλαμβάνεται η διαδικασία μέχρι να φτάσουμε το επίπεδο *0* ή να δοκιμαστούν όλοι οι συνδυασμοί συμβατών ενεργειών χωρίς επιτυχία. Αν δεν βρούμε λύση τότε ο γράφος επεκτείνεται και ελέγχεται για λύση. Αυτό επαναλαμβάνεται μέχρι να βρεθεί κάποια λύση ή ο γράφος να ισορροπήσει, δηλαδή μετά από μια επέκταση καταλήξουμε σε επίπεδο 2*i* που περιέχει τα ίδια γεγονότα και σχέσεις αμοιβαίου αποκλεισμού με το επίπεδο 2*i*-2.

Παράδειγμα 2-9

Στο Διάγραμμα 2-12 φαίνεται ο γράφος σχεδιασμού, του προβλήματος που ορίζεται στο Διάγραμμα 2-2 και στο Διάγραμμα 2-3, μέχρι την εμφάνιση των στόχων στο τρίτο επίπεδο. Παραλείπονται οι σταθερές σχέσεις για οικονομία χώρου.

Επιλέγουμε την ενέργεια *move(r,b,c)* αφού είναι η μόνη που πετυχαίνει τον στόχο μιας και τα προαπαιτούμενα της δεν είναι ασύμβατα μεταξύ τους. Οι προτάσεις *in(b,r)* και *empty(c)* γίνονται οι νέοι στόχοι. Επιλέγουμε για το *in(b,r)* την ενέργεια *move(r,a,b)* και για το *empty(c)* την ενέργεια διατήρησης. Φτάσαμε επιτυχώς στο επίπεδο 0 και η λύση είναι η {*move(r,a,b), move(r,b,c)*}.



Διάγραμμα 2-12 : Ο γράφος του προβλήματος

2.6 ΕΥΡΕΤΙΚΟΙ ΜΗΧΑΝΙΣΜΟΙ

Στις προηγούμενες ενότητες παρουσιάσαμε διάφορους αλγόριθμους και μεθόδους για την επίλυση προβλημάτων σχεδιασμού ενεργειών και εξαιρουμένων των αλγόριθμων αναζήτηση προς τα εμπρός και αναζήτηση προς τα πίσω, που είναι αλγόριθμοι τυφλής αναζήτησης, οι υπόλοιποι αλγόριθμοι μπορούν να χρησιμοποιήσουν ευρετικούς μηχανισμούς για την καθοδήγηση της έρευνας και των επιλογών τους.

Η χρήση ευρετικών μηχανισμών γίνεται αναγκαιότητα όταν το πρόβλημα σχεδιασμού αρχίσει να μεγαλώνει, λόγω της συνδυαστικής έκρηξης που παρουσιάζουν τα προβλήματα σχεδιασμού. Συνηθίζεται στον σχεδιασμό ενεργειών, οι ευρετικοί μηχανισμοί, να είναι αλγόριθμοι σχεδιασμού με χαλαρωμένους κάποιους από τους περιορισμούς τους. Αυτό γίνεται επειδή ο ευρετικός μηχανισμός πρέπει να υπολογίζεται εύκολα και σε λίγο χρόνο. Επίσης οι ευρετικοί μηχανισμοί χρησιμοποιούνται για να βαθμολογήσουν μια κατάσταση, ενέργεια ή επιλογή, παράγοντας εκτιμήσεις για την αξία της κατάστασης, ενέργειας ή επιλογής.

Αυτές οι εκτιμήσεις όμως δεν είναι ακριβείς, μπορεί να μεγαλύτερες ή μικρότερές της πραγματικής αξίας μιας κατάστασης, ενέργειας ή επιλογής. Με την έννοια αξία εννοούμε την δυνατότητα μιας κατάστασης, ενέργειας ή επιλογής να οδηγήσει σε πλάνο που βελτιστοποιεί κάποια ή κάποιες παραμέτρους του προβλήματος. Στον κλασσικό σχεδιασμό η παράμετρος που προσπαθούμε να βελτιστοποιήσουμε είναι το πλήθος των ενεργειών σε κάποιο πλάνο. Συνηθίζεται ευρετικοί μηχανισμοί που παράγουν υποεκτιμήσεις να ονομάζονται και αποδεκτοί επειδή αν χρησιμοποιηθούν σε συνδυασμό με αποδεκτούς ευρετικούς αλγόριθμους, όπως ο A*, παράγουν βέλτιστες λύσεις.

Όπως αναφέρθηκε πιο πάνω συνηθίζεται να χρησιμοποιούνται μέθοδοι σχεδιασμού ενεργειών ως ευρετικοί μηχανισμοί. Για παράδειγμα οι σχεδιαστές HSP[2] και GRT[22] χρησιμοποιούν ως ευρετικό μηχανισμό την αναζήτηση στον χώρο των καταστάσεων, ο σχεδιαστής UnPOP[14] την αναζήτηση στον χώρο των πλάνων και ο σχεδιαστής FF[11] τον γράφο σχεδιασμού. Στις επόμενες ενότητες θα παρουσιαστούν απλοί ευρετικοί μηχανισμοί που βασίζονται στις μεθόδους που παρουσιάσαμε στις ενότητες 2.5.1, 2.5.2 και 2.5.3.

2.6.1 Αναζήτηση στον χώρο των καταστάσεων ως ευρετικός μηχανισμός

Η αναζήτηση στον χώρο των καταστάσεων χρησιμοποιείται ως ευρετικός μηχανισμός δημιουργώντας ένα χαλαρό πρόβλημα με βάση το πραγματικό πρόβλημα που προσπαθούμε να λύσουμε. Το χαλαρό πρόβλημα δημιουργείται χαλαρώνοντας την συνάρτηση μετάβασης *γ(s,α)*, έτσι ώστε να μην λαμβάνονται υπόψη οι διαγραφές γεγονότων από τις ενέργειες.

Έτσι χρησιμοποιώντας ως συνάρτηση μετάβασης την $\gamma(s,o) = s \cup add(o)$ αντί της $\gamma(s,o) = (s - delete(o)) \cup add(o)$ και λύνοντας το χαλαρό πρόβλημα για κάποια κατάσταση *s* υπολογίζουμε μια εκτίμηση του πλήθους των ενεργειών που χρειάζονται για να φτάσουμε από την κατάσταση *s* σε κάποια λύση. Αυτός ο μηχανισμός παράγει υποεκτιμήσεις και άρα θεωρείται αποδεκτός.

2.6.2 Αναζήτηση στον χώρο των πλάνων ως ευρετικός μηχανισμός

Στην αναζήτηση στον χώρο των πλάνων μπορούμε να χρησιμοποιήσουμε ευρετικούς μηχανισμούς σε διάφορες φάσεις της λειτουργίας του αλγόριθμού, όπως στην επιλογή του ανικανοποίητου στόχου που θα ικανοποιήσουμε στην συνέχεια, στην επιλογή του επόμενου ημιτελούς πλάνου που θα χρησιμοποιήσουμε και στην επιλογή της επίλυσης κάποιας απειλής.

Ένας απλός ευρετικός μηχανισμός που χρησιμοποιείται για την επιλογή του επόμενου στόχου που θα ικανοποιηθεί είναι να επιλεγεί ο στόχος που ικανοποιείται με τους λιγότερους τρόπους. Αν υπάρχει στόχος που δεν ικανοποιείται επιλέγεται αμέσως ούτως ώστε ο αλγόριθμος να αποτύχει και να οπισθοχωρήσει. Επίσης εάν υπάρχει στόχος που ικανοποιείται μόνο με ένα τρόπο επιλέγεται αμέσως, αφού υπάρχει μόνο ένας τρόπος να το πετύχουμε. Για την επιλογή του επόμενου ημιτελούς πλάνου που θα χρησιμοποιηθεί, ένας απλός ευρετικός μηχανισμός είναι το πλήθος το ανικανοποίητων στόχων κάποιου ημιτελούς πλάνου. Επιλέγουμε ως επόμενο το πλάνο με τους λιγότερους ανικανοποίητους στόχους. Αυτός ο μηχανισμός παράγει υπερεκτιμήσεις αν υπάρχουν ενέργειες που πετυχαίνουν περισσότερους από ένα στόχους ταυτόχρονα και υποεκτιμήσεις όταν υπάρχουν ενέργειες που απειλούν η μια την άλλη στο πλάνο.

2.6.3 Ο γράφος σχεδιασμού ως ευρετικός μηχανισμός

Όπως αναφέρθηκε και στην ενότητα 2.5.3 ο γράφος σχεδιασμού χρησιμοποιείται κυρίως ως ευρετικός μηχανισμός λόγω του μικρού χρόνου δημιουργίας του, του μικρού του μεγέθους και του πλήθους της χρήσιμης πληροφορίας που περιέχει. Δημιουργώντας ένα χαλαρό γράφο, δηλαδή ένα γράφο που δεν υπολογίζουμε τις σχέσεις αμοιβαίου αποκλεισμού μπορούμε να εκτιμήσουμε την απόσταση μιας κατάστασης s από μια κατάσταση λύση ως το πλήθος των επιπέδων ενεργειών που εμφανίζονται μεταξύ της κατάστασης s και του επιπέδου γεγονότων που εμφανίζονται όλα τα γεγονότα στόχοι.

Επειδή όμως ο γράφος σχεδιασμού επιτρέπει την παράλληλη εκτέλεση ενεργειών, ο μηχανισμός αυτός τείνει να υποεκτιμά υπερβολικά και δεν είναι πολύ χρήσιμος. Μπορούμε να βελτιώσουμε τον μηχανισμό αυτό προσθέτοντας μεταξύ των ενεργειών ανά δύο μια σχέση αμοιβαίου αποκλεισμού αναγκάζοντας έτσι την εκτέλεση μιας ενέργειας σε κάθε επίπεδο. Αυτό το είδος του γράφου ονομάζεται σειριακός γράφος και ο ευρετικός μηχανισμός παράγει υποεκτιμήσεις που είναι πιο κοντά στην πραγματικότητα.

Στην βιβλιογραφία μπορούμε να βρούμε πολλά άρθρα που αναφέρονται στις δυνατότητες του γράφου σχεδιασμού ως ευρετικού μηχανισμού με κυριότερα τα άρθρα του Kambhampati [17, 18].

3 ΙΚΑΝΟΠΟΙΗΣΗ ΠΕΡΙΟΡΙΣΜΩΝ

3.1 ΕΙΣΑΓΩΓΗ

Η ικανοποίηση περιορισμών είναι μια μέθοδος επίλυσης προβλημάτων με πολλές εφαρμογές που ορίζετε ως *η διαδικασία απόδοσης τιμών σε κάποιο σύνολο μεταβλητών από τα αντίστοιχα πεδία τιμών, τέτοια ώστε να ικανοποιεί κάποιο σύνολο περιορισμών*. Μια από τις εφαρμογές της είναι η επίλυση προβλημάτων χρονοπρογραμματισμού. Ως χρονοπρογραμματισμός ορίζεται *το πρόβλημα της αντιστοίχησης ενός συνόλου ενεργειών σε κάποιο σύνολο πόρων οι οποίοι έχουν κάποιους περιορισμούς*. Σε αυτή την εργασία, ως σύνολο των ενεργειών για το πρόβλημα του χρονοπρογραμματισμού θα χρησιμοποιήσουμε το πλάνο λύση που θα παράγει κάποιος σχεδιαστής ενεργειών. Ως σύνολο πόρων, ένα σύνολο που περιέχει μόνο ένα στοιχείο/πόρο, τον χρόνο και ένα σύνολο περιορισμών ποσοτικών και ποιοτικών μεταξύ των ενεργειών.

Εάν για την αναπαράσταση του χρόνου χρησιμοποιήσουμε άλγεβρα χρονικών σημείων στην οποία ο χρόνος θεωρείται μια ακολουθία χρονικών σημείων και για κάθε ενεργεία του πλάνου αντιστοιχίσουμε δύο χρονικά σημεία, την έναρξη και το τέλος, τότε το πρόβλημα του χρονοπρογραμματισμού μετατρέπεται σε πρόβλημα ικανοποίησης περιορισμών. Σε αυτό το πρόβλημα η μεταβλητές μας είναι χρονικά σημεία με πεδίο τιμών τον χρόνο. Το σύνολο των περιορισμών θα περιέχει ποιοτικούς περιορισμούς, όπως η έναρξη της ενέργειας *α* να είναι μετά το τέλος της ενέργειας *β*, και ποσοτικούς, όπως η έναρξη της ενέργειας *α* μετά τις 10:00. Αυτό το είδος του προβλήματος ονομάζεται *Απλό χρονικό πρόβλημα* (*Simple Temporal Problem*)[4, 6].

3.2 ΠΡΟΒΛΗΜΑΤΑ ΙΚΑΝΟΠΟΙΗΣΗΣ ΠΕΡΙΟΡΙΣΜΩΝ

Ένα πρόβλημα ικανοποίησης περιορισμών ορίζεται ως η τριάδα *(V, C, D)* όπου:

- *V* το σύνολο των μεταβλητών $v_i \in V$
- *D* το σύνολο των πεπερασμένων πεδίων τιμών $D(v_i)$, τέτοιο ώστε κάθε v_i παίρνει τιμές μόνο από το πεδίο τιμών $D(v_i)$
- *C* [3] το σύνολο των περιορισμών που περιέχει αποδόσεις τιμών σε μεταβλητές, $v_1 \leftarrow val_1, ..., v_n \leftarrow val_n$ που θεωρούνται μη έγκυρες.

Για κάποιο πρόβλημα ικανοποίησης περιορισμών (V, C, D) ένα σύνολο αποδόσεων τιμών $S = \{v_1 \leftarrow val_1, ..., v_n \leftarrow val_n\}$, όπου $v_i \in V$ και $val_i \in D(v_i)$, θεωρείται μη συνεπές, εάν $S \cap C \neq \varnothing$, δηλαδή περιέχει κάποια απόδοση τιμής που θεωρείται μη έγκυρη. Ένα σύνολο αποδόσεων τιμών ονομάζεται συνεπές εάν $S \cap C = \varnothing$. Ένα συνεπές και πλήρες σύνολο, πλήρες με την έννοια ότι αποδίδει τιμές σε όλες τις μεταβλητές του V, ονομάζεται λύση του προβλήματος ικανοποίησης περιορισμών. Ένα πρόβλημα ικανοποίησης περιορισμών για το οποίο υπάρχει λύση ονομάζεται συνεπές.

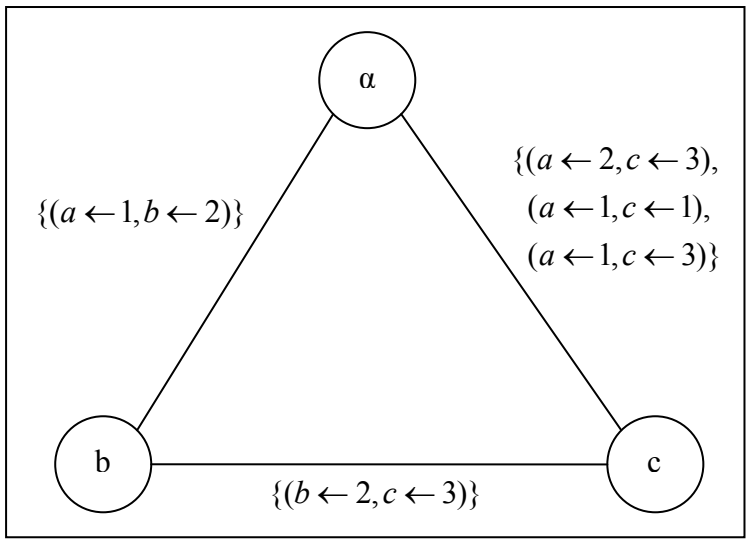
Παράδειγμα 3-1

Έχουμε ένα πρόβλημα ικανοποίησης περιορισμών όπου $V = \{a, b, c\}$, $D(a) = D(b) = D(c) = \{1,\ 2,\ 3\}$ και $C = \{C_1 = \{a \leftarrow 1, b \leftarrow 2\},\ C_2 = \{a \leftarrow 2, c \leftarrow 3\},\ C_3 = \{b \leftarrow 2, c \leftarrow 3\},\ C_4 = \{a \leftarrow 1,\ c \leftarrow 1\},\ C_5 = \{a \leftarrow 1, c \leftarrow 3\}\}$. Υπάρχουν τρεις μεταβλητές, πέντε περιορισμοί και τρεις τιμές σε κάθε πεδίο τιμών. Μία από τις

[3] Σε ισοδύναμους ορισμούς το σύνολο C περιέχει τις αποδόσεις τιμών που θεωρούνται έγκυρες.

λύσεις του προβλήματος είναι η $S=\{a \leftarrow 2, b \leftarrow 1, c \leftarrow 1\}$, και άρα το πρόβλημα είναι συνεπές.

Συνηθίζεται τα προβλήματα ικανοποίησης περιορισμών να αναπαρίστανται με ένα γράφο, το δίκτυο περιορισμών (constraint network), που έχει για κόμβους τις μεταβλητές του προβλήματος και ακμές τους περιορισμούς του προβλήματος. Πιο κάτω φαίνεται το δίκτυο περιορισμών για το πρόβλημα που περιγράφεται στο Παράδειγμα 3-1.



Διάγραμμα 3-1 : Το δίκτυο περιορισμών

Στην βιβλιογραφία υπάρχουν αρκετά τυπικά παραδείγματα προβλημάτων που μπορούν να επιλυθούν ως προβλήματα ικανοποίησης περιορισμών, όπως είναι το *πρόβλημα των βασιλισσών* [15] και το *πρόβλημα του χρωματισμού γράφου* [26].

Στο πρόβλημα των βασιλισσών προσπαθούνε να το τοποθετήσουμε *n* βασίλισσες σε μια σκακιέρα $n \times n$, τέτοια ώστε καμία βασίλισσα να μην απειλεί κάποια άλλη. Στην βιβλιογραφία έχουν προταθεί πολλοί τρόποι αναπαράστασης αυτού του προβλήματος ως πρόβλημα ικανοποίησης περιορισμών. Σε έναν από αυτούς τους τρόπους για κάθε βασίλισσα έχουμε μια μεταβλητή *v*, $V = \{v_1, ..., v_n\}$, με πεδίο τιμών το σύνολο *{1,...,n}*, που αντιστοιχεί στην γραμμή της σκακιέρας που θα τοποθετηθεί κάποια βασίλισσα (θεωρούμε ότι κάθε βασίλισσα θα βρίσκεται σε διαφορετική στήλη) και ένα σύνολο περιορισμών που δεν επιτρέπει τις απειλές μεταξύ βασιλισσών. Η λύση του προβλήματος είναι οι γραμμές στις οποίες πρέπει να τοποθετηθούν οι βασίλισσες ούτως ώστε να μην απειλούν η μια την άλλη.

Στο πρόβλημα του χρωματισμού του γράφου προσπαθούμε να χρωματίσουμε τους κόμβους κάποιου γράφου με τέτοιο τρόπο ώστε κάθε κόμβος να έχει διαφορετικό χρώμα από τους κόμβους με τους οποίους συνδέεται. Κάθε κόμβος του γράφου αναπαρίσταται με μια μεταβλητή το πεδίο τιμών της οποίας είναι το σύνολο των επιτρεπτών χρωμάτων. Τα τόξα του γράφου αναπαρίστανται με περιορισμούς ανισότητας μεταξύ των κόμβων που συνδέουν.

3.3 ΕΠΙΛΥΣΗ ΠΡΟΒΛΗΜΑΤΩΝ ΙΚΑΝΟΠΟΙΗΣΗΣ ΠΕΡΙΟΡΙΣΜΩΝ

Σε γενικές γραμμές τα προβλήματα ικανοποίησης περιορισμών λύνονται με αναζήτηση η/και εξαγωγή συμπερασμάτων (ανάλυση). Υπάρχουν δύο τύποι αναζήτησης, η συστηματική αναζήτηση και η στοχαστική (τοπική) αναζήτηση. Στην συστηματική αναζήτηση, εξετάζουμε όλο τον χώρο των αποδόσεων τιμών, όπου ένας κόμβος αναπαριστά μια μη-πλήρη απόδοση τιμών και οι κόμβοι φύλλα πλήρεις

αποδόσεις τιμών. Ο χώρος που εξετάζουμε στην στοχαστική αναζήτηση, αποτελείται από κόμβους που αναπαριστούν πλήρης αποδόσεις τιμών. Επίσης στην στοχαστική αναζήτηση δεν εξετάζουμε ολόκληρο τον χώρο, αλλά κινούμαστε συνήθως με μια άπληστη πολιτική επιλογών, όπως αναρρίχηση λόφου, και σταματούμε την αναζήτηση μετά από κάποιο διάστημα χρόνου ή πλήθος επιλογών.

Το κυριότερο μειονέκτημα της στοχαστικής αναζήτησης είναι ότι αν δεν βρεθεί λύση δεν μπορούμε να πούμε με σιγουριά ότι δεν υπάρχει λύση, όπως στην περίπτωση της συστηματικής αναζήτησης που επειδή εξετάζει όλο τον χώρο, αν δεν βρεθεί λύση τότε είμαστε σίγουροι ότι δεν υπάρχει λύση. Στις επόμενες ενότητες θα εξετάσουμε διάφορες μεθόδους συστηματικής αναζήτησης, διότι στο σύστημα που υλοποιούμε είναι σημαντικό να μπορούμε να καθορίσουμε αν κάποιο πρόβλημα ικανοποίησης περιορισμών έχει λύση και άρα είναι συνεπές.

3.3.1 Παραγωγή και Δοκιμή

Η απλούστερη μέθοδος επίλυσης προβλημάτων ικανοποίησης περιορισμών είναι η παραγωγή και δοκιμή. Δημιουργούμε κάθε δυνατό συνδυασμό αποδόσεων τιμών και κατόπιν ελέγχουμε αν παραβιάζει κάποιο περιορισμό. Αυτή η μέθοδος όμως πάσχει από την συνδυαστική έκρηξή που παρουσιάζουν τα προβλήματα ικανοποίησης περιορισμών.

Αν είχαμε ένα πρόβλημα παρόμοιο με το Παράδειγμα 3-1 άλλα με 10 μεταβλητές με πεδία τιμών με 100 στοιχεία θα είχαμε 100^{10} συνδυασμούς που θα έπρεπε να δημιουργηθούν και να εξεταστούν.

3.3.2 Χρονολογική αναζήτηση (Αναζήτηση με οπισθοχώρηση)

Στην χρονολογική αναζήτηση το σύνολο αποδόσεων τιμών δημιουργείται σταδιακά, χρησιμοποιώντας αναζήτηση κατά βάθος με οπισθοχώρηση. Ο αλγόριθμός χωρίζεται σε δύο φάσεις, την επέκταση και τον έλεγχο. Στην φάση επέκτασης επιλέγουμε μια τιμή από το πεδίο τιμών κάποιας μεταβλητής που δεν υπάρχει στο σύνολο αποδόσεων τιμών. Στην φάση ελέγχου, ελέγχουμε αν το μερικό σύνολο απόδοσης τιμών παραβιάζει κάποιο περιορισμό.

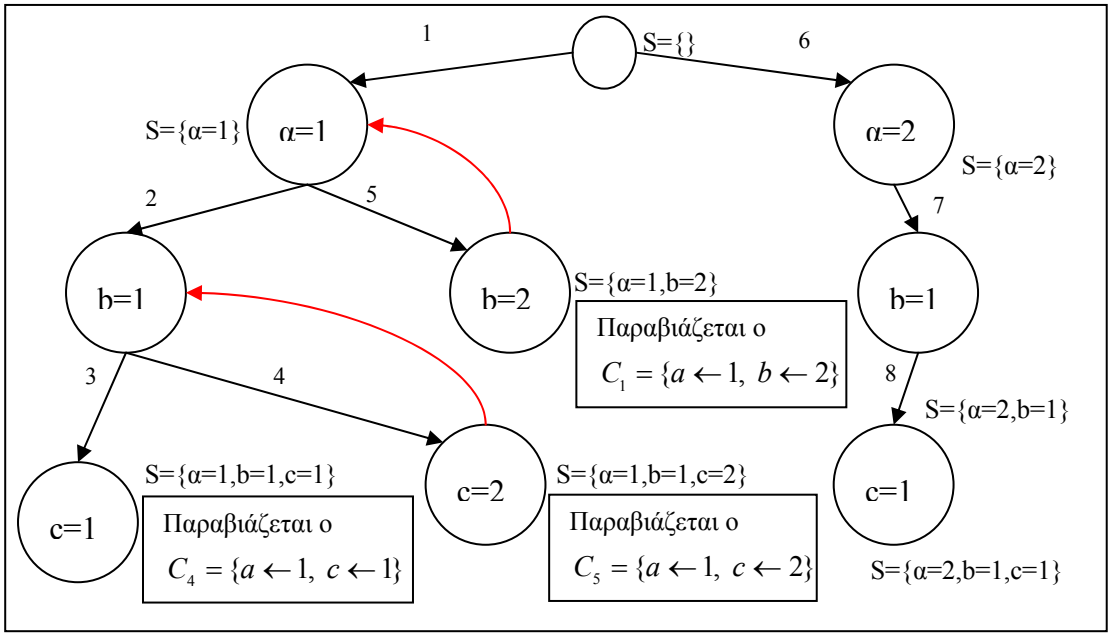
Αν δεν παραβιάζεται κάποιος περιορισμός τότε περνάμε ξανά στην φάση της επέκτασης του μερικού συνόλου απόδοσης τιμών. Αν κάποιος περιορισμός παραβιάζεται τότε επιλεγούμε την επόμενη δυνατή τιμή από το πεδίο τιμών της μεταβλητής που προστέθηκε τελευταία στο μερικό σύνολο απόδοσης τιμών και επαναλαμβάνουμε την φάση ελέγχου.

Αν όλες οι τιμές του πεδίου τιμών της μεταβλητής που προστέθηκε τελευταία στο μερικό σύνολο παραβιάζουν περιορισμούς τότε οπισθοχωρούμε. Δηλαδή η μεταβλητή αφαιρείται από το σύνολο και επιλέγουμε την επόμενη δυνατή τιμή της μεταβλητής που προστέθηκε πριν από την μεταβλητή που αφαιρέσαμε. Η διαδικασία αυτή συνεχίζεται μέχρι να δημιουργήσουμε κάποιο πλήρες και συνεπές σύνολο αποδόσεων τιμών.

Παράδειγμα 3-2

Ένα πρόβλημα ικανοποίησης περιορισμών με τρεις μεταβλητές $V = \{a, b, c\}$, $D(a) = D(b) = D(c) = \{1,\ 2\}$ και $C = \{C_1 = \{a \leftarrow 1, b \leftarrow 2\},\ C_2 = \{a \leftarrow 2, c \leftarrow 2\},\ C_3 = \{b \leftarrow 2, c \leftarrow 2\},\ C_4 = \{a \leftarrow 1,\ c \leftarrow 1\},\ C_5 = \{a \leftarrow 1, c \leftarrow 2\}\}$.

Στο Διάγραμμα 3-2 εμφανίζεται η λύση του προβλήματος εκτελώντας χρονολογική αναζήτηση. Το σύνολο λύση που προκύπτει είναι το $S = \{a \leftarrow 2, b \leftarrow 1, c \leftarrow 1\}$. Τα σημεία οπισθοχώρησης εμφανίζονται με κόκκινα τόξα.



Διάγραμμα 3-2 : Χρονολογική αναζήτηση

Η επιλογή της επόμενης μεταβλητής καθώς και της τιμής που θα της αποδοθεί, όπως και στα προβλήματα σχεδιασμού ενεργειών, μπορούν να γίνουν με την χρήση ευρετικών μηχανισμών. Δύο απλοί κανόνες που χρησιμοποιούνται στα προβλήματα ικανοποίησης περιορισμών είναι να επιλέγουμε την μεταβλητή που συμμετέχει στους περισσότερους περιορισμούς ή την μεταβλητή με το μικρότερο πεδίο τιμών. Αν κάποιο μεταβλητή περιέχει μόνο μια τιμή στο πεδίο τιμών της τότε πρέπει να την επιλέξουμε άμεσα.

3.4 Έλεγχος συνέπειας και Αναπαραγωγή περιορισμών

Ο έλεγχος συνέπειας[4] (consistency enforcing) και η αναπαραγωγή περιορισμών (constraint propagation) προσπαθούν, αναλύοντας τους περιορισμούς κάποιου προβλήματος να αφαιρέσουν μη συνεπής, με τους περιορισμούς, τιμές από τα πεδία τιμών των μεταβλητών του προβλήματος και να προσθέσουν καινούργιους περιορισμούς που προκύπτουν από τους υπάρχοντες, περιορίζοντας έτσι τις δυνατές επιλογές που θα εκτελέσει η μετέπειτα αναζήτηση. Αυτό γίνεται είτε πριν την αναζήτηση είτε κατά την διάρκεια της αναζήτησης (ο έλεγχος συνέπειας κατά την διάρκεια της αναζήτησης ονομάζεται και *έλεγχος προς τα εμπρός*). Συνηθίζεται αυτά τα δύο βήματα, αφαίρεση τιμών και προσθήκη περιορισμών να εκτελούνται εναλλάξ μέχρι το πρόβλημα να ισορροπήσει, δηλαδή να μην είναι δυνατή η αφαίρεση κάποιας τιμής ούτε και η προσθήκη κάποιου περιορισμού. Το πρόβλημα που προκύπτει από αυτές τις τεχνικές είναι ισοδύναμο με το αρχικό πρόβλημα. Υπάρχουν διάφοροι τύποι συνεπείας που μπορούμε να εφαρμόσουμε σε ένα πρόβλημα ικανοποίησης, όπως συνέπεια κόμβων [16], τόξων [12] και μονοπατιού [25].

Στην συνέπεια κόμβων λαμβάνονται υπόψη περιορισμοί που αφορούν μόνο μια μεταβλητή (μοναδιαίοι περιορισμοί όπως $x > 3$) και αφαιρούνται οι μη συνεπείς τιμές από το πεδίο τιμών της μεταβλητής. Παρόμοια είναι και η συνέπεια κόμβων, μόνο που σε αυτήν λαμβάνονται υπόψη περιορισμοί που αφορούν δύο μεταβλητές (δυαδικοί περιορισμοί, όπως $x > y + 4$). Στην συνέπεια μονοπατιού λαμβάνουμε υπόψη περιορισμούς που αφορούν τρεις μεταβλητές (όπως $x > y + z$).

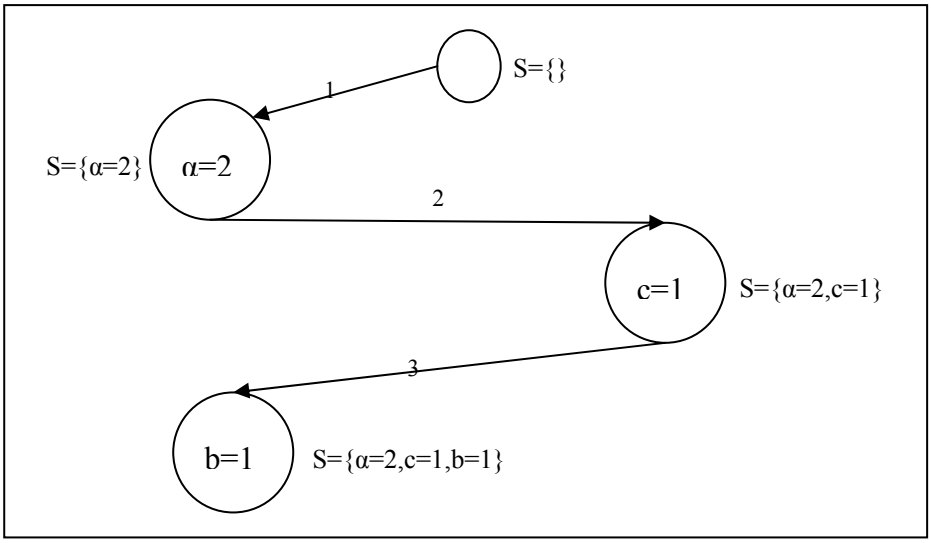
[4] Μια άλλη ονομασία είναι τεχνικές φιλτραρίσματος.

Στα προβλήματα ικανοποίησης περιορισμών χρησιμοποιείται επίσης και η έννοια της k-συνέπειας (k-consistency). Ένα πρόβλημα ικανοποίησης περιορισμών είναι k-συνεπές όταν για κάποιο μερικό σύνολο αποδόσεων τιμών σε k-1 μεταβλητές υπάρχει συνεπής απόδοση τιμής σε μια επιπλέον μεταβλητή. Σε κάποιο πρόβλημα που περιέχει μόνο δυαδικούς περιορισμούς, η συνέπεια κόμβων, αντιστοιχεί σε 1-συνέπεια, η συνέπεια τόξων σε 2-συνέπεια και η συνέπεια μονοπατιού σε 3-συνέπεια.

Παράδειγμα 3-3

Θα εφαρμόσουμε συνέπεια τόξων στο πρόβλημα που παρουσιάζεται στο Παράδειγμα 3-2, και κατόπιν θα το λύσουμε χρησιμοποιώντας χρονολογική αναζήτηση και τηρώντας τους απλούς κανονισμούς επιλογής μεταβλητών και τιμών που παρουσιάστηκαν στην ενότητα 3.3.2. Από τους περιορισμούς C_4 και C_5 προκύπτει ότι σε καμία πλήρη απόδοση τιμών δεν μπορεί να εμφανίζεται η απόδοση $a \leftarrow 1$ και γι' αυτό αφαιρούμε την τιμή 1 από το πεδίο τιμών της μεταβλητής α. Το πρόβλημα μετατρέπεται στο ισοδύναμο $V = \{a, b, c\}$, $D(a) = \{2\}, D(b) = D(c) = \{1, 2\}$. Επιλέγουμε την μεταβλητή α (συμμετέχει σε 4 περιορισμούς και έχει μόνο μια δυνατή τιμή) και της αποδίδουμε την τιμή 2. Επιλέγουμε την μεταβλητή c (συμμετέχει σε 4 περιορισμούς και έχει μόνο μια δυνατή τιμή) και της αποδίδουμε την τιμή 1. Επιλέγοντας κάποια τιμή από το πεδίο της μεταβλητής b το πρόβλημα λύνεται.

Στο Διάγραμμα 3-3 εμφανίζεται η λύση του προβλήματος εκτελώντας χρονολογική αναζήτηση. Το πρόβλημα λύνεται σε τρία βήματα αντί οκτώ, όπως φαίνεται στο Διάγραμμα 3-2. Το σύνολο λύση που προκύπτει είναι το $S = \{a \leftarrow 2, b \leftarrow 1, c \leftarrow 1\}$.



Διάγραμμα 3-3 : Έλεγχος συνέπειας και λύση προβλήματος

3.5 ΈΛΕΓΧΟΣ ΠΡΟΣ ΤΑ ΕΜΠΡΟΣ

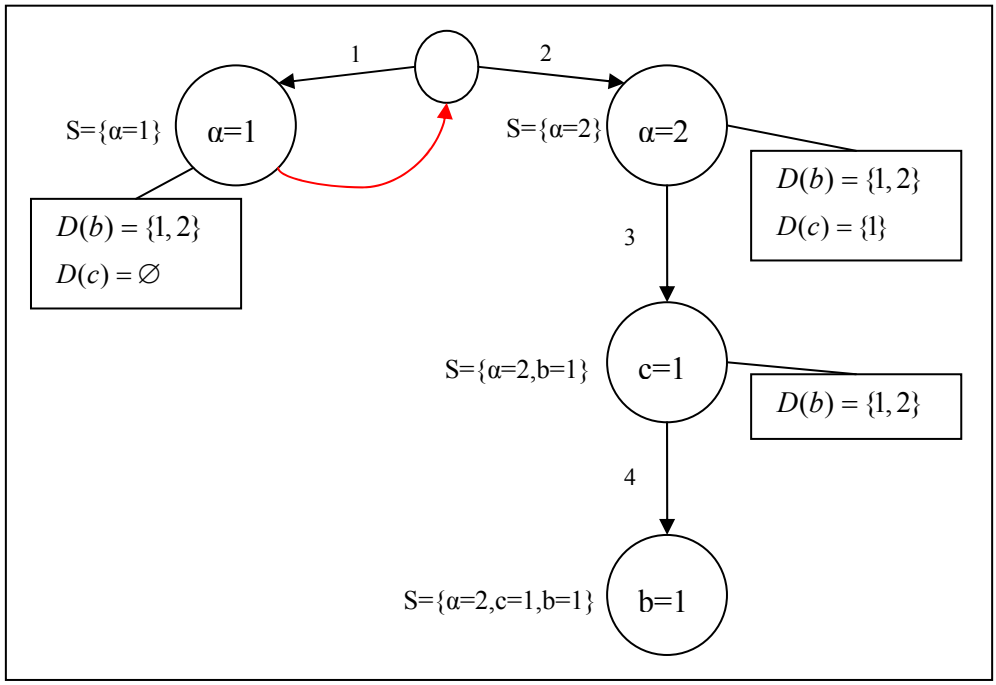
Έλεγχος προς τα εμπρός ονομάζεται ο έλεγχος συνέπειας και αναπαραγωγή περιορισμών (ενότητα 3.4) που εκτελείται κατά την διάρκεια της αναζήτησης. Αφού επιλεγεί μια τιμή για κάποια μεταβλητή τότε αναλύοντας του περιορισμούς που αφορούν την μεταβλητή προσπαθούμε να αφαιρέσουμε τιμές από τα πεδία τιμών των μεταβλητών που δεν έχουν πάρει τιμή. Αν κάποια μεταβλητή καταλήξει με πεδίο τιμών το κενό σύνολο τότε έχουμε ασυνέπεια και οπισθοχωρούμε. Κατά την οπισθοχώρηση οι αλλαγές που προκλήθηκαν από την απόδοση τιμής αναιρούνται. Ο έλεγχος προς τα εμπρός επιτρέπει την μείωση του χώρου αναζήτησης σε κάθε επιλογή που γίνεται καθώς και την γρήγορη ανακάλυψη ασυνεπειών και την αποφυγή

τους. Σε μια παραλλαγή αυτής της μεθόδου, μπορούμε να διατηρούμε κάποιο πίνακα με τις αλλαγές που προκάλεσε κάθε απόδοση τιμής με σκοπό να χρησιμοποιήσουμε αυτή την πληροφορία ώστε να οπισθοχωρούμε σε σημείο που διορθώνει την ασυνέπεια. (βλέπε σελ. 32)

Παράδειγμα 3-4

Έχουμε πρόβλημα ικανοποίησης περιορισμών με τρεις μεταβλητές $V = \{a, b, c\}$, $D(a) = D(b) = D(c) = \{1,\ 2\}$ και $C = \{C_1 = \{a \leftarrow 1, b \leftarrow 2\},\ C_2 = \{a \leftarrow 2, c \leftarrow 2\},$ $C_3 = \{b \leftarrow 2, c \leftarrow 2\},\ C_4 = \{a \leftarrow 1,\ c \leftarrow 1\},\ C_5 = \{a \leftarrow 1, c \leftarrow 2\}\}$.

Θα λύσουμε το πρόβλημα χρησιμοποιώντας χρονολογική αναζήτηση και έλεγχο προς τα εμπρός τηρώντας τους απλούς κανονισμούς επιλογής μεταβλητών και τιμών που παρουσιάστηκαν στην ενότητα 3.3.2.



Διάγραμμα 3-4 : Λύση του προβλήματος με έλεγχο προς τα εμπρός

Επιλέγουμε την μεταβλητή *α* (συμμετέχει σε 4 περιορισμούς) και της αποδίδουμε την τιμή *1*. Εκτελώντας έλεγχο προς τα εμπρός, από τον περιορισμό C_1 αφαιρείται η τιμή *2* από την μεταβλητή *b* και από τους περιορισμούς C_4 και C_5 οι τιμές *1* και *2* από την μεταβλητή *c*. Το πεδίο τιμών της μεταβλητής *c* είναι το κενό σύνολο, άρα έχουμε ασυνέπεια και οπισθοχωρούμε.

Αποδίδουμε την τιμή 2 στην μεταβλητή *α*. Εκτελώντας έλεγχο προς τα εμπρός, από τον περιορισμό C_2 αφαιρείται η τιμή *2* από την μεταβλητή *c*. Το πρόβλημα μετατρέπεται στο ισοδύναμο $V = \{a, b, c\}, D(a) = \{2\}, D(b) = \{1, 2\}, D(c) = \{1\}$. Δεν υπάρχει ασυνέπεια και συνεχίζουμε επιλέγοντας την μεταβλητή *c* (συμμετέχει σε 4 περιορισμούς και έχει μόνο μια δυνατή τιμή) και της αποδίδουμε την τιμή *1*. Χρησιμοποιώντας έλεγχο συνέπειας δεν είναι δυνατή καμία αφαίρεση τιμών από το πεδίο τιμών της μεταβλητής *b*. Επιλέγοντας κάποια τιμή από το πεδίο της μεταβλητής *b* το πρόβλημα λύνεται.

Στο Διάγραμμα 3-4 εμφανίζεται η λύση του προβλήματος εκτελώντας χρονολογική αναζήτηση και έλεγχο προς τα εμπρός. Το πρόβλημα λύνεται σε τέσσερα βήματα αντί οκτώ, όπως φαίνεται στο Διάγραμμα 3-2. Το σύνολο λύση που προκύπτει είναι το $S = \{a \leftarrow 2, b \leftarrow 1, c \leftarrow 1\}$.

3.6 ΘΕΩΡΙΑ ΑΣΥΝΕΠΕΙΩΝ

Η *θεωρία των ασυνεπειών* (theory of no-goods)[24] είναι μια τεχνική προσδιορισμού ασυνεπών συνόλων απόδοσης τιμών που μπορεί να ελαττώσει δραματικά το χώρο αναζήτηση κάνοντας έτσι ευκολότερη την λύση κάποιου προβλήματος ικανοποίησης περιορισμών.

Χρησιμοποιούμε τον συμβολισμό $A \downarrow V$ για να ορίσουμε την προβολή κάποιου μερικού συνόλου απόδοσης τιμών A στο σύνολο μεταβλητών V. Έτσι $A \downarrow V$ για $A = \{a \leftarrow 2, b \leftarrow 1, c \leftarrow 1\}$ και $V = \{a, c\}$ ισούται με $\{a \leftarrow 2, c \leftarrow 1\}$. Επίσης με C_v, οπού $C_v \subseteq C$, το σύνολο περιορισμών που περιέχει μόνο τους περιορισμούς στους οποίους συμμετέχουν μεταβλητές του συνόλου V και με V_c, όπου $V_c \subseteq V$, το σύνολο μεταβλητών οι οποίες συμμετέχουν σε περιορισμούς του συνόλου C.

Χρησιμοποιώντας τα πιο πάνω μια *ασυνέπεια* (no-good) ορίζεται ως το ζεύγος (A, J), όπου Α κάποιο σύνολο μερικών αποδόσεων τιμών σε μεταβλητές του συνόλου V και J υποσύνολο του V ($J \subseteq V$). Για μια συνέπεια (A, J) ισχύει επίσης ότι καμία λύση S, πλήρης απόδοση τιμών, του προβλήματος ικανοποίησης (V, C_j, D) δεν μπορεί να είναι υπερσύνολο της μερικής απόδοσης τιμών Α ($A \not\subset S$). Το σύνολο J ονομάζεται *αιτία* της ασυνέπειας, και περιέχει τις μεταβλητές των οποίων οι περιορισμοί προκαλούν την ασυνέπεια. Η ασυνέπεια $(\varnothing, J)$, ονομάζεται κενή ασυνέπεια και προσδιορίζει ότι το πρόβλημα είναι ασυνεπές και δεν έχει λύση.

Μεταξύ ασυνεπειών υπάρχουν οι εξής σχέσεις [24]:

1. Αν (A, J) είναι ασυνέπεια, τότε κάθε (A', J') όπου $A \subseteq A'$ και $J \subseteq J'$ είναι ασυνέπεια.
2. Αν (A, J) είναι ασυνέπεια, τότε $(A \downarrow J, J)$ είναι ασυνέπεια.
3. A μερική απόδοση τιμών, v μεταβλητή που δεν έχει πάρει τιμή και $\{A_1, ..., A_m\}$ τα μερικά σύνολα απόδοσης τιμών που προκύπτουν από την προσθήκη στο A και απόδοση όλων των δυνατών τιμών στην μεταβλητή v. Αν $(A_1, J_1), ..., (A_m, J_m)$ είναι ασυνέπειες τότε $(A, \cup_i J_i)$ είναι ασυνέπεια.

Όπως αναφέρθηκε πιο πάνω οι ασυνέπειες χρησιμοποιούνται για να ορίσουν μερικά σύνολα απόδοσης τιμών που πρέπει να αποφεύγονται γιατί δεν οδηγούν σε λύση. Από την σχέση 1 προκύπτει ότι ασυνέπειες όπου το Α έχει μικρό πλήθος είναι προτιμότερες γιατί περιορίζουν περισσότερο τον χώρο αναζήτησης.

Παράδειγμα 3-5

Έχουμε πρόβλημα ικανοποίησης περιορισμών με τρεις μεταβλητές $V = \{a, b, c\}$, $D(a) = D(b) = D(c) = \{1, 2\}$ και $C = \{C_1 = \{a \leftarrow 1, b \leftarrow 2\}, C_2 = \{a \leftarrow 2, c \leftarrow 2\}, C_3 = \{b \leftarrow 2, c \leftarrow 2\}, C_4 = \{a \leftarrow 1, c \leftarrow 1\}, C_5 = \{a \leftarrow 1, c \leftarrow 2\}\}$.

Κάθε περιορισμός υπονοεί και ασυνέπεια, έτσι $ng_1 = (C_1, \{a, b\})$, $ng_2 = (C_2, \{a, c\})$, $ng_3 = (C_3, \{b, c\})$, $ng_4 = (C_4, \{a, c\})$ και $ng_5 = (C_5, \{a, c\})$ είναι ασυνέπειες.

Χρησιμοποιώντας την σχέση 3 και τα ng_4 και ng_5 και επειδή $D(c) = \{1, 2\}$ προκύπτει ότι $(\{a \leftarrow 1\}, \{a, c\})$ είναι ασυνέπεια. Το σύνολο ασυνεπειών μετά την αντικατάσταση των ασυνεπειών ng_4 και ng_5 είναι $NG = \{ng_1 = (C_1, \{a, b\}), ng_2 = (C_2, \{a, c\}), ng_3 = (C_3, \{b, c\}), ng_4 = (\{a \leftarrow 1\}, \{a, c\})\}$.

3.6.1 Αλγόριθμος μάθησης ασυνεπειών

Ο αλγόριθμος μάθησης ασυνεπειών [24] προσπαθεί να δημιουργήσει ένα σύνολο ασυνεπειών το οποίο σταδιακά βελτιώνει, χρησιμοποιώντας τις σχέσεις 2 και 3, δημιουργώντας μικρότερες ασυνέπειες. Ο αλγόριθμος παίρνει ως είσοδο ένα μερικό σύνολο απόδοσης τιμών A και ένα σύνολο μεταβλητών V. Στην αρχική εκτέλεση το σύνολο Α είναι κενό και το σύνολο V περιέχει όλες τις μεταβλητές του προβλήματος. Ο αλγόριθμος τερματίζει όταν $V = \varnothing$ και επιστρέφει την λύση ή όταν δημιουργηθεί κενή ασυνέπεια και επιστρέφει το αίτιο της ασυνέπειας (σύνολο J) .

```
ng = ∅
MA(A,V)
  Αν V = ∅ τότε
      A είναι λύση
      Τερμάτισε και επέστρεψε το A
  αλλιώς
      x = επιλογή-μεταβλητης(V), J = ∅, B_j = 0
      Για κάθε v ∈ D(v) και B_j = 0
          A' = A ∪ {x ← v}
          K = έλεγχος-συνέπειας(A')
          Αν K = ∅ τότε
              J' = MA(A', V − {x})
              Αν x ∈ J' τότε
                  J = J ∪ J'
              αλλιώς
                  J = J', B_j = 1
              Τέλος Αν
          αλλιώς
              J = J ∪ K
          Τέλος Αν
      Τέλος Για κάθε
      Αν B_j = 0 τότε
          ng ← ng ∪ καταγραφή-ασυνέπειας((A ↓ J), J)
      Τέλος Αν
      Επέστρεψε το J
Τέλος Αν
```

Διάγραμμα 3-5 : Ο αλγόριθμος μάθησης ασυνεπειών

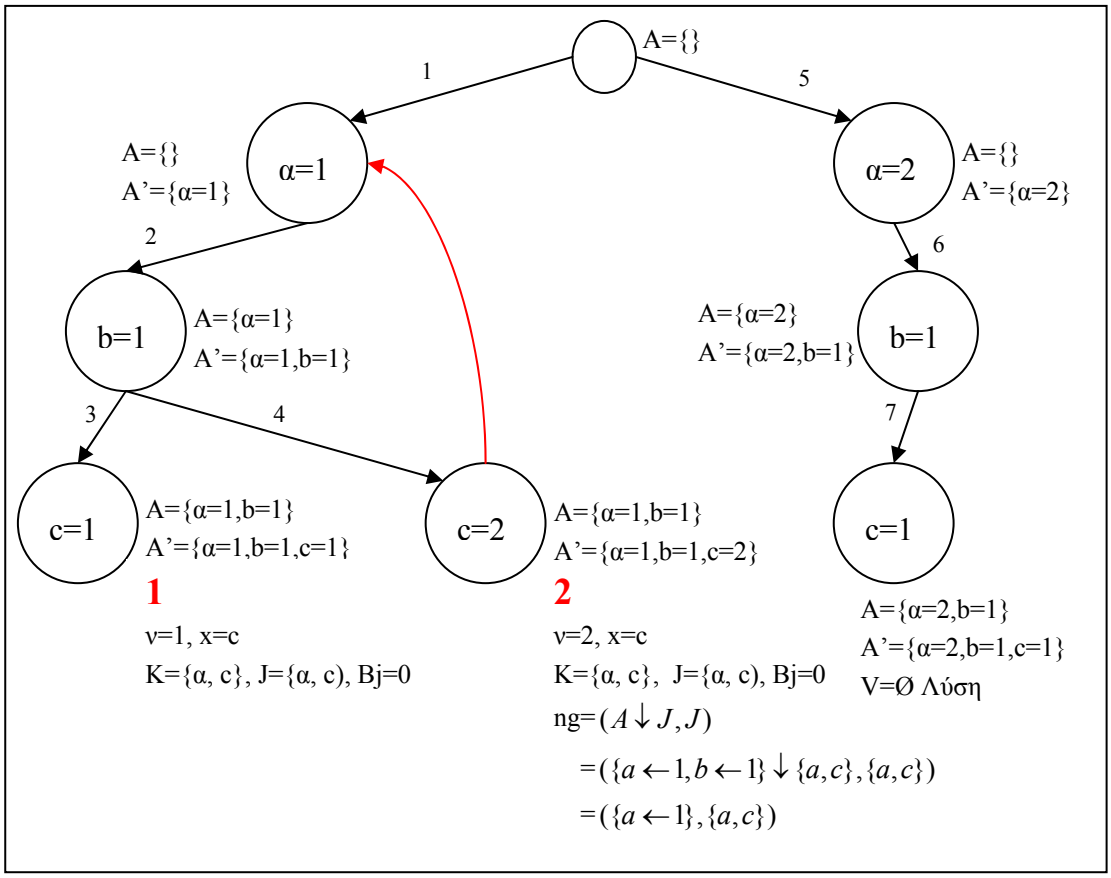
Ο αλγόριθμος κατά την εκτέλεση του χρησιμοποιεί τρεις βοηθητικές μεθόδους, την μέθοδο **επιλογή-μεταβλητής**, **έλεγχος-συνέπειας** και **καταγραφή-ασυνέπειας**. Η μέθοδος **επιλογή-μεταβλητής** χρησιμοποιείται για την επιλογή της επόμενης μεταβλητής που θα εξετάσουμε (εδώ θα μπορούσαμε να χρησιμοποιήσουμε του απλούς ευρετικούς κανόνες επιλογής μεταβλητών που αναφέραμε στην ενότητα 3.3.2). Η μέθοδος **έλεγχος-συνέπειας** ελέγχει μια μερική απόδοση τιμών κατά πόσον παραβιάζει κάποιον περιορισμό ή μια ασυνέπεια και επιστρέφει σε περίπτωση ασυνέπειας το αίτιο (αν πρόκειται για παραβίαση περιορισμού επιστρέφει ως αίτιο τις

μεταβλητές που συμμετέχουν στον περιορισμό, V_c) και σε περίπτωση που δεν παραβιάζεται τίποτα το κενό σύνολο. Η μέθοδος **καταγραφή-ασυνέπειας** προσθέτει στο σύνολο ασυνεπειών μια καινούργια ασυνέπεια προσπαθώντας ταυτόχρονα να την βελτιώσει. Στο Διάγραμμα 3-5 φαίνεται ο αλγόριθμος μάθησης ασυνεπειών.

Παράδειγμα 3-6

Έχουμε πρόβλημα ικανοποίησης περιορισμών με τρεις μεταβλητές $V = \{a, b, c\}$, $D(a) = D(b) = D(c) = \{1,\ 2\}$ και $C = \{C_1 = \{a \leftarrow 1, b \leftarrow 2\},\ C_2 = \{a \leftarrow 2, c \leftarrow 2\},\ C_3 = \{b \leftarrow 2, c \leftarrow 2\},\ C_4 = \{a \leftarrow 1,\ c \leftarrow 1\},\ C_5 = \{a \leftarrow 1, c \leftarrow 2\}\}$.

Πιο κάτω εμφανίζεται ή λύση του προβλήματος χρησιμοποιώντας τον αλγόριθμο μάθησης ασυνεπειών. Στα σημεία 1 και 2 (εμφανίζονται στο διάγραμμα με κόκκινους αριθμούς) η μερικές αποδόσεις τιμών παραβιάζουν περιορισμούς και ο αλγόριθμος δημιουργεί την ασυνέπεια $(\{a \leftarrow 1\}, \{a, c\})$.



Διάγραμμα 3-6 : Λύση του προβλήματος με μάθηση ασυνεπειών

Σε αντίθεση με την λύση του προβλήματος χρησιμοποιώντας χρονολογική αναζήτηση, όπως φαίνεται στο Διάγραμμα 3-2, ο αλγόριθμος μάθησης ασυνεπειών μόλις δημιούργησε την ασυνέπεια $(\{a \leftarrow 1\}, \{a, c\})$ δεν έλεγξε άλλες τιμές του *b* αλλά οπισθοχώρησε στην μεταβλητή *α* και επέλεξε άλλη τιμή. Αν για παράδειγμα το πεδίο τιμών της μεταβλητής *b* περιείχε 100 τιμές τότε ο αλγόριθμος μάθησης ασυνεπειών θα τις αγνοούσε, ενώ ο αλγόριθμος χρονολογικής αναζήτησης θα τις εξέταζε. Αυτό οφείλεται στην "άγνοια" του αλγόριθμου χρονολογικής αναζήτησης ότι η απόδοση της τιμής *1* στην μεταβλητή *α* δεν οδηγεί σε λύση, κάτι που ο αλγόριθμός μάθησης ασυνεπειών "γνωρίζει" και έτσι είναι σε θέση να αγνοήσει εντελώς όλο το χώρο που βρίσκεται κάτω από την απόδοση τιμής *1* στην μεταβλητή *α* μειώνοντας έτσι τον χώρο αναζήτησης.

Στο πιο πάνω παράδειγμα είδαμε ότι ο αλγόριθμος μάθησης ασυνεπειών οπισθοχωρεί σε περίπτωση αδιεξόδου όχι στην προηγούμενη μεταβλητή που προστέθηκε αλλά στην μεταβλητή που προκαλεί το αδιέξοδο. Αυτό γίνεται εφικτό χρησιμοποιώντας το αίτιο της ασυνέπειας (σύνολο *J*), και οπισθοχωρώντας στην μεταβλητή που βρίσκεται πιο κοντά στο σημείο της ασυνέπειας (δηλαδή η πιο πρόσφατη μεταβλητή που έχει επιλεγεί για απόδοση τιμής) και ανήκει ταυτόχρονα στο αίτιο της ασυνέπειας.

Στο Παράδειγμα 3-6 το αίτιο ήταν το σύνολο *{α,c}* και η ασυνέπεια σημειώθηκε κατά τον έλεγχο της μεταβλητής *c*. Έτσι ο αλγόριθμος οπισθοχώρησε απευθείας στην μεταβλητή *α*, αφού αυτή είναι η μόνη και ταυτόχρονα η κοντινότερη μεταβλητή που ανήκει στο αίτιο και δεν είναι η μεταβλητή *c*. Αυτός ο τύπος "έξυπνης" οπισθοχώρησης βασίζεται στην ιδέα της *κατευθυνόμενης από συγκρούσεις οπισθοχώρηση* (conflict-directed backjumping) [20] και χρησιμοποιείται επίσης και σε υλοποιήσεις της μεθόδου έλεγχος προς τα εμπρός. Ο αλγόριθμος μάθησης ασυνεπειών με μικρές τροποποιήσεις μπορεί να χρησιμοποιεί κατά τις επιλογές του έλεγχο προς τα εμπρός μειώνοντας έτσι περισσότερο τον χώρο αναζήτησης.

4 ΧΡΟΝΙΚΟΣ ΣΧΕΔΙΑΣΜΟΣ ΕΝΕΡΓΕΙΩΝ

4.1 ΕΙΣΑΓΩΓΗ

Όπως αναφέρθηκε στην εισαγωγή αυτής της εργασίας, ο σκοπός μας είναι η δημιουργία συστήματος που βοηθά τον χρήστη στις καθημερινές του εργασίες. Ένα τέτοιο σύστημα για να χρησιμοποιηθεί στον πραγματικό κόσμο πρέπει να μπορεί να αποφασίσει εκτός για το πώς θα κάνει κάτι (σχεδιασμός ενεργειών) και για το πότε θα κάνει κάτι (χρονοπρογραμματισμός). Στα δύο προηγούμενα κεφάλαια ασχοληθήκαμε με τον σχεδιασμό ενεργειών και τον χρονοπρογραμματισμό ως πρόβλημα ικανοποίησης περιορισμών. Σε αυτό το κεφάλαιο θα ασχοληθούμε με την ενοποίηση των δύο αυτών προβλημάτων σε ένα και την παρουσίαση μεθόδων λύσης τέτοιων προβλημάτων. Ένα τέτοιο ενοποιημένο πρόβλημα μπορεί να λυθεί είτε εκτελώντας σχεδιασμό ενεργειών και μετά χρονοπρογραμματισμό στο πλάνο που δημιουργήθηκε, είτε με σχεδιασμό ενεργειών και χρονοπρογραμματισμό ταυτόχρονα (πεπλεγμένο σχεδιασμό και χρονοπρογραμματισμό).

Σε γενικές γραμμές, στην πρώτη μέθοδο ο σχεδιαστής παράγει ένα πλάνο λύση το οποίο χρησιμοποιείται ως πρόβλημα για τον χρονοπρογραμματιστή, ο οποίος παράγει το τελικό πλάνο λύση. Το κυριότερο μειονέκτημα αυτής της μεθόδου είναι η πιθανότητα παραγωγής από την διαδικασία σχεδιασμού, πλάνων (σειριακών ή μη) που η διαδικασία χρονοπρογραμματισμού να μην μπορεί να λύσει, δηλαδή τα πλάνα να είναι μεν συνεπή σύμφωνα με της αρχές του σχεδιασμού ενεργειών αλλά μη συνεπή για τον χρονοπρογραμματιστή.

Ακόμη υπάρχει η περίπτωση, όλα τα δυνατά πλάνα λύσης που παράγει ο σχεδιαστής να μην λύνονται από τον χρονοπρογραμματιστή. Για να μπορέσουμε να βγάλουμε το συμπέρασμα ότι το πρόβλημα δεν λύνεται πρέπει να δημιουργηθούν όλα τα δυνατά πλάνα από πλευράς σχεδιαστή και να αποτύχουν στην φάση χρονοπρογραμματισμού. Κάτι τέτοιο δεν είναι ούτε επιθυμητό ούτε εφικτό για μεγάλα προβλήματα.

Ο πεπλεγμένος σχεδιασμός και χρονοπρογραμματισμός, λαμβάνει υπόψη τις παραμέτρους του προβλήματος και από την σκοπιά του σχεδιαστή και από την σκοπιά του χρονοπρογραμματιστή, αποφεύγοντας έτσι τα προβλήματα της πρώτης μεθόδου. Πρέπει όμως να σημειωθεί ότι το μέρος του προβλήματος που αφορά τον χρονοπρογραμματισμό είναι δυναμικό και αλλάζει με την εισαγωγή νέων ενεργειών από την πλευρά του σχεδιαστή. Έτσι επιλογές του χρονοπρογραμματιστή, περιορίζουν τις επιλογές του σχεδιαστή και καθώς το πρόβλημα από πλευράς χρονοπρογραμματισμού είναι δυναμικό, δεν είναι σίγουρο ότι οι επιλογές που κάνει ο χρονοπρογραμματιστής σε κάθε φάση είναι και σωστές.

Επίσης ακόμα και αν ο σχεδιαστής παράγει μη διατεταγμένα πλάνα κάθε φάση χρονοπρογραμματισμού θα μετατρέπει το πλάνο σε σειριακό ακυρώνοντας έτσι όλα τα πλεονεκτήματα ενός σχεδιαστή μη διατεταγμένων πλάνων. Είναι προφανές ότι είναι προτιμότερο τα πλάνα που δημιουργεί ο σχεδιαστής να είναι μη διατεταγμένα γιατί έτσι δίνεται περισσότερη ελευθερία στον χρονοπρογραμματιστή να λύσει και να βελτιώσει το πλάνο.

Στις επόμενες ενότητες θα παρουσιαστεί μια μέση λύση, που προσπαθεί να αποφύγει τα προβλήματα που παρουσιάζουν οι πιο πάνω μέθοδοι καθώς και μια μέθοδος αναπαράστασης των χρονικών προβλημάτων σχεδιασμού που βασίζεται στις μεταβλητές καταστάσεων (ενότητα 2.4.2). Σε αυτή την μέθοδο εκτελείται πρώτα σχεδιασμός ενεργειών με αναζήτηση στον χώρο των πλάνων (ενότητα 2.5.2) λαμβάνοντας όμως υπόψη τις παραμέτρους του δυναμικού προβλήματος χρονοπρογραμματισμού καθώς αυτό δημιουργείται. Με αυτό τον τρόπο τα πλάνα που

παράγονται από την φάση του σχεδιασμού μπορούν να λυθούν από την φάση του χρονοπρογραμματισμού. Επειδή δεν εκτελείται χρονοπρογραμματισμός αλλά λαμβάνονται υπόψη οι παράμετροι του προβλήματος χρονοπρογραμματισμού ο σχεδιαστής περιορίζεται στις επιλογές του πολύ λίγο και έτσι διατηρεί όλα τα πλεονεκτήματα σχεδιαστή μη διατεταγμένων πλάνων. Επίσης αν δεν υπάρχει λύση στο πρόβλημα η μέθοδος αυτή ανακαλύπτει από την φάση του σχεδιασμού την ασυνέπεια και τερματίζει.

4.2 ΑΝΑΠΑΡΑΣΤΑΣΗ ΠΡΟΒΛΗΜΑΤΩΝ ΜΕ ΤΗΝ ΧΡΗΣΗ ΧΡΟΝΙΚΩΝ

Όπως αναφέρθηκε και πιο πάνω η αναπαράσταση με την χρήση χρονικών βασίζεται στις μεταβλητές κατάστασης (state variables) [10], που σε αντίθεση με την κλασσική αναπαράσταση, περιγράφουν ένα πρόβλημα σχεδιασμού ενεργειών χρησιμοποιώντας συναρτήσεις. Στην ενότητα 2.4.2 αναφέραμε ότι μια λογική πρόταση, όπως in(α,r) από το Παράδειγμα 2-2, χρησιμοποιώντας σημειογραφία συναρτήσεων γράφετε ως $rloc : Robots \rightarrow Rooms$. Το αλφαριθμητικό $rloc$ ονομάζεται σύμβολο μεταβλητής κατάστασης και αναφέρεται σε μια συνάρτηση μεταβλητής κατάστασης.

Χρησιμοποιώντας αυτή την αναπαράσταση ο χρόνος μπορεί να αναπαρασταθεί σαν παράμετρος της συνάρτησης ορίζοντας έτσι μια σχέση μεταξύ τύπων αντικειμένων και χρόνου. Στην γενική της μορφή μια χρονική μεταβλητή κατάστασης ορίζεται ως $x : D_1^x \times ... \times D_n^x \times time \rightarrow D_{n+1}^x$ οπού:

- x αλφαριθμητικό, σύμβολο μεταβλητής κατάστασης.
- $D_1^x, ..., D_n^x, D_{n+1}^x$ τύποι αντικειμένων.
- $time$ ο χρόνος.

Για την περιγραφή σταθερών σχέσεων που δεν επηρεάζονται από τον χρόνο χρησιμοποιείται ένα σύνολο σταθερών σχέσεων (rigid relations). Μια σταθερή σχέση ορίζεται ως η ν-άδα $r(v_1,v_2,\ldots,v_\kappa)$ οπού

- r το σύμβολο της σταθερής σχέσης.
- $v_i \in D$, όπου D κάποιος τύπος αντικειμένων ή η ένωση κάποιων τύπων αντικειμένων.
- $r \subseteq D_1^r \times ... \times D_k^r$

Σε αυτή την αναπαράσταση, λόγω της ύπαρξης χρόνου, παύει να υπάρχει πλέον η έννοια της κατάστασης και στην θέση της θα χρησιμοποιήσουμε μια πιο ευέλικτη δομή, το χρονικό. Το χρονικό δεν παρουσιάζει ένα στιγμιότυπο του κόσμου σε μια χρονική στιγμή, όπως μια κατάσταση, αλλά την εξέλιξη του κόσμου σε βάθος χρόνου. Επίσης περιέχει τους περιορισμούς μεταξύ των χρονικών μεταβλητών και τον μεταβλητών αντικειμένων. Για την αναπαράσταση των αλλαγών και της διάρκειας των γεγονότων θα χρησιμοποιηθούν δύο κατηγορήματα λογικής δεύτερης τάξης, η *διατήρηση* (persistence) και το *γεγονός* (event). Τα δύο αυτά κατηγορήματα ονομάζονται χρονικές βεβαιώσεις (temporal assertions).

- Το *γεγονός* (event), ορίζεται ως $x@t : (v_1, v_2)$, $v_1 \neq v_2$ και αναπαριστά την στιγμιαία αλλαγή της τιμής της μεταβλητής κατάστασης x από v_1 σε v_2 την χρονική στιγμή t.
- Η *διατήρηση* (persistence), ορίζεται ως $x@[t_1,t_2) : v$, $t_1 < t_2$ και αναπαριστά την διατήρηση της τιμής της μεταβλητής κατάστασης x σε v κατά το διάστημα $[t_1,t_2)$.

4.2.1 Χρονικά

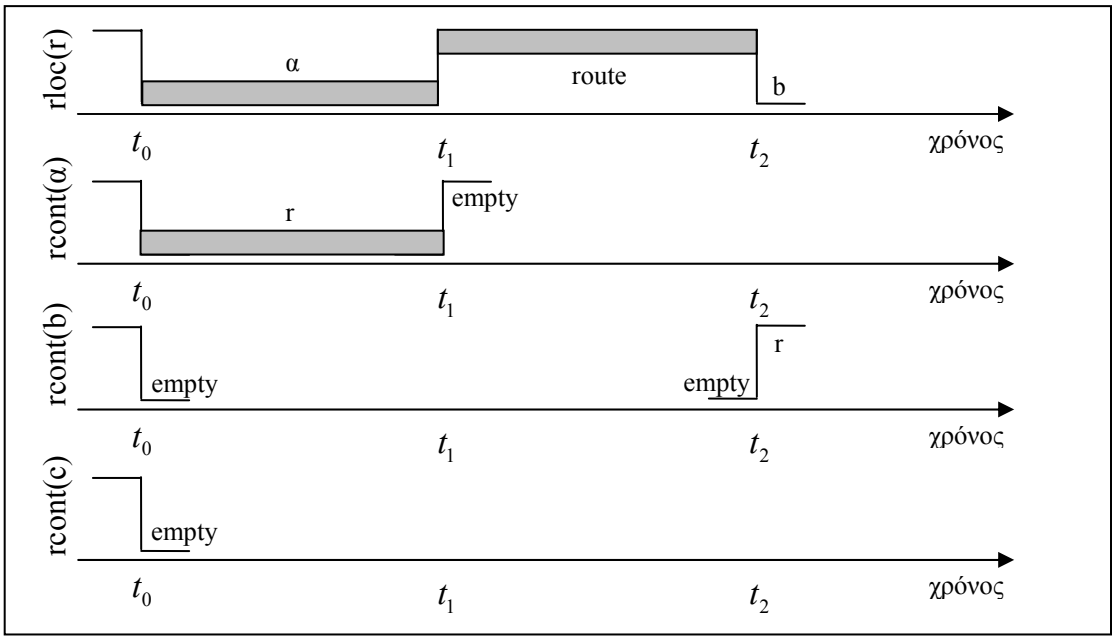
Ένα χρονικό κάποιου συνόλου μεταβλητών κατάστασης $\{x_1,...,x_n\}$ ορίζεται ως το ζεύγος $\Phi = (F, C)$ όπου:

- F το σύνολο των χρονικών βεβαιώσεων για τις μεταβλητές κατάστασης.
- C το σύνολο περιορισμών μεταξύ χρονικών μεταβλητών και μεταβλητών αντικειμένων.

Παράδειγμα 4-1

Το χρονικό πιο κάτω περιγράφει την εξέλιξη του κόσμου από την χρονική στιγμή t_0 μέχρι την χρονική στιγμή t_2. Η χρονική στιγμή t_0 θεωρείται η αρχή του κόσμου και οτιδήποτε πριν από αυτήν μπορεί να αγνοηθεί. Έτσι το $rloc(r)@t_0 : (b, a)$ υποδηλώνει ότι στην αρχή του κόσμου το ρομπότ *r* βρίσκεται στο δωμάτιο *α* και όχι ότι βρισκόταν στο δωμάτιο *b* και μετά στο *α*. Αυτό ισχύει μόνο για όλα τα γεγονότα που συμβαίνουν την χρονική στιγμή t_0.

$$
\begin{aligned}
(\{\ & rloc(r)@t_0 : (b, a),\\
& rloc(r)@[t_0, t_1) : a,\\
& rloc(r)@t_1 : (a, route),\\
& rloc(r)@[t_1, t_2) : route,\\
& rloc(r)@t_2 : (route, b),\\
& rcontains(a)@t_0 : (empty, r),\\
& rcontains(a)@[t_0, t_1) : r,\\
& rcontains(a)@t_1 : (r, empty),\\
& rcontains(b)@t_0 : (r, empty),\\
& rcontains(b)@t_2 : (empty, r),\\
& rcontains(c)@t_0 : (r, empty)\ \},\\
\{\ & adj(a,b), adj(a,c), adj(b,c),\ t_0 < t_1, t_0 < t_2, t_1 < t_2\})
\end{aligned}
$$



Διάγραμμα 4-1 : Γραφική αναπαράσταση χρονικού

Ένα χρονικό το οποίο περιέχει χρονικές βεβαιώσεις που αφορούν μόνο μια μεταβλητή κατάστασης ονομάζεται χρονική ακολουθία (timeline). Η χρονική ακολουθία της μεταβλητής κατάστασης *rloc* του πιο πάνω παραδείγματος είναι το πιο κάτω χρονικό:

$$
\begin{aligned}
(\{\ & rloc(r)@t_0 : (b,a),\\
& rloc(r)@[t_0,t_1) : a,\\
& rloc(r)@t_1 : (a,route),\\
& rloc(r)@[t_1,t_2) : route,\\
& rloc(r)@t_2 : (route,b)\},\\
\{\ & adj(a,b), adj(a,c), adj(b,c),\ t_0 < t_1, t_0 < t_2, t_1 < t_2\})
\end{aligned}
$$

4.2.2 Συνέπεια χρονικών

Ένα χρονικό Φ θεωρείται συνεπές εάν το πρόβλημα, ικανοποίησης περιορισμών (V,C) όπου V το σύνολο των μεταβλητών, χρονικών η αντικειμένων, που συμμετέχουν στο χρονικό Φ και C το σύνολο περιορισμών του χρονικού, είναι συνεπές και όλες οι χρονικές ακολουθίες που απαρτίζουν το χρονικό είναι συνεπείς, δηλαδή να μην περιέχουν *αβεβαιότητες*. Αβεβαιότητα ονομάζουμε την ύπαρξη αβεβαιότητας για την τιμή μιας μεταβλητής κατάστασης κάποια χρονική στιγμή. Είναι προφανές ότι αβεβαιότητα μεταξύ χρονικών βεβαιώσεων είναι δυνατό να υπάρξει μονό μεταξύ χρονικών βεβαιώσεων που ανήκουν στην ίδια χρονική ακολουθία, αφού αφορούν την ίδια μεταβλητή κατάστασης. Για την αποφυγή αβεβαιοτήτων σε μια χρονική ακολουθία μπορούμε να απαιτήσουμε το σύνολο περιορισμών C να περιέχει ή να υπονοεί *διαχωριστικούς περιορισμούς* (separation constraints) μεταξύ χρονικών βεβαιώσεων που είναι πιθανό να προκαλέσουν αβεβαιότητα σε μια χρονική ακολουθία.

Μια χρονική ακολουθία είναι συνεπής αν ισχύουν τα πιο κάτω:

1. Για κάθε ζεύγος βεβαιώσεων διατήρησης $x@[t_1,t_2) : v_1$ και $x@[t_3,t_4) : v_2$ που ανήκουν στο F πρέπει το σύνολο C να περιέχει ή να υπονοεί ένα από τους ακόλουθους περιορισμούς $\{t_2 \le t_3\}$ ή $\{t_4 \le t_1\}$ ή $\{v_1 = v_2\}$.
2. Για κάθε ζεύγος βεβαιώσεων γεγονότων $x@t_1 : (v_1,v_2)$ και $x@t_2 : (v_3,v_4)$ που ανήκουν στο F πρέπει το σύνολο C να περιέχει ή να υπονοεί ένα από τους ακόλουθους περιορισμούς $\{t_1 \ne t_2\}$ ή $\{v_1 = v_3, v_2 = v_4\}$.
3. Για μια βεβαίωση γεγονότος $x@t_1 : (v_1,v_2)$ και μια βεβαίωση διατήρησης $x@[t_2,t_3) : v_3$ που ανήκουν στο F πρέπει το σύνολο C να περιέχει ή να υπονοεί ένα από τους ακόλουθους περιορισμούς $\{t_1 < t_2\}$ ή $\{t_3 < t_1\}$ ή $\{t_1 = t_2, v_1 = v_3\}$ ή $\{t_1 = t_3, v_2 = v_3\}$.

4.2.3 Υποστήριξη και ενεργοποίηση

Ένα συνεπές χρονικό $\Phi = (F,C)$ *υποστηρίζει* μια χρονική βεβαίωση α ($x@t : (v,v')$ ή $x@[t,t') : v$), αν υπάρχει στο χρονικό Φ μια χρονική βεβαίωση β ($x@\tau : (w',w)$ ή $x@[\tau',\tau) : w$) και κάποιο σύνολο από διαχωριστικούς περιορισμούς c τέτοιο ώστε $\Phi \cup (\{a, x@[\tau,t) : v\}, \{w = v, \tau < t\} \cup c)$.

Δηλαδή για να υποστηρίζει ένα χρονικό μια χρονική βεβαίωση α που αλλάζει την τιμή ή διατηρεί την τιμή κάποιας μεταβλητής κατάστασης, το χρονικό πρέπει να

περιέχει μια χρονική βεβαίωση β που να θέτει ή να διατηρεί την τιμή της χρονικής μεταβλητής που εξετάζουμε σε αυτήν που η χρονική βεβαίωση α χρειάζεται. Επίσης η χρονική βεβαίωση β πρέπει συμβαίνει χρονικά πριν από την χρονική βεβαίωση α και να μπορούμε επίσης να προθέσουμε στο χρονικό και μια χρονική βεβαίωση διατήρησης που να διατηρεί την επιθυμητή τιμή μέχρι να συμβεί το α.

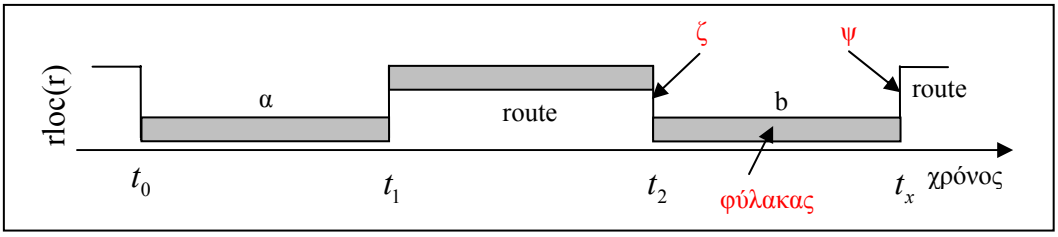
Η χρονική βεβαίωση β ονομάζεται *υποστηρικτής* της χρονικής βεβαίωσης α στο χρονικό Φ. Το $\delta = (\{a, x@[\tau, t) : v\}, \{w = v, \tau < t\} \cup c)$ ονομάζεται *ενεργοποιητής* της χρονικής βεβαίωσης α, και περιέχει την βεβαίωση α, μια βεβαίωση διατήρησης (*φύλακα*) που διατηρεί την απαραίτητη τιμή για την βεβαίωση α, το σύνολο διαχωριστικών περιορισμών μεταξύ βεβαίωσης α και *φύλακα* και ένα σύνολο διαχωριστικών περιορισμών μεταξύ των δύο βεβαιώσεων του *ενεργοποιητή* και των βεβαιώσεων του χρονικού Φ. Η βεβαίωση διατήρησης, *φύλακας* είναι το αντίστοιχο του *αιτιολογικού συνδέσμου* (ενότητα 2.5.2) και διατηρεί την τιμή της μεταβλητής κατάστασης μέχρι την εμφάνιση της χρονικής βεβαίωσης α. Επίσης δεν επιτρέπει την χρήση της χρονικής βεβαίωσης β ως υποστήριξη για άλλες χρονικές βεβαιώσεις που πιθανό να άλλαζαν την τιμή της μεταβλητής κατάστασης.

Ο ενεργοποιητής δ είναι ένα χρονικό και επειδή περιέχει τους απαραίτητους διαχωριστικούς περιορισμούς ώστε αν εισαχθεί στο συνεπές χρονικό Φ το αποτέλεσμα είναι ένα συνεπές χρονικό. Επίσης ο ενεργοποιητής δ δεν είναι μοναδικός επειδή είναι πιθανό να υπάρχουν πολλές βεβαιώσεις υποστήριξης β καθώς και περισσότερα από ένα σύνολα διαχωριστικών περιορισμών για κάποιες βεβαιώσεις α και β. Το σύνολο των ενεργοποιητών του α στο Φ γράφεται ως $\theta(a / \Phi)$

Παράδειγμα 4-2

Θέλουμε να δημιουργήσουμε τον ενεργοποιητή δ για την χρονική βεβαίωση $\psi = rloc(r)@t_x : (b, route)$ στο χρονικό που παρουσιάζεται στο Παράδειγμα 4-1. Από το χρονικό προκύπτει ότι η μονή χρονική βεβαίωση υποστήριξης είναι η $\zeta = rloc(r)@t_2 : (route, b)$.

Ο ενεργοποιητής δ θα είναι $(\{rloc(r)@t_x : (b, route), rloc@[t_2, t_x) : b\}, \{t_2 < t_x\} \cup c)$ και από τους κανόνες συνέπειας χρονικής ακολουθίας (ενότητα 4.2.2) χρειάζονται οι εξής περιορισμοί $t_1 < t_x$ (κανόνας 3), $t_1 \neq t_x$ (κανόνας 2) και $t_2 \neq t_x$ (κανόνας 2)



Διάγραμμα 4-2 : Η χρονική ακολουθία με τον ενεργοποιητή

Ένα συνεπές χρονικό $\Phi = (F, C)$ *υποστηρίζει* ένα σύνολο βεβαιώσεων ε αν κάθε βεβαίωση α_i που ανήκει στο ε, υποστηρίζεται από το χρονικό $(F \cup \varepsilon - \{a_i\}, C)$ με ενεργοποιητή δ_i τέτοιον ώστε $\varphi = \cup_i \delta_i$ και $\Phi \cup \varphi$ συνεπές χρονικό. Πρέπει να σημειωθεί ότι επιτρέπεται η υποστήριξη βεβαιώσεων που ανήκουν στο ε σε άλλες βεβαιώσεις του ε.

Ένα συνεπές χρονικό $\Phi = (F, C)$ *υποστηρίζει* ένα χρονικό $\Phi' = (F', C')$ αν το Φ υποστηρίζει το F' με ενεργοποιητή φ τέτοιο ώστε $(\varphi \cup \{\varnothing, C')) \cup \Phi$ συνεπές χρονικό.

4.3 ΣΧΕΔΙΑΣΜΟΣ ΕΝΕΡΓΕΙΩΝ ΜΕ ΤΗΝ ΧΡΗΣΗ ΧΡΟΝΙΚΩΝ

Στις προηγούμενες ενότητες παρουσιάστηκε το χρονικό ως μέθοδος αναπαράστασης κάποιο συστήματος σε βάθος χρόνου. Τα χρονικά επίσης χρησιμοποιούνται και για την αναπαράσταση των ενεργειών κάποιου προβλήματος σχεδιασμού ενεργειών. Με αυτό τον τρόπο η υποστήριξη κάποιας ενέργειας από κάποιο χρονικό και οι αλλαγές που επιφέρει σε ένα χρονικό με την εισαγωγή της καθορίζονται από τους κανόνες υποστήριξης που παρουσιάστηκαν στην προηγούμενη ενότητα.

4.3.1 Ενέργειες

Ένα χρονικό ενέργειας αναπαρίσταται ως το ζεύγος $o = (name(o), (F(o), C(o)))$.οπού

- name(o) το όνομα της ενέργειας. Έχει την μορφή $n(x_1,\ldots,x_k)$, όπου n ένα αλφαριθμητικό και x_1 έως x_k η μεταβλητές που εμφανίζονται στο o.
- $(F(o), C(o))$ το χρονικό που περιγράφει την ενέργεια.

Παράδειγμα 4-3

Το σχήμα ενέργειας move (Παράδειγμα 2-7) ορίζεται ως εξής

$move(R, L_1, L_2)$

$(\{\ rloc(R)@t_s : (L_1, route),$

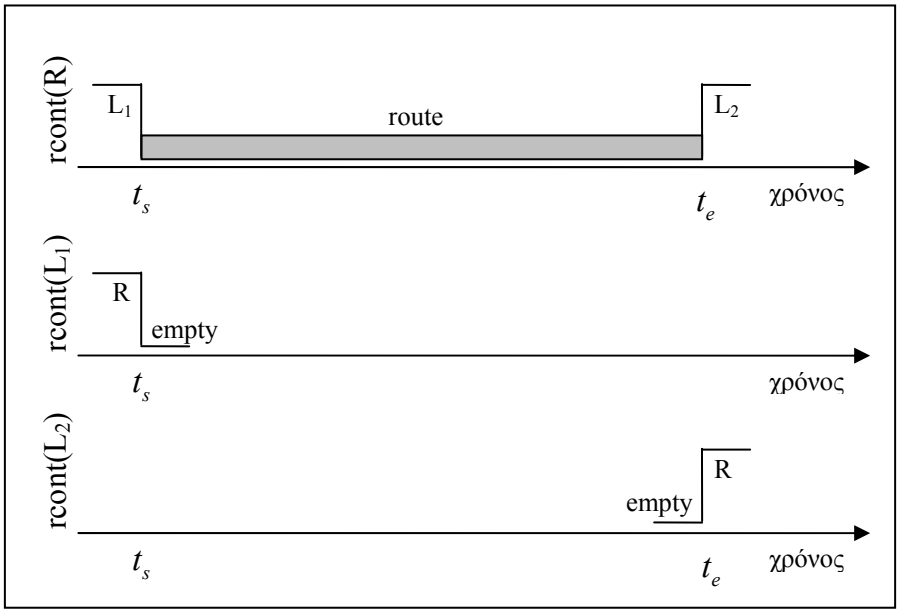
$rloc(R)@[t_s, t_e) : route,$

$rloc(R)@t_e : (route, L_2),$

$rcontains(L_1)@t_s : (R, empty),$

$rcontains(L_2)@t_e : (empty, R),$

$\{\ adj(L_1, L_2), t_s < t_e\})$



Διάγραμμα 4-3 : Το χρονικό της ενέργειας move

Ένα σχήμα ενέργειας γίνεται συγκεκριμένη ενέργεια με την απόδοση τιμής στις μεταβλητές αντικειμένου που περιέχει. Οι χρονικές μεταβλητές δεν παίρνουν τιμή γιατί κάτι τέτοιο θα σήμαινε χρονοπρογραμματισμό του προβλήματος περιορίζοντας έτσι τον σχεδιαστή ενεργειών, και γι' αυτό δεν εμφανίζονται στο όνομα του σχήματος ενέργειας.

4.3.2 Χρονικά προβλήματα σχεδιασμού

Ένα χρονικό πρόβλημα σχεδιασμού ενεργειών ορίζεται ως η τριάδα $P = (O, \Phi_0, \Phi_g)$ οπού:

- O το σύνολο χρονικών που περιγράφουν τις ενέργειες.
- Φ_0 το αρχικό χρονικό που περιγράφει την αρχική κατάσταση του κόσμου.
- Φ_g το χρονικό που περιέχει τις χρονικές βεβαιώσεις στόχους και τους περιορισμούς μεταξύ τους.

Η λύση κάποιου χρονικού προβλήματος σχεδιασμού ενεργειών είναι ένα σύνολο ενεργειών-χρονικών $\pi = \{a_1, ..., a_n\}$ τέτοιο ώστε $\Phi_g \subseteq \gamma(\Phi, \pi)$.

Σε αντιστοιχία με τον σχεδιασμό ως αναζήτηση στον χώρο των ημιτελών πλάνων θα μπορούσαμε να χαρακτηρίσουμε τον χώρο αναζήτησης που εξετάζουμε ως τον χώρο των ημιτελών χρονικών. Ο χώρος των ημιτελών χρονικών έχει ως κόμβους ημιτελή χρονικά, δηλαδή χρονικά τα οποία περιέχουν ελαττώματα και δεν είναι κατ' ανάγκη συνεπή. Τα τόξα μεταξύ των κόμβων είναι ενέργειες διόρθωσης των ελαττωμάτων του ημιτελούς χρονικού. Επίσης τα πλάνα λύσεις που παράγονται από αυτήν την διαδικασία δεν είναι ακολουθίες ενεργειών αλλά σύνολα χρονικών-ενεργειών με χρονικούς περιορισμούς διάταξης των ενεργειών.

Ελαττώματα ημιτελών χρονικών

Σε ένα ημιτελές χρονικό μπορούν να υπάρξουν δύο ειδών ελαττώματα. Στόχοι που δεν έχει ακόμα ικανοποιηθεί και απειλές μεταξύ χρονικών βεβαιώσεων. Μια χρονική βεβαίωση *α* απειλεί μια χρονική βεβαίωση *β* όταν το σύνολο C των περιορισμών δεν περιέχει ή υπονοεί τους απαραιτήτους διαχωριστικούς περιορισμούς που περιγράφονται στην ενότητα 4.2.2.

Ενέργειες διόρθωσης

Ανοικτοί στόχοι, δηλαδή στόχοι που δεν έχουν ακόμα ικανοποιηθεί, διορθώνονται με την εισαγωγή κάποιας ενέργειας ή με την χρήση κάποιας ενέργειας που υπάρχει ήδη στο ημιτελές πλάνο. Οι απειλές διορθώνονται με την εισαγωγή των κατάλληλων διαχωριστικών περιορισμών ώστε οι δύο χρονικές βεβαιώσεις που προκαλούν την απειλή να διαχωριστούν.

4.3.3 Ο αλγόριθμος CP

Στο *Automated planning, Theory and practice*[10] παρουσιάζεται ο αλγόριθμός *CP* που επιλύει χρονικά προβλήματα σχεδιασμού ακολουθώντας την μεθοδολογία του αλγόριθμου *POP* προσαρμοσμένου στην αναπαράσταση με χρονικά. Η βασική διαφορά του αλγορίθμου *CP* από τον αλγόριθμο *POP* είναι στα διάφορα σημεία επιλογής, μεταξύ διαφορετικών τρόπων διόρθωσης ελαττωμάτων του ημιτελούς χρονικού, που παρουσιάζονται δεν γίνεται επιλογή αλλά αποθήκευση των δυνατών επιλογών σε ένα σύνολο *K* (όμοια φιλοσοφία ακολουθείται και στο [21] αλλά με την χρήση χρονικών γράφων σχεδιασμού). Το σύνολο *K* είναι ένα σύνολο που περιέχει σύνολα με ενεργοποιητές. Κάθε σύνολο ενεργοποιητών που ανήκει στο *K* αναφέρεται σε ένα ελάττωμα και περιέχει του ενεργοποιητές που διορθώνουν το ελάττωμα στο πλάνο.

Έτσι στην πρώτη φάση του ο αλγόριθμος *CP* προσπαθεί να επιλύσει όλα τα ελαττώματα του χρονικού, αποθηκεύοντας όλους τους ενεργοποιητές στο σύνολο *K* μέχρι να μην υπάρχουν άλλα ελαττώματα στο χρονικό. Στη δεύτερη φάση του ο αλγόριθμος επιλέγει ένα ενεργοποιητή από κάποιο σύνολο τον προσθέτει στο χρονικό

και ελέγχει την συνέπεια του χρονικού. Αν το χρονικό δεν είναι συνεπές επιλέγει άλλο ενεργοποιητή από το ίδιο σύνολο. Αν το χρονικό είναι συνεπές τότε επιλέγει από άλλο σύνολο ενεργοποιητών κάποιο ενεργοποιητή, τον προσθέτει στο χρονικό και ελέγχει την συνέπεια. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να επιλεγεί ένας ενεργοποιητής από κάθε σύνολο και οι προσθήκη τους στο χρονικό να μην προκαλεί ασυνέπεια.

Η πιο πάνω διαδικασία μπορεί να αναπαρασταθεί και με ένα πρόβλημα ικανοποίησης περιορισμών με μία μεταβλητή για κάθε σύνολο ενεργοποιητών που υπάρχει στο K με πεδίο τιμών το σύνολο των αντίστοιχων ενεργοποιητών. Το σύνολο των περιορισμών του προβλήματος ικανοποίησης περιορισμών που δημιουργείται περιέχει μόνο ένα περιορισμό, την συνέπεια του χρονικού. Αυτό το πρόβλημα μπορεί να λυθεί με οποιαδήποτε μέθοδο επίλυσης προβλημάτων ικανοποίησης (ενότητα 3.3).

Για παράδειγμα αν λύναμε το πρόβλημα ικανοποίησης με χρονολογική οπισθοχώρηση θα επιλέγαμε ένα ενεργοποιητή από κάποιο σύνολο θα τον προσθέταμε στο χρονικό και θα ελέγχαμε την συνέπεια του χρονικού. Αν το χρονικό δεν είναι συνεπές επιλέγουμε άλλο ενεργοποιητή από το ίδιο σύνολο. Αν το χρονικό είναι συνεπές τότε επιλέγουμε από άλλο σύνολο ενεργοποιητών κάποιο ενεργοποιητή, τον προσθέτουμε στο χρονικό και ελέγχουμε την συνέπεια. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να επιλεγεί ένας ενεργοποιητής από κάθε σύνολο και οι προσθήκη τους στο χρονικό να μην προκαλεί ασυνέπεια. Το τελικό χρονικό που παράγεται είναι συνεπές και ικανοποιεί όλους του στόχους. Στην συνέχεια το χρονικό δίνεται στον χρονοπρογραμματιστή ο οποίος αποδίδει τιμές σε όλες τις χρονικές μεταβλητές δημιουργώντας το τελικό πλάνο λύση.

Ο αλγόριθμος CP λύνει ένα χρονικό πρόβλημα $P=(O,\Phi_0,\Phi_g)$ όπου $\Phi_0=(F_0,C_0)$ και $\Phi_g=(F_g,C_g)$ ξεκινώντας με $\Phi=(F_0,C_0\cup C_g)$, $G=F_g$, $\pi=\varnothing$ και $K=\varnothing$.

Στην πρώτη φάση οι ανικανοποίητοι στόχοι του προβλήματος ικανοποιούνται ως εξής:

- Αν α ανικανοποίητος στόχος και αν $\theta(a/\Phi)\neq\varnothing$ τότε υπάρχει υποστήριξη στο Φ για την βεβαίωση α. Το πρόβλημα ανανεώνεται ως εξής:
  $K\leftarrow K\cup\{\theta(a/\Phi)\}$
  $G\leftarrow G-\{a\}$
- Αν α ανικανοποίητος στόχος και αν $\theta(a/\Phi)=\varnothing$ τότε δεν υπάρχει υποστήριξη στο Φ για την βεβαίωση α. Επιλέγουμε ενέργεια o στην οποία υπάρχει υποστήριξη για το α. Το πρόβλημα ανανεώνεται ως εξής:
  $\pi\leftarrow\pi\cup\{o\}$
  $\Phi\leftarrow\Phi\cup(F(a),C(a))$
  $K\leftarrow K\cup\{\theta(a/\Phi)\}$
  $G\leftarrow G\cup F(o)$

Στην δεύτερη φάση εκτελείται η επιλογή ενεργοποιητών από το σύνολο K ως εξής:

- Αν $K\neq\varnothing$ τότε επιλέγουμε ένα ενεργοποιητή φ, $\varphi\in C$ και $C\in K$, τέτοιο ώστε $\Phi\cup\varphi$ συνεπές χρονικό και τον προσθέτουμε στο χρονικό. Το πρόβλημα ανανεώνεται ως εξής:
  $K\leftarrow K-C$
  $\Phi\leftarrow\Phi\cup\varphi$

CP (Φ, G, K, π)
  Αν $G = K = \varnothing$ τότε επέστρεψε το π
  Αν $G \neq \varnothing$ τότε
    επέλεξε $\alpha,\ \alpha \in G$
    Αν $\theta(\alpha/\Phi) \neq \varnothing$ τότε
      επέστρεψε $(CP\ (\Phi, G - \{\alpha\}, K \cup \theta(\alpha / \Phi), \pi))$
    αλλιώς
      $relevant \leftarrow \{o \mid o$ υποστηρίζει το $\alpha\}$
      Αν $relevant = \varnothing$ επέστρεψε ΑΠΟΤΥΧΙΑ
      επέλεξε $o,\ o \in relevant$
      επέστρεψε $(CP\ (\Phi \cup (F(o), C(o)), G \cup F(o), K \cup \theta(\alpha / \Phi), \pi \cup \{o\}))$
    Τέλος Αν
  Τέλος Αν
  Αν $K \neq \varnothing$ τότε
    επέλεξε $C,\ C \in K$
    $enablers \leftarrow \{\varphi \mid \varphi \in C$ και $\Phi \cup \varphi$ συνεπές$\}$
    Αν $enablers = \varnothing$ επέστρεψε ΑΠΟΤΥΧΙΑ
    επέλεξε $\varphi,\ \varphi \in enablers$
    επέστρεψε $(CP\ (\Phi \cup \varphi, G, K - C, \pi))$
  Τέλος Αν

Διάγραμμα 4-4 : Ο αλγόριθμος CP

4.3.4 Ο αλγόριθμος CPOP

Ο αλγόριθμος *CPOP* επιλύει χρονικά προβλήματα σχεδιασμού ακολουθώντας την μεθοδολογία του αλγορίθμου *POP*, όπως και ο αλγόριθμος *CP* (ενότητα 4.3.3). Ο αλγόριθμος προσπαθεί να ικανοποιήσει όλους τους ανικανοποίητους στόχους, καθοδηγώντας της επιλογές του με ευρετικό μηχανισμό που βασίζεται σε γράφο σχεδιασμού (ενότητα 2.5.3).

Ο αλγόριθμος ξεκινά επιλέγοντας τον πρώτο στόχο στο σύνολο *G* για τον οποίο δεν υπάρχει υποστήριξη στο χρονικό *Φ*. Αυτό βασίζεται στην αρχή της ελάχιστης δέσμευσης καθώς μια πρόωρη υποστήριξη βεβαίωσης από το χρονικό *Φ* θα περιόριζε τις δυνατότητες επιλογής του σχεδιαστή. Υποστήριξη στόχων από το χρονικό *Φ* γίνεται μόνο όταν δεν υπάρχει στο σύνολο *G* βεβαίωση που δεν υποστηρίζεται από το χρονικό *Φ*. Με αυτό τον τρόπο καθυστερούμε την υποστήριξη βεβαιώσεων από το αρχικό χρονικό *Φ* μέχρι αυτή να είναι αναγκαία.

Αν ο στόχος *α* που επιλέξαμε δεν υποστηρίζεται από το χρονικό *Φ* τότε ο αλγόριθμος δημιουργεί ένα σύνολο με όλες τις δυνατές ενέργειες που υποστηρίζουν την βεβαίωση *α*. Χρησιμοποιώντας τον ευρετικό μηχανισμό επιλέγεται μια από αυτές και προσθέτουμε στο χρονικό *Φ* την ενέργεια *o* και τον ενεργοποιητή *φ* της βεβαίωσης *α* (η βεβαίωση *α* περιέχετε στον ενεργοποιητή). Προσθέτουμε επίσης τις χρονικές βεβαιώσεις της ενέργειας *o* στο σύνολο ανικανοποίητων στόχων *G*.

Αν ο στόχος *α* που επιλέξαμε υποστηρίζεται από το χρονικό *Φ* τότε ο αλγόριθμος παράγει το σύνολο $\theta(a / \Phi)$ και επιλέγει βάση του ευρετικού μηχανισμού ένα

ενεργοποιητή φ, $\varphi \in \theta(a/\Phi)$ και τον προσθέτει στο χρονικό Φ. Το τελικό χρονικό που παράγεται είναι συνεπές και ικανοποιεί όλους του στόχους. Στην συνέχεια το χρονικό δίνεται στον χρονοπρογραμματιστή ο οποίος αποδίδει τιμές σε όλες τις χρονικές μεταβλητές δημιουργώντας το τελικό πλάνο λύση. Ο αλγόριθμος CPOP λύνει ένα χρονικό πρόβλημα $P = (O, \Phi_0, \Phi_g)$ όπου $\Phi_0 = (F_0, C_0)$ και $\Phi_g = (F_g, C_g)$ ξεκινώντας με $\Phi = (F_0, C_0 \cup C_g)$, $G = F_g$, $\pi = \varnothing$.

Στην πρώτη φάση οι ανικανοποίητοι στόχοι του προβλήματος, ικανοποιούνται ως εξής:

- Αν α ανικανοποίητος στόχος και αν $\theta(a/\Phi) \neq \varnothing$ τότε υπάρχει υποστήριξη στο Φ για την βεβαίωση α. Το πρόβλημα ανανεώνεται ως εξής:
  $\Phi \cup \varphi$, $\varphi \in \{\theta(a/\Phi)\}$
  $G \leftarrow G - \{a\}$
- Αν α ανικανοποίητος στόχος και αν $\theta(a/\Phi) = \varnothing$ τότε δεν υπάρχει υποστήριξη στο Φ για την βεβαίωση α. Επιλέγουμε ενέργεια o στην οποία υπάρχει υποστήριξη για το α, με ενεργοποιητή $\varphi \in \theta(a/o)$. Το πρόβλημα ανανεώνεται ως εξής:
  $\pi \leftarrow \pi \cup \{o\}$
  $\Phi \leftarrow \Phi \cup (F(a), C(a)) \cup \varphi$, και $\varphi \in \{\theta(a/\Phi)\}$
  $G \leftarrow G \cup F(o)$

```
CPOP (Φ, G, π)
  Αν G = ∅ τότε επέστρεψε το π
  a ← επέλεξε_στόχο(G)
  Αν θ(α/Φ) ≠ ∅ τότε
    επέστρεψε (CP (Φ ∪ φ, G − {α}, π))
    φ ← h(θ(α/Φ))
  αλλιώς
    relevant ← {o | o υποστηρίζει το α}
    φ ← h(θ(α/o))
    Αν relevant = ∅ επέστρεψε ΑΠΟΤΥΧΙΑ
    o ← h(relevant), o ∈ relevant
    επέστρεψε (CP (Φ ∪ (F(o), C(o)) ∪ φ, (G ∪ F(o)) − {a}, π ∪ {o}))
  Τέλος Αν
```

Διάγραμμα 4-5 : Ο αλγόριθμος CPOP

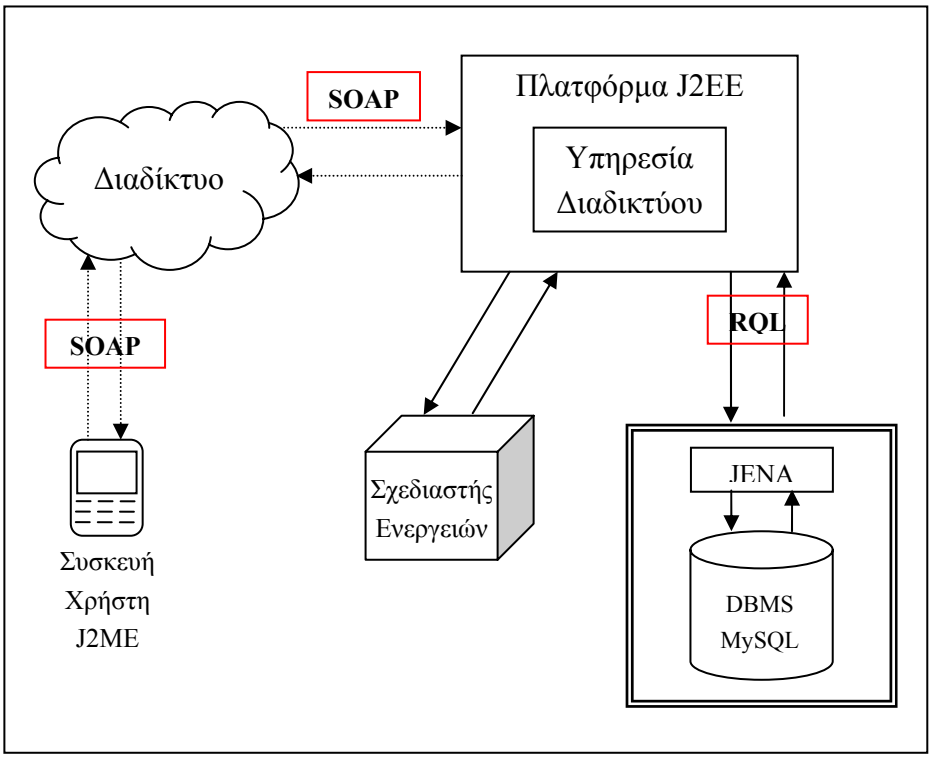
5 ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΟΣ

5.1 ΕΙΣΑΓΩΓΗ

Η υλοποίηση του συστήματος παροχής υπηρεσιών σχεδιασμού και χρονο-προγραμματισμού δεν ήταν μια εύκολη υπόθεση. Από την ξεκίνημα της υλοποίηση φάνηκε το όφελος από την χρήση προτύπων και ελεύθερων πλατφορμών ανάπτυξης, τόσο στην επιλογή πρωτόκολλων επικοινωνίας όσο και στην υλοποίηση των επιμέρους υποσυστημάτων. Γι' αυτό προσπαθήσαμε να χρησιμοποιήσουμε όπου ήταν εφικτό κοινώς αποδεκτά πρότυπα, όπως το πρωτόκολλο επικοινωνιών SOAP, την γλώσσα περιγραφής οντολογιών RDF και ελεύθερες πλατφόρμες εφαρμογών όπως J2ME και J2EE.

Το σύστημα υλοποιήθηκε χρησιμοποιώντας τα περιβάλλοντα ανάπτυξης εφαρμογών Visual Studio .NET, Netbeans και Sun One Studio, το περιβάλλον ανάπτυξης οντολογιών Protégé, το σύστημα διαχείρισης βάσεων δεδομένων MySQL και την βιβλιοθήκη διαχείρισης οντολογιών Jena. Για την υλοποίηση του σχεδιαστή ενεργειών χρησιμοποιήθηκε η γλώσσα προγραμματισμού C++, για την υλοποίηση της μικροεφαρμογής χρήστη η πλατφόρμα J2ME και η γλώσσα προγραμματισμού Java και για την υλοποίηση της υπηρεσίας διαδικτύου η πλατφόρμα J2EE και η γλώσσα προγραμματισμού Java.

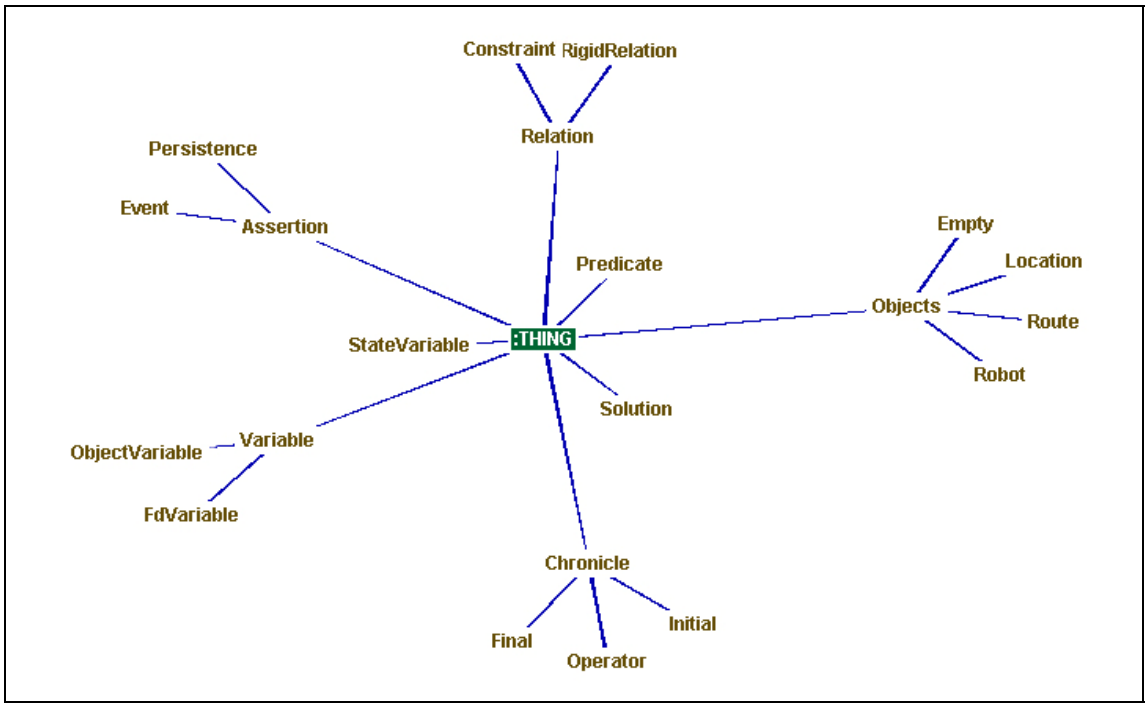
Για την επικοινωνία μεταξύ της υπηρεσίας διαδικτύου και της μικροεφαρμογής χρήστη χρησιμοποιείται το πρωτόκολλο SOAP και για την επικοινωνία μεταξύ της βάσης οντολογιών και της υπηρεσίας διαδικτύου η γλώσσα ερωτημάτων RQL. Η επικοινωνία μεταξύ υπηρεσίας διαδικτύου και σχεδιαστή ενεργειών γίνεται χρησιμοποιώντας μια παραλλαγή της γλώσσας περιγραφής προβλημάτων σχεδιασμού PDDL (βλέπε ενότητα 5.3).



Διάγραμμα 5-1 : Η υλοποίηση του συστήματος

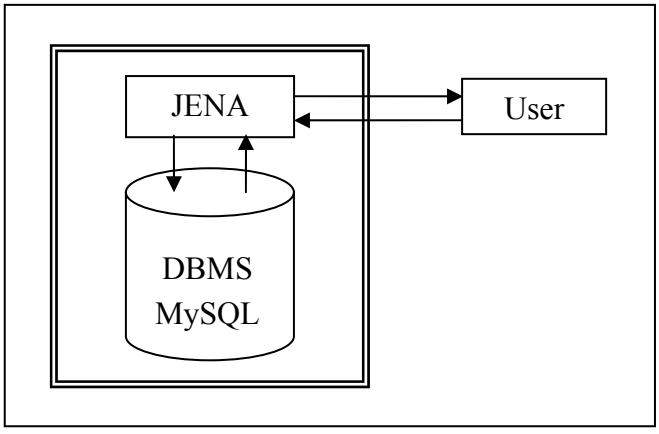
5.2 Η ΟΝΤΟΛΟΓΙΑ ΠΕΡΙΓΡΑΦΗΣ

Για την περιγραφή του κόσμου του προβλήματος και στιγμιότυπων του προβλήματος δημιουργήσαμε μια οντολογία περιγραφής χρονικών προβλημάτων (βλέπε Παράρτημα Α). Η οντολογία ορίστηκε χρησιμοποιώντας την γλώσσα περιγραφής οντολογιών RDF, που είναι διάλεκτος της γλώσσας περιγραφής XML. Για την ανάπτυξη της οντολογίας χρησιμοποιήθηκε η εφαρμογή ανάπτυξης οντολογιών Protégé. Η οντολογία περιγραφής χρονικών προβλημάτων χρησιμοποιείται για την αποθήκευση της πληροφορίας που αφορά τα προβλήματα χρονικού σχεδιασμού στην βάση οντολογιών του συστήματος καθώς και για την επικοινωνία μεταξύ της υπηρεσίας διαδικτύου και μικροεφαρμογής χρήστη.



Διάγραμμα 5-2 : Η ιεραρχία της οντολογίας περιγραφής

Η χρήση οντολογιών για επικοινωνία και περιγραφή των προβλημάτων, ήταν φυσική επιλογή καθώς βασίζονται σε κοινώς αποδεκτά πρότυπα και η έννοια της οντολογίας είναι η περιγραφή ιδεών. Επίσης χρησιμοποιώντας οντολογίες είναι δυνατή η εξαγωγή συμπερασμάτων από τα δεδομένα με προκαθορισμένες μεθόδους και η εύκολη επέκταση των δυνατοτήτων περιγραφής του συστήματος.



Διάγραμμα 5-3 : Η βάση οντολογιών

Για την αποθήκευση των οντολογιών χρησιμοποιήθηκε βάση οντολογιών που αποτελείται από το σύστημα διαχείρισης βάσεων δεδομένων MySQL και την βιβλιοθήκη διαχείρισης οντολογιών JENA. Η βιβλιοθήκη διαχείρισης οντολογιών JENA προσθέτει ένα επίπεδο αφαίρεσης σε μια βάση δεδομένων επεκτείνοντας την με μεθόδους διαχείρισης βάσης οντολογιών.

Επιπλέον η βιβλιοθήκη Jena παρέχει στου χρήστες της μεθόδους για την εξαγωγή συμπερασμάτων από τα αντικείμενα της οντολογίας κάτι που χρησιμοποιείται αρκετά κατά την μετατροπή της οντολογίας σε γλώσσα PDDL. Επίσης είναι δυνατή η επικοινωνία μεταξύ χρήστη και βάσης οντολογιών με την χρήση της γλώσσας ερωτημάτων RQL, που βασίζεται στην γλώσσα ερωτημάτων SQL, και επιτρέπει την εκτέλεση ερωτημάτων παρόμοιων με αυτά της γλώσσας SQL άλλα προσαρμοσμένων για διαχείριση οντολογιών.

5.3 Ο ΣΧΕΔΙΑΣΤΗΣ ΕΝΕΡΓΕΙΩΝ

Ο σχεδιαστής ενεργειών λόγω του είδους της εργασίας που εκτελεί (απαιτείται αρκετή υπολογιστική ισχύς) υλοποιήθηκε στην γλώσσα προγραμματισμού C++. Παρόλο που για την ανάπτυξη του χρησιμοποιήθηκε το περιβάλλον ανάπτυξης εφαρμογών Visual Studio .NET δόθηκε ιδιαίτερη προσοχή στην δυνατότητα εκτέλεσης της εφαρμογής του σχεδιαστή και σε άλλα λειτουργικά συστήματα (Linux, Solaris).

Ο σχεδιαστής ενεργειών δέχεται ως είσοδο δύο αρχεία κειμένου που περιγράφουν τον κόσμου του προβλήματος και το πρόβλημα χρησιμοποιώντας μια παραλλαγή της γλώσσας PDDL (ενότητα 2.4.1). Χρησιμοποιήθηκε η γλώσσα PDDL ως βάση της γλώσσας περιγραφής χρονικών προβλημάτων με την χρήση χρονικών λόγω της φυσικότητας της και της ευρείας χρήσης της στην περιγραφή προβλημάτων. Στο Διάγραμμα 5-4 φαίνεται η περιγραφή του κόσμου του προβλήματος μετακίνησης κάποιου ρομπότ μεταξύ τοποθεσιών και στο Διάγραμμα 5-5 η περιγραφή ενός προβλήματος. Για την δημιουργία του συμβολομεταφραστή (parser), ο οποίος βασίζεται σε συμβολομεταφραστή της γλώσσας PDDL, συγκεκριμένα στον συμβολομεταφραστή που χρησιμοποιεί ο σχεδιαστή HSP [2], χρησιμοποιήθηκαν οι εφαρμογές δημιουργίας συμβολομεταφραστών Yacc και Lex.

Έκτος από τα δύο αρχεία εισόδου ο χρήστης μπορεί να καθορίσει τον τύπο της αναζήτησης, το βάθος της αναζήτησης, τον ευρετικό μηχανισμό που θα χρησιμοποιηθεί κατά την αναζήτηση καθώς και την μορφή των αποτελεσμάτων Ο σχεδιαστής ενεργειών μπορεί να επιστρέψει τα αποτέλεσμα από την επίλυση κάποιου χρονικού προβλήματος είτε σε μορφή PDDL είτε ως αντικείμενο λύσης της οντολογίας περιγραφής (ενότητα 5.2). Η επιστροφή της λύσης στην γλώσσα περιγραφής PDDL γίνεται όταν ο σχεδιαστής ενεργειών χρησιμοποιείται ως αυτόνομη εφαρμογή ενώ όταν ο σχεδιαστής χρησιμοποιείται από την υπηρεσία διαδικτύου η λύση επιστρέφεται ως αντικείμενο οντολογίας αφού αυτός είναι ο τύπος δεδομένων που χρησιμοποιεί εσωτερικά η υπηρεσία διαδικτύου.

Ο σχεδιαστής ενεργειών περιέχει μια υλοποίηση του αλγόριθμου CPOP (ενότητα 4.3.4) που επιλύει χρονικά προβλήματα σχεδιασμού. Ο χειρισμός του χρονο-προγραμματιστικού μέρους του προβλήματος γίνεται από ένα επιλύτη προβλημάτων ικανοποίησης περιορισμών που υλοποίει την μέθοδο χρονολογικής αναζήτησης (ενότητα 3.3.2) που ενσωματώνει και έλεγχο προς τα εμπρός (ενότητα 3.5). Ο σχεδιαστής μπορεί επίσης να εκτελέσει αναζήτηση στον χώρο των ημιτελών χρονικών χρησιμοποιώντας τους αλγόριθμους πρώτα κατά βάθος, πρώτα κατά πλάτος, πρώτα στο καλύτερο και A*(ενότητα 2.5.1).

```
(define
(:domain export)
(:object_symbols Empty Route Robot Location)
(:predicates (adjacent ?x - Location ?y - Location ?z - temporal))
(:state_variables
        (lcontains (Location ) (Empty Robot ))
        (rloc (Robot ) (Route Location )))
(:operator Move ?L1 ?L2
        :parameters (?L1 ?L2 ?R ?T3)
        :assertions (
                event (rloc (?R ) ?T1 ?L1 route)
                persistence (rloc (?R ) ?T1 ?T2 route)
                event (rloc (?R ) ?T2 route ?L2)
                event (lcontains (?L1 ) ?T1 ?R empty)
                event (lcontains (?L2 ) ?T2 empty ?R))
        :rigid_relations ((adjacent ?L1 ?L2 ?T3))
        :constraints (
                (?T3 = ?T2 - ?T1) (?T1 < ?T2 ))))
```

Διάγραμμα 5-4 : Η περιγραφή του κόσμου του προβλήματος

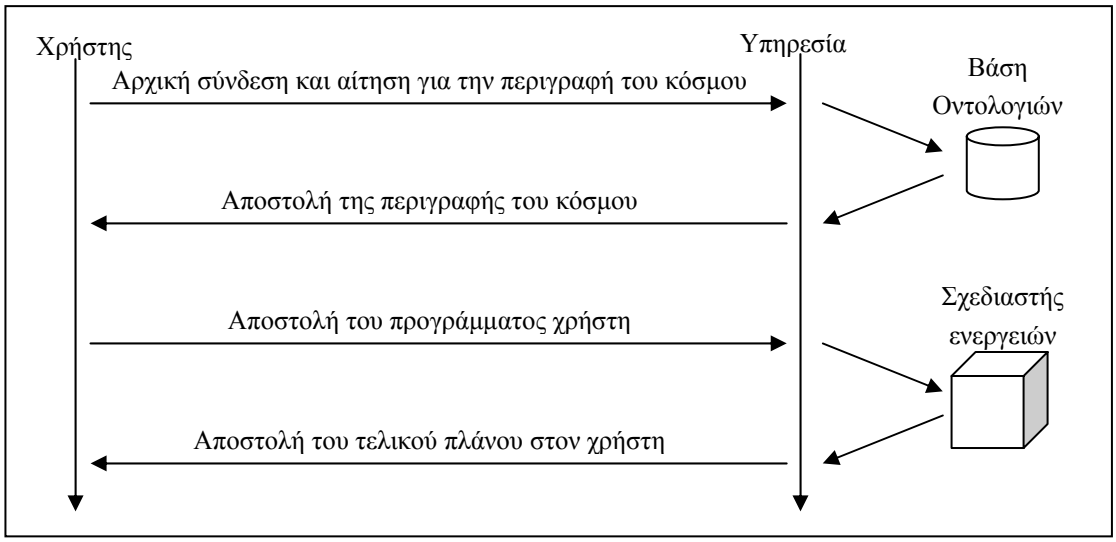
```
(define
(:problem export_problem)
(:domain export)
(:duration 100)
(:objects empty – Empty route – Route r1 – Robot loc2 loc3 loc1 – Location)
(:operator init
        :parameters (?Var0 ?Var1 ?Var2 ?Var3 ?Var4 ?Var5 ?Var6)
        :assertions (
                event (rloc (?Var3 ) ?Var0 ?Var2 ?Var6)
                event (lcontains (?Var6 ) ?Var0 ?Var1 ?Var3)
                event (lcontains (?Var4 ) ?Var0 ?Var1 ?Var1)
                event (lcontains (?Var5 ) ?Var0 ?Var1 ?Var1))
        :rigid_relations (
                (adjacent loc2 loc1 10) (adjacent loc3 loc2 20)
                (adjacent loc2 loc3 20) (adjacent loc1 loc2 10))
        :constraints (
                (?Var0 = 0) (?Var1 = empty) (?Var2 = route)
                (?Var3 = r1) (?Var4 = loc2) (?Var5 = loc3) (?Var6 = loc1)))
(:operator final
        :parameters (?TVar1 ?OVar1 ?TVar3 ?Var0 ?TVar2 ?OVar0 ?TVar4 ?Var1)
        :assertions (
                persistence (rloc (?Var1 ) ?TVar3 ?TVar4 ?OVar1)
                persistence (rloc (?Var0 ) ?TVar1 ?TVar2 ?OVar0))
        :constraints (
                (?OVar0 = loc2) (?TVar3 > 179) (?OVar1 = loc3)
                (?TVar2 = ?TVar1 + 30) (?TVar1 = 119) (?Var1 = r1)
                (?TVar4 = ?TVar3 + 20) (?Var0 = r1))))
```

Διάγραμμα 5-5 : Η περιγραφή του προβλήματος

5.4 Η ΥΠΗΡΕΣΙΑ ΔΙΑΔΙΚΤΥΟΥ

Η υπηρεσία διαδικτύου υλοποιήθηκε χρησιμοποιώντας το περιβάλλον ανάπτυξης εφαρμογών Sun One Studio. Η διαδικτυακή υπηρεσία βασίζεται στα πρότυπα υλοποίησης και επικοινωνίας υπηρεσιών διαδικτύου που καθορίζονται από τον οργανισμό W3C γι' αυτό μπορεί να χρησιμοποιηθεί από οποιανδήποτε εφαρμογή πελάτη με δυνατότητες επικοινωνίας με υπηρεσίες διαδικτύου.

Η υπηρεσία υλοποιήθηκε ως εφαρμογή της πλατφόρμας J2EE με την γλώσσα προγραμματισμού Java και για την λειτουργία της χρειάζεται κάποιος συμβατός με J2EE εξυπηρετητής, όπως Sun Application Server, Netscape Application Server, JBoss, Bea Logic κ.α.



Διάγραμμα 5-6 : Η φάσεις επικοινωνίας και λειτουργίας της υπηρεσίας

Ο συνηθισμένος τρόπος χρήσης της υπηρεσίας αποτελείται από τα εξής βήματα:

1. Ο χρηστής επικοινωνεί αρχικά με την υπηρεσία και ζητά την περιγραφή του κόσμου, που περιέχει τις γνωστές στην υπηρεσία τοποθεσίες και ενέργειες.
2. Η υπηρεσία εκτελεί ερωτήματα στην βάση οντολογιών και αποστέλλει την περιγραφή του κόσμου, που περιέχει τις επιτρεπτές ενέργειες και τοποθεσίες στον συνδεδεμένο χρήστη.
3. Ο χρήστης αφού δημιουργήσει το πρόγραμμα του, το αποστέλλει στην υπηρεσία.
4. Η υπηρεσία λαμβάνει το πρόγραμμα του χρήστη, το μετατρέπει στην γλώσσα περιγραφής προβλημάτων του σχεδιαστή (ενότητα 5.3), χρησιμοποιώντας ταυτόχρονα μεθόδους εξαγωγής συμπερασμάτων, και καλεί τον σχεδιαστή ενεργειών να το λύσει.
5. Ο σχεδιαστής λύνει το πρόβλημα και το επιστρέφει στην υπηρεσία σαν αντικείμενο της οντολογίας περιγραφής.
6. Η υπηρεσία αποστέλλει την λύση στον χρήστη.

Για την επικοινωνία με τους χρήστες της υπηρεσίας χρησιμοποιείται το πρωτόκολλο SOAP που είναι μια διάλεκτος βασισμένη στην γλώσσα περιγραφής XML (όπως και η γλώσσα περιγραφής οντολογιών RDF). Η επικοινωνία με την βάση οντολογιών γίνεται με ερωτήματα RQL. Η υπηρεσία διαδικτύου μετατρέπει αντικείμενα της οντολογίας περιγραφής προβλημάτων στην γλώσσα PDDL και καλεί τον σχεδιαστή ενεργειών με ορίσματα τα αρχεία που δημιούργησε.

Η υπηρεσία διαδικτύου δημοσιοποιεί προς τους χρήστες της, τις εξής μεθόδους και λειτουργίες:

- Αποστολή στον χρήστη της περιγραφής του κόσμου.
- Αποστολή στον χρήστη των δυνατών ενεργειών που μπορεί να χρησιμοποιήσει.
- Παραλαβή από τον χρήστη των ενεργειών του προβλήματος.
- Αποστολή στον χρήστη της λύσης του προβλήματος.

5.5 Η ΜΙΚΡΟΕΦΑΡΜΟΓΗ ΧΡΗΣΤΗ

Για την υλοποίηση της εφαρμογής χρήστη χρησιμοποιήθηκε το περιβάλλον ανάπτυξης μικροεφαρμογων (midlets) NetBeans, η πλατφόρμα μικροεφαρμογών J2ME και η γλώσσα προγραμματισμού Java. Πρόκειται για μια μικροεφαρμογή με δυνατότητες επικοινωνίας με υπηρεσίες διαδικτύου χρησιμοποιώντας το πρωτόκολλο SOAP. Για την γραφική αναπαράσταση του πλάνου αναπτύχθηκε βιβλιοθήκη γραφικών αντικειμένων (widgets) και για την διεπαφή χρήστη χρησιμοποιήθηκε η βιβλιοθήκη γραφικών Thinlet.

Η μικροεφαρμογή μπορεί να εκτελεστεί σε όλες της μικροσυσκευές που υποστηρίζουν το πρότυπο J2ME όπως κινητά τηλέφωνα και PDAs. Όταν χρήστης εκτελέσει την μικροεφαρμογή αυτή επικοινωνεί με την υπηρεσία διαδικτύου (ενότητα 5.4), λαμβάνει την περιγραφή του κόσμου και εμφανίζει το κεντρικό μενού επιλογών στον χρήστη.

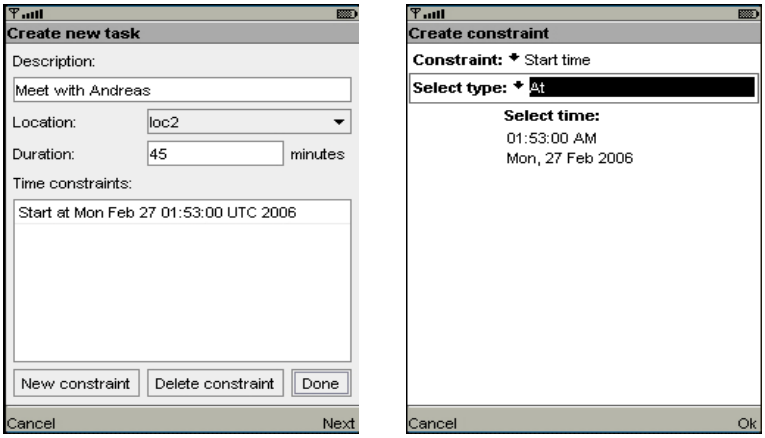
Το κεντρικό μενού περιέχει τις εξής επιλογές:

- New Task Εισαγωγή νέας εργασίας.
- Edit Task Αλλαγή στοιχείων κάποιας εργασίας.
- Delete Task Διαγραφή κάποιας εργασίας.
- Schedule Αποστολή των εργασιών στην υπηρεσία διαδικτύου.
- View Plan Εμφάνιση του τελικού πλάνου σε απλή μορφή.
- Plan viewer Εμφάνιση του πλάνου σε γράφο Gantt.

Διάγραμμα 5-7 : Το κεντρικό μενού της μικροεφαρμογής

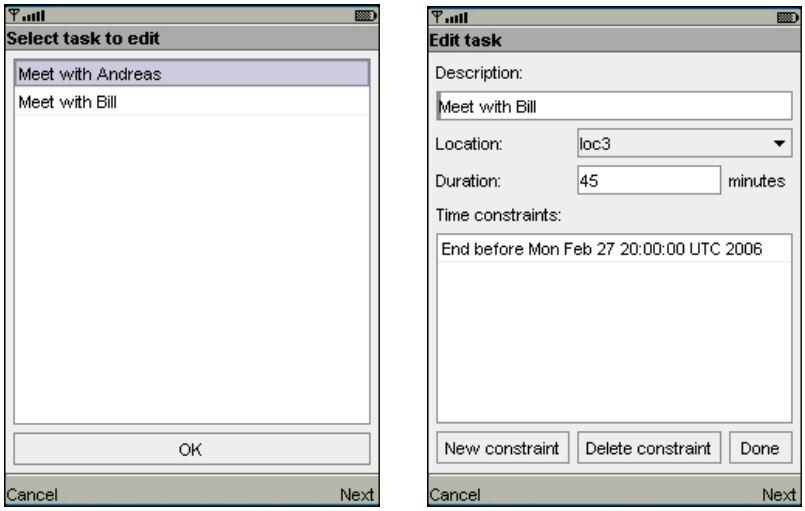
Για την εισαγωγή μιας ενέργειας απαιτείται από τον χρήστη να εισάγει μια σύντομη περιγραφή της ενέργειας, την τοποθεσία που θα εκτελεστεί την διάρκεια της καθώς και τυχόν περιορισμούς στον χρόνο έναρξης ή λήξης της εργασίας. Για την εισαγωγή περιορισμών εμφανίζεται η ανάλογη σελίδα στην οποία ο χρήστης μπορεί να ορίσει περιορισμούς στον χρόνο έναρξης ή λήξης. Οι τύποι των περιορισμών είναι πριν από

κάποια χρονική στιγμή, μετά από κάποια χρονική στιγμή, ή η έναρξη ή λήξη να συμβεί σε κάποια χρονική στιγμή.



Διάγραμμα 5-8 : Η σελίδες εισαγωγής νέας ενέργειας και περιορισμών

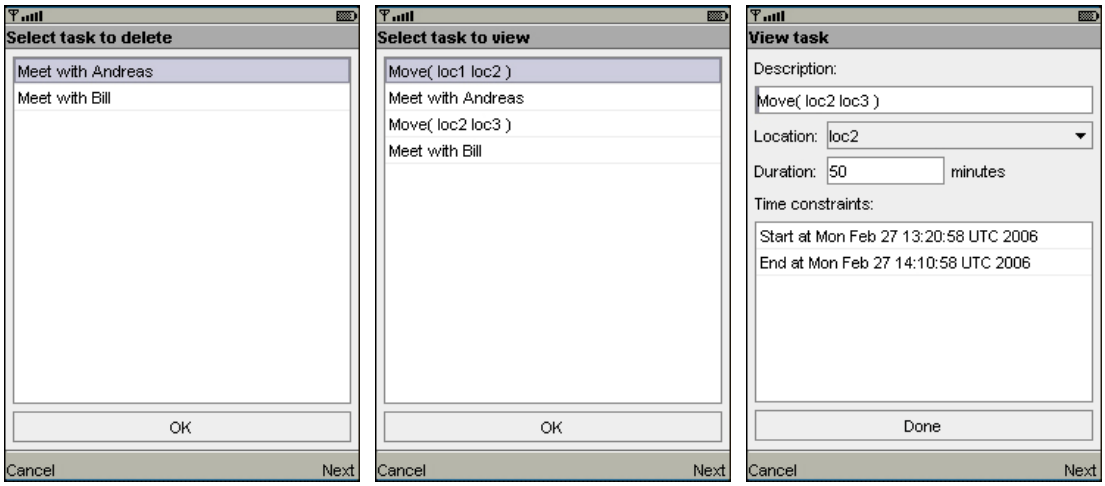
Αν κατά την εισαγωγή εργασιών έγινε κάποιο λάθος ή αν ο χρήστης επιθυμεί την αλλαγή κάποιων στοιχείων μιας εργασίας μπορεί να χρησιμοποιήσει την επιλογή Edit Task. Θα εμφανιστεί μια λίστα με τις εργασίες που ο χρήστης έχει εισάγει έως τώρα για να επιλέξει την εργασία που θα επεξεργαστεί. Αφού επιλέξει μια εργασία εμφανίζεται η σελίδα επεξεργασίας όπου ο χρήστης μπορεί να αλλάξει την σύντομη περιγραφή της εργασίας, την τοποθεσία, την διάρκεια και να δημιουργήσει ή να διαγράψει του περιορισμούς έναρξης ή λήξης της εργασίας.



Διάγραμμα 5-9 : Η λίστα εργασιών και η σελίδα επεξεργασίας

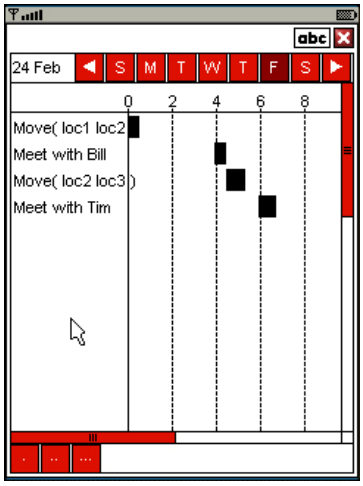
Για τη διαγραφή κάποιας εργασίας ο χρήστης μπορεί να επιλέξει από το κεντρικό μενού την εντολή Delete Task και θα εμφανιστεί η λίστα εργασιών. Αφού ο χρήστης επιλέξει μια εργασία πατώντας OK η εργασία διαγράφεται. Όταν ο χρήστης εισάγει όλες τις εργασίες του, τότε επιλέγοντας από το κεντρικό μενού την εντολή Schedule οι εργασίες αποστέλλονται στην υπηρεσία διαδικτύου για την δημιουργία του τελικού πλάνου.

Όταν το πλάνο επιστραφεί από την υπηρεσία διαδικτύου ενημερώνεται ο χρήστης και επιλέγοντας από το κεντρικό μενού την εντολή View plan θα εμφανιστεί η λίστα εργασιών του τελικού πλάνου. Επιλέγοντας μια εργασία από την λίστα εργασιών εμφανίζεται η σελίδα προβολής εργασίας που εμφανίζει τα στοιχεία της εργασίας με του περιορισμούς έναρξης και λήξης.



Διάγραμμα 5-10 : Η σελίδες διαγραφής εργασιών και προβολής τελικού πλάνου

Το τελικό πλάνο μπορεί επίσης να εμφανιστεί ως γράφημα Gantt επιλέγοντας από το κεντρικό μενού την εντολή Plan Viewer. Αυτό είναι εφικτό με την δημιουργία ειδικής βιβλιοθήκης γραφικών (σημειώνεται ότι ένα midlet δεν υποστηρίζει γραφικό περιβάλλον και αντικείμενα) η οποία όμως καταναλώνει αρκετή από την μνήμη της μικροσυσκευής. Η βιβλιοθήκη γραφικών υποστηρίζει κουμπιά, ετικέτες κειμένου, πεδία κειμένου, κυλιόμενες μπάρες, και ένα ποντίκι που κινείται χρησιμοποιώντας το χειριστήριο που βρίσκεται στο κέντρο των τηλεφωνικών συσκευών.



Διάγραμμα 5-11 : Η γραφική αναπαράσταση του τελικού πλάνου

6 ΕΠΙΛΟΓΟΣ

6.1 ΣΥΜΠΕΡΑΣΜΑΤΑ

Η παρούσα εργασία τελείωσε με την δημιουργία ενός πρότυπου συστήματος παροχής υπηρεσιών σχεδιασμού και χρονοπρογραμματισμού για τις καθημερινές εργασίες κάποιου χρήστη. Το σύστημα παράγει πλάνα που πετυχαίνουν τους στόχους που θέτει ο χρήστης μέσω μικροσυσκευής και τα επιστρέφει στον χρήστη. Όπως αναφέραμε και στην εισαγωγή της εργασίας το σύστημα είναι περιορισμένων δυνατοτήτων και απέχει πολύ από το ιδεατό έξυπνο σύστημα-βοηθός που περιγράψαμε στην εισαγωγή, αλλά μπορεί να αποτελέσει την βάση για ένα πλήρες σύστημα-βοηθό για τον άνθρωπο.

Το κυριότερο πρόβλημα που αντιμετωπίσαμε ήταν οι μεγάλες απαιτήσεις σε επεξεργαστική ισχύ για την λύση ακόμα και μικρών προβλημάτων κυρίως λόγω της συνδυαστικής έκρηξης που παρουσιάζουν αυτού του είδους τα προβλήματα. Φυσικά καθώς εξελίσσονται οι υπολογιστές και η τεχνητή νοημοσύνη μπορούμε να λύνουμε μεγαλύτερα προβλήματα στον ίδιο χρόνο και να χρησιμοποιούμε καλύτερους αλγόριθμους και πιο ενημερωμένους ευρετικούς μηχανισμούς (συνήθως όσο πιο ενημερωμένος είναι ένας ευρετικός μηχανισμός τόσο μεγαλύτερο το κόστος να τον δημιουργήσεις).

Παρόλο που στην βιβλιογραφία υπάρχουν αμέτρητα άρθρα που ασχολούνται με τον κλασσικό σχεδιασμό ενεργειών πολύ λίγα από αυτά ασχολούνται ικανοποιητικά με τον χρόνο σε κάποιο πρόβλημα και το κάθε ένα από διαφορετική σκοπιά. Πρόβλημα ήταν επίσης και η πολυπλοκότητα όχι μόνο του προβλήματος που προσπαθήσαμε να λύσουμε άλλα και του ίδιο του σχεδιαστή ενεργειών που αποτελείται από πάνω από τριάντα χιλιάδες γραμμές κώδικα.

Προβλήματα αντιμετωπίσαμε επίσης και κατά την ανάπτυξη της μικροεφαρμογής χρήστης λόγω των περιορισμένων δυνατοτήτων προγραμματισμού και μνήμης των μικροσυσκευών. Για παράδειγμα το πρότυπο MIDP (αφορά κινητά τηλέφωνα) στο οποίο βασίζεται η μικροεφαρμογή μας δεν υποστηρίζει γραφικό περιβάλλον, αναγκάζοντας μας έτσι να δημιουργήσουμε βιβλιοθήκη γραφικών ειδικά για την εφαρμογή μας.

Από την ενασχόληση με αυτό το θέμα καταλήξαμε στο συμπέρασμα ότι ένα πλήρες σύστημα βοηθός στις καθημερινές εργασίες που θα περιλάμβανε τεχνολογίες όπως σχεδιασμό ενεργειών, χρονοπρογραμματισμό, μηχανική μάθηση, ευφυείς πράκτορες, επικοινωνία με GPS και άλλα συστήματα πληροφοριών παρόλο που στην θεωρία μπορεί είναι υλοποιήσιμο, στην πράξη απαιτεί μεγάλη υπολογιστική ισχύ για τα σημερινά δεδομένα. Επίσης απαιτεί συνεργασία από διάφορους τομείς της τεχνητής νοημοσύνης και του κλάδου της πληροφορικής.

6.2 ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Το σύστημα για να θεωρηθεί ένα ολοκληρωμένο σύστημα βοηθός απαιτεί την προσθήκη επιπλέον δυνατοτήτων. Επιθυμητές δυνατότητες είναι η διατήρηση προφίλ του χρήστη που θα κωδικοποιεί τις προτιμήσεις και το ημερολόγιο του. Επίσης μια χρήσιμη επέκταση είναι η παρακολούθηση της εκτέλεσης του πλάνου από τον χρήστη σε συνδυασμό με μηχανική μάθηση και ταυτόχρονη προσαρμογή του προφίλ του χρήστη ώστε να αντικατοπτρίζει τις επιθυμίες του. Χρήσιμη είναι επίσης η δυνατότητα επικοινωνίας μεταξύ παρόμοιων συστημάτων, για την δημιουργία καλύτερων και πιο ενημερωμένων με την κατάσταση του κόσμου προγραμμάτων του

χρήστη, καθώς και η δυνατότητα εκτέλεσης από την συσκευή κάποιων εργασιών εκ μέρους του χρήστη.

Βελτίωση μπορεί να γίνει και στον τρόπο που δουλεύει το σύστημα, όπως την εισαγωγή σε υπάρχον πλάνο νέων ενεργειών χωρίς επαναυπολογισμό όλου του πλάνου. Επίσης το υπάρχον σύστημα περιέχει αρκετά σημεία που μπορούν να βελτιωθούν. Όπως η χρήση του αλγόριθμου μάθησης ασυνεπειών (ενότητα 3.6.1) που με μικρές τροποποιήσεις μπορεί να χρησιμοποιεί κατά τις επιλογές του έλεγχο προς τα εμπρός μειώνοντας έτσι περισσότερο τον χώρο αναζήτησης. Επίσης η χρήση πιο ενημερωμένου ευρετικού μηχανισμού, όπως σειριακού γράφου σχεδιασμού (ενότητα 2.6.3) θα βελτίωνε την απόδοση του συστήματος. Τέλος υπάρχουν πολλά σημεία που μπορεί να βελτιωθεί ο κώδικας που υλοποιεί τον αλγόριθμο CPOP για την βελτίωση της ταχύτητας του συστήματος.

ΠΑΡΑΡΤΗΜΑ Α

Στην πιο κάτω λίστα εμφανίζεται η ιεραρχία της οντολογίας περιγραφής προβλημάτων.

- Assertion
  Η γενική κλάση περιγραφής χρονικών βεβαιώσεων.
  - Event
    Κλάση περιγραφής βεβαιώσεων γεγονότων.
  - Persistence
    Κλάση περιγραφής βεβαιώσεων διατήρησης.
- Chronicle
  Η γενική κλάση περιγραφής χρονικών.
  - Final
    Η κλάση περιγραφής του χρονικού στόχου.
  - Initial
    Η κλάση περιγραφής του αρχικού χρονικού.
  - Operator
    Η κλάση περιγραφής μη συγκεκριμένων ενεργειών.
    - GroundedOperator
      Η κλάση περιγραφής συγκεκριμένων ενεργειών.
- Objects
  Η γενική κλάση περιγραφής αντικειμένων του προβλήματος.
  - Empty
  - Location
  - Robot
  - Route
- Predicate
  Η κλάση περιγραφής λογικών προτάσεων.
- Relation
  Η γενική κλάση περιγραφής σχέσεων μεταξύ μεταβλητών χρόνου και μεταξύ μεταβλητών αντικειμένου.
  - Constraint
    Η κλάση περιγραφής των περιορισμών μεταξύ μεταβλητών.
    - FdConstraint
      Η κλάση περιγραφής των χρονικών περιορισμών.
    - ObjectConstaint
      Η κλάση περιγραφής των περιορισμών μεταξύ αντικειμένων.
  - RigidRelation
    Η κλάση περιγραφής των σταθερών σχέσεων.
- Solution
  Η κλάση περιγραφής των λύσεων του προβλήματος.
- StateVariable
  Η κλάση περιγραφής των μεταβλητών κατάστασης.
- Variable
  Η γενική κλάση περιγραφής των μεταβλητών του προβλήματος.
  - FdVariable
    Η κλάση περιγραφής των χρονικών μεταβλητών του προβλήματος.
  - ObjectVariable
    Η κλάση περιγραφής των μεταβλητών αντικειμένου του προβλήματος.

Πιο κάτω εμφανίζεται το σχήμα της οντολογίας στην γλώσσα περιγραφής RDF.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [
       <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
       <!ENTITY kb 'http://localhost/kb#'>
       <!ENTITY a 'http://protege.stanford.edu/system#'>
       <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
]>
<rdf:RDF xmlns:kb="&kb;"
       xmlns:rdf="&rdf;"
       xmlns:a="&a;"
       xmlns:rdfs="&rdfs;">
<rdfs:Class rdf:about="&kb;Adjacent"
       rdfs:label="Adjacent">
      <rdfs:subClassOf rdf:resource="&kb;RigidRelation"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Assertion"
       rdfs:label="Assertion">
      <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Chronicle"
       rdfs:label="Chronicle">
      <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Constraint"
       rdfs:label="Constraint">
      <rdfs:subClassOf rdf:resource="&kb;Relation"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Empty"
       rdfs:label="Empty">
      <rdfs:subClassOf rdf:resource="&kb;Objects"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Event"
       rdfs:label="Event">
      <rdfs:subClassOf rdf:resource="&kb;Assertion"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;FdConstraint"
       rdfs:label="FdConstraint">
      <rdfs:subClassOf rdf:resource="&kb;Constraint"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;FdConstraint_t"
       rdfs:label="FdConstraint_t">
      <rdfs:subClassOf rdf:resource="&kb;FdConstraint"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;FdVariable"
       rdfs:label="FdVariable">
      <rdfs:subClassOf rdf:resource="&kb;Variable"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Final"
       rdfs:label="Final">
      <rdfs:subClassOf rdf:resource="&kb;Chronicle"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;GroundFdVariable"
       rdfs:label="GroundFdVariable">
      <rdfs:subClassOf rdf:resource="&kb;FdVariable"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;GroundObjectVariable"
       rdfs:label="GroundObjectVariable">
      <rdfs:subClassOf rdf:resource="&kb;ObjectVariable"/>
```

```
</rdfs:Class>

<rdfs:Class rdf:about="&kb;GroundedOperator"
       rdfs:label="GroundedOperator">
      <rdfs:subClassOf rdf:resource="&kb;Operator"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Initial"
       rdfs:label="Initial">
      <rdfs:subClassOf rdf:resource="&kb;Chronicle"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Location"
       rdfs:label="Location">
      <rdfs:subClassOf rdf:resource="&kb;Objects"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;ObjectConstaint"
       rdfs:label="ObjectConstaint">
      <rdfs:subClassOf rdf:resource="&kb;Constraint"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;ObjectVariable"
       rdfs:label="ObjectVariable">
      <rdfs:subClassOf rdf:resource="&kb;Variable"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Objects"
       rdfs:label="Objects">
      <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Operator"
       rdfs:label="Operator">
      <rdfs:subClassOf rdf:resource="&kb;Chronicle"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Persistence"
       rdfs:label="Persistence">
      <rdfs:subClassOf rdf:resource="&kb;Assertion"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Predicate"
       rdfs:label="Predicate">
      <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Relation"
       rdfs:label="Relation">
      <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;RigidRelation"
       rdfs:label="RigidRelation">
      <rdfs:subClassOf rdf:resource="&kb;Relation"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Robot"
       rdfs:label="Robot">
      <rdfs:subClassOf rdf:resource="&kb;Objects"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Route"
       rdfs:label="Route">
      <rdfs:subClassOf rdf:resource="&kb;Objects"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;Solution"
       rdfs:label="Solution">
      <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="&kb;StateVariable"
       rdfs:label="StateVariable">
      <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
```

```
</rdfs:Class>

<rdfs:Class rdf:about="&kb;Variable"
       rdfs:label="Variable">
      <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
<rdf:Property rdf:about="&kb;adjX"
       rdfs:label="adjX">
      <rdfs:domain rdf:resource="&kb;Adjacent"/>
      <rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;adjY"
       rdfs:label="adjY">
      <rdfs:domain rdf:resource="&kb;Adjacent"/>
      <rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;assertionID"
       rdfs:label="assertionID">
      <rdfs:domain rdf:resource="&kb;Assertion"/>
      <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;duration"
       rdfs:label="duration">
      <rdfs:domain rdf:resource="&kb;Adjacent"/>
      <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;durationVar"
       rdfs:label="durationVar">
      <rdfs:domain rdf:resource="&kb;Adjacent"/>
      <rdfs:range rdf:resource="&kb;FdVariable"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasAssertion"
       rdfs:label="hasAssertion">
      <rdfs:range rdf:resource="&kb;Assertion"/>
      <rdfs:domain rdf:resource="&kb;Chronicle"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasChronicle"
       rdfs:label="hasChronicle">
      <rdfs:range rdf:resource="&kb;Chronicle"/>
      <rdfs:domain rdf:resource="&kb;Solution"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasDuration"
       rdfs:label="hasDuration">
      <rdfs:domain rdf:resource="&kb;Initial"/>
      <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasFdLBound"
       rdfs:label="hasFdLBound">
      <rdfs:domain rdf:resource="&kb;FdVariable"/>
      <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasFdOffset"
       rdfs:label="hasFdOffset">
      <rdfs:domain rdf:resource="&kb;FdConstraint"/>
      <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasFdUBound"
       rdfs:label="hasFdUBound">
      <rdfs:domain rdf:resource="&kb;FdVariable"/>
      <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
```

```
<rdf:Property rdf:about="&kb;hasFdValue"
       rdfs:label="hasFdValue">
      <rdfs:domain rdf:resource="&kb;GroundFdVariable"/>
      <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasNameVariables"
       rdfs:label="hasNameVariables">
      <rdfs:domain rdf:resource="&kb;Operator"/>
      <rdfs:range rdf:resource="&kb;Variable"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasObjectOffset"
       rdfs:label="hasObjectOffset">
      <rdfs:domain rdf:resource="&kb;ObjectConstaint"/>
      <rdfs:range rdf:resource="&kb;Objects"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasObjectRange"
       rdfs:label="hasObjectRange">
      <rdfs:domain rdf:resource="&kb;ObjectVariable"/>
      <rdfs:range rdf:resource="&rdfs;Class"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasObjectValue"
       rdfs:label="hasObjectValue">
      <rdfs:domain rdf:resource="&kb;GroundObjectVariable"/>
      <rdfs:range rdf:resource="&kb;Objects"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasOperatorType"
       rdfs:label="hasOperatorType">
      <rdfs:domain rdf:resource="&kb;GroundedOperator"/>
      <rdfs:range rdf:resource="&kb;Operator"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasOperators"
       rdfs:label="hasOperators">
      <rdfs:range rdf:resource="&kb;Operator"/>
      <rdfs:domain rdf:resource="&kb;Solution"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasParamCardinality"
       rdfs:label="hasParamCardinality">
      <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasParamDomain"
       rdfs:label="hasParamDomain">
      <rdfs:range rdf:resource="&rdfs;Class"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasPredicate"
       rdfs:label="hasPredicate">
      <rdfs:range rdf:resource="&kb;Predicate"/>
      <rdfs:domain rdf:resource="&kb;RigidRelation"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasRelation"
       rdfs:label="hasRelation">
      <rdfs:domain rdf:resource="&kb;Chronicle"/>
      <rdfs:range rdf:resource="&kb;Relation"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasSvDomain"
       rdfs:label="hasSvDomain">
      <rdfs:range rdf:resource="&kb;Objects"/>
      <rdfs:domain rdf:resource="&kb;StateVariable"/>
</rdf:Property>
```

```
<rdf:Property rdf:about="&kb;hasSvRange"
       rdfs:label="hasSvRange">
      <rdfs:range rdf:resource="&kb;Objects"/>
      <rdfs:domain rdf:resource="&kb;StateVariable"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasSvSymbol"
       rdfs:label="hasSvSymbol">
      <rdfs:domain rdf:resource="&kb;StateVariable"/>
      <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasSvType"
       rdfs:label="hasSvType">
      <rdfs:domain rdf:resource="&kb;Assertion"/>
      <rdfs:range rdf:resource="&kb;StateVariable"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasSvVariable"
       rdfs:label="hasSvVariable">
      <rdfs:domain rdf:resource="&kb;Assertion"/>
      <rdfs:range rdf:resource="&kb;ObjectVariable"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasT"
       rdfs:label="hasT">
      <rdfs:domain rdf:resource="&kb;Event"/>
      <rdfs:range rdf:resource="&kb;FdVariable"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasT1"
       rdfs:label="hasT1">
      <rdfs:range rdf:resource="&kb;FdVariable"/>
      <rdfs:domain rdf:resource="&kb;Persistence"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasT2"
       rdfs:label="hasT2">
      <rdfs:range rdf:resource="&kb;FdVariable"/>
      <rdfs:domain rdf:resource="&kb;Persistence"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasV"
       rdfs:label="hasV">
      <rdfs:range rdf:resource="&kb;ObjectVariable"/>
      <rdfs:domain rdf:resource="&kb;Persistence"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasV1"
       rdfs:label="hasV1">
      <rdfs:domain rdf:resource="&kb;Event"/>
      <rdfs:range rdf:resource="&kb;ObjectVariable"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasV2"
       rdfs:label="hasV2">
      <rdfs:domain rdf:resource="&kb;Event"/>
      <rdfs:range rdf:resource="&kb;ObjectVariable"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasVarA"
       rdfs:label="hasVarA">
      <rdfs:domain rdf:resource="&kb;Constraint"/>
      <rdfs:range rdf:resource="&kb;Variable"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasVarB"
       rdfs:label="hasVarB">
      <rdfs:domain rdf:resource="&kb;Constraint"/>
      <rdfs:range rdf:resource="&kb;Variable"/>
```

```
</rdf:Property>

<rdf:Property rdf:about="&kb;hasVarC"
       rdfs:label="hasVarC">
      <rdfs:domain rdf:resource="&kb;FdConstraint_t"/>
      <rdfs:range rdf:resource="&kb;FdVariable"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;hasVariable"
       rdfs:label="hasVariable">
      <rdfs:domain rdf:resource="&kb;Chronicle"/>
      <rdfs:range rdf:resource="&kb;Variable"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;operator"
       rdfs:label="operator">
      <rdfs:domain rdf:resource="&kb;Constraint"/>
      <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;paramDur"
       rdfs:label="paramDur">
      <rdfs:domain rdf:resource="&kb;Predicate"/>
      <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;paramX"
       rdfs:label="paramX">
      <rdfs:domain rdf:resource="&kb;Predicate"/>
      <rdfs:range rdf:resource="&rdfs;Class"/>
</rdf:Property>
<rdf:Property rdf:about="&kb;paramY"
       rdfs:label="paramY">
      <rdfs:domain rdf:resource="&kb;Predicate"/>
      <rdfs:range rdf:resource="&rdfs;Class"/>
</rdf:Property>
<rdf:Property rdf:about="&a;_name"
       rdfs:label=":NAME">
      <rdfs:domain rdf:resource="&kb;Assertion"/>
      <rdfs:domain rdf:resource="&kb;Chronicle"/>
      <rdfs:domain rdf:resource="&kb;Objects"/>
      <rdfs:domain rdf:resource="&kb;Predicate"/>
      <rdfs:domain rdf:resource="&kb;Relation"/>
      <rdfs:domain rdf:resource="&kb;StateVariable"/>
      <rdfs:domain rdf:resource="&kb;Variable"/>
</rdf:Property>
</rdf:RDF>
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] A. Blum and M. Furst, *Fast planning through planning graph analysis*, in proceedings of International Joint Conference on Artificial Inteligence (IJCAI) (1995), pp. 1636-1642.

[2] B. Bonet and H. Geffner, *Planning as a heuristic search*, Artificial Intelligence, special issue on heuristic search (2001).

[3] H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. V. Pynadath, T. A. Russ and M. Tambe, *Electric Elves: Agent Technology for Supporting Human Organizations*, AI Magazine (2002), pp. 11-24.

[4] L. Dean and D. McDermott, *Temporal Data Base Management*, Artificial Intelligence, 32 (1987), pp. 51-55.

[5] T. Dean and M. Wellman, *Planning and Control*, Morgan Kaufman, 1991.

[6] R. Dechter and I. Meiri, *Temporal Constraints Networks*, Artificial Intelligence, 49 (1991), pp. 61-95.

[7] B. Drabble and N. Haq, *Dynamic Schedule Management: Lessons from the Air Campaign Domain*, 1999.

[8] B. Drabble, T. Lydiard and A. Tate, *Workflow support in the air campaign planning process*, in proceedings of AIPS Workshop on Interactive and Collaborative Planning (1998).

[9] M. Ghallab and H. Laruelle, *Representation and control in IxTeT, a temporal planner*, in proceedings of International Conference on AI Planning Systems (AIPS) (1994), pp. 61-67.

[10] M. Ghallab, D. Nau and P. Traverso, *Automated planning, Theory and practice*, Morgan Kaufman, 2004.

[11] J. Hoffman and B. Nebel, *The FF planning system: Fast plan generation through heuristic search*, Journal of Artificial Intelligence Research, 14 (2001), pp. 255-302.

[12] K. Mackworth, *Consistency in networks of relations*, Artificial Intelligence, 8 (1977), pp. 118-121.

[13] D. McDermott, *PDDL, the Planning Domain Definition Language*, Yale Centre for Computational Vision and Control, 1998.

[14] D. McDermott, *Using Regression-Match Graphs to Control Search in Planning*, AI Magazine (1999).

[15] S. Minton and M. Johnston, *Solving Large-Scale Constraint Satisfaction Problems and Scheduling Problems Using a Heuristic Repair Method*, in proceedings of AAAI (1990), pp. 17-24.

[16] U. Montanari, *Networks of Constraints: Fundamental properties and applications to picture processing*, Information Science, 7 (1974), pp. 95-132.

[17] X. Nguyen and S. Kambhampati, *Extracting Effective and Admissible State Space Heuristics from the Planning Graph* in proceedings of AAAI (2000).

[18] R. Nigenda, X. Nguyen and S. Kambhampati, *AltALt: Combining the Advantages of Graphplan and Heuristic State Search* AI Magazine, 22 (2001), pp. 88-90.

[19] M. Pollack, C. McCarthy, S. Ramakrishnan, I. Tsamardinos, L. Brown, S. Carrion, D. Colbry, C. Orosz and B. Peintner, *Autominder: A Planning, Monitoring, and Reminding Assistive Agent*, *7th International Conference on Intelligent Autonomous Systems*, 2002.

[20] P. Proser, *Forward Checking with Backmarking*, University of Strathclyde, 1993, PhD Thesis.

[21] I. Refanidis, *Stratified Heuristic POCL Temporal Planning based on Planning Graphs and Constraint Programming*, (2005).

[22] I. Refanidis and I. Vlahavas, *The GRT Planning System: Backward Heuristic Construction in Forward State-Space Planning*, Journal of Artificial Intelligence Research, 15 (2001), pp. 115-161.

[23] E. Sacerdoti, *The Nonlinear Nature of Plans*, in proceedings of IJCAI (1975), pp. 206-214.

[24] T. Schiex and G. Verfaillie, *Nogood recording for Static and Dynamic CSPs*, International Journal of Artificial Intelligence Tools, 3 (1994), pp. 187-200.

[25] E. Tsang, *Foundation of Constraint Satisfaction*, Academic Press, 1993.

[26] Q. Yang, *Intelligent Planning: A Decomposition and Abstraction Based Approach*, Springer, New York, 1997.

[27] J. Youll, J. Morris, R. Krikorian and P. Maes, *Impulse: Location-based Agent Assistance, Software Demos*, in proceedings of Fourth International Conference on Autonomous Agents (Agents 2000) (2000).