ΣΧΕΔΙΑΣΗ ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΝΔΙΑΜΕΣΟΥ ΛΟΓΙΣΜΙΚΟΥ ΕΝΗΜΕΡΟΥ ΠΕΡΙΕΧΟΜΕΝΟΥ ΓΙΑ ΚΙΝΗΤΕΣ ΣΥΣΚΕΥΕΣ

Η

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από τον

Αναστάσιο Κοντογιώργη

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟΝ ΤΟΜΕΑ ΤΕΧΝΟΛΟΓΙΕΣ ΚΑΙ ΕΦΑΡΜΟΓΕΣ

Ιούλιος 2005

Στους γονείς μου Αποστόλη και Δήμητρα, που δεν με είχαν όταν με χρειάζονταν.

Στην Πέννυ, που μου παρείχε το context για την πραγματοποίηση αυτής της εργασίας.

ΕΥΧΑΡΙΣΤΙΕΣ

Ολοκληρώνοντας τη συγγραφή της μεταπτυχιακής μου διατριβής αισθάνομαι την ανάγκη να ευχαριστήσω κάποιους ανθρώπους, που με βοήθησαν στη διεκπεραίωση αυτής της εργασίας.

Καταρχάς θα ήθελα να ευχαριστήσω τον κ. Δημήτριο Ι. Φωτιάδη για την επιλογή του θέματος και την επίβλεψη της εργασίας. Μου έδωσε τη δυνατότητα να εργαστώ απερίσπαστος στο Εργαστήριο Ιατρικής Τεχνολογίας και Ευφυών Πληροφοριακών Συστημάτων (*MedLab*) παρέχοντας μου τα απαραίτητα εφόδια για να πετύχω το στόχο μου.

Οφείλω ένα μεγάλο ευχαριστώ στον Απόστολο Ζάρρα, Λέκτορα του τμήματος Πληροφορικής, του Πανεπιστημίου Ιωαννίνων, για τη συνεπίβλεψη και την καθοδήγηση αυτής της εργασίας. Ήταν παρών όποτε τον χρειάστηκα και με στήριξε πρακτικά και ψυχολογικά.

Ευχαριστώ όλα τα μέλη του *MedLab*, για την υποστήριξη και το κουράγιο που μου έδιναν. Ιδιαίτερα ευχαριστώ τους Στέφανο Πέτσιο, Βασίλη Πρωτόπαπα, Φώτη Βαρτζιώτη και Γιάννη Αντωνίου για τις χρήσιμες συμβουλές τους σε τεχνικά θέματα της εργασίας.

Επίσης θα ήθελα να ευχαριστήσω όλους τους φίλους που με βοήθησαν, ο καθένας με τον τρόπο του, σε όλη τη διάρκεια της προσπάθειάς μου. Ευχαριστώ τους Πέννυ Μπούγια, Βασίλη Παπαδόπουλο (Λάκης), Νίκο Φωλίνα (Τραμπούκος), Χαλίλ Χούρι, Γιώργο Ρήγα (Ψηλός), Ανδρέα Φωτίου και Άκη Καραμουρατίδη. Χωρίς εσάς δεν ξέρω αν θα τα είχα καταφέρει.

Θα ήταν παράλειψη να μην ευχαριστήσω τους γονείς μου Αποστόλη και Δήμητρα, για την αμέριστη συμπαράσταση και την υπομονή τους σε όλη τη διάρκεια των σπουδών μου.

Αναστάσιος Α. Κοντογιώργης

Ιούλιος 2005

ΠΕΡΙΛΗΨΗ

Αναστάσιος Κοντογιώργης του Αποστόλου και της Δήμητρας.
MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιούλιος, 2005.
Τίτλος: Σχεδίαση και Ανάπτυξη Ενδιάμεσου Λογισμικού Ενήμερου Περιεχομένου για Κινητές Συσκευές. Επιβλέποντας: Δημήτριος Ι. Φωτιάδης.

Ένα από τα προβλήματα που αντιμετωπίζουν σήμερα οι μηχανικοί λογισμικού είναι η ανάπτυξη εφαρμογών, οι οποίες θα παρακολουθούν το περιβάλλον τους και θα προσαρμόζονται κατάλληλα στις αλλαγές που πραγματοποιούνται. Τέτοιου είδους εφαρμογές είναι απαραίτητες για κινητά και ενσωματωμένα συστήματα, όπου το περιβάλλον λειτουργίας και οι παράγοντες που επηρεάζουν την εφαρμογή μεταβάλλονται συνεχώς.

Η λύση σε αυτό το πρόβλημα μοιάζει να είναι η ανάπτυξη κατάλληλου ενδιάμεσου λογισμικού, για την υποστήριξη αυτών των εφαρμογών. Η ανάπτυξη ενδιάμεσου λογισμικού για κινητές συσκευές, παρουσιάζει σημαντικά προβλήματα που σχετίζονται με την πολυπλοκότητα του σχεδιασμού και τους περιορισμένους υπολογιστικούς πόρους των συσκευών. Σε αυτήν την εργασία προτείνουμε μία υπηρεσία ενδιάμεσου λογισμικού για την υποστήριξη και την προσαρμογή εφαρμογών σε δυναμικά μεταβαλλόμενα περιβάλλοντα.

Η υπηρεσία σχεδιάστηκε εξ ολοκλήρου για κινητές συσκευές. Για το λόγο αυτό υλοποιήθηκε με ειδικό «ελαφρύ» λογισμικό. Η υπηρεσία παρακολουθεί το περιβάλλον εκτέλεσης, επεξεργάζεται τα δεδομένα και προσαρμόζει αυτόματα τη λειτουργία της εφαρμογής. Η λειτουργία του ενδιάμεσου λογισμικού γίνεται με διαφάνεια προς το χρήστη, παρέχοντας όμως την δυνατότητα να συμμετέχει και ο ίδιος στη διαδικασία προσαρμογής δηλώνοντας τις προτιμήσεις του.

Η υπηρεσία μπορεί να χρησιμοποιηθεί για την ανάπτυξη ευφυών εφαρμογών που δέχονται γεγονότα από το περιβάλλον και προσαρμόζουν ανάλογα τη λειτουργία τους. Επιπλέον μπορεί εύκολα να ενσωματωθεί σε υπάρχον ενδιάμεσο λογισμικό, αυξάνοντας τη λειτουργικότητά του.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

1.1 Κινητά και Διάχυτα Συστήματα

Την τελευταία δεκαετία παρατηρείται μια ραγδαία εξάπλωση των κινητών συσκευών στην καθημερινή μας ζωή. Οι υπολογιστές αποκτούν όλο και μικρότερες διαστάσεις, ώστε να μεταφέρονται εύκολα και να μεσολαβούν σε περισσότερες διαδικασίες στην καθημερινότητα του χρήστη. Οι κινητές συσκευές σήμερα περιλαμβάνουν φορητούς υπολογιστές (*laptops*), προσωπικούς ψηφιακούς βοηθούς (*PDAs*), υπολογιστές παλάμης (*palmtops*), έξυπνα κινητά τηλέφωνα και φορέσιμες συσκευές, όπως ρολόγια.

Η ευρεία χρήση κινητών υπολογιστικών συσκευών έχει ως αποτέλεσμα τη σταδιακή μετατροπή των σύγχρονων υπολογιστικών συστημάτων από *κατανεμημένα* σε *κινητά*. Οι υπολογιστές δεν συνδέονται πλέον, μόνο μέσω του *Internet*, αλλά σχηματίζουν και ασύρματα δίκτυα με κινητούς κόμβους. Οι κόμβοι αυτών των δικτύων αποτελούνται από σταθερούς υπολογιστές, αλλά και κινητές συσκευές, οι οποίες συνεχώς αλλάζουν θέση και περιβάλλον λειτουργίας.

Τα κινητά συστήματα περιλαμβάνουν ένα μεγάλο αριθμό προσπελάσιμων και συχνά αόρατων υπολογιστικών συσκευών. Οι συσκευές αυτές μπορεί να βρίσκονται ενσωματωμένες στο περιβάλλον ή στο χρήστη. Συνδέονται σε μία ταχύτατα και αυθαίρετα μεταβαλλόμενη δικτυακή υποδομή, στην οποία βρίσκονται κατανεμημένες υπηρεσίες, διαθέσιμες προς τους χρήστες.

Οι κινητές συσκευές έχουν κάποια ιδιαίτερα χαρακτηριστικά, που τις διαφοροποιούν από τους σταθερούς ηλεκτρονικούς υπολογιστές, ως προς το λογισμικό που μπορούν να υποστηρίξουν:

- Έχουν περιορισμένους υπολογιστικούς πόρους όπως: υπολογιστική ισχύ, μνήμη και μόνιμο αποθηκευτικό χώρο.
- Επικοινωνούν μεταξύ τους και με άλλους υπολογιστές ασύρματα, σχηματίζοντας δίκτυα με ακαθόριστη δομή και συχνές συνδέσεις και αποσυνδέσεις κόμβων (*ad – hoc* δίκτυα). Το εύρος ζώνης στην επικοινωνία των συσκευών αυτών είναι συνήθως περιορισμένο, ενώ η συνεχής διαθεσιμότητα του δικτύου δεν είναι δεδομένη.
- Ο χρήστης έχει τη δυνατότητα να κινείται στο χώρο αλλάζοντας συνεχώς το περιβάλλον λειτουργίας της συσκευής. Σε κάποιες περιπτώσεις η γεωγραφική θέση του χρήστη και περιβαντολογικοί παράγοντες μπορεί να επηρεάζουν την εκτέλεση της εφαρμογής που τρέχει στη συσκευή.
- Οι κινητές συσκευές έχουν περιορισμένη αυτονομία. Η λειτουργία τους εξαρτάται απόλυτα από τη μπαταρία τους.

Τα συστήματα με τα παραπάνω χαρακτηριστικά ονομάζονται *διάχυτα* (*pervasive ή ubiquitous*). Ο όρος *pervasive* αναφέρεται για πρώτη φορά από τον *Weiser* το *1991* [1]. Ο όρος περιγράφει τη *διαφανή ενσωμάτωση των υπολογιστικών συσκευών στην καθημερινότητα του χρήστη, όπου η τεχνολογία χάνεται στο παρασκήνιο εστιάζοντας στο χρήστη και τις δραστηριότητές του*.

Οι εφαρμογές που λειτουργούν σε διάχυτα συστήματα πρέπει να μπορούν να *προσαρμόζονται* σε μεταβαλλόμενες συνθήκες. Για να το επιτύχουν αυτό απαιτείται να έχουν *επίγνωση* του περιβάλλοντος στο οποίο λειτουργούν και των *παραγόντων* που επηρεάζουν τη λειτουργία τους. Το σύνολο των παραγόντων που επηρεάζουν τη λειτουργία μιας εφαρμογής ονομάζεται *context*, π.χ. τοποθεσία, χρόνος, θερμοκρασία, άλλοι άνθρωποι σε κοντινή απόσταση κ.α. Οι εφαρμογές που μπορούν να γνωρίζουν το *context* και να το χρησιμοποιήσουν κατάλληλα, ώστε να προσαρμόσουν τις λειτουργίες τους, χαρακτηρίζονται ως *Context - Aware (CA)*.

Με τη χρήση του *context*, τα υπολογιστικά συστήματα μπορούν να γίνουν φιλικότερα προς τους χρήστες, πιο αποδοτικά και να παρέχουν ποιοτικότερες υπηρεσίες. Τα χαρακτηριστικά αυτά είναι ιδιαίτερα σημαντικά για διάχυτα συστήματα, όπου το *context* μεταβάλλεται συχνά και απρόβλεπτα και οι υπηρεσίες πρέπει να προσαρμόζονται στις μεταβολές.

Για να αποκτήσει μια εφαρμογή *CA* χαρακτηριστικά, πρέπει να αναπτυχθεί επιπρόσθετο λογισμικό. Το πρόσθετο λογισμικό αφορά, την εύρεση, τη διαχείριση, και τη χρησιμοποίηση του *context*, για την παροχή ποιοτικότερων υπηρεσιών στο χρήστη. Σύμφωνα με τις αρχές της τεχνολογίας λογισμικού τα *CA* συστήματα θα πρέπει να καλύπτουν τις απαιτήσεις που σχετίζονται με *ετερογένεια, κατανομή, διαλειτουργικότητα και επεκτασιμότητα* [2]. Επιπρόσθετα θα πρέπει να σχεδιαστούν ώστε να ανταποκρίνονται σε παράγοντες όπως, ξαφνικά ή απρόβλεπτα φαινόμενα, μετακίνηση φυσικών ή λειτουργικών οντοτήτων, διαμόρφωση, αβεβαιότητα, κακόβουλες ενέργειες και αποτυχίες που σχετίζονται με το φυσικό περιβάλλον [3].

Η ανάπτυξη συστημάτων με αυτά τα χαρακτηριστικά είναι προφανώς, δύσκολη και χρονοβόρα διαδικασία για τους μηχανικούς λογισμικού. Το ζητούμενο είναι η σχεδίαση και ανάπτυξη μιας κοινής πλατφόρμας, πάνω στην οποία να μπορούν εύκολα και ευέλικτα να χτιστούν *CA* εφαρμογές. Μία κοινή βάση ανάπτυξης εξασφαλίζει την επαναχρησιμοποίηση λογισμικού και ανεξαρτητοποιεί την εφαρμογή από το λογισμικό που αφορά τη διαχείριση του *context*. Το λογισμικό υποστήριξης *CA* εφαρμογών παίζει το ρόλο *ενδιάμεσου λογισμικού* κάτω από την εφαρμογή και παρέχει υπηρεσίες διαχείρισης του *context*.

Η σχεδίαση και ανάπτυξη ενδιάμεσου λογισμικού για κινητές συσκευές περιλαμβάνει ένα πολύ βασικό πρόβλημα. Οι λιγοστοί υπολογιστικοί πόροι των κινητών συσκευών περιορίζουν σε μεγάλο βαθμό τις λειτουργικές ικανότητες, αλλά και τον τρόπο ανάπτυξης του λογισμικού. Το ενδιάμεσο λογισμικό για να είναι εύχρηστο πρέπει να είναι οικονομικό στην κατανάλωση πόρων, ώστε να μην μειώνει τη λειτουργικότητα της κινητής εφαρμογής. Αυτή η απαίτηση αποτελεί ίσως τη μεγαλύτερη δυσκολία, αφού κατά κανόνα το ενδιάμεσο λογισμικό περιλαμβάνει υπολογιστικά σύνθετες λειτουργίες και απαιτεί μεγάλο διαθέσιμο αποθηκευτικό χώρο.

Η ανάπτυξη κατάλληλου ενδιάμεσου λογισμικού για *CA* εφαρμογές έχει προσελκύσει το ενδιαφέρον της ερευνητικής κοινότητας, την τελευταία δεκαετία. Η κατάλληλη αρχιτεκτονική, οι υπηρεσίες που παρέχονται αλλά και ο τρόπος αλληλεπίδρασης με το επίπεδο εφαρμογής είναι τα βασικά θέματα έρευνας.

Στη συνέχεια του κειμένου με τον όρο *εφαρμογή* θα αναφερόμαστε στο λογισμικό που τρέχει στο επίπεδο εφαρμογής, πάνω στην κινητή συσκευή. Με τον όρο *σύστημα* θα αναφερόμαστε, σε όλο το σύστημα ενδιάμεσου λογισμικού - εφαρμογής, καθώς και των υπόλοιπων συστατικών, υλικού ή λογισμικού που σχετίζονται με αυτά.

1.2 Περιεχόμενο Εργασίας

Σκοπός της μεταπτυχιακής αυτής εργασίας είναι η σχεδίαση και ανάπτυξη ενδιάμεσου λογισμικού για την υποστήριξη *CA* εφαρμογών σε κινητές συσκευές. Για το σκοπό αυτό αναλύουμε τα ιδιαίτερα χαρακτηριστικά και τις απαιτήσεις ενός τέτοιου λογισμικού. Εντοπίζουμε βασικές υπηρεσίες που πρέπει να παρέχονται και προτείνουμε μια κατάλληλη αρχιτεκτονική για την υποστήριξή τους. Με βάση τις προδιαγραφές που προέκυψαν από την ανάλυση απαιτήσεων υλοποιήσαμε το ενδιάμεσο λογισμικό.

Ο στόχος της σχεδίασης και ανάπτυξης του ενδιάμεσου λογισμικού ήταν, να προκύψει μία υπηρεσία που θα διαχειρίζεται το *context*, με διαφάνεια προς την εφαρμογή και θα προσαρμόζει ανάλογα τη λειτουργία της. Για τη διαχείριση του *context* λάβαμε υπόψιν βασικές ιδιότητες της *context* πληροφορίας, όπως η *ετερογένεια*, η *αξιοπιστία* και η *χρονική εξάρτηση*. Οι ιδιότητες αυτές δημιουργούν επιπλέον απαιτήσεις για το ενδιάμεσο λογισμικό. Για την κάλυψη αυτών των απαιτήσεων υλοποιήθηκαν κατάλληλοι μηχανισμοί, που αποκρύπτουν τα ιδιαίτερα χαρακτηριστικά του *context*, κάνοντας αποτελεσματική διαχείριση της πληροφορίας.

Ένα ακόμα ζητούμενο για το ενδιάμεσο λογισμικό, είναι να λειτουργεί πάνω σε κινητές συσκευές. Οι κινητές συσκευές χαρακτηρίζονται από περιορισμένους υπολογιστικούς πόρους και οι εφαρμογές που λειτουργούν πάνω σε αυτές είναι περιορισμένων δυνατοτήτων. Επιπλέον, η διαχείριση του *context* περιλαμβάνει σύνθετες διαδικασίες, τόσο υπολογιστικά, όσο και σε κατανάλωση μνήμης και αποθηκευτικού χώρου.

Για το λόγο αυτό, η υλοποίηση της υπηρεσίας έγινε με «ελαφρύ» λογισμικό και με γνώμονα την ελάχιστη κατανάλωση υπολογιστικών πόρων.

Για να αποτιμήσουμε τη διαχείριση των πόρων του λογισμικού, εκτελέσαμε ένα σύνολο πειραμάτων, όπου μετρήσαμε την κατανάλωση των κρίσιμων υπολογιστικών παραμέτρων όπως, η μνήμη, η χρήση της *CPU* και ο μόνιμος αποθηκευτικός χώρος μιας συσκευής. Επιπλέον υλοποιήσαμε μια εφαρμογή για κινητές συσκευές, «πάνω» από το ενδιάμεσο λογισμικό, δίνοντας της *CA* χαρακτηριστικά, για να ελέγξουμε την ορθή λειτουργία και την αξιοπιστία του λογισμικού. Στην εφαρμογή δοκιμάστηκαν διάφορα σενάρια λειτουργίας, ώστε να αποτιμηθεί η χρηστικότητα και η αξιοπιστία του ενδιάμεσου λογισμικού.

Ο χειρισμός των χρονικών εξαρτήσεων του *context* αποτελεί τη βασική καινοτομία αυτής της εργασίας. Συγκεκριμένα προτείνουμε ένα μοντέλο που περιγράφει τα χρονικά χαρακτηριστικά του *context* και μία μέθοδο για το χειρισμό τους. Η δεύτερη καινοτομία της εργασίας είναι η δυνατότητα του ενδιάμεσου λογισμικού να λειτουργεί εξ ολοκλήρου πάνω σε κινητές συσκευές. Η διαχείριση του *context* και η αυτόματη προσαρμογή της εφαρμογής εκτελούνται πάνω στην κινητή συσκευή παρέχοντας αυτονομία κίνησης στο χρήστη της εφαρμογής.

1.3 Οργάνωση Εργασίας

Στο Κεφάλαιο 2 περιγράφονται τα *CA* συστήματα και αναφέρονται βασικές έννοιες και ορισμοί. Στο Κεφάλαιο 3 γίνεται αναφορά στις σχετικές εργασίες που έχουν γίνει στο αντικείμενο αυτό. Στο Κεφάλαιο 4 γίνεται η ανάλυση απαιτήσεων του συστήματος που υλοποιήσαμε. Στο Κεφάλαιο 5 περιγράφονται οι λειτουργίες του συστήματος για την κάλυψη αυτών των απαιτήσεων. Το Κεφάλαιο 6 αναφέρεται στο μοντέλο του *context*, με το βάση το οποίο γίνεται η διαχείριση της πληροφορίας. Στο Κεφάλαιο 7 περιγράφεται η αρχιτεκτονική του λογισμικού και η κατανομή των ρόλων μεταξύ των υποσυστημάτων. Στο Κεφάλαιο 8 αναφέρονται θέματα υλοποίησης και περιγράφεται η κινητή εφαρμογή που αναπτύχθηκε πάνω από το ενδιάμεσο λογισμικό. Επιπλέον περιγράφονται διάφορα σενάρια χρήσης της εφαρμογής με τα οποία ελέγξαμε την ορθή λειτουργία του λογισμικού. Στο Κεφάλαιο 9 παρουσιάζονται τα πειράματα που έγιναν και γίνεται σχολιασμός των

αποτελεσμάτων. Στο Κεφάλαιο 10 συγκρίνουμε την παρούσα εργασία με σχετικές εργασίες και επισημαίνουμε τα καινοτόμα στοιχεία. Επιπλέον στο Κεφάλαιο 10 αναφέρουμε συμπεράσματα και δίνουμε κατευθύνσεις για μελλοντική έρευνα.

ΚΕΦΑΛΑΙΟ 2. ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΚΑΙ ΟΡΙΣΜΟΙ

Στο κεφάλαιο αυτό ορίζουμε τις βασικές έννοιες και περιγράφουμε τα σημαντικότερα θέματα, που αφορούν τα *CA* συστήματα.

2.1 Έννοια του Context

Το *context* σαν έννοια δεν είναι καινούργια. Από τις αρχές της δεκαετίας του *'60* χρησιμοποιήθηκε σε πεδία όπως λειτουργικά συστήματα, γλώσσες προγραμματισμού, τεχνητή νοημοσύνη και επικοινωνία ανθρώπου μηχανής. Τα τελευταία χρόνια συνδέθηκε άμεσα με τον τομέα των διάχυτων υπολογιστικών συστημάτων. Υπάρχουν πολλοί ορισμοί για το *context* στη βιβλιογραφία. Ένας από τους πιο εύστοχους δόθηκε από τον *Dey* [4] :

«Context είναι οποιαδήποτε πληροφορία που μπορεί να χρησιμοποιηθεί για να χαρακτηρίσει μια κατάσταση ή μία οντότητα. Μια οντότητα είναι ένα άτομο, μία θέση ή ένα αντικείμενο που μπορεί να θεωρηθεί σχετικό με την αλληλεπίδραση μεταξύ ενός χρήστη και μιας εφαρμογής, συμπεριλαμβανομένου του χρήστη και της εφαρμογής»

Πολλές κατηγοριοποιήσεις έχουν προταθεί για τις παραμέτρους του *context*, με βάση την πηγή προέλευσης τους [5,6]. Μια πλήρης κατηγοριοποίηση είναι η εξής:

- *Φυσικό context*: π.χ. παράγοντες από το φυσικό περιβάλλον, όπως φωτεινότητα, επίπεδο θορύβου, θερμοκρασία, ταχύτητα κ.α.

- *Υπολογιστικό context*: π.χ. δικτυακή σύνδεση, εύρος ζώνης, υπολογιστικοί πόροι σε κοντινή απόσταση, όπως εκτυπωτές ή τερματικά κ.α.
- *Χαρακτηριστικά της συσκευής*: π.χ. μέγεθος, βάρος, κλίση της συσκευής, μπαταρία, μνήμη, ταχύτητα *CPU*, αποθηκευτικός χώρος κ.α.
- *Context χρήστη*: π.χ. προφίλ, διάθεση, προτιμήσεις, θέση του χρήστη κ.α.
- *Χρόνος*: π.χ. ώρα, ημέρα, ημερομηνία κ.α.
- *Πληροφορία από την εφαρμογή*: *e-mails*, σελίδες επισκέφτηκε ο χρήστης, ηλεκτρονικό ημερολόγιο, πληροφορία από βάσεις δεδομένων κ.α.

Επιπλέον το *context* μπορεί να κατηγοριοποιηθεί σε *στατικό* ή *δυναμικό* [7]. Στατικό είναι το *context* του οποίου η πληροφορία είναι αμετάβλητη, όπως η ημερομηνία γέννησης κάποιου ατόμου. Δυναμικό είναι το *context* που μπορεί να αλλάξει.

Άλλη μία διάκριση μεταξύ, *ενεργού* και *παθητικού context* μπορεί να γίνει με βάση το τρόπο με τον οποίο το χειρίζεται η εφαρμογή [6]. Στο ενεργό *context*, μια εφαρμογή αυτόματα προσαρμόζεται στο νέο *context* αλλάζοντας τη συμπεριφορά της ή εκτελώντας κατάλληλες ενέργειες. Στο παθητικό *context*, η εφαρμογή παρουσιάζει το νέο ή ενημερωμένο *context* στον ενδιαφερόμενο χρήστη ή το αποθηκεύει για μελλοντική χρήση. Είναι προφανές ότι σε διάχυτα περιβάλλοντα υπολογισμού το δυναμικό και ενεργό *context* είναι η πιο ενδιαφέρουσα περίπτωση, αλλά και πιο η δύσκολη στο χειρισμό.

Η πληροφορία που συνθέτει το *context* έχει από τη φύση της ένα σύνολο ιδιαίτερων χαρακτηριστικών. Τα χαρακτηριστικά αυτά καθορίζουν το μοντέλο που θα χρησιμοποιηθεί για τη διαχείριση του *context* σε ένα *CA* σύστημα [7].

Αναξιοπιστία

Η αναξιοπιστία οφείλεται κυρίως στο ότι προέρχεται και από το φυσικό περιβάλλον το οποίο χαρακτηρίζεται από αστάθεια και ξαφνικές αλλαγές. Οι συχνές και απότομες μεταβολές έχουν ως αποτέλεσμα να γίνεται ασυνεπής η πληροφορία χρονικά. Η χρονική ασυνέπεια σημαίνει ότι, η τιμή μιας παραμέτρου μπορεί να αλλάζει ταχύτατα, κάνοντας ασυνεπείς τις τιμές που ανιχνεύτηκαν πρόσφατα.

Επιπλέον η ακρίβεια με την οποία οι αισθητήρες υλικού παρέχουν δεδομένα δεν είναι εξασφαλισμένη. Οι αισθητήρες μπορεί να επηρεάζονται από περιβαντολλογικούς παράγοντες, φυσικές φθορές ή κακόβουλες ενέργειες. Επίσης είναι πολύ πιθανή η απώλεια δεδομένων κατά τη μετάδοση, λόγω ασταθών συνδέσεων.

Διάφορες Αναπαραστάσεις

Το είδος της πληροφορίας μπορεί να κυμαίνεται σε πολλά επίπεδα από χαμηλού επιπέδου φυσικά δεδομένα, μέχρι υψηλού εννοιολογικού επιπέδου πληροφορία. Συνήθως υπάρχει μεγάλο χάσμα μεταξύ της πληροφορίας που παρέχουν οι αισθητήρες και αυτής που μπορεί να χρησιμοποιήσει μία *CA* εφαρμογή, π.χ., ένας αισθητήρας που ανιχνεύει τη γεωγραφική θέση, μπορεί να επιστρέφει αριθμητικές τιμές που συμβολίζουν γεωγραφικές συντεταγμένες, ενώ η εφαρμογή να απαιτεί την οδό που βρίσκεται ο χρήστης.

Αλληλοεξάρτηση

Πληροφορία που προέρχεται από πολλές διαφορετικές πηγές μπορεί να συνδυαστεί και να εξαχθούν συμπεράσματα για το *context*. Πληροφορίες που αφορούν διάφορες οντότητες μπορούν να συσχετιστούν για σύνθεση επιπλέον πληροφορίας, π.χ., αν η γεωγραφική θέση του χρήστη είναι μέσα σε ένα γραφείο και τα επίπεδα θορύβου σε αυτό το γραφείο είναι υψηλά, μπορεί ο χρήστης να είναι σε σύσκεψη.

Χρονική Εξάρτηση

Εκτός από το διαχωρισμό της πληροφορίας σε δυναμική και στατική, το *context* περιλαμβάνει και άλλες χρονικές εξαρτήσεις. Το τρέχον *context* μπορεί να εξαρτάται από γεγονότα του *παρελθόντος* ή να συνδέεται με γεγονότα του *μέλλοντος*. Η περιγραφή του *context* σε πολλές περιπτώσεις περιέχει καταστάσεις που ίσχυαν ή που θα ισχύσουν. Συνεπώς, ο χρόνος είναι ένα βασικό γνώρισμα που θα πρέπει να συμπεριληφθεί στο μοντέλο διαχείρισης του *context*. Είναι αξιοσημείωτο ότι, τα υπάρχοντα συστήματα που διαχειρίζονται *context* πληροφορία, δεν παρέχουν κάποια μέθοδο για τον χειρισμό των χρονικών εξαρτήσεων.

2.2 Context Aware (CA) Συστήματα

Για μια εφαρμογή, ενημερότητα για το *context* σημαίνει ότι μπορεί να κάνει χρήση της πληροφορίας του *context*. Μια εφαρμογή είναι *CA* αν μπορεί να αποκτήσει, να ερμηνεύσει, να χρησιμοποιήσει το *context* και να προσαρμόσει ανάλογα τη λειτουργία της. Η συμπεριφορά των *CA* εφαρμογών περιλαμβάνει δύο βασικά χαρακτηριστικά [4,8]:

- Οι υπηρεσίες της εφαρμογής παρέχονται στο χρήστη προσαρμοσμένες, ανάλογα με το τρέχον *context*.
- Συγκεκριμένο *context* μπορεί αυτόματα να οδηγήσει στην εκτέλεση κάποιων ενεργειών.

Η αυτόματη προσαρμογή της εφαρμογής, με οποιονδήποτε από τους δύο παραπάνω τρόπους, απαιτεί από το ενδιάμεσο λογισμικό να *αποφασίζει* κατάλληλες ενέργειες αντίδρασης. Οι ενέργειες αυτές πρέπει να εκτελεστούν πάνω στη συσκευή που λειτουργεί η εφαρμογή καθώς μπορεί να απαιτούν δεδομένα που βρίσκονται μόνο εκεί [9]. Η ανάπτυξη ενδιάμεσου λογισμικού που λειτουργεί πάνω σε κινητές συσκευές και παίρνει αποφάσεις είναι ιδιαίτερα δύσκολη, λόγω των περιορισμένων πόρων των συσκευών. Για το λόγο αυτό, όπως θα δούμε και στη συνέχεια, αρκετά συστήματα ενδιάμεσου λογισμικού στοχεύουν στη συλλογή, επεξεργασία και διανομή της πληροφορίας στις εφαρμογές. Η μετέπειτα χρήση της πληροφορίας υλοποιείται από το σχεδιαστή της εφαρμογής.

Τα *CA* συστήματα έχουν κάποια βασικά χαρακτηριστικά που σχετίζονται με τη διαχείριση του *context*. Τα χαρακτηριστικά αυτά περιλαμβάνουν [10]:

- Προμηθευτές *context*.
- Εξαγωγή σημασιολογικής πληροφορίας για το *context*.
- Προσαρμογή στο *context*.
- Δυνατότητα αποθήκευσης του *context* για μελλοντική χρήση.

Τελευταία επισημάνθηκε η ανάγκη για την προσθήκη ενός ακόμα χαρακτηριστικού στα *CA* συστήματα. Το χαρακτηριστικό αυτό αφορά την ικανότητα του συστήματος να εξάγει συμπεράσματα για την *αξιοπιστία* της πληροφορίας του *context* [11]. Στο θέμα της

αξιοπιστίας του *context* θα αναφερθούμε πιο αναλυτικά στην παράγραφο 3.4. Τα παραπάνω χαρακτηριστικά περιγράφονται αναλυτικότερα στη συνέχεια.

Προμηθευτές Context

Ένα *CA* σύστημα μπορεί να αποκτήσει *context* από διάφορες πηγές, όπως απευθείας από το χρήστη, βάσεις δεδομένων ή αισθητήρες. Οι αισθητήρες μπορεί να είναι συσκευές εξωτερικές στο περιβάλλον της εφαρμογής, ενσωματωμένες στη συσκευή που εκτελείται η εφαρμογή ή επιπρόσθετο λογισμικό. Όλες οι πηγές, από τις οποίες προέρχεται το *context* ονομάζονται γενικά *προμηθευτές context* [12].

Στη βιβλιογραφία επίσης χρησιμοποιείται και ο όρος *αισθητήρας* για όλους του τύπους προμηθευτών *context*, από απλές συσκευές υλικού μέχρι σύνθετες υπολογιστικές μονάδες με πολλαπλές ικανότητες [13]. Οι αισθητήρες μπορεί να χρησιμοποιηθούν ως απλές μηχανές συλλογής φυσικής πληροφορίας από το περιβάλλον ή ως πιο σύνθετοι μηχανισμοί εξαγωγής *context* από φυσικά δεδομένα.

Ο συνδυασμός φυσικής πληροφορίας για την εξαγωγή ανωτέρου επιπέδου *context*, ονομάζεται *εξαγωγή* ή *ανίχνευση* του *context*. Λιγότερο χρησιμοποιείται ο όρος πρόβλεψη, ο οποίος δεν θεωρείται δόκιμος. Στην αρχιτεκτονική ενός *CA* συστήματος, η σχεδίαση του υποσυστήματος διαχείρισης των προμηθευτών περιλαμβάνει τα εξής θέματα:

1. Επικοινωνία προμηθευτών εφαρμογής: Το *context* λαμβάνεται ύστερα από *ερώτηση* της εφαρμογής στον προμηθευτή ή με τη μέθοδο εγγραφής - ενημέρωσης, μέσω αποστολής *γεγονότων*.
2. Συχνότητα ενημέρωσης: Όταν η επικοινωνία βασίζεται σε γεγονότα, πρέπει να καθοριστεί η συχνότητα ενημέρωσης της εφαρμογής ή τα φίλτρα γεγονότων. Τα φίλτρα καθορίζουν ποια γεγονότα είναι «αρκετά σημαντικά», ώστε να προωθηθούν για επεξεργασία.
3. Εύρεση κατανεμημένων προμηθευτών, σε διάχυτο περιβάλλον: Σε διάχυτα περιβάλλοντα, οι προμηθευτές του *context* δεν είναι πάντα γνωστοί ή ορατοί στην εφαρμογή. Στην πρώτη περίπτωση η εφαρμογή γνωρίζει τους διαθέσιμους προμηθευτές, αλλά δεν ξέρει ποιος παρέχει την πληροφορία που επιθυμεί [14]. Στη δεύτερη περίπτωση η εφαρμογή αναζητά κατάλληλους προμηθευτές σε ένα

κατανεμημένο δίκτυο προμηθευτών [15]. Σε κάθε περίπτωση, η εφαρμογή πρέπει να περιλαμβάνει κατάλληλους μηχανισμούς εύρεσης και επικοινωνίας με τους προμηθευτές του *context*.

Εξαγωγή Σημασιολογικής Πληροφορίας για το Context

Η πληροφορία που συλλέγεται από τους προμηθευτές είναι συνήθως χαμηλού εννοιολογικού επιπέδου. Αφορά φυσικές αριθμητικές τιμές που προέρχονται από αισθητήρες, τεχνική ή ειδική πληροφορία, η οποία ακατέργαστη δεν είναι χρήσιμη από την εφαρμογή. Ένα *CA* σύστημα απαιτείται να εμπεριέχει μηχανισμούς ερμηνείας του *context*, μετατρέποντάς το σε μορφή που μπορεί να χρησιμοποιηθεί από την εφαρμογή.

Ακόμα και μεταφρασμένη η τιμή μίας παραμέτρου του *context* δεν είναι πάντα ιδιαίτερα χρήσιμη. Το ζητούμενο είναι ο *συνδυασμός του χαμηλού εννοιολογικού επιπέδου context και η εξαγωγή υψηλότερου επιπέδου σημασιολογικής πληροφορίας*. Το *context* υψηλότερου εννοιολογικού επιπέδου δεν μπορεί να αποκτηθεί άμεσα, αλλά συμπεραίνεται, συνδυάζοντας τη φυσική πληροφορία που παρέχουν οι αισθητήρες, μέσω κάποιας διαδικασίας εξαγωγής συμπερασμάτων, π.χ. αν τρία άτομα βρεθούν ταυτόχρονα στο ίδιο γραφείο μπορεί να σημαίνει ότι έχουν σύσκεψη. Οι διαδικασίες εξαγωγής συμπερασμάτων απαιτούν την ύπαρξη *ευφυίας* στο σύστημα. Η ευφυία ενσωματώνεται συνήθως με κανόνες δημιουργώντας ένα *έμπειρο σύστημα*.

Για τη διαχείριση του *context* απαιτείται ένα μοντέλο που ορίζει και αποθηκεύει το *context*, σε μία μορφή κατάλληλη για χρήση και επεξεργασία. Το μοντέλο θα πρέπει να είναι αρκετά γενικό ώστε να περιλαμβάνει όλα τα γνωρίσματα της πληροφορίας και να παρέχει μία ευέλικτη *μέθοδο αναπαράστασης*. Η μέθοδος αναπαράστασης είναι ιδιαίτερα σημαντική γιατί με βάση αυτή γίνεται όλη η διαχείριση της πληροφορίας.

Προσαρμογή στο Context

Η επεξεργασία του *context* μπορεί να έχει δύο αποτελέσματα:

1. Το *context* υψηλού εννοιολογικού επιπέδου παρέχεται στην εφαρμογή, ώστε να χρησιμοποιηθεί από τις υπηρεσίες της: π.χ. εμφανίζονται στο χρήστη πληροφορίες για την περιοχή που βρίσκεται.

2. Προσαρμόζεται αυτόματα η λειτουργία της εφαρμογής ανάλογα με το *context*. Η επιλογή των κατάλληλων ενεργειών προσαρμογής απαιτεί επιπλέον εμπειρία από το *CA* σύστημα. Οι ενέργειες αυτές μπορεί είτε να είναι φανερές στο χρήστη, είτε όχι.

Αποθήκευση του Context

Σε πολλές περιπτώσεις είναι απαραίτητο οι τιμές του *context* να αποθηκεύονται ώστε να χρησιμοποιηθούν από τις υπηρεσίες του συστήματος στο μέλλον. Επιπλέον οι παλιές τιμές του *context*, μπορεί να είναι χρήσιμες για την εξαγωγή συμπερασμάτων ή την αποτίμηση του τρέχοντος *context*. Παρόλα αυτά η αποθήκευση τιμών απαιτεί αρκετό αποθηκευτικό χώρο, ο οποίος πιθανώς να μην είναι διαθέσιμος σε κινητές συσκευές.

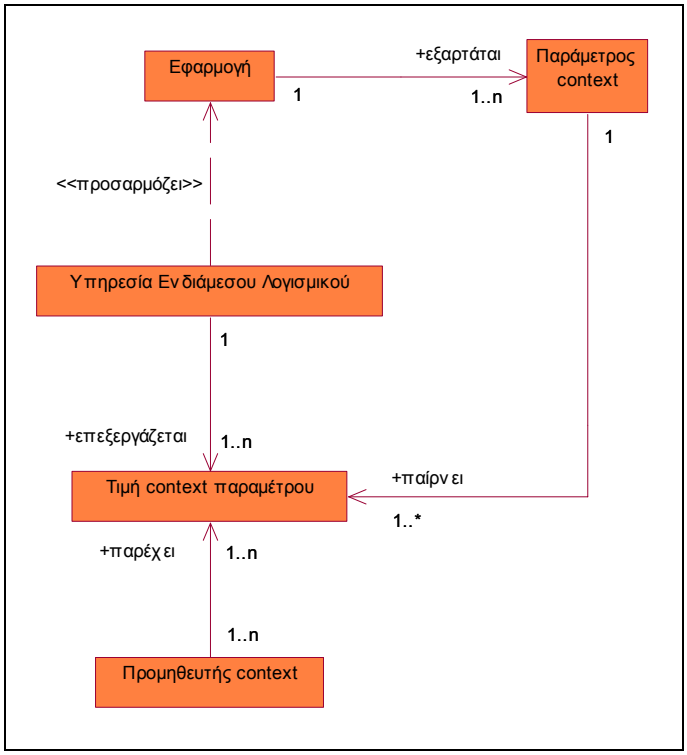
2.3 Ενδιάμεσο Λογισμικό

Η πληροφορία του *context* έχει κάποια ιδιαίτερα χαρακτηριστικά που κάνουν δύσκολο το χειρισμό της [7]. Το αποτέλεσμα είναι η επεξεργασία και η διαχείριση του *context* να αποτελούν σύνθετες διαδικασίες, που είναι δύσκολο να υλοποιηθούν σε ένα υπολογιστικό σύστημα. Επιπλέον απαιτείται ένα σύνολο υποσυστημάτων ανεξάρτητων της εφαρμογής που υλοποιούν τις διαδικασίες διαχείρισης του *context*. Η ανάπτυξη αυτών των υποσυστημάτων για κάθε εφαρμογή είναι μία δύσκολη και χρονοβόρα διαδικασία.

Αυτά τα προβλήματα στην ανάπτυξη *CA* συστημάτων οδήγησαν τους σχεδιαστές *CA* εφαρμογών σε μια αρχιτεκτονική προσέγγιση όπου, η διαχείριση του *context* γίνονταν ανεξάρτητα από την εφαρμογή ως *ενδιάμεσο λογισμικό* (*middleware*) [16]. Το ενδιάμεσο λογισμικό αναλαμβάνει όλη την επεξεργασία και διαχείριση του *context*, παρέχοντας στην εφαρμογή έτοιμες υπηρεσίες και εργαλεία προσαρμογής (Σχήμα 2.1).

Το ενδιάμεσο λογισμικό δεν είναι καινούργια ιδέα και έχει ερευνηθεί ιδιαίτερα στο χώρο της τεχνολογίας λογισμικού και των κατανεμημένων συστημάτων. Η χρήση του παρέχει γνωστά πλεονεκτήματα όπως, επαναχρησιμοποίηση λογισμικού, επεκτασιμότητα και διαλειτουργικότητα. Το ενδιάμεσο λογισμικό για *CA* εφαρμογές περιλαμβάνει ένα σύνολο έτοιμων και διαμορφώσιμων υπηρεσιών ή εργαλείων, που μπορεί να χρησιμοποιήσει ο προγραμματιστής για την υλοποίηση της εφαρμογής. Η εφαρμογή χτίζεται πάνω στο ενδιάμεσο λογισμικό, απαλλάσσοντας τον προγραμματιστή από την υλοποίηση λειτουργιών

διαχείρισης του *context*. Το ενδιάμεσο λογισμικό που διαχειρίζεται το *context* και προσδίδει *CA* χαρακτηριστικά στην εφαρμογή ονομάζεται *Context Aware Middleware (CAM)*.



Σχήμα 2.1 Context – Aware (CA) Σύστημα με Χρήση Ενδιάμεσου Λογισμικού.

2.4 Context Aware Middleware (CAM)

Οι παραδοσιακές πλατφόρμες ενδιάμεσου λογισμικού, όπως *CORBA* [L1], *J2EE* [L2], *DCOM* [L3], έλυναν προβλήματα όπως η κλιμάκωση και η ετερογένεια και παρείχαν διαφάνεια στην κατανομή και στο διαμοιρασμό των πόρων. Παρείχαν στους σχεδιαστές, ένα υψηλό επίπεδο αφαίρεσης κρύβοντας τις λεπτομέρειες της κατανομής στην ανάπτυξη του συστήματος. Σε αυτές τις τεχνολογίες το ενδιάμεσο λογισμικό κτίζονταν σαν ένα ξεχωριστό σύστημα που έπαιρνε είσοδο από την εφαρμογή και παρείχε διαφάνεια στους χρήστες και τους σχεδιαστές των εφαρμογών [17].

Οι τεχνολογίες αυτές έχουν χρησιμοποιηθεί με επιτυχία για στατικά κατανεμημένα συστήματα, σχεδιασμένα για ενσύρματα δίκτυα. Παρόλα αυτά δεν μπορούν να αντεπεξέλθουν επαρκώς στις απαιτήσεις των κινητών εφαρμογών, που μπορούν να παρέχουν υπηρεσίες οπουδήποτε, κάθε στιγμή. Αυτές οι τεχνολογίες υποθέτουν *μεγάλο διαθέσιμο*

εύρος ζώνης, καθώς και *σταθερή διαθεσιμότητα*. Σε κινητά συστήματα αυτές οι υποθέσεις δεν ισχύουν. Επιπλέον, αντικειμενοστρεφείς πλατφόρμες ενδιάμεσου λογισμικού όπως η *CORBA* [18], υποστηρίζουν κυρίως *σύγχρονη επικοινωνία*, ενώ σε κινητά περιβάλλοντα είναι πολύ συνηθισμένο ο εξυπηρέτης και ο πελάτης να μην είναι ταυτόχρονα συνδεδεμένοι. Το ενδιάμεσο λογισμικό που προορίζεται για *CA* εφαρμογές πρέπει να είναι σχεδιασμένο με βάση τα χαρακτηριστικά των διάχυτων συστημάτων και των συσκευών που τα συνθέτουν.

Η προσέγγιση ενός ενδιάμεσου λογισμικού με στατική συμπεριφορά δεν είναι η κατάλληλη για ένα *CAM* [19]. Το *CAM* πρέπει να ανιχνεύει τις αλλαγές που γίνονται στο περιβάλλον του και να προσαρμόζει τις υπηρεσίες του. Το ενδιάμεσο λογισμικό, λαμβάνοντας συγκεκριμένη πληροφορία από την εφαρμογή και το περιβάλλον, θα μπορούσε να πάρει πιο αποτελεσματικές αποφάσεις, παρέχοντας ποιοτικότερες υπηρεσίες.

Αυτό το επιχείρημα βασίζεται ουσιαστικά στην ιδιότητα της *ανάκλασης* [20]. Ένα σύστημα που έχει την ιδιότητα της ανάκλασης μπορεί να τροποποιήσει τη συμπεριφορά του μέσω *αυτοεξέτασης και/ή προσαρμογής*. Η αυτοεξέταση μπορεί να επιτευχθεί κάνοντας ορατά στο σύστημα κάποια στοιχεία της εσωτερικής του δομής, ενώ η προσαρμογή επιτυγχάνεται τροποποιώντας τα στοιχεία αυτά. Η αρχή της ανάκλασης δεν είναι καινούργια και έχει ερευνηθεί αρκετά στις γλώσσες προγραμματισμού, αλλά και σε συστήματα ενδιάμεσου λογισμικού [21-23].

Η δυνατότητα προσαρμογής μέσω ανάκλασης είναι ένα ακόμα χαρακτηριστικό που πρέπει να έχει ένα *CAM* [24-26]. Ένα ανακλαστικό *CAM* δίνει τη δυνατότητα στην εφαρμογή να ελέγχει ένα μέρος της εσωτερικής του δομής που την ενδιαφέρει (*reification*). Επιπλέον η εφαρμογή μπορεί να τροποποιήσει την εσωτερική δομή του *CAM* σύμφωνα με τις ανάγκες της (*absorption*). Για παράδειγμα, η τροποποίηση μπορεί να αφορά το σύνολο των πόρων που παρακολουθούνται ή τους κανόνες προσαρμογής στο *context*.

Οι υπηρεσίες διαχείρισης ενός *CAM* μπορεί να παρέχονται κεντρικά από κάποια συσκευή με αυξημένες υπολογιστικές δυνατότητες ή αποκεντρικοποιημένα, από το ενδιάμεσο λογισμικό, που βρίσκεται πάνω στη κινητή συσκευή. Στην πρώτη περίπτωση οι σύνθετες διαδικασίες επιτελούνται σε κάποιους στατικούς εξυπηρέτες, οι οποίοι παρέχουν υπηρεσίες διαχείρισης του *context* στους κινητούς πελάτες, όπου τρέχουν οι εφαρμογές. Στη δεύτερη

περίπτωση όλη η διαχείριση και επεξεργασία του *context* γίνεται από το ενδιάμεσο λογισμικό που βρίσκεται πάνω στην κινητή συσκευή.

Η κεντρικοποιημένη αρχιτεκτονική είναι πιο εύκολη στην υλοποίηση, καθώς το ενδιάμεσο λογισμικό δεν εξαρτάται από τους υπολογιστικούς πόρους της συσκευής για την οποία προορίζεται. Αυτή η προσέγγιση όμως, έχει περιορισμένες δυνατότητες κλιμάκωσης και δεν μπορεί να εφαρμοστεί σε διάχυτα δίκτυα μεγάλης γεωγραφικής έκτασης. Επιπλέον δεν είναι ανεκτική σε σφάλματα καθώς εξαρτάται από ένα κεντρικό συστατικό.

Τα *CAMs* με κεντρικοποιημένη αρχιτεκτονική σχεδιάζονται κυρίως για τη δημιουργία *«ευφυών χώρων»* (*intelligent spaces*) ή *«ενεργών χώρων»* (*active spaces*). Οι *φυσικοί χώροι* είναι περιορισμένες γεωγραφικά περιοχές, όπως δωμάτια, γραφεία, εργαστήρια, οχήματα κ.α., οι οποίοι περιλαμβάνουν φυσικά αντικείμενα, ετερογενείς διασυνδεδεμένες συσκευές και χρήστες που επιτελούν ένα σύνολο δραστηριοτήτων. Οι *ευφυείς χώροι* ή *ενεργοί χώροι* είναι φυσικοί χώροι, που περιλαμβάνουν μία υποδομή *CA* λογισμικού που συντονίζει την αλληλεπίδραση των χρηστών με το περιβάλλον τους [27].

ΚΕΦΑΛΑΙΟ 3. ΑΝΑΣΚΟΠΗΣΗ ΒΙΒΛΙΟΓΡΑΦΙΑΣ

Στη δεκαετία του '*90*, η έρευνα στόχευε κυρίως σε *CA εφαρμογές* για συγκεκριμένο σκοπό. Στη συνέχεια έγινε μία μετατόπιση της έρευνας, προς το πρόβλημα της σχεδίασης και ανάπτυξης ενδιάμεσου λογισμικού για την υποστήριξη *CA* εφαρμογών. Σήμερα η διαχείριση του *context* και οι διαδικασίες που τη συνθέτουν, αποτελούν θέμα πολλών εργασιών.

3.1 CA Εφαρμογές

Ο *Schilit* [5] όρισε τέσσερις κλάσεις *CA* εφαρμογών, με βάση τη χρήση του *context*:

1. *Πλησιέστερης επιλογής*. Πρόκειται για τεχνική γραφικής διεπαφής με το χρήστη. Τα αντικείμενα που βρίσκονται σε κοντινότερη απόσταση από το χρήστη εμφανίζονται με έμφαση (φωτεινά ή με έντονα χρώματα) ή με τρόπο ώστε να επιλέγονται ευκολότερα.
2. *Αυτόματης αναδιαμόρφωσης*. Περιλαμβάνουν την πρόσθεση νέων τμημάτων του συστήματος, αφαίρεση παλαιών, ή αλλαγή των συνδέσεων μεταξύ των υπαρχόντων, λόγω αλλαγής στο *context*.
3. *Λειτουργίας με βάση το context*. Εντολές που μπορεί να παράγουν διαφορετικά αποτελέσματα σε διαφορετικό *context*. Για παράδειγμα αν η εντολή *print* τύπωνε πάντα στον πλησιέστερο εκτυπωτή, θα είχε διαφορετικά αποτελέσματα η εκτέλεσή της, ανάλογα με τη θέση του χρήστη.

4. *Ενεργοποίησης λειτουργιών από τις αλλαγές του context*. Απλοί κανόνες συνθήκης-ενέργειας, που χρησιμοποιούνται για να καθορίσουν, πως θα πρέπει η εφαρμογή να προσαρμοστεί στις αλλαγές του *context*.

Οι *CA* εφαρμογές, που έχουν παρουσιαστεί, αποτελούν κυρίως κινητές εφαρμογές, αφού αυτές παρουσιάζουν το μεγαλύτερο ενδιαφέρον. Οι εφαρμογές αυτές περιλαμβάνουν βοηθούς γραφείου, ηλεκτρονικά ημερολόγια, τουριστικούς οδηγούς και εφαρμογές υπενθύμισης. Αναφέρουμε ενδεικτικά τις πιο χαρακτηριστικές και αναλύουμε περισσότερο τις σημαντικότερες από αυτές.

Από τις πρώτες κινητές *CA* εφαρμογές που εμφανίστηκαν ήταν το σύστημα *Active Badge* [28]. Με το σύστημα αυτό οι τηλεφωνικές κλήσεις σε ένα εσωτερικό τηλεφωνικό δίκτυο μπορούν να προωθηθούν στο τηλέφωνο που βρίσκεται πλησιέστερα στον παραλήπτη της κλήσης.

Ένα πολύ γνωστό σύστημα που αναπτύχθηκε στις αρχές τις δεκαετίας του *'90* ήταν το *ParcTab* [29]. Το σύστημα βασίζεται σε έναν φορητό υπολογιστή που επικοινωνεί ασύρματα με έναν κεντρικό εξυπηρέτη. Το *ParcTab* λειτουργεί σαν προσωπικός ηλεκτρονικός βοηθός γραφείου και έχουν αναπτυχθεί διάφορες *CA* εφαρμογές για να ελεγχθεί η λειτουργικότητά του. Οι πιο χαρακτηριστικές από αυτές περιλαμβάνουν:

- Παρουσίαση πληροφοριών στην οθόνη της συσκευής σχετικά με το δωμάτιο που βρίσκεται ο χρήστης [30].
- Υπόδειξη στο χρήστη για τον πιο κοντινό τοπικό πόρο, π.χ. τον πιο κοντινό εκτυπωτή σε μια εντολή εκτύπωσης [30].
- Σύνδεση ενός συγκεκριμένου καταλόγου του *UNIX* με ένα δωμάτιο. Όταν ο χρήστης μπαίνει στο συγκεκριμένο δωμάτιο μπορεί να δει όλα τα περιεχόμενα του καταλόγου. Στον κατάλογο μπορεί να γράψει οποιοσδήποτε βρίσκεται μέσα στο δωμάτιο. Έτσι κάποιος μπορεί να βγάλει μια ανακοίνωση για όσους βρίσκονται σε εκείνο το δωμάτιο δημιουργώντας ένα αρχείο στον προσαρτημένο κατάλογο [30].
- Εντοπισμός άλλων ατόμων που μεταφέρουν ένα *ParcTab*. Ο εντοπισμός γίνεται με την εμφάνιση ενός χάρτη στην οθόνη της συσκευής [31].

- Χρήση του *ParcTab* ως τηλεχειριστήριο εξ αποστάσεως με διαφορετικές δυνατότητες τηλεχειρισμού σε διαφορετικά δωμάτια.

Σημαντική έρευνα για *CA* συστήματα έχει γίνει από το Ινστιτούτο Τεχνολογίας της *Georgia* (*GeorgiaTech*) [L5]. Μία από τις εφαρμογές που αναπτύχθηκαν εκεί ήταν το *In/Out board* [32]. Πρόκειται για μία εφαρμογή γραφείου σε *Java*, η οποία έδειχνε πότε κάποιος βρίσκονταν σε ένα γραφείο. Συγκεκριμένα παρουσίαζε την ακριβή ώρα που κάποιος μπήκε ή βγήκε από ένα γραφείο. Η πληροφορία *context* που χρησιμοποιούσε ήταν η ώρα και η ταυτότητα των χρηστών.

Μια άλλη ενδιαφέρουσα εφαρμογή από την ίδια ομάδα ήταν το *DUMMBO* (*Dynamic Ubiquitous Mobile Meeting Board*) [32]. Το *DUMMBO* είναι μια εφαρμογή που αναγνωρίζει και διαχειρίζεται τις ανεπίσημες και ακανόνιστες συσκέψεις. Το σύστημα περιλαμβάνει έναν λευκό πίνακα μαρκαδόρου, ο οποίος πλαισιώνεται από ψηφιακούς αισθητήρες. Η αναγνώριση μιας σύσκεψης γίνεται όταν γράφονται ή σβήνονται δεδομένα από τον πίνακα ανιχνεύοντας το μελάνι πάνω στον πίνακα και μέσω ψηφιακής καταγραφής συνομιλιών. Η εφαρμογή περιλαμβάνει αυτόματη έναρξη καταγραφής συνομιλιών όταν δύο ή παραπάνω άτομα βρίσκονται γύρω από έναν πίνακα. Το υλικό που καταγράφεται κατά τη διάρκεια της σύσκεψης είναι στη συνέχεια διαθέσιμο μαζί με πληροφορίες για τα άτομα που παρευρίσκονταν, την ώρα και στην τοποθεσία της σύσκεψης.

Το C*onference Assistant* [33] είναι μια εφαρμογή που αποσκοπεί στην εξυπηρέτηση των μελών ενός συνεδρίου. Η εφαρμογή είναι σχεδιασμένη για διαφορετικές συσκευές και πλατφόρμες λειτουργικών συστημάτων, όπως φορητούς υπολογιστές με *Windows 95/98* και *Hewlett Packard 620LX WinCe*. Οι σύνεδροι όταν δηλώνουν συμμετοχή, δίνουν τα προσωπικά τους στοιχεία και παίρνουν την εφαρμογή, που μπορούν να εγκαταστήσουν και να τρέξουν στην προσωπική τους φορητή συσκευή. Ο C*onference Assistant* παρουσιάζει το πρόγραμμα του συνεδρίου με τις ομιλίες που ταιριάζουν με τα ενδιαφέροντα του χρήστη τονισμένες. Όταν ένας σύνεδρος μπαίνει σε ένα δωμάτιο που γίνεται παρουσίαση, εμφανίζονται πληροφορίες σχετικά με το άτομο που μιλάει και παρουσιάζεται στην οθόνη το υλικό της παρουσίασης. Ο σύνεδρος έχει πρόσβαση στο υλικό της παρουσίασης και μπορεί να το τροποποιήσει και να το αποθηκεύσει κρατώντας σημειώσεις. Η πληροφορία του

context που διαχειρίζεται η εφαρμογή είναι η τοποθεσία, η ώρα, η δραστηριότητα του δωματίου και τα ενδιαφέροντα των χρηστών.

Τα συστήματα που συλλέγουν πληροφορία σχετικά με τη γεωγραφική θέση που κινείται ο χρήστης αποτελούν τον πιο δημοφιλή τύπο *CA* εφαρμογών. Αρκετές τέτοιες εφαρμογές είχαν αναπτυχθεί για το *ParcTab* και στο πλαίσιο άλλων ερευνητικών προγραμμάτων στις αρχές της δεκαετίας του *'90*. Χαρακτηριστικό παράδειγμα ήταν το πρόγραμμα *Chameleon* του Πανεπιστημίου του *Toronto*. Το πρόγραμμα αυτό ερευνούσε πως οι φορητές συσκευές μπορούν να είναι ενήμερες της γεωγραφικής τους θέσης και κατεύθυνσης και να παρέχουν πληροφορίες, σχετικά με φυσικά αντικείμενα στο περιβάλλον τους [34].

Στην κατηγορία των εφαρμογών που εξαρτώνται από τη γεωγραφική θέση του χρήστη ανήκουν και οι οδηγοί (τουριστικοί, αναζήτησης κ.α.) που περιλαμβάνουν και αρκετές εμπορικές εφαρμογές. Ακόμα και τα αυτοκίνητα τα οποία χρησιμοποιούν *GPS* μπορούν να θεωρηθούν κινητές *CA* εφαρμογές. Επιπλέον αρκετές εταιρείες κινητής τηλεφωνίας παρέχουν υπηρεσίες πληροφόρησης βασισμένες στη θέση του χρήστη.

Χαρακτηριστικά παραδείγματα τέτοιων εφαρμογών αναπτύχθηκαν στο πλαίσιο του προγράμματος *Cyberguide* [35]. Ο σκοπός αυτών των εφαρμογών ήταν να εμφανίζονται στην οθόνη της φορητής συσκευής πληροφορίες σχετικές με αξιοθέατα στους τουρίστες, με βάση τη θέση και την κατεύθυνσή τους.

Η εφαρμογή *GUIDE* [36] σχεδιάστηκε ώστε να εξυπηρετεί τους τουρίστες των αρχαιολογικών χώρων του *Lancaster*. Η εφαρμογή είναι τύπου πελάτη - εξυπηρέτη και αποτελείται από έναν πελάτη *Java browser*, ο οποίος παρέχει πληροφορίες σε ιστοσελίδες μέσω *http* στον εξυπηρέτη. Παρόμοια εφαρμογή ήταν και ο τουριστικός οδηγός *SmartSight* που αναπτύχθηκε από το πανεπιστήμιο του *Carnegie Mellon* και διαχειριζόταν πληροφορία μόνο για τη θέση του χρήστη [37].

Μια άλλη σημαντική κατηγορία *CA* εφαρμογών ήταν οι εφαρμογές που λειτουργούσαν σαν βοηθοί υπενθύμισης για τους χρήστες. Στη δεκαετία του *΄90* έγιναν πολλά πειράματα, ώστε να αναπτυχθούν εφαρμογές οι οποίες έκαναν υπενθυμίσεις,

χρησιμοποιώντας το τρέχον *context* ή συνέλεγαν πληροφορία ώστε να την εμφανίσουν στο κατάλληλο *context* στο μέλλον.

Από τα πρώτα συστήματα της παραπάνω κατηγορίας ήταν το *Forgot-Me-Not* [38]. Η εφαρμογή τρέχει σε μια συσκευή *ParceTab* και καταγράφει πληροφορίες όπως, που βρίσκεται ο χρήστης, με ποιους είναι μαζί, με ποιον συνομίλησε τηλεφωνικά και άλλες πληροφορίες και τις αποθηκεύει σε μια βάση δεδομένων για μελλοντική ανάκτηση.

Το *1996* στο *MIT* αναπτύχθηκε ο *Remembrance Agent* [39]. Αυτή η εφαρμογή περιλαμβάνει έναν πράκτορα, που τρέχει στο παρασκήνιο και κάνει υπενθυμίσεις με βάση το φυσικό *context* της φορέσιμης συσκευής στην οποία λειτουργεί. Οι υπενθυμίσεις περιλαμβάνουν περίληψη παλιών σημειώσεων, παλιά *e-mails*, επιστημονικά άρθρα και άλλες πληροφορίες που μπορεί να είναι σχετικές με το τρέχον *context* του χρήστη. Η εφαρμογή χρησιμοποιεί πέντε είδη πληροφορίας *context*: τη φυσική θέση του χρήστη που φοράει τη συσκευή, τα άτομα που βρίσκονται γύρω του, την ημερομηνία, την ώρα, μια χρονική σφραγίδα και το κείμενο των αποθηκευμένων εγγράφων.

Το *StartleCam* [40] ήταν άλλη μία *CA* εφαρμογή που αναπτύχθηκε από το *MIT* και βασίζεται σε φορέσιμη συσκευή. Το σύστημα αποτελείται από μια *video* κάμερα, ένα φορητό υπολογιστή και έναν αισθητήρα που λαμβάνει σήματα από το δέρμα. Η κάμερα καταγράφει συνεχώς και ο υπολογιστής αποθηκεύει τις εικόνες, όταν ανιχνεύει ότι ο χρήστης ενδιαφέρεται για αυτές. Η ανίχνευση γίνεται μετρώντας σήματα διέγερσης από το δέρμα. Η κάμερα αποθηκεύει συνεχώς εικόνες στη μνήμη του υπολογιστή. Όταν ο αισθητήρας ανιχνεύει αυξημένη διέγερση στέλνει ασύρματα όλες τις εικόνες από τη μνήμη σε έναν απομακρυσμένο εξυπηρέτη.

Το *Fieldwork* [8,10] είναι μια εφαρμογή σχεδιασμένη για τη βοήθεια όσων εργάζονται σε εξωτερικούς χώρους και κάνουν επιστημονικές παρατηρήσεις ή συλλέγουν δεδομένα. Περιλαμβάνει ένα σύνολο εργαλείων, που παρέχουν αυτόματη συλλογή δεδομένων, όπως θέση και τα οποία σχετίζονται με τις παρατηρήσεις του ερευνητή. Το *context* της εφαρμογής είναι η θέση και ο χρόνος.

Μια εφαρμογή με κωδικό όνομα *"Adaptive GSM phone and PDA"*, παρουσιάστηκε από την ομάδα *TEA* (*Technology for Enabling Awareness*) [L5] από το *Startlab* [41]. Πρόκειται για μια εφαρμογή για κινητές συσκευές, σχεδιασμένη ώστε να μπορεί να προσαρμόζεται στην κατάσταση του χρήστη. Το μέγεθος των γραμμάτων της οθόνης μεγαλώνει όταν ο χρήστης κινείται, ενώ μικραίνει όταν είναι σε σταθερή θέση. Επιπλέον, η εφαρμογή επιλέγει ηχητική ειδοποίηση, δόνηση, ρύθμιση της έντασης του ήχου ή σιωπηλή ειδοποίηση, ανάλογα με το αν η συσκευή ήταν στο χέρι του χρήστη, σε τραπέζι, σε βαλίτσα ή σε εξωτερικό χώρο. Το *context* για αυτή την εφαρμογή αποτελείται από τη δραστηριότητα του χρήστη, το επίπεδο φωτεινότητας του περιβάλλοντος, την πίεση της συσκευής και το αν βρίσκονται άλλα άτομα κοντά στο χρήστη.

Η εφαρμογή *Office Assistant* [42] αναπτύχθηκε από το *MIT* και διαχειρίζεται τις δραστηριότητες ενός γραφείου. Ένας πράκτορας αλληλεπιδρά με τους επισκέπτες και διαχειρίζεται το πρόγραμμα συναντήσεων του γραφείου. Δύο χαλιά δαπέδου, με αισθητήρες πίεσης είναι τοποθετημένα στα άκρα της πόρτας του γραφείου. Όταν οι αισθητήρες ανιχνεύσουν την παρουσία επισκέπτη ενεργοποιούν τον πράκτορα. Ο πράκτορας ανιχνεύει πληροφορία *context* όπως, η ταυτότητα του επισκέπτη, το πρόγραμμα του ιδιοκτήτη του γραφείου, αν είναι απασχολημένος και την επιθυμία του να δει τον επισκέπτη. Αναλόγως προσαρμόζει τη συμπεριφορά του αλληλεπιδρώντας με τον επισκέπτη.

Οι περισσότερες από τις υπάρχουσες εφαρμογές χρησιμοποιούν περιορισμένου τύπου πληροφορία για το *context*, όπως τοποθεσία, ώρα, ημερομηνία, ταυτότητα οντότητας και στατική πληροφορία. Ο λόγος για τη χρήση περιορισμένης πληροφορίας *context* είναι κυρίως η δυσκολία που έχουν τα συστήματα λογισμικού στη συλλογή και επεξεργασία του *context*. Οι περισσότερες εφαρμογές έχουν αναπτυχθεί για ακαδημαϊκούς σκοπούς σε ερευνητικά εργαστήρια. Στο εμπόριο υπάρχουν ελάχιστες εφαρμογές παρέχοντας υπηρεσίες με βάση τη θέση.

Η περιορισμένη διαθεσιμότητα τέτοιων εφαρμογών στο εμπόριο οφείλεται, εκτός από τη δυσκολία που αναφέρθηκε παραπάνω και στην πολυπλοκότητα που περιλαμβάνει η υλοποίηση τους. Συνήθως απαιτούν, αρκετούς διαθέσιμους υπολογιστικούς πόρους αφού χρειάζεται συνεχή παρακολούθηση των αισθητήρων ή σύνθετοι υπολογισμοί για την εξαγωγή συμπερασμάτων. Οι υπολογιστικοί πόροι είναι ιδιαίτερα περιορισμένοι στις φορητές

συσκευές, όπου προορίζονται οι *CA εφαρμογές* και αυτό αποτελεί τροχοπέδη, για τους σχεδιαστές των εφαρμογών.

3.2 Λογισμικό για Υποστήριξη CA Εφαρμογών

Οι εργασίες που αναφέρθηκαν έως τώρα περιγράφουν συγκεκριμένες εφαρμογές, οι οποίες κάνουν χρήση καθορισμένου *context*, για να παρέχουν αυτοματοποιημένα συγκεκριμένες υπηρεσίες στους χρήστες. Όπως αναφέρθηκε και προηγουμένως διαδικασίες όπως η συλλογή, η επεξεργασία και η διαχείριση του *context* είναι ιδιαίτερα σύνθετες. Για το λόγο αυτό είναι επιθυμητό, η υλοποίηση των παραπάνω λειτουργιών να γίνεται ανεξάρτητα από την εφαρμογή. Ένα επιπλέον ζητούμενο είναι να υπάρχει διαλειτουργικότητα στις *CA* εφαρμογές. Αυτή η ανάγκη δημιουργεί την απαίτηση για ένα κοινό λογισμικό που βοηθάει στην ανάπτυξη και υποστηρίζει *CA* εφαρμογές.

3.2.1 Κατηγορίες Ενδιάμεσου Λογισμικού

Το λογισμικό υποστήριξης μπορεί να διακριθεί στις εξής κατηγορίες [43]:

- Βιβλιοθήκες
- Πλαίσια
- Εργαλεία
- Υποδομές

Βιβλιοθήκη είναι ένα γενικευμένο σύνολο από υλοποιημένους αλγορίθμους, σχετικούς με κάποιο θέμα. Οι βιβλιοθήκες στοχεύουν αποκλειστικά στην επαναχρησιμοποίηση κώδικα.

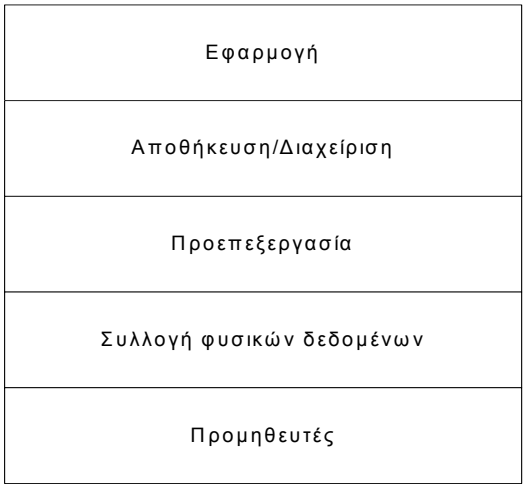
Τα *πλαίσια* στοχεύουν κυρίως στην επαναχρησιμοποίηση αρχιτεκτονικού σχεδιασμού. Παρέχουν μια βασική αρχιτεκτονική δομή, για την ανάπτυξη συγκεκριμένης κατηγορίας εφαρμογών. Επιπλέον ένα πλαίσιο παρέχει τρόπους για προσαρμογή της εφαρμογής σύμφωνα με τις ανάγκες ή προτιμήσεις του σχεδιαστή.

Τα *εργαλεία* υλοποιούνται πάνω σε κάποιο πλαίσιο παρέχοντας, ένα σύνολο από επαναχρησιμοποιήσιμα συστατικά, τα οποία προσθέτουν επιπλέον λειτουργικότητα.

Η *υποδομή* είναι ένα σύνολο από αξιόπιστες και προσβάσιμες τεχνολογίες που λειτουργούν σαν βάση για την ανάπτυξη άλλων συστημάτων. Οι υποδομές παρέχουν σχήματα, πρωτόκολλα και άλλα *standards*, τα οποία μπορεί να χρησιμοποιήσει ένα σύστημα που θα χτιστεί πάνω στην υποδομή. Χαρακτηριστικά παραδείγματα υποδομών αποτελούν οι πλατφόρμες ενδιάμεσου λογισμικού *J2EE*, *CORBA* και *DCOM*.

3.2.2 Χαρακτηριστικά Ενός CAM

Τα συστήματα ενδιάμεσου λογισμικού που έχουν προταθεί μέχρι σήμερα εμφανίζουν κάποια κοινά λειτουργικά χαρακτηριστικά. Στο Σχήμα 3.1 φαίνεται αφαιρετικά η αρχιτεκτονική ενός *CAM*. Η αρχιτεκτονική κατανέμει σε τέσσερα επίπεδα τη λειτουργικότητα ενός *CAM*, ενώ στο πέμπτο επίπεδο βρίσκεται η εφαρμογή [44].



Σχήμα 3.1 Αφαιρετική Αρχιτεκτονική Ενός Context Aware συστήματος.

Το πρώτο επίπεδο αποτελείται από τους προμηθευτές. Σύμφωνα με την κατηγοριοποίηση που έγινε παραπάνω, οι προμηθευτές μπορεί να είναι οποιαδήποτε πηγή υλικού ή λογισμικού που παρέχει τιμές για τις παραμέτρους του *context*.

Στο δεύτερο επίπεδο γίνεται η συλλογή της φυσικής πληροφορίας. Στο επίπεδο αυτό χρησιμοποιούνται οδηγοί για την επικοινωνία με προμηθευτές υλικού και *APIs* για την επικοινωνία με το λογισμικό. Το λογισμικό απόκτησης της πληροφορίας υλοποιείται με

επαναχρησιμοποιήσιμα τμήματα, που κρύβουν τα τεχνικά χαρακτηριστικά του προμηθευτή και αποκτούν την πληροφορία με διαφάνεια προς τα ανώτερα επίπεδα.

Στο τρίτο επίπεδο γίνεται η ερμηνεία των τιμών που συλλέχθηκαν και η εξαγωγή ανώτερου εννοιολογικά *context*. Το επίπεδο αυτό δεν είναι απαραίτητο για ένα *CA* σύστημα, αλλά είναι ιδιαίτερα χρήσιμο αν η εφαρμογή απαιτεί υψηλού εννοιολογικού επιπέδου πληροφορία. Η πληροφορία από διαφορετικούς προμηθευτές συνδυάζεται και προκύπτει υψηλότερο εννοιολογικά *context*. Επίσης σε αυτό το επίπεδο διευθετούνται προβλήματα που σχετίζονται με την ασάφεια και την ανακρίβεια της πληροφορίας που παράγουν οι προμηθευτές.

Στο τέταρτο επίπεδο γίνεται η μοντελοποίηση και αποθήκευση του *context*. Στο επίπεδο αυτό υλοποιείται μια διεπαφή για την παροχή της πληροφορίας στις εφαρμογές. Η πρόσβαση της εφαρμογής στα δεδομένα μπορεί να γίνει με δύο τρόπους: *σύγχρονα* και *ασύγχρονα*. Στη σύγχρονη επικοινωνία, η εφαρμογή ρωτάει το επίπεδο διαχείρισης για την πληροφορία που την ενδιαφέρει. Η ασύγχρονη επικοινωνία γίνεται μέσω *εγγραφών* και *ειδοποιήσεων*. Η εφαρμογή εγγράφεται στη λίστα ειδοποίησης για κάποια παράμετρο του *context* και όταν συμβεί κάποιο γεγονός ειδοποιείται λαμβάνοντας τη νέα τιμή. Στις περισσότερες περιπτώσεις η ασύγχρονη επικοινωνία είναι πιο κατάλληλη, λόγω των συχνών αλλαγών στις τιμές του *context*.

Στο πέμπτο επίπεδο βρίσκεται η εφαρμογή που χρησιμοποιεί το *context*. Στο επίπεδο αυτό υλοποιείται ο τρόπος που η εφαρμογή αντιδρά στις αλλαγές του *context* και η χρήση της πληροφορίας από τις υπηρεσίες της εφαρμογής.

Αρκετές εργασίες έχουν παρουσιαστεί προτείνοντας γενικά πλαίσια και υποδομές για την υποστήριξη *CA* εφαρμογών. Τα πλαίσια αυτά στην πλειοψηφία τους υιοθετούν τα επίπεδα λειτουργιών που περιγράφηκαν παραπάνω. Σε κάποιες εργασίες προτείνονται μόνο αρχιτεκτονικά πλαίσια χωρίς να παρουσιάζεται κάποια υλοποίηση πάνω σε αυτά.

Η πρώτη σημαντική προσέγγιση προς αυτή τη κατεύθυνση, ήταν από τον *W. Schilit* [45]. Ο *Schilit* στη διδακτορική του διατριβή πρότεινε ένα γενικό πλαίσιο για *CA*

εφαρμογές, επικεντρώνοντας κυρίως σε *context* που αφορούσε τη γεωγραφική θέση του χρήστη.

Ο *B. Schilit* [46] πρότεινε ένα πλαίσιο με κεντρικοποιημένη αρχιτεκτονική. Το πλαίσιο όριζε ένα *περιβάλλον* σαν ένα σύνολο από *ονόματα παραμέτρων* και *τιμών* για τις παραμέτρους. Στατικοί εξυπηρέτες διαχειρίζονταν το περιβάλλον και μετέφεραν την πληροφορία, στους πελάτες (εφαρμογές) που είχαν ενδιαφερθεί πιο πριν κάνοντας εγγραφή στους εξυπηρέτες. Τυπικά αντιστοιχούσε ένα περιβάλλον ανά χρήστη και επιπρόσθετα περιβάλλοντα για συγκεκριμένες οντότητες όπως δωμάτια, ομάδες ατόμων κ.α.

Παρόμοιες προσεγγίσεις με αυτήν του *B. Schilit* προτάθηκαν και στις εργασίες [47] και [10]. Προτάθηκαν πλαίσια που δρουν, σαν ενδιάμεσο λογισμικό που συλλέγει το *context* από τους αισθητήρες και παρέχει μεταφρασμένη την πληροφορία στην εφαρμογή, μέσω ενός συγκεκριμένου *API*.

Ένα άλλο γενικό πλαίσιο, προέκυψε από τις εφαρμογές *Stick-e Notes*. Η ιδέα των εφαρμογών αυτών ήταν, η σύνδεση σημειώσεων με διαφορετικά *context* (π.χ. τοποθεσία, ώρα, μέρα, θερμοκρασία κ.α.) και η αυτόματη εμφάνιση των σημειώσεων κάθε φορά που ίσχυε το αντίστοιχο *context*. Για παράδειγμα ο χρήστης μπορεί να συνδέσει εικονικά ένα αρχείο με μία τοποθεσία και κάθε φορά που θα περνάει από τη συγκεκριμένη τοποθεσία το αρχείο θα εμφανίζεται στην οθόνη του [8,48].

Ο *Brown* [30] εμπνευσμένος από τις εφαρμογές *Stick-e Notes* πρότεινε ένα πλαίσιο ανάπτυξης *CA εφαρμογών* αποτελούμενο από τρεις βασικές αρχιτεκτονικές μονάδες:

- Η μονάδα ενεργοποίησης της σημείωσης, που ταιριάζει το τρέχον *context* του χρήστη με το *context* με το οποίο έχει γίνει η διασύνδεση της σημείωσης.
- Η μονάδα εκτέλεσης. Μπορεί να είναι οποιοδήποτε πρόγραμμα εκτελεί την εμφάνιση της σημείωσης.
- Ένα σύνολο από αισθητήρες οι οποίοι παρέχουν τις πληροφορίες του *context* και κρύβουν από το υπόλοιπο σύστημα τις λεπτομέρειες της λειτουργίας τους.

Ο *Pascoe* [49] βασισμένος στις ιδέες του *Brown* πρότεινε μια αντικειμενοστρεφή αρχιτεκτονική. Το πλαίσιο αναπτύχθηκε σε *C++* και το *1997* παρουσιάστηκε η υλοποίηση μιας *Stick-e Note* εφαρμογής για φορητούς υπολογιστές.

Η *Hewlett Packard* πρότεινε ένα πλαίσιο με το όνομα "*An environment for situational Computing*" [47]. Η ιδέα βασίζεται στην ύπαρξη ενός εξυπηρέτη για την συλλογή και επεξεργασία του *context*. Ο εξυπηρέτης παρέχει ένα σύνολο υπηρεσιών, για την μετατροπή των φυσικών δεδομένων που έδιναν οι αισθητήρες, σε γεγονότα *context*. Οι υπηρεσίες αυτές, είναι υπεύθυνες για τη συλλογή της πληροφορίας από τους αισθητήρες και την παροχή της σε κατανοητή και χρήσιμη μορφή στην εφαρμογή.

Θεμελιώδης για την έρευνα στα *CA* συστήματα θεωρείται η εργασία του *Dey* από το *GeorgiaTech*. Ο *Dey* στο πλαίσιο της διδακτορικής του διατριβής πρότεινε μια γενική αρχιτεκτονική για την υποστήριξη και ανάπτυξη *CA* εφαρμογών. Υλοποίησε μια υποδομή βασισμένη σε αυτήν την αρχιτεκτονική και ένα εργαλείο με το όνομα *Context Toolkit* [4,32]. Στο πρώτο άρθρο που δημοσιεύτηκε μαζί με τους *Abowd* και *Salber* γίνεται μια ολοκληρωμένη μελέτη στη θεωρία των *CA εφαρμογών* [32]. Ορίζεται το *context* με τρόπο αρκετά γενικό και εντοπίζονται οι δυσκολίες στο χειρισμό του, εξαιτίας των πολυάριθμων κατανεμημένων αισθητήρων και συσκευών που απαιτούνται. Επισημαίνονται τα βασικά προβλήματα προς επίλυση, όπως η ανάγκη για ερμηνεία της πληροφορίας με αφαιρετικό τρόπο, η ανάγκη για μόνιμη αποθήκευση του *context* και η πρόσβαση στην πληροφορία.

Η αρχιτεκτονική του *Context Toolkit* βασίζεται σε μια αντικειμενοστρεφή προσέγγιση και περιλαμβάνει τρεις τύπους αντικειμένων: *widgets*, *servers* ή *aggregators* και *interpreters*. Το *context widget*[1] είναι ένα μια μονάδα λογισμικού που παρέχει στην εφαρμογή πληροφορία *context* από το περιβάλλον εκτέλεσής της. Ο ρόλος των *widgets* είναι να απομονώσουν την εφαρμογή από τις διαδικασίες συλλογής της *context* πληροφορίας. Κρύβουν την πολυπλοκότητα των αισθητήρων παρέχοντας αφαιρετικά την πληροφορία, ώστε να μπορεί να χρησιμοποιηθεί από την εφαρμογή.

Οι *context server*s είναι υπεύθυνοι για τη συλλογή της πληροφορίας του *context* που αφορά μια συγκεκριμένη οντότητα, όπως για παράδειγμα ένας χρήστης. Αποτελούν

[1] Το όνομα *widget* προέρχεται από την αντίστοιχη έννοια των *GUIs*.

υποκλάσεις των *widgets* και κληρονομούν όλες τις ιδιότητες και τις μεθόδους ενός *widget*. Ο *context server* κάνει εγγραφή στο *widget* που τον ενδιαφέρει, αυτό δηλαδή που παρέχει την πληροφορία που σχετίζεται με κάποια οντότητα και λειτουργεί σαν ένα *proxy* μεταξύ *widget* και εφαρμογής.

Οι *context interpreters* μεταφράζουν το *context* σε κατανοητή μορφή για την εφαρμογή. Μπορούν να κάνουν μετατροπές της πληροφορίας, σε διάφορους τύπους αναπαράστασης και να συνδυάσουν διαφορετικούς τύπους πληροφορίας συνθέτοντας και περαιτέρω *context*.

Κάθε ένα από τα παραπάνω αντικείμενα μπορούν να εκτελεστούν αυτόνομα. Τα αντικείμενα μπορούν να δημιουργηθούν όλα σε έναν κόμβο ή σε πολλούς διαφορετικούς. Για την επικοινωνία μεταξύ των αντικειμένων χρησιμοποιείται *http* και *XML* [50]. Ωστόσο και άλλες τεχνολογίες μπορούν να χρησιμοποιηθούν. Η βασική υλοποίηση είχε γίνει σε *Java*, αλλά οι μηχανισμοί που χρησιμοποιούνται είναι ανεξάρτητοι από τη γλώσσα προγραμματισμού.

Από τότε που παρουσιάστηκε μέχρι και σήμερα, το *Context Toolkit* ενέπνευσε πολλούς ερευνητές οι οποίοι επέκτειναν ή στήριξαν τις δουλειές τους στις ιδέες του *Dey* [43,51]. Οι αρχιτεκτονικές αρχές του *Toolkit* βρήκαν μεγάλη απήχηση στην ερευνητική κοινότητα και αποτέλεσαν τη βάση για αρκετά πλαίσια ανάπτυξης *CA* εφαρμογές, που προτάθηκαν από τότε.

Οι *Hong* και *Landay* [43] επέκτειναν τη δουλειά του *Dey*, μελετώντας την ιδέα μιας υποδομής υπηρεσιών, ενδιάμεσου λογισμικού. Προτείνουν μια γενική αρχιτεκτονική, όπου η διαχείριση του *context* παρέχεται σαν ένα σύνολο υπηρεσιών από μια υποδομή ενδιάμεσου λογισμικού. Τα *widgets* αντικαθίστανται από τεχνικές εύρεσης υπηρεσιών μεταξύ των διάχυτων κόμβων. Αυτή η προσέγγιση παρέχει ανοχή σε σφάλματα που μπορεί να προκύψουν από την κατάρρευση κάποιου *widget*, αλλά χάνει σε αποτελεσματικότητα, λόγω της αυξημένης δικτυακής επικοινωνίας που απαιτεί μεταξύ των συστατικών του συστήματος.

Ο *Winogrard* [52] σύγκρινε διαφορετικές αρχιτεκτονικές εκδοχές για το χτίσιμο *CA* συστημάτων και όρισε κριτήρια και *trade – offs* στην επιλογή του αρχιτεκτονικού μοντέλου.

Κατέληξε ότι, μια προσέγγιση *μαυροπίνακα* (*blackboard-based*) είναι πιο ευέλικτη, από ότι αρχιτεκτονικής βασισμένες σε *widgets*, όπως το *ContextToolkit*. Σε αυτήν την αρχιτεκτονική οι εφαρμογές γράφουν μηνύματα σε ένα κοινό μέσο, το μαυροπίνακα και εγγράφονται σε κάποια υπηρεσία ειδοποίησης, ώστε να ειδοποιηθούν, όταν συμβούν συγκεκριμένα γεγονότα.

Στην εργασία [53] προτείνεται ένα πλαίσιο με κατανεμημένη αρχιτεκτονική για τη διαχείριση του *context*. Το σύστημα *Rome*, βασίζεται στην έννοια του ενεργοποιητή *context* (*context trigger*). Οι ενεργοποιητές αποτελούνται από συνθήκες και μία ενέργεια που εκτελείται αν ικανοποιηθούν οι συνθήκες. Οι ενεργοποιητές βρίσκονται αποθηκευμένοι σε έναν κεντρικό κόμβο και η αποτίμηση τους μπορεί να γίνει είτε στον κεντρικό κόμβο ή στις συσκευές. Ο κεντρικός κόμβος συντονίζει τη διαδικασία αποτίμησης των ενεργοποιητών, στέλνοντας τους στις συσκευές, συμβάλλοντας με δεδομένα στη διαδικασία και καθορίζοντας το χρόνο ενεργοποίησης.

Στην εργασία [54] εξετάζονται τα *CA* συστήματα ευρύτερα και όχι μόνο στο επίπεδο αλληλεπίδρασης της εφαρμογής με το περιβάλλον. Προτείνεται μια αρχιτεκτονική όπου, το *CA* σύστημα εκτός της εφαρμογής περιλαμβάνει και ένα δυναμικό σύνολο από πράκτορες που κινούνται μέσω δικτύου. Οι πράκτορες, όπως και οι εφαρμογές αλληλεπιδρούν σε ένα ευρύτερα κατανεμημένο περιβάλλον, με διάχυτη πληροφορία από *context* και πιο σύνθετες διαδικασίες αποφάσεων.

Στην εργασία [55] περιγράφεται ένα απλό αρχιτεκτονικό πλαίσιο που έχει χρησιμοποιηθεί για την ανάπτυξη δύο *CA* εφαρμογών για μουσεία. Το πλαίσιο στοχεύει στην υποστήριξη εφαρμογών σε έξυπνους χώρους όπως ένα μουσεία. Παρέχει μια υπηρεσία βασισμένη σε απλούς κανόνες για την επιλογή των πληροφοριών που θα σταλούν στη συσκευή του χρήστη. Όλη η διαχείριση γίνεται από έναν κεντρικό εξυπηρέτη που δέχεται το *context* από την εφαρμογή και το περιβάλλον εκτέλεσης και στέλνει δεδομένα στη συσκευή του χρήστη.

Το *CoolAgent* [56] είναι ένα πλαίσιο βασισμένο σε μια αρχιτεκτονική με πολλαπλούς πράκτορες. Υποστηρίζει υπηρεσίες συλλογής και διαμοιρασμού *context* και εξαγωγής συμπερασμάτων. Το *context* μοντελοποιείται με χρήση οντολογίας και *RDF* (*Resource Description Framework*) [57]. Οι υπηρεσίες παρέχονται μέσω πρακτόρων με

συγκεκριμένους ρόλους, ενώ η εξαγωγή συμπερασμάτων γίνεται τοπικά πάνω σε μία βάση δεδομένων *context*, με ένα σύστημα κανόνων υλοποιημένο σε *Prolog*. Η βάση δεδομένων βρίσκεται σε έναν κεντρικό κόμβο εξυπηρέτη, που διαχειρίζεται το *context* και το παρέχει στις εφαρμογές.

Στο [58] προτείνεται ένα πλαίσιο που στοχεύει στην ανακάλυψη και αυτόματη εκτέλεση κατάλληλων υπηρεσιών σε διάχυτα περιβάλλοντα. Στο σύστημα *CAPEUS* που υλοποιήθηκε, η επιλογή της κατάλληλης υπηρεσίας προς εκτέλεση γίνεται με βάση τις ανάγκες του χρήστη, που καθορίζονται από το *context* και τις παρεχόμενες υπηρεσίες. Οι ανάγκες του χρήστη και οι διαθέσιμες υπηρεσίες κωδικοποιούνται υπό μορφή «περιορισμών», δίνοντας τιμές σε κάποια γνωρίσματα του *context*. Οι περιορισμοί καταχωρούνται μέσα σε έγγραφα τα οποία μεταφέρονται για να βρεθεί η κατάλληλη υπηρεσία. Τα έγγραφα ονομάζονται *CAPs* (*context aware packets)* και περιέχουν επιπλέον ένα πλάνο, που υποδεικνύει από ποιους παροχείς υπηρεσιών πρέπει να περάσει το πακέτο. Κάθε παροχέας παραλήπτης ελέγχει αν η υπηρεσία που παρέχει ικανοποιεί τους περιορισμούς του *CAP*. Επιπλέον μπορεί να κάνει υποθέσεις για τον κατάλληλο παροχέα και να του προωθήσει το *CAP*.

Η χρησιμοποίηση των *CAPs* έχει προταθεί από την ίδια ερευνητική ομάδα για την εύρεση *context* πληροφορίας σε ένα διάχυτο περιβάλλον [14,59]. Η εύρεση του κατάλληλου προμηθευτή *context*, ακολουθεί την ίδια φιλοσοφία, με την εύρεση της κατάλληλης υπηρεσίας, όπως περιγράφηκε παραπάνω. Παρόλα αυτά τα *CAPs* δεν μπορούν να χρησιμοποιηθούν για την ανίχνευση μεταβολών στο *context*.

Το *Context Fabric* [60] είναι ουσιαστικά η συνέχεια της εργασίας [43] από το *Hong*. Σε αυτήν την εργασία προτείνεται ένα αρχιτεκτονικό πλαίσιο για την ανάπτυξη και υποστήριξη *CA* εφαρμογών. Τα βασικά δομικά κομμάτια της αρχιτεκτονικής είναι: μία βάση δεδομένων για αποθήκευση και μοντελοποίηση του *context*, μία γλώσσα αναπαράστασης (*context specification language*) και μηχανισμοί ασφαλείας για την εμπιστευτικότητα των δεδομένων.

Το *Solar* [61,62] αποτελεί μία υποδομή για τη συλλογή και το διαμοιρασμό του *context* σε διάχυτα περιβάλλοντα. Το *context* που παράγεται από τις πηγές μοντελοποιείται

σε γεγονότα. Η εφαρμογή μπορεί να ορίσει ένα γράφο από τελεστές (φίλτρα, μετατροπείς κ.α.) οι οποίοι μεσολαβούν μεταξύ των προμηθευτών και της εφαρμογής. Τα γεγονότα διασχίζουν τους κόμβους του γράφου και παραδίδονται στις εφαρμογές που έχουν ενδιαφερθεί. Τα χαρακτηριστικά του *context*, που επιθυμεί η εφαρμογή περιγράφονται με μία συγκεκριμένη γλώσσα αναπαράστασης.

Το *MyCampus* [63] είναι ένα πλαίσιο υποστήριξης *CA* εφαρμογών που στοχεύει κυρίως στην ανακάλυψη και παροχή διάχυτων υπηρεσιών. Αποτελείται από ένα σύνολο προσαρμοζόμενων πρακτόρων, για την αυτόματη ανακάλυψη και χρήση κατανεμημένων υπηρεσιών. Χρησιμοποιείται ένα σύνολο οντολογιών για την περιγραφή του *context* και της συμπεριφοράς των πρακτόρων. Οι πράκτορες έχουν ικανότητες να φιλτράρουν και να προωθούν μηνύματα, να κάνουν υπενθυμίσεις και να εκτελούν αυτόματα υπηρεσίες διαδικτύου.

Το πλαίσιο *Chisel* [64] παρέχει μηχανισμούς για αυτόματη προσαρμογή υπηρεσιών κατά τη διάρκεια εκτέλεσης μιας εφαρμογής. Ο μηχανισμός προσαρμογής βασίζεται στην επιλογή πιθανών συμπεριφορών για τα αντικείμενα της εφαρμογής με βάση το *context*. Οι συμπεριφορές αντιστοιχίζονται σε πολιτικές προσαρμογής. Καθώς το *context* αλλάζει, το αντικείμενο προσαρμόζει ανάλογα τη συμπεριφορά του. Η συμπεριφορά ενός αντικειμένου κωδικοποιείται με τη γλώσσα *Iguana/J*.

Η *Gaia* [27] είναι μια υποδομή ενδιάμεσου λογισμικού, που επεκτείνει τα τυπικά λειτουργικά συστήματα παρέχοντας *CA* χαρακτηριστικά στις εφαρμογές. Στοχεύει στην ανάπτυξη και υποστήριξη *CA* εφαρμογών για ενεργούς χώρους. Παρέχει υπηρεσίες για εύρεση και χρήση υπολογιστικών πόρων, κατανομή της *context* πληροφορίας και εξαγωγή συμπερασμάτων. Το *context* μοντελοποιείται σε κατηγορήματα τεσσάρων πεδίων και χρησιμοποιούνται κανόνες κατηγορηματικής λογικής πρώτης τάξεως για την εξαγωγή συμπερασμάτων. Οι κανόνες γράφονται σε *DAML+OIL*.

Το *IrisNet* [15] είναι πλαίσιο που υποστηρίζει την ανάπτυξη και παροχή υπηρεσιών που απαιτούν πληροφορία από αισθητήρες, σε δίκτυα ευρείας κλίμακας. Η προσέγγιση του *IrisNet* στοχεύει σε ένα μελλοντικό διαδίκτυο αισθητήρων και εστιάζεται κυρίως σε πληροφορία *context* από *video* κάμερες. Το *IrisNet*, παρέχει υπηρεσίες όπως συλλογή,

φιλτράρισμα και συνδυασμός δεδομένων από αισθητήρες, ενώ εκτελεί και κατανεμημένες ερωτήσεις με λογική χρονική απόκριση. Η απόκτηση του *context* γίνεται μέσω ερωτήσεων, ενώ η εύρεση του ζητούμενου αισθητήρα γίνεται με ένα μηχανισμό αναζήτησης βασισμένο σε πράκτορες.

Το πλαίσιο *SCI* [65] παρέχει υπηρεσίες ενδιάμεσου λογισμικού για διαχείριση του *context*, όπως αναζήτηση, σύνθεση και παροχή πληροφορίας. Το *SCI* στοχεύει κυρίως στη δυναμική σύνθεση της πληροφορίας, την ανοχή σε σφάλματα και την κλιμάκωση σε έξυπνους χώρους. Την πληροφορία διαχειρίζονται κεντρικοί εξυπηρέτες και μοντελοποιείται σε κατευθυνόμενους γράφους. Οι εφαρμογές κάνουν ερωτήσεις για την απόκτηση της πληροφορίας.

Στην εργασία [66] προτείνεται το *AURA*, μια υποδομή για την ανάπτυξη *CA* εφαρμογών. Βασίζεται στην ιδέα ότι, στις *CA* εφαρμογές ο πιο «περιορισμένος» και κρίσιμος πόρος είναι η *προσοχή* του χρήστη. Η προσοχή ως παράμετρος του *context*, έχει την έννοια της συγκέντρωσης και της διάθεσης χρόνου από το χρήστη για να ασχοληθεί με την εφαρμογή του. Για το λόγο αυτό στοχεύει στην ανάπτυξη εφαρμογών, που προσαρμόζονται αυτόματα στις αλλαγές του περιβάλλοντος, ελαχιστοποιώντας την συμμετοχή του χρήστη. Το *AURA* στοχεύει στη παροχή τριών βασικών υπηρεσιών: διαφανή επιλογή του κατάλληλου μοντέλου αλληλεπίδρασης του χρήστη με μια διαδικασία, ενημερότητα για το *context*, πρόβλεψη της επιθυμητής ενέργειας και πραγματοποίηση της, χωρίς την συμμετοχή του χρήστη.

Στην εργασία [25] προτείνεται το *CARISMA*, ένα πλαίσιο ενδιάμεσου λογισμικού για *CA* εφαρμογές. Το *CARISMA* χρησιμοποιεί την αρχή της *ανάκλασης* [67] και τα μεταδεδομένα για παροχή υπηρεσιών προσαρμοσμένων στο *context* [68]. Το σύστημα βασίζεται στην παροχή πολλαπλών υλοποιήσεων της ίδιας υπηρεσίας, ώστε να παρέχεται με διαφορετικό τρόπο ανάλογα με το *context*. Οι διαφορετικοί τρόποι με τους οποίους παρέχεται μια υπηρεσία ονομάζονται *πολιτικές*. Η συμπεριφορά του ενδιάμεσου λογισμικού περιγράφεται σαν ένα σύνολο συσχετίσεων μεταξύ των παρεχόμενων υπηρεσιών, των πολιτικών με τις οποίες παρέχονται οι υπηρεσίες και τις συνθήκες του *context*, που πρέπει να ισχύουν για την εφαρμογή μιας πολιτικής.

Η σημασιολογική πληροφορία για το *context* της εφαρμογής κωδικοποιείται σε *XML* έγγραφα, σχηματίζοντας το προφίλ της εφαρμογής. Όταν εκτελείται μια υπηρεσία της εφαρμογής, το ενδιάμεσο λογισμικό ελέγχει το προφίλ της εφαρμογής και το συγκρίνει με το τρέχον *context*. Ανάλογα επιλέγει μια κατάλληλη πολιτική για την εκτέλεση της υπηρεσίας. Το προφίλ της εφαρμογής μπορεί να αλλάξει κατά τη διάρκεια εκτέλεσης. Επιπλέον χρησιμοποιείται ένας μηχανισμός για την επίλυση συγκρούσεων στην επιλογή της κατάλληλης πολιτικής, βασισμένος σε μια μικροοικονομική προσέγγιση.

Το *CMF (Context Manager Framework)* [69] είναι ένα πλαίσιο διαχείρισης του *context* που στοχεύει στη συλλογή, ερμηνεία και παροχή της *context* πληροφορίας. Το σύστημα διαχειρίζεται κυρίως *context* από το περιβάλλον (θόρυβο, θέση, θερμοκρασία κ.α.), το οποίο μοντελοποιείται με χρήση μιας οντολογίας. Χρησιμοποιείται ασαφής λογική και Μπεϋζιανή μάθηση για τη εξαγωγή υψηλότερου σημασιολογικά *context* από χαμηλού επιπέδου πληροφορία.

Το *CMF* ακολουθεί κλασική ιεραρχική αρχιτεκτονική, με τα κομμάτια του συστήματος τοποθετημένα σε επίπεδα. Η αρχιτεκτονική περιλαμβάνει τέσσερις λειτουργικές οντότητες: το διαχειριστή του *context*, τους εξυπηρέτες που παρέχουν το *context* (προμηθευτές), τις υπηρεσίες αναγνώρισης του *context* και την εφαρμογή. Οι εξυπηρέτες είναι κατανεμημένοι, ενώ ο διαχειριστής του *context*, λειτουργεί σαν μια κεντρικοποιημένη μονάδα κάτω από την εφαρμογή. Στις αρμοδιότητές του είναι να αποθηκεύει κατάλληλα τα δεδομένα που του στέλνουν οι προμηθευτές και να παρέχει την εξαγόμενη πληροφορία στις εφαρμογές. Οι υπηρεσίες αναγνώρισης χρησιμοποιούνται για την ερμηνεία του *context* με βάση μια καθορισμένη οντολογία και βρίσκονται κατανεμημένες στο περιβάλλον λειτουργίας.

Το πλαίσιο ενδιάμεσου λογισμικού *Hydrogen* [70] υιοθετεί μία *peer-to-peer* αρχιτεκτονική, όπου όλα τα τμήματα για τη συλλογή και διαχείριση του *context* βρίσκονται πάνω στην κινητή συσκευή. Το σύστημα είναι ανεξαρτητοποιημένο από κεντρικούς κόμβους παροχής υπηρεσιών και παρέχει τη δυνατότητα στην εφαρμογή να λειτουργεί, ακόμα και όταν η συσκευή δεν είναι συνδεδεμένη σε κάποιο ασύρματο δίκτυο. Παρέχει υπηρεσίες για συλλογή του *context* από κατανεμημένους προμηθευτές και άλλους κόμβους.

Το ενδιάμεσο λογισμικό έχει τη δυνατότητα να αποθηκεύσει περιορισμένη ποσότητα *context* πληροφορίας. Η πληροφορία μπορεί να μεταφερθεί μεταξύ των εφαρμογών με μία υπηρεσία διαμοιρασμού του *context*. Όταν οι συσκευές βρεθούν σε κοντινή απόσταση πραγματοποιείται ασύρματα διαμοιρασμός πληροφορίας, ώστε να περιλαμβάνεται σε παραπάνω από έναν κόμβους. Η επικοινωνία μεταξύ των εφαρμογών γίνεται πάνω από *TCP/IP*, με *XML* μηνύματα.

Το πλαίσιο *CoBrA* (*Context Broker Architecture*) [71,72] ακολουθεί μια αρχιτεκτονική βασισμένη σε πράκτορες για την παροχή *CA* υπηρεσιών σε ευφυής χώρους. Το σύστημα περιλαμβάνει μια κεντρική μονάδα διαχείρισης του *context* και το διανέμει στους πράκτορες. Η κεντρική μονάδα είναι ένας ευφυής *broker*, ενώ οι πράκτορες μπορεί να είναι κινητές εφαρμογές, υπηρεσίες που παρέχονται από εφαρμογές στο χώρο εκτέλεσης ή εφαρμογές διαδικτύου. Η κεντρική μονάδα αποτελείται από μια βάση *context* δεδομένων, μια μηχανή εξαγωγής συμπερασμάτων, ένα τμήμα για τη συλλογή του *context* και ένα τμήμα για τη διαχείριση της ασφάλειας των δεδομένων. Το *CoBrA* στοχεύει στην ανάπτυξη *CA* εφαρμογών σε έξυπνους χώρους, όπως ένα δωμάτιο συνεδριάσεων [73,74]. Για την αναπαράσταση της πληροφορίας με μορφή οντολογίας χρησιμοποιείται η *OWL* [L6].

Το *SOCAM* (*Service Oriented Context-Aware Middleware)* [75] είναι επίσης ένα πλαίσιο που στοχεύει στη συλλογή και παροχή της *context* πληροφορίας. Ακολουθεί την φιλοσοφία ενός κεντρικού υποσυστήματος για τη διαχείριση του *context*. Ο εξυπηρέτης (*context interpreter*) συλλέγει τα δεδομένα από κατανεμημένους αισθητήρες, τα επεξεργάζεται και τα παρέχει στις εφαρμογές μέσω ερωτήσεων. Οι υπηρεσίες, που μπορούν να χρησιμοποιήσουν οι εφαρμογές, βρίσκονται στην κορυφή της αρχιτεκτονικής, είτε τοπικά στη συσκευή είτε κατανεμημένες στο δίκτυο. Οι εφαρμογές μπορούν να κάνουν χρήση διαφορετικών επιπέδων *context* ρωτώντας την κεντρική μονάδα ή ακούγοντας τα γεγονότα που στέλνουν οι προμηθευτές. Η μοντελοποίηση του *context* έγινε με χρήση οντολογιών και η εξαγωγή συμπερασμάτων με το εργαλείο *Jena2* [L7].

Στην εργασία [76], από την ομάδα που πρότεινε το *SOCAM*, οι οντολογίες του συστήματος επεκτείνονται για να συμπεριλάβουν *context* για ευφυείς χώρους. Το νέο πλαίσιο ονομάζεται *Semantic Space* και στοχεύει στην ανάπτυξη και υποστήριξη *CA* εφαρμογών για ευφυείς χώρους.

Στην εργασία [13] προτείνεται ένα αρχιτεκτονικό πλαίσιο για συλλογή και παροχή *context* πληροφορίας. Στο σύστημα *Merino* οι αισθητήρες ποικίλουν από απλές συσκευές υλικού μέχρι πιο σύνθετες λειτουργικές μονάδες με περισσότερες ικανότητες επεξεργασίας. Οι αισθητήρες ακολουθούν μια ιεραρχική κατηγοριοποίηση σε επίπεδα, με βάση τις ικανότητές τους και ένα ευφυές υποσύστημα είναι υπεύθυνο για τη διαχείρισή τους.

Το *CASS* [77] είναι ένα ακόμα πλαίσιο που υιοθετεί την αρχιτεκτονική με έναν κεντρικό κόμβο για την επεξεργασία του *context*. Οι εφαρμογές - πελάτες είναι ανεξαρτητοποιημένες από την επεξεργασία του *context* και επικοινωνούν με τον κεντρικό κόμβο λαμβάνοντας την επεξεργασμένη πληροφορία. Το ενδιάμεσο λογισμικό περιλαμβάνει υπηρεσίες για επικοινωνία με κατανεμημένους αισθητήρες, μια μηχανή εξαγωγής συμπερασμάτων, διερμηνέα του *context* και μία βάση δεδομένων για αποθήκευση και ανάκτηση της πληροφορίας. Ο πελάτης περιλαμβάνει τμήματα για επικοινωνία με τις υπηρεσίες του ενδιάμεσου λογισμικού.

Στο πλαίσιο *CORTEX* [78] δεν υπάρχει κάποιος κεντρικός εξυπηρέτης για τη διαχείριση του *context*. Το ενδιάμεσο λογισμικό στέκεται κάτω από κάθε ξεχωριστή εφαρμογή. Περιλαμβάνει υπηρεσίες για συλλογή, αποτίμηση και εξαγωγή *context* πληροφορίας. Χρησιμοποιούνται Μπεϋζιανά δίκτυα, για τη διαχείριση της αβεβαιότητας των δεδομένων από τους αισθητήρες. Για την εξαγωγή συμπερασμάτων χρησιμοποιείται ένα σύστημα γραμμένο σε *CLIPS*. Το πλαίσιο περιλαμβάνει μια υπηρεσία για ενημερότητα θέσης και η επικοινωνία μεταξύ των υποσυστημάτων είναι βασισμένη σε γεγονότα (*STEAM*) [79]. Επιπλέον παρέχει ένα γραφικό εργαλείο για την ανάπτυξη των αντικειμένων που παρέχουν τις υπηρεσίες του συστήματος. Παρόλα αυτά δεν δίνονται καθόλου στοιχεία και περιγραφές, για την υλοποίηση όλων των παραπάνω υπηρεσιών.

Το σύστημα *WASP* (*Web Architectures for Services Platforms*) [80] είναι μια υποδομή για την ανάπτυξη και υποστήριξη *CA* εφαρμογών, που παρέχει υπηρεσίες διαδικτύου, κινητής ομιλίας και δίκτυα κινητών τηλεφώνων τρίτης γενιάς (*WASP Project,* [L7]). Στην αρχιτεκτονική του *WASP* ένα κεντρικό υποσύστημα (*Context Interpreter*) είναι υπεύθυνο για τη συλλογή και παροχή της πληροφορίας. Η πληροφορία παρέχεται στο υποσύστημα παρακολούθησης (*Monitor Module*), το οποίο υποστηρίζεται από ένα σύνολο μονάδων

αποθήκευσης, για καταχώρηση της πληροφορίας για μελλοντική χρήση. Για την διασύνδεση μιας εφαρμογής με τη πλατφόρμα *WASP*, χρησιμοποιείται η γλώσσα *WSL*, στην οποία κωδικοποιούνται η συμπεριφορά και τα στοιχεία της εφαρμογής. Η αναπαράσταση της πληροφορίας γίνεται με χρήση οντολογιών. Η διαχείριση των οντολογιών γίνεται με την *OWL* [L6].

Στην εργασία [81] εξετάζονται τα θέματα που αφορούν τη σχεδίαση και ανάπτυξη ενός *CA* συστήματος, από την πλευρά της Τεχνολογίας Λογισμικού. Προτείνεται ένα πλαίσιο ανάπτυξης *CA* συστημάτων που στοχεύει στην απλοποίηση και τυποποίηση των διαδικασιών ανάπτυξης *CA* λογισμικού. Το πλαίσιο περιλαμβάνει μεθόδους μοντελοποίησης του *context*, μία τεχνική για μοντελοποίηση των προτιμήσεων του χρήστη και δύο συμπληρωματικά προγραμματιστικά μοντέλα.

Επιπλέον έχουν προταθεί αρχιτεκτονικά μοντέλα εξειδικευμένα για εφαρμογές που επεξεργάζονται πληροφορία θέσης [82,83].

3.3 Μοντελοποίηση του Context

Έχουν προταθεί πολλά διαφορετικά μοντέλα για το *context*, με στόχο να διευκολύνουν τις διαδικασίες σχεδιασμού και ανάπτυξης, ενός *CAS* [84]. Ο στόχος ενός μοντέλου είναι να συμπεριλάβει τα γνωρίσματα του *context* και να τα απεικονίσει με μια ευέλικτη και εύχρηστη αναπαράσταση. Η μέθοδος αναπαράστασης αποτελεί βασικό κομμάτι όλων των υπηρεσιών ενός *CAM*. Επιπλέον είναι επιθυμητό, το μοντέλο του *context* να είναι αρκετά ρεαλιστικό και να συμβαδίζει με την ανθρώπινη θεώρηση για το *context*.

Μια σημαντική μελέτη για τη μοντελοποίηση του *context* γίνεται στην εργασία [7]. Επισημαίνονται οι βασικές ιδιότητες της πληροφορίας του *context* και με βάση αυτές παρατίθεται ένα σύνολο από γνωρίσματα, που πρέπει να περιλαμβάνει ένα μοντέλο. Επιπλέον προτείνεται ένα αντικειμενοστρεφές μοντέλο στο οποίο, η πληροφορία δομείται, γύρω από ένα σύνολο φυσικών (συσκευές) ή εννοιολογικών (άτομα, κανάλια επικοινωνίας) οντοτήτων. Οι ιδιότητες των αντικειμένων αναπαρίστανται σαν γνωρίσματα και οι οντότητες συνδέονται με τα γνωρίσματα και άλλες ιδιότητες μέσω συσχετίσεων. Στις συσχετίσεις

επιβάλλονται περιορισμοί μέσω εξαρτήσεων. Συνολικά ένα τέτοιο μοντέλο μπορεί να απεικονιστεί σαν ένας κατευθυνόμενος γράφος.

Στο [85] ορίζεται ένα λειτουργικό μοντέλο για το *context*, χρήσιμο για το σχεδιασμό και την ανάπτυξη *CA* συστημάτων. Στο μοντέλο εισάγεται η έννοια της *κατάστασης*. Η *κατάσταση* ορίζεται σαν το σύνολο των γνωρισμάτων που χαρακτηρίζουν μια οντότητα *context*, σε ένα στιγμιότυπο της λειτουργίας του συστήματος (*observables*). Το *context* ορίζεται, σαν η σύνθεση όλων των καταστάσεων στο χρόνο. Ένα αφαιρετικό μοντέλο λογισμικού (*Contextor*), προτείνεται για τη μοντελοποίηση των συσχετίσεων μεταξύ των *observables*. Ο *Contextor* επιστρέφει την τιμή μιας μεταβλητής που ανήκει σε κάποιο συγκεκριμένο *context*.

Μια διαφορετική προσέγγιση στην μοντελοποίηση του *context*, δίνεται στο [86]. Το μοντέλο που προτείνεται επικεντρώνεται στην εκτέλεση μιας δραστηριότητας από έναν πράκτορα, με βάση τη γνώση. Η ιδέα είναι ότι, το *context* μιας δραστηριότητας (ή μιας ομάδας δραστηριοτήτων), μπορεί να μοντελοποιηθεί εκ των προτέρων, για κάποια πεδία δραστηριοτήτων. Ο πράκτορας εκτελεί τη δραστηριότητα με βάση το προκαθορισμένο *context* και μία κεντρική μονάδα διαχείρισης (*Context Manager*) τον καθοδηγεί κάνοντας του προτάσεις και ελέγχοντας τις ενέργειές του. Με χρήση τεχνικών μηχανικής μάθησης με επίβλεψη, το *context* της δραστηριότητας εξελίσσεται, ώστε να περιγράφει καλύτερα τη δραστηριότητα.

Στην εργασία [87] προτείνεται ένα μοντέλο, που βασίζεται στην αντίληψη για το *context*. Σε αυτή τη προσέγγιση, το *context* είναι ένα δίκτυο *καταστάσεων*. Η *Κατάσταση* ορίζεται σαν μια συγκεκριμένη ανάθεση ρόλων στις οντότητες μαζί με τις *σχέσεις* μεταξύ των οντοτήτων. Οι ρόλοι είναι *υπηρεσίες ή συναρτήσεις* σχετικές με μια διεργασία και μπορούν να σχετίζονται με παραπάνω από μία οντότητες. Μια *σχέση* είναι ένα κατηγόρημα ορισμένο πάνω στα *γνωρίσματα* των οντοτήτων. Οι οντότητες και οι σχέσεις μεταξύ τους είναι κατηγορήματα ορισμένα σε ένα σύνολο από μεταβλητές που συνθέτουν το *context*.

3.4 Μοντελοποίηση Αξιοπιστίας

Ένα βασικό γνώρισμα της πληροφορίας που συνθέτει το *context* είναι η *ατέλεια* (*imperfection*) [7]. Οι τιμές του *context* σε διάχυτα συστήματα προέρχονται κυρίως από αναξιόπιστες ή ανακριβείς πηγές (*αισθητήρες*). Επιπλέον λόγω των ασταθών συνδέσεων υπάρχει μεγάλη πιθανότητα να χαθεί μέρος των δεδομένων κατά τη μετάδοση. Η δεδομένη αμφιβολία για την ακρίβεια του *context* αφορά ζητήματα ποιότητας της πληροφορίας που λαμβάνεται. Το πρόβλημα αυτό είναι γνωστό στη βιβλιογραφία, με τον όρο *ποιότητα του context (QoC)* [7]. Στη συνέχεια αναφέρουμε τις πιο χαρακτηριστικές εργασίες που ασχολούνται με θέματα ποιότητας του *context*.

Το μοντέλο για το *context* που προτείνεται στην εργασία [7] δανείζεται ιδέες από τον *Wang* [88] ώστε να συμπεριλάβει και θέματα ποιότητας της πληροφορίας. Τα γνωρίσματα των οντοτήτων και οι συσχετίσεις, χαρακτηρίζονται από κάποιους δείκτες ποιότητας. Οι δείκτες αυτοί είναι ανεξάρτητοι της ιδιότητας που χαρακτηρίζουν και οι τιμές τους καθορίζουν την ποιότητα του *context*.

Οι *Gray* και *Salber* [11] ακολουθούν το ίδιο μοντέλο ποιότητας και παρουσιάζουν συγκεκριμένα γνωρίσματα (δείκτες) ποιότητας για την πληροφορία. Επιπλέον προτείνουν γνωρίσματα ποιότητας και για τα αντικείμενα (αισθητήρες) τα οποία παρέχουν την πληροφορία.

Αρκετή δουλειά έχει γίνει στη χρήση πιθανοτικών δικτύων για τη μοντελοποίηση της αβεβαιότητας στην πληροφορία του *context*. Χαρακτηριστικό παράδειγμα είναι το *MUSE* [89], το οποίο χρησιμοποιεί Μπεϋζιανά δίκτυα και κρυμμένα Μαρκοβιανά Μοντέλα για τη διαχείριση των δεδομένων του *context*.

ΚΕΦΑΛΑΙΟ 4. ΑΝΑΛΥΣΗ ΑΠΑΙΤΗΣΕΩΝ

Στο κεφάλαιο αυτό γίνεται περιγραφή των απαιτήσεων του συστήματος ενδιάμεσου λογισμικού που έχουμε υλοποιήσει. Αρχικά γίνεται μια σύντομη περιγραφή των στόχων του λογισμικού. Στη συνέχεια ορίζονται οι χρήστες και αναλύονται οι λειτουργικές απαιτήσεις του λογισμικού. Τέλος αναφέρονται τα μη λειτουργικά χαρακτηριστικά που σχετίζονται με συγκεκριμένες ποιοτικές ιδιότητες και περιορισμούς του λογισμικού.

4.1 Στόχοι Ενδιάμεσου Λογισμικού

Οι πλατφόρμες ενδιάμεσου λογισμικού παρέχουν ένα σύνολο υπηρεσιών και προγραμματιστικών διεπαφών για την ανάπτυξη και υποστήριξη εφαρμογών. Το σύνολο των υπηρεσιών εξαρτάται από το πεδίο εφαρμογής του λογισμικού, την αρχιτεκτονική προσέγγιση με την οποία σχεδιάστηκε και το είδος των εφαρμογών για τις οποίες προορίζεται. Το ενδιάμεσο λογισμικό για *CA* εφαρμογές παρουσιάζει επιπλέον λειτουργικά χαρακτηριστικά, που σχετίζονται με τις απαιτήσεις στη διαχείριση της *context* πληροφορίας. Οι βασικές απαιτήσεις για ένα *CAM*, συνοψίζονται στις εξής:

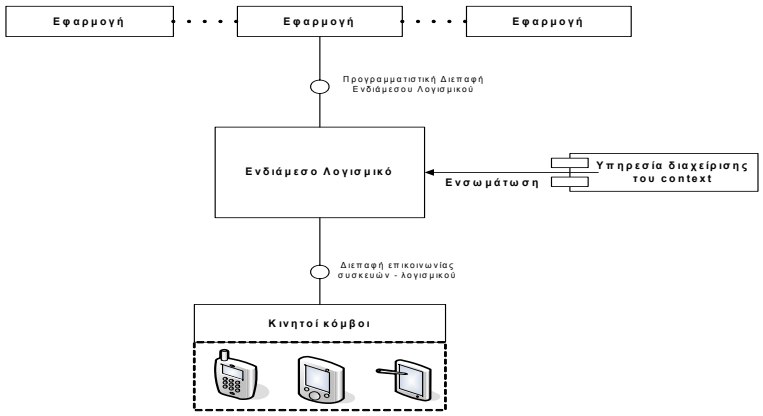
1. Ένα μοντέλο για την αναπαράσταση και διαχείριση του *context*. Το μοντέλο πρέπει να περιγράφει τα χρονικά χαρακτηριστικά της *context* πληροφορίας.
2. Μία μέθοδο για το χειρισμό των χρονικών εξαρτήσεων του *context*.
3. Ένα μηχανισμό συλλογής του *context* από τους προμηθευτές. Η ενημέρωση για τις αλλαγές του *context* θα πρέπει να γίνεται με ασύγχρονη επικοινωνία.

4. Ερμηνεία και μετατροπή του *context* σε μορφή κατάλληλη για χρήση.
5. Αποτίμηση του *context* και εξαγωγή συμπερασμάτων για την αντίδραση της εφαρμογής.
6. Έλεγχος αξιοπιστίας των δεδομένων που συλλέγονται.
7. Προσαρμογή του συστήματος στις αλλαγές του *context*.
8. Δυναμική διαμόρφωση του συστήματος.
9. Προγραμματιστικότητα.

Κάποια συστήματα ενδιάμεσου λογισμικού προορίζονται για περιβάλλοντα στα οποία η εφαρμογή μπορεί να προσπελάσει και να χρησιμοποιήσει υπηρεσίες από κατανεμημένους εξυπηρέτες ή άλλες εφαρμογές. Οι υπηρεσίες παρέχονται συνήθως από στατικές μονάδες λογισμικού σε έξυπνους χώρους ή από κινητές εφαρμογές. Σε τέτοιες συνθήκες απαιτείται ένας μηχανισμός *εύρεσης* ή *επιλογής* της κατάλληλης υπηρεσίας, η οποία μπορεί να κληθεί από την κινητή εφαρμογή του χρήστη [58].

Μια επιπλέον υπηρεσία αφορά την εύρεση προμηθευτών *context*. Σε κάποια συστήματα διάχυτου υπολογισμού γίνεται η θεώρηση ότι οι προμηθευτές του *context* δεν είναι πάντα γνωστοί στην εφαρμογή. Επίσης μπορεί να μην είναι πάντα προφανές ποιος, από μια λίστα γνωστών προμηθευτών παρέχει το επιθυμητό *context*. Έτσι η συλλογή της πληροφορίας πρέπει να γίνει από άγνωστους ή επιλεγμένους προμηθευτές. Ο προμηθευτής πρέπει να αναζητηθεί ή να επιλεγεί με βάση τα χαρακτηριστικά του. Σε αυτές τις συνθήκες απαιτούνται επιπλέον υπηρεσίες για εύρεση ή επιλογή του κατάλληλου προμηθευτή [15,59].

Η εργασία μας επικεντρώθηκε στην κάλυψη των εννιά βασικών απαιτήσεων. Το σύστημα που σχεδιάσαμε και υλοποιήσαμε αποτελεί μια *υπηρεσία ενδιάμεσου λογισμικού* για τη διαχείριση του *context* και την ανάπτυξη *CA* εφαρμογών. Η υπηρεσία παρέχεται σαν ένα πακέτο λογισμικού. Το πακέτο μπορεί να χρησιμοποιηθεί από προγραμματιστές κινητών εφαρμογών ή/και πεπειραμένους χρήστες. Για την ενσωμάτωση της υπηρεσίας σε ένα σύστημα προστίθεται λογισμικό διασύνδεσης για την επικοινωνία με τα υπόλοιπα τμήματα (Σχήμα 4.1). Η υπηρεσία μπορεί να ενσωματωθεί σε οποιοδήποτε συμβατό λογισμικό και λειτουργεί χωρίς επίβλεψη. Στη συνέχεια περιγράφονται κάποια σενάρια χρήσης του ενδιάμεσου λογισμικού για να γίνει πιο κατανοητή η λειτουργία του.



Σχήμα 4.1 Θέση της Υπηρεσίας σε Ένα CA Σύστημα.

4.1.1 Σενάρια Χρήσης

Μια κινητή εφαρμογή σχεδιάζεται ώστε να παρέχει ένα σύνολο υπηρεσιών στο χρήστη. Οι υπηρεσίες αυτές μπορεί να αφορούν υπηρεσίες διαδικτύου, όπως εμπορικές συναλλαγές, προσπέλαση ιστοσελίδων ή τοπικές λειτουργίες όπως, ηλεκτρονική ατζέντα, διαχείριση τοπικών δεδομένων και χειρισμός εισερχόμενων κλήσεων και *sms* αν πρόκειται για κινητά τηλέφωνα κ.α.

Η εφαρμογή αποκτά *CA* χαρακτηριστικά αν οι λειτουργίες της δεν εκτελούνται πάντα με τον ίδιο τρόπο και μόνο με επιλογή του χρήστη, αλλά η εκτέλεσή τους εξαρτάται από το *context*. Η προσαρμοζόμενη λειτουργία της εφαρμογής επιτυγχάνεται με την προσθήκη ενδιάμεσου λογισμικού. Η υπηρεσία ενδιάμεσου λογισμικού αναλαμβάνει να παρακολουθεί το *context* και να προσαρμόζει τη λειτουργία της εφαρμογής. Η υπηρεσία μπορεί να υποστηρίξει την *CA* συμπεριφορά της εφαρμογής με τρεις τρόπους:

1. Αυτόματη εκτέλεση λειτουργιών
2. Προσαρμοσμένη εκτέλεση λειτουργιών
3. Αυτοδιαμόρφωση

Σενάριο 1: Αυτόματη Εκτέλεση Λειτουργιών

Ένα σενάριο λειτουργίας της υπηρεσίας είναι η αυτόματη εκτέλεση λειτουργιών της εφαρμογής. Ας υποθέσουμε ότι χρησιμοποιείται μια εφαρμογή που τρέχει σε κινητές συσκευές για τη ξενάγηση επισκεπτών σε αρχαιολογικούς χώρους ή μουσεία. Η εφαρμογή

μπορεί να παρέχει στο χρήστη τη δυνατότητα να επιλέξει από ένα μενού επιλογής, στην οθόνη του κινητού του, κάποιο έκθεμα του μουσείου για το οποίο θα ήθελε να δει πληροφορίες. Χτίζοντας την εφαρμογή πάνω από την υπηρεσία η παραπάνω λειτουργία θα μπορούσε να γίνεται αυτόματα, χρησιμοποιώντας *context* που αφορά τη θέση του επισκέπτη. Η υπηρεσία παίρνοντας δεδομένα από εξωτερικούς αισθητήρες για τη θέση του επισκέπτη μέσα στο μουσείο θα μπορούσε να εμφανίζει αυτόματα πληροφορίες για το έκθεμα στο οποίο βρίσκεται *κοντά* ο επισκέπτης.

Σενάριο 2: Προσαρμοσμένη Εκτέλεση Λειτουργιών

Η υπηρεσία μπορεί να εκτελεί λειτουργίες της εφαρμογής προσαρμοσμένες κατάλληλα στο τρέχον *context*. Ας υποθέσουμε ότι μια εφαρμογή μπορεί να στέλνει δεδομένα ασύρματα με δύο τρόπους: συμπιεσμένα ή ασυμπίεστα. Η επιλογή συμπίεσης ενός αρχείου μπορεί να γίνει από το χρήστη προτού σταλεί. Το ενδιάμεσο λογισμικό μπορεί να αποφασίζει αυτόματα πως θα γίνει η αποστολή των δεδομένων ελέγχοντας το εύρος ζώνης και τη διαθέσιμη μνήμη της εφαρμογής κάθε χρονική στιγμή. Αν το εύρος ζώνης είναι μικρό τότε γίνεται αποστολή του συμπιεσμένου αρχείου. Αν η διαθέσιμη μνήμη τη χρονική στιγμή της αποστολής είναι μικρή, η συμπίεση αποφεύγεται. Συνεπώς το ενδιάμεσο λογισμικό παρακολουθώντας *context*, όπως μνήμη και εύρος ζώνης, μπορεί να προσαρμόζει κατάλληλα την εκτέλεση της λειτουργίας αποστολής μηνυμάτων.

Σενάριο 3: Αυτοδιαμόρφωση

Η υπηρεσία μπορεί να διαμορφώνει αυτόματα την εσωτερική της δομή ανάλογα με το *context*. Με την αυτοδιαμόρφωση προσαρμόζεται αυτόματα ο τρόπος με τον οποίο καθοδηγείται η συμπεριφορά της εφαρμογής. Η υπηρεσία μπορεί να βασίζεται σε ένα σύνολο καταγεγραμμένων πολιτικών προσαρμογής της εφαρμογής. Κάθε φορά που πρέπει να αποφασίσει ποια πολιτική να εφαρμόσει ψάχνει την κατάλληλη ελέγχοντας όλο το σύνολο.

Ας θεωρήσουμε ότι υπάρχουν τρεις πολιτικές χειρισμού εισερχόμενων μηνυμάτων ηλεκτρονικού ταχυδρομείου σε ένα κινητό τηλέφωνο: ηχητική ειδοποίηση και δόνηση, μόνο δόνηση και σιωπηλό χωρίς δόνηση. Η επιλογή κάποιας πολιτικής θα μπορούσε να γίνεται ελέγχοντας το χώρο στον οποίο βρίσκεται ο χρήστης, π.χ. όταν ο χρήστης βρίσκεται στην

αίθουσα συνεδριάσεων η ειδοποίηση θα είναι σιωπηλή, όταν βρίσκεται σε κάποια συνάντηση με δόνηση και σε κάθε άλλη περίπτωση με ήχο και δόνηση.

Το ενδιάμεσο λογισμικό θα μπορούσε να παρακολουθεί τη διάρκεια ζωής της μπαταρίας και να διαγράψει τις δύο πολιτικές χειρισμού με δόνηση και ήχο, όταν η μπαταρία πέσει κάτω από κάποιο όριο, π.χ. *20%* της συνολικής διάρκειας. Η διαγραφή των δύο πολιτικών αποσκοπεί στην οικονομία της μπαταρίας. Την επόμενη φορά που θα έρθει κάποιο μήνυμα, η σιωπηλή ειδοποίηση θα επιλεγεί άμεσα χωρίς να χρειαστεί να ελεγχθούν άσκοπα και άλλες περιπτώσεις. Με αυτόν τον τρόπο το ενδιάμεσο λογισμικό διαμορφώνει αυτόματα τη συμπεριφορά του προς την εφαρμογή, ανάλογα με το *context*.

4.2 Λειτουργικές Απαιτήσεις

Οι λειτουργικές απαιτήσεις περιγράφουν τις λειτουργίες που παρέχει η υπηρεσία στις οντότητες (*actors*) με τις οποίες αλληλεπιδρά. Οι λειτουργικές απαιτήσεις αφορούν κυρίως την υποστήριξη της εφαρμογής που «τρέχει πάνω» από την υπηρεσία. Επιπλέον παρέχονται και λειτουργίες που αφορούν τους χρήστες των *CA* εφαρμογών. Στη συνέχεια ορίζονται οι *actors* της υπηρεσίας και γίνεται η περιγραφή των λειτουργικών απαιτήσεων.

4.2.1 Actors Ενδιάμεσου Λογισμικού

Σε ένα *CA* σύστημα η υπηρεσία αλληλεπιδρά με τις εξής οντότητες:

- Τον προγραμματιστή της εφαρμογής. Ο προγραμματιστής αναλαμβάνει να συνδέσει την εφαρμογή με το ενδιάμεσο λογισμικό και να προσαρμόσει κατάλληλα τη λειτουργία της. Για το λόγο αυτό χρησιμοποιεί τις προγραμματιστικές διεπαφές που του παρέχει η υπηρεσία.

- Τους προμηθευτές του *context*. Προμηθευτής είναι οποιαδήποτε οντότητα παρέχει στην υπηρεσία τιμές για τις παραμέτρους του *context*. Προμηθευτής μπορεί να είναι και η εφαρμογή, αφού κάποιες παράμετροι του *context* μπορεί να σχετίζονται με λειτουργικά χαρακτηριστικά της. Οι προμηθευτές ανιχνεύουν τις αλλαγές στο

context και χρησιμοποιούν το μηχανισμό επικοινωνίας που παρέχει η υπηρεσία, για να στείλουν τις τιμές των παραμέτρων.

- Το χρήστη της εφαρμογής. Ο χρήστης μπορεί να χρησιμοποιήσει τις διεπαφές που παρέχει η υπηρεσία για να τροποποιήσει κάποια εσωτερικά χαρακτηριστικά της. Επιπλέον μπορεί να λειτουργήσει σαν προμηθευτής *context* καταχωρώντας τις προτιμήσεις του από μια γραφική διεπαφή της εφαρμογής.

Οι οντότητες αυτές αποτελούν τους *actors* του ενδιάμεσου λογισμικού και χειρίζονται τις λειτουργίες που παρέχει. Στη συνέχεια περιγράφονται οι λειτουργικές απαιτήσεις της υπηρεσίας και πως σχετίζονται με τους *actors*.

4.2.2 Περιγραφή Λειτουργικών Απαιτήσεων

Μοντέλο Διαχείρισης του Context

Η χρήση του *context* απαιτεί ένα μοντέλο διαχείρισης και αναπαράστασης της πληροφορίας. Το μοντέλο χρησιμοποιείται για την αποθήκευση του *context* σε κατάλληλη μορφή που διευκολύνει τη διαχείρισή του. Επιπλέον καθορίζει τον τρόπο που υλοποιούνται οι διαδικασίες διαχείρισης του *context*.

Χειρισμός Χρονικών Εξαρτήσεων

Η πληροφορία του *context* είναι σε μεγάλο βαθμό χρονικά εξαρτημένη. Η περιγραφή του *context* την τρέχουσα χρονική στιγμή μπορεί να περιέχει γεγονότα που συνέβησαν στο παρελθόν και τη σειρά με την οποία πραγματοποιήθηκαν. Συνεπώς η διαχείριση του *context* πρέπει να έχει και χρονικά χαρακτηριστικά. Το μοντέλο του *context* πρέπει να περιλαμβάνει κατάλληλες αφαιρέσεις για το χειρισμό των χρονικών εξαρτήσεων της πληροφορίας. Επιπλέον, όπως σε κάθε αλληλεπιδραστικό σύστημα, ο χρόνος υπεισέρχεται και στην περιγραφή της αλληλεπίδραση της υπηρεσίας με το περιβάλλον. Το μοντέλο πρέπει να παρέχει τη δυνατότητα για χρονική περιγραφή, των ενεργειών προσαρμογής της εφαρμογής. Για το χειρισμό των χρονικών εξαρτήσεων απαιτείται μία κατάλληλη μέθοδος που θα περιλαμβάνει χρονικά χαρακτηριστικά στην επεξεργασία του *context*.

Συλλογή του Context

Το *context* στην πλειοψηφία του συλλέγεται από ετερογενείς πηγές δεδομένων. Η συλλογή της πληροφορίας απαιτεί έναν κοινό μηχανισμό επικοινωνίας της υπηρεσίας με τους προμηθευτές, για την απόκτηση του *context*. Η επικοινωνία σε επίπεδο ενδιάμεσου λογισμικού θα πρέπει να είναι ανεξάρτητη από τη φύση της πηγής (συσκευή, λογισμικό, προτιμήσεις χρήστη) και τα τεχνικά χαρακτηριστικά της. Ο μηχανισμός επικοινωνίας πρέπει να μεταφέρει τις τιμές του *context* από τους προμηθευτές στην υπηρεσία διατηρώντας τη συνέπεια και τη χρονική τους πληροφορία.

Ασύγχρονη Ενημέρωση

Οι αλλαγές στο *context* πρέπει να γίνονται άμεσα αντιληπτές από την υπηρεσία προκειμένου να αποτιμηθούν και να προσαρμοστεί ανάλογα η εφαρμογή. Με δεδομένο ότι μεταβολές συμβαίνουν αρκετά συχνά και απρόβλεπτα, η ενημέρωση πρέπει να γίνεται με ασύγχρονο τρόπο. Ο μηχανισμός ειδοποίησης πρέπει να είναι ανεξάρτητος του τύπου της πληροφορίας που μεταφέρεται στα γεγονότα. Επιπλέον πρέπει να παρέχεται η δυνατότητα στο προγραμματιστή της εφαρμογής να προσθέσει πιο σύνθετους μηχανισμούς ειδοποίησης.

Η ενημέρωση για το *context* θα μπορούσε εναλλακτικά να γίνεται με απ' ευθείας ερωτήσεις στους προμηθευτές. Αυτή η μέθοδος δεν μπορεί να χρησιμοποιηθεί σαν βασικός μηχανισμός συλλογής των δεδομένων. Οι τιμές των γνωρισμάτων του *context*, στην πλειοψηφία τους, μεταβάλλονται ταχύτατα και οι ερωτήσεις μπορεί να οδηγήσουν σε ασυνεπή πληροφορία. Επιπλέον οι συνεχείς ερωτήσεις θα οδηγούσαν σε χρονοβόρες διακοπές στη λειτουργία της εφαρμογής.

Μετατροπή του Context

Οι προμηθευτές παρέχουν ετερογενή πληροφορία η οποία πρέπει να αναγνωριστεί και να μετατραπεί σε μία χρήσιμη αναπαράσταση. Η μορφή του *context* μπορεί να είναι αριθμητικές τιμές από αισθητήρες υλικού ή συμβολοσειρές. Αυτές οι τιμές δεν μπορούν να χρησιμοποιηθούν άμεσα από την υπηρεσία. Προτού γίνει χρήση του *context* πρέπει να ερμηνευτεί και να μετατραπεί σε κατάλληλη μορφή. Συνεπώς η υπηρεσία πρέπει να έχει ένα μηχανισμό απόκρυψης της ετερογένειας και της φυσικής μορφής της *context* πληροφορίας.

Ικανότητα Συλλογισμού

Η υπηρεσία πρέπει να έχει ικανότητες συλλογισμού (*reasoning*) για την αποτίμηση του *context*. Μέσω του συλλογισμού η υπηρεσία επεξεργάζεται τις τιμές του *context* και προσαρμόζει ανάλογα τη συμπεριφορά της εφαρμογής κάθε φορά που είναι απαραίτητο. Η ικανότητα συλλογισμού απαιτεί ενσωματωμένη ευφυία, ώστε να παίρνονται οι αποφάσεις που οδηγούν σε ενέργειες προσαρμογής. Δηλαδή η υπηρεσία γνωρίζοντας το εκάστοτε *context*, πρέπει να εξάγει συμπεράσματα που αφορούν την κατάλληλη συμπεριφορά της εφαρμογής.

Διαχείριση Αξιοπιστίας

Το *context* δεν μπορεί να θεωρηθεί πάντα αξιόπιστη πληροφορία. Συνεπώς τα συμπεράσματα που εξάγονται από την επεξεργασία της αναξιόπιστης πληροφορίας μπορεί να είναι λανθασμένα. Μια υπηρεσία ενδιάμεσου λογισμικού πρέπει έχει τη δυνατότητα να διαχειρίζεται την αβεβαιότητα του *context*, βγάζοντας συμπεράσματα για την ορθότητα της πληροφορίας που συλλέγεται.

Προσαρμογή στις Αλλαγές του Context

Η υπηρεσία θα πρέπει να προσαρμόζει τη λειτουργία της εφαρμογής εκτελώντας κατάλληλες ενέργειες ανάλογα με το *context*. Για την προσαρμογή απαιτείται, επιλογή των κατάλληλων ενεργειών με βάση τα συμπεράσματα που εξάγονται από τη διαδικασία συλλογισμού. Η υπηρεσία πρέπει να περιλαμβάνει ένα μηχανισμό αυτόματης αντιστοίχισης των συμπερασμάτων, σε ενέργειες προσαρμογής της εφαρμογής.

Δυναμική Διαμόρφωση

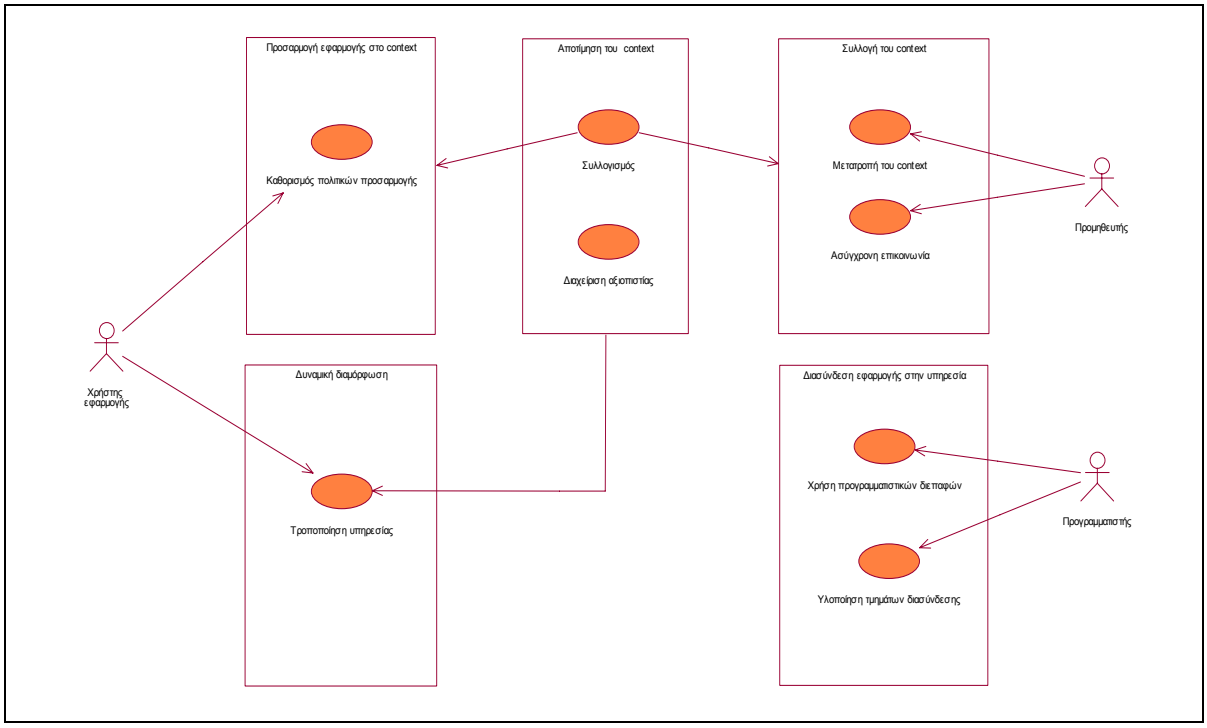
Το ενδιάμεσο λογισμικό παρέχει, σε ένα βαθμό, διαφάνεια στην εφαρμογή. Παρόλα αυτά, η απόλυτη διαφάνεια δεν είναι επιθυμητή. Η συμπεριφορά του ενδιάμεσου λογισμικού και οι πολιτικές αλληλεπίδρασης του με την εφαρμογή πρέπει να προσαρμόζονται και στις ανάγκες του χρήστη. Ο χρήστης πρέπει να έχει τη δυνατότητα να διαμορφώνει την υπηρεσία καταχωρώντας τις προσωπικές του προτιμήσεις. Η διαμόρφωση πρέπει να γίνεται δυναμικά, καθώς κατά τη διάρκεια εκτέλεσης της εφαρμογής οι ανάγκες του χρήστη αλλάζουν. Ο χρήστης πρέπει να έχει τη δυνατότητα, να βλέπει ένα μέρος της εσωτερικής δομής της υπηρεσίας και να μπορεί να την αλλάξει ανά πάσα στιγμή.

Επιπλέον, όπως αναφέρθηκε στην παράγραφο 4.2.1, η υπηρεσία θα πρέπει να αυτοδιαμορφώνεται. Ανάλογα με το *context* η συμπεριφορά της υπηρεσίας πρέπει να προσαρμόζεται, αν αυτό είναι απαραίτητο. Αυτή η προσαρμογή έχει σαν αποτέλεσμα την αποδοτικότερη λειτουργία και την καλύτερη υποστήριξη της εφαρμογής.

Προγραμματιστικότητα

Μια σημαντική απαίτηση για το ενδιάμεσο λογισμικό είναι η παροχή ενός προγραμματιστικού πλαισίου για την ανάπτυξη εφαρμογών. Το πλαίσιο πρέπει να περιλαμβάνει εργαλεία, που βοηθούν τον προγραμματιστή να χτίσει *CA* εφαρμογές πάνω στην υπηρεσία. Ο προγραμματιστής πρέπει να μπορεί να προσθέσει επιπλέον συστατικά που επεκτείνουν ή προσαρμόζουν την εφαρμογή πάνω στο ενδιάμεσο λογισμικό. Τα προγραμματιστικά εργαλεία μπορεί να είναι στη μορφή κάποιας βιβλιοθήκης λογισμικού συμβατής με μία γλώσσα υψηλού επιπέδου.

Οι λειτουργικές απαιτήσεις που περιγράφηκαν είναι οι βασικές για ένα ενδιάμεσο λογισμικό *CA* εφαρμογών. Διάφορες άλλες, λιγότερο κρίσιμες απαιτήσεις, θα μπορούσαν να αναφερθούν. Στο Σχήμα 4.2 φαίνεται η αλληλεπίδραση των *actors* με την υπηρεσία.



Σχήμα 4.2 Διάγραμμα Χρήσης της Υπηρεσίας.

4.3 Μη Λειτουργικές Απαιτήσεις

Οι μη λειτουργικές απαιτήσεις της υπηρεσίας σχετίζονται με περιορισμούς που επιβάλει το περιβάλλον στο οποίο προορίζεται να λειτουργήσει. Η υπηρεσία αποτελεί ένα αλληλεπιδραστικό σύστημα πραγματικού χρόνου. Συνεπώς περιλαμβάνει όλες τις μη λειτουργικές απαιτήσεις των συστημάτων αυτής της κατηγορίας. Επιπλέον η *CA* συμπεριφορά επιβάλλει κάποιους περιορισμούς στην υλοποίηση και τη σχεδίαση του ενδιάμεσου λογισμικού. Ακολουθεί η ανάλυση των μη λειτουργικών απαιτήσεων.

Χαρακτηριστικά Context Μοντέλου

Το μοντέλο του *context* πρέπει να έχει τα εξής χαρακτηριστικά:

- *Απλότητα*. Οι εκφράσεις και οι περιγραφές πρέπει να είναι όσο το δυνατόν πιο απλές για να διευκολύνουν τη δουλειά του προγραμματιστή και να μπορούν χρησιμοποιηθούν και από το χρήστη.
- *Επεκτασιμότητα*. Οι παράμετροι του *context* που μπορούν να χρησιμοποιηθούν πρέπει να είναι απεριόριστες και να μπορούν να προστεθούν εύκολα επιπλέον παράμετροι.
- *Εκφραστικότητα*. Το μοντέλο πρέπει να έχει μεγάλες δυνατότητες έκφρασης ώστε να μην θέτει περιορισμούς στις συνθήκες που μπορούν να περιγραφούν.
- *Γενικότητα*. Το μοντέλο πρέπει να είναι αρκετά γενικό, ώστε να μπορεί να συμπεριλάβει όλες τις ιδιότητες της *context* πληροφορίας.
- Επιπλέον η αναπαράσταση πρέπει να είναι *τυπική*, ώστε να εκφράζει με σαφήνεια την πληροφορία.

Αυτόνομη Αρχιτεκτονική

Το ενδιάμεσο λογισμικό που σχεδιάσαμε στοχεύει στην υποστήριξη εφαρμογών που λειτουργούν σε μία κινητή συσκευή. Μια προφανής αρχιτεκτονική για την υπηρεσία είναι μια προσέγγιση πελάτη - εξυπηρέτη. Σε αυτή την αρχιτεκτονική ένα «ελαφρύ» υποσύστημα τρέχει πάνω στην κινητή συσκευή και επικοινωνεί με την εφαρμογή. Ένα άλλο υποσύστημα βρίσκεται σε κάποιο σταθερό κόμβο και αναλαμβάνει τη συλλογή και επεξεργασία του *context*, που αποτελούν σύνθετες υπολογιστικά διαδικασίες επικοινωνώντας με το υποσύστημα που βρίσκεται στην κινητή συσκευή.

Αυτή η προσέγγιση έχει χρησιμοποιηθεί αρκετά στη βιβλιογραφία, γιατί παρέχει λύση στο πρόβλημα της αδυναμίας των κινητών συσκευών να υποστηρίξουν βαριές υπολογιστικές διεργασίες [27,32,55,56,60,65,66,69,72,75,77,80]. Παρόλα αυτά η αρχιτεκτονική με έναν κεντρικό κόμβο παροχής κρίσιμων υπηρεσιών δεν είναι η πλέον κατάλληλη αφού περιορίζει γεωγραφικά το περιβάλλον υπολογισμού σε μικρούς χώρους. Ο χρήστης δεν μπορεί να κινηθεί ελεύθερα αφού η λειτουργικότητα του ενδιάμεσου λογισμικού χάνεται μακριά από τον εξυπηρέτη που επικοινωνεί με τη συσκευή. Επιπλέον σε ένα περιβάλλον που χαρακτηρίζεται από ασταθείς και αναξιόπιστες συνδέσεις και χαμηλό εύρος ζώνης, η επικοινωνία με άλλους κόμβους, μπορεί να σταθεί εμπόδιο στη λειτουργικότητα και την απόδοση του ενδιάμεσου λογισμικού.

Συνεπώς το λογισμικό πρέπει να είναι ανεξάρτητο από κεντρικές μονάδες διαχείρισης και πρέπει να παρέχει υπηρεσίες οπουδήποτε και ανά πάσα στιγμή. Η υπηρεσία πρέπει να λειτουργεί αυτόνομα πάνω στη συσκευή συμβάλλοντας σε ένα πλήρως αποκεντρικοποιημένο *peer – to – peer* περιβάλλον.

Ταυτοχρονισμός

Η εκτέλεση της υπηρεσίας θα πρέπει να είναι ανεξάρτητη από το υπόλοιπο σύστημα και να εκτελείται ταυτόχρονα με άλλα τμήματα λογισμικού. Οι λειτουργίες της εφαρμογής δεν πρέπει να εμποδίζονται από την υπηρεσία και το αντίστροφο. Για το λόγο αυτό, όλες οι υπολογιστικές διεργασίες υπηρεσίας θα πρέπει να εκτελούνται παράλληλα με το υπόλοιπο σύστημα.

Ο ταυτοχρονισμός επιτυγχάνεται με την εκτέλεση της υπηρεσίας σε διαφορετικό νήμα ελέγχου από το υπόλοιπο σύστημα, ώστε να εκτελούνται «*ψευδοπαράλληλα»* στον επεξεργαστή της συσκευής. Λέγοντας ψευδοπαράλληλα εννοούμε ότι, το κάθε νήμα δεν εκτελείται σε διαφορετικό επεξεργαστή, αλλά μοιράζονται τον ίδιο. Η χρήση του επεξεργαστή εναλλάσσεται μεταξύ των νημάτων, τα οποία εκτελούνται για ορισμένο χρονικό διάστημα και μετά διακόπτονται. Η παράλληλη λειτουργία επιτρέπει στην υπηρεσία να εκτελείται με απόλυτη διαφάνεια, χωρίς να επηρεάζει αισθητά, χρονικά ή λειτουργικά, την απόδοση της εφαρμογής.

Χαμηλή Κατανάλωση Υπολογιστικών Πόρων

Το χαμηλό υπολογιστικό κόστος λειτουργίας είναι πάντα επιθυμητό σε συστήματα λογισμικού. Ιδιαίτερα επιθυμητό είναι για λογισμικό που προορίζεται για κινητές συσκευές, όπου οι υπολογιστικοί πόροι είναι περιορισμένοι. Επιπλέον τα αλληλεπιδραστικά χαρακτηριστικά της υπηρεσίας κάνουν ακόμα πιο κρίσιμη την απαίτηση για «ελαφριά» υπολογιστική λειτουργία. Ο ταυτοχρονισμός, ο χειρισμός των εξωτερικών γεγονότων, η λήψη αποφάσεων και οι ενέργειες προσαρμογής επιβαρύνουν υπολογιστικά τους πόρους τους συστήματος.

Η λειτουργικότητα και η απόδοση της εφαρμογής πρέπει να επηρεάζονται το λιγότερο δυνατό από την υπολογιστική επιβάρυνση που προσθέτει η υπηρεσία. Συνεπώς η υπηρεσία θα πρέπει να κάνει οικονομική διαχείριση των απαραίτητων πόρων για το σύστημα, όπως μνήμη, επεξεργαστική ισχύ και αποθηκευτικό χώρο. Η σχεδίαση και η ανάπτυξη της υπηρεσίας θα πρέπει να γίνει με γνώμονα αυτήν την απαίτηση.

Βελτιστοποίηση Απόδοσης

Προκειμένου η υπηρεσία να λειτουργεί σε πραγματικό χρόνο και να υποστηρίζει αποτελεσματικά την εφαρμογή απαιτείται ελάχιστη χρονική απόκριση κατά την εκτέλεσή της. Η μεγιστοποίηση της απόδοσης επιβάλει σχεδιαστικούς αλλά και προγραμματιστικούς περιορισμούς στην ανάπτυξη στης εφαρμογής.

Βασικές αρχές της αντικειμενοστρεφούς σχεδίασης όπως η αφαίρεση και η τμηματοποίηση (*modularity*) είναι απαραίτητες για σύνθετα συστήματα πραγματικού χρόνου, αλλά μπορεί να οδηγήσουν σε σημαντική μείωση της απόδοσης. Ένα λογισμικό με πολλά επίπεδα αφαίρεσης επιφέρει ένα κόστος απόδοσης, που οφείλεται στην αυξημένη επικοινωνία μεταξύ αντικειμένων στα διάφορα επίπεδα, για το χειρισμό μιας κατάστασης.

Το πρόβλημα αυτό μπορεί να διευθετηθεί σε επίπεδο υλοποίησης δημιουργώντας συντομεύσεις στην επικοινωνία μεταξύ των αντικειμένων. Έτσι ένα αντικείμενο μπορεί να έχει πρόσβαση σε ένα άλλο που κανονικά δεν θα του επιτρεπόταν. Επιπλέον μπορεί να γίνει «σύμπτυξη» κλάσεων, ώστε να μειωθεί ο αριθμός των επιπέδων μεταξύ των διαφόρων τμημάτων του λογισμικού.

Αυτές οι λύσεις αυξάνουν τη χρονική απόδοση της υπηρεσίας, αλλά επιφέρουν μείωση της τμηματοποίησης στο λογισμικό. Το λογισμικό αποκτά πιο συμπαγή μορφή και χάνει την ευέλικτη και εύχρηστη δομή του. Η εύρεση της χρυσής τομής μεταξύ τμηματικής σχεδίασης και απόδοσης είναι μια κρίσιμη απαίτηση για την υπηρεσία.

Μεταφερσιμότητα

Η υπηρεσία θα πρέπει να υλοποιηθεί με κατάλληλο λογισμικό ώστε να μπορεί να λειτουργήσει ανεξαρτήτως κινητής συσκευής. Αυτό μέχρι στιγμής δεν είναι εφικτό, καθώς αν και έχουν γίνει προσπάθειες να καθοριστούν κάποιες προδιαγραφές κινητού λογισμικού (*MIDP*), δεν έχουν υιοθετηθεί από όλους τους κατασκευαστές κινητών συσκευών. Παρόλα αυτά η υπηρεσία θα πρέπει να υλοποιηθεί το λογισμικό που θα παρέχει την μεγαλύτερη δυνατή μεταφερσιμότητα.

Αξιοπιστία

Η απαίτηση για αξιοπιστία είναι ιδιαίτερα κρίσιμη, όπως και σε όλα τα συστήματα που παρέχουν υπηρεσίες σε πραγματικό χρόνο. Η αλληλεπίδραση με το περιβάλλον μπορεί να κρύβει απρόβλεπτες καταστάσεις και δημιουργία απροσδόκητων σεναρίων. Η υπηρεσία πρέπει να λειτουργεί υπό οποιεσδήποτε συνθήκες και για κάθε πιθανό σενάριο. Επιπλέον πρέπει να ανταποκρίνεται στα γεγονότα που δέχεται, ανεξάρτητα τον αριθμό και τη σειρά με την οποία πραγματοποιούνται.

Ο ταυτοχρονισμός επιβάλλει επιπρόσθετη πολυπλοκότητα στην υπηρεσία που εύκολα μπορεί να οδηγήσει σε σφάλματα κατά την εκτέλεση. Η υπηρεσία πρέπει να χειρίζεται κατάλληλα την παράλληλη εκτέλεση των τμημάτων της και να αποτρέπει την πραγματοποίηση ανεπιθύμητων καταστάσεων (αδιέξοδα, ασιτία, κρίσιμες καθυστερήσεις). Ακόμα και στην περίπτωση σφαλμάτων ή εξαιρέσεων, ο χειρισμός τους θα πρέπει να γίνεται εσωτερικά από την υπηρεσία, παραμένοντας κάθε στιγμή σε λειτουργία.

Απλότητα

Ο σημαντικότερος ίσως παράγοντας, που κρίνει τη χρηστικότητα ενός αλληλεπιδραστικού συστήματος είναι η ανθρώπινη αντίληψη. Ένα σύστημα που είναι δύσκολο στη κατανόηση, θα χρησιμοποιηθεί λιγότερο από αυτούς που θέλουν να εκμεταλλευτούν τις υπηρεσίες του. Ένα ενδιάμεσο λογισμικό πρέπει να είναι απλό και

κατανοητό στις παροχές του, καθώς και στον τρόπο που μπορεί κάποιος να τις χρησιμοποιήσει. Η απλότητα αφορά τόσο την ενσωμάτωση της υπηρεσίας σε ένα *CA* σύστημα, όσο και την αλληλεπίδραση με το χρήστη και τον προγραμματιστή της εφαρμογής.

ΚΕΦΑΛΑΙΟ 5. ΠΕΡΙΓΡΑΦΗ ΛΕΙΤΟΥΡΓΙΩΝ

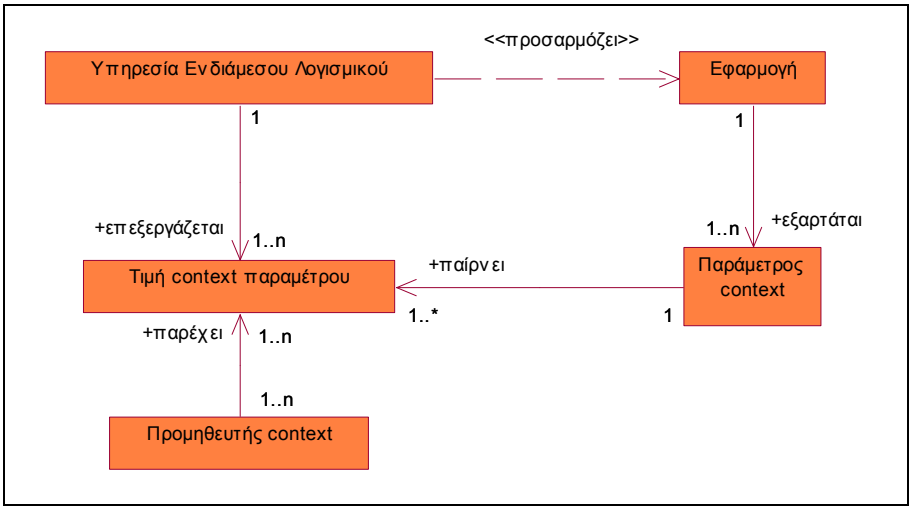
Σε αυτό το κεφάλαιο περιγράφεται πιο αναλυτικά το ενδιάμεσο λογισμικό που υλοποιήσαμε. Αρχικά καθορίζεται ο ρόλος της υπηρεσίας σε ένα *CA* σύστημα. Περιγράφονται οι λειτουργίες της υπηρεσίας, καθώς και οι αλληλεπιδράσεις, με το υπολογιστικό και φυσικό περιβάλλον και τους χρήστες.

5.1 Αλληλεπίδραση Υπηρεσίας με το CA Σύστημα

Η υπηρεσία ενδιάμεσου λογισμικού λειτουργεί αυτόνομα ή σαν τμήμα ενός ευρύτερου *CA* συστήματος. Στοχεύει στη διαχείριση του *context* μιας κινητής εφαρμογής, σε συνεργασία και με άλλα συστατικά λογισμικού ή συσκευές. Μπορεί να λειτουργήσει αυτόνομα, υποστηρίζοντας την *CA* συμπεριφορά της εφαρμογής χωρίς την συνδρομή επιπλέον υποσυστημάτων. Στο Σχήμα 5.1 φαίνεται η γενική αρχιτεκτονική της υπηρεσίας.

Κάθε *CA* εφαρμογή εξαρτάται από ένα σύνολο παραμέτρων που συνθέτουν το *context* και περιλαμβάνει ένα σύνολο προμηθευτών που παρέχουν τιμές για αυτές τις παραμέτρους. Η υπηρεσία μεσολαβεί μεταξύ της εφαρμογής και των προμηθευτών και διαχειρίζεται το *context*. Οι προμηθευτές επικοινωνούν με την υπηρεσία και την ενημερώνουν για τις αλλαγές στο *context*, παρέχοντας τις νέες τιμές των παραμέτρων. Η υπηρεσία κάνει επεξεργασία των τιμών και εξάγει συμπεράσματα για την αντίδραση της εφαρμογής. Τα

συμπεράσματα αντιστοιχίζονται σε ενέργειες προσαρμογής της εφαρμογής, αν αυτό είναι απαραίτητο.



Σχήμα 5.1 Γενική Αρχιτεκτονική της Υπηρεσίας.

Στη συνέχεια περιγράφονται οι διαδικασίες που σχετίζονται με τη διαχείριση του *context*. Επιπλέον, αναλύεται η επικοινωνία της υπηρεσίας με τις υπόλοιπες οντότητες του συστήματος και ο τρόπος με τον οποίο συμμετέχουν αυτές, στη διαχείριση του *context*.

5.2 Διαχείριση του Context

Για τη διαχείριση του *context* η υπηρεσία συνεργάζεται με τους προμηθευτές και την εφαρμογή και επιτελεί ένα σύνολο διαδικασιών. Στη συνέχεια περιγράφονται οι βασικές λειτουργίες της υπηρεσίας που σχετίζονται με την εφαρμογή και τους προμηθευτές του *context*.

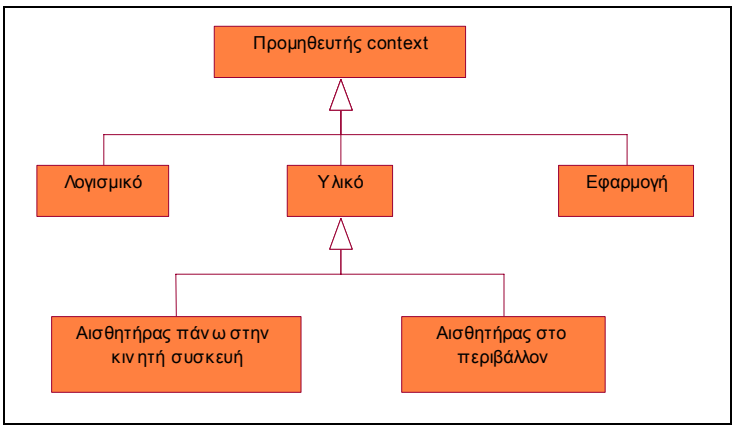
5.2.1 Επικοινωνία Υπηρεσίας - Προμηθευτών

Σε ένα *CA* σύστημα, προμηθευτές είναι όλες οι οντότητες (φυσικές ή εννοιολογικές) οι οποίες παρέχουν τιμές για τις παραμέτρους του *context*. Λόγω της ποικιλομορφίας και της ετερογένειας στην *context* πληροφορία, μια εφαρμογή είναι πιθανό να χρησιμοποιεί ένα

πλήθος διαφορετικών προμηθευτών για τη παροχή του *context*. Συγκεκριμένα ένας προμηθευτής σε ένα *CA* σύστημα μπορεί να είναι:

- Συσκευή στο περιβάλλον που κινείται ο χρήστης, που λειτουργεί σαν αισθητήρας: π.χ. ηλεκτρονικό θερμόμετρο, μετρητής θορύβου, ανιχνευτής κίνησης κ.α.
- Υλικό ενσωματωμένο στην κινητή συσκευή: π.χ. συσκευή που μετράει ταχύτητα, γωνία κλίσης, φωτεινότητα κ.α.
- Λογισμικό που λειτουργεί ανεξάρτητα από την εφαρμογή: π.χ. μία συνάρτηση που επιστρέφει την ώρα ή την ημερομηνία ή ένας δαίμονας που παρακολουθεί τη μνήμη, τη *CPU* ή υπολογίζει το εύρος ζώνης της ασύρματης επικοινωνίας.
- Η ίδια η εφαρμογή παρέχοντας συγκεκριμένη πληροφορία: π.χ. πληροφορίες από ηλεκτρονικό ημερολόγιο, πληροφορίες από βάσεις δεδομένων κ.α. Επιπλέον μέσω της εφαρμογής μπορεί και ο χρήστης, να εισάγει δεδομένα στο σύστημα.

Στο Σχήμα 5.2 φαίνεται μια ιεραρχική κατηγοριοποίηση των προμηθευτών.

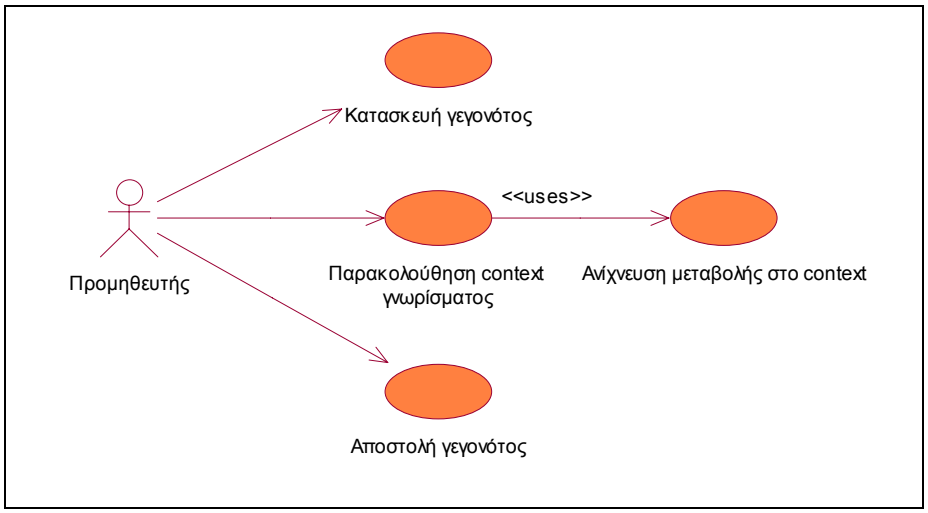


Σχήμα 5.2 Κατηγοριοποίηση Προμηθευτών.

Οι προμηθευτές μεταφέρουν στην υπηρεσία τις τιμές που παράγουν. Η υπηρεσία περιλαμβάνει ένα μηχανισμό ασύγχρονης επικοινωνίας, που βασίζεται σε αποστολή γεγονότων, για τη συλλογή των τιμών του *context* από τους προμηθευτές. Οι τιμές ερμηνεύονται και μετατρέπονται σε γεγονότα τα οποία στέλνονται στην υπηρεσία για επεξεργασία και αποτίμηση.

Η επικοινωνία πραγματοποιείται μονόπλευρα, με τους προμηθευτές μόνο να στέλνουν γεγονότα στην υπηρεσία χωρίς να περιμένουν απάντηση. Για την επικοινωνία χρησιμοποιείται η μέθοδος *εγγραφής-ειδοποίησης*. Η εγγραφή γίνεται από τον προγραμματιστή ο οποίος υλοποιεί ένα σύνδεσμο μεταξύ του προμηθευτή και της υπηρεσίας. Κάθε φορά που συμβαίνει ένα νέο γεγονός αποστέλλεται αυτόματα μεσώ του συνδέσμου στην υπηρεσία. Με αυτόν το μηχανισμό γίνεται αυτόματη ενημέρωση της υπηρεσίας για τις αλλαγές που συμβαίνουν στο *context* ή για τις τρέχουσες τιμές που έχουν τα γνωρίσματά του.

Το Σχήμα 5.3 απεικονίζει τις γενικές λειτουργίες που αφορούν τους προμηθευτές.



Σχήμα 5.3 Λειτουργίες Προμηθευτών.

5.2.2 Επεξεργασία του Context

Κατά την επεξεργασία του *context* η υπηρεσία ελέγχει την πληροφορία που αποκτήθηκε και αποφασίζει αν πρέπει να εκτελέσει ενέργειες προσαρμογής. Η διαδικασία ελέγχου πραγματοποιείται σε δύο στάδια και περιλαμβάνει αποτίμηση της πληροφορίας και έλεγχο της αξιοπιστίας της. Έπειτα προωθούνται κατάλληλες ενέργειες αντίδρασης, αν κριθεί απαραίτητο. Στη συνέχεια περιγράφεται αναλυτικά η φάση της επεξεργασίας.

Αποτίμηση του Context

Τα γεγονότα συλλέγονται από την κεντρική μονάδα της υπηρεσίας για εξαγωγή του *context* και αποτίμηση. Ένα γεγονός απεικονίζει ότι ένα γνώρισμα του *context* έχει μια συγκεκριμένη τιμή, τη συγκεκριμένη χρονική στιγμή. Αυτή η πληροφορία ενδεχομένως να

σημαίνει ότι η εφαρμογή πρέπει να πραγματοποιήσει κάποια ενέργεια. Κάθε φορά που προωθείται ένα γεγονός για επεξεργασία, η υπηρεσία αποφασίζει αν αυτό υποδεικνύει την εκτέλεση ενεργειών που σχετίζονται με την εφαρμογή.

Για τη λήψη αποφάσεων χρησιμοποιείται εκ των προτέρων γνώση για την επιθυμητή συμπεριφορά της εφαρμογής ανάλογα με το *context*. Η γνώση κωδικοποιείται σε ένα σύνολο κανόνων συνθήκης ενέργειας που ακολουθούν το μοντέλο αναπαράστασης του *context*. Οι κανόνες περιγράφουν τις ενέργειες προσαρμογής, που απαιτούνται για συγκεκριμένους συνδυασμούς αναθέσεων τιμών σε παραμέτρους του *context* και σε *συγκεκριμένες χρονικές στιγμές*. Η απόφαση λαμβάνεται εξετάζοντας το σύνολο των κανόνων και ελέγχοντας αν το νέο γεγονός οδηγεί την ικανοποίηση κάποιου κανόνα. Σε περίπτωση ικανοποίησης οι ενέργειες του κανόνα θα πρέπει να εκτελεστούν, εάν αυτός κριθεί αξιόπιστος.

Έλεγχος Αξιοπιστίας

Πριν εκτελεστούν οι ενέργειες ενός κανόνα που ικανοποιείται, πρέπει πρώτα να ελεγχθεί η αξιοπιστία του. Η υπηρεσία παρέχει τη δυνατότητα να συμπεριληφθεί ο παράγοντας αξιοπιστία στην διαδικασία αποτίμησης του *context*.

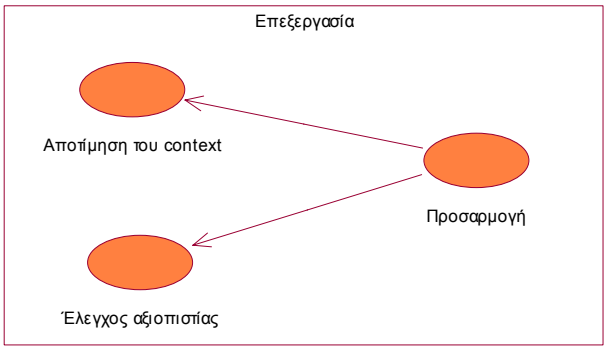
Συγκεκριμένα περιλαμβάνεται μία μονάδα διαχείρισης αξιοπιστίας, που χρησιμοποιεί εκ των προτέρων γνώση για τους προμηθευτές του *context*. Η γνώση αυτή αφορά πιθανότητες, που χαρακτηρίζουν το ενδεχόμενο οι τιμές που λαμβάνονται από κάποιο προμηθευτή να είναι σωστές. Με βάση αυτές τις πιθανότητες, η υπηρεσία αποφασίζει αν τα συμπεράσματα που εξάγονται είναι αξιόπιστα ή όχι.

Η συνολική αξιοπιστία ενός κανόνα υπολογίζεται με βάση την αξιοπιστία των επιμέρους τιμών των παραμέτρων του *context* που συμμετέχουν στον κανόνα. Με αυτόν τον τρόπο αποφεύγονται συμπεράσματα τα οποία βασίζονται σε εσφαλμένα γεγονότα.

Προσαρμογή στις Αλλαγές του Context

Η προσαρμογή της εφαρμογής πραγματοποιείται εκτελώντας το σύνολο των ενεργειών που συμπεραίνονται από τον έλεγχο των κανόνων. Οι ενέργειες εκτελούνται αυτόματα από την υπηρεσία και με απόλυτη διαφάνεια, χωρίς να διακόπτεται η λειτουργία της εφαρμογής.

Στο Σχήμα 5.4 φαίνονται οι λειτουργίες επεξεργασίας του *context*.



Σχήμα 5.4 Λειτουργίες Επεξεργασίας του Context.

5.3 Χρήστες του Λογισμικού

Οι χρήστες που σχετίζονται με την υπηρεσία περιλαμβάνουν:

- Τον προγραμματιστή (ή σχεδιαστή) της εφαρμογής, «κάτω» από την οποία λειτουργεί η υπηρεσία.
- Το χρήστη της κινητής συσκευής στην οποία εκτελείται η εφαρμογή.

Επειδή η υπηρεσία αποτελεί ενδιάμεσο λογισμικό δεν υπάρχει χρήστης που να αλληλεπιδρά άμεσα με την υπηρεσία κατά τη διάρκεια λειτουργίας της. Η αλληλεπίδραση του προγραμματιστή σχετίζεται με τη διασύνδεση της εφαρμογής με την υπηρεσία. Η επαφή του χρήστη με την υπηρεσία είναι έμμεση και γίνεται μέσω της εφαρμογής. Στη συνέχεια περιγράφεται πιο αναλυτικά η αλληλεπίδραση της υπηρεσίας με τους χρήστες.

5.3.1 Αλληλεπίδραση Υπηρεσίας με τον Προγραμματιστή της Εφαρμογής

Ο προγραμματιστής της εφαρμογής έχει την ευθύνη να συνδέσει προγραμματιστικά την εφαρμογή με την υπηρεσία και να προσθέσει κάποια τμήματα κώδικα που καθορίζουν τον τρόπο αλληλεπίδρασης με την εφαρμογή. Η υπηρεσία περιλαμβάνει ένα

προγραμματιστικό πλαίσιο για τη σύνδεση και την προσαρμογή μιας εφαρμογής. Ο προγραμματιστής πρέπει να αναπτύξει επιπλέον λογισμικό για να υλοποιήσει τους συνδέσμους της υπηρεσίας με την εφαρμογή. Συγκεκριμένα υλοποιεί μια προγραμματιστική διεπαφή που ορίζει η υπηρεσία.

Στα επιπρόσθετα τμήματα λογισμικού καθορίζεται η συμπεριφορά της υπηρεσίας με βάση τα λειτουργικά χαρακτηριστικά της εφαρμογής και το *context* από το οποίο εξαρτάται. Τα τμήματα αυτά αποτελούν ουσιαστικά τη διεπαφή μιας εφαρμογής με την υπηρεσία. Όλες οι υπόλοιπες διαδικασίες που αφορούν τη λειτουργία της υπηρεσίας εκτελούνται αυτοματοποιημένα, χωρίς την προσθήκη επιπλέον λογισμικού ή την επίβλεψη του χρήστη. Στη συνέχεια περιγράφονται τα τμήματα διασύνδεσης που υλοποιεί ο προγραμματιστής.

Ορισμός του Context και Σύνθεση Κανόνων Προσαρμογής

Ο προγραμματιστής ορίζει τις παραμέτρους του *context* από τις οποίες εξαρτάται η εφαρμογή και το σύνολο των τιμών που μπορούν να πάρουν. Οι παράγοντες του *context* κωδικοποιούνται σύμφωνα με το μοντέλο αναπαράστασης. Η κωδικοποιημένη μορφή του *context* είναι αυτή που χρησιμοποιείται από την υπηρεσία.

Ευθύνη του προγραμματιστή είναι να καθορίσει και τους κανόνες που περιγράφουν την προσαρμογή της εφαρμογής στο *context*. Η σύνθεση των κανόνων γίνεται σύμφωνα το συντακτικό που επιβάλει το μοντέλο του *context*. Ο προγραμματιστής συνθέτει τους κανόνες με βάση την επιθυμητή συμπεριφορά και τους στόχους της εφαρμογής. Επιπλέον επιλέγει και τον τρόπο εισαγωγής των κανόνων στο σύστημα, υλοποιώντας μία μέθοδο που περιλαμβάνεται στην προγραμματιστική διεπαφή.

Σύνδεση Υπηρεσίας με τους Προμηθευτές

Οι τιμές για κάθε γνώρισμα του *context* παρέχονται από τον αντίστοιχο προμηθευτή. Ο προγραμματιστής είναι υπεύθυνος να υλοποιήσει τις συνδέσεις προμηθευτών - υπηρεσίας. Η σύνδεση αποτελεί ουσιαστικά την εγγραφή της υπηρεσίας, ώστε να ενημερώνεται για τις τιμές του *context* που παρέχει ο προμηθευτής. Συγκεκριμένα η εγγραφή περιλαμβάνει την υλοποίηση δύο τμημάτων: τον καθορισμό *πολιτικών ενημέρωσης* και την *κατασκευή γεγονότων*.

Οι πολιτικές ενημέρωσης χρησιμοποιούνται για να δείξουν στον προμηθευτή πότε πρέπει να ενημερώσει για την τιμή ενός γνωρίσματος του *context*. Λειτουργούν σαν *φίλτρα γεγονότων*, καθορίζοντας πότε η αλλαγή στην τιμή ενός γνωρίσματος θεωρείται «αρκετά σημαντική», ώστε να αποτελέσει ένα γεγονός και να σταλεί προς αποτίμηση. Πιθανές πολιτικές ενημέρωσης που μπορούν να υλοποιηθούν ανάλογα με τις ανάγκες της εφαρμογής και το είδος του προμηθευτή είναι οι εξής:

- Ένας προμηθευτής στέλνει γεγονότα σε σταθερά, μεταβλητά ή τυχαία χρονικά διαστήματα.
- Ένα γεγονός υποδεικνύεται όταν υπάρχει σταθερού ή μεταβλητού μεγέθους μεταβολή στην τιμή ενός γνωρίσματος.
- Όταν η τιμή ενός γνωρίσματος πάρει ή ξεπεράσει συγκεκριμένα κατώφλια.
- Ένα γεγονός στέλνεται κάθε φορά που ένα γνώρισμα παίρνει καινούργια τιμή.

Η κατασκευή ενός γεγονότος καθορίζει τη μορφή που παίρνει η τιμή του αντίστοιχου γνωρίσματος ώστε να μπορεί να την επεξεργαστεί η υπηρεσία. Η μετατροπή γίνεται με βάση τον προμηθευτή που παρήγαγε την τιμή και σύμφωνα με το μοντέλο αναπαράστασης του *context*. Με αυτόν τον τρόπο πραγματοποιείται η ενθυλάκωση της ετερογενούς πληροφορίας σε μία αφαιρετική μορφή αναπαράστασης.

Εισαγωγή Παραμέτρων Ποιότητας

Ο προγραμματιστής (ή/και ο χρήστης) έχει τη δυνατότητα να εκφράσει πιθανοτικά την πεποίθησή του, ότι οι τιμές από έναν προμηθευτή είναι εσφαλμένες. Ο χαρακτηρισμός της αξιοπιστίας των προμηθευτών γίνεται με την εισαγωγή ενός συνόλου πιθανοτήτων στην υπηρεσία. Οι πιθανότητες αυτές εκφράζουν την ποιότητα της πληροφορίας που παράγουν οι προμηθευτές. Για την καταχώρηση των πιθανοτήτων, η υπηρεσία παρέχει ένα σύνολο μεθόδων που μπορεί να χρησιμοποιήσει ο προγραμματιστής.

Ο έλεγχος αξιοπιστίας φυσιολογικά επιφέρει επιπλέον υπολογιστικό κόστος στην εκτέλεση της υπηρεσίας. Ο χρήστης έχει τη δυνατότητα να απενεργοποιήσει τη λειτουργία ελέγχου αν το επιθυμεί. Η απενεργοποίηση μπορεί να γίνει είτε επειδή όλοι οι προμηθευτές θεωρούνται αξιόπιστοι ή διότι η απόδοση του συστήματος κρίνεται πιο σημαντική από την πιθανότητα λανθασμένης αποτίμησης ενός κανόνα.

Αντιστοίχιση Συμπερασμάτων και Ενεργειών

Στο τελευταίο κομμάτι της διασύνδεσης ο προγραμματιστής δίνει πραγματική υπόσταση στις ενέργειες προσαρμογής. Οι ενέργειες που εξάγονται από τον έλεγχο των κανόνων πραγματοποιούνται με την εκτέλεση ενός τμήματος λογισμικού της εφαρμογής. Ο προγραμματιστής συσχετίζει κάθε λειτουργία της εφαρμογής με μία ενέργεια, η οποία πιθανώς να συμπεριληφθεί στο σύνολο των κανόνων. Με αυτήν τη συσχέτιση υλοποιείται η αυτόματη προσαρμογή της εφαρμογής κάθε φορά που αποφασίζεται μία ενέργεια.

Η αντιστοίχιση συμπερασμάτων - ενεργειών αφορά τις ενέργειες που περιλαμβάνονται στο σύνολο των κανόνων. Προληπτικά, είναι προτιμότερο να γίνει εξ αρχής αντιστοίχιση όλων των λειτουργιών της εφαρμογής σε κάποια ενέργεια. Έτσι μπορεί να προστεθεί δυναμικά κάποιος κανόνας με νέες ενέργειες, χωρίς να χρειαστεί να γίνουν αλλαγές στο ενδιάμεσο λογισμικό.

Προσθήκη GUI για τη Διαμόρφωση της Υπηρεσίας από το Χρήστη

Η υπηρεσία παρέχει έτοιμο ένα *GUI* που επιτρέπει τη διαμόρφωσή της από το επίπεδο εφαρμογής. Ο προγραμματιστής μπορεί να συμπεριλάβει αυτό το *GUI* στο μενού της εφαρμογής. Μέσω αυτού του *GUI* ο χρήστης μπορεί δυναμικά να διαμορφώσει τη λειτουργία της υπηρεσίας από την οθόνη της κινητής του συσκευής. Η χρήση αυτής της διεπαφής δεν είναι απαραίτητη για την *CA* λειτουργία της εφαρμογής, σε αντίθεση με τις τέσσερις προηγούμενες διαδικασίες διασύνδεσης. Παρόλα αυτά είναι ιδιαίτερα σημαντικό να μπορεί ο χρήστης να προσαρμόζει τη λειτουργία της εφαρμογής άμεσα, δηλώνοντας τις προτιμήσεις του.

5.3.2 Αλληλεπίδραση Χρήστη - Υπηρεσίας

Η λειτουργία της υπηρεσίας είναι αόρατη στο χρήστη της εφαρμογής και η επικοινωνία υπηρεσίας - εφαρμογής είναι απόλυτα διαφανής. Παρόλα αυτά ο χρήστης μπορεί να διαμορφώσει άμεσα τη λειτουργία της υπηρεσίας (συνεπώς έμμεσα την εφαρμογή) μέσω του *GUI* που παρέχει η υπηρεσία. Ο χρήστης μπορεί να κωδικοποιήσει τις προτιμήσεις

του σε κανόνες και να τους καταχωρήσει στην υπηρεσία. Στη συνέχεια περιγράφονται τα χαρακτηριστικά της υπηρεσίας στα οποία έχει πρόσβαση ο χρήστης.

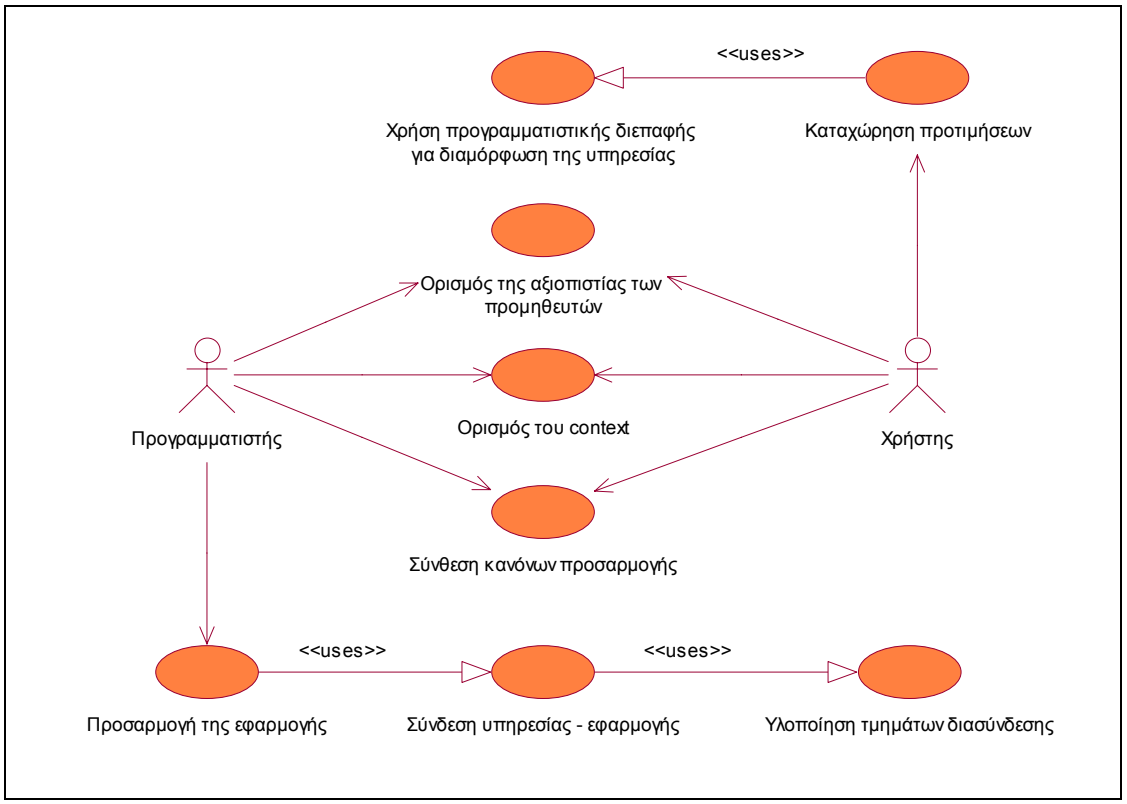
Καταχώρηση Προτιμήσεων και Διαμόρφωση Κανόνων

Οι κανόνες ορίζονται πριν την έναρξη της εφαρμογής από το χρήστη ή/και τον προγραμματιστή και καταχωρούνται στο ενδιάμεσο λογισμικό. Ο χρήστης έχει τη δυνατότητα να συνθέσει κανόνες της προτίμησής του και να τους καταχωρήσει δυναμικά στην υπηρεσία. Η υπηρεσία παρέχει το *GUI* για εισαγωγή, διαγραφή και αντικατάσταση όλων των κανόνων σε πραγματικό χρόνο. Ο χρήστης μπορεί από την οθόνη της συσκευής του να διαμορφώσει το σύνολο των κανόνων σύμφωνα με τις ανάγκες και τις προτιμήσεις του.

Διαμόρφωση Ελέγχου Αξιοπιστίας

Ο χρήστης μπορεί να έχει πρόσβαση και στη μονάδα ελέγχου αξιοπιστίας. Οι πιθανότητες που περιγράφουν την ποιότητα της πληροφορίας που παράγουν οι προμηθευτές μπορεί να μεταβάλλονται χρονικά. Ο χρήστης μπορεί εισάγει τις νέες πιθανότητες δυναμικά από την οθόνη της κινητής συσκευής. Η υπηρεσία παρέχει ένα ακόμα *GUI*, για την αλλαγή των πιθανοτήτων ποιότητας. Επιπλέον υπάρχει η δυνατότητα απενεργοποίησης της λειτουργίας ελέγχου.

Στο Σχήμα 5.5 φαίνεται το διάγραμμα χρήσης της υπηρεσίας από τους χρήστες του *CA* συστήματος. Όπως φαίνεται στο Σχήμα 5.5, ο χρήστης της κινητής εφαρμογής και ο προγραμματιστής της υπηρεσίας δεν έχουν απόλυτα διαχωρισμένες αρμοδιότητες. Με την υπάρχουσα τεχνολογία όλο και περισσότεροι χρήστες μπορούν να αναπτύξουν λογισμικό και να το εγκαταστήσουν στην κινητή συσκευή τους.



Σχήμα 5.5 Διάγραμμα Χρήσης της Υπηρεσίας από τον Προγραμματιστή και το Χρήστη της Συσκευής.

ΚΕΦΑΛΑΙΟ 6. ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΤΟΥ CONTEXT

Στο κεφάλαιο αυτό περιγράφεται το μοντέλο που χρησιμοποιείται για τη διαχείριση του *context* και δίνονται παραδείγματα μοντελοποίησης. Εισάγεται η έννοια του χρονικά εξαρτημένου *context* και τα χαρακτηριστικά του μοντέλου που σχετίζονται με το χρόνο. Τέλος περιγράφεται η χρήση του μοντέλου για τη διαχείριση του *context*.

6.1 Ορισμός του Context σε Ένα CA Σύστημα

Η υπηρεσία αναπαριστά και διαχειρίζεται το *context* σύμφωνα με ένα καθορισμένο μοντέλο. Το *context* σχετίζεται με το *CA* σύστημα (*CAS*), στο οποίο συμμετέχει η υπηρεσία. Στην πιο απλή περίπτωση, το *CAS* αποτελείται από την υπηρεσία, την εφαρμογή που τρέχει από πάνω της και πιθανώς κάποιους εξωτερικούς προμηθευτές. Συνολικά, για το *context* που διαχειρίζεται η υπηρεσία έχουμε κάνει τις εξής θεωρήσεις:

- Ένα *CAS* αποτελείται από ένα πεπερασμένο σύνολο αρχιτεκτονικών στοιχείων *E* και ένα πεπερασμένο σύνολο οντοτήτων (φυσικών ή εννοιολογικών) *D* σχετικών με το σύστημα. Μια οντότητα μπορεί να είναι ένας χρήστης ή κάποιο άλλο σύστημα.
- Ένα *CAS* περιλαμβάνει ένα πεπερασμένο σύνολο προμηθευτών *P*. Σε κάθε αρχιτεκτονικό στοιχείο E_i και σε κάθε οντότητα D_i αντιστοιχούν ένας ή παραπάνω προμηθευτές P_i.
- Το *context C* σε ένα *CAS* αποτελείται από ένα πεπερασμένο σύνολο παραμέτρων $C = \{c_1, c_2, \ldots c_k\}$.

- Κάθε παράμετρος c_i παίρνει τιμές από ένα σύνολο τιμών V_i. Το V_i μπορεί είναι υποσύνολο του συνόλου των πραγματικών αριθμών R ή ένα πεπερασμένο σύνολο συμβολοσειρών S.
- Κάθε παράμετρος c_i σχετίζεται με κάποιο αρχιτεκτονικό στοιχείο E_i ή κάποια οντότητα D_i και ο αντίστοιχος προμηθευτής P_i παράγει τιμές από το σύνολο V_i, για την παράμετρο c_i (Σχήμα 6.1).
- Σε κάθε προμηθευτή P_i αντιστοιχούν τρεις παράμετροι που χαρακτηρίζουν την ποιότητα των τιμών που παράγει. Συγκεκριμένα οι παράμετροι που συνθέτουν την ποιότητα των τιμών που παράγει ένας προμηθευτής είναι οι εξής:

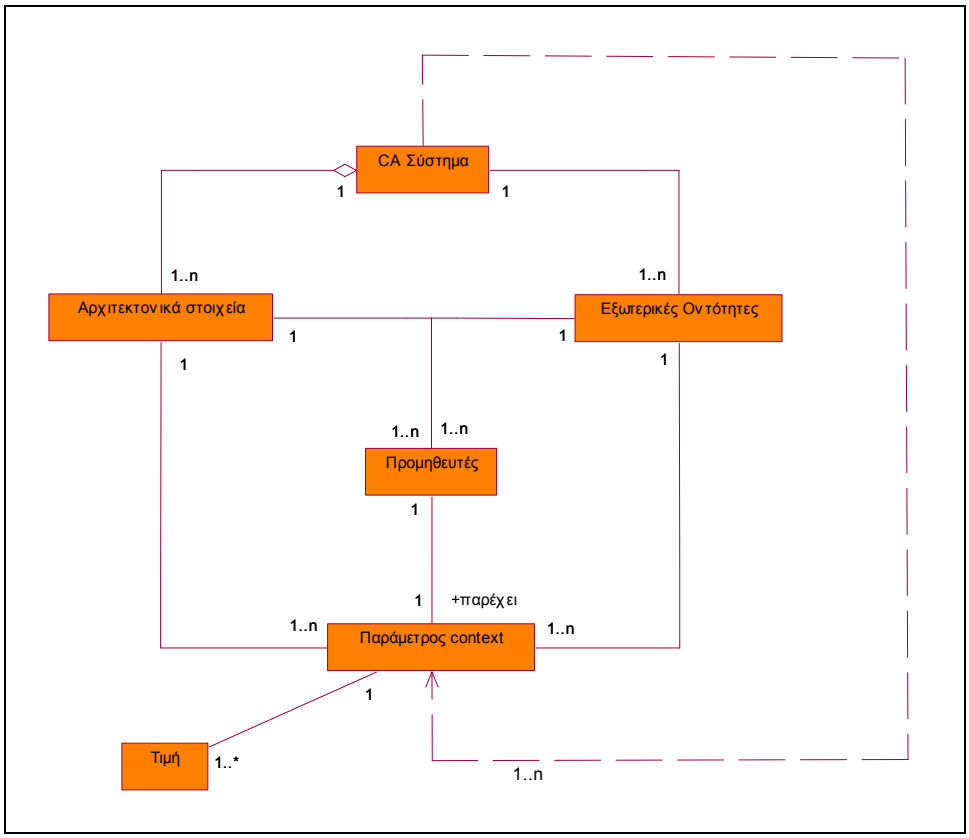
1. *Αναξιοπιστία*: Η πιθανότητα ένας προμηθευτής να παράγει λάθος τιμές, λόγω τυχαίου σφάλματος.
2. *Ανασφάλεια*: Η πιθανότητα ένας προμηθευτής να παράγει λάθος τιμές, που προκαλούνται από κακόβουλες παρεμβολές.
3. *Ανακρίβεια*: Το ποσοστό ακριβείας που χάνεται από την πραγματική τιμή της ιδιότητας κατά τη μέτρηση του προμηθευτή. Ο λόγος ανακρίβειας είναι: $/$*πραγματική τιμή – τιμή προμηθευτή* $/ /$*πραγματική τιμή*.

Αυτές οι παράμετροι ποιότητας καλύπτουν τις απαιτήσεις για αξιοπιστία στο μοντέλο του *context*. Διάφορες επιπλέον παράμετροι έχουν προταθεί για την περιγραφή της ποιότητας του *context* [11].

Στο Σχήμα 6.2 φαίνεται η αρχιτεκτονική ενός *CAS* που ακολουθεί το παραπάνω μοντέλο. Το συγκεκριμένο *CAS* περιλαμβάνει την υπηρεσία διαχείρισης του *context* και μία εφαρμογή. Τα σκούρα τετράγωνα αναπαριστούν τα μέλη της υπηρεσίας, ενώ τα ανοιχτά αναπαριστούν τα μέλη της εφαρμογής.

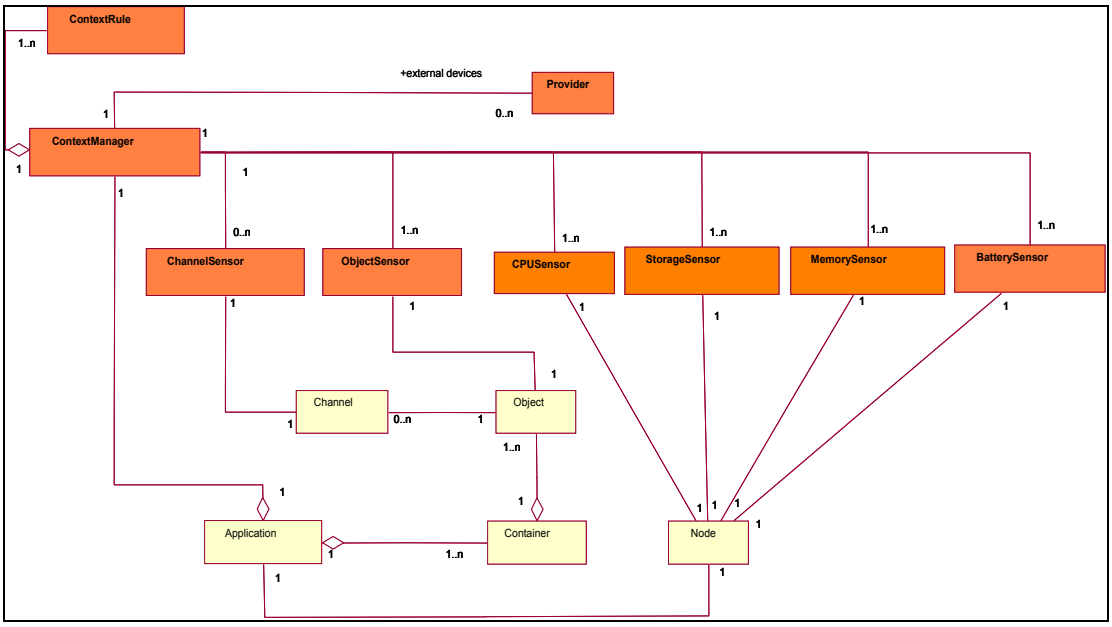
Η εφαρμογή αποτελείται από ένα σύνολο από *containers*, οι οποίοι μοιράζονται τους ίδιους υπολογιστικούς και αποθηκευτικούς πόρους. Η εφαρμογή εκτελείται σε μία συσκευή (*node*), η οποία παρέχει τους πόρους αυτούς. Ένας *container* περιλαμβάνει ένα σύνολο αντικειμένων. Τα αντικείμενα αυτά παρέχουν λειτουργίες, που μπορούν να χρησιμοποιηθούν για την πρόσβαση και τη μετατροπή της καταστάσεώς τους. Αντικείμενα που βρίσκονται σε άλλες εφαρμογές μπορούν να επικοινωνήσουν μέσω καναλιών ενδιάμεσου λογισμικού.

Τα παραπάνω δομικά συστατικά της εφαρμογής σχετίζονται με ένα σύνολο προμηθευτών. Κάθε δομικό συστατικό σχετίζεται με έναν ή περισσότερους προμηθευτές, οι οποίοι παρέχουν τιμές για κάποια χαρακτηριστικά του συστατικού. Τα χαρακτηριστικά αυτά αποτελούν τις παραμέτρους του *context*.



Σχήμα 6.1 Μοντελοποίηση Ενός CAS.

Όλοι οι προμηθευτές του συστήματος είναι αισθητήρες υλικού ή λογισμικού. Συγκεκριμένα ένα αντικείμενο σχετίζεται με έναν αισθητήρα αντικειμένου, ένα κανάλι με έναν αισθητήρα καναλιού κ.τ.λ. Οι προμηθευτές επικοινωνούν με την υπηρεσία για την αποστολή των τιμών των παραμέτρων του *context*. Στον Πίνακα 6.1 φαίνονται κάποιες παράμετροι του *context* για ένα *CAS*. Ο Πίνακας 6.1 περιλαμβάνει ένα σύνολο *context* παραμέτρων, τα δομικά συστατικά του συστήματος που σχετίζονται με αυτές, τις τιμές που λαμβάνουν και το είδος του προμηθευτή που τις παράγει.

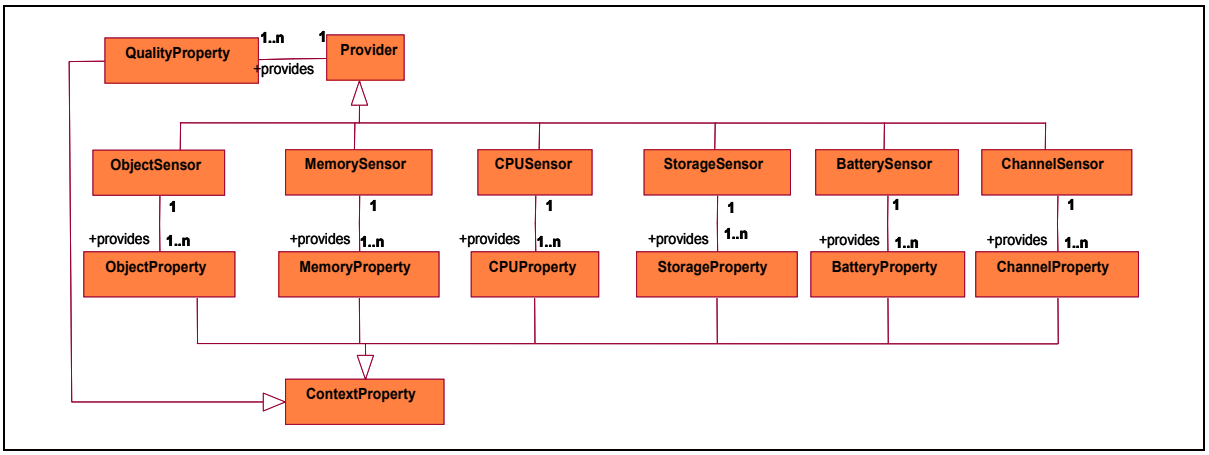


Σχήμα 6.2 Αρχιτεκτονική Ενός CAS που Χρησιμοποιεί την Υπηρεσία Διαχείρισης του Context.

Πίνακας 6.1 Παραδείγματα Παραμέτρων του Context για Ένα CAS.

Αρχιτεκτονική Μονάδα	**Παράμετρος *Context***	**Τύπος**	**Προμηθευτής**
Node	CPURate	Float	**CPUSensor**
	CPULoad	Float	
	MemTotal	Int	**MemorySensor**
	MemAvail	Int	
	StorTotal	Int	**StorageSensor**
	StorAvail	Int	
	BattTotalTime	Int	**BatterySensor**
	BattAvailTime	Int	
	Mode	enum{economy, normal}	
Channel	ChanDelay	Int	**ChannelSensor**
	ChanDropRate	Int	
	ChanSecurity	enum{high, low, med}	
Object	Persistence	enum{ON, OFF}	**ObjectSensor**
	Transactions	enum{FLAT, NESTED, OFF}	
	Replication	enum{OFF, PASS, ACT, SEMI}	
	NumOfRepl	Int	
Provider	Unreliability	0..1	**Provider**
	Inaccuracy	0..1	
	Insecurity	0..1	

Ανάλογα με την εφαρμογή και το *context* από το οποίο εξαρτάται, θα μπορούσαν να υπάρχουν και εξωτερικοί προμηθευτές, όπως αισθητήρες εντοπισμού γεωγραφικής θέσης, αισθητήρες θερμότητας κ.α. Επιπλέον θεωρούμε ότι κάθε προμηθευτής σχετίζεται με τρεις παραμέτρους αξιοπιστίας. Όπως φαίνεται στο Σχήμα 6.3, κάθε προμηθευτής *context* επεκτείνει ένα βασικό στοιχείο *Provider*, το οποίο σχετίζεται με τις τρεις παραμέτρους που χαρακτηρίζουν την ποιότητα των τιμών που παρέχονται από τους προμηθευτές.



Σχήμα 6.3 Ταξινόμηση Προμηθευτών και Context Παραμέτρων.

Όπως φαίνεται στον Πίνακα 6.1, το *context* για το *CAS* του Σχήματος 6.1 μοντελοποιείται ως εξής:

- Το σύνολο *E* περιλαμβάνει τις αρχιτεκτονικές μονάδες {*Node*, *Channel*, *Object*}.
- Το σύνολο *P* αποτελείται από τους αισθητήρες που φαίνονται στη στήλη των προμηθευτών του Πίνακα 6.1.
- Το σύνολο *C* αποτελείται από τις παραμέτρους που φαίνονται στη στήλη των παραμέτρων του Πίνακα 6.1 και οι τιμές των συνόλων V_i αντιστοιχούν στους τύπους της κάθε παραμέτρου.
- Στο συγκεκριμένο παράδειγμα δε συμπεριλάβαμε κάποια παράμετρο c_i, που να σχετίζεται με στοιχείο του συνόλου *D*. Το σύνολο *D* θα μπορούσε να περιέχει το χρήστη και να σχετίζεται με τις παραμέτρους που χαρακτηρίζουν τη διάθεσή του.

6.2 Αναπαράσταση του Context

Το μοντέλο αναπαράστασης που χρησιμοποιούμε για το *context* βασίζεται στην κατηγορηματική λογική πρώτης τάξεως. Το *context* περιγράφεται σαν ένα σύνολο από *λογικούς κανόνες* και *γεγονότα*. Ένας κανόνας αποτελείται από *εκφράσεις* λογικά συνδεδεμένες μεταξύ τους.

Γεγονότα

Τα γεγονότα αναπαριστούν τις αλλαγές στο *context*. Ένα γεγονός αντικατοπτρίζει την καινούργια τιμή σε μία παράμετρο του *context*. Τα γεγονότα μεταφέρουν την καινούργια γνώση στο σύστημα. Στα συστήματα διαχείρισης γνώσης, η γνώση διακρίνεται σε στατική (*facts*) και δυναμική (*events*). Η στατική γνώση αφορά πληροφορίες που δεν αλλάζουν με το πέρασμα του χρόνου, π.χ. το έτος γέννησης ενός ατόμου. Η δυναμική γνώση αφορά «πράγματα» που συμβαίνουν κάποια χρονική στιγμή. Στο μοντέλο που χρησιμοποιούμε η πληροφορία για το *context* είτε είναι στατική, είτε δυναμική, αναπαρίσταται με γεγονότα που προέρχονται από κάποιο προμηθευτή.

Εκφράσεις

Αναπαριστούν συνθήκες ή ενέργειες. Συσχετίζουν μία παράμετρο του *context* με μία τιμή. Σε έναν κανόνα, ένα σύνολο συνθηκών που ικανοποιούνται, συνεπάγεται ένα σύνολο ενεργειών που πρέπει να εκτελεστούν.

Κανόνες

Περιγράφουν τον τρόπο με τον οποίο το σύστημα θα πρέπει να αντιδράσει στις αλλαγές του *context*. Αποτελούνται από τη λογική σύνδεση συνθηκών και ενεργειών.

Οι κανόνες αποτελούν τη γνώση που έχει η υπηρεσία, προκειμένου να γνωρίζει πως θα προσαρμόσει αυτόματα την εφαρμογή στις αλλαγές του *context*. Οι αλλαγές του *context* αναπαρίστανται με γεγονότα, τα οποία συλλέγονται από την υπηρεσία. Στη συνέχεια περιγράφεται η δομή των κανόνων και ο τρόπος που χρησιμοποιούνται από την υπηρεσία.

6.2.1 Κανόνες Προσαρμογής

Ένας κανόνας αποτελείται από δύο κομμάτια: το κομμάτι των συνθηκών και το κομμάτι των ενεργειών. Τα δύο κομμάτια χωρίζονται μεταξύ τους με συνεπαγωγή, π.χ.: *(A && B) || C* $\Rightarrow$ *D && E*[2].

Το κομμάτι των συνθηκών αποτελείται από συνθήκες που συνδέονται μεταξύ τους με σύζευξη ή διάζευξη. Μία συνθήκη αποτελείται από μία παράμετρο του *context* συγκριτικά συσχετισμένη με μία τιμή. Μια συνθήκη ικανοποιείται όταν, η παράμετρος του *context* της συνθήκης πάρει μια συγκεκριμένη τιμή, *μια συγκεκριμένη χρονική στιγμή*. Ο τύπος της τιμής εξαρτάται από την παράμετρο και μπορεί να είναι ακέραιος, δεκαδικός ή συμβολοσειρά.
Π.χ. (*CPULoad < 30.67 || BattAvailTime< 50) && (Location = Office)*.

Το κομμάτι των ενεργειών αποτελείται από ένα σύνολο ενεργειών που συνδέονται μεταξύ τους μόνο με λογικές συζεύξεις. Μια ενέργεια αποτελείται, είτε από την ανάθεση μιας αριθμητικής τιμής σε μία παράμετρο του *context* ή από μία συμβολοσειρά που δηλώνει την εκτέλεση μίας λειτουργίας. Η ανάθεση μιας τιμής σε μία παράμετρο του *context* ισοδυναμεί με ένα νέο γεγονός για την υπηρεσία. Η χρησιμοποίηση μόνο συζεύξεων για τις ενέργειες έχει την έννοια ότι, μια ενέργεια που αποφασίστηκε θα πρέπει πάντα να εκτελείται.
Π.χ. *(Mode = economy) && (ChanSecurity = high) && (freememory)*.

Το μοντέλο που χρησιμοποιούμε ονομάζεται *ECA* (*Event Condition Action*) και είχε προταθεί για χρήση σε ενεργές βάσεις δεδομένων [90]. Έχει χρησιμοποιηθεί για την υλοποίηση αλληλεπιδραστικών συστημάτων [91] και σε *CA* συστήματα για την προσαρμογή εφαρμογών στο *context* [92].

Οι λογικοί κανόνες πρώτης τάξεως μπορούν να αναπαραστήσουν, λογικές συσχετίσεις μεταξύ των παραμέτρων του *context*. Παρόλα αυτά δεν μπορούν να περιγράψουν τις χρονικές εξαρτήσεις του *context*. Τα γεγονότα που συνθέτουν το *context* είναι χρονικά εξαρτημένα και η διάσταση του χρόνου θα πρέπει να συμπεριληφθεί στη διαχείριση της

[2] Για το συμβολισμό της λογικής σύζευξης και της λογικής διάζευξης, χρησιμοποιήθηκαν τα σύμβολα && και || αντίστοιχα και όχι τα κλασικά $\wedge$ και $\vee$. Τα σύμβολα αυτά προτιμήθηκαν διότι είναι αυτά που χρησιμοποιήθηκαν και στην υλοποίηση της υπηρεσίας.

πληροφορίας. Στη συνέχεια περιγράφεται, πως ενσωματώνεται η έννοια του χρόνου στο μοντέλο διαχείρισης του *context*.

6.3 Χρονική Μοντελοποίηση

Πολλές φορές η τιμή μιας παραμέτρου του *context* δεν είναι αρκετή για να χρησιμοποιηθεί από ένα *CAS*. Αυτό που θα ήταν χρήσιμο, είναι η προσθήκη χρονικής πληροφορίας, για το πότε πήρε αυτή η παράμετρος τιμή. Τα γεγονότα πραγματοποιούνται κατά τη λειτουργία της εφαρμογής και ο χρόνος πραγματοποίησης τους καθορίζει τον τρόπο που επηρεάζουν την εφαρμογή. Λέγοντας ότι η *context* πληροφορία είναι χρονικά εξαρτημένη εννοούμε ότι:

«*Το CAS δεν επηρεάζεται μόνο από τις μεταβολές του context, αλλά και από το πως σχετίζονται οι μεταβολές αυτές, με το χρόνο του συστήματος και τη σειρά πραγματοποίησής τους*.»

Σε αρκετές περιπτώσεις είναι χρήσιμο να συμπεριλάβουμε τη διάσταση του χρόνου και στην εκτέλεση των ενεργειών προσαρμογής. Οι ενέργειες μεταβάλουν την κατάσταση του *CAS* και πρέπει να περιγραφούν χρονικά προκειμένου να επιτύχουν τα επιθυμητά αποτελέσματα.

Προκειμένου να περιγράψουμε τις χρονικές εξαρτήσεις του *context* ενσωματώνουμε στο μοντέλο μας κάποια χαρακτηριστικά από την *Modal Χρονική Λογική Πρώτης Τάξεως* (*First Order Modal Temporal Logic* ή *MTL*). Η *MTL* είναι μια επέκταση της *modal* λογικής με την προσθήκη χρονικών τελεστών [93]. Οι χρονικοί τελεστές εμπεριέχονται στους κανόνες και προσδιορίζουν χρονικά συνθήκες και ενέργειες. Στον Πίνακα 6.2 φαίνονται όλοι οι τελεστές που χρησιμοποιούνται στο συντακτικό των κανόνων[3].

Στο μοντέλο του *context* ενσωματώνεται η έννοια του σχετικού και όχι του πραγματικού χρόνου. Οι χρονικοί τελεστές διατάσσουν τα γεγονότα με βάση τη σειρά που έφτασαν στην υπηρεσία. Οι χρονικές εξαρτήσεις των γεγονότων κωδικοποιούνται στους

[3] Λόγω περιορισμών στα διαθέσιμα σύμβολα δεν ήταν δυνατόν να χρησιμοποιήσουμε τους γνωστούς συμβολισμούς της *MTL* (◊, • κ.α.) και επιλέξαμε την αναπαράσταση που φαίνεται στον Πίνακα 6.2.

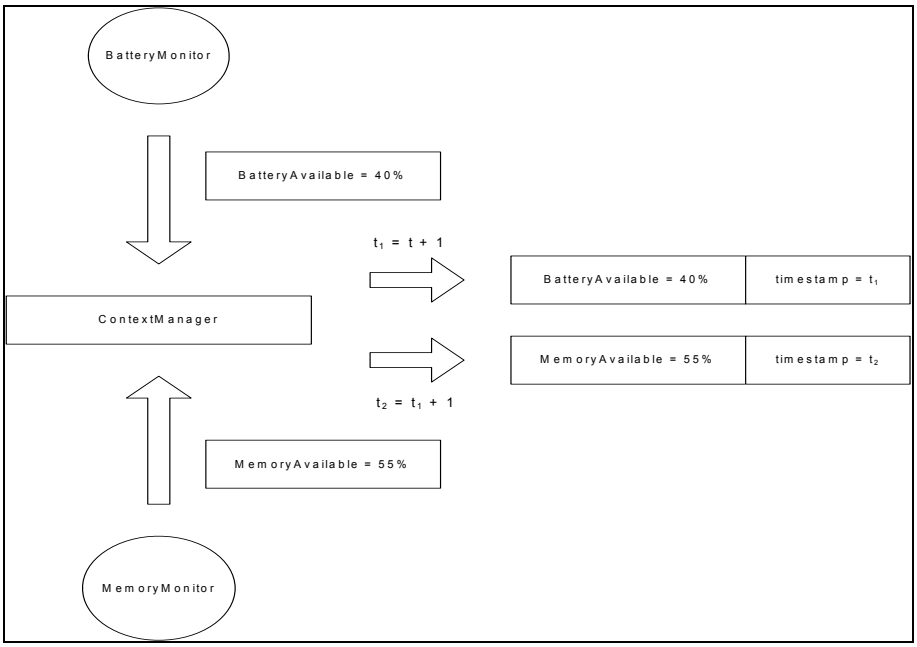
κανόνες με τη χρήση των χρονικών τελεστών και προσθέτουν τη διάσταση του χρόνου στην περιγραφή του *context*.

Πίνακας 6.2 Τελεστές που Χρησιμοποιούνται στο Συντακτικό των Κανόνων.

Τελεστές	
Λογικοί τελεστές	**||, &&, ⇒**
Τελεστές σύγκρισης	**=, <, >, <=, >=, !=**
Παρελθοντικοί τελεστές	***[previous]***
	[once]
	[always_been]
Μελλοντικοί τελεστές	***[next]***
	[until]
	[henceforth]

Για τη χρονική μοντελοποίηση του συστήματος κάνουμε τις εξής θεωρήσεις:

- Κάθε χρονική στιγμή το σύστημα χαρακτηρίζεται από μία κατάσταση (ή στιγμιότυπο) *t*.
- Όταν ξεκινά η λειτουργία του συστήματος: $t = 0$.
- Κάθε φορά που φτάνει ένα καινούργιο γεγονός στην υπηρεσία, το σύστημα περνάει σε μια νέα κατάσταση: $t = t + 1$.
- Όλα τα γεγονότα είναι χρονικά διατεταγμένα μεταξύ τους. Δηλαδή αν το γεγονός *x* πραγματοποιήθηκε τη χρονική στιγμή t_1 και το γεγονός *y* τη χρονική στιγμή t_2 θα ισχύει: $(t_1 < t_2) \;||\; (t_1 > t_2)$. Δύο γεγονότα μπορεί να πραγματοποιήθηκαν ταυτόχρονα σε πραγματικό χρόνο, αλλά η διαχείρισή τους γίνεται σειριακά. Η υπηρεσία θεωρεί ότι το ένα συνέβη πριν από το άλλο, ανάλογα με τη σειρά με την οποία διαχωρίζονται από αυτή (Σχήμα 6.4).



Σχήμα 6.4 Μεταβολή της Χρονικής Καταστάσεως του Συστήματος.

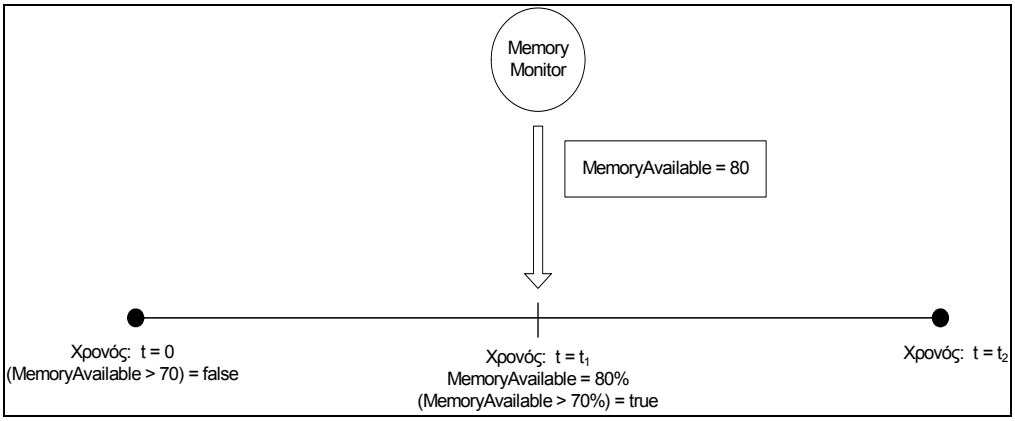
Στη συνέχεια περιγράφεται η σημασιολογία των χρονικών τελεστών με βάση τις παραπάνω θεωρήσεις.

Παρελθοντικοί Τελεστές

Οι παρελθοντικοί τελεστές προσδιορίζουν χρονικά την ικανοποιησιμότητα μιας συνθήκης. Ο ορισμός της ικανοποιησιμότητας για μία συνθήκη, που *δεν* συνοδεύεται από χρονικό τελεστή είναι ο εξής:

Μία συνθήκη ισχύει σε μία χρονική στιγμή t αν, η παράμετρος του context που συμμετέχει στη συνθήκη έχει τη χρονική στιγμή t τιμή, η οποία ικανοποιεί τη συνθήκη.

Κατά την έναρξη λειτουργίας του συστήματος, τη χρονική στιγμή $t = 0$, καμία συνθήκη δεν είναι αληθής. Για να ικανοποιηθεί μία συνθήκη πρέπει, η υπηρεσία να λάβει ένα γεγονός, το οποίο να δίνει μία τιμή, στην παράμετρο του *context* της συνθήκης, που να την ικανοποιεί (Σχήμα 6.5).



Σχήμα 6.5 Ικανοποίηση Συνθήκης μία Χρονική Στιγμή *t*.

Για μία συνθήκη *A*, η σημασιολογία των παρελθοντικών τελεστών είναι η εξής:

- *[previous]*

  [previous] A = true τη χρονική στιγμή *t*, αν η συνθήκη *A* ήταν αληθής τη χρονική στιγμή *t – 1*. Ο τελεστής *[previous]* χρησιμοποιείται για να δηλώσει ότι μία συνθήκη πρέπει να ίσχυε στην προηγούμενη κατάσταση του συστήματος, ανεξάρτητα με το αν ισχύει τώρα. Σε συγκεκριμένες περιπτώσεις μπορεί να χρησιμοποιηθεί για να δηλώσει ότι, η ικανοποίηση μίας συνθήκης προηγήθηκε από την ικανοποίηση μίας άλλης.

- *[once]*

  [once] A = true τη χρονική στιγμή *t*, αν η συνθήκη *A* ικανοποιήθηκε κάποια χρονική στιγμή t_1 και $0 < t_1 < t$. Ο τελεστής *[once]* χρησιμοποιείται για να δηλώσει ότι μία συνθήκη πρέπει να έχει ικανοποιηθεί τουλάχιστον μία φορά, από την έναρξη λειτουργίας της εφαρμογής.

- *[always_been]*

  [always_been] A = true τη χρονική στιγμή *t*, αν η συνθήκη *A* ικανοποιήθηκε κάποια χρονική στιγμή t_1, $0 < t_1 < t$ και στο διάστημα (t_1, t) δεν έχει γίνει πότε *false*. Ο τελεστής *[always_been]* χρησιμοποιείται για να δηλώσει ότι η παράμετρος του *context* δεν έχει πάρει ποτέ τιμή, η οποία να μην ικανοποιεί τη συνθήκη.

Μελλοντικοί Τελεστές

Οι μελλοντικοί τελεστές προσδιορίζουν χρονικά την πραγματοποίηση μιας ενέργειας. Μια ενέργεια αντιστοιχεί στην εκτέλεση μιας λειτουργίας. Αν η ενέργεια συμβολίζεται με την ανάθεση τιμής σε μια παράμετρο του *context*, τότε η λειτουργία που εκτελείται θέτει την τιμή στην παράμετρο του *context*. Σε αυτή την περίπτωση η ενέργεια αποτελεί γεγονός για την υπηρεσία. Τα γεγονότα που προκύπτουν από την εκτέλεση κάποιας ενέργειας ονομάζονται *εσωτερικά*. Αν η ενέργεια συμβολίζεται μόνο από μία συμβολοσειρά, η συμβολοσειρά αντιστοιχεί στην εκτέλεση μιας λειτουργίας, που δεν αναθέτει τιμή σε κάποια παράμετρο.

Για μία ενέργεια *A*, η σημασιολογία των μελλοντικών τελεστών είναι η εξής:

- *[next]*

  Ο τελεστής *[next]* καθορίζει τη σειρά με την οποία θα πραγματοποιηθούν οι ενέργειες: π.χ. *[next] A && B*, δηλώνει ότι θα γίνει πρώτα η ενέργεια *B* και μετά η *A*. Ο τελεστής *[next]* μας δίνει τη δυνατότητα να καθορίσουμε τη σειρά εκτέλεσης παραπάνω από δύο ενεργειών, π.χ. αν θέλουμε να εκτελεστεί πρώτα η ενέργεια *A* μετά η *B* και μετά η *C*, το τμήμα ενεργειών θα γραφεί ως εξής: *A && [next](B && [next] C)*.

- *[until]*

  Η πρόταση *A [until] B*, δηλώνει ότι η ενέργεια *A* θα ισχύει στο μέλλον μέχρι να ικανοποιηθεί η συνθήκη *B*. Πρακτικά, η παράμετρος του *context* της ενέργειας *A*, δεν θα αλλάξει τιμή από άλλη ενέργεια, μέχρι να ικανοποιηθεί η συνθήκη *B*. Μπορεί όμως, όπως θα δούμε παρακάτω, να αλλάξει τιμή απ' ευθείας από το χρήστη.

- *[henceforth]*

  [henceforth] A, δηλώνει ότι η ενέργεια *A* θα πραγματοποιηθεί τώρα και οι συνέπειές της θα ισχύουν συνέχεια στο μέλλον. Πρακτικά αυτό σημαίνει ότι η παράμετρος του *context* στην οποία θα ανατεθεί μία τιμή δεν πρόκειται ποτέ να αλλάξει τιμή στο μέλλον, μέσω άλλης ενέργειας από την υπηρεσία. Μπορεί όμως να αλλάξει τιμή από επιλογή του χρήστη.

Η απουσία χρονικού τελεστή σε μία συνθήκη δηλώνει ότι, η ορθότητα της συνθήκης ελέγχεται την τρέχουσα χρονική στιγμή. Η απουσία χρονικού τελεστή σε μία ενέργεια δηλώνει ότι, η ενέργεια εκτελείται χωρίς περαιτέρω περιορισμούς και προηγείται αυτών που συνοδεύονται από τελεστή *[next]*.

Στις παραπάνω περιγραφές κάναμε σιωπηρά τη θεώρηση ότι οι παρελθοντικοί τελεστές χρησιμοποιούνται μόνο στο κομμάτι των συνθηκών, ενώ οι μελλοντικοί μόνο σε αυτό των ενεργειών. Οι λόγοι είναι οι εξής:

- Μια συνθήκη που προσδιορίζεται από μελλοντικό τελεστή ικανοποιείται σε κάθε χρονική στιγμή t, αν ικανοποιείται σε κάποια χρονική στιγμή $t_1 > t$ στο μέλλον. Αυτό δεν είναι δυνατό να το γνωρίζει η υπηρεσία, ώστε να αποτιμήσει θετικά ή αρνητικά τη συνθήκη σε κάποια χρονική στιγμή t.

- Μια ενέργεια έχει νόημα να εκτελεστεί αμέσως ή σε κάποια μελλοντική χρονική στιγμή. Ο προσδιορισμός μιας ενέργειας με παρελθοντικό τελεστή δηλώνει κάτι πρακτικά μη εφαρμόσιμο.

6.4 Σύνταξη Κανόνων

Η υπηρεσία δεν μπορεί να επεξεργαστεί κανόνες με αυθαίρετη μορφή. Οι κανόνες που διαχειρίζεται ακολουθούν ένα συγκεκριμένο συντακτικό. Η επιβολή συγκεκριμένου συντακτικού έγινε για λόγους απλότητας, αλλά και απόδοσης. Στη συνέχεια περιγράφεται το συντακτικό των κανόνων και οι λόγοι που επιλέχθηκε.

6.4.1 Τμήμα Συνθηκών

Το τμήμα συνθηκών, μπορεί να ακολουθεί *Κανονική Συζευκτική Μορφή* (*Conjunctive Normal Form* ή *CNF*) ή *Κανονική Διαζευκτική μορφή* (*Disjunctive Normal Form* ή *DNF*). Το συντακτικό του τμήματος συνθηκών φαίνεται στην Εικόνα 6.1.

$$\text{CNF}: \ R = \wedge_{i=1}^{n} C_i, \ C_i = \left(\vee_{j=1}^{m} c_j \right)$$
$$\text{DNF}: \ R = \vee_{i=1}^{n} D_i, \ D_i = \left(\wedge_{j=1}^{m} c_j \right)$$
$$C_i = < \text{past operator} > \left(\wedge_{j=1}^{m} c_j \right)$$
$$D_i = < \text{past operator} > \left(\vee_{j=1}^{m} c_j \right)$$
$$c_{i,j} = < \text{simple\_condition} >$$

Εικόνα 6.1 Συντακτικό του Τμήματος Συνθηκών των Κανόνων.

Το τμήμα συνθηκών ενός κανόνα είναι σε *CNF*, όταν αποτελείται από τη σύζευξη ενός ή περισσότερων σύνθετων συνθηκών (C_i), όπου κάθε σύνθετη συνθήκη αποτελείται από τη διάζευξη ενός ή περισσότερων απλών συνθηκών (c_j). Τα C_i ονομάζονται *clauses* του κανόνα σε Κανονική Συζευκτική Μορφή. Το τμήμα συνθηκών ενός κανόνα είναι σε *DNF* όταν αποτελείται από τη διάζευξη ενός ή περισσότερων σύνθετων συνθηκών (D_i), όπου κάθε σύνθετη συνθήκη αποτελείται από τη σύζευξη ενός ή περισσοτέρων απλών συνθηκών. Τα D_i ονομάζονται *implicants* του κανόνα σε Κανονική Διαζευκτική Μορφή. Τα $c_{i,j}$ ονομάζονται *literals* του κανόνα σε *CNF* ή *DNF* μορφή.

Αν το τμήμα συνθηκών αποτελείται από τη σύζευξη σύνθετων συνθηκών και μία σύνθετη συνθήκη συνοδεύεται από χρονικό τελεστή, τότε οι απλές συνθήκες της σύνθετης συνθήκης *μπορούν* να συνδέονται με συζεύξεις. Δηλαδή μπορεί να υπάρξει τμήμα συνθηκών της μορφής *A && [previous] (B && C)*. Σε αυτήν την περίπτωση ο κανόνας δεν είναι σε *CNF* ή *DNF* μορφή και η σύνθετη συνθήκη *(B && C)* δεν μπορεί να χαρακτηριστεί *clause* ή *implicant*.

Αντίστοιχα, αν το τμήμα των συνθηκών αποτελείται από τη διάζευξη σύνθετων συνθηκών και μία σύνθετη συνθήκη συνοδεύεται από χρονικό τελεστή, τότε οι απλές συνθήκες της σύνθετης συνθήκης *μπορούν* να συνοδεύονται με διαζεύξεις. Δηλαδή μπορεί να υπάρξει τμήμα συνθηκών της μορφής *A || [previous] (B || C)*. Ομοίως και σε αυτήν την περίπτωση ο κανόνας δεν είναι σε *CNF* ή *DNF* μορφή και η σύνθετη συνθήκη (*B || C*) δεν μπορεί να χαρακτηριστεί *clause* ή *implicant*.

Οι εκφράσεις της μορφής: *A && [previous] (B && C)* και *A || [previous] (B || C)* έχουν νόημα μόνο με την ύπαρξη των χρονικών τελεστών, αφού: *A &&(B && C) ≡ (A &&B && C)* και *A || (B || C) ≡ (A || B || C)*. Επιπλέον εκφράσεις της

μορφής *A && (B && ¬A)* δεν έχουν νόημα, αφού τα *A* και *¬A* δεν μπορούν να είναι ποτέ ταυτόχρονα αληθή και συνεπώς η έκφραση δεν μπορεί να ικανοποιηθεί ποτέ. Αν όμως η σύνθετη συνθήκη συνοδεύεται από χρονικό τελεστή, η έκφραση *A && [previous] (B && ¬A)* έχει νόημα, αφού απαιτείται τα *A* και *¬A*, να ικανοποιούνται σε διαφορετικές χρονικές στιγμές.

Σύμφωνα με θεμελιώδες θεώρημα της λογικής, οποιαδήποτε λογική έκφραση μπορεί να γραφεί ισοδύναμα σε μία σε Κανονική Συζευκτική Μορφή και σε μία σε Κανονική Διαζευκτική Μορφή [94]. Το θεώρημα αυτό μας εξασφαλίζει ότι το συντακτικό που χρησιμοποιούμε μπορεί να περιγράψει οποιαδήποτε λογική έκφραση.

6.4.2 Τμήμα Ενεργειών

Το τμήμα των ενεργειών ενός κανόνα είναι λιγότερο σύνθετο, αφού αποτελείται μόνο από συζεύξεις ενεργειών. Σύνθετες εκφράσεις μπορούν να δημιουργηθούν όταν υπάρχουν πολλές ενέργειες που πρέπει να εκτελεστούν με συγκεκριμένη σειρά. Σε αυτήν την περίπτωση μπορεί να δημιουργηθούν σύνθετες ενέργειες όταν, περισσότερες από μία ενέργειες συνοδεύονται από έναν τελεστή *[next]*, π.χ.: *[next] (A && B && C)*. Στην περίπτωση που μία ενέργεια, μέλος μιας σύνθετης ενέργειας, συνοδεύεται από τελεστή *[next]*, η ενέργεια αυτή θα πρέπει να εκτελεστεί έπειτα από τα υπόλοιπα μέλη, π.χ.: *[next] (A && [next] B && C)*.

6.4.3 Αξιολόγηση Συντακτικού

Το συντακτικό των κανόνων σχεδιάστηκε με βασικό γνώμονα την απλότητα. Όσο πιο απλό είναι το συντακτικό, τόσο περισσότερο μειώνεται η πολυπλοκότητα της επεξεργασίας του *context* από την υπηρεσία. Περιττοί τελεστές που δεν προσέφεραν κάτι παραπάνω στην εκφραστικότητα του μοντέλου απορρίφθηκαν για χάρη της απόδοσης. Στη συνέχεια περιγράφονται κάποιες βασικές περικοπές που έγιναν στο μοντέλο αναπαράστασης και οι λόγοι για τους οποίους απορρίφθηκαν.

- Ο λογικός τελεστής *not* δεν συμπεριλήφθηκε διότι, η άρνηση μίας συνθήκης μπορεί εύκολα να αντικατασταθεί από τη συμπληρωματική της, π.χ.: *[not] (A > B)* ⇔ *A <= B, [not] (A < B)* ⇔ *A >= B, [not] (A = B)* ⇔ *A != B* κτλ.

- Έχουν προταθεί και άλλοι χρονικοί τελεστές στη βιβλιογραφία οι οποίοι δεν προσφέρουν πρακτικά παραπάνω εκφραστικές δυνατότητες στο μοντέλο, π.χ.: ο τελεστής *[eventually]*.

- Το λογικό *OR* δεν χρησιμοποιείται στο κομμάτι των ενεργειών, αφού δεν έχει ιδιαίτερη χρησιμότητα. Η διάζευξη ενεργειών θα απαιτούσε και μία λογική επιλογής, μίας από τις διαζευγμένες ενέργειες προς εκτέλεση.

- Θα μπορούσε να συμπεριληφθεί στο μοντέλο η έννοια του πραγματικού χρόνου με τη χρήση ενός ακόμα τελεστή *A(t)*, όπου *t* μια χρονική σταθερά. Η ανάγκη ύπαρξης ενός τέτοιου τελεστή εξαρτάται από την εκάστοτε εφαρμογή. Γενικά, αφήνεται στην ευχέρεια του σχεδιαστή της εφαρμογής να προσθέσει μία επιπλέον συνθήκη *context* η οποία να περιέχει τον πραγματικό χρόνο για κάποια παράμετρο, π.χ. *IncomingCallTime = 16,* όπου *16* μπορεί να συμβολίζει την ώρα *16:00*.

- Η σύνθεση των κανόνων γίνεται σύμφωνα με συγκεκριμένο συντακτικό που περιγράφηκε. Η υπηρεσία δεν μπορεί να ερμηνεύσει κανόνες αυθαίρετης μορφής, καθώς κάτι τέτοιο θα έκανε πολύ δύσκολη και χρονοβόρα τη συντακτική ανάλυση, αλλά και τη διαδικασία αποτίμησης. Ο περιορισμός αυτός δεν μειώνει τις εκφραστικές δυνατότητες του μοντέλου, καθώς ιδιαίτερα σύνθετες λογικές εκφράσεις δεν έχουν πρακτική χρησιμότητα, ενώ μπορούν εύκολα να μετατραπούν σε μορφή συμβατή με του συντακτικού των κανόνων. Στην Εικόνα 6.2 φαίνεται το συντακτικό των κανόνων σε *BNF* περιγραφή.

Στην Εικόνα 6.3 φαίνονται δύο παραδείγματα κανόνων *context* που ακολουθούν το παραπάνω συντακτικό. Οι κανόνες αυτοί περιγράφουν δύο προτάσεις οι οποίες δεν θα μπορούσαν να εκφραστούν χωρίς τη χρήση χρονικών τελεστών.

```
<rule> ::= <conditional_part> ⇒ <action_part>

<conditional_part> ::=  <c_statement> | <d_statement>

<c_statement> ::= <past_op>(<c_condition>) <log_op> | <c_statement> |
<past_op>(<c_condition>)

<d_statement> ::= <past_op>(<d_condition>) <log_op> | <d_statement> |
<past_op>(<d_condition>)

<c_condition> ::= <op_condition> || | <c_condition> | <op_condition>
<d_condition> ::= <op_condition> && | <d_condition> | <op_condition>

<op_condition> ::= (<past_op> <condition>)
<condition> ::= <context_property> <comp_op> <value>
<past_op> ::= [previous] | [once] | [always_been] | ε

<action_part> ::= <a_statement>
<a_statement> ::= <action> && <a_statement> | <action> && [next]
(<a_statement>) | <action>

<action>::= ( <future_op> <context_property> = <value> ) |
[next] ( <context_property> = <value> [until] <condition> ) |
<context_property>
<future_op>::=  [next] | [henceforth] | ε

<log_op> ::= && | ||
<comp_op>::= > | < | <= | = | >= | !=
<context_property> ::= set of context properties
<value>::= type of context property
```

Εικόνα 6.2 Το Συντακτικό των Κανόνων σε BNF Περιγραφή.

$R1 = [\text{previous}]\begin{pmatrix}\text{channel.Security} = \text{low} \;\&\&\\ \text{node.MemAvail} > 70\end{pmatrix} \&\& \; (\text{node.MemAvail} < 20) \Rightarrow [\text{henceforth}]\,(\text{channel.Security} = \text{high})$	$R2 = (\text{node.BattAvailTime} <= 60) \Rightarrow \left(\begin{array}{l}\text{object}_1.\text{Replication} = \text{OFF} \;\&\&\\ [\text{next}]\begin{pmatrix}\text{object}_1.\text{Persistence} = \text{OFF} \;\&\&\\ [\text{next}](\text{node.Mode} = \text{economy})\end{pmatrix}\end{array}\right)$
(α)	(β)

Εικόνα 6.3 Παραδείγματα Κανόνων.

Ο κανόνας της Εικόνας 6.3(α) αναφέρει ότι:

- αν παρατηρηθεί πτώση της διαθέσιμης μνήμης κάτω από το *20%* της συνολικής και
- την προηγούμενη χρονική στιγμή η διαθέσιμη μνήμη ήταν πάνω από το *70%* της συνολικής και η ασφάλεια του καναλιού επικοινωνίας ήταν απενεργοποιημένη,
- τότε είναι πολύ πιθανό το σύστημα να έχει δεχθεί κάποια δικτυακή επίθεση που καταναλώνει τους υπολογιστικούς του πόρους.
- Αν ο κανόνας ικανοποιηθεί κάποια χρονική στιγμή, τότε θα πρέπει να ενεργοποιηθεί η ασφάλεια του καναλιού και να μείνει ενεργοποιημένη για πάντα.

Με τη χρήση του τελεστή *[previous]* στο συγκεκριμένο παράδειγμα επιτυγχάνουμε να καθορίσουμε τη σειρά με την οποία θα πρέπει να ικανοποιηθούν οι δύο συνθήκες (η απλή και η σύνθετη), ώστε να ικανοποιηθεί όλος ο κανόνας. Χωρίς τη χρήση του τελεστή *[previous]* ο κανόνας αυτός δεν θα μπορούσε να περιγραφεί, αφού οι συνθήκες $node.MemAvail > 70$ και $node.MemAvail < 20$ δεν μπορούν να ισχύουν ποτέ ταυτόχρονα.

Ο κανόνας της Εικόνας 6.3(β) αναφέρει ότι:

- αν ο διαθέσιμος χρόνος της μπαταρίας της συσκευής πέσει κάτω από τη μία ώρα, τότε για το αντικείμενο $object_1$:
- θα πρέπει πρώτα να απενεργοποιηθεί η ιδιότητα *Replication* του αντικειμένου,
- στη συνέχεια να απενεργοποιηθεί η ιδιότητα *Persistence*,
- και τέλος η λειτουργία της συσκευής να τεθεί σε οικονομικό μοτίβο.

Οι τρεις αυτές ενέργειες θα έχουν σαν αποτέλεσμα λιγότερη κατανάλωση και εξοικονόμηση μπαταρίας. Με τη χρήση του τελεστή *[next]* μπορούμε να διατάξουμε της σειρά εκτέλεσης αυτών των ενεργειών και επιβάλουμε την απενεργοποίηση του *Replication* πριν από το *Persistence*. Αν οι ενέργειες εκτελούνταν με αντίθετη σειρά, θα μπορούσε να προκληθεί σφάλμα, αν η ιδιότητα *Replication* εξαρτιούνταν από την ιδιότητα *Persistence*.

Περισσότερα παραδείγματα κανόνων περιγράφονται στο Κεφάλαιο 8, όπου αναλύεται η κινητή εφαρμογή που υλοποιήσαμε πάνω από την υπηρεσία και οι συγκεκριμένοι κανόνες που χρησιμοποιήσαμε.

6.5 Αξιολόγηση Μοντέλου

Σε αυτό το σημείο θα ήταν χρήσιμο να κάνουμε μία αξιολόγηση του μοντέλου σύμφωνα με τις απαιτήσεις που είχαμε θέσει στο Κεφάλαιο 4.

Απλότητα

Το μοντέλο είναι αρκετά απλό και μπορεί να χρησιμοποιηθεί και από χρήστες ελάχιστης εμπειρίας. Οι κανόνες δημιουργούνται από απλές εκφράσεις, ενώ τα ονόματα των παραμέτρων ορίζονται από το χρήστη ή τον προγραμματιστή. Σύνθετοι ή δυσνόητοι τελεστές δε χρησιμοποιούνται και οι κανόνες μπορούν εύκολα να περιγραφούν με φυσική γλώσσα.

Επεκτασιμότητα

Το μοντέλο δεν περιορίζει το πλήθος των παραμέτρων του *context* που μπορούν να χρησιμοποιηθούν και τις τιμές που μπορούν να πάρουν. Ο χρήστης μπορεί εύκολα να ορίσει ονόματα για καινούργιες παραμέτρους του *context* και να τα χρησιμοποιήσεις απ' ευθείας στους κανόνες.

Εκφραστικότητα

Οι λογικοί και χρονικοί τελεστές παρέχουν επαρκή εκφραστικότητα στο μοντέλο. Μπορούν να περιγράψουν σύνθετες εκφράσεις, οι οποίες περιλαμβάνουν και χρονική πληροφορία.

Γενικότητα

Το μοντέλο είναι αρκετά γενικό, καθώς μπορεί να χρησιμοποιηθεί για να περιγράψει οποιαδήποτε μορφή *context*.

Τυπική Περιγραφή

Η αναπαράσταση του *context* είναι τυπική, χωρίς να δημιουργεί ασάφειες στην ερμηνεία των εκφράσεων. Επιπλέον η χρήση των χρονικών τελεστών περιγράφει τυπικά τη χρονική συσχέτιση των γεγονότων.

ΚΕΦΑΛΑΙΟ 7. ΑΡΧΙΤΕΚΤΟΝΙΚΟΣ ΣΧΕΔΙΑΣΜΟΣ

Σε αυτό το κεφάλαιο περιγράφεται η αρχιτεκτονική της υπηρεσίας. Αναλύονται οι δομικές μονάδες της υπηρεσίας, καθώς και ο τρόπος που επικοινωνούν μεταξύ τους. Για κάθε υποσύστημα επισημαίνονται τα ιδιαίτερα χαρακτηριστικά και ο ρόλος του στη διαχείριση του *context*. Επιπλέον περιγράφονται οι διεπαφές και οι μηχανισμοί επικοινωνίας μεταξύ των υποσυστημάτων.

7.1 Αρχιτεκτονική Υπηρεσίας

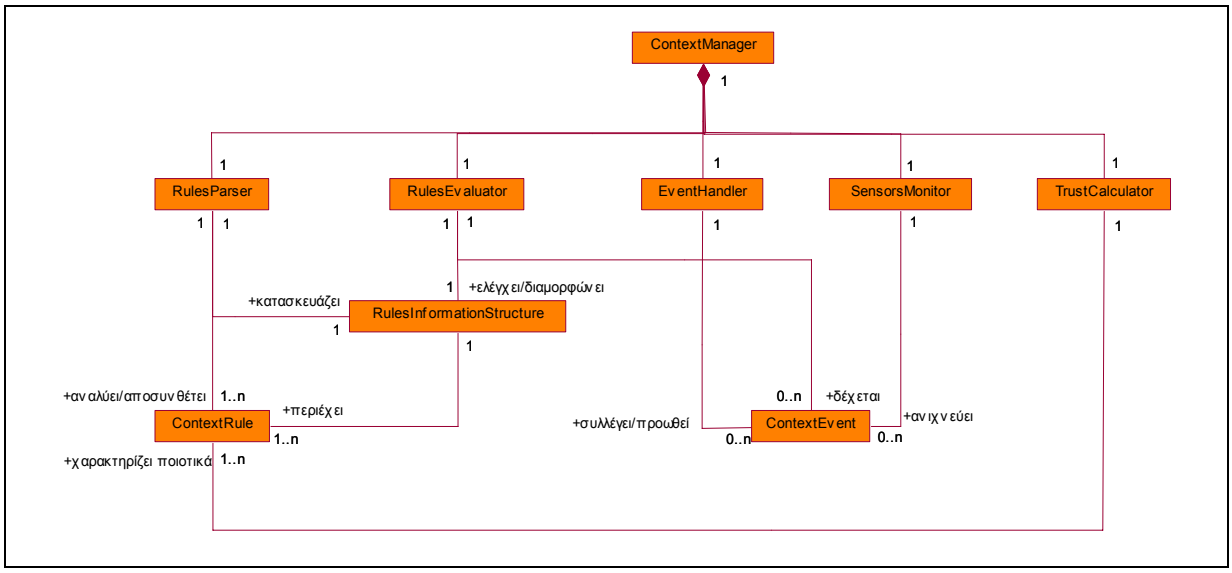
Η υπηρεσία παρέχεται με τη μορφή ενός υποσυστήματος λογισμικού που λειτουργεί παράλληλα με την εφαρμογή. Περιλαμβάνει ένα σύνολο δομικών μονάδων που υλοποιούν τις διαδικασίες διαχείρισης του *context*. Τα συστατικά της υπηρεσίας είναι τα εξής:

- *ContextManager*. Είναι η κεντρική μονάδα της υπηρεσίας. Μεσολαβεί μεταξύ της εφαρμογής και των υπόλοιπων υποσυστημάτων και αρχικοποιεί την υπηρεσία.
- *RulesParser*. Αναλύει συντακτικά τους κανόνες και αποθηκεύει τα περιεχόμενα τους. «Κατασκευάζει» τη γνώση του συστήματος και την κάνει διαθέσιμη στα υπόλοιπα υποσυστήματα.
- *SensorsMonitor*. Ανιχνεύει αλλαγές στις παραμέτρους του *context*, των οποίων οι τιμές μεταβάλλονται με μεγάλη συχνότητα. Παρακολουθεί τους αντίστοιχους προμηθευτές και εντοπίζει αξιοσημείωτες αλλαγές στις τιμές που παράγουν.

- *EventHandler*. Συλλέγει τις τιμές του *context* και κατασκευάζει γεγονότα. Μεσολαβεί μεταξύ των προμηθευτών και της μονάδας επεξεργασίας των γεγονότων.
- *RulesEvaluator*. Επεξεργάζεται τα γεγονότα και αποφασίζει την εκτέλεση ενεργειών. Προσαρμόζει τη συμπεριφορά της εφαρμογής και της υπηρεσίας και διαμορφώνει τη γνώση του συστήματος.
- *TrustCalculator*. Υπολογίζει την αξιοπιστία των κανόνων βασιζόμενος στην ποιότητα των τιμών που παρέχουν οι προμηθευτές.

Εκτός από τα υποσυστήματα που συμμετέχουν στη διαχείριση του *context*, η υπηρεσία περιλαμβάνει και ένα συστατικό που αφορά τη δυναμική διαμόρφωση της. Ο *RulesRecordManager* είναι το υποσύστημα που παρέχει στην εφαρμογή μεθόδους και ένα *GUI*, για την τροποποίηση των κανόνων και των παραμέτρων ποιότητας κατά τη λειτουργία του συστήματος.

Η αρχιτεκτονική της υπηρεσίας φαίνεται στο Σχήμα 7.1. Η υπηρεσία παρέχει επιπλέον μία προγραμματιστική διεπαφή για τη διασύνδεσή της με την εφαρμογή. Η διεπαφή υλοποιείται από τον προγραμματιστή και εγκαθίστανται οι σύνδεσμοι της επικοινωνίας υπηρεσίας – εφαρμογής. Η διεπαφή αυτή περιγράφεται λεπτομερώς στο Κεφάλαιο 8. Στη συνέχεια αναλύονται τα δομικά συστατικά της υπηρεσίας και περιγράφονται τα ιδιαίτερα χαρακτηριστικά τους.



Σχήμα 7.1 Αρχιτεκτονική Υπηρεσίας.

7.2 Κεντρική Μονάδα Διαχείρισης

Ο *ContextManager* είναι η κεντρική μονάδα της υπηρεσίας. Επικοινωνεί απ' ευθείας με την εφαρμογή και συντονίζει τις διαδικασίες των υπολοίπων. Καλείται από την εφαρμογή, παίρνει τις εισόδους της υπηρεσίας και αρχικοποιεί τα υπόλοιπα τμήματα. Οι είσοδοι της υπηρεσίας είναι οι κανόνες του *context*, οι παράμετροι ποιότητας των προμηθευτών και τα κατώφλια αξιοπιστίας των κανόνων.

7.2.1 Αρχικοποίηση Υποσυστημάτων

Κατά την έναρξη της εφαρμογής καλείται ο *ContextManager* και εκτελεί τις ακόλουθες ενέργειες:

- Καλεί τον *RulesParser* με είσοδο τους κανόνες, για να γίνει η συντακτική ανάλυση.
- Καλεί τον *EventHandler* και αρχικοποιείται η ουρά αναμονής των γεγονότων.
- Καλεί τον *SensorsMonitor* για να ξεκινήσει η παρακολούθηση των προμηθευτών και η συλλογή γεγονότων.

Επίσης, αν επιλεγεί από το χρήστη καλεί τον *TrustCalculator* και του περνάει τις παραμέτρους ποιότητας των προμηθευτών, για να γίνει ο υπολογισμός της αξιοπιστίας των κανόνων.

7.2.2 Πέρασμα Κανόνων

Η υπηρεσία δίνει τη δυνατότητα στον προγραμματιστή της εφαρμογής να επιλέξει τον τρόπο φόρτωσης κανόνων. Η διαδικασία φόρτωσης της εισόδου ελέγχεται από τον *ContextManager*. Οι κανόνες μπορούν να δοθούν με τους ακόλουθους τρόπους:

1. Μέσα στον κώδικα της εφαρμογής.
2. Με ένα αρχείο από εγγραφές, όπου κάθε εγγραφή είναι ένας κανόνας.

Στη πλατφόρμα *J2ME*, όπου έγινε η υλοποίηση του ενδιάμεσου λογισμικού, δεν υπάρχει η έννοια του αρχείου όπως είναι γνωστή γενικότερα στα συστήματα λογισμικού για

σταθερούς υπολογιστές. Οι κινητές εφαρμογές χειρίζονται ένα τύπο αρχείων που έχουν τη μορφή συστάδων από εγγραφές. Περισσότερες λεπτομέρειες για θέματα υλοποίησης δίνονται στο Κεφάλαιο 8.

Στην πρώτη περίπτωση οι κανόνες καταχωρούνται από τον προγραμματιστή μέσα στον κώδικα της εφαρμογής στα τμήματα διασύνδεσης της υπηρεσίας με την εφαρμογή. Στη δεύτερη περίπτωση δίνεται ένα αρχείο στο οποίο έχουν εισαχθεί οι κανόνες. Ακόμα και αν οι κανόνες δοθούν προγραμματιστικά, ο *ContextManager* τους περνάει σε ένα αρχείο, ώστε να υπάρχουν αποθηκευμένοι για την επόμενη φορά που θα εκτελεστεί η εφαρμογή. Οπότε και στις δύο περιπτώσεις, ο *RulesParser* διαβάζει τους κανόνες από ένα αρχείο με καθορισμένο όνομα, στο μόνιμο αποθηκευτικό χώρο της συσκευής.

Το αρχείο μπορεί στη συνέχεια να σβηστεί είτε από το χρήστη, είτε από την υπηρεσία, για εξοικονόμηση χώρου. Στη συνέχεια θα δούμε ότι μπορεί να ξαναδημιουργηθεί κατά τη διάρκεια λειτουργίας του συστήματος και ακολούθως να μετατραπεί ή να σβηστεί. Αυτό μπορεί να γίνει είτε με επιλογές του χρήστη μέσω της διεπαφής που παρέχει ο *RulesRecordManager*, ή σαν ενέργεια κάποιου κανόνα (Εικόνα 7.1).

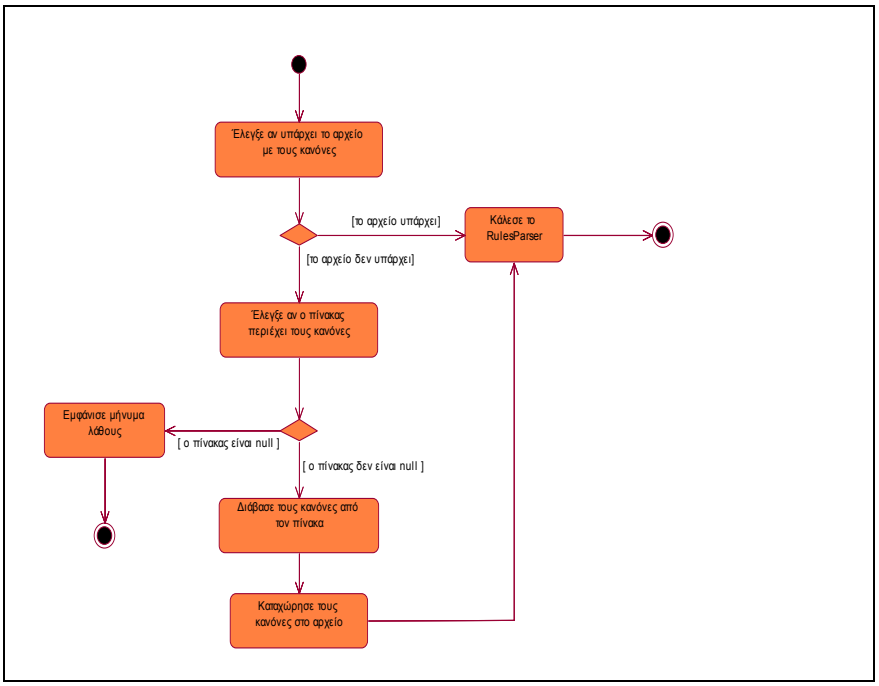
```
 (incomingMMS = true) && ([previous] StorSizeAvail < 10) =>
(deleteRulesRecordStore) && [next] (storeMMS)
```

Σημασιολογία:

“ Αν έρθει ένα MMS και την προηγούμενη χρονική στιγμή ο διαθέσιμος αποθηκευτικός χώρος ήταν μικρότερος από το 10% του συνολικού, τότε πρώτα σβήσε το αρχείο με τους κανόνες και μετά αποθήκευσε το MMS. ”

Εικόνα 7.1 Διαγραφή του Αρχείου των Κανόνων από Ενέργεια Κανόνα.

Κατά την έναρξη της εφαρμογής ο *ContextManager* ελέγχει αν υπάρχει αποθηκευμένο το αρχείο με τους κανόνες στο μόνιμο αποθηκευτικό χώρο της συσκευής. Αν δεν υπάρχει διαβάζει τους κανόνες από ένα πίνακα στον κώδικα της διεπαφής. Σε κάθε περίπτωση είτε το αρχείο ή ο πίνακας θα πρέπει να περιέχει τους κανόνες, αλλιώς η υπηρεσία δεν θα ξεκινήσει να λειτουργεί. Στο Σχήμα 7.2 φαίνεται διαγραμματικά η διαδικασία φόρτωσης των κανόνων από τον *ContextManager* στον *RulesParser*.



Σχήμα 7.2 Διάγραμμα Δραστηριότητας για τη Φόρτωση των Κανόνων.

7.2.3 Πέρασμα Παραμέτρων Ποιότητας

Ο *ContextManager* ελέγχει και τη διαδικασία φόρτωσης των παραμέτρων ποιότητας των προμηθευτών και τις περνάει στον *TrustCalculator*. Όπως ακριβώς και οι κανόνες, οι παράμετροι ποιότητας μπορούν να δοθούν είτε σε ένα καθορισμένο αρχείο ή μέσα από τη διεπαφή που υλοποιεί ο προγραμματιστής.

Σε αντίθεση με τους κανόνες, οι παράμετροι ποιότητας δεν είναι απαραίτητες για τη λειτουργία της υπηρεσίας. Ο υπολογισμός αξιοπιστίας των κανόνων μπορεί να ενεργοποιηθεί και να απενεργοποιηθεί με επιλογή του χρήστη. Αρχικά η επιλογή είναι απενεργοποιημένη. Συγκεκριμένα η διαδικασία φόρτωσης των παραμέτρων έχει ως εξής:

- Αν ο χρήστης επιλέξει να ενεργοποιήσει τον έλεγχο αξιοπιστίας, καλείται ο *TrustCalculator* για την εξαγωγή και τον υπολογισμό της αξιοπιστίας.
- Ενεργοποιείται η λειτουργία ελέγχου αξιοπιστίας των κανόνων.
- Η παράμετροι λαμβάνονται όπως στη περίπτωση των κανόνων, σε αρχείο ή μέσα από τον κώδικα της διεπαφής υπηρεσίας – εφαρμογής.

7.3 Συντακτική Ανάλυση Κανόνων

Ο *RulesParser* είναι το υποσύστημα που εκτελεί συντακτική ανάλυση στους κανόνες και εξάγει την πληροφορία που χρησιμοποιεί η υπηρεσία. Καλείται από τον *ContextManager* και διαβάζει τους κανόνες από ένα καθορισμένο αρχείο. Κατά τη διάρκεια της συντακτικής ανάλυσης αποσυνθέτει τους κανόνες στα δομικά τους συστατικά και δημιουργεί μία δομή στη μνήμη όπου αποθηκεύει το περιεχόμενο.

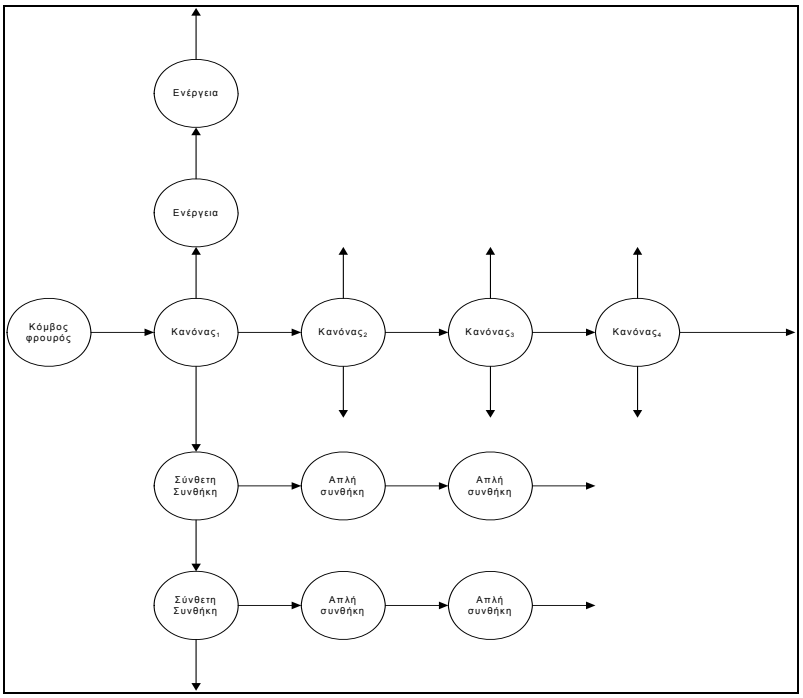
Το πέρασμα των κανόνων στη μνήμη επιτρέπει τη γρήγορη προσπέλαση των δεδομένων και κοστίζει λιγότερο υπολογιστικά από την αποθήκευση σε αρχείο. Η δομή είναι ένα δέντρο αυθαίρετης μορφής και η αποθήκευση γίνεται με κατάλληλο τρόπο ώστε να διευκολύνεται η προσπέλαση των δεδομένων κατά τη διάρκεια του ελέγχου των κανόνων. Η κατασκευή του δέντρου γίνεται ως εξής:

- Κάθε κανόνας αποθηκεύεται σε έναν κόμβο μιας απλά συνδεδεμένης λίστας.
- Στο κόμβο του κάθε κανόνα συνδέονται δύο ακόμα λίστες στις οποίες αποθηκεύεται το τμήμα συνθηκών και το τμήμα ενεργειών.
- Στη λίστα των συνθηκών, σε κάθε κόμβο αποθηκεύεται μία συνθήκη. Αν η συνθήκη είναι σύνθετη, δηλαδή αποτελείται από δύο ή παραπάνω απλές συνθήκες, συνδέεται μία ακόμα λίστα που κάθε κόμβος της περιέχει μία απλή συνθήκη.
- Στη λίστα των ενεργειών σε κάθε κόμβο αποθηκεύεται μία ενέργεια. Οι ενέργειες αποθηκεύονται ταξινομημένα στη λίστα με βάση τη σειρά εκτέλεσής τους όπως υποδεικνύεται από τους τελεστές *[next]*. Η ταξινόμηση της λίστας ενεργειών βοηθάει στη γρηγορότερη εκτέλεση των ενεργειών του κανόνα, αν αυτός ικανοποιηθεί.

Αφού τελειώσει η συντακτική ανάλυση το δέντρο γίνεται διαθέσιμο μέσω του *ContextManager* στα υπόλοιπα υποσυστήματα της υπηρεσίας. Συγκεκριμένα ο *RulesEvaluator* και ο *TrustCalculator* χρησιμοποιούν τα δεδομένα του δέντρου κατά την επεξεργασία των γεγονότων και τον υπολογισμό αξιοπιστίας των κανόνων αντίστοιχα.

Οι κόμβοι του δέντρου περιέχουν την απαραίτητη πληροφορία ώστε να γίνονται όσο το δυνατόν λιγότερες αναζητήσεις. Συγκεκριμένα κάθε κόμβος περιέχει δεδομένα για τις

λίστες που συνδέονται σε αυτόν, ώστε κατά την αναζήτηση να αποφεύγονται άσκοπες διασχίσεις μονοπατιών. Π.χ.: ο κόμβος ενός κανόνα περιέχει όλη τη συμβολοσειρά του κανόνα, ώστε να ανιχνεύονται όλες οι υποσυμβολοσειρές. Στο Σχήμα 7.3 φαίνεται η μορφή που έχει το δέντρο των κανόνων.



Σχήμα 7.3 Δέντρο Κανόνων.

7.4 Παρακολούθηση Προμηθευτών

Το υποσύστημα *SensorsMonitor* είναι υπεύθυνο για την παρακολούθηση των προμηθευτών και την ανίχνευση γεγονότων. Όπως αναφέρθηκε στο Κεφάλαιο 6, το *context* και οι προμηθευτές που το παράγουν ορίζονται από τον προγραμματιστή της εφαρμογής, ανάλογα με τις απαιτήσεις της εφαρμογής. Οι προμηθευτές υλοποιούνται ανεξάρτητα από την υπηρεσία ή συνδέονται στο ενδιάμεσο λογισμικό, αν πρόκειται για συσκευές, με χρήση των αντίστοιχων διεπαφών λογισμικού.

Στο υποσύστημα *SensorsMonitor* γίνεται η παρακολούθηση των προμηθευτών σε ξεχωριστό νήμα εκτέλεσης, ανεξάρτητα από τις υπόλοιπες διεργασίες της εφαρμογής και της υπηρεσίας. Ο προγραμματιστής της εφαρμογής συνδέει τους προμηθευτές με το υποσύστημα παρακολούθησης. Επιπλέον υλοποιεί φίλτρα γεγονότων για κάθε προμηθευτή. Τα φίλτρα

καθορίζουν τη σημαντικότητα μιας τιμής που παράγει ο προμηθευτής και αποφασίζουν αν αποτελεί γεγονός.

Το υποσύστημα παρακολούθησης καλεί συνεχώς τους προμηθευτές και με βάση τα φίλτρα ανιχνεύει γεγονότα. Τα γεγονότα αποστέλλονται στον *EventHandler*, ο οποίος τα προωθεί για επεξεργασία. Ο *EventHandler* παρέχει μία μέθοδο για την αποστολή των γεγονότων, η οποία καλείται από το νήμα παρακολούθησης.

Μπορούμε να διακρίνουμε δύο κατηγορίες παραμέτρων *context* με βάση το τρόπο με τον οποίο μεταβάλλονται οι τιμές των παραμέτρων:

1. *Context* εξωτερικό της εφαρμογής.
2. *Context* που προέρχεται από την εφαρμογή.

Στην πρώτη κατηγορία ανήκουν οι περισσότερες γενικές παράμετροι του *context*, όπως η μνήμη που χρησιμοποιεί η εφαρμογή, η μπαταρία της συσκευής, η χρήση της *CPU*, η τοποθεσία του χρήστη κ.α. Το χαρακτηριστικό γνώρισμα αυτών των παραμέτρων είναι ότι μεταβάλλονται απρόβλεπτα και απαιτούν συνεχή παρακολούθηση.

Στη δεύτερη κατηγορία ανήκουν *context* ιδιότητες που αποτελούν συγκεκριμένα χαρακτηριστικά της εφαρμογής. Για παράδειγμα μια επιλογή του χρήστη από κάποιο μενού της οθόνης, που επηρεάζει τον τρόπο παροχής κάποιων υπηρεσιών μπορεί να αποτελεί *context* πληροφορία και να περιλαμβάνεται στους κανόνες. Χαρακτηριστική περίπτωση για τις περισσότερες εφαρμογές είναι το μοτίβο λειτουργίας που χαρακτηρίζει την κατανάλωση μπαταρίας, ανάλογα με τη διαθέσιμη μπαταρία της συσκευής. Αυτές οι παράμετροι μεταβάλλονται προβλέψιμα γιατί, ο προγραμματιστής της εφαρμογής μπορεί να ξέρει εκ των προτέρων, σε ποιο σημείο στον κώδικα η εφαρμογή θα διαβάσει κάποια καινούργια τιμή.

Οι τιμές της πρώτης κατηγορίας παρακολουθούνται από το υποσύστημα *SensorsMonitor* το οποίο λειτουργεί ανεξάρτητα από την εφαρμογή. Οι τιμές της δεύτερης κατηγορίας αφορούν συγκεκριμένες παραμέτρους της εφαρμογής. Οι τιμές αυτές προέρχονται από επιλογές και προτιμήσεις του χρήστη ή τιμές που σχετίζονται με δεδομένα που παίρνει η εφαρμογή από άλλες πηγές (π.χ. βάσεις δεδομένων κ.α.). Το βασικό

χαρακτηριστικό αυτών των παραμέτρων είναι ότι μεταβάλλονται προβλέψιμα. Συνεπώς ο προγραμματιστής μπορεί να γνωρίζει τα σημεία στον κώδικα της εφαρμογής που δέχονται τιμές οι παράμετροι και εκεί να ενσωματώσει κλήσεις στον *EventHandler*, ώστε κάθε φορά να ειδοποιείται η υπηρεσία.

7.5 Χειρισμός Γεγονότων

Ο *EventHandler* είναι το υποσύστημα που δέχεται τα γεγονότα από τους προμηθευτές. Συντονίζει τη διαδικασία συλλογής των γεγονότων και ενημερώνει τη χρονική κατάσταση του συστήματος. Επικοινωνεί με το υποσύστημα *SensorsMonitor* και την εφαρμογή, από όπου συλλέγει τα γεγονότα. Τα γεγονότα στη συνέχεια προωθούνται στο *RulesEvaluator* για επεξεργασία και αποτίμηση.

Το *SensorsMonitor* περνάει στον *EventHandler* τις τιμές για τις παραμέτρους του *context*. Οι τιμές μπορεί να είναι αριθμητικές ακέραιες ή δεκαδικές, ή συμβολοσειρές. Ο *EventHandler* αποκρύπτει την ετερογένεια των τιμών, που παρέχουν οι προμηθευτές, κατασκευάζοντας γεγονότα στα οποία ενθυλακώνει την πληροφορία. Οι τιμές μετατρέπονται, σύμφωνα με το μοντέλο αναπαράστασης, σε συμβολοσειρές της μορφής <*context_property*> = <*property_value*>. Αυτήν είναι η μορφή που μπορεί να επεξεργαστεί ο *RulesEvaluator*.

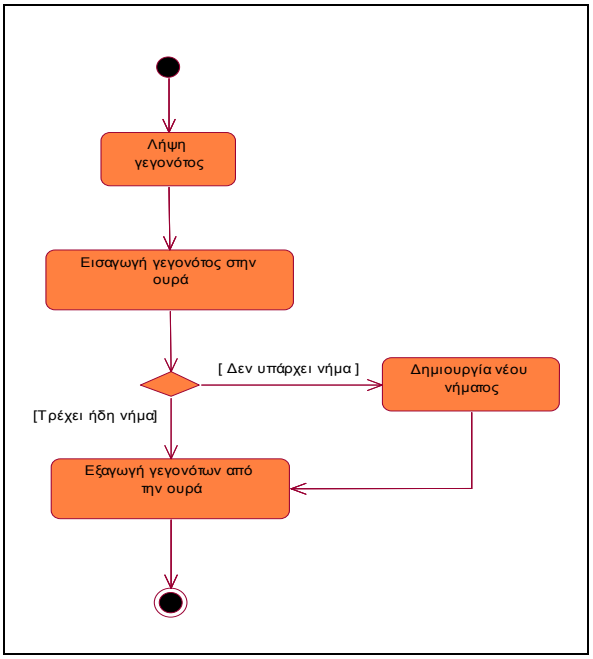
Επειδή τα γεγονότα φτάνουν σε αυθαίρετες χρονικές στιγμές στο σύστημα, είναι πολύ πιθανό κάποιο γεγονός να χρειαστεί να περιμένει μέχρι να τελειώσει η επεξεργασία του προηγούμενου. Για αυτό το λόγο, ο *EventHandler* διατηρεί μια ουρά (*FIFO*) γεγονότων. Τα γεγονότα αποθηκεύονται στην ουρά με τη σειρά που φτάνουν και πρέπει να περιμένουν να τελειώσει η επεξεργασία των προηγούμενων γεγονότων για να εξυπηρετηθούν.

Ο *EventHandler* παρέχει τη μέθοδο *EventListener* για τη συλλογή των γεγονότων. Κάθε φορά που ο *EventListener* καλείται να συλλέξει μία τιμή εκτελούνται διαδοχικά τις παρακάτω ενέργειες:

- Η τιμή ενσωματώνεται σε ένα γεγονός και μπαίνει στην ουρά.
- Δημιουργείται ένα νέο νήμα εκτέλεσης για το χειρισμό της ουράς.

- Το νήμα αυτό εκτελεί ένα βρόχο βγάζοντας γεγονότα από την ουρά, μέχρι αυτή να αδειάσει. Τότε το νήμα τερματίζει τη λειτουργία του.
- Κάθε γεγονός που εξάγεται από την ουρά περνάει στον *RulesEvaluator* για να ελεγχθεί η ικανοποιησιμότητα των κανόνων.

Κάθε φορά που ο *EventListener* λαμβάνει ένα γεγονός το βάζει στην ουρά και ελέγχει αν εκτελείται το νήμα που βγάζει τα γεγονότα από την ουρά. Αν δεν εκτελείται ξεκινάει ένα καινούργιο νήμα ώστε να επεξεργαστεί τουλάχιστον το γεγονός που μόλις έφτασε και ενδεχομένως και όποια άλλα φτάσουν κατά τη διάρκεια της επεξεργασίας του. Αν εκτελείται ήδη το νήμα εξαγωγής γεγονότων, δεν ξεκινάει κάποιο νέο νήμα, αφού τα γεγονότα πρέπει να προωθηθούν στον *RulesEvaluator* με τη σειρά που φτάνουν. Στο Σχήμα 7.4 φαίνεται διαγραμματικά η λειτουργία του *EventHandler*.



Σχήμα 7.4 Χειρισμός Γεγονότος.

Κάθε φορά που βγαίνει ένα γεγονός από την ουρά το νήμα του *EventHandler* αυξάνει κατά ένα το χρόνο του συστήματος. Κάθε γεγονός υποδεικνύει ότι ο χρόνος του συστήματος αυξήθηκε κατά ένα, δηλαδή το σύστημα πέρασε σε μια νέα κατάσταση. Ο νέος χρόνος μεταφέρεται στον *RulesEvaluator*, όπου θα χρησιμοποιηθεί για τους χρονικούς ελέγχους των κανόνων.

Συνεπώς στον *EventHandler*, κάθε χρονική στιγμή, υπάρχει μόνο ένα νήμα που εξάγει γεγονότα από την ουρά, ενώ ο *EventListener* εκτελείται ανεξάρτητα σε ένα άλλο νήμα, εισάγοντας γεγονότα στην ουρά. Με αυτόν τον τρόπο επιτυγχάνεται η χρονική συνέπεια στον έλεγχο των κανόνων, ενώ ελαχιστοποιείται η απόκριση της υπηρεσίας, καθώς ακόμα και όταν έχουμε καταιγισμό γεγονότων υπάρχει πάντα ένα νήμα να τα εξυπηρετεί. Στη συνέχεια αναλύουμε την έννοια της χρονικής συνέπειας στο χειρισμό των γεγονότων και τους λόγους που επιλέξαμε αυτήν τη μέθοδο χειρισμού γεγονότων, συγκρίνοντας εναλλακτικές προσεγγίσεις.

7.5.1 Διατήρηση Χρονικής Συνέπειας

Ο χειρισμός των χρονικών εξαρτήσεων του *context* απαιτεί η υπηρεσία να γνωρίζει τη χρονική πληροφορία, που χαρακτηρίζει τα γεγονότα και να τη χρησιμοποιεί κατάλληλα στην αποτίμηση των κανόνων. Ο *EventHandler* είναι υπεύθυνος για τη συλλογή και προώθηση των γεγονότων με κατάλληλο τρόπο, ώστε να διατηρείται η χρονική συνέπεια στο σύστημα. Λέγοντας χρονική συνέπεια εννοούμε ότι:

«Προτού ένα γεγονός προωθηθεί για επεξεργασία, η υπηρεσία θα πρέπει να γνωρίζει την πραγματοποίηση, όλων των γεγονότων που συνέβησαν πριν από αυτό το γεγονός»

Ο παραπάνω ορισμός ουσιαστικά επιβάλλει ότι, τα γεγονότα θα πρέπει να προωθούνται για επεξεργασία με τη σειρά με την οποία πραγματοποιήθηκαν και επιπλέον, δεν μπορεί να γίνει παράλληλα η επεξεργασία δύο ή περισσότερων γεγονότων. Ένα γεγονός, ενδεχομένως, μεταφέρει στην υπηρεσία κάποια πληροφορία. Η πληροφορία αυτή αποθηκεύεται στη γνώση του συστήματος (στο δέντρο) και χρησιμοποιείται για τον έλεγχο των κανόνων. Αν η υπηρεσία επιχειρήσει να ελέγξει τους κανόνες, χωρίς να έχει αποθηκευτεί η πληροφορία κάποιου προηγούμενου γεγονότος, ενδεχομένως, τα συμπεράσματα που θα προκύψουν θα είναι λανθασμένα, αφού η γνώση του συστήματος θα είναι ασυνεπής.

Η πληροφορία που μεταφέρει ένα γεγονός, καθορίζει την ικανοποίηση των συνθηκών των κανόνων. Η πληροφορία αυτή μπορεί να περιλαμβάνει τα εξής δεδομένα:

1. Ο χρόνος του συστήματος αυξήθηκε κατά ένα. Η πληροφορία αυτή χρησιμοποιείται στην αποτίμηση όλων των συνθηκών.
2. Μία συνθήκη έγινε *false/true*. Η πληροφορία αυτή χρησιμοποιείται στην αποτίμηση όλων των συνθηκών.
3. Μία συνθήκη ικανοποιήθηκε για πρώτη φορά. Η πληροφορία αυτή χρησιμοποιείται στην αποτίμηση συνθηκών που συνοδεύονται από τελεστή *[once]* ή *[always_been]*.
4. Μία συνθήκη έγινε *false*, ενώ την προηγούμενη χρονική στιγμή ήταν *true*. Η πληροφορία αυτή χρησιμοποιείται στην αποτίμηση συνθηκών που συνοδεύονται από τελεστή *[previous]* ή *[always_been]*.

Είναι προφανές ότι, αν γίνει επεξεργασία ενός γεγονότος, χωρίς η υπηρεσία να έχει επεξεργαστεί ένα γεγονός που συνέβη νωρίτερα, είναι πιθανό κάποια από τις παραπάνω πληροφορίες να μην έχει αποθηκευτεί στη γνώση του συστήματος. Συνεπώς τα συμπεράσματα που θα εξαχθούν με βάση το νέο γεγονός, θα είναι λανθασμένα.

Η μέθοδος που χρησιμοποιεί ο *EventHandler* για το χειρισμό των γεγονότων εξασφαλίζει ότι, δεν πρόκειται η επεξεργασία ενός γεγονότος να προηγηθεί της επεξεργασίας ενός άλλου. Τα γεγονότα εξάγονται από μία ουρά και μόνο ένα νήμα τα προωθεί για επεξεργασία. Η επεξεργασία γίνεται σειριακά, με τη σειρά που πραγματοποιήθηκαν τα γεγονότα.

7.5.2 Προσεγγίσεις Χειρισμού Γεγονότων

Μία εναλλακτική προσέγγιση θα ήταν να υπήρχαν περισσότερα από ένα νήματα για να βγάζουν γεγονότα, από μία ή περισσότερες ουρές. Τα νήματα αυτά θα έπρεπε να καλούν ταυτόχρονα τον *RulesEvaluator* για την αποτίμηση των γεγονότων. Σε αυτήν την περίπτωση, θα είχαμε ταυτόχρονη προσπέλαση και επεξεργασία κοινής μνήμης, αφού κατά τον έλεγχο των κανόνων γίνονται αρκετές ενημερώσεις και αλλαγές στα πεδία των κόμβων του δέντρου. Η προσέγγιση αυτή περιέχει τον κίνδυνο να δημιουργηθούν πολλές ασυνέπειες και σφάλματα στον έλεγχο των κανόνων.

Ακόμα και αν συγχρονίζαμε τα νήματα με χρήση μεθόδων διαχείρισης κοινής μνήμης (π.χ. η επιλογή *synchronized* της *Java*, σημαφόροι κ.α.) θα ήταν πολύ πιθανό να

δημιουργηθούν χρονικές ασυνέπειες στην αποτίμηση των κανόνων. Θα ήταν απίθανο να αποκλειστεί το ενδεχόμενο, κάποιο από τα νήματα που βγάζουν γεγονότα από την ουρά και καλούν τον *RulesEvaluator*, να προωθήσει ένα γεγονός για επεξεργασία, πριν ακόμα ένα άλλο νήμα τελειώσει την ενημέρωση των πεδίων του δέντρου για τη νέα γνώση, που μετέφερε, ένα γεγονός που είχε προηγηθεί χρονικά. Για παράδειγμα, έστω ότι για μία εφαρμογή είχαμε τον κανόνα:

```
([previous] BatteryAvailable < 20) && (incomingCall = true) =>
(BatteryMode=Silent) && [next] (incCallHandler)
```

Ο κανόνας λέει ότι αν έρθει μια εισερχόμενη κλήση και τη προηγούμενη χρονική στιγμή η διαθέσιμη μπαταρία της συσκευής είχε πέσει κάτω από το *20%*, πρώτα θέσε το μοτίβο στο αθόρυβο και στη συνέχεια κάλεσε τη μέθοδο της εφαρμογής που χειρίζεται τις εισερχόμενες κλήσεις, έτσι ώστε να μην τελειώσει η μπαταρία του κινητού. Ένα πολύ πιθανό σενάριο είναι το εξής:

- Ο *EventListener* δέχεται το γεγονός Α: `BatteryAvailable = 18`, τη χρονική στιγμή *t* και το γεγονός Β: `incomingcall = true` τη χρονική στιγμή *t+1*.
- Τα δύο γεγονότα έρχονται σχεδόν ταυτόχρονα και δύο διαφορετικά νήματα αναλαμβάνουν την επεξεργασία τους.
- Τότε είναι πολύ πιθανό λόγω καθυστερήσεων και διακοπών μεταξύ των νημάτων, το νήμα του *B* να τελειώσει ή να εκτελέσει κάποιο κρίσιμο (για τον χρονικό έλεγχο) κομμάτι κώδικα πριν το *A*.
- Αυτό θα έχει σαν συνέπεια η συνθήκη `incomingCall = true` να χαρακτηριστεί αληθής τη χρονική στιγμή *t+1*, αλλά η συνθήκη `[previous] batteryAvailable < 20`, να χαρακτηριστεί ψευδής, αφού δεν θα έχει γίνει ακόμα η χρονική ενημέρωση του *RulesEvaluator*, ότι το γεγονός *A* προηγήθηκε του *B*. Οπότε ο κανόνας, εσφαλμένα, δεν θα ικανοποιηθεί.

Επιπλέον στη συνεργασία μεταξύ νημάτων, που διαχειρίζονται συνεχώς κοινή μνήμη, είναι πολύ πιθανό να δημιουργηθούν αδιέξοδα ή φαινόμενα ασιτίας (*starvation*). Κάθε φορά που ένα νήμα εκτελεί το κρίσιμο κομμάτι κώδικα τα υπόλοιπα μπλοκάρουν και περιμένουν να τελειώσει. Σε μια χρονικά κρίσιμη υπηρεσία όπως ο *ContextManager*, οι αλλεπάλληλες

διακοπές στην εκτέλεση λόγο συγχρονισμού θα μπορούσαν να προκαλέσουν μεγάλες καθυστερήσεις στην επεξεργασία των γεγονότων, παραπλανώντας το *RulesEvaluator* σε λάθος αποτιμήσεις κανόνων.

Τέλος η δημιουργία και η διαχείριση νημάτων είναι υπολογιστικά ευαίσθητες διαδικασίες, προσθέτοντας επιπλέον φόρτο στη *CPU*, αλλά και στη μνήμη, για κάθε επιπλέον νήμα που δημιουργείται. Βασικό μέλημα στην ανάπτυξη της υπηρεσίας είναι η οικονομική διαχείριση των περιορισμένων πόρων της συσκευής. Συνεπώς πρέπει να απορριφθεί μια προσέγγιση που απαιτεί την κατασκευή και διαχείριση αντικειμένων (νημάτων) που καταναλώνουν πόρους, χωρίς να αυξάνουν τη χρονική απόκριση του συστήματος. Συνολικά τα νήματα που μπορεί να τρέχουν στο σύστημα είναι τα εξής:

1. Το νήμα που τρέχει η εφαρμογή (*MIDlet*).
2. Το νήμα του *SensorsMonitor*.
3. Το νήμα που παράγει ο *EventListener*.
4. Άλλα νήματα της εφαρμογής, όπως το νήμα που «ακούει» εισερχόμενα *SMSs*.

Από τα παραπάνω νήματα, συνεχώς τρέχουν μόνο τα δύο πρώτα, ενώ το $3^{ο}$ μόνο όποτε υπάρχουν γεγονότα στην ουρά.

7.6 Αποτίμηση Κανόνων

Ο *RulesEvaluator* καλείται από τον *EventHandler* και ελέγχει την ικανοποιησιμότητα όλων των κανόνων σε κάθε χρονική κατάσταση. Σε αυτό το υποσύστημα ενσωματώνεται ουσιαστικά η ευφυία της υπηρεσίας, που δίνει ικανότητες συλλογισμού στο σύστημα. Ο *RulesEvaluator* λαμβάνει τα γεγονότα από την ουρά του *EventHandler* και ελέγχοντας τους κανόνες αποφασίζει αν πρέπει να εκτελεστεί το τμήμα ενεργειών κάποιου κανόνα. Επιπλέον είναι υπεύθυνος να διατηρήσει τη συνέπεια της πληροφορίας στο δέντρο των κανόνων κάνοντας τις απαραίτητες ανανεώσεις.

7.6.1 Διαχείριση Γνώσης Συστήματος

Ο *RulesEvaluator* αποτελεί ουσιαστικά ένα έμπειρο σύστημα, που λαμβάνει αποφάσεις με βάση τους κανόνες και τη γνώση που παρέχουν τα γεγονότα που συλλέγει η υπηρεσία. *Για την αποθήκευση της γνώσης δεν χρησιμοποιείται ιστορικό για το context*. Οι τιμές των γεγονότων δεν αποθηκεύονται σε κάποιο αρχείο ώστε να χρησιμοποιηθούν για την αποτίμηση των συνθηκών. Η γνώση της υπηρεσίας αποθηκεύεται σε κάποιες λογικές μεταβλητές, που χρησιμοποιούνται στον έλεγχο ικανοποιησιμότητας των κανόνων. Συγκεκριμένα στους κόμβους των συνθηκών κρατούνται, μεταξύ των άλλων, τα εξής πεδία:

- *boolean sat*. Δηλώνει την ικανοποιησιμότητα την τρέχουσα χρονική στιγμή, που γίνεται ο έλεγχος.
- *boolean prev_sat*. Δηλώνει την ικανοποιησιμότητα την προηγούμενη χρονική στιγμή.
- *boolean once_sat*. Δηλώνει την ικανοποιησιμότητα τουλάχιστον μία φορά στο παρελθόν.
- *boolean alwb_sat*. Δηλώνει την ικανοποιησιμότητα συνεχώς στο παρελθόν.

Τα πεδία αυτά ενημερώνονται κάθε φορά που αλλάζει η ικανοποιησιμότητα μίας συνθήκης, ως αποτέλεσμα της πραγματοποίησης ενός γεγονότος, αν είναι απαραίτητο. Η ενημέρωση των πεδίων γίνεται μόνο στους κόμβους που έχουν τον αντίστοιχο χρονικό τελεστή. Επιπλέον κρατείται στους κόμβους ο χρόνος ικανοποίησης των συνθηκών και άλλες τιμές που χρησιμοποιεί ο *RulesEvaluator* για την αποτίμηση των κανόνων. Με βάση αυτά τα πεδία γίνεται τελικά η λογική αποτίμηση του κανόνα.

7.6.2 Ενημέρωση Πεδίων στους Κόμβους του Δέντρου

Για κάθε γεγονός που εξάγεται από την ουρά ο *RulesEvaluator* εκτελεί ένα σύνολο ενεργειών για την ενημέρωση της γνώσης του συστήματος πριν πραγματοποιηθεί ο έλεγχος ικανοποιησιμότητας των κανόνων. Το δέντρο με τους κανόνες διασχίζεται και ενημερώνονται τα πεδία ικανοποίησης στις λίστες των συνθηκών. Η διάσχιση γίνεται με βάση το νέο γεγονός ως εξής:

- Διασχίζεται η λίστα των κανόνων και ελέγχεται αν η παράμετρος του *context* στο γεγονός συμμετέχει σε κάποιο κανόνα.
- Αν συμμετέχει, διασχίζεται η λίστα των συνθηκών του κανόνα και ελέγχεται αν συμμετέχει σε κάποια σύνθετη ή απλή συνθήκη.
- Αν συμμετέχει σε μια απλή συνθήκη το πεδίο ικανοποίησης της συνθήκης αλλάζει ή παραμένει το ίδιο, ανάλογα με το αν ικανοποιείται η συνθήκη από τη νέα τιμή της παραμέτρου, τη συγκεκριμένη χρονική στιγμή.
- Αν συμμετέχει σε μία σύνθετη συνθήκη, διασχίζεται η λίστα των απλών υποσυνθηκών και ενημερώνεται το πεδίο ικανοποίησης σε κάθε κόμβο απλής συνθήκης. Στη συνέχεια ενημερώνεται και το πεδίο ικανοποίησης της σύνθετης συνθήκης, αν αυτό έχει αλλάξει.

Η παραπάνω μέθοδος μας απαλλάσσει από την ανάγκη να κρατήσουμε ιστορικό τιμών για τα γεγονότα. Ένα αρχείο ή μία δομή ιστορικού θα αύξανε σημαντικά τις απαιτήσεις της υπηρεσίας για αποθηκευτικό χώρο. Επιπλέον θα πρόσθετε σημαντική χρονική επιβάρυνση, καθώς θα έπρεπε να αναζητούνται οι τιμές μιας παραμέτρου στο ιστορικό και να γίνεται σύγκριση παλαιών και νέων τιμών. Η έλλειψη βάσης ιστορικού, μας επιτρέπει να πραγματοποιούμε εξαγωγή συμπερασμάτων πάνω στην κινητή συσκευή και όχι σε κάποιο κεντρικό κόμβο με περίσσεια αποθηκευτικού χώρου.

7.6.3 Ικανοποίηση Κανόνων και Εκτέλεση Ενεργειών

Με βάση τις καινούργιες τιμές στα πεδία των συνθηκών ελέγχεται η ικανοποιησιμότητα των κανόνων. Ο *RulesEvaluator* ελέγχει, αν ένας κανόνας ικανοποιείται λογικά με βάση τις συζεύξεις και τις διαζεύξεις μεταξύ των συνθηκών. Σε περίπτωση που κάποιος κανόνας ικανοποιείται καλείται ο *TrustCalculator* (περιγράφεται στη συνέχεια) για να αποφασίσει αν ο κανόνας είναι αξιόπιστος.

Η ικανοποίηση των κανόνων ελέγχεται σε κάθε χρονική στιγμή, ανεξάρτητα από το αν, το γεγονός που ήρθε επηρεάζει κάποιο κανόνα. Ο λόγος είναι ότι ένας κανόνας μπορεί να ικανοποιηθεί τη χρονική στιγμή *t* και να παραμένει ικανοποιημένος στις υπόλοιπες καταστάσεις του συστήματος. Επιπλέον, αν μία συνθήκη που συνοδεύεται από *[previous]* ικανοποιηθεί τη χρονική στιγμή *t*, η συνθήκη θα είναι αληθής τη χρονική στιγμή $t + 1$, ακόμα

και αν το γεγονός που ήρθε τη χρονική $t + 1$, δεν επηρέασε την ικανοποιησιμότητα κάποιας συνθήκης του κανόνα. Συνεπώς όλοι οι κανόνες θα πρέπει να ελέγχονται κάθε φορά που λαμβάνεται ένα γεγονός.

Αν ένας κανόνας ικανοποιηθεί και κριθεί αξιόπιστος καλείται το τμήμα ενεργοποίησης του κανόνα. Η λίστα των ενεργειών διασχίζεται και οι ενέργειες εκτελούνται ακολουθώντας τη σειρά διάσχισης των κόμβων. Θυμίζουμε ότι στη φάση της συντακτικής ανάλυσης οι λίστες των ενεργειών είχαν ταξινομηθεί με βάση τη σειρά εκτέλεσής τους για να επιταχυνθεί η εκτέλεση των ενεργειών. Συνεπώς, οι ενέργειες εκτελούνται απ' ευθείας με τη χρονική σειρά που υποδεικνύουν οι τελεστές *[next]*, αν υπάρχουν.

Για κάθε ενέργεια εκτελείται η αντίστοιχη μέθοδος της εφαρμογής που υλοποιεί την ενέργεια. Η αντιστοίχιση ενεργειών και μεθόδων υλοποιείται από τον προγραμματιστή, σε μία μέθοδο που ορίζεται από την προγραμματιστική διεπαφή διασύνδεσης της εφαρμογής με την υπηρεσία. Η μέθοδος αυτή καλείται από τον *RulesEvaluator* για κάθε ενέργεια ενός κανόνα, με όρισμα τη συμβολοσειρά που συμβολίζει την ενέργεια. Η μέθοδος αντιστοιχίζει τη συμβολοσειρά με την κλήση μίας μεθόδου την εφαρμογής που πραγματοποιεί αυτήν την ενέργεια. Η μέθοδος που καλεί ο *RulesEvaluator* ονομάζεται *ActionResolve* και αποτελεί ένα *callback* που χρησιμοποιεί η υπηρεσία για να ειδοποιήσει την εφαρμογή για τις ενέργειες προσαρμογής. Η μέθοδος περιγράφεται αναλυτικότερα στο Κεφάλαιο 8.

Αν μια ενέργεια αποτελείται από την ανάθεση τιμής σε μία παράμετρο του *context*, η ενέργεια αποτελεί εσωτερικό γεγονός, το οποίο πρέπει να ελεγχθεί από τον *RulesEvaluator*. Παρόλα αυτά δεν ελέγχεται αμέσως, αλλά αφού εκτελεστεί, τοποθετείται στην ουρά του *EventHandler* ώστε να εξεταστεί με τη σωστή χρονική σειρά.

7.6.4 Διατήρηση Χρονικής Συνέπειας

Ο *RulesEvaluator* δίνει νόημα στους χρονικούς τελεστές που συνοδεύουν τις ενέργειες ενός κανόνα. Ο περιορισμός που επιβάλει ο τελεστής *[next]* ικανοποιείται με την διατεταγμένη εκτέλεση των ενεργειών. Οι τελεστές *[henceforth]* και *[until]* επιβάλουν την εκτέλεση κάποιων επιπλέον ενεργειών που διατηρούν τη χρονική συνέπεια στο σύστημα. Οι

ενέργειες αυτές αφορούν ανανεώσεις στο δέντρο των κανόνων ώστε να ικανοποιηθούν οι περιορισμοί που επιβάλλονται από τους δύο χρονικούς τελεστές.

Εκτέλεση Ενέργειας με τον Τελεστή [until]

Ο τελεστής *[until]* επιβάλει σε μια παράμετρο του *context* μια τιμή, η οποία δεν πρέπει να αλλάξει μέχρι να ικανοποιηθεί μια συγκεκριμένη ενέργεια. Αν μια ενέργεια ικανοποιηθεί και είναι της μορφής *A [until] B*, όπου *A* είναι ενέργεια και *B* είναι συνθήκη τότε ο *RulesEvaluator* κάνει διαδοχικά τα εξής:

- Εκτελεί την ενέργεια *A*.
- Προσθέτει στη λίστα των κανόνων ένα ακόμα κόμβο με τη συνθήκη *B*, χωρίς τμήμα ενεργειών.
- Διατρέχει τις λίστες ενεργειών όλων των κανόνων και «κλειδώνει» τις ενέργειες που περιέχουν την παράμετρο της συνθήκης *A*. Το κλείδωμα γίνεται με την αλλαγή ενός πεδίου *lock* στους κόμβους των ενεργειών. Αν ικανοποιηθεί κάποιος κανόνας, οι ενέργειες του πραγματοποιούνται εφόσον *δεν* είναι κλειδωμένες από κάποια συνθήκη *[until]*.

Η συνθήκη που προστέθηκε ελέγχεται μαζί με τους υπόλοιπους κανόνες, σε κάθε γεγονός που λαμβάνει η υπηρεσία. Όταν ικανοποιηθεί η συνθήκη, η λίστα των κανόνων διασχίζεται πάλι και οι ενέργειες που είχαν κλειδωθεί από τη συγκεκριμένη συνθήκη ξεκλειδώνονται. Έπειτα η συνθήκη αφαιρείται από το δέντρο.

Έτσι εξασφαλίζεται ότι μια παράμετρος του *context*, δεν θα αλλάξει τιμή από την υπηρεσία μέχρι να ικανοποιηθεί μια συνθήκη. Παρόλα αυτά μια κλειδωμένη παράμετρος μπορεί να αλλάξει τιμή, αν αυτό προέλθει από το χρήστη. Ο χρήστης μπορεί να επιλέξει να εισάγει μια διαφορετική τιμή από αυτή που έχει η κλειδωμένη παράμετρος, μέσω της εφαρμογής. Για παράδειγμα έστω ο κανόνας:

```
([once]BatteryAvail<30) =>
(BatteryMode=Economy [until] BatteryAvail>20)
```

Αν ικανοποιηθεί η συνθήκη του κανόνα και κλειδωθεί η ενέργεια *BatteryMode = Economy* σε όλους του κανόνες, η τιμή του *BatteryMode*, δεν μπορεί να αλλάξει από την ενέργεια κάποιου άλλου κανόνα, μέχρι να ικανοποιηθεί η συνθήκη *(BatteryAvail > 20)*. Ο χρήστης όμως μπορεί από την οθόνη της συσκευής του να αλλάξει τη ρύθμιση του *BatteryMode* αν το επιθυμεί, π.χ.: *BatteryMode = Normal*. Σε αυτήν την περίπτωση το νέο γεγονός, *BatteryMode = Normal,* περνάει κανονικά στον *EventHandler*, αλλά δεν ξεκλειδώνονται οι κλειδωμένες ενέργειες των κανόνων, αφού η συνθήκη *BatteryAvail > 20* εξακολουθεί να μην ισχύει.

Εκτέλεση Ενέργειας με τον Τελεστή [henceforth]

Ο τελεστής *[henceforth]* επιβάλει σε μία παράμετρο του *context* μία τιμή, η οποία δεν μπορεί να αλλάξει ξανά κατά τη διάρκεια εκτέλεσης της εφαρμογής. Αν κάποια ενέργεια είναι της μορφής *[henceforth] A* εκτελούνται διαδοχικά τα εξής:

- Εκτελείται η ενέργεια *A*.
- Διασχίζονται οι λίστες των ενεργειών και αφαιρούνται οι ενέργειες που αναθέτουν κάποια τιμή στην παράμετρο της ενέργειας *A*. Οι κόμβοι που περιέχουν αυτές τις ενέργειες αφαιρούνται από το δέντρο και έτσι δεν ενεργοποιούνται πότε ξανά μέχρι την επανέναρξη της εφαρμογής και το νέο φόρτωμα των κανόνων.
- Αν αφαιρεθούν όλες οι ενέργειες από κάποιο κανόνα, αυτός δεν έχει πλέον νόημα ύπαρξης, αφού η ικανοποίηση του δεν συνεπάγεται πλέον καμία ενέργεια. Έτσι διαγράφεται από τη λίστα των κανόνων και ταυτόχρονα μειώνεται το μέγεθος της λίστας και οι άσκοπες αναζητήσεις.
- Διασχίζεται η λίστα των κανόνων και από τους κανόνες που έχουν ενέργειες με την παράμετρο *A,* αφαιρείται από τις συμβολοσειρές που συμβολίζουν το τμήμα ενεργειών η υποσυμβολοσειρά με την παράμετρο του *A*. Αυτό γίνεται για βελτίωση της απόδοσης αφού, σε μελλοντική αναζήτηση ενεργειών για κλείδωμα, από κάποια συνθήκη *[until]*, η ενέργεια δε θα βρεθεί στη συμβολοσειρά του κανόνα και δεν θα γίνει η άσκοπη αναζήτηση της ενέργειας στη λίστα ενεργειών του κανόνα. Ακολουθεί ένα παράδειγμα αυτής της λειτουργίας.

Έστω οι ενέργειες, οι οποίες βρίσκονται σε διαφορετικούς κανόνες:

E_1: [henceforth] (BatteryMode = Silent)
E_2: (BatteryMode = Normal [until] Location = INSIDE)

Όταν εκτελεστεί η E_1, θα αφαιρεθούν από όλες τις λίστες ενεργειών, οι ενέργειες που περιέχουν την παράμετρο *BatteryMode*. Οι συμβολοσειρές των κανόνων όμως θα περιέχουν ακόμα την υποσυμβολοσειρά *BatteryMode* στο τμήμα ενεργειών. Αυτό μπορεί να οδηγήσει σε άσκοπες αναζητήσεις στις λίστες των ενεργειών. Οι αναζητήσεις μπορεί να αφορούν την εύρεση ενεργειών για κλείδωμα, μετά την εκτέλεση της E_2. Συνεπώς ενημερώνεται η συμβολοσειρά αναπαράστασης του κανόνα, αφαιρώντας τη συμβολοσειρά *BatteryMode*.

Έτσι οι ενέργειες που εκτελούνται και προσδιορίζονται από τον τελεστή *[henceforth]* αναθέτουν μία τιμή σε μία παράμετρο, η οποία δεν αλλάζει ποτέ ξανά από μία ενέργεια. Μπορεί όμως πάλι να αλλάξει απ' ευθείας από το χρήστη. Σε αυτήν την περίπτωση το γεγονός που δημιουργείται είναι έγκυρο, περνάει από τους κανόνες και χρησιμοποιείται για τον έλεγχο ικανοποίησης τους. Δεν υπάρχει όμως πλέον ενέργεια, σε κανέναν κανόνα, που να αλλάζει την τιμή αυτής της ιδιότητας.

Η υπηρεσία μέσω των ενεργειών διαμόρφωσης του δέντρου, ουσιαστικά *αυτοδιαμορφώνει* τη λειτουργία της. Οι περιττοί κόμβοι αφαιρούνται και οι υπόλοιποι ενημερώνονται με βάση τις αλλαγές του *context*. Οι ενέργειες αυτές διαμορφώνουν τη συμπεριφορά της υπηρεσίας, αυξάνουν τη διαθέσιμη μνήμη και βελτιώνουν την χρονική της απόκριση.

7.7 Υπολογισμός Αξιοπιστίας των Κανόνων

Ο *TrustCalculator* είναι το υποσύστημα της υπηρεσίας που ενσωματώνει την ποιότητα του *context* στη διαδικασία συλλογισμού που εκτελεί ο *RulesEvaluator*. Η ποιότητα του *context*, που παράγει ένας προμηθευτής, περιγράφεται από τις τρεις πιθανότητες που αναφέρθηκαν στο Κεφάλαιο 6: *αναξιοπιστία*, *ανασφάλεια* και *ανακρίβεια*. Με βάση αυτές τις πιθανότητες ο *TrustCalculator* υπολογίζει τις τρεις συνολικές πιθανότητες για κάθε κανόνα, με μία απλή πιθανοτική μέθοδο.

Συγκεκριμένα, μια τιμή που περιλαμβάνεται σε ένα γεγονός, σχετίζεται με την ανασφάλεια, την αναξιοπιστία και την ανακρίβεια, που χαρακτηρίζουν τον προμηθευτή που την παρήγαγε. Ο *TrustCalculator* βασιζόμενος στις πιθανότητες των τιμών των παραμέτρων που συμμετέχουν σε ένα κανόνα, υπολογίζει τις συνολικές πιθανότητες του κανόνα. Ο υπολογισμός των παραμέτρων ποιότητας για έναν κανόνα γίνεται με την εξής μέθοδο:

- Για μια απλή συνθήκη, η μία πιθανότητα που τη χαρακτηρίζει είναι ίση με την αντίστοιχη πιθανότητα του προμηθευτή που σχετίζεται με την παράμετρο της συνθήκης.
- Για μία σύνθετη συνθήκη, αν οι συνθετικές της συνθήκες είναι διαζευγμένες, η συνολική πιθανότητα υπολογίζεται, σαν τον άθροισμα δύο διαζευγμένων τιμών, μείον το γινόμενό τους, π.χ.: για τη συνθήκη $A \,||\, B$, η συνολική πιθανότητα είναι $P_A + P_B - P_A * P_B$.
- Για μία σύνθετη συνθήκη, αν οι συνθετικές της συνθήκες είναι συζευγμένες, η συνολική πιθανότητα υπολογίζεται, σαν το γινόμενο δύο συζευγμένων τιμών, π.χ.: για τη συνθήκη $A \,\&\&\, B$, η συνολική πιθανότητα είναι $P_A * P_B$.

Με αυτήν τη μέθοδο υπολογίζονται και οι τρεις πιθανότητες για τον κανόνα. Οι πιθανότητες συγκρίνονται με τα αντίστοιχα προκαθορισμένα κατώφλια αξιοπιστίας. Αν καμία πιθανότητα δεν υπερβαίνει το αντίστοιχο κατώφλι, οι ενέργειες του κανόνα εκτελούνται. Σε αντίθετη περίπτωση ο κανόνας κρίνεται αναξιόπιστος. Στην Εικόνα 7.2 φαίνεται ένα παράδειγμα υπολογισμού μιας πιθανότητας ενός κανόνα.

$$\text{R1} = ((\text{node.CPULoad} \geq 2) \,||\, (\text{node.MemAvail} \leq 512)) \,\&\&\, (\text{node.Location} = \text{INCAMPUS}) \Rightarrow$$
$$(\text{channel.Security} = \text{low} \;\; [\text{until}] \; \text{node.Location} = \text{OUTSIDE})$$

$$\text{Insecurity}_{\text{R1}} = \text{Insecurity}_{\text{location} \,\wedge\, (\text{CPUSensor} \vee \text{MemSensor})} = \text{Insecurity}_{\text{location}} * \text{Insecurity}_{(\text{CPUSensor} \vee \text{MemSensor})} =$$
$$\text{Insecurity}_{\text{location}} * (\text{Insecurity}_{\text{CPUSensor}} + \text{Insecurity}_{\text{MemSensor}} - \text{Insecurity}_{\text{CPUSensor}} * \text{Insecurity}_{\text{MemSensor}})$$

Εικόνα 7.2 Υπολογισμός Ανασφάλειας για Έναν Κανόνα.

Οι παράμετροι ποιότητας των προμηθευτών και τα κατώφλια αξιοπιστίας για κάθε κανόνα λαμβάνονται σαν είσοδο από τον *ContextManager* κατά την αρχικοποίηση του συστήματος. Τα κατώφλια αποθηκεύονται στο δέντρο, στον αντίστοιχο κόμβο για κάθε

κανόνα. Οι παράμετροι ποιότητας των προμηθευτών αποθηκεύονται προσωρινά σε μία λίστα, όπου κάθε κόμβος αντιστοιχεί σε έναν προμηθευτή. Έπειτα διασχίζεται το δέντρο με τους κανόνες και για κάθε κανόνα υπολογίζονται οι συνολικές πιθανότητες σύμφωνα με την παραπάνω μέθοδο. Οι πιθανότητες για κάθε κανόνα, αποθηκεύονται στον αντίστοιχο κόμβο του δέντρου.

Κάθε φορά που ο *RulesEvaluator* αποφασίζει την εκτέλεση των ενεργειών κάποιου κανόνα, συγκρίνει τις τιμές των παραμέτρων ποιότητας, που είναι αποθηκευμένες στον κόμβο του κανόνα, με τα αντίστοιχα κατώφλια. Ο υπολογισμός των συνολικών τιμών έχει γίνει από πριν και συνεπώς ο έλεγχος αξιοπιστίας δεν επιβαρύνει χρονικά την υπηρεσία, κατά την αποτίμηση των κανόνων. Αν ένας κανόνας κριθεί αναξιόπιστος δεν αφαιρείται από το δέντρο, παρόλο που δεν υπάρχει περίπτωση να εκτελεστούν οι ενέργειές του. Ο λόγος είναι ότι οι τιμές των παραμέτρων ποιότητας των προμηθευτών μπορεί να αλλάξουν από το χρήστη ή τον προγραμματιστή και συνεπώς να αλλάξει η αξιοπιστία του κανόνα.

7.8 Δυναμική Διαμόρφωση Υπηρεσίας

Το υποσύστημα *RulesRecordManager* περιλαμβάνει ένα σύνολο μεθόδων για τη δυναμική διαμόρφωση της υπηρεσίας από το χρήστη και παρέχει δύο *GUIs* που μπορεί να χρησιμοποιήσει ο χρήστης για να προσπελάσει αυτές τις μεθόδους: ένα για τη διαμόρφωση των κανόνων και ένα για τη διαμόρφωση των πιθανοτήτων αξιοπιστίας των προμηθευτών.

7.8.1 Διαμόρφωση Κανόνων

Το *GUI* για τη διαμόρφωση των κανόνων περιλαμβάνει επιλογές για τις εξής λειτουργίες:

Εμφάνιση Κανόνων

Ο χρήστης μπορεί να δει στην οθόνη της συσκευής, όλους τους κανόνες που βρίσκονται εκείνη τη στιγμή στο δέντρο. Οι κανόνες εμφανίζονται με τη σειρά εισαγωγής τους, συνοδευόμενοι από ένα αύξων *id*.

Εισαγωγή Νέου Κανόνα

Ο χρήστης μπορεί να γράψει ένα νέο κανόνα στην οθόνη της συσκευής του, χρησιμοποιώντας τα πλήκτρα, όπως στην περίπτωση του *SMS*. Μετά την εισαγωγή καλείται ο *RulesParser*, ο οποίος αποσυνθέτει τον κανόνα και τον αποθηκεύει στο δέντρο.

Διαγραφή Κανόνα

Ο χρήστης μπορεί να διαγράψει ένα κανόνα. Ο κανόνας επιλέγεται με το *id* του και διαγράφεται από το δέντρο.

Αποθήκευση Κανόνων

Ο χρήστης μπορεί να αποθηκεύσει την τρέχουσα μορφή των κανόνων στο αρχείο με τους κανόνες, αν επιθυμεί να αποθηκευτούν οι αλλαγές που έχει κάνει. Αν το αρχείο υπάρχει, σβήνεται και δημιουργείται καινούργιο.

Διαγραφή Αρχείου Κανόνων

Ο χρήστης μπορεί να διαγράψει το αρχείο με τους κανόνες αν το επιθυμεί. Αυτό μπορεί να γίνει είτε για εξοικονόμηση χώρου, αν δεν υπάρχει κάποιος κανόνας που το κάνει αυτόματα ή για να σβήσει άμεσα όλους τους κανόνες και να προσθέσει άλλους στη συνέχεια.

7.8.2 Διαμόρφωση Πιθανοτήτων Αξιοπιστίας

Το *GUI* για τη διαμόρφωση των πιθανοτήτων αξιοπιστίας περιλαμβάνει επιλογές για τις εξής λειτουργίες:

Εμφάνιση Πιθανοτήτων

Ο χρήστης μπορεί να δει στην οθόνη της συσκευής του τις πιθανότητες των παραμέτρων αξιοπιστίας κάθε προμηθευτή.

Αλλαγή Πιθανοτήτων

Ο χρήστης μπορεί να επιλέξει μία πιθανότητα και να αλλάξει τη τιμή της.

Αποθήκευση Πιθανοτήτων

Ο χρήστης μπορεί να αποθηκεύσει τις τρέχουσες πιθανότητες αξιοπιστίας στο αρχείο, αν επιθυμεί να αποθηκευτούν οι αλλαγές που έχει κάνει. Αν το αρχείο υπάρχει, σβήνεται και δημιουργείται καινούργιο.

Διαγραφή Αρχείου Πιθανοτήτων

Ο χρήστης μπορεί να σβήσει το αρχείο με τις πιθανότητες αξιοπιστίας των προμηθευτών είτε για εξοικονόμηση χώρου ή για να εισάγει νέες πιθανότητες για όλους τους προμηθευτές.

Απενεργοποίηση Ελέγχου Αξιοπιστίας

Ο χρήστης μπορεί να απενεργοποιήσει τον έλεγχο αξιοπιστίας των κανόνων. Όταν ο έλεγχος αξιοπιστίας απενεργοποιείται, οι ενέργειες των κανόνων που ικανοποιούνται εκτελούνται χωρίς να ελεγχθεί, αν οι τρεις πιθανότητες που χαρακτηρίζουν την αξιοπιστία του κανόνα είναι μικρότερες από τα αντίστοιχα κατώφλια. Η απενεργοποίηση του ελέγχου αξιοπιστίας μπορεί να γίνει, για να βελτιωθεί η χρονική απόκριση της εφαρμογής. Όπως θα δούμε στο Κεφάλαιο 9, η χρονική επιβάρυνση που προσθέτει ο έλεγχος αξιοπιστίας, είναι αμελητέα κατά την επεξεργασία των γεγονότων, αλλά είναι ιδιαίτερα σημαντική, κατά τη διάρκεια της συντακτικής ανάλυσης των κανόνων.

Ενεργοποίηση Ελέγχου Αξιοπιστίας

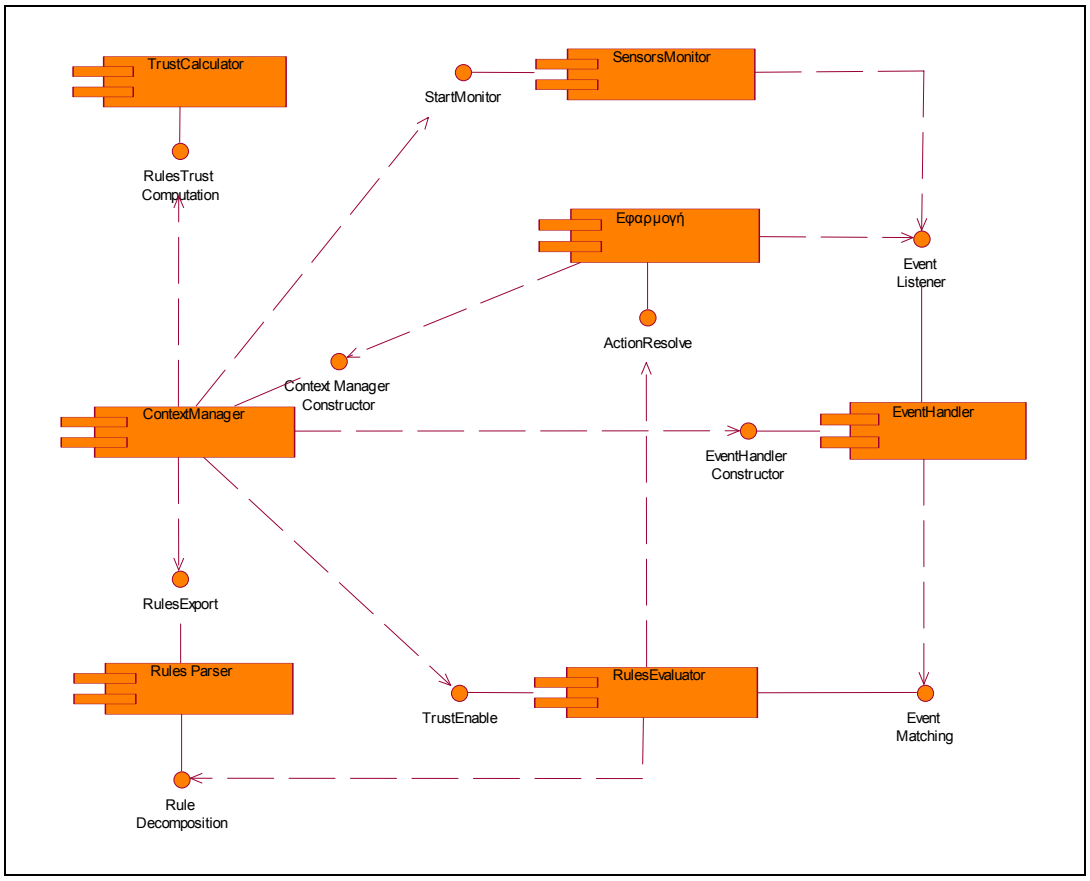
Ο χρήστης μπορεί να ενεργοποιήσει τον έλεγχο αξιοπιστίας αν είχε απενεργοποιηθεί πιο πριν. Η ενεργοποίηση του ελέγχου αξιοπιστίας μπορεί να γίνει και σαν αποτέλεσμα της ικανοποίησης κάποιου κανόνα, π.χ.: για την αύξηση της ασφάλειας της εφαρμογής.

7.9 Εξαρτήσεις Μεταξύ των Υποσυστημάτων

Στο Σχήμα 7.5 φαίνεται το διάγραμμα των εξαρτήσεων μεταξύ των υποσυστημάτων της υπηρεσίας και της εφαρμογής. Το Σχήμα 7.5 περιγράφει τις διεπαφές που χρησιμοποιούν τα υποσυστήματα για να αλληλεπιδρούν.

Ο *ContextManager* επικοινωνεί με τον *RulesParser* καλώντας τη μέθοδο που εκτελεί τη συντακτική ανάλυση στους κανόνες. Επιπλέον αρχικοποιεί τον *EventHandler*

κατασκευάζοντας ένα αντικείμενο του. Στη συνέχεια περνάει μια αναφορά του αντικειμένου στον *SensorsMonitor*, καλώντας τη μέθοδο που ξεκινάει την παρακολούθηση των προμηθευτών. Αν ενεργοποιηθεί ο έλεγχος αξιοπιστίας, ο *ContextManager* καλεί μία μέθοδο του *TrustCalculator* για τον υπολογισμό της αξιοπιστίας των κανόνων. Παράλληλα ειδοποιεί τον *RulesEvaluator* να ενεργοποιήσει ή να απενεργοποιήσει τον έλεγχο της αξιοπιστίας στους κανόνες.



Σχήμα 7.5 Εξαρτήσεις των Υποσυστημάτων και της Εφαρμογής.

Ο *SensorsMonitor* επικοινωνεί με τον *EventHandler* καλώντας τη μέθοδο *EventListener* που συλλέγει τις τιμές του *context*. Η μέθοδος αυτή καλείται μέσα στο νήμα παρακολούθησης, κάθε φορά που ανιχνεύεται ένα γεγονός.

Ο *EventHandler* επικοινωνεί με τον *RulesEvaluator* καλώντας τη μέθοδο που ελέγχει την αποτίμηση των κανόνων, με βάση το καινούργιο γεγονός. Η μέθοδος αυτή καλείται από το νήμα της ουράς, για κάθε γεγονός που εξάγεται για αποτίμηση.

Ο *RulesEvaluator* επικοινωνεί μόνο με την εφαρμογή, μέσω της *callback* μεθόδου, που έχει υλοποιήσει ο προγραμματιστής της εφαρμογής. Η κλήση αυτής της μεθόδου προσαρμόζει αυτόματα τη λειτουργία της εφαρμογής.

Η εφαρμογή καλεί τον κατασκευαστή του *ContextManager* και ενεργοποιεί τη λειτουργία της υπηρεσίας. Επιπλέον είναι δυνατόν να επικοινωνήσει με τον *EventHandler* για την αποστολή γεγονότων, μέσω του *EventListener*. Τα γεγονότα που στέλνει η εφαρμογή απευθείας στον *EventHandler*, αφορούν παραμέτρους του *context* που μεταβάλλονται μέσα στην εφαρμογή και πρέπει οι τιμές τους να συλλεχθούν εκεί. Για τη συλλογή γεγονότων θα αναφερθούμε ξανά στο Κεφάλαιο 8.

ΚΕΦΑΛΑΙΟ 8. ΥΛΟΠΟΙΗΣΗ ΥΠΗΡΕΣΙΑΣ ΚΑΙ ΑΝΑΠΤΥΞΗ ΚΙΝΗΤΗΣ ΕΦΑΡΜΟΓΗΣ

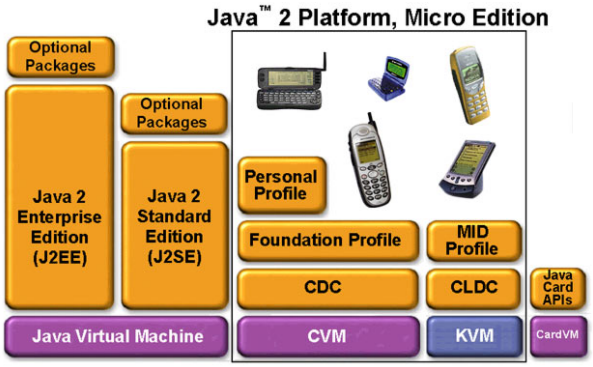
Στο κεφάλαιο αυτό περιγράφεται η υλοποίηση της υπηρεσίας και δίνονται προγραμματιστικές λεπτομέρειες και στοιχεία για την ανάπτυξη του ενδιάμεσου λογισμικού. Προκειμένου να ελέγξουμε τη χρηστικότητα και την ορθή λειτουργία του ενδιάμεσου λογισμικού υλοποιήσαμε μία κινητή εφαρμογή πάνω από την υπηρεσία. Επιπλέον περιγράφεται η διασύνδεση της εφαρμογής με την υπηρεσία και η επικοινωνία μεταξύ τους.

8.1 Πλατφόρμα Υλοποίησης

Η υλοποίηση της υπηρεσίας έγινε στην πλατφόρμα *J2ME (Java 2 Micro Edition)* της *Java*. Οι τεχνολογίες αυτής της πλατφόρμας χρησιμοποιούνται από ένα πλήθος κινητών συσκευών, όπως κινητά τηλέφωνα, *PDAs*, φορέσιμες συσκευές, ενσωματωμένα σύστημα αυτοκινήτων κ.α. Αποτελείται από ένα σύνολο από καθορισμένα πρότυπα, *Java APIs* και προαιρετικά πακέτα. Στην Εικόνα 8.1 φαίνεται η αρχιτεκτονική της πλατφόρμας *J2ME* [L8].

Η πλατφόρμα *J2ME* περιλαμβάνει ένα σύνολο συστατικών που επιτρέπουν την ανάπτυξη εφαρμογών για κινητές συσκευές. Τα συστατικά αυτά απευθύνονται σε ποικίλες συσκευές με διαφορετικές δυνατότητες και χαρακτηριστικά. Η *J2ME* υποστηρίζεται από δύο κατηγορίες κινητών συσκευών, με βάση το μέγεθος των πόρων που διαθέτουν. Για κάθε μία

παρέχει ξεχωριστό περιβάλλον εκτέλεσης. Ακολουθεί μια σύντομη περιγραφή των συστατικών της πλατφόρμας *J2ME*.



Εικόνα 8.1 Αρχιτεκτονική της Πλατφόρμας J2ME.

CDC

Το *CDC (Connected Device Configuration)* αποτελεί την προδιαγραφή για το ένα από τα δύο περιβάλλοντα εκτέλεσης της *J2ME* (*CDC* και *CLDC*). Παρέχει μια εικονική μηχανή (*CVM*) και μαζί με το *FP* (*Foundation Profile)* αποτελεί ένα πλήρες περιβάλλον εκτέλεσης *J2ME*. Απευθύνεται σε κινητές συσκευές μεσαίου μεγέθους, όπως μεγάλα *PDAs*, συστήματα πλοήγησης αυτοκινήτων κ.α. Οι συσκευές αυτές έχουν σχετικά περιορισμένους υπολογιστικούς πόρους.

FP

Το *FP* είναι το βασικό προφίλ του *CDC*. Περιλαμβάνει τα *APIs* για την ανάπτυξη εφαρμογών πάνω από το *CDC*. Το *FP* προορίζεται για εφαρμογές που δεν απαιτούν γραφικές διεπαφές. Άλλα προφίλ, όπως το *Personal Basis Profile* και το *Personal Profile* είναι χτισμένα πάνω στο *FP* παρέχοντας γραφικές διεπαφές και άλλες επιπλέον δυνατότητες.

CLDC

Το *CLDC* (*Connected Limited Device Configuration*) είναι η βάση του περιβάλλοντος εκτέλεσης της *J2ME*, που στοχεύει σε μικρές κινητές συσκευές με ιδιαίτερα περιορισμένους υπολογιστικούς πόρους. Οι συσκευές αυτές περιλαμβάνουν κυρίως απλά κινητά τηλέφωνα, μικρούς ψηφιακούς βοηθούς κ.α. Το *CLDC* περιλαμβάνει δύο εικονικές μηχανές για την εκτέλεση των εφαρμογών, την *KVM* και την *CLDC HotSpot*. Η *KVM* σχεδιάστηκε για συσκευές με εξαιρετικά μικρές υπολογιστικές δυνατότητες, που μπορούν να υποστηρίξουν

υποτυπώδεις κινητές εφαρμογές[4]. Η *CLDC HotSpot* είναι σχεδιασμένη για συσκευές νέας γενιάς, με περισσότερη διαθέσιμη μνήμη. Οι βασικές συσκευές που υλοποιούν το *CLDC* για την ανάπτυξη εφαρμογών είναι τα κινητά τηλέφωνα.

MIDP

Το *MIDP* (*Mobile Information Device Profile*) μαζί με το *CLDC* συνθέτουν το περιβάλλον εκτέλεσης *Java* για κινητές συσκευές περιορισμένων πόρων. Το *MIDP* περιλαμβάνει ένα σύνολο από *APIs* που παρέχουν τη βασική λειτουργικότητα για κινητές εφαρμογές, όπως διεπαφές χρηστών, δικτυακή επικοινωνία, τοπικό αποθηκευτικό χώρο και διαχείριση του κύκλου ζωής των εφαρμογών.

Προαιρετικά Πακέτα

Η πλατφόρμα *J2ME* περιλαμβάνει ένα σύνολο προαιρετικών πακέτων, που επεκτείνουν τη λειτουργικότητα των κινητών εφαρμογών. Τα πιο γνωστά από αυτά είναι το *MMA* (*Mobile Media API*) για υπηρεσίες πολυμέσων, το *WMA* (*Wireless Messaging API*) για ασύρματη αποστολή μηνυμάτων, το *WSA* (*Web Services API*) για προσπέλαση υπηρεσιών διαδικτύου από την κινητή εφαρμογή κ.α. Διάφορα άλλα *APIs* παρέχονται από συγκεκριμένους κατασκευαστές κινητών συσκευών και είναι συμβατά μόνο με συγκεκριμένες συσκευές, όπως το *Location API* της *NOKIA*.

Επιλογή Πλατφόρμας

Η βασική αντίπαλη τεχνολογία της *J2ME* είναι το *.Net Compact Framework* της *Microsoft*. Και οι δύο τεχνολογίες υποστηρίζουν ένα πλήθος λειτουργικών και προγραμματιστικών εργαλείων που διευκολύνουν την αποτελεσματική ανάπτυξη κινητών εφαρμογών. Η βασική διαφορά τους είναι η ανεξαρτησία από *hardware* πλατφόρμες.

Οι τεχνολογίες *J2ME* τρέχουν σε ένα μεγάλο εύρος συσκευών. Οι προδιαγραφές της *J2ME* έχουν υλοποιηθεί από όλους τους βασικούς κατασκευαστές κινητών τηλεφώνων, όπως *Nokia, Ericsson, Motorola, Siemens* κ.α. Οποιαδήποτε εφαρμογή σε *MIDP* (θεωρητικά) μπορεί να τρέξει σε όλες τις συσκευές που υποστηρίζουν εφαρμογές *J2ME*. Συνεπώς, το λογισμικό *MIDP* ικανοποιεί σε μεγάλο βαθμό τη βασική απαίτηση της *μεταφερσιμότητας*.

[4] Το όνομα της K Virtual Machine, δηλώνει ότι το μέγεθός της είναι της τάξεως των δεκάδων από Kilobytes.

Αντίθετα, το *.Net Compact Framework* τρέχει μόνο στις συσκευές που έχουν την άδεια *Pocket PC* της *Microsoft*. Προτιμήσαμε την πλατφόρμα με τη μεγαλύτερη διαθεσιμότητα, η οποία επιπλέον παρέχονταν και δωρεάν.

8.2 Κινητές Εφαρμογές

Οι εφαρμογές που υλοποιούνται σε *MIDP* ονομάζονται *MIDlets*. Το *MIDlet* είναι το κωδικό όνομα για τις εφαρμογές, που έχουν αναπτυχθεί με τεχνολογίες *Java* και τρέχουν πάνω σε ασύρματες και κινητές συσκευές. Τα *MIDlets* είναι το ανάλογο των *Applets*, για κινητές συσκευές. Όπως και τα *Applets*, τα *MIDlets* ελέγχονται από το λογισμικό που τα τρέχει. Στην περίπτωση των *Applets*, το λογισμικό στο οποίο τρέχουν είναι ένας *browser* ή κάποιο ανάλογο εργαλείο. Τα *MIDlets* τρέχουν στην υλοποίηση του *CLDC* σε μία κινητή συσκευή.

Η υπηρεσία ενδιάμεσου λογισμικού υλοποιήθηκε αποκλειστικά σε *MIDP*, ώστε να μπορεί να τρέξει κάτω από *MIDlets*. Για την υλοποίηση και τον έλεγχο του λογισμικού χρησιμοποιήθηκαν προσομοιωτές κινητών συσκευών. Συγκεκριμένα το λογισμικό αναπτύχθηκε και ελέγχθηκε με τα εργαλεία *Nokia Developers Suite 2.2* [L9] της *Nokia* και *J2ME Wireless Toolkit (WTK) 2.2* [L10] της *SUN*. Διάφορα άλλα εργαλεία προσομοίωσης παρέχονται από κατασκευαστές όπως η *Siemens*, η *Motorola* και η *Ericsson*.

Οι προσομοιωτές στοχεύουν τόσο, στη προσομοίωση της λειτουργίας μιας συσκευής, όσο και στα ιδιαίτερα χαρακτηριστικά κάποιου πραγματικού μοντέλου. Παρέχουν ένα σύνολο επιλογών για την ακριβέστερη προσομοίωση στην εκτέλεση της εφαρμογής. Επιπλέον παρέχουν εργαλεία για την παρακολούθηση των βασικών παραμέτρων στην εκτέλεση της εφαρμογής, όπως η κατανάλωση μνήμης, η χρήση της *CPU* κ.α.

8.3 Υλοποίηση Υπηρεσίας

Η υπηρεσία παρέχεται με τη μορφή ενός *MIDP* πακέτου. Κάθε υποσύστημα της υπηρεσίας υλοποιήθηκε σαν μια ξεχωριστή κλάση με μεθόδους και παραμέτρους που υλοποιούν τις λειτουργίες του υποσυστήματος. Επίσης το πακέτο περιλαμβάνει και κάποιες

βοηθητικές κλάσεις. Σχεδιαστικά, ίσως ήταν προτιμότερο οι λειτουργίες των υποσυστημάτων να κατανεμηθούν σε περισσότερες κλάσεις επιτυγχάνοντας μία περισσότερο δομημένη αρχιτεκτονική. Οι κλάσεις που υλοποιήθηκαν είναι οι ελάχιστες απαραίτητες, αφού μεγάλος αριθμός κλάσεων, μειώνει τη χρονική απόδοση, ενώ αυξάνει το μέγεθος του τελικού εκτελέσιμου αρχείου (*JAR*) που θα φορτωθεί στη μνήμη της κινητής συσκευής.

8.3.1 Πακέτο contextManager

Συνοπτικά η υπηρεσία αποτελείται από τα εξής:

Package
contextManager

Interface
ContextManagement

Classes
ContextManager
RulesParser
SensorsMonitor
RulesEvaluator
EventHandler
TrustCalculator
RulesRecordManagement

Οι κλάσεις του πακέτου υλοποιούν τα ομώνυμα υποσυστήματα που περιγράφηκαν στο Κεφάλαιο 7. Η διεπαφή *ContextManagement* ορίζει τις μεθόδους που συνδέουν μία εφαρμογή με την υπηρεσία και περιγράφεται στη συνέχεια.

Αν κάποιος θέλει να υλοποιήσει μια *CA* εφαρμογή, μπορεί να το κάνει χρησιμοποιώντας το πακέτο *contextManager*. Για να χρησιμοποιηθεί η υπηρεσία από μία εφαρμογή, ο προγραμματιστής της εφαρμογής θα πρέπει να προσθέσει κώδικα διασύνδεσης. Η διασύνδεση εφαρμογής - υπηρεσίας συνοψίζεται στις ακόλουθες ενέργειες:

1. Φορτώνεται (*import)* στην εφαρμογή το πακέτο *contextmanager*, κάνοντας διαθέσιμες τις ορατές κλάσεις του.
2. Η βασική κλάση (*public*) της εφαρμογής που επεκτείνει την κλάση *MIDlet*, πρέπει να υλοποιεί (*implements*) τη διεπαφή *ContextManagement*. Η διεπαφή αυτή περιλαμβάνει τρεις μεθόδους που καθορίζουν την αλληλεπίδραση της εφαρμογής με την υπηρεσία.
3. Κατά την έναρξη της εφαρμογής (είτε στο κατασκευαστή, είτε στη μέθοδο *startApp()* που ξεκινά η ροή του *MIDlet*) κατασκευάζεται ένα αντικείμενο της κλάσης *ContextManager*. Ο κατασκευαστής καλείται με όρισμα τον πίνακα των κανόνων και των παραμέτρων ποιότητας και αυτόματα γίνονται κάποιες διαδικασίες αρχικοποίησης της υπηρεσίας.
4. Στη μέθοδο *destroyApp()* που τερματίζει το *MIDlet* της εφαρμογής καλείται η μέθοδος του *ContextManager*, *stopMonitor()*, που τερματίζει το νήμα παρακολούθησης των προμηθευτών.

8.3.2 Διεπαφή ContextManagement

Η διεπαφή *ContextManagement* ορίζει τρεις μεθόδους για τη διασύνδεση με μία εφαρμογή. Ο προγραμματιστής της εφαρμογής πρέπει απαραίτητα να υλοποιήσει τη διεπαφή συμπληρώνοντας το σώμα των μεθόδων. Η διεπαφή περιλαμβάνει τις εξής μεθόδους:

Interface ContextManagement

- `public String [] defineContextRules(String [] rules)`
- `public String [] defineSensorTrust(String [] trust)`
- `public void ActionResolve(String property, String value);`

Ακολουθεί περιγραφή των μεθόδων της διεπαφής.

- ***public String [] defineContextRules(String [] rules)***

Στη μέθοδο αυτή ο προγραμματιστής καταχωρεί τους κανόνες του *context*. Δημιουργεί ένα πίνακα από συμβολοσειρές, του οποίου κάθε στοιχείο είναι ένας κανόνας. Η μέθοδος επιστρέφει τον πίνακα στον *ContextManager* κατά την έναρξη της υπηρεσίας. Αν υπάρχει το αρχείο με τους κανόνες στη μνήμη της συσκευής, η υπηρεσία θα τους διαβάσει από εκεί (Σχήμα 7.2). Αν ο πίνακας δεν οριστεί μέσα στη μέθοδο, το αρχείο με τους κανόνες δεν θα πρέπει να σβηστεί ποτέ από τη μνήμη της συσκευής.

Η μέθοδος δεν περιορίζει τον προγραμματιστή στον τρόπο κατασκευής του πίνακα. Μέσα στη μέθοδο μπορεί να υλοποιηθεί οποιαδήποτε διαδικασία απόκτησης των κανόνων. Για παράδειγμα μπορεί η μέθοδος μπορεί να υλοποιεί μία δικτυακή σύνδεση σε ένα κόμβο προσπελάσιμο μέσω δικτύου και να λαμβάνει ένα αρχείο με κανόνες. Το αρχείο μπορεί να είναι σε μορφή εγγραφών και να βρίσκεται σε κάποια άλλη κινητή συσκευή, οπότε μπορεί να αποκτηθεί ασύρματα μέσω *Bluetooth, SMS* κ.α. Μπορεί όμως να είναι και απλό αρχείο κειμένου (*text*) και να βρίσκεται σε κάποιο γνωστό εξυπηρέτη στο διαδίκτυο, από όπου μπορούν να διαβαστούν τα περιεχόμενά του μέσω *http* σύνδεσης.

- ***public String [] defineSensorTrust(String [] trust)***

Σε αυτήν τη μέθοδο ο προγραμματιστής κατασκευάζει ένα πίνακα με τις παραμέτρους αξιοπιστίας των προμηθευτών και τα κατώφλια των κανόνων. Επιπλέον στον πίνακα ορίζονται οι παράμετροι του *context* με τις οποίες σχετίζεται ο κάθε προμηθευτής. Ο *TrustCalculator* χρειάζεται αυτή την αντιστοιχία για να υπολογίσει την αξιοπιστία των κανόνων. Ο χειρισμός της μεθόδου είναι ο ίδιος με την *defineContextRules*.

- ***public void ActionResolve(String property, String value)***

Η μέθοδος αυτή ερμηνεύει τις ενέργειες που προκύπτουν από την ικανοποίηση των κανόνων, σε ενέργειες προσαρμογής της εφαρμογής. Η μέθοδος αποτελεί *callback* και καλείται μόνο από την υπηρεσία. Συγκεκριμένα καλείται από τον *RulesEvaluator* για την

πραγματοποίηση των ενεργειών που αποφασίζει. Για κάθε ενέργεια ενός κανόνα που ικανοποιείται, ο *RulesEvaluator* καλεί τη μέθοδο με ορίσματα μία παράμετρο του *context* και την τιμή που της αναθέτει η ενέργεια. Αν η ενέργεια δεν αναθέτει κάποια τιμή σε μία παράμετρο, αλλά εκτελεί μία μέθοδο χωρίς ορίσματα, στο δεύτερο όρισμα περνάει *null*.

Ο προγραμματιστής υλοποιεί στο σώμα της μεθόδου μία αντιστοίχιση των παραμέτρων του *context* με μεθόδους της εφαρμογής, οι οποίες υλοποιούν τις ενέργειες προσαρμογής. Οι τιμές των παραμέτρων, αν δεν είναι *null*, περνάνε σαν ορίσματα στις μεθόδους της εφαρμογής. Αν η τιμή είναι *null* σημαίνει ότι, η συγκεκριμένη μέθοδος που πρέπει να εκτελεστεί δεν έχει ορίσματα.

Οι ενέργειες που αποφασίζονται σε έναν κανόνα μπορεί να μην αφορούν μόνο την εφαρμογή, αλλά και την υπηρεσία. Για παράδειγμα η ικανοποίηση ενός κανόνα μπορεί να έχει σαν αποτέλεσμα το σβήσιμο του αρχείου των κανόνων, ή την απενεργοποίηση του ελέγχου αξιοπιστίας, ή τον τερματισμό του νήματος παρακολούθησης των προμηθευτών. Οι ενέργειες αυτές υλοποιούνται με κλήσεις στις αντίστοιχες μεθόδους του *ContextManager*. Συνεπώς οι μέθοδοι που καλούνται μέσα από την *ActionResolve()* μπορεί να είναι και μέθοδοι της υπηρεσίας.

Η υπηρεσία καλώντας αυτή τη μέθοδο προσαρμόζει αυτόματα τη λειτουργία της εφαρμογής, αλλά και τη δικιά της, στις αλλαγές του *context*. Η υλοποίηση της μεθόδου πρέπει να αντιστοιχίζει όλες τις πιθανές ενέργειες που επιστρέφει η υπηρεσία σε μεθόδους που υλοποιούν τις ζητούμενες λειτουργίες. Μια τυπική υλοποίηση της μεθόδου είναι μια σειρά από *if-else if* προτάσεων.

8.4 Χρήση Υπηρεσίας από Μία Κινητή Εφαρμογή

Για να γίνει επίδειξη της υπηρεσίας και να ελεγχθεί η λειτουργία της σε πραγματικές συνθήκες υλοποιήθηκε μια *MIDP* εφαρμογή για κινητές συσκευές. Υλοποιήθηκε η διασύνδεση της εφαρμογής με την υπηρεσία ακολουθώντας τα βήματα που περιγράφηκαν παραπάνω και πραγματοποιήθηκαν διάφορα σενάρια λειτουργίας. Η εφαρμογή, όπως και η υπηρεσία, υλοποιήθηκε και εκτελέστηκε με τους προσομοιωτές του *WTK* και της *Nokia*.

Η υλοποίηση της εφαρμογής δεν είχε στόχο την κατασκευή ενός σύνθετου *MIDlet* με πολλές λειτουργικές δυνατότητες. Το ζητούμενο ήταν, να υλοποιηθεί μία εφαρμογή, η οποία να εξαρτάται από πραγματικό *context* και μέσω της υπηρεσίας να χρησιμοποιεί το *context* για να προσαρμόζει τις λειτουργίες της. Για το λόγο αυτό ορίστηκαν διάφορες παράμετροι του *context* και υλοποιήθηκαν οι μηχανισμοί που προμηθεύουν τις τιμές των παραμέτρων στην υπηρεσία.

8.4.1 Περιγραφή Εφαρμογής

Το *MIDlet* που υλοποιήσαμε αποτελεί ένα βοηθό κινητού τηλεφώνου (για αυτό ονομάστηκε *MobiPal*) και παρέχει στο χρήστη βασικές υπηρεσίες όπως:

- Αποστολή και λήψη *SMS*.
- Χειρισμό εισερχόμενων/εξερχόμενων κλήσεων.
- Δυνατότητες αναπαραγωγής πολυμέσων.
- Σύνδεση στο *Internet*.
- Επιλογή μοτίβου μπαταρίας.
- Επιλογή μοτίβου *privacy*.
- Ενεργοποίηση/Απενεργοποίηση ασφάλειας.

Επιπλέον, στο μενού επιλογών της εφαρμογής προστέθηκαν επιλογές εμφάνισης των γραφικών διεπαφών που παρέχει η υπηρεσία, για τη διαμόρφωση των κανόνων και την ενεργοποίηση και απενεργοποίηση του ελέγχου αξιοπιστίας. Στην Εικόνα 8.2 φαίνονται οι δύο αρχικές γραφικές διεπαφές της εφαρμογής.

Εικόνα 8.2 Διεπαφές από την Έναρξη της Υπηρεσίας σε Προσομοιωτές της NOKIA.

Στη συνέχεια περιγράφονται οι λειτουργίες της εφαρμογής και οι επιλογές καθορισμού μοτίβου λειτουργίας.

Αποστολή και Λήψη *SMS*

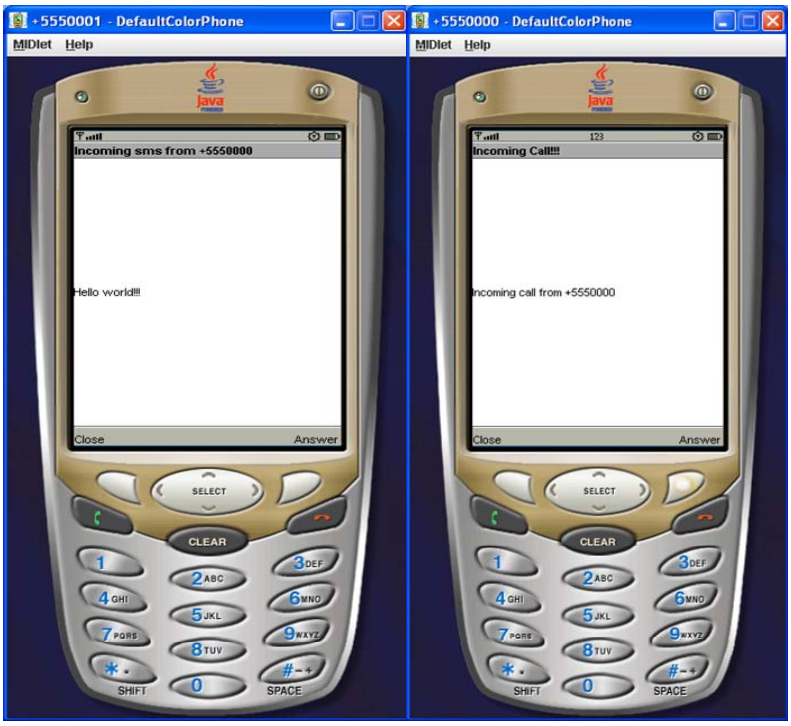
Η εφαρμογή μπορεί να στέλνει και να δέχεται γραπτά μηνύματα τα οποία και εμφανίζει στην οθόνη του κινητού (Εικόνα 8.3(α)). Η αποστολή των μηνυμάτων γίνεται στον αριθμό που αντιστοιχεί σε κάθε προσομοιωτή (π.χ. *+5550000*). Για την υλοποίηση αυτής της λειτουργίας χρησιμοποιήθηκε επιπρόσθετα το *WMA 2.0*.

Χειρισμός Κλήσεων

Η εφαρμογή μπορεί να δέχεται και να κάνει κλήσεις σε άλλες συσκευές. Το *MIDP* παρέχει μία μέθοδο για πραγματοποίηση εξερχόμενης κλήσης σε άλλη συσκευή, η οποία όμως δεν λειτουργεί στους προσομοιωτές. Για το λόγο αυτό η κλήση υλοποιήθηκε έμμεσα στέλνοντας ένα *SMS* με το περιεχόμενο *call*. Αυτό αποτελεί αρκετά ρεαλιστική λύση αφού η διαδικασία με την οποία μια εφαρμογή «ακούει» εισερχόμενες κλήσεις και *SMSs* είναι παρόμοια (Εικόνα 8.3(β)).

Η εισερχόμενη κλήση μπορεί να συνοδεύεται από ήχο ή να είναι αθόρυβη. Το αρχείο ήχου μπορεί να είναι σε διάφορους τύπους (*wav, mp3* κ.α.) και βρίσκεται αποθηκευμένο στη

μνήμη της συσκευής, όπου αποθηκεύονται τα εκτελέσιμα αρχεία της εφαρμογής. Για την αναπαραγωγή ήχου χρησιμοποιήθηκε το *MMA (Mobile Media API) 2.0*.



(α) (β)

Εικόνα 8.3 (α) Εισερχόμενο SMS. (β) Εισερχόμενη Κλήση.

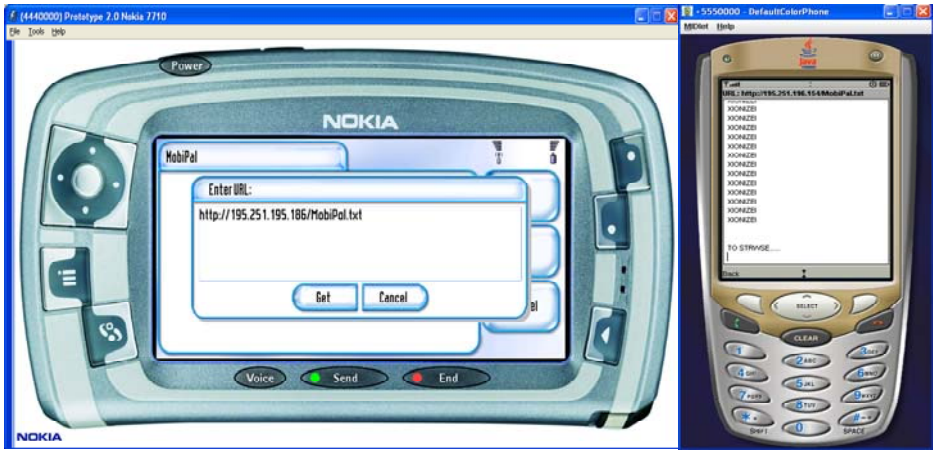
Σύνδεση στο Internet

Η εφαρμογή παρέχει τη δυνατότητα σύνδεσης στο *Internet* μέσω *http*. Συγκεκριμένα ο χρήστης μπορεί:

1. Να ανοίξει από το *browser* της συσκευής του κάποια σελίδα.
2. Να κατεβάσει και να δει στην οθόνη του κινητού του αρχείο κειμένου (*txt*) που βρίσκεται σε κάποιο εξυπηρέτη του *web*.

Η μέθοδος που υλοποιεί την πρώτη λειτουργία απαιτεί η εφαρμογή να υποστηρίζει κάποια τεχνολογία προσπέλασης ιστοσελίδων και *web browser* που επιτρέπει τη σύνδεση σε ειδικές σελίδες του διαδικτύου για κινητά (π.χ. *iMode, c-HTML*). Η μέθοδος αναζητά κάποιο διαθέσιμο *web browser* στο λογισμικό που εκτελείται η εφαρμογή και ανοίγει εκεί τη σελίδα. Στην εκτέλεση της εφαρμογής στον προσομοιωτή, η σελίδα άνοιγε με το *browser* των *Windows*, αφού αυτός υπήρχε διαθέσιμος.

Η δεύτερη λειτουργία υλοποιήθηκε με *http* σύνδεση και μεταφορά των δεδομένων του αρχείου που βρίσκεται στον εξυπηρέτη του *web*. Τα δεδομένα εμφανίζονται στην οθόνη του κινητού. Χρησιμοποιήθηκε και η μέθοδος για σύνδεση με *https* αν το υποστηρίζει ο εξυπηρέτης (Εικόνα 8.4).



Εικόνα 8.4 Εμφάνιση του Αρχείου MobiPal.txt από τον Εξυπηρέτη 195.251.195.186.

Ρυθμίσεις Κατανάλωσης Μπαταρίας

Θεωρούμε ότι η εφαρμογή μπορεί να λειτουργήσει σε 3 μοτίβα λειτουργίας, που σχετίζονται με τη διαθέσιμη μπαταρία:

- *Economy*
- *Silent*
- *Normal*

Τα μοτίβα λειτουργίας της μπαταρίας επηρεάζουν την εκτέλεση κάποιων λειτουργιών της εφαρμογής. Ο χρήστης μπορεί να επιλέξει κάθε στιγμή ένα μοτίβο λειτουργίας από ένα μενού στην οθόνη του κινητού του, ανάλογα με τη διάθεσή του ή τις ανάγκες του (Εικόνα 8.5 (α)). Για παράδειγμα στο μοτίβο "*Economy*" για μια εισερχόμενη κλήση η ειδοποίηση θα ήταν μια ένδειξη στην οθόνη του κινητού, στο "*Silent*" μόνο δόνηση, ενώ στο "*Normal*" δόνηση και ηχητική ειδοποίηση ταυτόχρονα.

Μοτίβο Privacy

Η εφαρμογή μπορεί να λειτουργήσει σε *privacy* μοτίβο. Ο χρήστης μπορεί να επιλέξει από το μενού της εφαρμογής ενεργοποίηση ή απενεργοποίηση του *privacy* μοτίβου (Εικόνα 8.5(β)). Η ενεργοποίηση του *privacy* δηλώνει ότι ο χρήστης αυτή τη στιγμή είναι απασχολημένος και δεν θέλει να ενοχλείται. Οι λειτουργίες της εφαρμογής σε *privacy* μοτίβο εκτελούνται αθόρυβα και διακριτικά προς το χρήστη.

Ρυθμίσεις Ασφαλείας

Ομοίως θεωρούμε ότι υπάρχει ένα μοτίβο ασφαλείας στο οποίο κάποιες λειτουργίες της εφαρμογής παρέχονται με ασφαλή τρόπο (Εικόνα 8.5(γ)). Ο χρήστης μέσω ενός μενού μπορεί να ενεργοποιήσει ή να απενεργοποιήσει την ασφάλεια της εφαρμογής. Η εφαρμογή δεν περιλαμβάνει συγκεκριμένες λειτουργίες ασφαλείας. Η παράμετρος *Security* συμπεριλήφθηκε σαν μία επιπλέον παράμετρος *context*.

(α) (β) (γ)

Εικόνα 8.5 (α) Επιλογής Μοτίβου Μπαταρίας. (β) Επιλογή Μοτίβου Privacy. (γ) Επιλογή Μοτίβου Ασφαλείας.

8.4.2 Διαμόρφωση Υπηρεσίας από την Εφαρμογή

Στο μενού επιλογών της εφαρμογής προσθέσαμε δύο επιλογές που επιτρέπουν τη διαμόρφωση της υπηρεσίας απ' ευθείας από το χρήστη. Η μία επιλογή (*Rules Management*, Εικόνα 8.2) αφορά τη διαμόρφωση του συνόλου των κανόνων. Συγκεκριμένα ενεργοποιείται το *GUI* που παρέχει η υπηρεσία, μέσω του οποίου ο χρήστης μπορεί να δει τους κανόνες, να προσθέσει και να αφαιρέσει κανόνες, να αποθηκεύσει τους κανόνες που υπάρχουν στη μνήμη και να σβήσει το αρχείο με τους κανόνες αν υπάρχει. Στην Εικόνα 8.6 (α) φαίνεται το *GUI* διαμόρφωσης των κανόνων.

Η άλλη επιλογή που έχει ο χρήστης είναι η ενεργοποίηση ή απενεργοποίηση του ελέγχου αξιοπιστίας. Αυτή η επιλογή (*Trustworthiness*, Εικόνα 8.2) ενεργοποιεί ένα άλλο *GUI* της υπηρεσίας, με δύο επιλογές, ενεργοποίησης και απενεργοποίησης του ελέγχου αξιοπιστίας. Στην Εικόνα 8.6 (β) φαίνεται το *GUI* για τον έλεγχο αξιοπιστίας.

(α) (β)

Εικόνα 8.6 (α) Το GUI για τη Διαμόρφωση των Κανόνων. (β) Το GUI για την Ενεργοποίηση/Απενεργοποίηση του Ελέγχου Αξιοπιστίας.

8.5 Context Aware Σχεδίαση της Εφαρμογής

Αφού υλοποιήσαμε την εφαρμογή το επόμενο βήμα ήταν να τη συνδέσουμε με την υπηρεσία και να της ενσωματώσουμε *CA* συμπεριφορά. Αρχικά καθορίστηκε η επιθυμητή συμπεριφορά της εφαρμογής. Κάποιες από τις πιθανές απαιτήσεις που θα μπορούσαν να έχουν οι χρήστες της εφαρμογής είναι οι εξής:

1. Η ειδοποίηση για μια εισερχόμενη κλήση να γίνεται ανάλογα με το *privacy* μοτίβο της εφαρμογής. Δηλαδή η κλήση να είναι ηχητική όταν είναι απενεργοποιημένο το *privacy* μοτίβο και σιωπηλή ή με δόνηση όταν είναι ενεργοποιημένο.
2. Το μοτίβο λειτουργίας της μπαταρίας να καθορίζεται αυτόματα ανάλογα με τη διαθέσιμη μπαταρία της συσκευής.
3. Όταν η διαθέσιμη μνήμη είναι περιορισμένη να προσαρμόζεται ανάλογα η λειτουργία της εφαρμογής.
4. Οι λειτουργίες της εφαρμογής να γίνονται προσαρμοσμένα, ανάλογα με το μοτίβο ασφαλείας. Επιπλέον η ενεργοποίηση του μοτίβου ασφαλείας να συνοδεύεται από ενεργοποίηση του ελέγχου αξιοπιστίας.
5. Αν η εφαρμογή λάβει ένα *SMS*, του οποίου το κείμενο περιέχει ένα *URL*, να ανοίγει αυτόματα το *URL* στο *browser* της συσκευής.

Οι παραπάνω απαιτήσεις λειτουργίας είναι ενδεικτικές για τη συγκεκριμένη εφαρμογή. Πιθανότατα, θα μπορούσαν να καθοριστούν και άλλες απαιτήσεις για τη συμπεριφορά της εφαρμογής, ανάλογα με τις προτιμήσεις του εκάστοτε χρήστη.

8.5.1 Καθορισμός του Context

Με βάση τις λειτουργίες της εφαρμογής και την επιθυμητή συμπεριφορά της καθορίστηκαν οι παράμετροι που συνθέτουν το *context*. Στον Πίνακα 8.1 φαίνονται οι παράμετροι του *context* και τα πεδίο τιμών που λαμβάνουν. Για τις παραμέτρους της εφαρμογής, το πεδίο τιμών θα μπορούσε να είναι οι συμβολοσειρές, καθώς η υπηρεσία μπορεί να κάνει συγκρίσεις και με συμβολοσειρές. Π.χ. το πεδίο τιμών για την παράμετρο *BatteryMode*, θα μπορούσε να είναι {*"Economy", "Silent", "Normal"*}. Η δική μας προτίμηση είναι να γίνεται απαρίθμηση και να χρησιμοποιούνται ακέραιοι.

Πίνακας 8.1 Παράμετροι του Context.

Κατηγορία	Παράμετρος	Πεδίο τιμών
Συσκευής	MemAvail	Int
	BatteryAvail	Int
Εφαρμογής	BatteryMode	enum{Economy, Silent, Normal}
	Security	enum{Disable, Enable}
	Privacy	enum{Disable, Enable}
	incCall	enum{false, true}
	incSMS	enum{false, true}
	containsURL	enum{false, true}
	httpRequest	enum{false,true}

Όπως φαίνεται στον Πίνακα 8.1, διακρίνουμε δύο κατηγορίες παραμέτρων με βάση την προέλευση των τιμών:

1. *Παράμετροι της συσκευής*. Η %διαθέσιμη μνήμη και η %διαθέσιμη μπαταρία είναι οι παράμετροι του *context* που σχετίζονται με τη συσκευή. Το χαρακτηριστικό των παραμέτρων αυτών είναι ότι μεταβάλλονται αρκετά συχνά και απρόβλεπτα. Συνεπώς οι τιμές τους πρέπει να παρακολουθούνται και να ανιχνεύονται οι σημαντικές αλλαγές.
2. *Παράμετροι εφαρμογής*. Οι παράμετροι της εφαρμογής είναι, οι παράμετροι του *context* των οποίων οι τιμές λαμβάνονται από την εφαρμογή. Το χαρακτηριστικό αυτών των παραμέτρων είναι ότι μεταβάλλονται προβλέψιμα. Δηλαδή ο προγραμματιστής γνωρίζει τα σημεία στον κώδικα της εφαρμογής, όπου αυτές οι παράμετροι παίρνουν τιμή. Συνεπώς οι τιμές των παραμέτρων της εφαρμογής δεν απαιτούν παρακολούθηση.

8.5.2 Καθορισμός Κανόνων

Στη συνέχεια καθορίστηκαν οι κανόνες που περιγράφουν την επιθυμητή συμπεριφορά της εφαρμογής, σε σχέση με το *context*. Στον Πίνακα 8.2 φαίνονται οι κανόνες που συνθέσαμε. Οι κανόνες είναι γραμμένοι με το συντακτικό που απαιτεί ο *RulesParser*.

Πίνακας 8.2 Κανόνες Context.

1. *[previous]*((*Privacy*=2) && (BatteryMode!=1)) && (incCall=1) => (BatteryMode=1) && *[next]*(callHandler)
2. ((BatteryMode=3) || (BatteryMode=2)) && (*[once]*BatteryAvail<30) =>(BatteryMode=1 *[until]* BatteryAvail=100)
3. ((BatteryAvail<50) && (BatteryAvail>30)) && (*[always_been]*BatteryAvail>50) => (BatteryMode=2 *[until]* BatteryAvail>=50)
4. (incSMS=1) && (containsURL=1) => (*[next]*browseURL) && (smsHandler)
5. ((incSMS=1) && (containsURL=1)) && (*[previous]*MemAvail<40) => (smsHandler)
6. (*[always_been]*MemAvail>30) && (MemAvail<10) => (freeMem)
7. *[previous]*((Security=1) && (MemAvail>60)) && (MemAvail<20) => *[henceforth]*Security=2)
8. ((BatteryAvail<=30) || *[once]*(BatteryAvail<20)) && (*[always_been]*MemAvail>40) && (*[previous]*Security=2) => *[next]*(BatteryMode=1 *[until]* BatteryAvail=100) && (Security=2)
9. (httpRequest=1) && (*[previous]*Security=1) => (*[next]*httpConnect) && (Security=2)
10. (*[previous]*Security=1) && (Security=2) => (trustCheckEnable)

Σημασιολογία Κανόνων

1. *[previous]*((*Privacy*=2) && (BatteryMode!=1)) && (incCall=1) => (BatteryMode=1) && *[next]*(callHandler)

Ο κανόνας 1 λέει ότι, αν γίνει μια εισερχόμενη κλήση και προηγουμένως το μοτίβο *privacy* ήταν ανενεργό και το μοτίβο της μπαταρίας δεν ήταν στο *Economy*, τότε πρώτα τίθεται το *BatteryMode* σε *Economy* και μετά γίνεται ειδοποίηση για την κλήση. Το νόημα

αυτού του κανόνα είναι ότι, αν το *privacy* μοτίβο είναι ενεργοποιημένο η ειδοποίηση των εισερχόμενων κλήσεων να γίνεται σιωπηλά.

2. ((BatteryMode=3) || (BatteryMode=2)) && (*[once]*BatteryAvail<30) =>
   (BatteryMode=1 *[until]* BatteryAvail=100)

Ο κανόνας 2 λέει ότι αν το μοτίβο της μπαταρίας είναι *Normal* ή *Silent* και κάποια στιγμή η μπαταρία είχε πέσει κάτω από το *30%*, πρέπει να τεθεί το μοτίβο σε *Economy* μέχρι η μπαταρία να ξαναφορτιστεί.

Ο κανόνας εκφράζει ότι, αν η ένδειξη της μπαταρίας πέσει κάποια στιγμή χαμηλά, ακόμα και αν στη συνέχεια ανέβει, η ένδειξη αυτή θα είναι εικονική αφού η πραγματική ένδειξη είναι πολύ μικρότερη. Αυτό είναι ένα χαρακτηριστικό πρόβλημα που έχουν οι μπαταρίες των κινητών ιδίως αυτές που έχουν χρησιμοποιηθεί αρκετά.

Έτσι το μοτίβο θα πρέπει να αλλάξει σε *Economy*, ώστε να γίνεται οικονομία στη μπαταρία, μέχρι η μπαταρία να ξαναφορτιστεί. Εναλλακτικά θα μπορούσε να χρησιμοποιηθεί και ο κανόνας:

(BatteryAvail>50) && (*[once]*BatteryAvail<30) => (BatteryMode=1)

3. ((BatteryAvail<50) && (BatteryAvail>30)) && (*[always_been]*BatteryAvail>50)
   => (BatteryMode=2 *[until]* BatteryAvail>=50)

Ο κανόνας *3* δηλώνει ότι, αν η μπαταρία ήταν πάντα (από τότε που φορτίστηκε τελευταία φορά) μεγαλύτερη από το *50%* της συνολικής και την τρέχουσα χρονική στιγμή είναι μεταξύ *30%* και *50%* τότε, το μοτίβο της μπαταρίας τίθεται στο *Silent*. Αφού η μπαταρία έπεσε κάτω από το μισό, η λειτουργία της συσκευής γίνεται σιωπηλή για να γίνει οικονομία. Ο τελεστής *[always_been]* χρησιμοποιείται για να δηλώσει ότι μέχρι τώρα η ένδειξη της μπαταρίας ήταν πραγματική και όχι εικονική.

4. (incSMS=1) && (containsURL=1) => (*[next]*browseURL) && (smsHandler)

Ο κανόνας αυτός λέει ότι, όταν η εφαρμογή λάβει ένα *SMS* και μέσα στο κείμενο περιέχεται ένα *URL*, πρέπει πρώτα μα εμφανιστεί το μήνυμα στην οθόνη και στη συνέχεια να

ανοίξει αυτόματα στο browser του κινητού το *URL*. Αυτός ο κανόνας αντικατοπτρίζει μια επιθυμία του χρήστη και θα μπορούσε κάλλιστα να έχει προστεθεί από αυτόν.

5. (incSMS=1) && (containsURL=1) && ([previous]MemAvail<40) => (smsHandler)

Ο κανόνας αυτός είναι εναλλακτικός του προηγούμενου και περιέχει έναν επιπρόσθετο περιορισμό. Συγκεκριμένα αν η διαθέσιμη μνήμη ήταν κάτω από το 40% της συνολικής, πριν έρθει το μήνυμα τότε το *URL* δεν θα πρέπει να ανοιχτεί αυτόματα στο browser για να μην επιβαρύνει και άλλο τη μνήμη του κινητού.

6. ([always_been]MemAvail>30) && (MemAvail<10) => (freeMem)

Ο κανόνας αυτός χρησιμοποιείται για να εξοικονομήσει μνήμη για την εφαρμογή, όταν η διαθέσιμη μνήμη πέσει κάτω από το *10%* της συνολικής. Αν η μνήμη ήταν πάντα πάνω από το 3*0%* και πέσει κάτω από *10%*, ίσως είναι ένδειξη ότι εκείνη τη στιγμή εκτελείται μια λειτουργία που δεσμεύει πολύ μνήμη. Συνεπώς πρέπει να εξοικονομηθεί άμεσα μνήμη, ώστε να μην επηρεαστεί η εκτέλεση της εφαρμογής. Η μέθοδος που αντιστοιχεί στην ενέργεια *freeMem* μπορεί να εκτελεί μία απλή συλλογή σκουπιδιών (*System.gc()*) ή κάποια πιο σύνθετη διαδικασία. Μια άλλη πιθανή υλοποίηση θα ήταν να καταστρέφει κάποια μη απαραίτητα αντικείμενα.

7. [previous]((Security=1) && (MemAvail>60)) && (MemAvail<20) =>
   [henceforth]Security=2)

Ο κανόνας αυτός περιγράφει το σενάριο όπου παρατηρείται απότομη πτώση στην ελεύθερη μνήμη, τη στιγμή που η ασφάλεια της εφαρμογής ήταν απενεργοποιημένη. Αυτό το φαινόμενο μπορεί να οφείλεται σε κακόβουλη ενέργεια στο λογισμικό της εφαρμογής μέσω δικτύου και επιτάσσει τη μόνιμη ενεργοποίηση της ασφάλειας στην εφαρμογή.

8. ((BatteryAvail<=30) || [once](BatteryAvail<20)) &&
   ([always_been]MemAvail>40) && ([previous]Security=2) =>
   [next](BatteryMode=1 [until] BatteryAvail=100) && (Security=2)

Αυτός ο κανόνας είναι σύνθετος και θα μπορούσε να σπάσει σε δύο πιο απλούς. Ο κανόνας αυτός δηλώνει ότι όταν η μπαταρία είναι χαμηλή θα πρέπει να απενεργοποιηθούν οι

μηχανισμοί ασφαλείας, που πιθανώς καταναλώνουν μπαταρία και να περάσει η εφαρμογή σε *Economy* μοτίβο. Για να απενεργοποιηθεί η ασφάλεια θα πρέπει να είναι εξασφαλισμένο ότι ποτέ δεν υπήρχαν ενδείξεις για ύποπτη συμπεριφορά της εφαρμογής, όπως ανεξήγητη πτώση μνήμης.

```
9. (httpRequest=1) && ([previous]Security=1) => ([next]httpConnect) &&
(Security=2)
```

Αυτός ο κανόνας επιβάλλει την ενεργοποίηση της ασφάλειας, πριν από τη σύνδεση σε κάποιο *web* εξυπηρέτη. Συγκεκριμένα, αν γίνει μία αίτηση για *http* σύνδεση, θα πρέπει πρώτα να ενεργοποιηθεί η ασφάλεια και έπειτα να γίνει η σύνδεση.

```
10. ([previous]Security=1) && (Security=2) => (trustCheckEnable)
```

Τέλος, ο κανόνας 10, συνδέει την ενεργοποίηση της ασφάλειας, με την ενεργοποίηση του ελέγχου αξιοπιστίας των κανόνων. Ο κανόνας δηλώνει ότι κάθε φορά που ενεργοποιείται η ασφάλεια της εφαρμογής, θα πρέπει να ενεργοποιείται και ο έλεγχος αξιοπιστίας.

Οι δέκα κανόνες που περιγράφηκαν είναι ενδεικτικά παραδείγματα όλων αυτών που θα μπορούσαν να οριστούν για τη συγκεκριμένη εφαρμογή. Σε κάποιες περιπτώσεις, πιθανώς να χρειάζονται και συμπληρωματικοί κανόνες για να εξασφαλισθεί η επιθυμητή λειτουργικότητα της εφαρμογής. Η σωστή σύνθεση των κανόνων, ώστε να μην δημιουργούνται συγκρούσεις και σφάλματα είναι ευθύνη του σχεδιαστή της εφαρμογής.

8.6 Προγραμματιστική Διασύνδεση

Αφού καθορίστηκε το *context* και οι κανόνες που περιγράφουν την προσαρμογή της εφαρμογής, υλοποιήθηκε η διασύνδεση της εφαρμογής με την υπηρεσία. Η διασύνδεση περιλαμβάνει τις ενέργειες που περιγράφηκαν στη παράγραφο 8.3.1. Στη συνέχεια περιγράφονται τα τρία βασικά βήματα διασύνδεσης:

- Ορισμός και υλοποίηση προμηθευτών.
- Υλοποίηση της διεπαφής *ContextManagement* από την εφαρμογή.

- Σύνδεση των *context* παραμέτρων της εφαρμογής με την υπηρεσία, για τη συλλογή των τιμών.

8.6.1 Υλοποίηση Προμηθευτών

Για τη συλλογή των τιμών, που αφορούν τις παραμέτρους της συσκευής, έπρεπε να υλοποιηθούν οι προμηθευτές που τις παράγουν. Συνολικά υλοποιήθηκαν δύο προμηθευτές: για τη μνήμη και τη μπαταρία της εφαρμογής.

Η μνήμη είναι η μόνη παράμετρος του *context*, εξωτερική της εφαρμογής, για την οποία μπορούσαμε να πάρουμε πραγματικές τιμές. Η μπαταρία της συσκευής αποτελεί ειδικό χαρακτηριστικό του υλικού των κινητών συσκευών και δυστυχώς δεν υπάρχει κάποιο *API* που να την προσομοιώνει. Παρόλα αυτά, επειδή είναι σημαντική παράμετρος για την εφαρμογή (όπως και για κάθε κινητή εφαρμογή) υλοποιήσαμε μία μέθοδο, η οποία προσομοιώνει πιθανοτικά τη διάρκεια ζωής μιας μπαταρίας.

Στη συνέχεια οι προμηθευτές ενσωματώθηκαν στο υποσύστημα *SensorsMonitor* για να γίνεται η παρακολούθησή τους. Για κάθε προμηθευτή υλοποιήθηκαν τα φίλτρα γεγονότων σύμφωνα με τις ανάγκες της εφαρμογής. Στη συνέχεια περιγράφονται οι προμηθευτές και η σύνδεσή τους με το υποσύστημα παρακολούθησης.

Προμηθευτής Μνήμης

Ο προμηθευτής της μνήμης παρέχει το ποσοστό επί τοις εκατό της διαθέσιμης μνήμης στο σωρό. Πρόκειται για μία μέθοδο που χρησιμοποιεί τις μεθόδους *freeMemory()* και *totalMemory()* της κλάσης *java.lang.Runtime*. Αυτές οι μέθοδοι επιστρέφουν την ελεύθερη και τη συνολική μνήμη στο σωρό αντίστοιχα. Η υλοποίηση του προμηθευτή φαίνεται στην Εικόνα 8.7.

Ο προμηθευτής μνήμης καλείται συνεχώς σε ένα βρόχο στο νήμα παρακολούθησης. Το νήμα ανιχνεύει γεγονότα με βάση ένα φίλτρο γεγονότων. Το φίλτρο υλοποιήθηκε να *κόβει* μία τιμή του προμηθευτή, αν η διαφορά της από την προηγούμενη είναι μικρότερη από *5*. Δηλαδή για την παράμετρο της διαθέσιμης μνήμης, γεγονός θεωρείται μία μεταβολή μεγαλύτερη του *5%*.

```
public int memoryMonitor(Runtime rt)
	{
		ret =
(((((double)rt.freeMemory())/1024)/(((double)rt.totalMemory())/1024))*100);

		return (int)ret;
	}
```

Εικόνα 8.7 Υλοποίηση προμηθευτή μνήμης.

Προμηθευτής Μπαταρίας

Ο προμηθευτής της μπαταρίας παρέχει το ποσοστό επί τοις εκατό της διαθέσιμης μπαταρίας. Ο προμηθευτής μπαταρίας που υλοποιήσαμε προσομοιώνει ικανοποιητικά τη διάρκειας ζωής μιας μπαταρίας. Πρόκειται για μία μέθοδο που παράγει τυχαίες μειώσεις του ποσοστού της διαθέσιμης μπαταρίας. Η διάρκεια της μπαταρίας ξεκινά από *100* και μειώνεται σταδιακά. Όταν φτάσει στο *0* θεωρείται ότι φορτίζεται και επανέρχεται στο *100*.

Οι μειώσεις στη διάρκεια της μπαταρίας δε γίνονται εντελώς τυχαία. Η νέα τιμή της μπαταρίας εξαρτάται από το προηγούμενο ποσοστό και από τις λειτουργίες της εφαρμογής, π.χ. μετά από μία εισερχόμενη κλήση η μπαταρία θα μειωθεί κατά ένα ποσοστό. Το ποσοστό μείωσης εξαρτάται από το τρέχον διαθέσιμο ποσοστό. Η μείωση θα είναι μεγαλύτερη αν η διαθέσιμη μπαταρία είναι ήδη πεσμένη. Το φίλτρο που χρησιμοποιήθηκε είναι το ίδιο με του προμηθευτή μνήμης. Γεγονότα θεωρούνται οι μεταβολές στην τιμή της μπαταρίας μεγαλύτερες ή ίσες με *5%*.

8.6.2 Υλοποίηση Προγραμματιστικής Διεπαφής

Το επόμενο βήμα για τη σύνδεση της εφαρμογής με την υπηρεσία είναι η υλοποίηση της διεπαφής *ContextManagement*. Η διεπαφή υλοποιήθηκε συμπληρώνοντας το σώμα των τριών μεθόδων που ορίζονται στη διεπαφή.

public String [] defineContextRules(String [] rules)

Στη μέθοδο αυτή κατασκευάζεται ένας πίνακας συμβολοσειρών με τους κανόνες. Διάφορα σύνολα κανόνων χρησιμοποιήθηκαν για τον έλεγχο και τη δημιουργία σεναρίων λειτουργίας της εφαρμογής. Στην Εικόνα 8.8 φαίνεται η υλοποίηση της μεθόδου.

```
public String [] defineContextRules(String [] rules) {

    rules  = new String [] {
       ……   // Δηλώσεις κανόνων
       ……
       ……   };
  return rules; }
```

Εικόνα 8.8 Υλοποίηση Μεθόδου Περάσματος Κανόνων.

public String [] defineSensorTrust(String [] trust)

Στη μέθοδο αυτή κατασκευάζεται ο πίνακας συμβολοσειρών με τις παραμέτρους αξιοπιστίας των προμηθευτών και τα κατώφλια των κανόνων. Επιπλέον δηλώνονται στον πίνακα οι παράμεροι με τις οποίες σχετίζονται οι προμηθευτές, ώστε να συσχετιστούν με τις αντίστοιχες πιθανότητες αξιοπιστίας των προμηθευτών στη φάση υπολογισμού της αξιοπιστίας των κανόνων.

public void ActionResolve(String property, String value)

Στη μέθοδο αυτή γίνεται η αντιστοίχιση των ενεργειών σε μεθόδους της εφαρμογής. Για μία ενέργεια, *<property> = <value>*, εκτελείται η μέθοδος που σχετίζεται με την παράμετρο *<property>* του *context* και πιθανώς με όρισμα την τιμή *<value>*, αν η μέθοδος παίρνει κάποιο ακέραιο όρισμα. Για παράδειγμα η κλήση της μεθόδου με *propery = "BatteryMode"* και *value = "2"* θα θέσει το μοτίβο λειτουργίας της μπαταρίας σε *Silent*. Στην Εικόνα 8.9 φαίνεται η υλοποίηση της μεθόδου, για το σύνολο των κανόνων του Πίνακα 8.2.

```
public void ActionResolve(String property, String s_value)
    {
        int value = 0;

        if(IsInteger(s_value))
            value = Integer.parseInt(s_value);

      if("BatteryMode".equals(property))
      setBatMode(value);

      else if("Security".equals(property))
      setSec(value);

      else if("Privacy".equals(property))
      setPr(value);

      else if("browseURL".equals(property))
      browseURL();

      else if("smsHandler".equals(property))
      smsHandler();

      else if("callHandler".equals(property))
      callHandler();

      else if("http".equals(property))
      httpConnect();

      else if("freeMem".equals(property))
      freeMem();

      else if("trustCheckEnable".equals(property))
      trustCheckEnable();        //Μέσα σε αυτή τη μέθοδο θα κληθεί μία
                          μέθοδος της υπηρεσίας
       else
       display.setCurrent(Error_Message);
       }
```

Εικόνα 8.9 Υλοποίηση της Μεθόδου ActionResolve.

8.6.3 Συλλογή Τιμών από τις Παραμέτρους της Εφαρμογής

Οι τιμές για τις παραμέτρους της εφαρμογής συλλέγονται μέσα από την εφαρμογή. Η συλλογή αυτών των τιμών γίνεται με απ' ευθείας κλήση του *EventListener* στο κώδικα της εφαρμογής. Για τις ιδιότητες *incSMS* και *incCall* χρησιμοποιήθηκε ένα ξεχωριστό νήμα παρακολούθησης μέσα στην εφαρμογή. Το *WMA* επιβάλει η παρακολούθηση των εισερχόμενων κλήσεων και μηνυμάτων να γίνεται σε διαφορετικό νήμα από αυτό της εφαρμογής, ώστε να μην μπλοκάρει τη λειτουργία της.

Το νήμα αυτό μπορεί επίσης να χρησιμοποιηθεί και για την παρακολούθηση και άλλων παραμέτρων. Αυτή η προσέγγιση απαιτεί κόπο από τον προγραμματιστή ώστε να εξασφαλίσει ότι η παρακολούθηση θα είναι αδιάκοπη και δεν υπάρχει περίπτωση να χαθούν κρίσιμα γεγονότα εξαιτίας καθυστέρησης στο χειρισμό των *SMSs*.

Η μέθοδος που χρησιμοποιήθηκε είναι, η απευθείας σύνδεση όλων των παραμέτρων της εφαρμογής με τον *EventListener*. Αυτή απαιτεί να γνωρίζει ο προγραμματιστής, που μπορούν να μεταβληθούν οι τιμές στο κώδικα και να καλέσει εκεί τον *EventListener*. Για παράδειγμα στα σημεία όπου διαβάζονται οι επιλογές του χρήστη από το μενού της οθόνης του μπορεί να γίνει άμεσα η ενημέρωση της υπηρεσίας. Αυτή η προσέγγιση εξασφαλίζει ότι η υπηρεσία θα ειδοποιηθεί σίγουρα για τις αλλαγές των τιμών και χωρίς καθυστέρηση.

ΚΕΦΑΛΑΙΟ 9. ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ

Μία βασική απαίτηση για το λογισμικό που αναπτύξαμε είναι να κάνει λογική και οικονομική χρήση των υπολογιστικών πόρων που παρέχουν οι κινητές συσκευές. Για να κάνουμε αποτίμηση της απόδοσης της υπηρεσίας, εκτελέσαμε μία σειρά πειραμάτων. Στα πειράματα μετρήθηκε η κατανάλωση των κρίσιμων υπολογιστικών πόρων καθώς και η χρονική απόκριση των βασικών λειτουργιών. Στο κεφάλαιο αυτό περιγράφουμε τα πειράματα που εκτελέστηκαν και παρουσιάζουμε τα αποτελέσματα που προέκυψαν από τις μετρήσεις.

9.1 Προσομοίωση Κινητής Εφαρμογής

Ο σκοπός των πειραμάτων ήταν να μετρήσουμε τους πόρους που καταναλώνει η υπηρεσία κατά τη λειτουργία της. Τα πειράματα εκτελέστηκαν πάνω στον προσομοιωτή του *WTK* ώστε να είναι αντιπροσωπευτικά της λειτουργίας μιας πραγματικής κινητής συσκευής. Οι παράμετροι που μετρήθηκαν στα πειράματα είναι οι εξής:

- Η συνολική μνήμη που απαιτεί η υπηρεσία.
- Η χρονική διάρκεια των βασικών λειτουργιών.
- Η χρήση της *CPU*.

Για κάθε μία από τις παραπάνω παραμέτρους εκτελέστηκε ένα ξεχωριστό σύνολο πειραμάτων. Οι τιμές μετρήθηκαν κατά τη διάρκεια των εκτελέσεων, μεταβάλλοντας ένα

σύνολο μεταβλητών που τις επηρεάζουν. Οι παράμετροι αυτές επιλέχθηκαν διότι αποτελούν τους πιο κρίσιμους υπολογιστικούς πόρους για μία κινητή συσκευή και επιπλέον είχαμε τα εργαλεία να τις μετρήσουμε. Η κατανάλωση της μπαταρίας, θα είχε ενδιαφέρον να μετρηθεί, αλλά δυστυχώς δεν υπάρχει στην πλατφόρμα *J2ME* κάποιο εργαλείο που να προσομοιώνει την μπαταρία μιας κινητής συσκευής.

9.1.1 Περιβάλλον Προσομοίωσης

Για την εκτέλεση των πειραμάτων υλοποιήσαμε ένα απλό *MIDlet* και το συνδέσαμε με την υπηρεσία. Για την προσομοίωση της λειτουργίας της εφαρμογής προσθέσαμε στο *MIDlet* ένα σύστημα παραγωγής δεδομένων. Το σύστημα αποτελείται από τρεις γεννήτορες: ένα γεννήτορα κανόνων, ένα γεννήτορα παραμέτρων ποιότητας και ένα γεννήτορα γεγονότων. Οι γεννήτορες παρέχουν τις εισόδους που θα έπαιρνε η υπηρεσία κατά τη λειτουργία της σε ένα *CAS*.

Γεννήτορας Κανόνων

Ο γεννήτορας κανόνων παράγει τυχαίους κανόνες. Οι κανόνες ακολουθούν το συντακτικό του *context* μοντέλου που φαίνεται στην Εικόνα 6.4 και συντίθενται επιλέγοντας τυχαία τα συστατικά τους. Τα συνθετικά των κανόνων επιλέγονται από το λεξικό που φαίνεται στους Πίνακες 6.1 και 6.2. Συγκεκριμένα οι κανόνες παράγονται ως εξής:

- Ο γεννήτορας παίρνει σαν είσοδο ένα σύνολο δομικών μονάδων: π.χ. $node_1 \ldots node_5$, $channel_1 \ldots channel_{10}$, $object_1 \ldots object_{20}$.
- Οι συνθήκες είναι της μορφής:
  <condition> :: = <architectural_element>.<context_property>
  <comparison_operator> <value>.
- Οι ενέργειες είναι της μορφής:
  <action> ::= <architectural_element>.<context_property> = <value>.
- Οι συνθήκες και οι ενέργειες μπορεί να συνοδεύονται από χρονικό τελεστή με μία πιθανότητα *p*.
- Ο αριθμός συνθηκών και ενεργειών κάθε κανόνα, μπορεί να είναι τυχαίος (π.χ. *1* ως *5* συνθήκες και *1* ως *3* ενέργειες) ή σταθερός.

- Μία σύνθετη συνθήκη δημιουργείται με πιθανότητα *1/3* και επιλέγεται τυχαία ο αριθμός των συνθετικών συνθηκών που την αποτελούν.
- Για κάθε συνθήκη ή ενέργεια επιλέγεται τυχαία, ένα δομικό στοιχείο, μία παράμετρος *context* που περιλαμβάνεται στις παραμέτρους του δομικού στοιχείου, ένας τελεστής σύγκρισης (για ενέργεια είναι πάντα το =) και μία τιμή από το σύνολο τιμών της παραμέτρου.

Κατά την παρασκευή των κανόνων κρατάμε σταθερές κάποιες παραμέτρους και μεταβάλουμε κάποιες που αναμένουμε να επηρεάζουν την απόδοση της υπηρεσίας. Οι παράμετροι που μεταβάλλονται ανάλογα με το πείραμα είναι οι εξής:

- Αριθμός κανόνων.
- Πολυπλοκότητα κανόνων. Δηλαδή:
  - Αριθμός συνθηκών.
  - Αριθμός ενεργειών.
  - Πιθανότητα εμφάνισης χρονικού τελεστή.

Ο γεννήτορας κανόνων εκτελείται πριν από την έναρξη της υπηρεσίας. Οι κανόνες εισάγονται στο καθορισμένο αρχείο και ο *ContextManager* τους διαβάζει από εκεί.

Γεννήτορας Παραμέτρων Ποιότητας

Ο γεννήτορας παράγει τυχαίες τιμές στο πεδίο *[0,1)* για τους έξι προμηθευτές του Πίνακα 6.1. Τα κατώφλια των κανόνων έχουν σταθερή τιμή ίση με *1*, ώστε να εκτελούνται πάντα οι ενέργειες των κανόνων. Όπως θα δούμε στη συνέχεια, η επιβάρυνση του ελέγχου αξιοπιστίας, υπάρχει μόνο στη φάση υπολογισμού της αξιοπιστίας των κανόνων και είναι μηδενική κατά τη φάση του ελέγχου ικανοποίησης.

Γεννήτορας Γεγονότων

Ο γεννήτορας γεγονότων παράγει τυχαία γεγονότα για την υπηρεσία. Ο γεννήτορας τοποθετείται μέσα στο νήμα του *SensorsMonitor* και προσομοιώνει το ρόλο των προμηθευτών. Εκτελεί ένα βρόχο στον οποίο παράγει ένα καθορισμένο αριθμό γεγονότων. Τα γεγονότα παράγονται τυχαία σύμφωνα με το λεξικό του Πίνακα 6.1. Δηλαδή ένα γεγονός είναι της μορφής: <*architectural_element*>.<*context_property*> = <*value*>.

9.1.2 Ρυθμίσεις Πειραμάτων

Η λειτουργία της υπηρεσίας μπορεί να χωριστεί σε δύο φάσεις: τη φάση της συντακτικής ανάλυσης και τη φάση της επεξεργασίας των γεγονότων. Οι μετρήσεις των παραμέτρων έγιναν ξεχωριστά για τις δύο φάσεις, αφού αναμένουμε η κατανάλωση των πόρων να είναι διαφορετική για κάθε φάση.

9.2 Μνήμη Κινητής Εφαρμογής

Η μνήμη είναι ένας από τους βασικούς υπολογιστικούς πόρους για μία εφαρμογή, ενώ ταυτόχρονα η διαθέσιμη ποσότητά της στις κινητές συσκευές είναι λιγοστή. Στα πειράματα που εκτελέστηκαν μετρήθηκε η μνήμη που απαιτεί η υπηρεσία για να εκτελεστεί σε μία κινητή συσκευή. Πριν από την περιγραφή των πειραμάτων κρίνεται σκόπιμο να περιγραφεί η δομή της μνήμης που παρέχουν οι κινητές συσκευές και πως αυτή χρησιμοποιείται από μία κινητή εφαρμογή.

9.2.1 Μνήμη του MIDlet

Η μνήμη των κινητών εφαρμογών που ακολουθούν το πρότυπο του *MIDlet* αποτελείται από τρία συστατικά: *τη μνήμη του προγράμματος*, *το σωρό* και *το μόνιμο αποθηκευτικό χώρο*.

Μνήμη Προγράμματος

Η μνήμη προγράμματος είναι η μνήμη της συσκευής όπου θα εγκατασταθούν τα εκτελέσιμα αρχεία του *MIDlet*. Τα αρχεία συμπιέζονται σε ένα αρχείο *JAR* και περιλαμβάνουν ένα πολύ μικρότερο αρχείο *JAD* που περιγράφει τα περιεχόμενα του *JAR*. Το αρχείο *JAR* αποθηκεύεται στη μνήμη προγράμματος. Το μέγεθος της μνήμης προγράμματος ποικίλει ανάλογα τη συσκευή και εξαρτάται από τον κατασκευαστή, πόση μνήμη διαθέτει για την αποθήκευση του *MIDlet*.

Το μέγεθος του αρχείου *JAR* είναι πάντα μικρότερο από τη μνήμη που τελικά θα χρησιμοποιηθεί στη μνήμη προγράμματος. Κατά την εγκατάσταση του *MIDlet* το λογισμικό

της συσκευής μετατρέπει τα περιεχόμενα του *JAR* σε κάποια εσωτερική αναπαράσταση, μειώνοντας σημαντικά τη μνήμη που καταλαμβάνει. Η ακριβής μνήμη προγράμματος που καταλαμβάνει τελικά ένα *JAR*, εξαρτάται και από τη συγκεκριμένη συσκευή που εγκαθίσταται.

Σωρός

Ο σωρός είναι το τμήμα της μνήμης που χρησιμοποιεί το *MIDlet* κατά την εκτέλεσή του. Όλες οι μεταβλητές και τα αντικείμενα που δημιουργεί και χρησιμοποιεί η εφαρμογή αποθηκεύονται στο σωρό. Το μέγεθος του σωρού επίσης καθορίζεται από τον κατασκευαστή της συσκευής.

Μόνιμος Αποθηκευτικός Χώρος

Ο τρίτος τύπος μνήμης που χρησιμοποιούν τα *MIDlets* είναι ο μόνιμος αποθηκευτικός χώρος. Εκεί αποθηκεύονται τα σύνολα εγγραφών, που αποτελούν τις βάσεις δεδομένων των κινητών *MIDP* συσκευών. Και αυτό το τμήμα μνήμης καθορίζεται από τον κατασκευαστή της συσκευής.

9.2.2 Προδιαγραφές Συσκευών

Η τελευταία προδιαγραφή για την ανάπτυξη κινητών εφαρμογών *Java Technology for the Wireless Industry (JTWI) (JSR 185)*, που χρησιμοποιεί το *MIDP 2.0*, καθορίζει τις ελάχιστες τιμές για κάθε τύπο μνήμης που θα πρέπει να έχουν οι συσκευές που υποστηρίζουν το *MIDlets* [L11]. Οι τιμές αυτές είναι:

- *64 KB* για αποθήκευση του *JAR* και 5 *KB* για το *JAD*.
- *30 KB* για μόνιμο αποθηκευτικό χώρο
- *256 KB* για το σωρό.

Οι παραπάνω τιμές είναι οι ελάχιστες και όλα τα σύγχρονα κινητά παρέχουν πολύ περισσότερη μνήμη, της τάξεως των δεκάδων *MB*. Σε κάποιες συσκευές διαχωρίζεται η μνήμη της συσκευής και επιβάλλεται μία μέγιστη τιμή, για κάθε τύπο μνήμης ξεχωριστά. Για παράδειγμα κάποιες αρχικές υλοποιήσεις του *CLDC* δεν επέτρεπαν την εγκατάσταση *JAR*

πάνω από 50 *KB*. Άλλες συσκευές παρέχουν στα *MIDlets* μία μέγιστη ποσότητα μνήμης και δεν επιβάλλουν κανένα περιορισμό στην κατανομή της μεταξύ των τριών τύπων μνήμης.

9.2.3 Ρυθμίσεις Προσομοίωσης

Το *WTK* παρέχει επιλογές για να τον καθορισμό της μνήμης του σωρού και του μόνιμου αποθηκευτικού χώρου. Για την εκτέλεση των πειραμάτων θέσαμε ένα *1 MB* μνήμη στο σωρό και ένα *1 MB* μόνιμο αποθηκευτικό χώρο. Όπως θα δούμε στη συνέχεια οι τιμές αυτές αποδείχτηκαν υπεραρκετές στις εκτελέσεις των πειραμάτων.

9.3 Κατανάλωση Μνήμης

Σε αυτό το σύνολο πειραμάτων μετρήθηκε η μνήμη που απαιτεί η υπηρεσία για τις τρεις κατηγορίες μνήμης που αναφέρθηκαν παραπάνω. Η μνήμη του σωρού και του μόνιμου αποθηκευτικού χώρου μετρήθηκαν σε συνάρτηση με τις παραμέτρους που την επηρεάζουν. Επιπλέον παρουσιάζεται το μέγεθος των εκτελέσιμων αρχείων που εγκαθίστανται στη συσκευή.

9.3.1 Μνήμη Σωρού

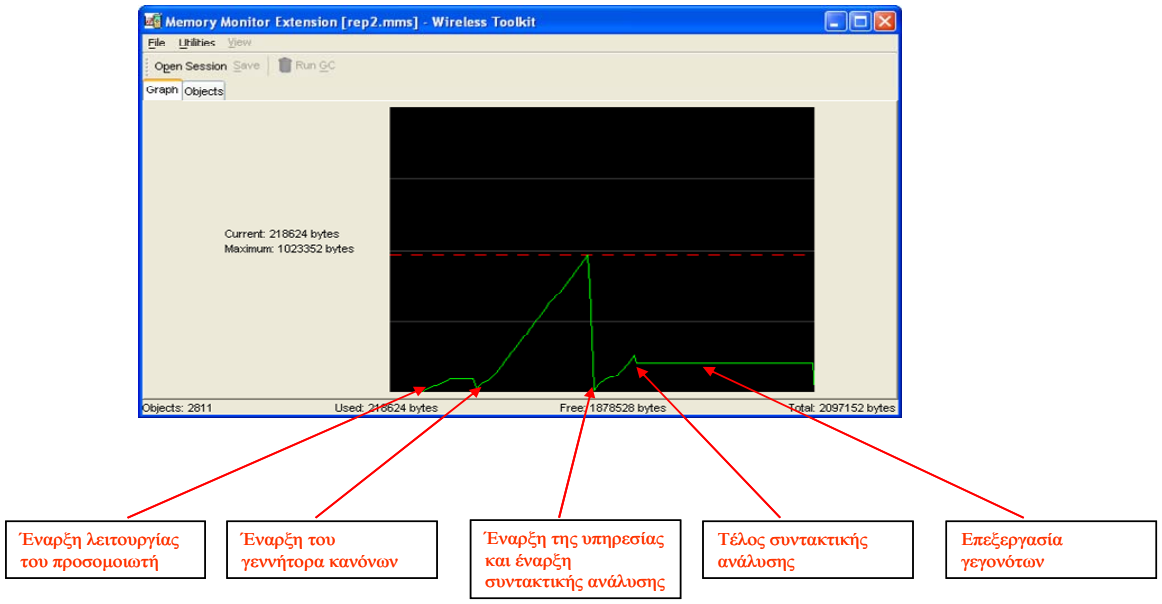
Στο πρώτο σύνολο πειραμάτων μετρήθηκε η μνήμη που δεσμεύει η υπηρεσία κατά τη λειτουργία της από το σωρό. Η μνήμη μετρήθηκε με τις μεθόδους *totalMemory()* και *freeMemory()* της κλάσης *java.lang.Runtime*. Οι μέθοδοι επιστρέφουν τη συνολική και την ελεύθερη μνήμη αντίστοιχα, στη *JVM*.

Οι παράμετροι που μεταβάλαμε σε αυτό το σύνολο πειραμάτων είναι:

- Ο αριθμός των κανόνων.
- Ο αριθμός των συνθηκών σε κάθε κανόνα.
- Ο αριθμός των ενεργειών σε κάθε κανόνα.
- Η πιθανότητα εμφάνισης χρονικού τελεστή σε συνθήκες και ενέργειες.

Αυτό που φυσιολογικά αναμένουμε είναι η μνήμη να αυξάνεται περίπου γραμμικά καθώς αυξάνονται οι παραπάνω παράμετροι. Η αύξηση της δεσμευμένης μνήμης αναμένεται να υπάρχει, μόνο κατά τη διάρκεια της συντακτικής ανάλυσης, όπου δημιουργείται το δέντρο των κανόνων. Στη συνέχεια κατά τη επεξεργασία των γεγονότων, οι καταχωρήσεις στη μνήμη είναι ελάχιστες και αμελητέες και αναμένουμε η μνήμη να παραμένει σχεδόν σταθερή. Στην Εικόνα 9.1 φαίνεται η μεταβολή της μνήμης κατά τη διάρκεια της προσομοίωσης. Το διάγραμμα παράγεται από το *WTK* και περιγράφει τη μεταβολή της μνήμης στη *JVM*.

Όπως φαίνεται στην Εικόνα 9.1, κατά τη διάρκεια της συντακτικής ανάλυσης η μνήμη που δεσμεύει η υπηρεσία αυξάνεται σταδιακά παίρνοντας τη μέγιστη τιμή της όταν έχει ολοκληρωθεί η κατασκευή του δέντρου των κανόνων. Μία μικρή πτώση που φαίνεται στο τέλος της συντακτικής ανάλυσης οφείλεται σε «συλλογή σκουπιδιών» (*garbage collection*). Στη συνέχεια βλέπουμε ότι η μνήμη παραμένει σταθερή κατά τη φάση της συλλογής και επεξεργασίας των γεγονότων μέχρι το τέλος της λειτουργίας προσομοίωσης. Το συγκεκριμένο πείραμα εκτελέστηκε για *60* κανόνες, *5* συνθήκες και *3* ενέργειες ανά κανόνα, *½* πιθανότητα εμφάνισης χρονικού τελεστή και συνολική διαθέσιμη μνήμη *2 MB*.



Εικόνα 9.1 Διάγραμμα Μεταβολής της Μνήμης Κατά την Εκτέλεση της Υπηρεσίας.

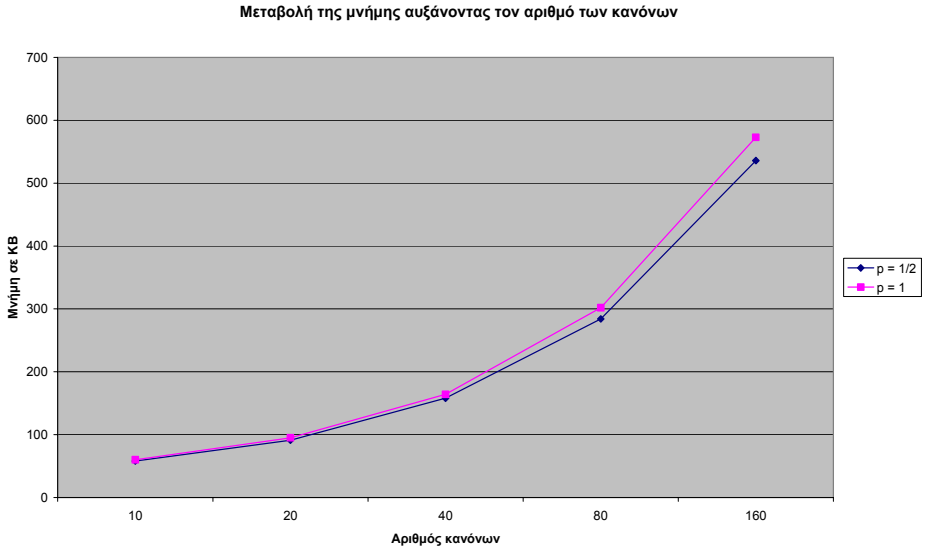
Στη συνέχεια περιγράφονται τα πειράματα που αφορούν τη μνήμη. Σε κάθε πείραμα μετρήθηκε η μέγιστη τιμή της δεσμευμένης μνήμης. Η δεσμευμένη μνήμη παίρνει τη μέγιστη τιμή της στο τέλος της συντακτικής ανάλυσης και πριν εκτελεστεί συλλογή σκουπιδιών.

Μεταβολή Κανόνων

Σε αυτό το σύνολο πειραμάτων μετρήθηκε η μέγιστη μνήμη που δεσμεύει η υπηρεσία αυξάνοντας τον αριθμό των κανόνων. Τα πειράματα εκτελέστηκαν δύο φορές για πιθανότητα εμφάνισης χρονικού τελεστή *½* και *1*. Κάθε κανόνας είχε σταθερό αριθμό συνθηκών και ενεργειών *5* και *3* αντίστοιχα. Στον Πίνακα 9.1 φαίνονται οι τιμές που προέκυψαν και στο Σχήμα 9.1 φαίνεται το διάγραμμα της μεταβολής της μνήμης σε συνάρτηση με τον αριθμό των κανόνων.

Πίνακας 9.1 Μεταβολή Μνήμης Συναρτήσει του Αριθμού των Κανόνων και των Χρονικών Τελεστών.

Αριθμός κανόνων	10	20	40	80	160
Μνήμη σε *KB* $p_{temp} = ½$	58	91	158	284	536
Μνήμη σε *KB* $p_{temp} = 1$	60	95	164	302	573



Σχήμα 9.1 Γραφική Αναπαράσταση της Μνήμης Συναρτήσει του Αριθμού των Κανόνων και των Χρονικών Τελεστών.

Στο Σχήμα 9.1 παρατηρούμε ότι η αύξηση της μνήμης είναι γραμμική όσο αυξάνονται οι κανόνες. Επίσης παρατηρούμε ότι η ύπαρξη χρονικών τελεστών επιβαρύνει ελάχιστα τη μνήμη του συστήματος, καθώς για λογικό αριθμό κανόνων οι διαφορές είναι ασήμαντες. Αυτή η αύξηση μνήμης που προκαλεί η αύξηση των χρονικών τελεστών, οφείλεται στις επιπλέον τιμές που αποθηκεύονται στους κόμβους του δέντρου.

Μεταβολή Αριθμού Συνθηκών

Σε αυτό το πείραμα μετρήθηκε η μεταβολή της μνήμης αυξάνοντας τον αριθμό των συνθηκών σε κάθε κανόνα. Τα πειράματα εκτελέστηκαν δύο φορές για πιθανότητες εμφάνισης χρονικού τελεστή *½* και *1*. Ο αριθμός των κανόνων ήταν *10* και ο αριθμός των ενεργειών σε κάθε κανόνα ήταν *3*. Στον Πίνακα 9.2 φαίνονται οι τιμές που προέκυψαν.

Πίνακας 9.2 Μεταβολή της Μνήμης Συναρτήσει του Αριθμού των Συνθηκών και των Χρονικών Τελεστών.

Αριθμός συνθηκών ανά κανόνα	2	4	8	16	32
Μνήμη σε *KB* $p_{temp} = 1/2$	45	56	73	109	180
Μνήμη σε *KB* $p_{temp} = 1$	46	58	76	113	187

Μεταβολή Αριθμού Ενεργειών

Σε αυτό το πείραμα μετρήθηκε η μεταβολή της μνήμης μεταβάλλοντας τον αριθμό των ενεργειών σε κάθε κανόνα. Τα πειράματα εκτελέστηκαν δύο φορές για πιθανότητες εμφάνισης χρονικού τελεστή *½* και *1*. Ο αριθμός των κανόνων είναι *10* και ο αριθμός των συνθηκών σε κάθε κανόνα ήταν *3*. Στον Πίνακα 9.3 φαίνονται οι τιμές που προέκυψαν.

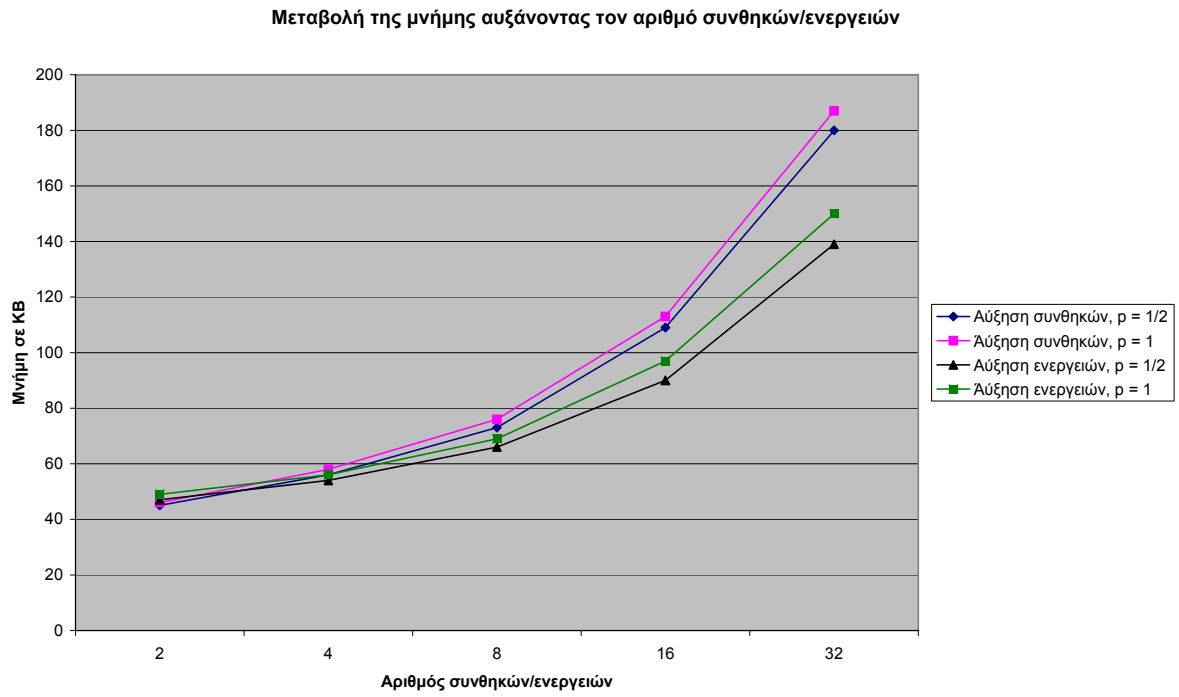
Στο Σχήμα 9.2 φαίνεται το διάγραμμα της μεταβολής της μνήμης σε συνάρτηση με τον αριθμό των συνθηκών και των ενεργειών. Παρατηρούμε ότι η μνήμη αυξάνεται περισσότερο όσο αυξάνονται οι συνθήκες ενός κανόνα. Αυτό συμβαίνει διότι στους κόμβους των συνθηκών κρατάμε περισσότερη πληροφορία από αυτούς των ενεργειών. Τα βασικά επιπλέον πεδία είναι ο λογικός τελεστής (στις ενέργειες είναι πάντα ο &&), ο τελεστής

σύγκρισης (στις ενέργειες είναι πάντα =) και η συμβολοσειρά των υποσυνθηκών που υπάρχει στις σύνθετες εκφράσεις.

Επιπλέον παρατηρούμε ότι και στις δύο περιπτώσεις υπάρχει μικρή αύξηση της μνήμης όταν όλες οι συνθήκες/ενέργειες έχουν χρονικό τελεστή. Η επιπλέον μνήμη οφείλεται στο επιπλέον πεδίο του χρονικού τελεστή στους κόμβους του δέντρου.

Πίνακας 9.3 Μεταβολή της Μνήμης Συναρτήσει του Αριθμού των Ενεργειών και των Χρονικών Τελεστών.

Αριθμός ενεργειών ανά κανόνα	2	4	8	16	32
Μνήμη σε *KB* $p_{temp} = 1/2$	47	54	66	90	139
Μνήμη σε *KB* $p_{temp} = 1$	49	56	69	97	150



Σχήμα 9.2 Γραφική Αναπαράσταση της Μνήμης Συναρτήσει της Πολυπλοκότητας των Κανόνων.

9.3.2 Μνήμη Μόνιμου Αποθηκευτικού Χώρου

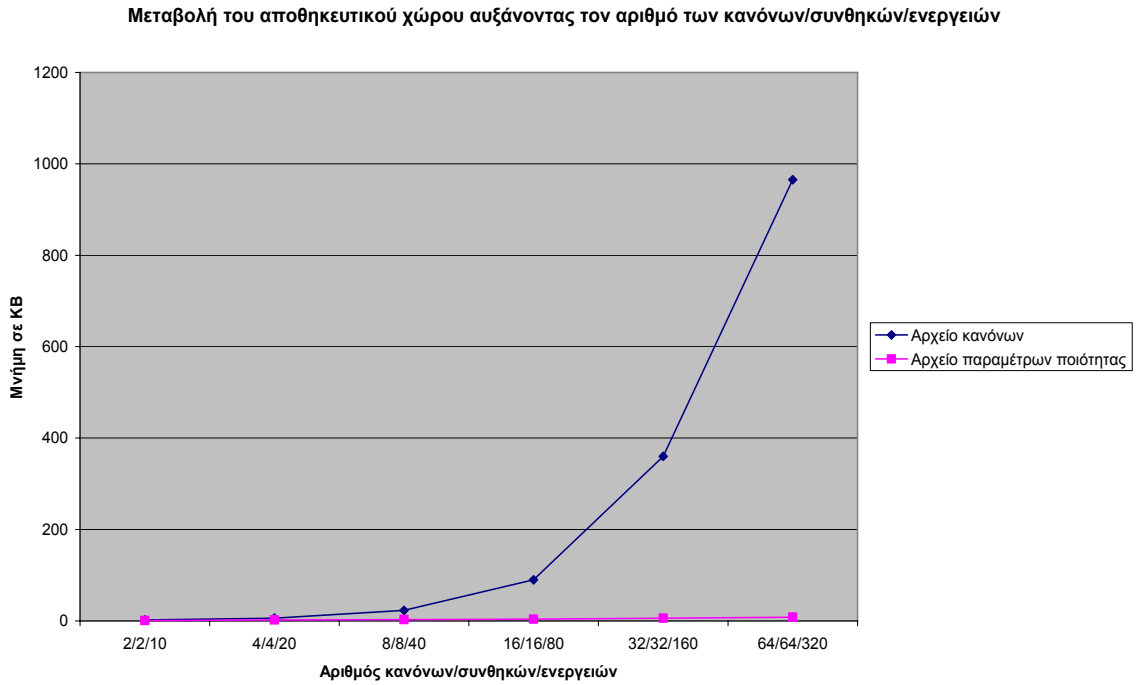
Στο τελευταίο πείραμα μετρήθηκε η μνήμη που καταλαμβάνει η υπηρεσία για μόνιμο αποθηκευτικό χώρο. Η μνήμη αυτή καταλαμβάνεται από τα αρχεία των κανόνων και των παραμέτρων ποιότητας, που παίρνει η υπηρεσία σαν είσοδο. Τα αρχεία αυτά μπορεί να σβηστούν ή να μεταβληθεί το μέγεθός τους κατά τη διάρκεια λειτουργίας της υπηρεσίας.

Στα πειράματα που εκτελέσαμε μετρήσαμε το μέγεθος των αρχείων αμέσως μόλις δημιουργούνται. Οι παράμετροι που μεταβάλαμε ήταν τα περιεχόμενα των αρχείων δηλαδή ο αριθμός των κανόνων, των συνθηκών και των ενεργειών. Μεταβάλαμε ταυτόχρονα αυτές τις παραμέτρους, διότι οι μεμονωμένες μεταβολές τους δεν προκαλούν ιδιαίτερη αύξηση στο μέγεθος των αρχείων. Η πιθανότητα εμφάνισης χρονικού τελεστή ήταν ίση με *1*, δηλαδή όλες οι συνθήκες και οι ενέργειες συνοδεύονταν από χρονικό τελεστή. Στον Πίνακα 9.4 φαίνονται οι τιμές που προέκυψαν από τις μετρήσεις των αρχείων.

Πίνακας 9.4 Μνήμη που Καταλαμβάνει το Αρχείο των Κανόνων και των Παραμέτρων Ποιότητας.

Αριθμός Συνθηκών/Ενεργειών/Κανόνων	2/2/10	4/4/20	8/8/40	16/16/80	32/32/160	64/64/320
Μέγεθος αρχείου κανόνων σε *KB*	2	6	23	90	360	965
Μέγεθος αρχείου παραμέτρων ποιότητας σε *KB*	1	2	3	4	6	8

Όπως φαίνεται στο Σχήμα 9.3 το μέγεθος του αρχείου των παραμέτρων αξιοπιστίας είναι αμελητέο όσο αυξάνεται ο αριθμός των κανόνων και των συνθετικών τους. Επιπλέον παρατηρούμε ότι η μνήμη που καταλαμβάνει το αρχείο των κανόνων είναι μικρή, όσο ο αριθμός των κανόνων και των συστατικών τους παραμένουν σε λογικά επίπεδα. Πιστεύουμε ότι ένα σύνολο κανόνων μεγαλύτερο του *20* δεν είναι πρακτικά χρήσιμο για μία κινητή εφαρμογή.



Σχήμα 9.3 Αύξηση της Μνήμης των Αρχείων της Υπηρεσίας, Αυξάνοντας τους Κανόνες και τα Συνθετικά τους.

9.3.3 Μνήμη Προγράμματος

Η μεταγλώττιση των αρχείων του *MIDlet* δημιουργεί το εκτελέσιμο αρχείο του *MIDlet* το *JAR* και την περιγραφή του, το *JAD*. Τα αρχεία της εφαρμογής συμπιέζονται στο *JAR* μειώνοντας το μέγεθος των εκτελέσιμων κλάσεων. Το αρχείο *JAR* είναι αυτό που θα εγκατασταθεί στην κινητή συσκευή καταλαμβάνοντας χώρο από τη μνήμη προγράμματος.

Το μέγεθος του *JAR* αρχείου μπορεί να ελαττωθεί περαιτέρω με χρήση ειδικών εργαλείων συμπίεσης. Τα εργαλεία αυτά ονομάζονται *obfuscators* και ελαχιστοποιούν το μέγεθος του εκτελέσιμου κώδικα. Οι *obfuscators* μετατρέπουν τις κλάσεις του *MIDlet*, απομακρύνοντας αχρησιμοποίητες μεθόδους και κλάσεις και μετονομάζοντας μεθόδους και μεταβλητές. Τα εργαλεία αυτά διανέμονται ελεύθερα προς χρήση, σε διάφορες εκδόσεις και εγκαθίστανται στον *J2ME* μεταγλωττιστή. Το *JAR* που παράγουν μπορεί να εκτελεστεί από την υλοποίηση του *CLDC* της συσκευής χωρίς διαφοροποιήσεις από το κανονικό.

Το μέγεθος των εκτελέσιμων κλάσεων της υπηρεσίας, που προκύπτουν από τη μεταγλώττιση είναι *57 KB*. Το αρχείο *JAR* που δημιουργείται κατά τη μεταγλώττιση, χωρίς χρήση *obfuscator* έχει μέγεθος *25 KB*, ενώ το *JAD 279 Bytes*. Με χρήση *obfuscator* το *JAR* που προκύπτει έχει μέγεθος *938 Bytes*, ενώ το *JAD 277 Bytes*! Ο λόγος που επιτυγχάνεται τόσο μεγάλη συμπίεση είναι επειδή στα αρχεία της υπηρεσίας περιλαμβάνεται μόνο κώδικας και όχι αρχεία ήχου ή εικόνες, όπως συμβαίνει στα περισσότερα *MIDlets*. Συνεπώς η μνήμη προγράμματος που καταλαμβάνει η υπηρεσία είναι ελάχιστη με τη χρήση *obfuscator* και αρκετά μικρή, ακόμα και χωρίς επιπλέον συμπίεση.

9.4 Χρονική Απόκριση

Σε αυτό το σύνολο πειραμάτων μετρήθηκε η χρονική απόκριση της υπηρεσίας στις δύο φάσεις της λειτουργίας της. Στη πρώτη φάση μετρήθηκε ο χρόνος της συντακτικής ανάλυσης των κανόνων. Στη δεύτερη φάση μετρήθηκε ο μέσος χρόνος αποτίμησης ανά γεγονός. Δηλαδή πόσο χρόνο, κατά μέσο όρο, διαρκεί η επεξεργασία ενός γεγονότος από τον *RulesEvaluator*.

Για τη μέτρηση του χρόνου χρησιμοποιήθηκε η μέθοδος *System.currentTimeMillis(),* της *Java*. Οι παράμετροι που μεταβάλαμε ήταν ο αριθμός των κανόνων, οι συνθήκες και οι ενέργειες του κανόνα και η πιθανότητα εμφάνισης χρονικού τελεστή. Φυσιολογικά αναμένουμε ότι η χρονική απόκριση θα αυξάνεται όσο μεγαλώνουν οι παραπάνω παράμετροι, σε άλλες περιπτώσεις περισσότερο και σε άλλες λιγότερο.

Ο χρόνος αποτίμησης ενός γεγονότος εξαρτάται από το αν υπάρχουν κανόνες που επηρεάζονται από το γεγονός και πόσοι. Αν το γεγονός δεν βρεθεί σε κανένα κανόνα δεν θα γίνει ενημέρωση πεδίων σε κανένα κόμβο. Ενώ κάθε κανόνας που περιέχει την *context* παράμετρο του γεγονότος, διασχίζεται για να βρεθεί η συνθήκη ή οι συνθήκες και να γίνουν οι ενημερώσεις. Επιπλέον ρόλο παίζει και ο αριθμός των κανόνων που ικανοποιούνται κάθε χρονική στιγμή, καθώς πρέπει να διασχιστεί η λίστα των ενεργειών του κάθε κανόνα.

Για να εξαλείψουμε την εξάρτηση από το σύνολο των κανόνων εκτελέσαμε τα πειράματα *10* φορές για *10* διαφορετικά σύνολα κανόνων. Σε κάθε εκτέλεση μετρήσαμε το χρόνο της συντακτικής ανάλυσης και το συνολικό χρόνο επεξεργασίας για *1000* γεγονότα και

πήραμε το μέσο χρόνο επεξεργασίας ανά γεγονός. Στη συνέχεια πήραμε το μέσο όρο των τιμών των δέκα εκτελέσεων.

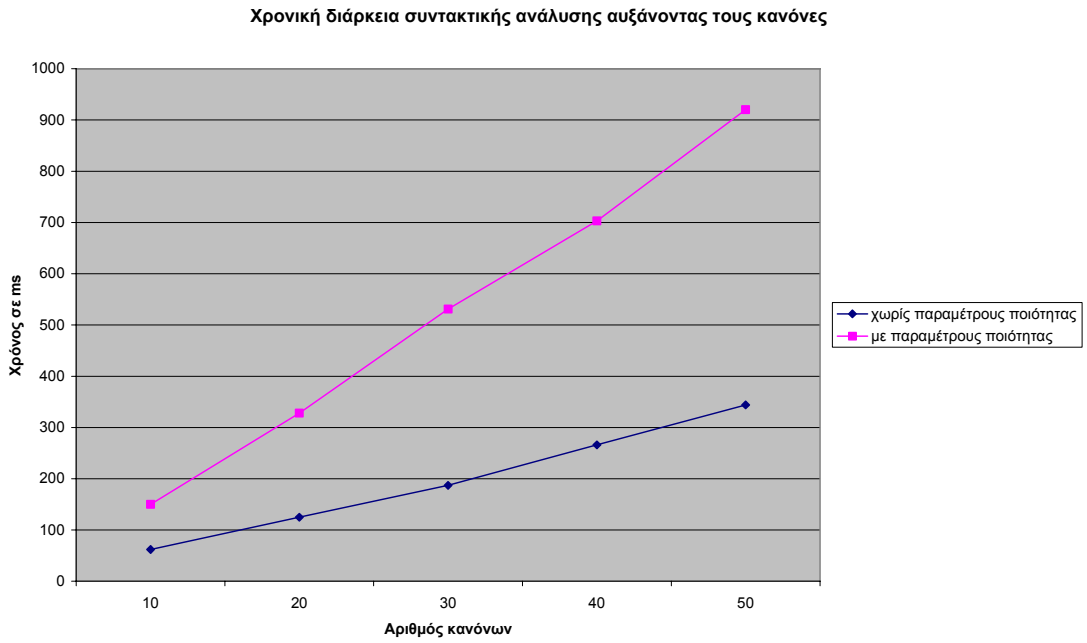
Ο αριθμός των συνθηκών σε κάθε κανόνα είναι *5* και των ενεργειών *3*. Η πιθανότητα εμφάνισης χρονικού τελεστή είναι ½. Τα πειράματα εκτελέστηκαν δύο φορές, μία με παραμέτρους ποιότητας και μία χωρίς. Σκοπός ήταν να δούμε πόσο επιβαρύνεται χρονικά η υπηρεσία από τους ελέγχους αξιοπιστίας.

Στο Σχήμα 9.4, φαίνεται η αύξηση του χρόνου της συντακτικής ανάλυσης, αυξάνοντας τον αριθμό των κανόνων. Χωρίς παραμέτρους αξιοπιστίας ο χρόνος αυξάνεται γραμμικά, όσο αυξάνονται οι κανόνες. Με παραμέτρους ποιότητας ο χρόνος της συντακτικής ανάλυσης είναι πολύ μεγαλύτερος.

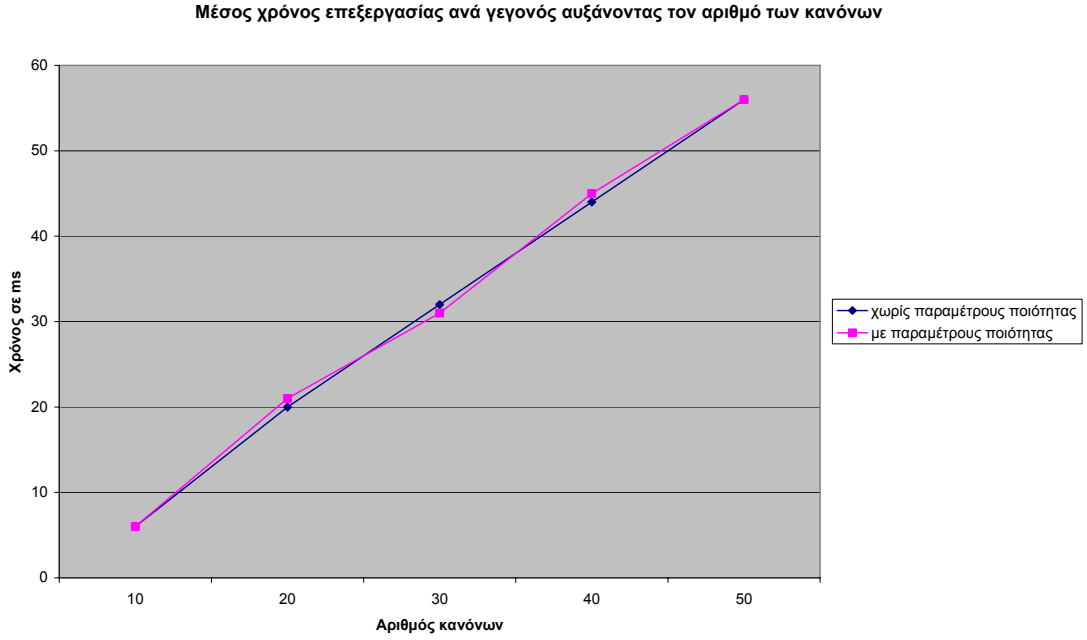
Στο Σχήμα 9.5 φαίνεται ο μέσος χρόνος επεξεργασίας ανά γεγονός αυξάνοντας τον αριθμό των κανόνων. Παρατηρούμε ότι ο έλεγχος αξιοπιστίας δεν επηρεάζει χρονικά αυτή τη φάση, καθώς όλοι οι υπολογισμοί έχουν γίνει κατά τη φάση της συντακτικής ανάλυσης.

Πίνακας 9.5 Χρονική Απόκριση της Υπηρεσίας Αυξάνοντας τον Αριθμό των Κανόνων.

Αριθμός Κανόνων	10	20	30	40	50
Χρόνος συντακτικής ανάλυσης σε *ms*	62	125	187	266	344
Με έλεγχο αξιοπιστίας	150	328	531	703	920
Μέσος χρόνος επεξεργασίας ανά γεγονός σε *ms*	6	20	32	44	56
Με έλεγχο αξιοπιστίας	6	21	31	45	56



Σχήμα 9.4 Γραφική Αναπαράσταση του Χρόνου της Συντακτικής Ανάλυσης Συναρτήσει του Αριθμού των Κανόνων.



Σχήμα 9.5 Γραφική Αναπαράσταση του Μέσου Χρόνου Επεξεργασίας Ανά Γεγονός Συναρτήσει του Αριθμού των Κανόνων.

Μεταβολή Αριθμού Συνθηκών και Ενεργειών

Σε αυτό το σύνολο πειραμάτων μεταβάλαμε τον αριθμό των συνθηκών και των ενεργειών στους κανόνες και μετρήσαμε τη χρονική απόκριση της υπηρεσίας. Στους Πίνακες 9.6 και 9.7 φαίνονται οι χρονικές τιμές που προέκυψαν μεταβάλλοντας τις συνθήκες και τις ενέργειες αντίστοιχα.

Όπως φαίνεται από τους Πίνακες 9.6 και 9.7, ο χρόνος της συντακτικής ανάλυσης αυξάνει περισσότερο αυξάνοντας τις συνθήκες των κανόνων. Αυτό συμβαίνει διότι το τμήμα των συνθηκών είναι πιο σύνθετο από αυτό των ενεργειών και προστίθενται περισσότεροι κόμβοι στο δέντρο. Επίσης και ο μέσος χρόνος ανά γεγονός αυξάνεται με την αύξηση του αριθμού των συνθηκών καθώς μεγαλώνει ο χρόνος αποτίμησης των κανόνων. Λιγότερο επηρεάζεται ο μέσος χρόνος ανά γεγονός από την αύξηση των ενεργειών αφού η εκτέλεση των ενεργειών δεν περιέχει κάποια σύνθετη διαδικασία.

Πίνακας 9.6 Χρονική Απόκριση Μεταβάλλοντας τον Αριθμό των Συνθηκών.

Αριθμός Συνθηκών	2	4	6	8	10
Χρόνος συντακτικής ανάλυσης σε *ms*	46	52	64	77	81
Με έλεγχο αξιοπιστίας	125	141	172	203	250
Μέσος χρόνος επεξεργασίας ανά γεγονός σε *ms*	0	3	9	17	22
Με έλεγχο αξιοπιστίας	0	4	9	15	20

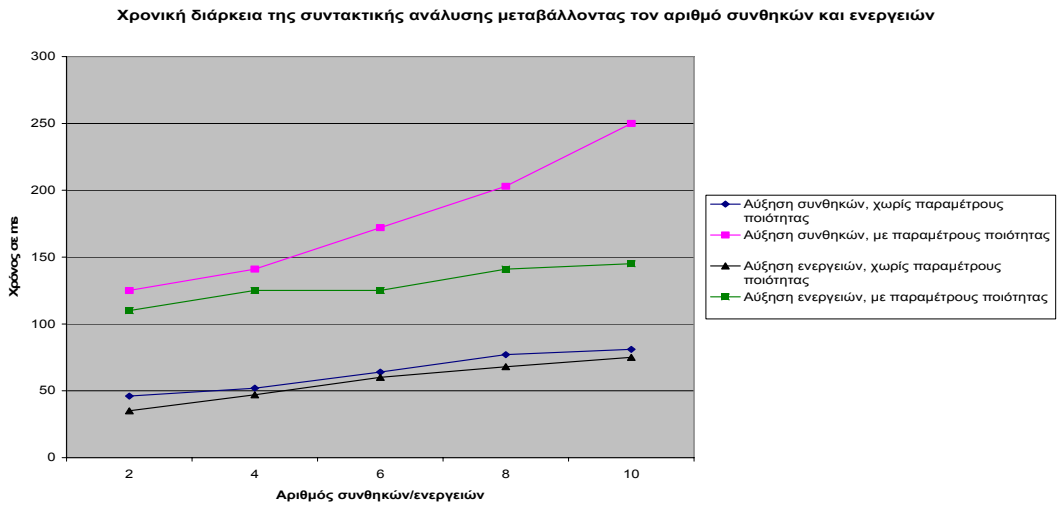
Κανόνες: *10*, Ενέργειες: *3*, Πιθανότητα εμφάνισης χρονικού τελεστή: ½.

Πίνακας 9.7 Χρονική Απόκριση Μεταβάλλοντας τον Αριθμό των Ενεργειών.

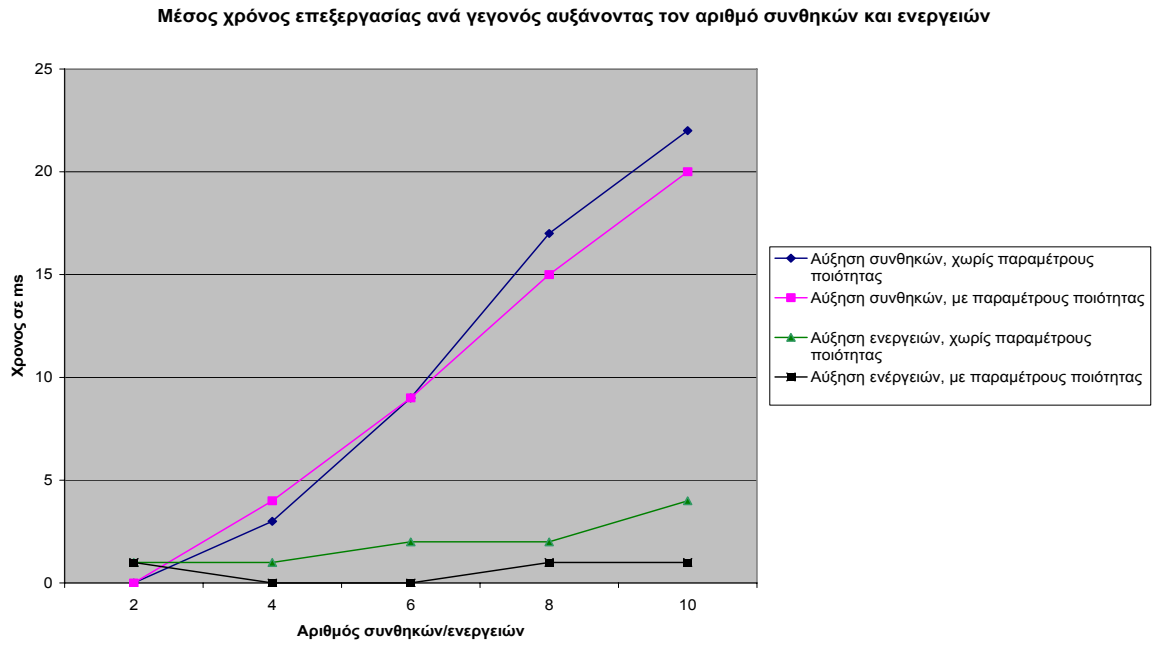
Αριθμός Ενεργειών	2	4	6	8	10
Χρόνος συντακτικής ανάλυσης σε *ms*	35	47	60	68	75
Με έλεγχο αξιοπιστίας	110	125	125	141	145
Μέσος χρόνος επεξεργασίας ανά γεγονός σε *ms*	1	1	2	2	4
Με έλεγχο αξιοπιστίας	1	0	0	1	1

Κανόνες: *10*, Συνθήκες: *3*, Πιθανότητα εμφάνισης χρονικού τελεστή: ½.

Επιπλέον παρατηρούμε ότι η προσθήκη του ελέγχου αξιοπιστίας επιβαρύνει χρονικά την υπηρεσία μόνο όμως κατά τη διάρκεια της συντακτικής ανάλυσης, όπου γίνονται οι υπολογισμοί αξιοπιστίας. Η επιβάρυνση είναι μεγαλύτερη όταν αυξάνονται οι συνθήκες των κανόνων και αμελητέα όταν αυξάνονται οι ενέργειες.



Σχήμα 9.6 Γραφική Αναπαράσταση του Χρόνου Συντακτικής Ανάλυσης Συναρτήσει του Αριθμού των Συνθηκών και των Ενεργειών.



Σχήμα 9.7 Γραφική Αναπαράσταση του Μέσου Χρόνου Επεξεργασίας Ανά Γεγονός Συναρτήσει του Αριθμού των Συνθηκών και των Ενεργειών.

Όπως βλέπουμε στο Σχήμα 9.6, ο υπολογισμός αξιοπιστίας των κανόνων αυξάνει αρκετά το χρόνο της συντακτικής ανάλυσης. Σε κάθε περίπτωση όπως και πριν, η αύξηση του αριθμού των συνθηκών επιφέρει μεγαλύτερη αύξηση στο χρόνο της συντακτικής ανάλυσης από ότι η αύξηση των ενεργειών. Στο Σχήμα 9.7 παρατηρούμε ότι η αύξηση των συνθηκών στους κανόνες αυξάνει περίπου γραμμικά το μέσο χρόνο επεξεργασίας ανά γεγονός. Αντίθετα η αύξηση των ενεργειών επηρεάζει ελάχιστα τη χρονική απόκριση της επεξεργασίας.

Μεταβολή Πιθανότητας Εμφάνισης Χρονικού Τελεστή

Σε αυτό το σύνολο πειραμάτων μεταβάλαμε την πιθανότητα εμφάνισης χρονικού τελεστή και μετρήσαμε το χρόνο της συντακτικής ανάλυσης και το μέσο χρόνο επεξεργασίας ανά γεγονός. Τα πειράματα εκτελέστηκαν με *10* κανόνες, *4* συνθήκες και *4* ενέργειες σε κάθε κανόνα. Οι λειτουργίες υπολογισμού και ελέγχου αξιοπιστίας δεν σχετίζονται με την ύπαρξη χρονικών τελεστών και δεν συμπεριλήφθηκαν σε αυτά τα πειράματα.

Λογικά αναμένουμε η αύξηση των χρονικών τελεστών στις συνθήκες και στις ενέργειες να μην επηρεάσει σημαντικά χρονικά τη συντακτική ανάλυση. Ο συντακτικός

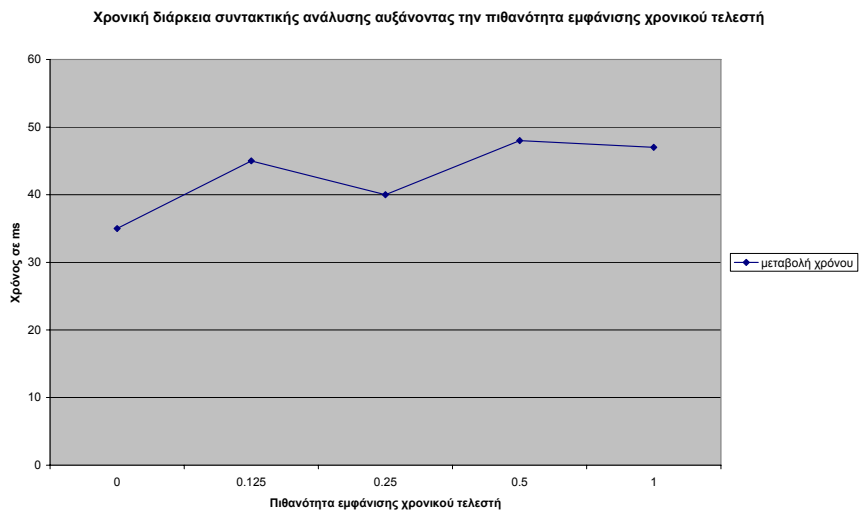
αναλυτής ελέγχει όλα τα συνθετικά του κανόνα αν έχουν χρονικό τελεστή και αν υπάρχει κάνει μια ανάθεση στο αντίστοιχο πεδίο ενός κόμβου του δέντρου.

Στη φάση της επεξεργασίας γεγονότων αναμένουμε ο χρόνος να αυξηθεί ελάχιστα, από την ύπαρξη χρονικών τελεστών. Ο λόγος είναι οι χρονικές ενημερώσεις που γίνονται στις συνθήκες που έχουν χρονικούς τελεστές και κυρίως οι ενέργειες ικανοποίησης των μελλοντικών τελεστών. Θυμίζουμε ότι οι τελεστές *[henceforth]* και *[until]* επιβάλλουν την εκτέλεση κάποιων ενεργειών διαμόρφωσης στο δέντρο των κανόνων, οι οποίες αυξάνουν το χρόνο επεξεργασίας του γεγονότος που τις προκάλεσε. Στον Πίνακα 9.8 φαίνονται οι τιμές που προέκυψαν από τα πειράματα.

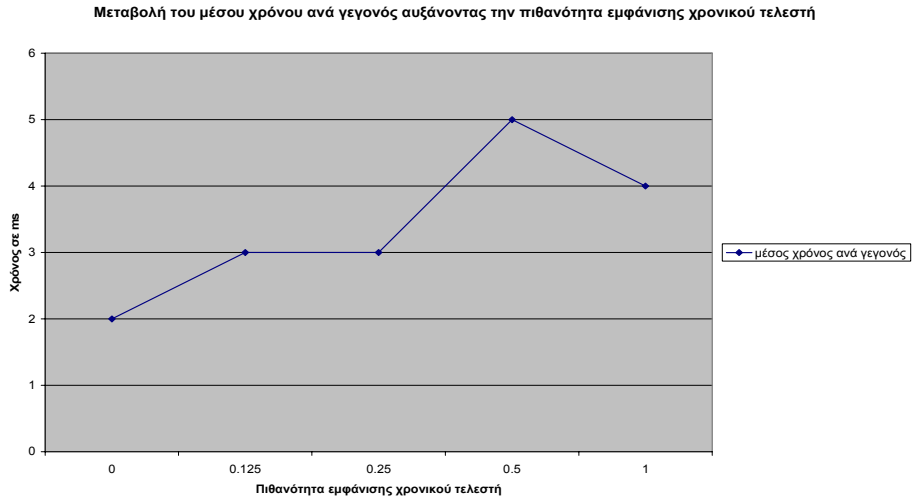
Στο Σχήμα 9.8 φαίνεται η μεταβολή του χρόνου συντακτικής ανάλυσης αυξάνοντας την πιθανότητα εμφάνισης χρονικού τελεστή. Παρατηρούμε ότι ο χρόνος της συντακτικής ανάλυσης είναι σχετικά ανεπηρέαστος από την ύπαρξη χρονικών τελεστών. Στο Σχήμα 9.9 φαίνεται η μεταβολή του μέσου χρόνου ανά γεγονός αυξάνοντας την πιθανότητα εμφάνισης χρονικού τελεστή. Παρατηρούμε ότι η ύπαρξη χρονικών τελεστών δεν επηρεάζει ιδιαίτερα το χρόνο επεξεργασίας.

Πίνακας 9.8 Χρονική Απόκριση Αυξάνοντας την Πιθανότητα Εμφάνισης Χρονικού Τελεστή.

Πιθανότητα εμφάνισης χρονικού τελεστή	0	1/8	1/4	1/2	1
Χρόνος συντακτικής ανάλυσης	35	47	40	50	47
Μέσος χρόνος επεξεργασίας ανά γεγονός σε *ms*	2	3	3	5	4



Σχήμα 9.8 Μεταβολή του Χρόνου Συντακτικής Ανάλυσης Αυξάνοντας την Πιθανότητα Εμφάνισης Χρονικού Τελεστή.



Σχήμα 9.9 Μεταβολή του Μέσου Χρόνου Επεξεργασίας Ανά Γεγονός Αυξάνοντας την Πιθανότητα Εμφάνισης Χρονικού Τελεστή.

9.5 Χρήση CPU

Σε αυτό το σύνολο πειραμάτων μελετήσαμε τη χρήση της *CPU* στις δύο φάσεις λειτουργίας της υπηρεσίας. Η χρήση της *CPU* μελετήθηκε με ένα εργαλείο του *WTK* που μετράει τους κύκλους της *CPU* για κάθε μέθοδο του προγράμματος που εκτελείται. Όπως αναφέρεται στο εγχειρίδιο του *WTK* οι τιμές δεν αντικατοπτρίζουν τις πραγματικές πάνω σε μία κινητή συσκευή αλλά είναι μία καλή προσέγγιση αυτών.

Η *CPU* μετρήθηκε συναρτήσει του αριθμού των κανόνων, του αριθμού συνθηκών/ενεργειών και της πιθανότητας εμφάνισης χρονικού τελεστή. Όλα τα πειράματα εκτελέστηκαν δύο φορές, με έλεγχο αξιοπιστίας και χωρίς, για να διαπιστώσουμε πόσο επηρεάζεται η χρήση της *CPU*. Τα αποτελέσματα που αναμένουμε από αυτό το σύνολο των πειραμάτων είναι ανάλογα με εκείνα του χρόνου. Η κύκλοι της *CPU* και ο χρόνος αυξάνονται παράλληλα και αναμένουμε να δούμε ανάλογα αποτελέσματα από τις μεταβολές των παραμέτρων.

Όπως και στα πειράματα του χρόνου, έτσι και σε αυτό το σύνολο πειραμάτων, αναμένουμε τα αποτελέσματα να εξαρτώνται μερικώς από το σύνολο των κανόνων που δημιουργείται. Παρόλα αυτά, σε αυτό το σύνολο, δεν είχαμε τη δυνατότητα να πάρουμε αυτόματα τους μέσους όρους από πολλαπλές εκτελέσεις, καθώς οι τιμές δίνονταν από έναν πίνακα του *WTK* στο τέλος κάθε εκτέλεσης.

Έτσι κάθε πείραμα εκτελέστηκε *5* φορές και επιλέχθηκαν οι μέγιστες τιμές της *CPU*, για τη φάση της συντακτικής ανάλυσης και της επεξεργασίας γεγονότων. Έπειτα από το συνολικό ποσό *CPU* που διήρκεσε η φάση της επεξεργασίας πήραμε το μέσο όρο ανά γεγονός.

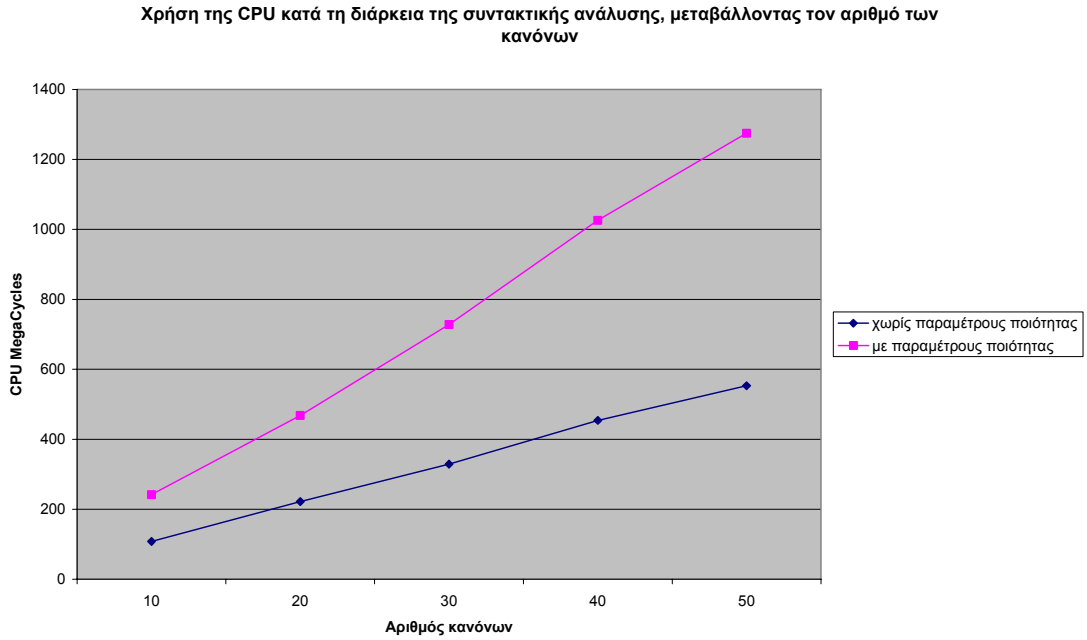
Μεταβολή Αριθμού Κανόνων

Σε αυτό το πείραμα μετρήσαμε τη χρήση της *CPU* μεταβάλλοντας τον αριθμό των κανόνων. Η αύξηση του αριθμού των κανόνων αναμένουμε να επιφέρει αύξηση της χρήσης της *CPU* και στις δύο φάσεις λειτουργίας. Τα πειράματα εκτελέστηκαν με *5* συνθήκες και *3* ενέργειες σε κάθε κανόνα και πιθανότητα εμφάνισης χρονικού τελεστή *½*.

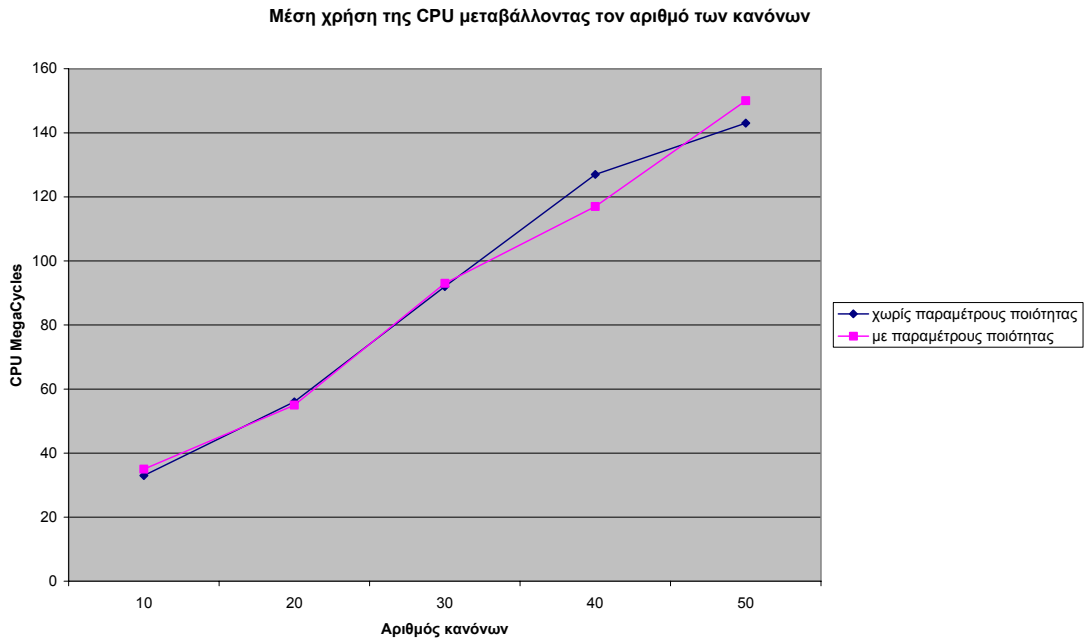
Στα Σχήματα 9.10 και 9.11 παρατηρούμε ότι η χρήση της *CPU* αυξάνει όσο αυξάνεται ο αριθμός των κανόνων και στις δύο φάσεις λειτουργίας. Στη φάση της συντακτικής ανάλυσης βλέπουμε ότι ο υπολογισμός αξιοπιστίας αυξάνει τη χρήση της *CPU*, κάτι που αναμενόταν. Αντίθετα στη φάση επεξεργασίας των γεγονότων η *CPU* μένει ανεπηρέαστη από τον έλεγχο αξιοπιστίας, καθώς δεν εκτελούνται επιπλέον πράξεις υπολογισμού.

Πίνακας 9.9 Χρήση της CPU Αυξάνοντας τον Αριθμό των Κανόνων.

Αριθμός Κανόνων	10	20	30	40	50
Χρήση της *CPU* κατά τη διάρκεια της συντακτικής ανάλυσης σε *MegaCycles*	108	222	329	454	553
Με έλεγχο αξιοπιστίας	242	468	728	1026	1275
Μέση χρήση της *CPU* ανά γεγονός σε *MegaCycles*	33	56	92	127	143
Με έλεγχο αξιοπιστίας	35	55	93	117	150



Σχήμα 9.10 Μεταβολή Χρήσης της CPU, Κατά τη Διάρκεια της Συντακτικής Ανάλυσης, Αυξάνοντας τον Αριθμό των Κανόνων.



Σχήμα 9.11 Μεταβολή της Μέσης Χρήσης της CPU Ανά Γεγονός, στη Φάση Επεξεργασίας των Γεγονότων, Αυξάνοντας τον Αριθμό των Κανόνων.

Μεταβολή Αριθμού Συνθηκών και Ενεργειών

Όπως και στα πειράματα μέτρησης του χρόνου αναμένουμε η αύξηση του αριθμού συνθηκών και ενεργειών να αυξάνει τη χρήση της *CPU* και στις δύο φάσεις λειτουργίας. Η αύξηση των συνθηκών αναμένουμε να αυξάνει αισθητά τη χρήση της *CPU* λόγω της αυξανόμενης πολυπλοκότητας στην αποτίμηση των κανόνων. Λιγότερο αναμένουμε να επηρεάζει τη χρήση της *CPU* η αύξηση του αριθμού των ενεργειών. Στους Πίνακες 9.10 και 9.11 φαίνονται οι τιμές των κύκλων της *CPU* που προέκυψαν από τις μεταβολές του αριθμού των συνθηκών και των ενεργειών αντίστοιχα.

Πίνακας 9.10 Χρήση της CPU Αυξάνοντας τον Αριθμό των Συνθηκών.

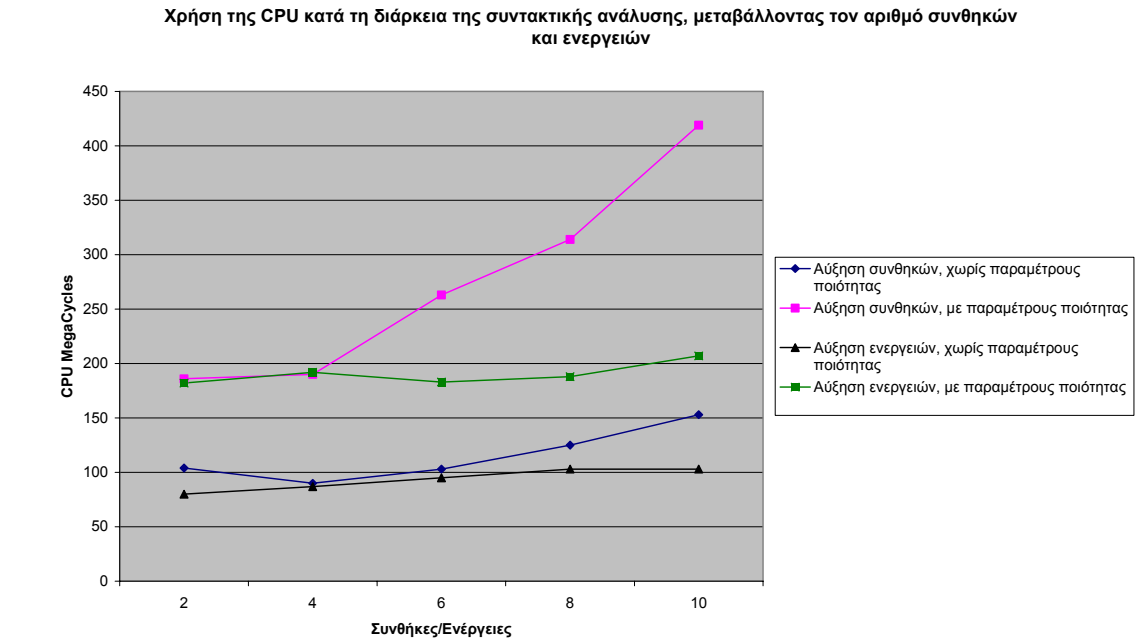
Αριθμός Συνθηκών	2	4	6	8	10
Χρήση της *CPU* κατά τη διάρκεια της συντακτικής ανάλυσης σε *MegaCycles*	104	90	103	125	153
Με έλεγχο αξιοπιστίας	186	190	263	314	419
Μέση χρήση της *CPU* ανά γεγονός σε *MegaCycles*	13	32	39	45	56
Με έλεγχο αξιοπιστίας	13	25	40	42	59

Κανόνες: *10*, Ενέργειες: *3*, Πιθανότητα εμφάνισης χρονικού τελεστή: ½.

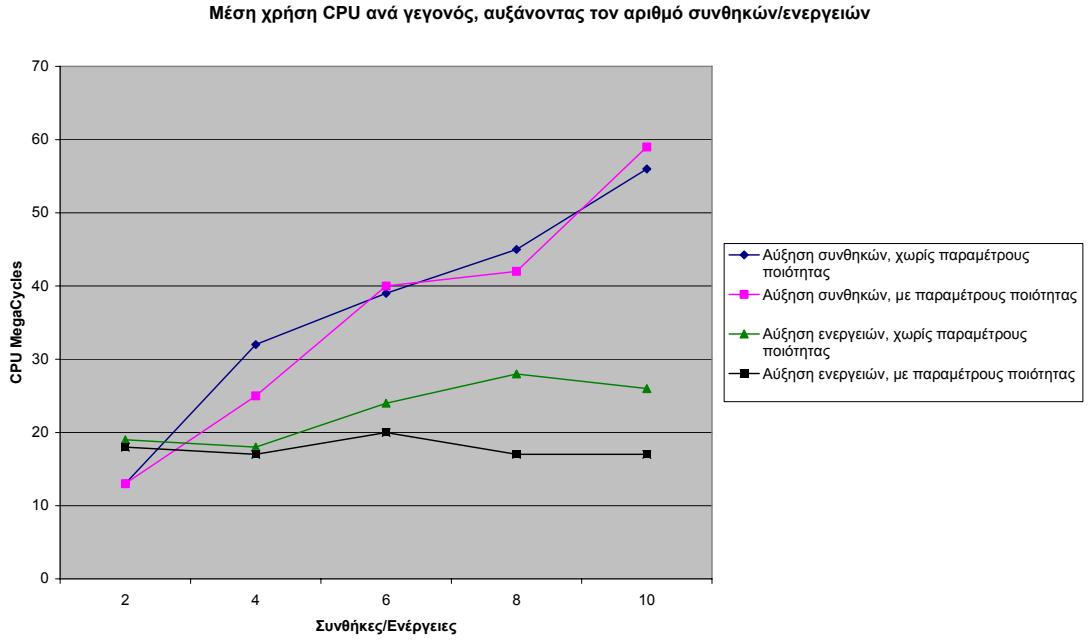
Πίνακας 9.11 Χρήση της CPU Αυξάνοντας τον Αριθμό των Ενεργειών.

Αριθμός Ενεργειών	2	4	6	8	10
Χρήση της *CPU* κατά τη διάρκεια της συντακτικής ανάλυσης σε *MegaCycles*	80	87	95	103	103
Με έλεγχο αξιοπιστίας	182	192	183	188	207
Μέση χρήση της *CPU* ανά γεγονός σε *MegaCycles*	19	18	24	28	26
Με έλεγχο αξιοπιστίας	18	17	20	17	17

Κανόνες: *10*, Συνθήκες: *3*, Πιθανότητα εμφάνισης χρονικού τελεστή: ½.



Σχήμα 9.12 Μεταβολή της CPU, στη Φάση της Συντακτικής Ανάλυσης, Αυξάνοντας τον Αριθμό Συνθηκών/Ενεργειών.



Σχήμα 9.13 Μεταβολή της Μέσης CPU Ανά Γεγονός, στη Φάση της Επεξεργασίας Γεγονότων, Αυξάνοντας τον Αριθμό Συνθηκών/Ενεργειών.

Στο Σχήμα 9.12 φαίνεται ότι η αύξηση των συνθηκών αυξάνει τη χρήση της *CPU* πολύ περισσότερο από ότι η αύξηση των ενεργειών, κάτι που το είδαμε και στις χρονικές

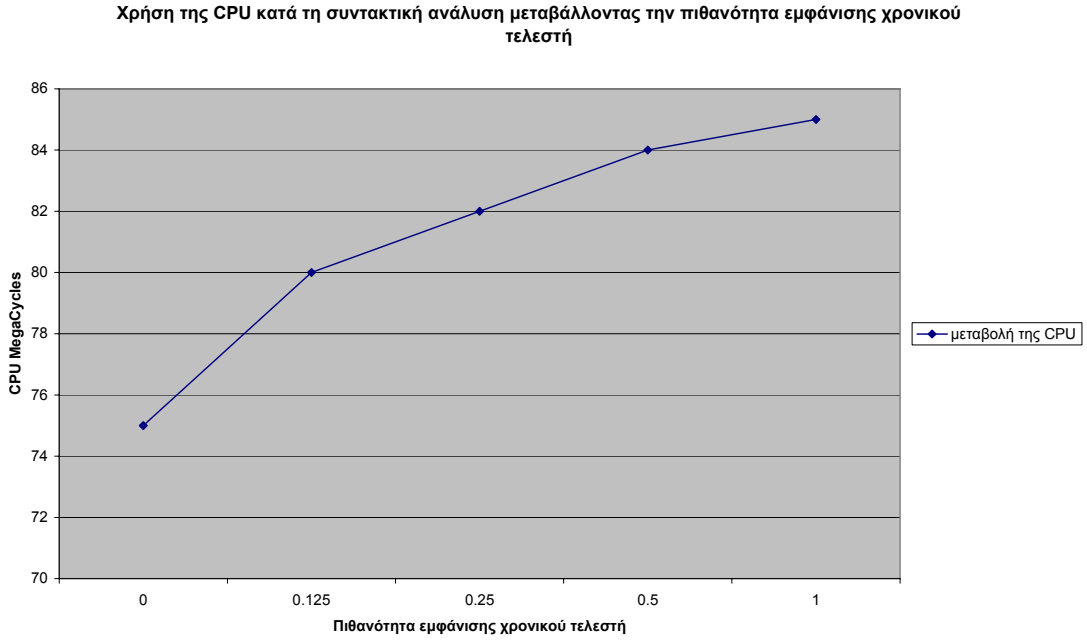
μετρήσεις. Στο Σχήμα 9.13 βλέπουμε ότι η μέση χρήση της *CPU* ανά γεγονός, αυξάνεται αισθητά με την αύξηση του αριθμού των ενεργειών, ενώ ελάχιστη αύξηση της *CPU* επιφέρει η αύξηση των ενεργειών.

Μεταβολή Πιθανότητας Εμφάνισης Χρονικού Τελεστή

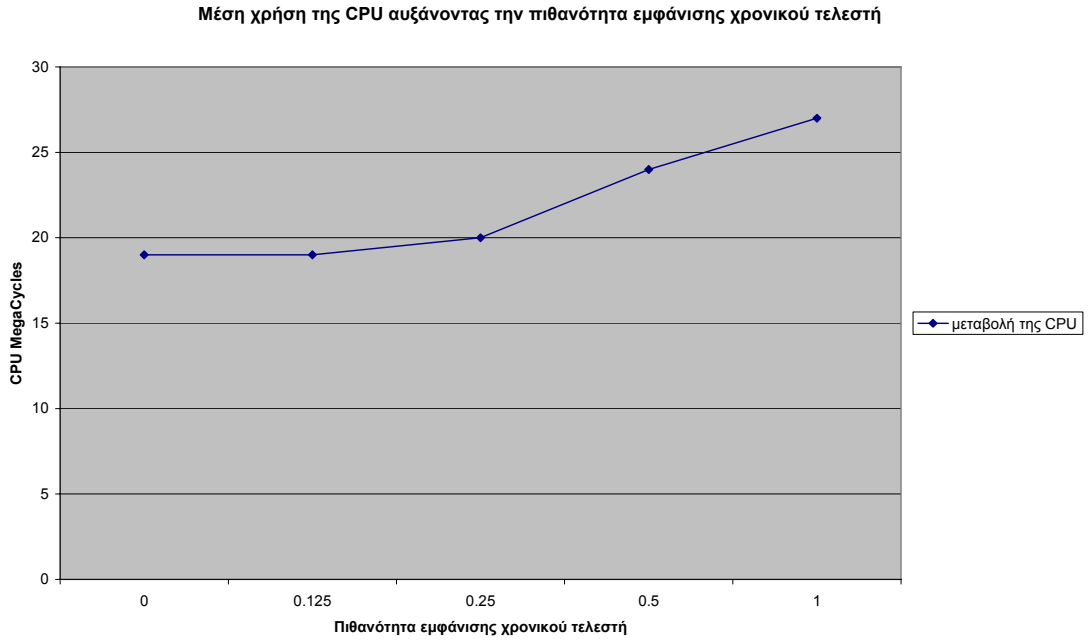
Η αύξηση των χρονικών τελεστών σε συνθήκες και ενέργειες αναμένουμε να αυξήσει ελάχιστα τη χρήση της *CPU* στις δύο φάσεις λειτουργίας.

Πίνακας 9.12 Χρήση της CPU Αυξάνοντας την Πιθανότητα Εμφάνισης Χρονικού Τελεστή.

Πιθανότητα εμφάνισης χρονικού τελεστή σε συνθήκη ή ενέργεια	0	1/8	1/4	1/2	1
Χρήση της *CPU* κατά τη διάρκεια της συντακτικής ανάλυσης σε *MegaCycles*	75	80	82	84	85
Μέση χρήση της *CPU* ανά γεγονός σε *MegaCycles*	19	19	20	24	27



Σχήμα 9.14 Μεταβολή της CPU στη Φάση της Συντακτικής Ανάλυσης, Αυξάνοντας την Πιθανότητα Εμφάνισης Χρονικού Τελεστή.



Σχήμα 9.15 Μεταβολή της Μέσης Χρήσης της CPU Ανά Γεγονός, Αυξάνοντας την Πιθανότητα Εμφάνισης Χρονικού Τελεστή.

Όπως βλέπουμε στα Σχήματα 9.14 και 9.15, η χρήση της *CPU* αυξάνεται ελάχιστα όσο αυξάνονται οι χρονικοί τελεστές και στις δύο φάσεις λειτουργίας. Στη φάση της συντακτικής ανάλυσης, η αύξηση οφείλεται στις αναθέσεις των χρονικών τελεστών στα πεδία των κόμβων του δέντρου. Στη φάση της επεξεργασίας γεγονότων η αύξηση οφείλεται στις χρονικές ενημερώσεις και στις ενέργειες ικανοποίησης των μελλοντικών τελεστών.

ΚΕΦΑΛΑΙΟ 10. ΣΥΜΠΕΡΑΣΜΑΤΑ

Στο κεφάλαιο αυτό συνοψίζουμε την παρούσα εργασία, αναφέρουμε τα καινοτόμα στοιχεία και συγκρίνουμε την εργασία με τις σχετικές εργασίες που υπάρχουν στη βιβλιογραφία.

10.1 Σύνοψη

Στη μεταπτυχιακή αυτή εργασία, εξετάσαμε τα θέματα που αφορούν τη σχεδίαση και ανάπτυξη ενδιάμεσου λογισμικού για την υποστήριξη *CA* εφαρμογών. Κάναμε μία ανασκόπηση της βιβλιογραφίας στο χώρο των *CA* υπολογιστικών συστημάτων και αναφέραμε τις πιο σημαντικές προσεγγίσεις που έχουν προταθεί.

Η δική μας έρευνα επικεντρώθηκε στην ανάπτυξη και σχεδίαση ενδιάμεσου λογισμικού για *CA* εφαρμογές, πάνω σε κινητές συσκευές. Προτείναμε μια υπηρεσία που διαχειρίζεται το *context* με διαφάνεια προς την εφαρμογή και προσαρμόζει αυτόματα, τη λειτουργία της εφαρμογής, στις αλλαγές του *context*. Η υπηρεσία σχεδιάστηκε και υλοποιήθηκε για να λειτουργεί πάνω σε κινητές συσκευές, κάτω από κινητές εφαρμογές. Μπορεί να ενσωματωθεί στο λογισμικό της κινητής εφαρμογής, λειτουργώντας αυτόνομα ή σαν μέλος ενός ευρύτερου *CA* ενδιάμεσου λογισμικού.

Η υπηρεσία αναλαμβάνει όλη τη διαχείριση της πληροφορίας που λαμβάνεται από τα χαμηλότερα επίπεδα και προσαρμόζει τις λειτουργίες της εφαρμογής, χωρίς τη συμμετοχή

του χρήστη. Μπορεί να χρησιμοποιηθεί σαν εργαλείο, από τους προγραμματιστές, μέσω του οποίου μπορούν να δώσουν *CA* συμπεριφορά σε μία κινητή εφαρμογή.

Παρέχει ένα πλαίσιο για την υλοποίηση της επικοινωνίας με εξωτερικούς προμηθευτές. Εκεί ο προγραμματιστής μπορεί να υλοποιήσει την επικοινωνία με τις πηγές του *context* ή με άλλα τμήματα ενδιάμεσου λογισμικού τα οποία παρέχουν επεξεργασμένη την πληροφορία (π.χ. *context* εξυπηρέτες).

Για τη διαχείριση του *context* και την προσαρμογή της εφαρμογής, προτείναμε ένα μοντέλο βασισμένο στην *MTL* πρώτης τάξης. Η συμπεριφορά της εφαρμογής σε σχέση με τις αλλαγές του *context* μοντελοποιείται σε ένα σύνολο λογικών κανόνων συνθήκης - ενέργειας. Οι κανόνες αυτοί καθορίζουν, τις ενέργειες που πρέπει να εκτελέσει η υπηρεσία, αν ισχύσουν συγκεκριμένες *context* συνθήκες. Οι κανόνες περιλαμβάνουν ένα σύνολο χρονικών τελεστών για το χρονικό προσδιορισμό συνθηκών και ενεργειών. Επίσης προσθέσαμε στο μοντέλο τρεις παραμέτρους που περιγράφουν την ποιότητα του *context*. Για την αποτίμηση των κανόνων και την επιλογή ενεργειών προσαρμογής, υλοποιήσαμε μία μέθοδο που λαμβάνει υπόψιν το χρόνο και την ποιότητα των δεδομένων κατά τη διαδικασία αποτίμησης.

Το *context* που συλλέγεται από τους προμηθευτές μοντελοποιείται σε εκφράσεις, οι οποίες ενθυλακώνονται σε γεγονότα. Στα γεγονότα περικλείεται και ο χρόνος που πραγματοποιήθηκαν. Οι κανόνες ελέγχονται σύμφωνα με τα γεγονότα και αποφασίζεται αν πρέπει να πραγματοποιηθούν ενέργειες προσαρμογής. Η κατηγορηματική λογική πρώτης τάξεως έχει προταθεί στη βιβλιογραφία για την αναπαράσταση και την πραγματοποίηση *context* συλλογισμού [92]. Το μοντέλο, όμως, που προτάθηκε δεν περιγράφει τις χρονικές εξαρτήσεις του *context*, ενώ η υπηρεσία προσαρμογής έχει υλοποιηθεί μόνο για περιορισμένους γεωγραφικά, ενεργούς χώρους.

Η υπηρεσία έχει την ικανότητα να αυτοδιαμορφώνει τη συμπεριφορά της, βελτιώνοντας την απόδοση του συστήματος. Η αυτοδιαμόρφωση δεν πραγματοποιείται άμεσα, σαν ενέργεια κάποιου κανόνα, αλλά μπορεί να προκύψει έμμεσα, σαν αποτέλεσμα των ενεργειών ενός κανόνα.

Τέλος, προσδώσαμε στην υπηρεσία ανακλαστικά χαρακτηριστικά, δίνοντας την δυνατότητα στο χρήστη να διαμορφώσει δυναμικά τη συμπεριφορά της. Η υπηρεσία παρέχει ένα *API*, με μεθόδους για τη δυναμική διαμόρφωση των κανόνων και συνεπώς του ενδιάμεσου λογισμικού, κατά την λειτουργία του συστήματος. Επιπλέον υλοποιήσαμε ένα *GUI* για κινητές συσκευές, με τις λειτουργίες διαμόρφωσης της υπηρεσίας, ώστε να διευκολυνθεί το έργο του προγραμματιστή.

10.1.1 Ικανοποίηση Απαιτήσεων

Η σχεδίαση και ανάπτυξη της υπηρεσίας έγινε με βάση τις απαιτήσεις που αναλύθηκαν στο Κεφάλαιο 4. Στη συνέχεια περιγράφονται τα χαρακτηριστικά της υπηρεσίας που σχετίζονται με τις λειτουργικές και μη λειτουργικές απαιτήσεις που είχαμε θέσει.

Λειτουργικές Απαιτήσεις

Η υπηρεσία χρησιμοποιεί ένα μοντέλο για την αναπαράσταση και διαχείριση του *context* βασισμένο στη χρονική κατηγορηματική λογική πρώτης τάξεως. Το μοντέλο μπορεί να περιγράψει τις χρονικές εξαρτήσεις του *context* με τη χρήση *6* χρονικών τελεστών. Η μέθοδος που υλοποιήθηκε με βάση αυτό το μοντέλο, επεξεργάζεται το *context* λαμβάνοντας υπόψιν και τα χρονικά χαρακτηριστικά της πληροφορίας.

Η συλλογή του *context* γίνεται με ένα μηχανισμό ασύγχρονης ενημέρωσης. Η υπηρεσία παρακολουθεί τους προμηθευτές και ανιχνεύει σημαντικές μεταβολές στις τιμές που παράγουν. Οι τιμές αυτές στέλνονται ασύγχρονα στην υπηρεσία για επεξεργασία. Η συλλογή των γεγονότων γίνεται ανεξάρτητα από τον τύπο της πληροφορίας που παράγουν οι προμηθευτές. Η υπηρεσία μπορεί να συλλέξει και να επεξεργαστεί ακεραίους, δεκαδικούς και συμβολοσειρές.

Η υπηρεσία με βάση τους κανόνες και τη νέα γνώση που μεταφέρουν τα γεγονότα στο σύστημα παίρνει αποφάσεις για την προσαρμογή της εφαρμογής. Το έμπειρο σύστημα με κανόνες που χρησιμοποιείται, παρέχει την ευφυία που απαιτεί η λήψη αποφάσεων. Η υπηρεσία έχει την ικανότητα, να εξάγει συμπεράσματα για τις ενέργειες που πρέπει να εκτελεστούν σε συγκεκριμένες μεταβολές του *context*.

Η προσαρμογή της εφαρμογής στο *context* γίνεται αυτόματα από την υπηρεσία. Χρησιμοποιεί μία *callback* μέθοδος που αντιστοιχίζει τις ενέργειες που αποφασίστηκαν από το έμπειρο σύστημα, σε εκτελέσεις μεθόδων της εφαρμογής. Η εκτέλεση των ενεργειών γίνεται με διαφάνεια προς την εφαρμογή.

Για το χειρισμό της αξιοπιστίας του *context*, χρησιμοποιούνται οι τρεις πιθανότητες που χαρακτηρίζουν την ποιότητα της πληροφορίας, που παράγουν οι προμηθευτές. Οι πιθανότητες αυτές παρέχουν ένα κριτήριο για την εγκυρότητα της ικανοποίησης των κανόνων. Η ικανοποίηση ενός κανόνα μπορεί να κριθεί έγκυρη ή άκυρη, ανάλογα με την αξιοπιστία των προμηθευτών, που παρήγαγαν τις τιμές, που προκάλεσαν την ικανοποίηση του κανόνα.

Τα δύο *GUIs* που παρέχει η υπηρεσία για τη διαχείριση των κανόνων και των πιθανοτήτων αξιοπιστίας, επιτρέπουν τη δυναμική διαμόρφωση της υπηρεσίας από το χρήστη. Ο χρήστης αλλάζοντας τους κανόνες και τις πιθανότητες αξιοπιστίας, μπορεί να διαμορφώσει τη συμπεριφορά της υπηρεσίας και συνεπώς να μεταβάλει τη συμπεριφορά της εφαρμογής.

Το λογισμικό της υπηρεσίας παρέχεται με τη μορφή ενός πακέτου κλάσεων και διεπαφών. Ο προγραμματιστής της εφαρμογής μπορεί να χρησιμοποιήσει το πακέτο αυτό, για να το ενσωματώσει σε ήδη υπάρχον ενδιάμεσο λογισμικό ή να προσαρμόσει κατάλληλα την εφαρμογή πάνω στην υπηρεσία. Επιπλέον μπορεί να χρησιμοποιήσει τις μεθόδους και τις κλάσεις του πακέτου για να επεκτείνει τη λειτουργικότητα της υπηρεσίας, προσθέτοντας νέα συστατικά.

Μη Λειτουργικές Απαιτήσεις

Στο Κεφάλαιο 6 έγινε μία αξιολόγηση του μοντέλου που χρησιμοποιεί η υπηρεσία με βάση τις μη λειτουργικές απαιτήσεις που είχαν αναφερθεί. Στην αξιολόγηση αυτή αναλύσαμε τα χαρακτηριστικά του μοντέλου, που ικανοποιούν τις μη λειτουργικές απαιτήσεις.

Όπως περιγράφηκε στο Κεφάλαιο 7 όλα τα συστατικά της υπηρεσίας βρίσκονται πάνω στην κινητή συσκευή και διαχειρίζονται τους ίδιους υπολογιστικούς πόρους. Αυτή η

αρχιτεκτονική εξασφαλίζει την αυτόνομη λειτουργία της υπηρεσίας και την ανεξαρτησία από σταθερές υπολογιστικές μονάδες.

Η υπηρεσία λειτουργεί παράλληλα με την εφαρμογή σε διαφορετικά νήματα εκτέλεσης. Η παρακολούθηση των προμηθευτών και η επεξεργασία των γεγονότων εκτελούνται σε ξεχωριστά νήματα, επιτρέποντας την ταυτόχρονη λειτουργία υπηρεσίας και εφαρμογής.

Όπως έδειξαν οι πειραματικές μετρήσεις στο Κεφάλαιο 9, η κατανάλωση των υπολογιστικών πόρων είναι ικανοποιητικά μικρή, ώστε να είναι εφικτή η λειτουργία της υπηρεσίας πάνω σε κινητές συσκευές. Επιπλέον η χρονική απόκριση της υπηρεσίας, ήταν αρκετά μικρή, ώστε να μην γίνεται αντιληπτή η λειτουργία της από το χρήστη της εφαρμογής. Για το σκοπό αυτό το λογισμικό αναπτύχθηκε ελαχιστοποιώντας τις κλάσεις και τα μηνύματα που ανταλλάσσονταν μεταξύ των αντικειμένων.

Η υλοποίηση της υπηρεσίας έγινε αποκλειστικά με τα *APIs* που παρέχει η πλατφόρμα *J2ME*. Το γεγονός αυτό εξασφαλίζει στο λογισμικό τη μεγαλύτερη δυνατή μεταφερσιμότητα. Οι προδιαγραφές λογισμικού που ορίζονται από αυτή τη πλατφόρμα, έχουν υλοποιηθεί από τους περισσότερους κατασκευαστές κινητών συσκευών. Αυτό σημαίνει ότι η υπηρεσία μπορεί να λειτουργήσει σε όλες αυτές τις συσκευές, χωρίς να χρειαστούν αλλαγές στον κώδικα του λογισμικού.

Για να εξασφαλίσουμε την αξιοπιστία στη λειτουργία της υπηρεσίας, συμπεριλάβαμε στον κώδικά τα απαραίτητα τμήματα για την πρόβλεψη και το χειρισμό σφαλμάτων. Τα νήματα που τρέχουν ταυτόχρονα συγχρονίστηκαν, ώστε να μην προσπελαύνουν ταυτόχρονα κοινή μνήμη. Ο χειρισμός των πιθανών σφαλμάτων ή εξαιρέσεων γίνεται εσωτερικά από την υπηρεσία, εμφανίζοντας κατάλληλα μηνύματα λάθους στην οθόνη της κινητής συσκευής.

10.2 Συνεισφορά στην Έρευνα

Αρκετή έρευνα έχει γίνει την τελευταία δεκαετία στα *CA* συστήματα. Η έρευνα έχει εστιαστεί κυρίως στη σχεδίαση και ανάπτυξη ενδιάμεσου λογισμικού, που διευκολύνει την υλοποίηση και υποστήριξη *CA* εφαρμογών. Το ενδιάμεσο λογισμικό παρέχει υπηρεσίες για

τη συλλογή και διαχείριση του *context* και την παροχή του στις εφαρμογές, που συνήθως τρέχουν πάνω σε συσκευές με περιορισμένους πόρους. Τα περισσότερα *CA* συστήματα υιοθετούν μία ιεραρχική αρχιτεκτονική, με τα τμήματα του λογισμικού κατανεμημένα σε επίπεδα. Συνήθως υπάρχουν ένα η περισσότερα κεντρικά τμήματα, με αφθονία υπολογιστικών πόρων τα οποία αναλαμβάνουν τις διαδικασίες συλλογής, επεξεργασίας και διαχείρισης του *context* και το διανέμουν στις εφαρμογές.

Το τμήμα του ενδιάμεσου λογισμικού που τρέχει πάνω στις κινητές συσκευές έχει περιορισμένη λειτουργικότητα λόγω των λιγοστών υπολογιστικών πόρων της συσκευής. Στην κινητή συσκευή υλοποιείται μόνο η επικοινωνία μεταξύ της κινητής εφαρμογής και του εξυπηρέτη – διαχειριστή του *context* για την απόκτηση της πληροφορίας. Η μετέπειτα χρήση της πληροφορίας υλοποιείται σαν τμήμα της εκάστοτε εφαρμογής.

Ελάχιστη εργασία έχει γίνει για τη διαχείριση του *context* πάνω στη συσκευή και την προσαρμογή της εφαρμογής. Η προσαρμογή της εφαρμογής στις αλλαγές του *context* απαιτεί ενδιάμεσο λογισμικό πάνω στις κινητές συσκευές, που διαχειρίζεται το *context* και εκτελεί αυτόματα ενέργειες προσαρμογής. Στις περισσότερες προσεγγίσεις η εξαγωγή συμπερασμάτων για το *context* γίνεται στον κεντρικό κόμβο διαχείρισης και αφορά το συνδυασμό *context* πληροφορίας και την εξαγωγή υψηλότερου εννοιολογικά *context*. Στη συνέχεια η πληροφορία μεταφέρεται στην εφαρμογή σύγχρονα ή ασύγχρονα.

Οι χρονικές εξαρτήσεις του *context* είναι άλλο ένα θέμα, το οποίο δεν έχει προσεχθεί ιδιαίτερα από την ερευνητική κοινότητα. Η ανάγκη για την προσθήκη του χρόνου στη μοντελοποίηση του *context* έχει επισημανθεί σε κάποιες εργασίες [72,95]. Παρόλα δεν έχει προταθεί κάποια μέθοδος που να περιλαμβάνει τη χρονική διάσταση του *context* στις διαδικασίες εξαγωγής συμπερασμάτων.

Συνοπτικά τα καινοτόμα στοιχεία της εργασίας είναι τα εξής:

- Η υπηρεσία ενδιάμεσου λογισμικού μπορεί να λειτουργήσει αυτόνομα πάνω σε κινητές συσκευές και είναι ανεξάρτητη από κεντρικούς κόμβους παροχής υπηρεσιών.
- Η μοντελοποίηση του *context* ενσωματώνει τη χρονική διάσταση της πληροφορίας.

- Η υπηρεσία εξάγει συμπεράσματα για τις αλλαγές του *context* και προσαρμόζει αυτόματα τη λειτουργία της εφαρμογής. Στην αποτίμηση του *context* λαμβάνονται υπόψιν και οι χρονικές εξαρτήσεις της πληροφορίας.

10.3 Σχετικές Εργασίες

Στις εργασίες [13,15,61] προτείνονται συστήματα ενδιάμεσου λογισμικού που παρέχουν υπηρεσίες συλλογής και παροχής *context* στις εφαρμογές. Η πληροφορία συλλέγεται και παρέχεται μεταφρασμένη στην εφαρμογή, χωρίς να πραγματοποιούνται σύνθετες διαδικασίες εξαγωγής συμπερασμάτων.

Τα πλαίσια και οι υποδομές ενδιάμεσου λογισμικού, που προτείνονται στις εργασίες [27,32,55,56,60,65,66,69,72,75,77,80], ακολουθούν την προσέγγιση του κεντρικού εξυπηρέτη – διαχειριστή του *context*, που παρέχει υπηρεσίες *context* στις εφαρμογές πελάτες. Το τμήμα του πελάτη του ενδιάμεσου λογισμικού είναι υπεύθυνο μόνο για την επικοινωνία της εφαρμογής με τον εξυπηρέτη, για την απόκτηση του *context*. Αυτή η αρχιτεκτονική προσέγγιση παρουσιάζει τα εξής προβλήματα:

- Οι εφαρμογές προορίζονται μόνο για ευφυείς χώρους, καθώς η λειτουργία τους είναι άμεσα εξαρτημένη από την ύπαρξη του στατικού κόμβου διαχείρισης του *context*.
- Η εξάρτηση από έναν κόμβο εξυπηρέτη καθιστά τις κινητές εφαρμογές μη ανεκτικές σε σφάλματα.
- Οι ασταθείς ασύρματες συνδέσεις μεταξύ εξυπηρέτη και κινητών συσκευών δημιουργεί προβλήματα στη συχνή επικοινωνία που απαιτείται για την παροχή του *context*.

Σε κανένα από τα παραπάνω συστήματα δεν υπάρχει κάποιο τμήμα του ενδιάμεσου λογισμικού που να προσαρμόζει τη λειτουργία της εφαρμογής με βάση το *context*. Οι διαδικασίες συλλογισμού που εκτελούνται αφορούν μόνο την εξαγωγή επιπλέον πληροφορίας και όχι τις ενέργειες προσαρμογής της εφαρμογής. Επιπλέον στη διαχείριση της πληροφορίας δεν χρησιμοποιείται κάποιος μηχανισμός που να λαμβάνει υπόψιν τα χρονικά χαρακτηριστικά του *context*.

Στη *Gaia* [27,92] παρέχεται μία υπηρεσία για την προσαρμογή των εφαρμογών που εκτελούνται σε ευφυείς χώρους. Η προσαρμογή βασίζεται σε κανόνες κατηγορηματικής λογικής πρώτης τάξεως και μοιάζει αρκετά με τη δική μας προσέγγιση. Παρόλα αυτά οι ενέργειες προσαρμογής συντονίζονται από τους κεντρικούς εξυπηρέτες των ευφυών χώρων (υπάρχει ένας εξυπηρέτης για κάθε φυσικό χώρο) και πραγματοποιούνται μόνο για εφαρμογές που εκτελούνται μέσα στα φυσικά όρια του χώρου. Επιπλέον η μέθοδος που ακολουθείται, έχει περιορισμένες δυνατότητες αποτίμησης, καθώς δεν χρησιμοποιεί τη χρονική πληροφορία του *context*.

Το σύστημα *Rome* [9] παρέχει μια υπηρεσία για την προσαρμογή της εφαρμογής, η οποία βασίζεται στην έννοια του ενεργοποιητή *context*. Ο ενεργοποιητής μπορεί να αποτιμηθεί στο ενδιάμεσο λογισμικό της συσκευής αλλά η διαδικασία αποτίμησης εξαρτάται από έναν κεντρικό εξυπηρέτη. Ο εξυπηρέτης αποθηκεύει όλους τους ενεργοποιητές και τους προωθεί στις συσκευές για αποτίμηση. Η *context* πληροφορία αποθηκεύεται στον εξυπηρέτη και η συσκευή συμβάλλει στην αποτίμηση με συγκεκριμένη πληροφορία της εφαρμογής. Το μοντέλο του *context* που χρησιμοποιείται στο *Rome* δεν είναι τυπικά ορισμένο, ενώ δεν περιλαμβάνεται ο χρόνος στην αποτίμηση των ενεργοποιητών.

Στο *Hydrogen* [70] η διαχείριση του *context* γίνεται εξ ολοκλήρου πάνω σε κινητές συσκευές. Το σύστημα αυτό δεν περιλαμβάνει κεντρικό κόμβο διαχείρισης και στοχεύει σε ανάπτυξη εφαρμογών για *peer – to – peer* δίκτυα, όπου οι χρήστες κινούνται ανεξέλεγκτα χωρίς γεωγραφικούς περιορισμούς. Το *Hydrogen* παρέχει υπηρεσίες που αφορούν τη συλλογή, την παροχή και το διαμοιρασμό του *context* μεταξύ των εφαρμογών. Δεν υπάρχει όμως υπηρεσία που να αποτιμά το *context* και να προσαρμόζει κατάλληλα τη λειτουργία της εφαρμογής.

Το *CARISMA* [25] είναι ένα ενδιάμεσο λογισμικό για κινητές συσκευές. Σχεδιάστηκε για να λειτουργεί πάνω στην κινητή συσκευή και προσαρμόζει τις υπηρεσίες που χρησιμοποιεί η εφαρμογή με βάση το *context*. Η προσαρμογή πραγματοποιείται, εξετάζοντας το προφίλ της εφαρμογής και το τρέχον *context* και αποφασίζοντας μία πολιτική, από ένα προκαθορισμένο σύνολο πολιτικών, για τον τρόπο εκτέλεσης της υπηρεσίας. Περισσότερες λεπτομέρειες για τη διαδικασία προσαρμογής δεν αναφέρονται, ενώ δεν παρέχονται υπηρεσίες ή μηχανισμοί για τη συλλογή του *context* από τους προμηθευτές.

Στο συντακτικό που χρησιμοποιείται για τη σύνθεση του προφίλ της εφαρμογής, δεν περιλαμβάνονται τελεστές που να χαρακτηρίζουν χρονικά, τις συνθήκες για την επιλογή των πολιτικών. Η υλοποίηση έγινε στην πλατφόρμα *J2SE 1.4.1*, ενώ χρησιμοποιήθηκαν *XML* τεχνολογίες, όπως οι συντακτικοί αναλυτές *DOM* και *XPath*, οι οποίοι διαφέρουν από τους αντίστοιχους που παρέχονται από το *Web Services API 1.0* της πλατφόρμας *J2ME*. Συνεπώς το τελικό σύστημα δεν είναι συμβατό για χρήση σε κινητές συσκευές και η πειραματική αποτίμηση του δεν είναι αντιπροσωπευτική της πραγματικότητας.

10.4 Μελλοντική Εργασία

Ένα θέμα που πρέπει να εξεταστεί άμεσα είναι η αποδοτικότερη υλοποίηση κάποιων λειτουργιών του ενδιάμεσου λογισμικού. Η αποθήκευση των περιεχομένων των κανόνων, θα μπορούσε να γίνει με τη δημιουργία ενός ευρετηρίου, για την αποδοτικότερη αναζήτηση των παραμέτρων του *context* στους κανόνες. Σε αυτήν την προσέγγιση, η δομή του ευρετηρίου αποτελείται από μία λίστα με τις παραμέτρους του *context* που συμμετέχουν στους κανόνες. Κάθε παράμετρος, περιέχει ένα δείκτη σε μια λίστα με τους κανόνες που περιέχουν την παράμετρο. Με τη χρήση αυτού του ευρετηρίου, αναμένουμε η αναζήτηση να γίνεται πιο αποδοτικά. Κάθε παράμετρος *context*, που περιέχεται σε ένα γεγονός, δεν αναζητείται σε όλους τους κανόνες, αλλά μόνο σε αυτούς που δείχνει η αντίστοιχη εγγραφή (παράμετρος *context*) του ευρετηρίου.

Θα είχε ενδιαφέρον να δούμε κατά πόσο βελτιώνεται ο χρόνος επεξεργασίας των γεγονότων με τη χρήση ευρετηρίου, αλλά και πόσο επιπλέον χρόνο απαιτεί η κατασκευή του ευρετηρίου, στη φάση της συντακτικής ανάλυσης. Ακόμα, θα πρέπει να ελέγξουμε την επιπλέον μνήμη, που απαιτεί η αποθήκευση του ευρετηρίου στη μνήμη.

Ένα άλλο συστατικό της υπηρεσίας, που θα μπορούσε να υλοποιηθεί πιο αποδοτικά είναι η μέθοδος εξαγωγής συμπερασμάτων. Στη βιβλιογραφία έχουν προταθεί αλγόριθμοι, που ενδεχομένως να είναι πιο αποδοτικοί για τον έλεγχο ικανοποίησης των κανόνων, όπως ο αλγόριθμός *RETE* [96]. Ο *RETE* είναι ο αλγόριθμος που χρησιμοποιείται σε γνώστες γλώσσες κατασκευής έμπειρων συστημάτων, όπως η *CLIPS* και η *JESS*. Έχει αποδειχθεί ότι

ο *RETE* είναι πιο αποδοτικός αλγόριθμος για το πρόβλημα ταιριάσματος, πολλών προτύπων σε πολλά αντικείμενα, από οποιονδήποτε αλγόριθμο που υλοποιείται με *if – else if* προτάσεις.

Να θυμίσουμε ότι η υπηρεσία απαιτεί έναν αλγόριθμο, που ταιριάζει ένα πρότυπο με πολλά αντικείμενα, καθώς κάθε φορά ελέγχεται ένα γεγονός με όλους τους κανόνες. Επιπλέον τα αντικείμενα (συνθήκες των κανόνων), που ελέγχονται με βάση ένα γεγονός είναι περιορισμένα σε αριθμό. Συνεπώς δεν είναι βέβαιο ότι ο *RETE* θα βελτιώσει σημαντικά τη χρονική απόκριση του έμπειρου συστήματος. Θα ήταν ενδιαφέρον να ενσωματώσουμε τον *RETE* στην υπηρεσία και να δούμε, αν υπάρχει σημαντική χρονική βελτίωση στη διαδικασία ελέγχου των κανόνων. Βέβαια θα πρέπει να δούμε αν η υλοποίηση του *RETE* απαιτεί επιπλέον δομές δεδομένων, που αυξάνουν την απαιτούμενη μνήμη στο σωρό και αν τελικά η απαιτούμενη μνήμη επιτρέπει την υλοποίηση του αλγορίθμου για κινητές συσκευές.

Πιστεύουμε ότι, όσο οι πόροι των σύγχρονων κινητών συσκευών θα αυξάνονται και το λογισμικό για κινητές συσκευές θα σταθεροποιείται με κοινές προδιαγραφές, τα πλαίσια ανάπτυξης κινητών εφαρμογών θα μετακινηθούν εξ ολοκλήρου πάνω στις κινητές συσκευές. Οι υπηρεσίες συλλογής, επεξεργασίας, αποθήκευσης και διανομής το *context* αυτήν τη στιγμή δεν μπορούν υλοποιηθούν σε ένα κοινό πλαίσιο για κινητές συσκευές. Στο προσεχές μέλλον όμως είναι βέβαιο ότι κάτι τέτοιο θα είναι εφικτό.

Πιστεύουμε ότι η έρευνα πρέπει να κινηθεί προς την κατεύθυνση της ενοποίησης των *CA* υποδομών, δημιουργώντας διάχυτα δίκτυα αισθητήρων, συσκευών και υπηρεσιών. Για τη δημιουργία τέτοιων δικτύων απαιτούνται κοινά μοντέλα περιγραφής και αναπαράστασης της πληροφορίας. Τα *CA* συστήματα πρέπει να υιοθετήσουν ένα κοινό πρότυπο μοντελοποίησης του *context*. Οι τεχνολογίες του *Semantic Web* (*OWL, RDF, DAML+OIL*) έχουν ήδη χρησιμοποιηθεί αρκετά προς αυτήν την κατεύθυνση και φαίνεται ότι μπορούν να βοηθήσουν, καθώς παρέχουν έτοιμα τμήματα λογισμικού και διεπαφές για την ενσωμάτωσή τους σε κάθε σύστημα.

Για την εύρεση *context* υπηρεσιών και κατάλληλων προμηθευτών φαίνεται ότι, η χρήση της τεχνολογίας των Υπηρεσιών Διαδικτύου είναι κατάλληλη. Το *WSDL* που βασίζεται στην *XML* μπορεί να χρησιμοποιηθεί για την περιγραφή μιας υπηρεσίας και την

επικοινωνία με αυτήν, ενώ το *UDDI* είναι κατάλληλο για την εύρεση και την εγγραφή σε *context* υπηρεσίες.

Πρωταρχικός μας στόχος είναι να μεταφέρουμε το λογισμικό που υλοποιήσαμε πάνω σε μία κινητή συσκευή και να ελέγξουμε και εκεί τη λειτουργικότητα του. Θα ήταν ενδιαφέρον να υλοποιήσουμε μία υποδομή προμηθευτών (π.χ. *GPS*) και κάποιες εφαρμογές που να χρησιμοποιούν την παραγόμενη πληροφορία, ώστε να παρακολουθήσουμε πιο σύνθετα σενάρια λειτουργίας.

Μελλοντικά σκοπεύουμε να επεκτείνουμε τη λειτουργικότητα του ενδιάμεσου λογισμικού που υλοποιήσαμε περιλαμβάνοντας υπηρεσίες για εύρεση προμηθευτών και υπηρεσιών διαδικτύου. Εφόσον η πλατφόρμα *J2ME*, προσφάτως συμπεριέλαβε το *Web Services API 1.0*, αυτό είναι εφικτό να γίνει χρησιμοποιώντας τις τεχνολογίες των Υπηρεσιών Διαδικτύου.

ΑΝΑΦΟΡΕΣ

[1] Mark Weiser, The Computer for the Twenty-First Century, Scientific American, (1991) 94-104.

[2] Ian Sommerville, Software Engineering, Addison-Wesley, 2001.

[3] G.Rey and J.Coutaz, The Contextor Infrastructure for Context-Aware Computing, Component-oriented Approaches to Context-aware Computing held in conjunction with ECOOP'04, 2004.

[4] A.K.Dey and G.D.Abowd. Towards a better understanding of context and context-awareness. 1999. College of Computing, Georgia Institute of Technology. Ref Type: Report

[5] B.N.Schilit, N.I.Adams, and R.Want, Context-Aware Computing Applications, Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, 1994, pp. 85-90.

[6] G.Chen and D.Kotz. A survey of context-aware mobile computing research. Tech. Rep. TR2000-381. 2000. Dartmouth University. Ref Type: Report

[7] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy, Modeling Context Information in Pervasive Compiting Systems, 1st International Conference on Pervasive Computing (Pervasive 2002), Springer Verlag, 2002, pp. 167-180.

[8] J.Pascoe, N.S.Ryan, and R.D.Morse, Issues in Developing Context-Aware Computing, International Symposium on Handheld and Ubiquitous Computing, Springer-Verlag, Karlsruhe, Germany, 1999, pp. 208-221.

[9] A. Huang, B. Ling, H. Ponnekanti, and A. Fox, Pervasive computing: What is it good for?, ACM International Workshop on Data Engineering for Wireless and Mobile Access, ACM Press, 1999, pp. 84-91.

[10] Jason Pascoe, Adding generic contextual capabilities to wearable computers. Second International Symposium on Wearable Computers, IEEE Computer Society Press, 1998.

[11] P.Gray and D.Salber, Modelling and Using Sensed Context Information in the Design of Interactive Applications, 8th IFIP Conference on Engineering for Human Computer Interaction (EHCI), 2001, pp. 317-335.

[12] Anand Ranganathan and Roy H.Campbell, A Middleware for Context-Aware Agents in Ubiquitous Computing Environments, ACM/IFIP/USENIX International Middleware Conference, 2003.

[13] Bob Kummerfeld, Aaron Quigley, Chris Johnson, and Rene Hexel, Merino:Towards an intelligent environment architecture for multi-granularity context description, Workshop on User Modeling for Ubiquitous Computing (UM 2003), 2003.

[14] Florian Michaheles, Michael Samulowitz, and Bernt Schiele, Detecting context in distributed sensor networks by using smart context-aware packets, ARCS, 2002.

[15] P.B.Gibbons, Karp, Y.Ke, S.Nath, and S.Seshan, IrisNet: An Architecture for a Worldwide Sensor Web, IEEE Pervasive Computing Mobile and Ubiquitous Computing, Special Issue Sensor & Actuatirs Networks, 2 (2003) 22-33.

[16] A.Zarras, A Comparison Framework for Middleware Infrastructures, Journal of Object Technology, 3 (2004) 103-123.

[17] W.Emmerich, Software Engineering and Middleware: A Roadmap., 22th Int. Conference on Software Engineering (ICSE2000), ACM Press, 2000, pp. 60-65.

[18] T.Mowbray and R.Zahavi, The Essential CORBA: Systems Integration Using Distributed Objects, Wiley, 1995.

[19] Licia Capra, Gordon Blair, Cecilia Mascolo, Wolfgang Emmerich, and Paul Grace, Exploiting Reflection in Mobile Computing Middleware, ACM SIGMOBILE Mobile Computing and Communications Review, 6 (2002) 34-44.

[20] P.Maes, Concepts and Experiments in Computational Reflection, ACM Conference in Object-Oriented Programming Systems, Languages and Applications (OOPSLA), ACM Press, 1987, pp. 147-155.

[21] G.S.Blair, G.Coulson, P.Robin, and M.Papathomas, An Architecture for Next Generation Middleware, Middleware '98, Spriger Verlag, 1998, pp. 191-206.

[22] M.Roman, F.Kon, and R.H.Campell, Design and Implementation of Runtime Reflection in Communication Middleware: the dynamicTao Case., Workshop on Middleware, ICDCS '99, 1999.

[23] T.Ledoux, OpenCorba: a Reflective Open Broker, Reflection '99, Springer, 1999.

[24] F.Kon, F.Costa, G.Blair, and R.H.Campbel, The case for reflective middleware, Communications of the ACM, 45 (2002) 33-38.

[25] L.Capra, Wolfgang Emmerich, and Cecilia Mascolo, CARISMA: Context - Aware Reflective Middleware System for Mobile Applications, IEEE Transactions on Software Engineering, 29 (2003) 929-945.

[26] N.Parlavantzas, G.Coulson, M.Clarke, and G.Blair, Towards a reflective component based middleware architecture, Workshop on Reflection and Metalevel Architectures, 2000.

[27] Manuel Roma'n, Christopher K.Hess, Renato Cerqueira, Anand Ranganathan, Roy H.Campbell, and Klara Nahrstedt, Gaia: A Middleware Infrastructure to Enable Active Spaces, IEEE Pervasive Computing, (2002) 74-83.

[28] R.Want, A.Hopper, V.Falcao, and J.Gibbons, The Active Badge Location System., ACM Transactions on Information Systems, 10 (1992) 91-102.

[29] R.Want and et.al., An Overview of the PARCTAB Ubiquitous Computing Environment, IEEE Personal Communications, 2 (1995) 28-43.

[30] P.Brown, J.D.Bovey, and X.Chen, Context-Aware Applications: from the Laboratory to the Marketplace., IEEE Personal Communications, 4 (1997) 58-64.

[31] M.Weiser, Some Computer Science Issues in Ubiquitous Computing, Communications of the ACM, 36 (1993) 74-84.

[32] Daniel Salber, Anind K.Dey, and Gregory D.Abowd, The context toolkit: Aiding the development of context-enabled applications. CHI 99, ACM Press, 1999, pp. 434-441.

[33] A.K.Dey, D.Salber, G.D.Abowd, and M.Futakawa, The Conference Assistant: Combining context-awareness with wearable computing, 3rd International Symposium on Wearable Computers, 1999, pp. 21-28.

[34] G.W.Fitzmaurice, Situated Information Spaces and Spatialy Aware Palmtop Computers, CACM, 36 (1993) 39-48.

[35] S.Long and et.al., Rapid Prototyping of Mobile Context-aware Applications: The Cyberguide Case Study., MobiCom'96, 1996.

[36] N.Davies, K.Mitchell, K.Cheverest, and G.Blair, Developing a Context Sensitive Tourist Guide, First Workshop on Human Computer Interaction with Mobile Devices, 1998.

[37] Jie Yang, Weiyi Yang, M.Denecke, and A.Waibel, Smart sight: a tourist assistant system, 3rd International Symposium on Wearable Computers, 1999, pp. 73-78.

[38] M.Lamming and M.Flynn, Forget-me-not: Intimate Computing in Support of Human Memory, International Symposium on Next Generation Human Interfaces, 1994, pp. 125-128.

[39] B.J.Rhodes, The wearable remembrance agent: a system for augmented memory, 1st International Symposium on Wearable Computers, 1997, pp. 123-128.

[40] J.Healey and R.W.Picard, Startlecam: A Cybernetic Wearable Camera, 2nd International Symposium on Wearable Computers, 1998, pp. 42-49.

[41] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde, Advanced interaction in context, HUC'99, Springer Verlag, 1999, pp. 89-101.

[42] Hao Yan and Ted Selker, Context-aware office assistant., 2000 International Conference on Intelligent User Interfaces , ACM Press, 2000, pp. 276-279.

[43] J.Hong and J.Landay, An Infrastructure Approach to Context-Aware Computing., In the special issue on context aware computing of the Human Computer Interaction Journal, 16 (2001) 287-303.

[44] H.Ailisto and et.al., Structuring Context Aware Applications: Five-Layer Model and Example Case., UbicompWorkshop on Concepts and Models for Ubiquitous Computing, 2002.

[45] William Noah Schilit. A System Architecture for Context-Aware Mobile Computing. 1995. Columbia University. Ref Type: Thesis/Dissertation

[46] B.N.Schilit, Marvin M.Theimer, and Brent B.Welch, Customizing mobile applications, Mobile & Location-Independent Computing Symposium, USENIX Association, 1993, pp. 129-138.

[47] Richard Hull, Philip Neaves, and JamesBedford-Roberts, Towards situated computing, First International Symposium onWearable Computers, IEEE Computer Society Press, 1997.

[48] J.Pascoe, N.S.Ryan, and D.R.Morse, Human Computer Giraffe Interaction - HCI in the Field, Workshop on Human Computer Interaction with Mobile Devices, 1998.

[49] J.Pascoe, The Stick-e Note Architecture: Extending the Interface Beyond the User, International Conference on Intelligent User Interfaces, ACM, 1997, pp. 261-264.

[50] T.Bray, J.Paoli, and C.M.Sperberg-McQueen, Extensible Murkup Language. Recomendation http://www.w3.org/TR/1998/REC-xml-19980210, World Wide Web Consortium, 1998.

[51] M.Przybilski, P.Nurmi, and P.Flore'en, A Framework for Context Reasoning Systems, IASTED International Conference on Software Engineering (SE2005), 2005.

[52] Terry Winograd, Architectures for Context, Human Computer Interaction, 16 (2001) 2-4.

[53] Andrew Huang, Benjamin C.Ling, Shankar Ponnekanti, and Armando Fox, Pervasive computing: What is it good for?, ACM International Workshop on Data Engineering for Wireless and Mobile Access, ACM Press, 1999, pp. 84-91.

[54] M.Benerecetti, P.Bouquet, and M.Bonifacio, Distributed context-aware systems, Human Computer Interaction, 16 (2001).

[55] Daniela Petrelli, Elena Not, Massimo Zancanaro, Carlo Strapparava, and Oliviero Stock, Modelling and Adapting to Context, Personal and Ubiquitous Computing, 5 (2001) 20-24.

[56] H.Chen, S.Tolia, C.Sayers, T.Finin, and A.Joshi, Creating Context Aware Software Agents., First GSFC/JPL Workshop on Radical Agent Concepts, 2002.

[57] O.Lassila and R.R.Swick, Resource Description Framework (RDF) Model and Syntax Specification, W3C, 1999.

[58] M.Samulowitz, F.Michahelles, and C.Linnhoff-Popien, Adaptive Interaction for Enabling Pervasive Services, 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE01), 2001.

[59] Florian Michahelles and Michael Samulowitz, Smart CAPs for Smart Its - Context Detection for Mobile Users, Personal and Ubiquitous Computing, 6 (2002) 269-275.

[60] J.I.Hong, The Context Fabric: An Infrastructure for Context-Aware Computing., Doctoral Consortium, Human Factors in Computing Systems: CHI 2002, 2002.

[61] G.Chen and D.Kotz, Solar: An Open Platform for Context -Aware Mobile Applications, 1st International Conference on Pervasive Computing, 2002, pp. 41-47.

[62] Guanling Chen and David Kotz, Context aggregation and dissemination in ubiquitous computing systems, Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002), 2002, pp. 105-114.

[63] N.Sadeh, E.Chan, Y.Shimazaki, and Van, MyCampus: An Agent-Based Environment for Context-Aware Mobile Services, AAMAS02 Workshop onUbiquitous Agents on Embedded, Wearable, and Mobile Devices, 2002.

[64] John Keeny and Vinny Cahill, Chisel: A Policy-Driven, Context-Aware Dynamic Adaptation Framework., Fourth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), 2003, pp. 3-14.

[65] R.Glassey, G.Stevenson, M.Richmond, F.Wang, P.Nixon, S.Terzis, and I.Ferguson, Towards a middleware for generalised context management, 1st MPAC' 03, 2003.

[66] G.Judd and P.Steenkiste, Providing Contextual Information to Pervasive Computing Applications., IEEE International Conference on Pervasive Computing, 2003.

[67] E.Eliassen, A.Andersen, G.S.Blair, F.Costa, G.Coulson, V.Goebel, O.Hansen, T.Kristensen, T.Plagemann, H.O.Rafaelsen, K.B.Saikoski, and W.Yu, Next Generation Middleware: Requirements, Architecture and Prototypes, IEEE Workshop on Future Trends in Distributed Computing Systems, IEEE Computer Society Press, 1999, pp. 60-65.

[68] L.Capra, W.Emmerich, and C.Mascolo, Reflective Middleware Solutions for Context-Aware Applications, Third International Conference on Metalevel Architectures and Seperation of Crosscutting Concerns, 2001, pp. 126-133.

[69] Panu Korpipaa, Jani Mantyjarvi, Juha Kela, Heikki Keranen, and Esko-Juhani Malm, Managing Context Information in Mobile Devices, IEEE Pervasive, 2 (2003) 42-51.

[70] J.Altmann, G.Leonhartsberger, M.Pichler, W.Schwinger, Th.Hofer, and W.Retschitzegger, Context-Awareness on Mobile Devices - the Hydrogen Approach, 36th Hawaii International Conference on System Sciences (HICSS-36), Minitrack on Mobile Distributed Information Systems, 2003, p. 292ff.

[71] Harry Chen, Tim Finin, and Anupam Joshi, Using OWL in a Pervasive Computing Broker, Workshop on Ontologies in Agent Systems, AAMAS 2003.

[72] Harry Chen, Semantic Web in a Pervasive Context-Aware Architecture, Artificial Intelligence in Mobile System 2003.

[73] Harry Chen, Tim Finin, and Anupam Joshi, An Intelligent Broker for Context-Aware Systems, Adjunct Proceedings of Ubicomp 2003, pp. 183-184.

[74] Harry Chen. An Intelligent Broker Architecture for Pervasive Context-Aware Systems. 2004. University of Maryland, Baltimore County. Ref Type: Thesis/Dissertation

[75] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang, A Middleware for Building Context-Aware Mobile Services. IEEE Vehicular Technology, 2004.

[76] Xiaohang Wang, Jin Song Dong, Chung Yau Chin, Sanka Ravipriya Hettiarachchi, and Daqing Zhang., Semantic Space: An Infrastructure for Smart Spaces., IEEE Pervasive Computing, 3 (2004) 32-39.

[77] Patrick Fahy and Siobhan Clarke, CASS - Middleware for Mobile Context-Aware Applications, MobiSys 2004, 2004.

[78] Gregory Biegel and Vinny Cahill, A Framework for Developing Mobile, Context-aware Applications, 2nd IEEE Conference on Pervasive computing and Communications, Percom, 2004.

[79] Ren Meier and Vinny Cahill, Exploiting Proximity in Event - Based Middleware for Collaborative Mobile Applications, 4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'03), Springer-Verlag Heidelberg, Germany, 2003.

[80] Patricia Dockhorn Costa, Luvs Ferreira Pires, Marten van Sinderen, and Josı Gonηalves Pereira Filho, Towards a Services Platform for Mobile Context-Aware Applications, First International Workshop on Ubiquitous Computing (IWUC 2004), 2004.

[81] K.Henricksen and J.Indulska, A software Engineering Framework for Context-aware Pervasive Computing., Pervasive Computing and Communications, IEEE Computer Society, 2004, pp. 77-86.

[82] M.Beigl, Memoclip:A location based rememberance appliance, 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2000), 2000, pp. 230-234.

[83] M.Beigl, H.W.Gellersen, and A.Schmidt, MediaCups: Experience with design and use of computer-augmented everyday objects, Computer Networks, 35 (2001) 401-409.

[84] T.Strang and C.Linnhoff-Popien, A Context Modeling Survey, Workshop on Advanced Context Modelling, Reasoning and Management associated with the Sixth International Conference on Ubiquitous Computing (UbiComp 2004), 2004.

[85] J.Coutaz and G.Rey, Foundations for a Theory of Contextors, Computer Aided Design of User Interfaces, Springer Verlag, 2002.

[86] Paul Preko and Mark Burnett, Activities context and ubiquitous computing, Computer Communications, 26 (2003) 1168-1176.

[87] James L.Crowley and Patrick Reignier, An Architecture for Context Aware Observation of Human Activity, Workshop on Computer Vision System Control Architectures (VSCA 2003), 2003.

[88] R.Wang, M.P.Reddy, and H.Kon, Towards quality data : An Attribute-based approach, Decision Support Systems, 13 (1995) 349-372.

[89] P.Castro and R.Muntz, Managing Context for Smart Spaces, IEEE Personal Communications, 7 (2000) 44-46.

[90] N.W.Paton and O.Diaz, Active Databases Survey, ACM Computing Surveys, 31 (1999) 63-103.

[91] Diego Lσpez de Ipiρa and Eleftheria Katsiri, An ECA Rule-Matching Service for Simpler Development of Reactive Applications, Published as a supplement to the Proceedings of Middleware 2001 at IEEE Distributed Systems Online, 2001.

[92] A.Ranganathan and R.Campbell, An Infrastructure for Context-Awareness based on First Order Logic., Personal and Ubiquitous Computer Journal, 7 (2003) 353-364.

[93] Tasos A.Kontogiorgis, Dimitrios I.Fotiadis, and Apostolos Zarras, A Middleware Service for Managing Time and Quality Dependent Context, IADIS International Conference WWW/Internet 2004, 2004, pp. 503-510.

[94] Herbert Enderton, A Mathematical Introduction to Logic, Academic Press, 1972.

[95] K.Henricksen, S.Livingstone, and J.Indulska, Towards a hybrid approach to context modelling, reasoning, and interoperation, First International Workshop on Advanced Context Modelling, Reasoning And Management, UbiComp'2004, 2004.

[96] C.L.Forgy, Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, Artificial Intelligence, (1982) 17-37.

LINKS

[L1] http://www.omg.org /technology/documents/formal/corba_iiop.htm

[L2] http://java.sun.com/j2ee/

[L3] http://msdn.microsoft.com/library/default.asp?url=/library/enus/cossdk/htm/pgservices_events_5x4j.asp

[L4] http://www.cc.gatech.edu/fce/

[L5] http://tea.startlab.net/

[L6] http://www.w3c.org/TR/owl-guide/

[L7] http://www.freeband.nl/projecten/wasp

[L8] http://java.sun.com/j2me

[L9] http://www.forum.nokia.com/main/0,6566,034-2,00.html

[L10] http://java.sun.com/products/sjwtoolkit/download-2_2.html

[L11] http://developers.sun.com/techtopics/mobility/midp/articles/jtwi/