ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή ασαφούς συλλογιστικής στην επίγνωση καταστάσεων (situation awareness)

Παναγιώτης Α. Πασιάς

ΕΠΙΒΛΕΠΟΝΤΕΣ:

Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

Χρήστος Αναγνωστόπουλος, Υποψήφιος Διδάκτορας ΕΚΠΑ

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2006

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή ασαφούς συλλογιστικής στην επίγνωση καταστάσεων (situation awareness)

Παναγιώτης Α. Πασιάς

Α.Μ.:1115200100143

ΕΠΙΒΛΕΠΟΝΤΕΣ:

Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

Χρήστος Αναγνωστόπουλος, Υποψήφιος Διδάκτορας ΕΚΠΑ

ΠΕΡΙΛΗΨΗ

Η πτυχιακή αυτή εργασία έχει ως αντικείμενο την ομαδοποίηση (clustering) κινητών χρηστών με βάση την κατάστασή τους (situational context) και πιο συγκεκριμένα με βάση τις μεταβάσεις που αυτοί πραγματοποιούν από κατάσταση σε κατάσταση. Το situational context προκύπτει από τη σύνθεση του activity context και του spatial context, μέσω ενός ταξινομητή καταστάσεων (situational classifier). Τα activity context και spatial context, επίσης προκύπτουν μέσω classification δεδομένων περιβάλλοντος μέσα στο οποίο δραστηριοποιείται ο χρήστης, με τη χρήση των Naïve Bayes και J48 classifier του WEKA. Για τη δημιουργία των ομάδων (clusters) χρησιμοποιούνται 2 αλγόριθμοι: Ο αλγόριθμος Fuzzy C-Means που επιτρέπει σε κάθε χρήστη να ανήκει πιθανολογικά σε περισσότερα του ενός cluster μέσω κάποιας μονάδας συγγένειας (membership function) και ο αλγόριθμος Hard K-Means σύμφωνα με τον οποίο κάθε χρήστης μπορεί να ανήκει σε ένα και μόνο cluster. Τα clusters δημιουργούνται και επαναπροσδιορίζονται δυναμικά, παρακολουθώντας τις μεταβάσεις που πραγματοποιούνται στο situational context των χρηστών (user tracing) καθώς αυτοί δραστηριοποιούνται εντός ενός καθορισμένου χώρου στη χρονική διάρκεια 60 λεπτών, ώστε να ανταποκρίνονται στις μεταβάσεις αυτές. Παράλληλα με τη διαδικασία δημιουργίας και επαναπροσδιορισμού των clusters, παρακολουθούνται και οι μεταβάσεις του situational context ενός χρήστη που δεν συμμετέχει στη διαδικασία του clustering και υπολογίζονται οι αποστάσεις του από κάθε ένα από τα τρέχοντα clusters, με σκοπό να εξαχθούν πληροφορίες για την ομοιότητα ή μη του χρήστη αυτού με τις ομάδες που συμμετέχουν στα αντίστοιχα clusters. Οι τύποι αποστάσεων που χρησιμοποιούνται είναι δύο: η ευκλείδεια απόσταση και η προτεινόμενη απόσταση min-max.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Διάχυτος υπολογισμός, επίγνωση πλαισίου και εφαρμογή ασαφούς συλλογιστικής στο συμπερασμός καταστάσεων.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: clustering, επίγνωση καταστάσεων, πληροφορία πλαισίου κατάστασης, fuzzy C-Means, hard K-Means, παρακολούθηση χρηστών.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κύριο Ευστάθιο Χατζηευθυμιάδη για την υπόδειξη του θέματος της πτυχιακής εργασίας καθώς και για τη βοήθεια που μου προσέφερε κατά την περίοδο εκπόνησης της εργασίας. Τέλος, θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτορα του ΕΚΠΑ κύριο Χρήστο Αναγνωστόπουλο για την πολύτιμη συνεισφορά και στήριξη καθ' όλη τη διάρκεια της ενασχόλησης μου με το αντικείμενο της εργασίας.

Πίνακας Περιεχομένων

ΚΕΦΑΛΑΙΟ 1

ΟΡΙΣΜΟΣ ΠΛΗΡΟΦΟΡΙΑΣ ΠΛΑΙΣΙΟΥ (DEFINITION OF CONTEXT)

Στην επιστήμη των Υπολογιστών και ιδιαίτερα στον τομέα του Διάχυτου Υπολογισμού (pervasive computing), ως context (πληροφορία πλαισίου) ορίζεται οποιαδήποτε πληροφορία μπορεί να περιγράψει την κατάσταση μιας οντότητας. Οντότητα μπορεί να αποτελεί μια συσκευή, μια εφαρμογή, ή και ένας άνθρωπος. Τα συστήματα που διαθέτουν επίγνωση πληροφορίας πλαισίου (context awareness) μπορούν να αλλάζουν τη λειτουργία, ή τη συμπεριφορά τους στον χρόνο εκτέλεσης για να ανταποκρίνονται στις αλλαγές του context (context adaptation).

1.1. Ορισμός δεδομένων context (context-data definition)

Στην παρούσα εργασία το context αποτελείται από δεδομένα κίνησης του χρήστη, ήχου περιβάλλοντος, φωτός περιβάλλοντος, τοποθεσίας του χρήστη, τοποθεσίας της συσκευής (PDA), χρήσης της συσκευής, εφαρμογών που εκτελούνται στη συσκευή και προσώπων που βρίσκονται στον ίδιο χώρο με το χρήστη. Τα δεδομένα κίνησης, φωτός και ήχου συλλέχθηκαν μέσω αισθητήρων, τα δεδομένα τοποθεσίας του χρήστη και της συσκευής μέσω WiFi, ενώ τα δεδομένα χρήσης της συσκευής και εφαρμογών που εκτελούνται σε αυτή μέσω της ίδιας της συσκευής. Κάποια από τα δεδομένα αυτά παίρνουν nominal τιμές και κάποια αποτελούν ασαφείς μεταβλητές (Fuzzy Variables – FV) και παίρνουν ως τιμές ασαφή σύνολα (Fuzzy Sets - FS) μέσω μονάδων συγγένειας (membership functions – *M*). Τα δεδομένα του context ορίζονται αναλυτικότερα παρακάτω:

- FV Κίνησης (movement) : FS= {walking : $M_W \in$ [0,1]

  walking_fast : $M_{WF} \in$ [0.1]

  running : $M_R \in$ [0.1]

  halt : $M_H \in$ [0.1] }
- FV ήχου (sound) : FS= {silent : $M_S \in$ [0,1]

  modest : $M_M \in$ [0,1]

loud : $M_L \in$ [0,1] }

- FV Φωτός (light) : FS = {natural : $M_{Na} \in$ [0,1]

  bright : $M_B \in$ [0,1]

  normal : $M_N \in$ [0,1]

  dark : $M_D \in$ [0,1]

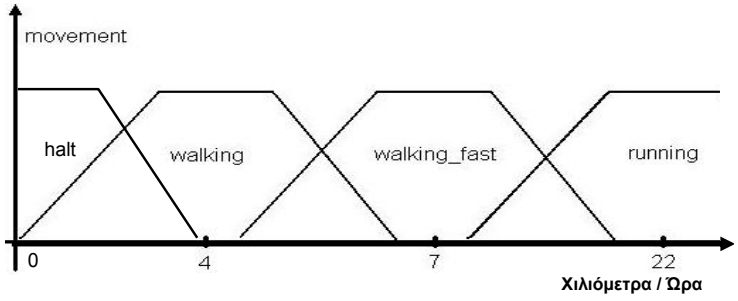
  total_darkness : $M_{TD} \in$ [0,1] }

- FV Χρήσης PDA (at_hand) :FS = {yes : $M_Y \in$ [0,1]

  no : $M_N \in$ [0,1] }

- Εφαρμογών (applications) : {web_browsing, agenda, work_app, .ppt, none} (nominal)
- Τοποθεσίας χρήστη (user_Location) : {office, corridor1, meeting_room, corridor2, coffee_room} (nominal)
- Τοποθεσίας συσκευής (pda_Location) : {office, corridor1, meeting_room, corridor2, coffee_room} (nominal)
- Προσώπων στον ίδιο χώρο (people_nearby) : {supervisor, friend, nobody} (nominal)

Κάθε μία από τις ασαφείς μεταβλητές που ορίστηκαν παραπάνω, παίρνει ως τιμή ένα ασαφές σύνολο. Για κάθε μία τιμή, ορίζεται μια μονάδα συγγένειας (membership function) δείχνοντας κατά πόσο η τιμή της μεταβλητής αντιπροσωπεύει τη μεταβλητή του context. Ακολουθεί ως παράδειγμα η διαγραμματική αναπαράσταση της ασαφούς μεταβλητής movement και των αντιστοίχων ασαφών συνόλων walking, walking_fast και running:



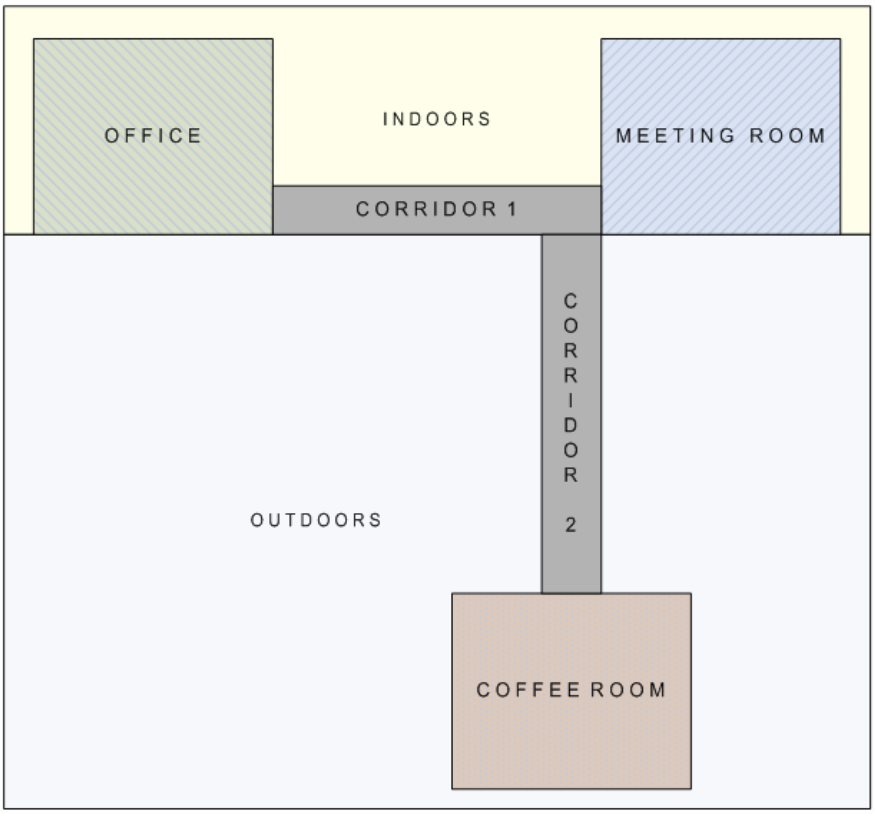
Σχήμα 1.1: Διαγραμματική αναπαράσταση της ασαφούς μεταβλητής movement.

1.2. Πλαίσιο Δραστηριότητας – Χώρου (Activity context – Spatial context)

Το context μπορεί να διαιρεθεί σε activity context και spatial context. Στο activity context ανήκουν οι συνθήκες περιβάλλοντος από τις οποίες μπορεί να εξαχθεί πληροφορία σε σχέση με τις δραστηριότητες του χρήστη, είτε αυτές αφορούν στον ίδιο (κίνηση, ομιλία κτλ.), είτε στη συσκευή (χρήση συσκευής, εφαρμογές που τρέχουν) είτε ακόμη και στο ίδιο το περιβάλλον του χρήστη (ένταση φωτός). Σε επίπεδο δεδομένων, στο activity context εντάσσονται οι τύποι δεδομένων light, sound, movement, at_hand και applications.

Στο spatial context ανήκουν οι συνθήκες περιβάλλοντος που αφορούν στη χωροταξική τοποθέτηση του χρήστη και της συσκευής στο περιβάλλον, καθώς και στα πρόσωπα που βρίσκονται στον ίδιο χώρο με το χρήστη. Σε επίπεδο δεδομένων, στο spatial context εντάσσονται τα: user_Location, pda_Location, και people_nearby.

Το περιβάλλον στο οποίο δραστηριοποιούνται οι χρήστες και παρακολουθείται η δράση τους απεικονίζεται σχηματικά παρακάτω:



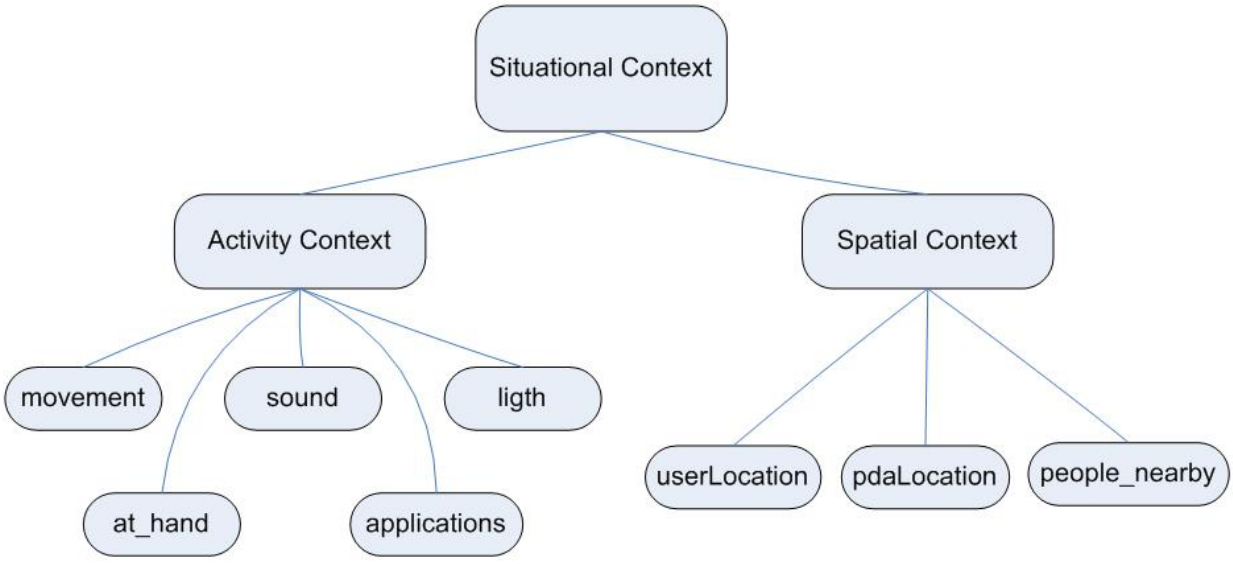
Σχήμα 1.2: Το περιβάλλον στο οποίο δραστηριοποιούνται οι χρήστες.

1.3. Εισαγωγή του Situational context ως Context model.

Η επίγνωση πλαισίου (context awareness) αφορά στη δυνατότητα αντίληψης του context και στην κατανόησή της έννοιάς του. Οι εφαρμογές που διαθέτουν situation awareness είναι μια νέα κατηγορία context-aware εφαρμογών που εκμεταλλευόμενες την αντίληψη του τρέχοντος situational context, έχουν τη δυνατότητα να παρέχουν στο χρήστη τις απαιτούμενες πληροφορίες για τη λήψη αποφάσεων με διάχυτο (pervasive) τρόπο. Επιπλέον, επειδή έχουν τη δυνατότητα προσαρμογής στο τρέχον situational context και πιθανόν στο εγγύς μελλοντικό, λειτουργούν με τρόπο εξαρτώμενο από την κατάσταση του χρήστη, Στην παρούσα εργασία το user context μοντελοποιείται ως situational context, δηλαδή κάθε χρονική στιγμή, κάθε χρήστης χαρακτηρίζεται από την κατάσταση (ή τις καταστάσεις) στις οποίες μπορεί να βρίσκεται. Το situational context προκύπτει από τη σύνθεση του activity context και του spatial context και αποτελεί μια προτεινόμενη προσέγγιση μοντελοποίησης του user context με σκοπό το συμπερασμό γνώσης για τις καταστάσεις στις οποίες βρίσκονται οι χρήστες.

Ορισμός situational context:

$$situational\_context = \{activity\_context \oplus spatial\_context\}$$



Σχήμα 1.3: Το situation context ως σύνθεση του activity και του spatial context.

ΚΕΦΑΛΑΙΟ 2

ΟΜΑΔΟΠΟΙΗΣΗ ΠΛΑΙΣΙΟΥ (CONTEXT CLASSIFICATION)

2.1. Αναπαράσταση δεδομένων του context με ασαφή συλλογιστική (fuzzy logic)

Μια λογική βασισμένη στις τιμές true και false μπορεί να αποδειχθεί ανεπαρκής για την περιγραφή του context. Fuzzy logic είναι η συλλογιστική που χρησιμοποιεί ολόκληρο το διάστημα μεταξύ του 0 (false) και του 1 (true) για να περιγράψει το context, παρέχοντας έτσι μεγαλύτερες δυνατότητες αναπαράστασης και χρησιμοποιείται συνήθως σε τομείς όπως η αναπαράσταση ανθρώπινου συλλογισμού (human reasoning) και η τεχνολογία αυτόματων ελεγκτών (fuzzy controllers). Οι παράμετροι movement, light, sound και at_hand του activity context αναπαρίστανται με fuzzy sets (ασαφή σύνολα). Έτσι, μια ασαφής τιμή movement δεν είναι απαραίτητα είτε walking, είτε walking_fast, είτε running, είτε halt, αφού μπορεί ο χρήστης να κινείται με ένα ρυθμό που δεν καθιστά εύκολη τη διάκριση ανάμεσα σε waking και walking_fast. Μπορεί να είναι κατά ένα ποσοστό walking, κατά ένα άλλο walking fast, κατά ένα τρίτο running και κατά ένα τέταρτο halt, με τον περιορισμό ότι το συνολικό άθροισμα των ποσοστών αυτών να είναι ίσο με τη μονάδα. Για παράδειγμα μια τιμή movement είναι η ακόλουθη :

movement={ 0.5/walking + 0.4/ walking_fast + 0.1/ running + 0.0/halt },

όπου το σύμβολο ‘+’ αναπαριστά την πράξη της ασαφούς πρόσθεσης.

Το ίδιο ισχύει και για την παράμετρο light, που αποτελεί συνδυασμό ποσοστών bright, normal, dark, natural και total_darkness, για την παράμετρο sound που αποτελεί συνδυασμό ποσοστών silent, modest και loud, καθώς και για την παράμετρο at_hand που αποτελούν συνδυασμό ποσοστών yes (ο χρήστης κρατάει το PDA) και no (ο χρήστης δεν κρατάει το PDA).

2.2. Πιθανολογική Ταξινόμηση (Naïve Bayes) και Επαγωγική Εκμάθηση (J48)

Για την ταξινόμηση των δεδομένων movement, sound και light, χρησιμοποιήθηκε ο Naïve Bayes Classifier του εργαλείου WEKA. Ο Naïve Bayes Classifier είναι ένας απλός πιθανολογικός ταξινομητής που βασίζεται στην εφαρμογή του θεωρήματος Bayes, με προϋποθέσεις ισχυρής ανεξαρτησίας. Παρά την απλή του λογική και τις υπεραπλουστευμένες υποθέσεις ανεξαρτησίας του, ο Naïve Bayes classifier έχει αποδειχθεί ότι πολύ συχνά παράγει πολύ πιο έγκυρα αποτελέσματα από τα αναμενόμενα σε συνθέτες πραγματικές καταστάσεις.

Μέθοδος ταξινόμησης του Naïve Bayes Classifier.

Δεδομένου ενός συνόλου πιθανών κλάσεων C = $\{C_1,C_2,\ldots,C_M\}$ και ενός τυχαίου datapoint X=$\{x_1,x_2,\ldots,x_N\}$ το datapoint X ανήκει στην κλάση που μεγιστοποιεί την πιθανότητα:

$$P\left(C_j \mid X\right) \propto P\left(C_j\right)\sum_{k=1}^{N} P\left(x_k \mid C_j\right)$$

όπου $P\left(C_j\right)$ η a-priori πιθανότητα της κλάσης C_j , δηλαδή

$$C_X = \arg\max_{C_j \in C}\left\{P\left(C_j \mid X\right)\right\}$$

Ο J48 Ταξινομητής

Αν και για τους τύπους δεδομένων movement, light και sound του activity context, για τον καθένα από τους οποίους η ταξινόμηση έγινε σε περισσότερες των δύο κλάσεων, ο Naïve Bayes classifier έδινε ιδιαίτερα ικανοποιητικά αποτελέσματα, για τον τύπο δεδομένων at_hand, για τον οποίο η ταξινόμηση των δεδομένων έγινε σε 2 κλάσεις, τα αποτελέσματα ήταν απογοητευτικά (υψηλό error rate). Για το λόγο αυτό προτιμήθηκε ο J48 classifier. Ο J48 classifier αποτελεί μια υλοποίηση του WEKA για τον αλγόριθμο ταξινόμησης C4.5, ο οποίος με τη σειρά του βασίζεται στον αλγόριθμο ID3, και ταξινομεί τα δεδομένα δημιουργώντας δέντρα απόφασης (decision trees). Τα βασικά χαρακτηριστικά του J48 που τον καθιστούν ιδιαίτερα ελκυστικό στις εφαρμογές λογισμικού είναι:

- Δυνατότητα επιλογής κατάλληλου μέτρου για attribute selection
- Δυνατότητα διαχείρισης training data με ελλιπείς τιμές στα attributes.
- Δυνατότητα διαχείρισης attributes με διαφορετικά κόστη.
- Δυνατότητα διαχείρισης attributes που παίρνουν συνεχείς τιμές.

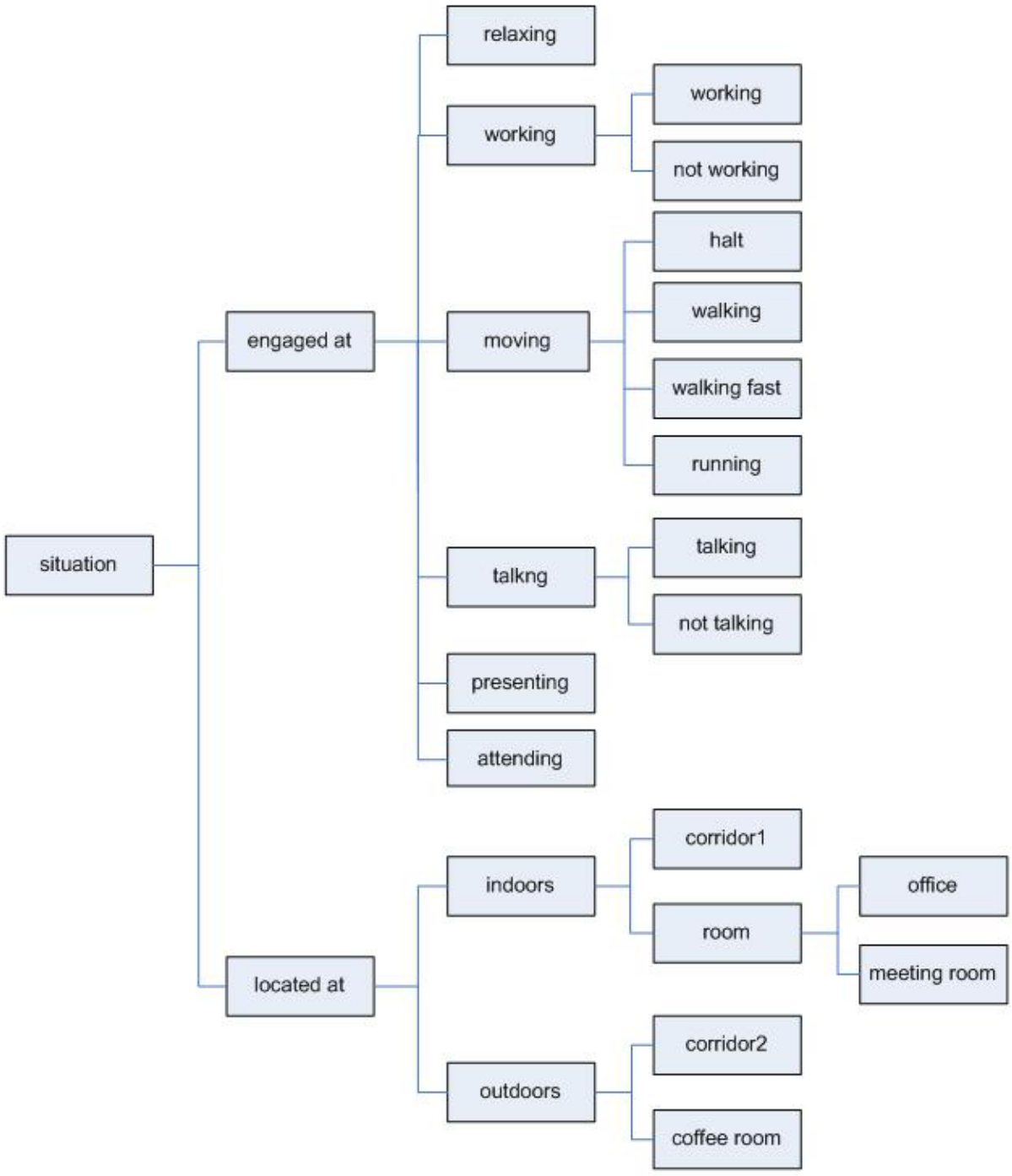
Οι επιδόσεις του J48 στην ταξινόμηση δεδομένων σε 2 κλάσεις ήταν πολύ ικανοποιητικές (χαμηλό error rate). Οι πιθανές κλάσεις για τα δεδομένα του activity context δίνονται στον παρακάτω πίνακα:

ΤΥΠΟΣ ΔΕΔΟΜΕΝΩΝ	ΠΙΘΑΝΕΣ ΚΛΑΣΕΙΣ
movement	walking
	walking_fast
	running
	halt
sound	slilent
	modest
	loud
light	bright
	normal
	dark
	natural
	total_darkness
at_hand	yes
	no

Πίνακας 2.1: Πιθανές κλάσεις των τύπων δεδομένων του activity context.

2.3. Το Situation context ως σύνθεση του Activity context και του Spatial context.

Όπως αναφέρθηκε στην ενότητα 2.3. το situation context προκύπτει από τη σύνθεση του activity context και του spatial context. Με την ταξινόμηση των δεδομένων του activity context και την σύνθεση τους με τις πληροφορίες του spatial context, η μοντελοποίηση του situation context είναι η ακόλουθη:



Σχήμα 2.1: Η ιεραρχία του situation context όπως προκύπτει από σύνθεση του activity context και του spatial context.

ΚΕΦΑΛΑΙΟ 3

ΣΥΜΠΕΡΑΣΜΟΣ ΠΛΑΙΣΙΟΥ (CONTEXT REASONING)

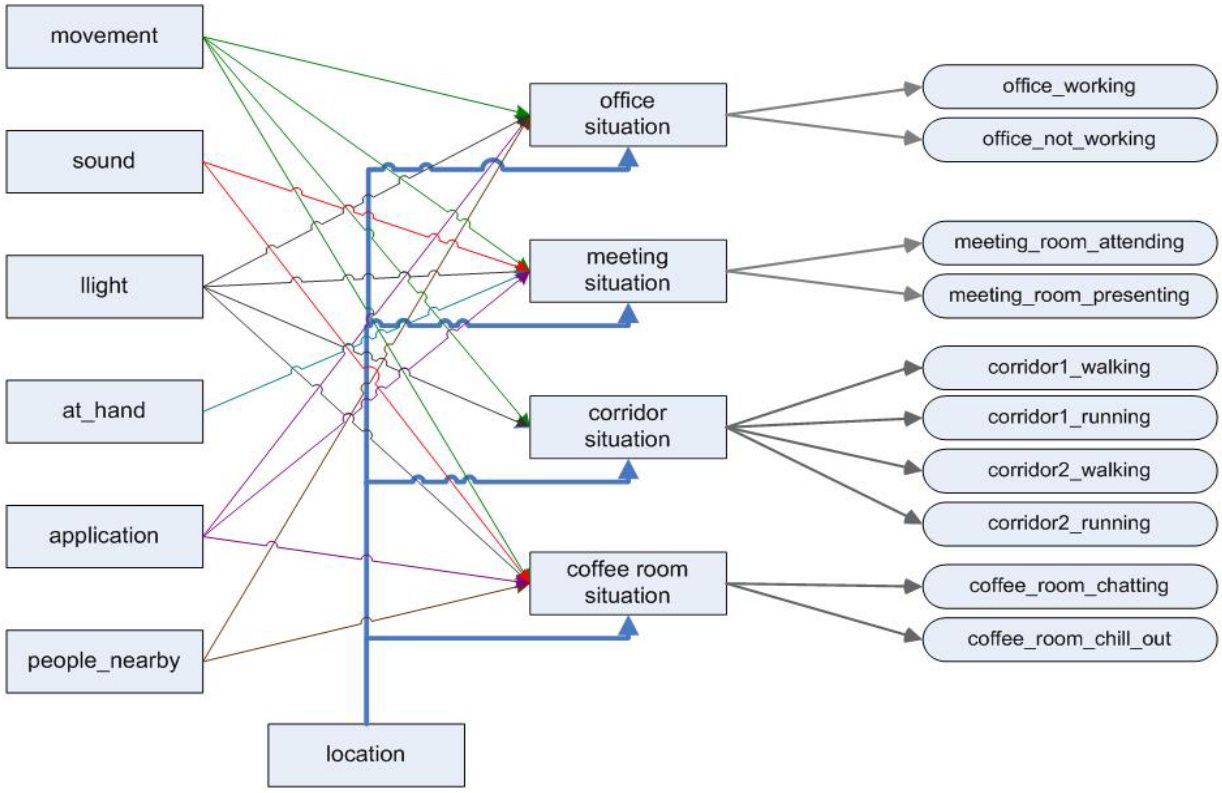
Το situational context όπως μοντελοποιήθηκε παραπάνω, περιέχει πληθώρα πληροφοριών που αφορούν στη χωροταξική τοποθέτηση και τις δραστηριότητες των χρηστών. Οι πληροφορίες αυτές όμως, δεν είναι οργανωμένες και δεν μπορούν να παρέχουν επαρκή γνώση για την κατάσταση των χρηστών στη μορφή που είναι. Οι γνώση θα πρέπει με κάποιο τρόπο να εξαχθεί από τις πληροφορίες. Το situational reasoning αφορά σε ακριβώς αυτή την εξαγωγή γνώσης περί της κατάστασης των χρηστών μέσα από το context.

3.1. Situational Reasoning μέσω ταξινόμησης.

Προκειμένου να εξάγουμε την τρέχουσα κατάσταση (situation), δημιουργήσαμε ένα μηχανισμό ταξινόμησης των χρηστών σε κατηγορίες με βάση τις πληροφορίες του current situational context, έναν situational classifier. Ο situational classifier, παίρνει ως είσοδο τα τρέχοντα classified δεδομένα του activity context καθώς και τις πληροφορίες του spatial context ενός user και ταξινομεί τον user αυτό σε μία από τις δέκα παρακάτω κατηγορίες:

- S1 : OFFICE_WORKING
- S2 : OFFICE_NOT_WORKING
- S3 : CORRIDOR1_WALKING
- S4 : CORRIDOR1_RUNNING
- S5 : MEETING_ROOM_ATTENDING
- S6 : MEETING_ROOM_PRESENTING
- S7 : CORRIDOR2_WALKING
- S8 : CORRIDOR2_RUNNING
- S9 : COFFEE_ROOM_CHATTING
- S10 : COFFEE_ROOM_CHILL_OUT

Η μέθοδος ταξινόμησης που ακολουθεί ο situational classifier είναι η ακόλουθη: Οι δέκα καταστάσεις διαχωρίζονται σε 4 κατηγορίες καταστάσεων (office (S_O), moving (S_M), meeting (S_{MT}), coffee break (S_{CB})) με βάση το είδος του χώρου στον οποίο γίνεται το tracing των χρηστών. Όπως γίνεται εύκολα αντιληπτό οι κατηγορίες office, meeting και coffee break περιλαμβάνουν 2 καταστάσεις η καθεμία, ενώ η κατηγορία moving περιλαμβάνει 4 καταστάσεις, δύο για κάθε ένα από τους corridor1 και corridor2. Ξεκινώντας με τη θέση (location) στην οποία βρίσκονται ο user και το PDA γίνεται κατάταξη του χρήστη σε μία από τις 4 κατηγορίες καταστάσεων. Προκειμένου να αποφανθεί σε ποια από τις δυνατές καταστάσεις της κατηγορίας βρίσκεται ο χρήστης, ο situational classifier εξετάζει τις κλάσεις στις οποίες έχουν ταξινομηθεί κάποια από τα δεδομένα του current activity context, καθώς την τιμή του people_nearby του spatial context. Δεν είναι απαραίτητο για όλες της κατηγορίες να εξεταστούν οι κλάσεις όλων των δεδομένων του activity context και το people_nearby. Για παράδειγμα, εάν το location του user και του PDA είναι corridor1, η κλάση του sound δεν παίζει ρόλο στον προσδιορισμό της κατάστασης, αφού η κίνηση του user, για την οποία και θέλει να αποφανθεί ο classifier, δεν επηρεάζεται από της ένταση του ήχου στο διάδρομο. Παρακάτω δίνεται μια σχηματική αναπαράσταση της μεθόδου ταξινόμησης του situation classifier:



Σχήμα 3.1: Σχηματική αναπαράσταση του situational classifier. Επίδραση του spatial context στον συμπερασμό καταστάσεων

3.2. Κανόνες του Situational Reasoning.

Οι κανόνες που ακολουθεί ο situational classifier προκειμένου να ταξινομήσει το current context σε μία από τις 10 δυνατές καταστάσεις, είναι οι ακόλουθοι:

- **If** location is office ∧ light is normal ∧ movement is halt ∧ people_nearby is nobody ∧ application is (work_app | agenda)

  then situation is office_working.

- **If** location is office ∧ light is normal ∧ movement is halt ∧ people_nearby is nobody ∧ application is none **then** situation is office_not_working.

- **If** location is meeting_room ∧ light is natural ∧ movement is halt ∧ sound is (modest | silent) ∧ at_hand is no ∧ application is .ppt

  then situation is meeting_room presenting.

- **If** location is meeting_room ∧ light is natural ∧ movement is halt ∧ sound is (modest | silent) ∧ at_hand is yes ∧ application is downloading_presentation

  then situation is meeting_room attending.

- **If** location is coffee_room ∧ light is natural ∧ movement is halt ∧ sound is modest ∧ people_nearby is friend ∧ application is none

  then situation is coffee_room_chatting.

- **If** location is coffee_room ∧ light is natural ∧ movement is halt ∧ sound is loud ∧ people_nearby is nobody ∧ application is (web_browsing | agenda | none)

  then situation is coffer_room_chill_out.

- **If** location is corridor1 ∧ light is dark ∧ movement is running

  then situation is corridor1_running.

- **If** location is corridor1 ∧ light is dark ∧ movement is (walking | walking_fast)

  then situation is corridor1_walking.

- **If** location is corridor2 ∧ light is dark ∧ movement is running

  then situation is corridor2_running.

- **If** location is corridor2 ∧ light is dark ∧ movement is (walking | walking_fast)

  then situation is corridor2_walking.

ΚΕΦΑΛΑΙΟ 4

ΜΕΤΑΒΑΣΗ ΚΑΤΑΣΤΑΣΕΩΝ (CONTEXT TRANSITION)

4.1. Πίνακες Μετάβασης Πλαισίου (Context Transition Matrices).

Καθώς οι χρήστες δραστηριοποιούνται στο χώρο παρακολουθούνται (user tracing) για χρονική διάρκεια 60 λεπτών και καταγράφεται το situational context τους ανά ένα δευτερόλεπτο. Το situational context μπορεί να αλλάζει ανάλογα με τις ενέργειες των χρηστών και να προκύπτουν μεταβάσεις καταστάσεων (context transitions). Οι μεταβάσεις αυτές καταγράφονται στους πίνακες μεταβάσεων κατάστασης (Situational Transition Matrices - STM). Για κάθε χρήστη που παρακολουθείται υπάρχει και ένας STM. Οι διαστάσεις κάθε πίνακα είναι 10x10, αφού 10 είναι και οι δυνατές καταστάσεις του situational context. Κάθε γραμμή και κάθε στήλη του πίνακα αντιστοιχούν σε μία από τις δυνατές καταστάσεις, ενώ κάθε στοιχείο του πίνακα σε μία από τις δυνατές 100 μεταβάσεις κατάστασης. Η παραμονή του χρήστη σε μία κατάσταση για περισσότερο από 1 δευτερόλεπτο θεωρείται επίσης μετάβαση κατάστασης και καταγράφεται. Τα στοιχεία της κυρίας διαγώνιου του πίνακα (στοιχεία (1,1),(2,2)…(10,10)) αντιστοιχούν σε αυτές ακριβώς τις “μεταβάσεις” από μία κατάσταση στην ίδια κατάσταση. Η αντιστοίχηση των καταστάσεων στις γραμμές και τις στήλες του STM είναι η ακόλουθη:

- S1 : OFFICE_WORKING → γραμμή 1, στήλη 1
- S2 : OFFICE_NOT_WORKING → γραμμή 2, στήλη 2
- S3 : CORRIDOR1_WALKING → γραμμή 3, στήλη 3
- S4 : CORRIDOR1_RUNNING → γραμμή 4, στήλη 4
- S5 : MEETING_ROOM_ATTENDING → γραμμή 5, στήλη 5
- S6 : MEETING_ROOM_PRESENTING → γραμμή 6, στήλη 6
- S7 : CORRIDOR2_WALKING → γραμμή 7, στήλη 7
- S8 : CORRIDOR2_RUNNING → γραμμή 8, στήλη 8
- S9 : COFFEE_ROOM_CHATTING → γραμμή 9, στήλη 9
- S10 : COFFEE_ROOM_CHILL_OUT → γραμμή 10, στήλη 10

Στην αρχή του tracing όλα τα στοιχεία του situation transition matrix έχουν την τιμή 0. Για κάθε μετάβαση που πραγματοποιείται, αυξάνεται κατά ένα η τιμή του στοιχείου που αντιστοιχεί στη μετάβαση αυτή. Για παράδειγμα για τη μετάβαση από την κατάσταση MEETING_ROOM_ATTENDING στην κατάσταση CORRIDOR2_WALKING, αυξάνεται η τιμή του στοιχείου (5,7). Ένα στιγμιότυπο του situation transition matrix ενός user δίνεται παρακάτω:

		S I T U A T I O N S									
		S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**
S I T U A T I O N S	**S1**	15	1	1	0	0	0	0	0	0	0
	S2	0	5	1	0	0	0	0	0	0	0
	S3	0	1	4	0	0	1	1	0	0	0
	S4	1	0	0	5	0	0	0	0	0	0
	S5	0	0	0	1	25	0	0	0	0	0
	S6	0	0	0	0	0	7	0	1	0	0
	S7	0	0		0	1	0	6	0	1	0
	S8	0	0	0	0	0	0	0	3	0	1
	S9	0	0	0	0	0	0	1	0	40	0
	S10	0	0	0	0	0	0	1	0	0	10

Πίνακας 4.1: Στιγμιότυπο του STM ενός user.

Έχοντας προηγουμένως διαχωρίσει τις 10 δυνατές καταστάσεις σε 4 κατηγορίες καταστάσεων (S_O, S_{MT}, S_{CB}, S_M) μπορούμε να ορίσουμε μια διαφορετική, πιο συμπαγή μορφή του STM, τον Abstract Situation Transition Matrix (ASTM), που να απεικονίζει τις μεταβάσεις μεταξύ των κατηγοριών καταστάσεων. Για το στιγμιότυπο του STM που δόθηκε παραπάνω, ακολουθεί το αντίστοιχο στιγμιότυπο του ASTM:

	S_O	S_{MT}	S_M	S_{CB}
S_O	20	0	2	0
S_{MT}	0	32	1	0
S_M	2	2	16	2
S_{CB}	0	0	2	50

Πίνακας 4.2: Αντίστοιχο στιγμιότυπο ASTM του STM που απεικονίζεται στον πίνακα 4.1

4.2. Situation Transition Tracing.

Στόχος του situation transition tracing είναι η εξαγωγή γνώσης γύρω από τη συμπεριφορά και τις συνήθειες των χρηστών. Η συμπεριφορά και οι συνήθειες των χρηστών γίνονται γνωστές μέσω των πινάκων μεταβάσεων κατάστασής τους, αφού οι πίνακες μας δίνουν πληροφορίες τόσο για το από ποιες καταστάσεις διέρχονται οι χρήστες και με ποια σειρά, όσο και για το χρονικό διάστημα που παραμένουν στην κάθε κατάσταση. Γνωρίζοντας τη συμπεριφορά και τις συνήθειες των χρηστών, μπορούμε εν συνεχεία να τους εντάξουμε σε κατηγορίες (ομάδες), κάτι που θα κάνουμε παρακάτω με διαδικασία του context clustering.

ΚΕΦΑΛΑΙΟ 5

ΔΙΑΧΩΡΙΣΜΟΣ ΤΩΝ ΧΡΗΣΤΩΝ ΣΕ ΟΜΑΔΕΣ ΜΕ ΒΑΣΗ ΤΟ CONTEXT (CONTEXT CLUSTERING)

Κατά τη διάρκεια του user tracing, παράλληλα με τις διαδικασίες του context reasoning και context transition, διεξάγεται και η διαδικασία του διαχωρισμού των users σε ομάδες (clustering). Η διαδικασία του clustering πραγματοποιείται με βάση το situational context των χρηστών, το οποίο απεικονίζεται στους Situational Transition Matrices (STMs). Η δημιουργία των clusters πραγματοποιείται για πρώτη φορά 300 δευτερόλεπτα μετά την έναρξη του user tracing, ενώ μετά τη δημιουργία τους, τα clusters επαναπροσδιορίζονται ανά 300 δευτερόλεπτα προκειμένου να ανταποκρίνονται στις μεταβάσεις του context. Χρησιμοποιήθηκαν 2 αλγόριθμοι για τη δημιουργία και τον επαναπροσδιορισμό των clusters. Ο αλγόριθμος Fuzzy C-Means που επιτρέπει σε κάθε χρήστη να ανήκει πιθανολογικά σε περισσότερα του ενός clusters μέσω κάποιας μονάδας συγγένειας (membership function) και ο αλγόριθμος Hard K-Means σύμφωνα με τον οποίο κάθε χρήστης μπορεί να ανήκει σε ένα και μόνο cluster. Στα πλαίσια του user tracing παρακολουθούνται και οι μεταβάσεις καταστάσεων ενός επιπλέον χρήστη, που δεν συμμετέχει στη διαδικασία του clustering, προκειμένου να υπολογιστούν εν συνεχεία οι αποστάσεις του από τα clusters.

5.1. Hard Clustering – Ο αλγόριθμος hard K-Means.

Ο αλγόριθμος Hard K Means διαχωρίζει μια ένα σύνολο N διανυσμάτων σε C ομάδες (clusters G_i, i = 1,2,…,C). Ο στόχος του αλγορίθμου είναι να βρει το κεντροειδές κάθε ομάδας (cluster centroid). Ο αλγόριθμος ελαχιστοποιεί μια συνάρτηση ανομοιότητας (ή απόστασης) η οποία ορίζεται ως:

$$J = \sum_{i=1}^{c} J_i = \sum_{k, x_k \in G_i} d(x_k - c_i) \qquad (5.1)$$

όπου c_i είναι το κεντροειδές του cluster i και $d(x_k - c_i)$ η απόσταση του i-στού κεντροειδούς από το k-στο datapoint.

Για λόγους απλότητας ως μέτρο ανομοιότητας χρησιμοποιείται η Ευκλείδεια απόσταση και η συνάρτηση ανομοιότητας εκφράζεται:

$$J = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c} \left(\sum_{k, x_k \in G_i} \| x_k - c_i \|^2 \right) \qquad (5.2)$$

Ο διαχωρισμός των clusters μπορεί να εκφραστεί μέσω ενός c*n δυαδικού πίνακα συγγένειας (U), για τον οποίο το στοιχείο u_{ij} παίρνει την τιμή 1, εάν το datapoint x_j ανήκει στο cluster i και 0 σε διαφορετική περίπτωση. Η εξήγηση αυτή διατυπώνεται παρακάτω:

$$u_{ij} = \begin{cases} 1 & if \| x_j - c_i \|^2 \le \| x_j - c_k \|^2, για \ κάθε \ k \neq i, \\ 0 & διαφορετικά \end{cases} \qquad (5.3)$$

Δεδομένου ότι ένα datapoint μπορεί μόνο να ανήκει σε ένα και μόνο cluster, ο πίνακας συγγένειας (U) έχει δύο ιδιότητες που δίνονται στις εξισώσεις 5.4 και 5.5:

$$\sum_{i=1}^{c} u_{ij} = 1, \forall j = 1,..., n \qquad (5.4)$$

$$\sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij} = n \qquad (5.5)$$

Το κεντροειδές κάθε cluster υπολογίζεται ως η μέση τιμή των δεδομένων που ανήκουν στο cluster αυτό:

$$c_i = \frac{1}{| G_i |} \sum_{k, x_k \in G_i} x_k \qquad (5.6)$$

|Gi| είναι το πλήθος των δεδομένων που ανήκουν στο cluster i.

Τα βήματα του αλγορίθμου Hard K Means για ένα σύνολο δεδομένων x_j, j=1,2,…,n είναι τα ακόλουθα:

Βήμα 1. Αρχικοποίησε τα κεντροειδή c_i, i=1..c. Αυτό συνήθως επιτυγχάνεται μέσω τυχαίας επιλογής c data points , μεταξύ όλων των δεδομένων.

Βήμα 2. Υπολόγισε τον πίνακα συγγένειας U μέσω της εξίσωσης 5.3

Βήμα 3. Υπολόγισε την τιμή της συνάρτησης ανομοιότητας με τη χρήση της εξίσωσης 5.2. Εάν η βελτίωση της από την προηγούμενη επανάληψη είναι κάτω από ένα προκαθορισμένο κατώτατο όριο, σταμάτα.

Βήμα 4. Υπολόγισε τα καινούργια κεντροειδή χρησιμοποιώντας την εξίσωση 5.6. Πήγαινε στο βήμα 2.

Η απόδοση του αλγορίθμου εξαρτάται από τις αρχικές θέσεις των κεντροειδών. Έτσι, ο αλγόριθμος δεν δίνει καμία εγγύηση για μια βέλτιστη λύση.

5.2. Fuzzy Clustering – Ο αλγόριθμος fuzzy C-Means.

Ο αλγόριθμος Fuzzy C Means, γνωστός και ως Fuzzy ISODATA, αποτελεί μια τεχνική clustering διαφορετική από την τεχνική του Hard K Means που υιοθετεί την προϋπόθεση κάθε datapoint (user στην περίπτωση που εξετάζουμε) να ανήκει σε ένα cluster. Ο Fuzzy C Means υιοθετεί τον ασαφή διαχωρισμό, έτσι ώστε κάθε datapoint να μπορεί να ανήκει σε όλα τα clusters με διαφορετικούς βαθμούς συγγένειας από 0 έως 1. Ο Fuzzy C Means είναι ένας επαναληπτικός αλγόριθμος. Ο στόχος του είναι να βρει τα κεντροειδή (cluster centroids) που ελαχιστοποιούν μια συνάρτηση ανομοιότητας.
Για να προσαρμοστεί στη λογική του ασαφούς διαχωρισμού, ο πίνακας συγγένειας U αρχικοποιείται με τυχαίες τιμές, στα πλαίσια που επιτρέπει ο περιορισμός που εκφράζει η εξίσωση 5.7:

$$\sum_{i=1}^{c} u_{ij} = 1, \forall j = 1,...,n \qquad (5.7)$$

Η συνάρτηση ανομοιότητας που χρησιμοποιεί ο Fuzzy C Means, δίνεται στην εξίσωση 5.8:

$$J(U, c_1, c_2, ..., c_c) = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c} \sum_{j=1}^{n} {u_{ij}}^m {d_{ij}}^2 \qquad (5.8)$$

Όπου το $u_{ij} \in [0,1]$,

c_i είναι το κεντροειδές του cluster i.

d_{ij} είναι η Ευκλείδεια απόσταση μεταξύ του κεντροειδούς i και του datapoint j.

$m \in [1, \infty]$ είναι ένας εκθέτης στάθμισης.

Προκειμένου να ελαχιστοποιηθεί η συνάρτηση ανομοιότητας πρέπει να ικανοποιούνται οι δύο παρακάτω συνθήκες:

$$c_i = \frac{\sum_{j=1}^{n} {u_{ij}}^m x_j}{\sum_{j=1}^{n} {u_{ij}}^m} \qquad (5,9)$$

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ij}}{d_{kj}} \right)^{2/(m-1)}} \qquad (5.10)$$

Τα βήματα του αλγορίθμου Fuzzy C Means είναι τα ακόλουθα:

Βήμα 1. Αρχικοποίησε τον πίνακα συγγένειας (U) με βάση τους περιορισμό της εξίσωσης 5.7

Βήμα 2. Υπολόγισε τα κεντροειδή (ci) χρησιμοποιώντας την εξίσωση 5.9

Βήμα 3. Υπολόγισε τη συνάρτηση ανομοιότητας μεταξύ των κεντροειδών και των δεδομένων χρησιμοποιώντας την εξίσωση 5.8 Εάν η βελτίωσή της από την προηγούμενη επανάληψη είναι κάτω από ένα προκαθορισμένο κατώτατο όριο σταμάτα.

Βήμα 4. Υπολόγισε το νέο πίνακα συγγένειας U μέσω της εξίσωσης 5.10. Πήγαινε στο βήμα 2.

Με τον επαναπροσδιορισμό των cluster centroids και των βαθμών συγγένειας, ο Fuzzy C Means επαναλαμβανόμενα μετακινεί τα cluster centroids στη “σωστή θέση”.

Λόγω του γεγονότος ότι τα κεντροειδή αρχικοποιούνται χρησιμοποιώντας τον πίνακα συγγένειας U ο οποίος αρχικοποιείται τυχαία, ο Fuzzy C Means δεν εξασφαλίζει ότι συγκλίνει σε μια βέλτιστη λύση. Η απόδοση του αλγορίθμου εξαρτάται από τα αρχικά κεντροειδή. Για τη βελτίωση της απόδοσης του Fuzzy C Means υπάρχουν δύο προσεγγίσεις:

- Χρήση ενός αλγορίθμου για τον καθορισμό των κεντροειδών (π.χ. αριθμητική μεση τιμή όλων των δεδομένων)
- Εκτέλεση του Fuzzy C Means πολλές φορές με διαφορετικά κάθε φορά αρχικά κεντροειδή.

5.3. Υπολογισμός αποστάσεων τυχαίου χρήστη από τα clusters.

Ο υπολογισμός των αποστάσεων του χρήστη που δεν συμμετέχει στη διαδικασία του clustering (τυχαίος χρήστης) από τα κεντροειδή των clusters που έχουν δημιουργηθεί πραγματοποιείται για πρώτη φορά 300 δευτερόλεπτα μετά την αρχή του user tracing και επαναλαμβάνεται στη συνέχεια ανά 300 δευτερόλεπτα, ακριβώς μετά από κάθε επαναπροσδιορισμό των clusters. Ο υπολογισμός των αποστάσεων πραγματοποιείται τόσο για τα clusters που προέκυψαν μέσω του Hard K Means (hard distances), όσο και για αυτά που προέκυψαν μέσω του Fuzzy C Means (fuzzy distances). Οι τύποι αποστάσεων που υπολογίζονται είναι δύο: Η Ευκλείδεια απόσταση και η προτεινόμενη απόσταση min-max.

Η ευκλείδεια απόσταση:

Η ευκλείδεια απόσταση d μεταξύ δύο σημείων P = { $p_1,p_2,\ldots,p_n$ } και Q = { $q_1,q_2,\ldots,q_n$ } ορίζεται ως:

$$d = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Η προτεινόμενη απόσταση min-max:

Η απόσταση min-max d μεταξύ δύο σημείων P = { $p_1,p_2,\ldots,p_n$ } και Q = { $q_1,q_2,\ldots,q_n$ } ορίζεται ως:

$$d = \sqrt{(\max P_i - \max Q_i)^2 - (\min P_i - \min Q_i)^2}$$

Όπου max Xi η μέγιστη συντεταγμένη του στοιχείου X = { $x_1,x_2,\ldots,x_n$ }.

Οι αποστάσεις αυτές του τυχαίου χρήστη από τα κεντροειδή των clusters δίνονται μέσω της εφαρμογής με τη μορφή πίνακα αποστάσεων κάθε φορά που υπολογίζονται. Ακολουθεί ένα στιγμιότυπου του πίνακα hard distances, καθώς και ένα του πίνακα fuzzy distances.

Fuzzy Distances		
	Eucledean	Min-Max
Cluster 0	9.9680221867464410e-01	9.9680221866817300e-01
Cluster 1	2.2957653742159185e-04	2.2957653083953989e-04
Cluster 2	9.6873391564660990e-05	9.6873200056763420e-05
Cluster 3	2.6759828982258167e-05	2.6748332757267157e-05
Cluster 4	1.4056112405768872e-04	1.4056105413178476e-04

Εικόνα 5.1: Στιγμιότυπο του πίνακα αποστάσεων από τα fuzzy clusters.

Hard Distances		
	Eucledean	Min-Max
Cluster 0	2.5994927497344336e-04	2.5994925797841444e-04
Cluster 1	2.4722470924407535e-04	2.4722468633491440e-04
Cluster 2	2.4018131051649415e-04	2.4018128650766946e-04
Cluster 3	1.7064375118744270e-04	1.7064360545107220e-04
Cluster 4	2.7752887745904865e-04	2.7749653918837220e-04

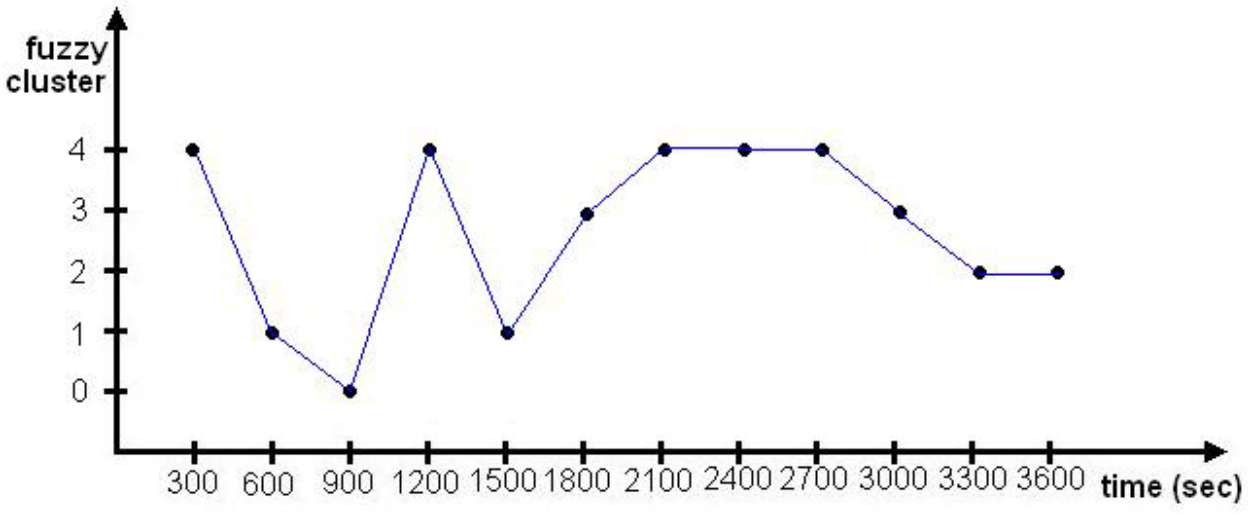
Εικόνα 5.2: Στιγμιότυπο του πίνακα αποστάσεων από τα hard clusters.

Παρατηρούμε πως τόσο η ευκλείδεια, όσο και η προτεινόμενη απόσταση min-max δίνουν περίπου τις ίδιες τιμές αποστάσεων με μικρές αποκλίσεις. Ως προς τις fuzzy distances, στο συγκεκριμένο στιγμιότυπο η μικρότερη απόσταση είναι αυτή από το fuzzy cluster 3, οπότε ο τυχαίος χρήστης έχει περισσότερες πιθανότητες να ανήκει στο cluster 3, ενώ είναι πολύ μικρή η πιθανότητα να ανήκει στο cluster 0 μια που η απόστασή του από αυτό είναι η μεγαλύτερη με μεγάλη διαφορά. Ως προς τις hard distances, στο συγκεκριμένο στιγμιότυπο η μικρότερη απόσταση είναι αυτή από το hard cluster 3, ενώ η μεγαλύτερη από το hard cluster 4. Οι διαφορές όμως μεταξύ των αποστάσεων στην περίπτωση αυτή είναι πολύ μικρές, οπότε δεν μπορούμε με σιγουριά να αποφανθούμε σε ποιο hard cluster ανήκει ο τυχαίος χρήστης.

5.4. Στόχος του context clustering και του υπολογισμού αποστάσεων – Πιθανές εφαρμογές.

Στόχος της διαδικασίας του context clustering είναι ο διαχωρισμός των χρηστών σε ομάδες. Οι χρήστες που ανήκουν στην ίδια ομάδα έχουν κοινά χαρακτηριστικά ως προς τη συμπεριφορά και τις συνήθειές τους, γεγονός από το οποίο προκύπτουν πλεονεκτήματα και προοπτικές, όπως η παροχή υπηρεσιών προσαρμοσμένων στα κοινά αυτά χαρακτηριστικά ώστε να ικανοποιούνται οι ανάγκες και οι απαιτήσεις της ομάδας.

Στόχος της διαδικασίας του υπολογισμού αποστάσεων του τυχαίου χρήστη από τα clusters, είναι η εξαγωγή πληροφοριών που αφορούν στην ομοιότητα ή μη της συμπεριφοράς του χρήστη με τη συμπεριφορά των χρηστών των ήδη υπαρχουσών ομάδων. Έχοντας εξάγει τις πληροφορίες αυτές μπορούμε να κατατάξουμε το χρήστη στην ομάδα με την οποία έχει τα περισσότερα κοινά χαρακτηριστικά (άρα απέχει τη μικρότερη απόσταση από αυτή). Επίσης, επειδή η διαδικασία υπολογισμού των αποστάσεων πραγματοποιείται αρκετές φορές και παράλληλα με τη διαδικασία του context clustering, μας δίνεται η δυνατότητα ανίχνευσης τυχόν ύποπτης συμπεριφοράς του τυχαίου χρήστη. Ως ύποπτη συμπεριφορά μπορεί να θεωρηθεί για παράδειγμα η έλλειψη σταθερότητας στις συνήθειες του χρήστη. Η έλλειψη σταθερότητας εκφράζεται μέσω πολύ συχνών αλλαγών του cluster στο οποίο κατατάσσεται ο χρήστης. Ακολουθεί μια διαγραμματική αναπαράσταση των fuzzy clusters από τα οποία περνά ο τυχαίος χρήστης διαδοχικά κατά τη διάρκεια του user tracing.



Σχήμα 5.1: Η διαδοχή των clusters στα οποία κατατάσσεται ο τυχαίος χρήστης.

Παρατηρούμε πως ο χρήστης χαρακτηρίζεται από ασταθή συμπεριφορά, ιδιαίτερα στο πρώτο μισό του tracing. Στο δεύτερο μισό, έχει τάσεις σταθεροποίησης, αφού παραμένει τόσο στο cluster 4, όσο και στο cluster 2 για αρκετά μεγάλα χρονικά διαστήματα.

Πιθανές εφαρμογές

Τα βασικά χαρακτηριστικά του συστήματος που σχεδιάστηκε και υλοποιήθηκε στα πλαίσια της εργασίας είναι η επίγνωση κατάστασης χρήστη (situation awareness), η μεταβάσεις κατάστασης (situation transitions), ο διαχωρισμός των χρηστών σε ομάδες με βάση τις μεταβάσεις κατάστασης που πραγματοποιούν (context clustering), καθώς και ο υπολογισμός της ομοιότητας ή μη ενός τυχαίου χρήστη με τις ήδη υπάρχουσες ομάδες χρηστών. Τα χαρακτηριστικά αυτά καθιστούν το σύστημα δυνατό να χρησιμοποιηθεί σε context-aware εφαρμογές (εφαρμογές που χρησιμοποιούν επίγνωση πληροφορίας πλαισίου). Μερικές κατηγορίες τέτοιων εφαρμογών είναι:

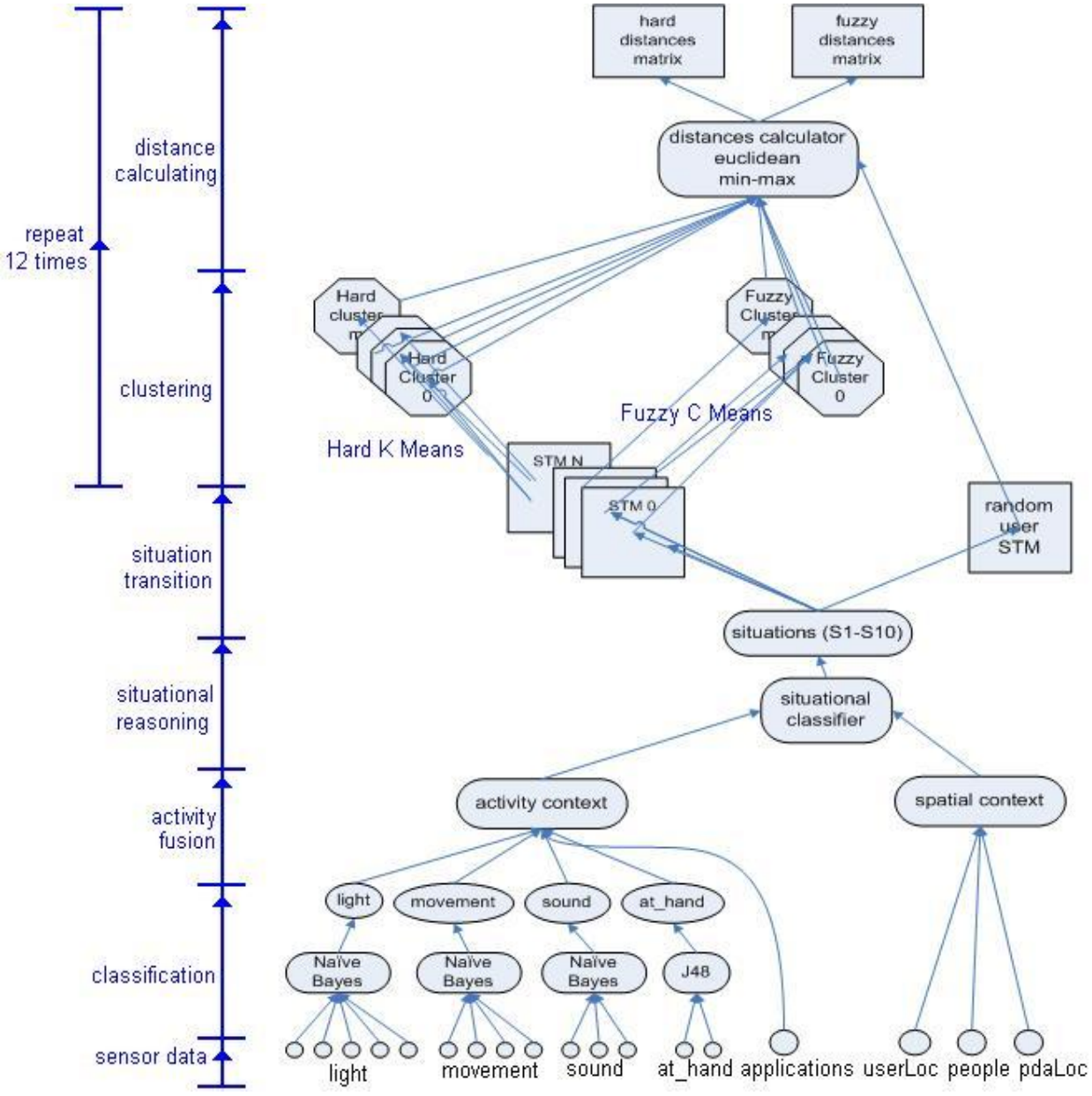
- Εφαρμογές που ασχολούνται με την εφαρμογή κανόνων ή/και την παροχή υπηρεσιών σε ομάδες χρηστών.
- Εφαρμογές που ασχολούνται με την εφαρμογή κανόνων ή/και την παροχή εξατομικευμένων υπηρεσιών σε χρήστες ανάλογα με την κατάσταση στην οποία βρίσκονται.
- Εφαρμογές που ασχολούνται με τη λήψη αποφάσεων σε επίπεδο ομάδων χρηστών.
- Εφαρμογές που ασχολούνται με την ανίχνευση “ύποπτων” συμπεριφορών σε ένα πολυχρηστικό περιβάλλον.

ΚΕΦΑΛΑΙΟ 6

ΛΕΠΤΟΜΕΡΕΙΕΣ ΥΛΟΠΟΙΗΣΗΣ

6.1. Σχηματική αναπαράσταση εφαρμογής.

Τα στάδια εκτέλεσης της εφαρμογής που υλοποιήθηκε στα πλαίσια της πτυχιακής αυτής εργασίας απεικονίζονται στο παρακάτω σχήμα.



Σχήμα 6.1: Τα στάδια της εφαρμογής που υλοποιήθηκε στα πλαίσια της εργασίας.

6.2. Το Εργαλείο WEKA.

Η υλοποίηση της εφαρμογής έγινε σε γλώσσα προγραμματισμού Java (Sun Microsystems, Java SDK 1.5). Για το φιλτράρισμα των δεδομένων του activity context καθώς και για την ταξινόμησή τους εν συνεχεία, χρησιμοποιήθηκαν το φίλτρο Remove, καθώς και οι ταξινομητές Naïve Bayes και J48 του WEKA. Το WEKA (Waikato Environment for Knowledge Analysis) είναι μια συλλογή από αλγορίθμους εκμάθησης (machine learning algorithms) που χρησιμοποιούνται σε εργασίες που αφορούν στην εξαγωγή μη τετριμμένων, άγνωστων προηγουμένως και ενδεχομένως χρήσιμων πληροφοριών από τα δεδομένα (data mining). Οι αλγόριθμοι του WEKA, είναι υλοποιημένοι σε Java και μπορούν είτε να εφαρμοστούν άμεσα πάνω στα δεδομένα, είτε να κληθούν μέσα από εξωτερικό κώδικα Java. Το WEKA περιέχει εργαλεία για προεπεξεργασία δεδομένων (π.χ. φίλτρα), ταξινόμηση, clustering, κανόνες συσχέτισης (association rules) καθώς και για απεικόνιση. Επίσης, μπορεί να χρησιμοποιηθεί και για τη δημιουργία νέων αλγορίθμων εκμάθησης.

6.3. Ο ταξινομητής καταστάσεων (situational classifier).

Η κλάση SituationClassifier αποτελεί την υλοποίηση του situational classifier. Η βασική μέθοδος της είναι η classifyInstance(String[] instance), που παίρνει ως όρισμα ένα πίνακα που περιέχει τα classified δεδομένα του activity context και τις πληροφορίες του spatial context και ανάλογα με το location, καλεί την ανάλογη μέθοδο για να αποφανθεί σε ποια κατάσταση αντιστοιχεί το τρέχον situational context. Ο πηγαίος κώδικας της κλάσης SituationClassifier είναι ο ακόλουθος:

```
/** Situation Classifier :: class that implements a situation
classifier
 * @author Panos Passias <FricK>
 */
public class SituationClassifier{
    String userLocation;
    String pdaLocation;
    String application;
    String peopleNearby;
    String movement;
    String light;
    String sound;
    String atHand;

    /** Constructor of the SituationClassifier class */
    public SituationClassifier(){
    }

    /** method that classifies a given instance
     * @param instance the instance to be classified
     * @return the class of the instance
     */
    public String classifyInstance(String[] instance){
        String situationClass = "";

        userLocation = instance[0];
        pdaLocation = instance[1];
        application = instance[2];
```

```
        peopleNearby = instance[3];
        movement = instance[4];
        light = instance[5];
        sound = instance[6];
        atHand = instance[7];

        if(userLocation.equalsIgnoreCase("OFFICE")){
            situationClass =
officeSituation(userLocation,pdaLocation,application,peopleNearby,move
ment,light,atHand);
            return situationClass;
        } else if(userLocation.equalsIgnoreCase("CORRIDOR1") ||
userLocation.equalsIgnoreCase("CORRIDOR2")){
            situationClass =
corridorSituation(userLocation,pdaLocation,movement,light);
            return situationClass;
        } else if(userLocation.equalsIgnoreCase("MEETING_ROOM")){
            situationClass =
meetingSituation(userLocation,pdaLocation,application,movement,atHand,
sound,light);
            return situationClass;
        } else if(userLocation.equalsIgnoreCase("COFFEE_ROOM")){
            situationClass =
coffeeSituation(userLocation,pdaLocation,application,movement,sound,li
ght,peopleNearby);
            return situationClass;
        } else{
            System.out.println("Unknown GPS user location -- unknown
situation");
        }
        return null;
    }

    /* method that classifies an instance given that the location is
in the Coffee room
     * @param userLoc the GPS location of the user
     * @param pdaLoc the GPS location of the pda
     * @param applic the application running on the pda
     * @param move movement value
```

```
     * @param sounds sound pressure value
     * @param lighting light value
     * @param people people nearby
     * @return class of the instance
     */
    protected String coffeeSituation(String userLoc,String
pdaLoc,String applic,
            String move,String sounds,String lighting,String people){
        String situation="";
        if(userLoc.equalsIgnoreCase(pdaLoc)){
            if(lighting.equalsIgnoreCase("NATURAL")){
                if(move.equalsIgnoreCase("HALT")){
                    if((sounds.equalsIgnoreCase("MODEST") ||
sounds.equalsIgnoreCase("LOUD")) && people.equalsIgnoreCase("FRIEND")
                    && applic.equalsIgnoreCase("NONE")){
                        situation = "COFFEE_ROOM_CHATTING";
                        return situation;
                    } else if(sounds.equalsIgnoreCase("LOUD") &&
people.equalsIgnoreCase("NOBODY")
                    && (applic.equalsIgnoreCase("WEB_BROWSING") ||
applic.equalsIgnoreCase("AGENDA") ||
applic.equalsIgnoreCase("NONE"))){
                        situation = "COFFEE_ROOM_CHILL_OUT";
                        return situation;
                    }
                }
            }
        } else{
            System.out.println("pda location different from user
location -- unkown situation");
            return null;
        }
        return null;
    }

    /* method that classifies an instance given that the location is
in the meeting room
     * @param userLoc the GPS location of the user
     * @param pdaLoc the GPS location of the pda
```

```
     * @param applic the application running on the pda
     * @param move movement value
     * @param hand atHand data value
     * @param sounds sound pressure value
     * @param lighting light value
     * @return class of the instance
     */
    protected String meetingSituation(String userLoc,String
pdaLoc,String applic,
            String move,String hand,String sounds,String lighting){
        String situation = "";
        if(userLoc.equalsIgnoreCase(pdaLoc)){
            if(lighting.equalsIgnoreCase("NATURAL")){
                if(move.equalsIgnoreCase("HALT")){
                    if (sounds.equalsIgnoreCase("MODEST") ||
sounds.equalsIgnoreCase("SILENT")){
                        if(hand.equalsIgnoreCase("NO") &&
applic.equalsIgnoreCase(".PPT")){
                            situation = "MEETING_ROOM_PRESENTING";
                            return situation;
                        } else if(hand.equalsIgnoreCase("YES") &&
applic.equalsIgnoreCase("DOWNLOADING_PRESENTATION")){
                            situation = "MEETING_ROOM_ATTENDING";
                            return situation;
                        }
                    }
                }
            }
        } else{
            System.out.println("pda location different from user
location -- unkown situation");
            return null;
        }
        return null;
    }

    /* method that classifies an instance given that the location is
in the office
     * @param userLoc the GPS location of the user
```

```
     * @param pdaLoc the GPS location of the pda
     * @param applic the application running on the pda
     * @param move movement value
     * @param lighting light value
     * @param people people nearby
     * @param hand atHand data value
     * @return class of the instance
     */
    protected String officeSituation(String userLoc,String
pdaLoc,String applic,String people,
            String move, String lighting, String hand){
        String situation = "";
        if(userLoc.equalsIgnoreCase(pdaLoc)){
            if(lighting.equalsIgnoreCase("NORMAL")){
                if(move.equalsIgnoreCase("HALT")){
                    if(people.equalsIgnoreCase("NOBODY")){
                        if(applic.equalsIgnoreCase("WORK_APP") ||
applic.equalsIgnoreCase("AGENDA")){
                            situation = "OFFICE_WORKING";
                            return situation;
                        } else{
                            situation = "OFFICE_NOT_WORKING";
                            return situation;
                        }
                    }
                }
            }
        } else{
            System.out.println("pda location different from user
location -- unkown situation");
            return null;
        }
        return null;
    }

    /* method that classifies an instance given that the location is
in a corridor
     * @param userLoc the GPS location of the user
     * @param pdaLoc the GPS location of the pda
```

```
     * @param move movement value
     * @param lighting light value
     * @return class of the instance
     */
    protected String corridorSituation(String userLoc,String
pdaLoc,String move,String lighting){
        String situation = "";
        if(userLoc.equalsIgnoreCase(pdaLoc)){
            if(lighting.equalsIgnoreCase("DARK")){
                if(move.equalsIgnoreCase("RUNNING")){
                    situation = userLoc+"_RUNNING";
                    return situation;
                } else if(move.equalsIgnoreCase("WALKING") ||
move.equalsIgnoreCase("WALKING_FAST")){
                    situation = userLoc+"_WALKING";
                    return situation;
                }
            } else{
                System.out.println("Unkown Situtation");
                return null;
            }
        } else{
            System.out.println("pda location different from user
location -- unkown situation");
            return null;
        }
        return null;
    }

    /** method that assigns a numbers from 0 to 9 to the situations
     * Given a situation as a String value it return its corresponding
number
     * @param situation the given situation as a String
     * @return the situations corresponding number
     */
    protected int situationDecoder(String situation){
        if(situation.equalsIgnoreCase("OFFICE_NOT_WORKING"))
            return 0;
        else if(situation.equalsIgnoreCase("OFFICE_WORKING"))
```

```
            return 1;
        else if(situation.equalsIgnoreCase("CORRIDOR1_WALKING"))
            return 2;
        else if(situation.equalsIgnoreCase("CORRIDOR1_RUNNING"))
            return 3;
        else if(situation.equalsIgnoreCase("MEETING_ROOM_PRESENTING"))
            return 4;
        else if(situation.equalsIgnoreCase("MEETING_ROOM_ATTENDING"))
            return 5;
        else if(situation.equalsIgnoreCase("CORRIDOR2_WALKING"))
            return 6;
        else if(situation.equalsIgnoreCase("CORRIDOR2_RUNNING"))
            return 7;
        else if(situation.equalsIgnoreCase("COFFEE_ROOM_CHILL_OUT"))
            return 8;
        else if(situation.equalsIgnoreCase("COFFEE_ROOM_CHATTING"))
            return 9;
        else{
            System.out.println("situationDecoder:: unkown situation");
            return -1;
        }
    }
}
```

6.4. Ο αλγόριθμος Hard K Means.

Η κλάση HardKMeans αποτελεί την υλοποίηση του αλγορίθμου clustering Hard K Means. Οι βασικές μέθοδοι της κλάσης είναι: η μέθοδος jFunction() που υλοποιεί τη συνάρτηση ανομοιότητας J, η μέθοδος initalizeCentroidMatrix() που αρχικοποιεί τα κεντροειδή των clusters, η μέθοδος updateUMatrix που ανανεώνει τον πίνακα συγγένειας U και η μέθοδος centroidCalculator που επαναπροσδιορίζει τα κεντροειδή των clusters. Ο πηγαίος κώδικας της κλάσης HardKMeans είναι ο ακόλουθος:

```
/** HardKMeans :: Class implementing the HardKMeans clustering algorithm
 *  @author Panos Passias <FricK>
 */
public class HardKMeans{
    double m;
    double e;
    double[][] hardKMeansUMatrix;
    double[][][] hardKMeansCentroidsMatrix;

    /** Constructor of the HardKMeans algorithm class
     * @param nofUsers the number of users used for clustering
     * @param nofClusters the number of clusters
     * @param nofSituations the number of possible situations a user can be in
     */
    public HardKMeans(double m, double e, int nofClusters, int nofUsers, int nofSituations){
        this.m = m;
        this.e = e;
        this.hardKMeansUMatrix = new double[nofClusters][nofUsers];
        this.hardKMeansCentroidsMatrix = new double[nofClusters][nofSituations][nofSituations];
    }

    /** The objective function to be minimized
     * @param users the users situation transition matrix
     * @return the value of the jFunction
```

```
     */
    public double jFunction(double[][][] users){
        double sum = 0.0;
        for(int i=0; i<hardKMeansCentroidsMatrix.length; i++)
            for(int k=0; k<users.length; k++)
                sum                                                    +=
squaredEuclidianDistance(users[k],hardKMeansCentroidsMatrix[i]);
        return sum;
    }

    /** Method that checks if the membership matrix is invalid */
    private boolean invalidUmatrix(){
        for(int i=0; i<hardKMeansUMatrix.length; i++){
            double sum = 0.0;
            for(int j=0; j<hardKMeansUMatrix[i].length; j++){
                sum+=hardKMeansUMatrix[i][j];
            }
            if(sum == 0.0)
                return true;
        }
        return false;
    }

    /** Method that updates the users' degree of membership in the
clusters
     * @param users the users’ situations transition matrix
     */
    protected void updateUMatrix(double[][][] users){
        boolean invalid = false;
        do{
            for(int i=0; i<hardKMeansCentroidsMatrix.length; i++){
                for(int j=0; j<users.length; j++){
                    hardKMeansUMatrix[i][j] = 0.0;
                    double[]          distances          =          new
double[hardKMeansCentroidsMatrix.length];
                    for(int   k=0;   k<hardKMeansCentroidsMatrix.length;
k++){
                        distances[k]                                    =
squaredEuclidianDistance(users[j],hardKMeansCentroidsMatrix[k]);
```

```
                    }
                    int cluster = minimumValueIndex(distances);
                    hardKMeansUMatrix[cluster][j] = 1.0;
                }
            }
            if(invalidUmatrix() == true){
                invalid = true;
                System.out.println("===== INVALID HARD U MATRIX - RE-
INITIALIZING CENTROIDS =====");
                initializeCentroidMatrix(users);
            }else{
                invalid = false;
            }
        }while(invalid);
    }

    /** method that finds the index of the minimum value in a matrix
     * @param matrix the matrix to be searched for the minimum value
     * @return the index of the minimum value
     */
    private int minimumValueIndex(double[] matrix){
        int minimumIndex = 0;
        double minValue = Double.MAX_VALUE;
        for(int i=0; i<matrix.length; i++){
            if(matrix[i]<minValue){
                minValue = matrix[i];
                minimumIndex = i;
            }
        }
        return minimumIndex;
    }

    /** Method that calculates the squared value of the Eucledean
distance
     * between two matrices
     * @return the squared value of the eucledean distance
     */
    private         double         squaredEuclidianDistance(double[][]
xMatrix,double[][] cMatrix){
```

```
        double norm = 0.0;
        for(int i=0; i<xMatrix.length; i++)
            for(int j=0; j<xMatrix[i].length; j++){
            norm = norm+Math.pow((xMatrix[i][j]-cMatrix[i][j]),2.0);
            }
        return norm;
    }

    /** Method that initializes the centroids of the clusters
     * @param users the users' situations transition matrix
     */
    protected void initializeCentroidMatrix(double[][][] users){
        Random rand = new Random();
        int[] used = new int[hardKMeansCentroidsMatrix.length];
        for(int i=0; i<used.length; i++)
            used[i] = -1;
        int datapoint = -1;
        for(int i=0; i<hardKMeansCentroidsMatrix.length; i++){
            do{
                datapoint = rand.nextInt(users.length);
            }while(existsInMatrix(datapoint,used));
            used[i] = datapoint;
            for(int k=0; k<hardKMeansCentroidsMatrix[i].length; k++)
                for(int l=0; l<hardKMeansCentroidsMatrix[i][k].length;
l++)
                    hardKMeansCentroidsMatrix[i][k][l]               =
users[datapoint][k][l];
        }
    }

    /** method that searches a matrix for a given value
     * @param number the value to be searched
     * @param matrix the matrix to search the value in
     * @return the boolean value, true if the value exists, false if
not
     */
    private boolean existsInMatrix(int number,int[] matrix){
        for(int i=0; i<matrix.length; i++){
            if(matrix[i] == number)
```

```
                return true;
            }
            return false;
        }

        /** method that calculates the number of users in a given cluster
         * @param nofCluster index of the cluster
         * @return the number of users in the given cluster
         */
        private double nofUsersInCluster(int nofCluster){
            double sum = 0.0;
            for(int j=0; j<hardKMeansUMatrix[nofCluster].length; j++){
                sum += hardKMeansUMatrix[nofCluster][j];
            }
            return sum;
        }

        /** method that nullifys the centroids matrix
         * @param centroids the centroids matrix
         */
        private void nullifyCentroids(double[][][] centroids){
            for(int i=0; i<centroids.length; i++)
                for(int j=0; j<centroids[i].length; j++)
                    for(int k=0; k<centroids[i][j].length; k++)
                        centroids[i][j][k] = 0.0;
        }

        /** Method that calculates the centroids of the clusters based on
the
         * users transitions matrix and and the cluster membership matrix
         * @param users the users situations transition matrix
         * @return the new centroids matrix
         */
        protected double[][][] centroidCalculator(double[][][] users){
            double[][][]            centers            =            new
double[hardKMeansCentroidsMatrix.length][hardKMeansCentroidsMatrix[0].
length][hardKMeansCentroidsMatrix[0][0].length];
            this.nullifyCentroids(centers);
            for(int i=0; i<centers.length; i++){
```

```
            double gi = this.nofUsersInCluster(i);
            for(int j=0; j<users.length; j++){
                if(hardKMeansUMatrix[i][j]==1.0){
                    centers[i] = addMatrices(centers[i],users[j]);
                }
            }
            centers[i] = multiplyNumberWithMatrix((1/gi),centers[i]);
        }
        return centers;
    }
    /** method that perfoms multiplication of a double value with a
matrix
     * @param number the double value
     * @param matrix the matrix
     * @return product (matrix) of the multiplication
     */
    private double[][] multiplyNumberWithMatrix(double
number,double[][] matrix){
        double[][] newMatrix = new
double[matrix.length][matrix[0].length];
        for(int i=0; i<matrix.length; i++)
            for(int j=0; j<matrix[i].length; j++)
                newMatrix[i][j] = number*matrix[i][j];
        return newMatrix;
    }

    /** method that adds two matrices
     * @return the matrix as sum of the two matrices
     */
    private double[][] addMatrices(double[][] matrix1,double[][]
matrix2){
        double[][] sumMatrix = new
double[matrix1.length][matrix1[0].length];
        for(int i=0; i<matrix1.length; i++)
            for(int j=0; j<matrix1[0].length; j++)
                sumMatrix[i][j] = matrix1[i][j] + matrix2[i][j];
        return sumMatrix;
    }
}
```

6.5. Ο αλγόριθμος Fuzzy C Means.

Η κλάση FuzzyCMeans αποτελεί την υλοποίηση του αλγορίθμου clustering Fuzzy C Means. Οι βασικές μέθοδοι της κλάσης είναι: η μέθοδος jFunction() που υλοποιεί τη συνάρτηση ανομοιότητας J, η μέθοδος initalizeCentroidMatrix() που αρχικοποιεί τα κεντροειδή των clusters, η μέθοδος initializeUMatrix() που αρχικοποιεί την πίνακα συγγένειας U, η μέθοδος updateUMatrix που ανανεώνει τον πίνακα συγγένειας U και η μέθοδος centroidCalculator που επαναπροσδιορίζει τα κεντροειδή των clusters. Ο πηγαίος κώδικας της κλάσης FuzzyCMeans είναι ο ακόλουθος:

```
/** FuzzyCMeans :: Class implementing the fuzzyCMeans clustering algorithm
 *  @author Panos Passias <FricK>
 */
public class FuzzyCMeans{
    double m;
    double e;
    protected double[][] fuzzyCMeansUMatrix;
    protected double[][][] fuzzyCMeansCentroidsMatrix;

    /** Constructor of the FuzzyCMeans algorithm class
     * @param nofUsers the number of users used for clustering
     * @param nofClusters the number of clusters
     * @param nofSituations the number of possible situations a user can be in
     */
    public FuzzyCMeans(double m, double e, int nofUsers, int nofClusters, int nofSituations){
        this.m = m;
        this.e = e;
        this.fuzzyCMeansUMatrix = new double[nofUsers][nofClusters];
        this.fuzzyCMeansCentroidsMatrix = new double[nofUsers][nofSituations][nofSituations];
    }

    /** The objective function to be minimized
     * @param users the users situation transition matrix
     * @return the value of the jFunction
```

```
     */
    public double jFunction(double[][][] users){
        double sum = 0.0;
        for(int i=0; i<fuzzyCMeansCentroidsMatrix.length; i++)
            for(int j=0; j<users.length; j++)
                sum = sum + Math.pow(fuzzyCMeansUMatrix[j][i],this.m)*squaredEuclidianDistance(fuzzyCMeansCentroidsMatrix[i],users[j]);
        return sum;
    }

    /** Method that updates the users' degree of membership in the clusters
     * @param situations the users situation transition matrix
     */
    protected void updateUMatrix(double[][][] situations){
        for(int i=0; i<fuzzyCMeansUMatrix.length; i++){
            for(int j=0; j<fuzzyCMeansUMatrix[i].length; j++){
                double sum = 0.0;
                double frob1, frob2;
                for(int k=0; k<fuzzyCMeansUMatrix[i].length; k++){
                    frob1 = frobeniusNorm(situations[i],fuzzyCMeansCentroidsMatrix[j]);
                    frob2 = frobeniusNorm(situations[i],fuzzyCMeansCentroidsMatrix[k]);
                    sum = sum + Math.pow(frob1/frob2,(2.0/(this.m-1.0)));
                }
                fuzzyCMeansUMatrix[i][j] = 1.0/sum;
            }
        }
    }

    /** Method that calculates the Eucledean distance between two matrices
     * @return the value of the Eucledean distance.
     */
    protected double euclidianDistance(double[][] xMatrix,double[][] cMatrix){
```

```
        double norm = 0.0;
        for(int i=0; i<xMatrix.length; i++)
            for(int j=0; j<xMatrix[i].length; j++)
                norm          =          norm+Math.pow((xMatrix[i][j]-
cMatrix[i][j]),2.0);
        norm = Math.sqrt(norm);
        return norm;
    }

    /** Method that calculates the squared value of the Eucledean
distance
     * between two matrices
     * @return the squared value of the eucledean distance
     */
    protected        double        squaredEuclidianDistance(double[][]
xMatrix,double[][] cMatrix){
        double norm = 0.0;
        for(int i=0; i<xMatrix.length; i++)
            for(int j=0; j<xMatrix[i].length; j++){
                norm          =          norm+Math.pow((xMatrix[i][j]-
cMatrix[i][j]),2.0);
            }
        return norm;
    }

    /** Method that calculates the distance between two matrices based
on the
     * frobenius norm
     * @return the value of the frobenius distance
     */
    protected   double   frobeniusNorm(double[][]   xMatrix,double[][]
cMatrix){
        double norm = 0.0;
        for(int i=0; i<xMatrix.length; i++)
            for(int j=0; j<xMatrix[i].length; j++)
                norm          =          norm+Math.pow((xMatrix[i][j]-
cMatrix[i][j]),2.0);
        norm = Math.sqrt(norm);
        return norm;
```

```
    }

    /** Method that initializes the users' membership matrix */
    protected void initalizeUMatix(){
        for(int i=0; i<fuzzyCMeansUMatrix.length; i++){
            double sum = 0.0;
            for(int j=0; j<fuzzyCMeansUMatrix[i].length-1; j++){
                double random;
                do{
                    random = Math.random();
                }while(random == 0.0 || random>(1.0-sum));
                sum = sum + random;
                fuzzyCMeansUMatrix[i][j] = random;
            }
            fuzzyCMeansUMatrix[i][fuzzyCMeansUMatrix[i].length-1]     =
1.0-sum;
        }
    }

    /** Method that initializes the centroids of the clusters */
    protected void initializeCentroidMatrix(){
        for(int i=0; i<fuzzyCMeansCentroidsMatrix.length; i++)
            for(int j=0; j<fuzzyCMeansCentroidsMatrix[i].length; j++)
                for(int                                          k=0;
k<fuzzyCMeansCentroidsMatrix[i][j].length; k++){
                    double random;
                    do{
                        random = Math.random();
                    }while(random == 0.0);
                    fuzzyCMeansCentroidsMatrix[i][j][k] = random;
                }
    }

    /** Method that calculates the centroids of the clusters based on
the
     * users transitions matrix and and the cluster membership matrix
     * @param users the users situations transition matrix
     * @return the new centroids matrix
     */
```

```
    protected double[][][] centroidCalculator(double[][][] users){
        double[] denom = calculateDenominators(fuzzyCMeansUMatrix);
        double[][][] centers = new
double[fuzzyCMeansUMatrix[0].length][users[0].length][users[0].length]
;
        for(int i=0; i<centers.length; i++)
            for(int j=0; j<centers[i].length; j++)
                for(int k=0; k<centers[i][j].length; k++)
                    centers[i][j][k] = 0.0;
        for(int j=0; j<centers.length; j++)
            for(int i=0; i<fuzzyCMeansUMatrix.length; i++)
                centers[j] =
addMatrices(centers[j],multiplyNumberWithMatrix(Math.pow(fuzzyCMeansUM
atrix[i][j],this.m),users[i]));
        for(int i=0; i<centers.length; i++)
            centers[i] =
multiplyNumberWithMatrix((1.0/denom[i]),centers[i]);
        return centers;
    }

    /** method that perfoms multiplication of a double value with a
matrix
     * @param number the double value
     * @param matrix the matrix
     * @return product (matrix) of the multiplication
     */
    private double[][] multiplyNumberWithMatrix(double
number,double[][] matrix){
        double[][] newMatrix = new
double[matrix.length][matrix[0].length];
        for(int i=0; i<matrix.length; i++)
            for(int j=0; j<matrix[i].length; j++)
                newMatrix[i][j] = number*matrix[i][j];
        return newMatrix;
    }

    /** method that adds two matrices
     * @return the matrix as sum of the two matrices
     */
```

```
    private double[][] addMatrices(double[][] matrix1,double[][]
matrix2){
        double[][]              sumMatrix              =              new
double[matrix1.length][matrix1[0].length];
        for(int i=0; i<matrix1.length; i++)
            for(int j=0; j<matrix1[0].length; j++)
                sumMatrix[i][j] = matrix1[i][j] + matrix2[i][j];
        return sumMatrix;
    }

    /** method that calculates the denominators of a sum used in the
     * FuzzyCMeans algorithm
     * @return the denominators matrix
     */
    private double[] calculateDenominators(double[][] matrix){
        double[] denominators = new double[matrix[0].length];
        for(int i=0; i<denominators.length; i++)
            denominators[i] = 0.0;
        for(int j=0; j<denominators.length; j++){
            for (int i = 0; i < matrix.length; i++)
                denominators[j]        =        denominators[j]        +
Math.pow(matrix[i][j], this.m);
        }
        return denominators;
    }
}
```

6.5. Υπολογισμός αποστάσεων (Distance Calculator)

Η κλάση DistanceCalculator αποτελεί την υλοποίηση του υπολογισμού των αποστάσεων του τυχαίου user από τα clusters. Οι βασικές μέθοδοι της κλάσης είναι: η μέθοδος eucledianDistancesMatrix() που υπολογίζει την ευκλείδια απόσταση και η μέθοδος minMaxDistancesMatrix() που υπολογίζει την προτεινόμενη απόσταση min-max. Ο πηγαίος κώδικας της κλάσης DistanceCalculator είναι ο ακόλουθος:

```
/** DistanceCalculator :: Class that calculates the eucledean and min-
max
 * distance between a given user and the cluster centroids.
 * @author Panos Passias <FricK>
 */
public class DistanceCalculator{
    int nofDifferentDistances;
    int nofSituations;
    int nofCentroids;

    /** Constructor of the DistanceCalculator class
    * @param nofSituations the number of different situations a user
can be in
    * @param nofCentroids the number of different clusters
    * @param nofDistances the number of different distances to be
calculated
    */
    public DistanceCalculator(int nofSituations,int nofCentroids, int
nofDistances){
        this.nofSituations = nofSituations;
        this.nofCentroids = nofCentroids;
        this.nofDifferentDistances = nofDistances;
    }

    /** method that calculates the distances between a given user and
the clusters'
     * centroids.
     * @param userMatrix the situation transition matrix of the given
user
     * @param centroidsMatrix the clusters' centroids matrix
```

```
     * @return the distances matrix
     */
    protected double[][] distancesFromCentroids(double[][]
userMatrix,double[][][] centroidsMatrix){
        double[][] distances = new
double[nofCentroids][nofDifferentDistances];
        double[][][] eMatrices = new
double[nofCentroids][nofDifferentDistances][userMatrix.length];

        for(int i=0; i<nofCentroids; i++){
            eMatrices[i][0] =
euclidianDistancesMatrix(userMatrix,centroidsMatrix[i]);
            eMatrices[i][1] =
minMaxDistancesMatrix(userMatrix,centroidsMatrix[i]);
            for(int j=0; j<nofDifferentDistances; j++){
                double nominator = 0.0;
                double denominator = 0.0;
                for(int k=0; k<eMatrices[i][j].length; k++){
                    nominator +=
eMatrices[i][j][k]*sumOfColumn(userMatrix,k);
                    denominator += sumOfColumn(userMatrix,k);
                }
                distances[i][j] = nominator/denominator;
            }
        }
        return distances;
    }

    /** method that calculates the sum of given column of a matrix
     * @param matrix the given matrix
     * @param nofColumn the index of the column
     * @return the sum of the column
     */
    private double sumOfColumn(double[][] matrix, int nofColumn){
        double sum = 0.0;
        for(int i=0; i<matrix.length; i++){
            sum += matrix[i][nofColumn];
        }
        return sum;
```

```
    }

    /** method that finds the index of the minimum value in a matrix
     * @param isIndex if true return the index of the min value
     *  if false return the minimum value
     * @param matrix the matrix to be searched.
     * @return the minimum value or its index depending on the value
of
     *  isIndex
     */
    protected double minimumValue(boolean isIndex, double[] matrix){
        int minimumIndex = 0;
        double minValue = Double.MAX_VALUE;
        for(int i=0; i<matrix.length; i++){
            if(matrix[i]<minValue){
                minValue = matrix[i];
                minimumIndex = i;
            }
        }
        if(isIndex)
          return (double)minimumIndex;
        return minValue;
    }

    /** method that finds the maximum value in a matrix
     * @param matrix the matrix to be searched.
     * @return the maximum value in the matrix
     */
    private double maximumValue(double[] matrix){
        int maximumIndex = 0;
        double maxValue = Double.MIN_VALUE;
        for(int i=0; i<matrix.length; i++){
            if(matrix[i]>maxValue){
                maxValue = matrix[i];
                maximumIndex = i;
            }
        }
        return maxValue;
    }
```

```
    /** method that calulates the min-max distances e-matrix
     * @param userMatrix the given user's situation transition matrix
     * @param centroidMatrix the clusters' centroids matrix
     * @return the min-max distances e-matrix
     */
    protected double[] minMaxDistancesMatrix(double[][]
userMatrix,double[][] centroidMatrix){
        double[] eMatrix = new double[this.nofSituations];
        for(int i=0; i<eMatrix.length; i++){
            eMatrix[i] =
Math.sqrt(Math.pow((maximumValue(userMatrix[i])-
maximumValue(centroidMatrix[i])),2)
            +Math.pow((minimumValue(false,userMatrix[i])-
minimumValue(false,centroidMatrix[i])),2));
        }
        return eMatrix;
    }

    /** method that calulates the eucledean distances e-matrix
     * @param userMatrix the given user's situation transition matrix
     * @param centroidMatrix the clusters' centroids matrix
     * @return the eucledean distances e-matrix
     */
    protected double[] euclidianDistancesMatrix(double[][]
userMatrix,double[][] centroidMatrix){
        double[] eMatrix  = new double[this.nofSituations];
        for(int i=0; i<eMatrix.length; i++){
            double sum = 0.0;
            for(int j=0; j<userMatrix[i].length; j++){
                sum = sum + Math.pow((userMatrix[i][j]-
centroidMatrix[i][j]),2);
            }
            eMatrix[i] = Math.sqrt(sum);
        }
        return eMatrix;
    }
}
```

6.6. Υλοποίηση σεναρίου.

Η κλάση Runner αποτελεί την υλοποίηση του σεναρίου που πραγματοποιεί το σύστημα και απεικονίζεται στο σχήμα 6.1. Είναι η κλάση που φορτώνει τα δεδομένα των χρηστών, τα περνάει από τα απαραίτητα φίλτρα και τα ταξινομεί, αφού πρώτα εκπαιδεύσει τους αντίστοιχους ταξινομητές (Naïve Bayes, J48). Στη συνέχεια δημιουργεί δύο νήματα. Το πρώτο εκτελεί το user tracing και καλεί τον situational classifier για τον προσδιορισμό της κατάστασης του κάθε χρήστη από το current situational context. Το δεύτερο ανά τακτά χρονικά διαστήματα εκτελεί τους αλγορίθμους Hard K Means και Fuzzy C Means και υπολογίζει της αποστάσεις του τυχαίου χρήστη από τα clusters. Ο πηγαίος κώδικας της κλάσης DistanceCalculator είναι ο ακόλουθος:

```
import java.io.*;

import weka.classifiers.*;
import weka.classifiers.bayes.*;
import weka.classifiers.trees.*;
import weka.core.*;
import weka.filters.unsupervised.attribute.*;

/** Runner :: class that implements a clustering scenario. 20 users
are used
 * to make the clusters. Several times during the proccess a users
distances
 * from the clusters' centers are being calculated.
 * @author Panos Passias <FricK>
 */
public class Runner {
  Instances[]  trainInstances;                 /** data to train the
classifiers with */
  Classifier[] classif;                 /** the classifiers */
  Instances[] data;                    /** users data to calculate the
centroids */
  Instances[][] testInstances;          /**  users filtered data */
  Instances randomUserData;             /** random user's data */
  Instances[] randomUserTestInstances;  /** random users filtered data
*/
```

```
  String trainData = "train_out_";
  String type1 = "movement";
  String type2 = "light";
  String type3 = "sound";
  String type4 = "at_hand";
  String extension = ".arff";
  int nofUsers = 20;        /** number of users */
  int nofSituations = 10;   /** number of different situations */
  int nofClusters = 5;      /** number of different clusters */
  int nofDistances = 2;       /** number of different distances to be
calculated */
  double[][][] situationMatrix = new double[nofUsers][nofSituations][
      nofSituations];      /** matrix holding the users' situation
transitions */
  double[][]            randomUserSitMatrix           =           new
double[nofSituations][nofSituations]; /** matrix holding the random
user's situation transitions */
  int[][] situationInts = new int[nofUsers][2]; /** matrix that holds
users' situation transition */
  int[] randomUserSitutationInts = new int[2];  /** matrix that holds
random user's situation transition */

  int nofRepetitions = 3600;    /** number of repetions (secs) */
  int curRepetition = 0;        /** current repetition (sec)  */
  MatrixUpdaterThread max = new MatrixUpdaterThread();   /** Thread
that updates the situations transition matrix */
  ClusteringAlgsThread dist = new ClusteringAlgsThread(); /** Thread
that performs the fuzzyCMeans
                                                          *       and
hardKMeans algorithm */

  double[][] fuzzyDistances = new double[nofClusters][nofDistances];
/** matrix that holds the fuzzy distances */
  double[][] hardDistances = new double[nofClusters][nofDistances];
/** matrix that holds the hard distances */

  /** Constructor of the Runner class */
  public Runner() {
  }
```

```
  /** method that initializes the situationInts matrix
   * @param matrix the situationInts matrix to be initialized
   */
  protected void sitIntsInit(int[][] matrix) {
    for (int i = 0; i < matrix.length; i++) {
      for (int j = 0; j < matrix[i].length; j++) {
        matrix[i][j] = -2;
      }
    }
  }

  /** method that loads the data to train the classifiers with
   * @return the loaded data in weka's Instances form.
   */
  protected Instances[] loadTrainData() {
    FileReader[] reader = new FileReader[4];
    Instances[] instances = new Instances[4]; // train datasets'
matrix
    try {
      reader[0] = new FileReader(trainData + type1 + extension);
//movement train data file
      reader[1] = new FileReader(trainData + type2 + extension);
//light train data file
      reader[2] = new FileReader(trainData + type3 + extension);
//sound train data file
      reader[3] = new FileReader(trainData + type4 + extension);
//at_hand train data file
      for (int i = 0; i < instances.length; i++) {
        instances[i] = new Instances(reader[i]); // Load the train
data
      }
      for (int i = 0; i < instances.length; i++) {
        instances[i].setClass(instances[i].attribute("Class"));
      }
      for (int i = 0; i < instances.length; i++) {
        reader[i].close();
      }
    }
```

```
    catch (FileNotFoundException ex) {
      ex.printStackTrace();
    }
    catch (IOException ex) {
      ex.printStackTrace();
    }
    return instances; // return the train data
  }

  /** method that builds the classifiers with the training data
   * @param instances the training data in weka's Instances form
   * @return the built classifiers
   */
  protected Classifier[] trainClassifiers(Instances[] instances) {
    Classifier[] classifs = new Classifier[4]; //classifiers
    for (int i = 0; i < classifs.length - 1; i++) {
      classifs[i] = new J48();
    }
    classifs[classifs.length - 1] = new NaiveBayes(); // naive-bayes
classifier for yes-no(at_hand) data
    try {
      for (int i = 0; i < instances.length; i++) {
        classifs[i].buildClassifier(instances[i]);    //    build    the
classifiers
      }
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
    return classifs; // return the classifiers
  }

  /** method that filters the users' data in order to be classifieed
   * @param dataset the dataset to be filtered
   * @param trainInst the dataset used to train the classifiers
   * @return the filtered datasets.
   */
  protected   Instances[]   filterData(Instances   dataset,   Instances[]
trainInst) {
```

```
    Remove[] filters = new Remove[4];
    Instances[] processedInst = new Instances[4]; // filtered
Instances matrix
    Instance[] processed = new Instance[4]; // filtered Instance
matrix
    for (int i = 0; i < filters.length; i++) {
      filters[i] = new Remove();
    }
    String[][] filterOptions = {
        {
        "-R", "30,31,32,33", "-V"}
        , {
        "-R", "15,16,17,18,19", "-V"}
        , {
        "-R", "27,28,29", "-V"}
        , {
        "-R", "12", "-V"}
    };
    try {
      for (int i = 0; i < filters.length; i++) {
        filters[i].setOptions(filterOptions[i]);
      }
      for (int i = 0; i < filters.length; i++) {
        filters[i].setInputFormat(dataset);
      }
      for (int i = 0; i < dataset.numInstances(); i++) {
        for (int j = 0; j < filters.length; j++) {
          filters[j].input(dataset.instance(i)); // input the
instances to the filters.
        }
      }
      for (int i = 0; i < filters.length; i++) {
        filters[i].batchFinished();
      }
      for (int i = 0; i < processedInst.length; i++) {
        processedInst[i] = filters[i].getOutputFormat(); // get the
output format
      }
      for (int i = 0; i < dataset.numInstances(); i++) {
```

```
        for (int j = 0; j < filters.length; j++) {
          processed[j] = filters[j].output(); // filtered instance
          processedInst[j].add(processed[j]);  //  add  the  filtered
insance to the filtered data
        }
      }
      for (int i = 0; i < processedInst.length; i++) { //insert the
class attribute to the filtered data

processedInst[i].insertAttributeAt(trainInst[i].classAttribute(),

processedInst[i].numAttributes());
        processedInst[i].setClass(trainInst[i].classAttribute());
      }
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
    return processedInst; // return the filtered data
  }

  /** main method of the Runner class */
  public static void main(String[] args) {
    Runner frick = new Runner();
    System.out.println("\nLoading training data...    ");
    frick.trainInstances = frick.loadTrainData();
    System.out.println("Training data loaded.");
    System.out.println("\nTraining classifiers...   ");
    frick.classif = frick.trainClassifiers(frick.trainInstances);
    System.out.println("Classifiers trained.");
    frick.data = new Instances[frick.nofUsers];
    System.out.println("\nLoading test data...      ");
    try {
      for (int i = 0; i < frick.data.length; i++) {
        frick.data[i] = new Instances(new FileReader("test" +
            Integer.toString(i + 1) + frick.extension));
      }
    }
    catch (FileNotFoundException ex) {
```

```
      ex.printStackTrace();
    }
    catch (IOException ex) {
      ex.printStackTrace();
    }
    System.out.println("Test data loaded.");
    System.out.println("\nFiltering test data...      ");
    frick.testInstances = new Instances[frick.nofUsers][];
    for (int i = 0; i < frick.testInstances.length; i++) {
      frick.testInstances[i] = frick.filterData(frick.data[i],
                                                frick.trainInstances);
    }
    System.out.println("Test data filtered.");
    System.out.println("\nLoading random user data...     ");
    try {
      frick.randomUserData         =         new         Instances(new
FileReader("random_user" +
          frick.extension));
    }
    catch (FileNotFoundException ex) {
      ex.printStackTrace();
    }
    catch (IOException ex) {
      ex.printStackTrace();
    }
    System.out.println("Random user data loaded.");
    System.out.println("\nFiltering random user data...     ");
    frick.randomUserTestInstances                                        =
frick.filterData(frick.randomUserData,
        frick.trainInstances);
    System.out.println("Random user data filtered.");

    System.out.println("\nStarting clustering...");
    frick.max.start();
    frick.dist.start();
    try {
      frick.max.join();
      frick.dist.join();
    }
```

```
    catch (InterruptedException ex) {
      ex.printStackTrace();
    }
  }

  /** MatrixUpdaterThread :: Thread class that updates the situations
transition matrix
   * implemented as inner class in order to have access to Runner
class' data.
   * @author Panos Passias <FricK>
   */
  class MatrixUpdaterThread
      extends Thread {
      /** the run method of the thread */
    public void run() {
      setName("MatrixUpdater");
      String[][] sClass = new String[nofUsers][8];
      double[][] klass = new double[nofUsers][4];
      String[] sUserClass = new String[8];
      double[] kUserClass = new double[4];
      randomUserSitutationInts[0] = -2;
      randomUserSitutationInts[1] = -2;
      SituationClassifier sit = new SituationClassifier();
      String[] situation = new String[nofUsers];
      for (int user = 0; user < nofUsers; user++) {
        situation[user] = new String();
      }
      String randomUserSituation = "";

      sitIntsInit(situationInts);

      for (int i = 0; i < data[0].numInstances(); i++) {
        for (int u = 0; u < data.length; u++) {
          sClass[u][0] = data[u].instance(i).toString(33);
          sClass[u][1] = data[u].instance(i).toString(37);
          sClass[u][2] = data[u].instance(i).toString(36);
          sClass[u][3] = data[u].instance(i).toString(35);
        }
        sUserClass[0] = randomUserData.instance(i).toString(33);
```

```
        sUserClass[1] = randomUserData.instance(i).toString(37);
        sUserClass[2] = randomUserData.instance(i).toString(36);
        sUserClass[3] = randomUserData.instance(i).toString(35);

        try {
          for (int u = 0; u < data.length; u++) {
            for (int v = 0; v < klass[u].length; v++) {
              klass[u][v]                                          =
classif[v].classifyInstance(testInstances[u][v].
                  instance(i));
              sClass[u][4 +
                  v] = trainInstances[v].classAttribute().value( (int)
klass[u][
                  v]);
            }
          }
          for (int v = 0; v < kUserClass.length; v++) {
            kUserClass[v]                                          =
classif[v].classifyInstance(randomUserTestInstances[
                v].instance(i));
            sUserClass[4 +
                v]  =  trainInstances[v].classAttribute().value(  (int)
kUserClass[
                v]);
          }
        }
        catch (Exception ex) {
          ex.printStackTrace();
        }
        for (int u = 0; u < sClass.length; u++) {
          situation[u] = sit.classifyInstance(sClass[u]);
          situationInts[u][0] = situationInts[u][1];
          situationInts[u][1] = sit.situationDecoder(situation[u]);
        }
        randomUserSituation = sit.classifyInstance(sUserClass);
        randomUserSitutationInts[0] = randomUserSitutationInts[1];
        randomUserSitutationInts[1]                              =
sit.situationDecoder(randomUserSituation);
        if (randomUserSitutationInts[0] != -2) {
```

```
          randomUserSitMatrix[randomUserSitutationInts[0]][
              randomUserSitutationInts[1]]++;
        }
        for (int r = 0; r < randomUserSitMatrix.length; r++) {
          for (int j = 0; j < randomUserSitMatrix[r].length; j++) {
            randomUserSitMatrix[r][j] = randomUserSitMatrix[r][j] /
                nofRepetitions;
          }
        }
        synchronized (situationMatrix) {
          updateSituationMatrix(situationInts);
          if ( ( ( (curRepetition + 1) % 300) == 0) && (curRepetition
!= 0)) {
            situationMatrix.notifyAll();
            try {
              situationMatrix.wait();
            }
            catch (InterruptedException ex) {
              ex.printStackTrace();
            }
          }
        }
        curRepetition++;
      }
      synchronized (situationMatrix) {
        situationMatrix.notifyAll();
      }
    }

    /** method that updates the situations transition matrix
     * @param matrix the situationInts matrix holding the current
situation
     * transition
     */
    protected void updateSituationMatrix(int[][] matrix) {
      for (int i = 0; i < matrix.length; i++) {
        if (matrix[i][0] != -2) {
          situationMatrix[i][matrix[i][0]][matrix[i][1]]++;
        }
```

```
      }
    }
  }

  /** ClusteringAlgsThread :: Thread class performing the fuzzyCMeans
and
   * hardKMeans algorithms and calculating the distances, implemented
as
   * inner class in order to hava access to Runner class' data.
   * @author Panos Passias <Frick>
   */
  class ClusteringAlgsThread
      extends Thread {
      /** Thread's run method */
    public void run() {
      setName("ClusteringAlgsThread");
      double[][][] normalizedMatrix;
      double oldJ = 0.0, newJ = 0.0;
      double oldJ2 = 0.0, newJ2 = 0.0;
      FuzzyCMeans       fuzzy       =       new       FuzzyCMeans(5.0,
0.00000001,nofUsers,nofClusters,nofSituations);
      HardKMeans        hard        =        new        HardKMeans(2.0,
0.00000001,nofClusters,nofUsers,nofSituations);
      synchronized (situationMatrix) {
        try {
          this.sleep(1000);
          situationMatrix.notifyAll();
        }
        catch (InterruptedException ex) {
          ex.printStackTrace();
        }
      }
      fuzzy.initalizeUMatix();
      fuzzy.initializeCentroidMatrix();
      normalizedMatrix     =     this.normalizeAndCopySituationMatrix(
(double)
          curRepetition);
      hard.initializeCentroidMatrix(normalizedMatrix);
      synchronized (situationMatrix) {
```

```
        while ( (curRepetition + 1) < nofRepetitions) {
          try {
            situationMatrix.wait();
            oldJ = 0.0;
            newJ = 0.0;
            normalizedMatrix  =  this.normalizeAndCopySituationMatrix(
(double)
                curRepetition);
            do {
              oldJ = newJ;
              fuzzy.fuzzyCMeansCentroidsMatrix                          =
fuzzy.centroidCalculator(normalizedMatrix);
              fuzzy.updateUMatrix(normalizedMatrix);
              newJ = fuzzy.jFunction(normalizedMatrix);
            }
            while ( (Math.abs(oldJ - newJ)) >= fuzzy.e);

            boolean finito = false;
            do {
              oldJ2 = newJ2;
              hard.updateUMatrix(normalizedMatrix);
              newJ2 = hard.jFunction(normalizedMatrix);
              if ( (Math.abs(oldJ2 - newJ2)) < hard.e) {
                finito = true;
              }
              else {
                hard.hardKMeansCentroidsMatrix                          =
hard.centroidCalculator(normalizedMatrix);
              }
            }
            while (finito == false);

            System.out.println("\n\n----------------------------------
---PROGRESS = " +Math.ceil( (100.0 / 3600.0) * curRepetition) +"% ----
----------------------------------------");
            DistanceCalculator          dista          =          new
DistanceCalculator(nofSituations,
                nofClusters, nofDistances);
```

```
            fuzzyDistances                                          =
dista.distancesFromCentroids(randomUserSitMatrix,fuzzy.fuzzyCMeansCent
roidsMatrix);
            System.out.println("Fuzzy  C  Means  :  Eucledean  current
cluster  = "+dista.minimumValue(true,cutColumn(fuzzyDistances,0)));
            System.out.println("Fuzzy  C  Means  :  Min-Max     current
cluster  = "+dista.minimumValue(true,cutColumn(fuzzyDistances,1)));
            printFuzzyDistances(fuzzyDistances);

            System.out.println();
            hardDistances                                           =
dista.distancesFromCentroids(randomUserSitMatrix,
              hard.hardKMeansCentroidsMatrix);
            System.out.println("Hard  K  Means  :  Eucledean  current
cluster  = "+dista.minimumValue(true,cutColumn(hardDistances,0)));
            System.out.println("Hard  K  Means  :  Min-Max     current
cluster  = "+dista.minimumValue(true,cutColumn(hardDistances,1)));
            printHardDistances(hardDistances);
            situationMatrix.notifyAll();
          }
          catch (InterruptedException ex) {
            ex.printStackTrace();
          }
        }
      }
    }

    /** method that cuts a column out of a given matrix
     * @param matrix the matrix
     * @param columnIndex the requested column's index
     * @return the column in array form
     */
    protected double[] cutColumn(double[][] matrix,int columnIndex){
        double[] column = new double[matrix.length];
        for(int i=0; i<column.length; i++)
            column[i] = matrix[i][columnIndex];
        return column;
    }
```

```
    /** method that prints the distances between the user and the
fuzzy cluster
     * centers
     * @param distances the distances to be printed
     */
    protected void printFuzzyDistances(double[][] distances){
        System.out.println("+-----------------------------Fuzzy
Distances--------------------+");
        System.out.println("|                 Eucledean               |
Min-Max            |");
        System.out.println("|-------------------------------------+---
----------------------|");
        for(int i=0; i<distances.length; i++){
            System.out.print("| Cluster "+i+" | ");
            for(int j=0; j<distances[i].length; j++){
             //   System.out.print(distances[i][j]+"  ");
                System.out.format("%1.16e  |  ",distances[i][j]);
            }
            System.out.println();
        }
        System.out.println("+-------------------------------------
----------------------+");
    }

    /** method that prints the distances between the user and the hard
cluster
     * centers
     * @param distances the distances to be printed
     */
    protected void printHardDistances(double[][] distances){
        System.out.println("+-----------------------------Hard
Distances--------------------+");
        System.out.println("|                 Eucledean               |
Min-Max            |");
        System.out.println("|-------------------------------------+---
----------------------|");
        for(int i=0; i<distances.length; i++){
            System.out.print("| Cluster "+i+" | ");
            for(int j=0; j<distances[i].length; j++){
```

```
                //   System.out.print(distances[i][j]+"  ");
                   System.out.format("%1.16e  |  ",distances[i][j]);
               }
               System.out.println();
           }
           System.out.println("+--------------------------------------
-----------------------+");
    }

    /** method that produces a nomralized copy of the situations
transition matrix
     * @param loop the current repetition (sec)
     * @return the normalized copy of the situations transition matrix
     */
    protected double[][][] normalizeAndCopySituationMatrix(double
loop) {
      double[][][] normalMatrix = new double[situationMatrix.length][
          situationMatrix[0].length][situationMatrix[0][0].length];
      for (int i = 0; i < situationMatrix.length; i++) {
        for (int j = 0; j < situationMatrix[i].length; j++) {
          for (int k = 0; k < situationMatrix[i][j].length; k++) {
            normalMatrix[i][j][k] = (situationMatrix[i][j][k] / loop);
          }
        }
      }
      return normalMatrix;
    }
  }
}
```

ΑΝΑΦΟΡΕΣ

1. Ian H. Witten and Eibe Frank: *Data Mining, Practical machine Learning Tools and Techniques, 2nd edition*, Elsevier.

2. Jan Jantzen: *Tutorial on Fuzzy Logic. Technical University of Denmark, Dept. of Automation, Pub No:1998-E-868*

3. J. C. Dunn : A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters", *Journal of Cybernetics* 3: 32-57, 1973

4. Christos B. Anagnostopoylos, Yiorgos Ntarlamidas, and Stathes Hadjiefthymiades*: Situation Awareness: Dealing with Vague Context, In proc. of the IEEE International Conference on Pervasive Services, ICPS2006, Lyon, France*