ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

«Διαμόρφωση προτύπου για ανάπτυξη προσαρμοστικών εφαρμογών ηλεκτρονικού εμπορίου»

Διπλωματική εργασία

Βλάχου Χριστίνα-Σαλτάρη Γεωργία

Επιβλέπων καθηγητής: Βασιλάκης Κωνσταντίνος

Τρίπολη: 2012

Ευχαριστίες

Θα θέλαμε να ευχαριστήσουμε θερμά τον καθηγητή κ. Βασιλάκη Κωνσταντίνο κυρίως για την εμπιστοσύνη που μας έδειξε, και την υπομονή που έκανε κατά τη διάρκεια υλοποίησης της διπλωματικής εργασίας. Όπως επίσης και για την πολύτιμη βοήθεια και καθοδήγηση του, για την επίλυση διαφόρων θεμάτων. Θα θέλαμε επίσης να απευθύνουμε τις ευχαριστίες μας στις οικογένειες μας που μας στήριξαν με διάφορους τρόπους, φροντίζοντας για την καλύτερη δυνατή επίδοση μας.

Περίληψη

Η παρούσα διπλωματική εργασία ασχολείται με τη διαμόρφωση ενός προτύπου για την ανάπτυξη προσαρμοστικών εφαρμογών ηλεκτρονικού εμπορίου. Το πρότυπο ενσωματώνει γενικές περιγραφές κλάσεων όπου κωδικοποιείται η εκάστοτε επιχειρηματική λογική, καθώς και δομές της Aspect Java με τις οποίες αναπτύσσονται ξεχωριστά οι δομές προσαρμογής και ενσωματώνονται τελικά στον κώδικα της εφαρμογής.
Τα αρχικά πρότυπα βασίστηκαν σε τεχνολογίες πρακτόρων λογισμικού (agents). Η σύγχρονη όμως μορφή του ηλεκτρονικού εμπορίου ήρθε να καθιερώσει νέα πρότυπα αλλά πάνω σε μία κοινή πλατφόρμα, το διαδίκτυο. Ο χρήστης παραμένει ο πυρήνας αυτών των προτύπων αλλά επιπλέον ενσωματώνονται πληροφορίες που προέρχονται από το περιβάλλον του δημιουργώντας ένα σύνολο από περιβάλλουσες συνθήκες (context). Προκειμένου να αξιοποιηθεί όλη αυτήν την πληροφορία (προφίλ και προτιμήσεις χρήστη, κίνηση/τοποθεσία χρήστη, χρονική στιγμή, διαστάσεις οθόνης κ.λπ.), συνθέτουμε μία αρχιτεκτονική προσαρμοστικότητας με βάση την περιβάλλουσα κατάσταση, η οποία μπορεί να υποστηρίξει οποιαδήποτε εφαρμογή κινητού ηλεκτρονικού εμπορίου.
Η προσαρμοστικότητα (adaptivity) βάσει όλων αυτών των στοιχείων βοηθάει στην αύξηση της παρεχόμενης ποιότητας υπηρεσιών προς τους πελάτες και κάνει κινητό ηλεκτρονικό εμπόριο πιο ελκυστικό και ανταγωνιστικό. Στην παρούσα πτυχιακή εργασία, η προσαρμοστικότητα αυτή να αφορά το περιεχόμενο (*τι* θέλει να δει ο χρήστης), τη λειτουργικότητα (*ποιές υπηρεσίες θέλει να έχει ο χρήστης*) όπως και τη διεπαφή (*πώς θα παρουσιαστεί στο χρήστη η πληροφορία*).
Η υλοποίηση του προτεινόμενου προτύπου έγινε με τη βοήθεια της του προσανατολισμένου σε απόψεις υπόδειγμα προγραμματισμού (aspect-oriented paradigm - AOP), αλλά και των γενικών αρχών της τεχνολογίας λογισμικού. Ως γενική αρχιτεκτονική, χρησιμοποιήθηκε η αρχιτεκτονική διαχείρισης της περιβάλλουσας κατάστασης που προτείνεται από τους Μπένου και Βασιλάκη.
Για την επιβεβαίωση της λειτουργικότητας της προτεινόμενης προσέγγισης, έχει αναπτυχθεί μία προσαρμοστική εφαρμογή για αναζήτηση σημείων ενδιαφέροντος, η οποία λαμβάνει υπ’ όψιν την τοποθεσία του χρήστη, την κατάσταση κινητικότητάς του (πεζός ή εποχούμενος), τη διαθεσιμότητα των σημείων ενδιαφέροντος (ανοικτά/κλειστά) και τις διαστάσεις της οθόνης της συσκευής του χρήστη. Η εφαρμογή μπορεί να επεκταθεί ενσωματώνοντας πρόσθετες διαστάσεις προσαρμοστικότητας.

Λέξεις κλειδιά: κινητό ηλεκτρονικό εμπόριο, περιβάλλουσα πληροφορία, προσαρμοστικότητα, αρχιτεκτονική συστήματος, προσανατολισμένος σε απόψεις προγραμματισμός

Abstract

This thesis deals with the formulation of a model for developing adaptive m-commerce applications. The model incorporates general descriptions of the classes where the relevant business logic is encoded and separately developed and maintained Aspect Java structures, which provide the desired adaptation.

The initial adaptation models were based on agent technologies. Nowadays, however, m-commerce has come to establish new models using a common platform, the internet. The user remains the core of these models but, additionally, information from her environment are incorporated, leading to a (potentially) broad set of *context information*. In order to exploit all this information (user profile and preferences, movement / location of user, time, screen size etc), we formulate an adaptation architecture that takes it into account; this architecture can support any m-commerce application.

Offering adaptivity based on all these elements elevates the quality of services provided to users and makes m-commerce more attractive and competitive. In the context of this thesis, adaptability applies to the content (what the users want to see), functionality (which services the users want to have) and the interface (how is information presented to the user).

The proposed model was implemented using the AOP (aspect-oriented programming) paradigm and software engineering principles. The general architecture proposed by Benou and Vassilakis was adopted.

As a proof-of-concept for the functionality of the proposed approach, an adaptive application for searching points-of-interest was developed. The application takes into account the user location, her mobility status (pedestrian or driving), the point-of-interest current state (open/closed), and the size of the user's screen. The application may be extended to incorporate any adaptivity dimension.

Keywords: m-commerce, context information, adaptivity, system architecture, aspect-oriented programming

Πίνακας Περιεχομένων

Ευρετήριο Εικόνων

Ευρετήριο πινάκων

1. Εισαγωγή

Η ανάδειξη και συνεχώς αυξανόμενη διάδοση του ηλεκτρονικού εμπορίου κυρίως μέσω του μοντέλου των ηλεκτρονικών πωλήσεων (π.χ. μέσω καταλόγων, e-shops) καθιστά αναγκαία την υλοποίηση ευέλικτων και προσαρμοστικών συστημάτων και αρχιτεκτονικών ηλεκτρονικού εμπορίου. Τέτοιου είδους συστήματα είναι αναγκαία, ειδικά για μικρές και μεσαίες επιχειρήσεις, που στοχεύουν στην υλοποίηση ενός κοινού προτύπου για εφαρμογές ηλεκτρονικού εμπορίου (π.χ. e-shop sites, business to business portals).

Επιπλέον, στόχος τους είναι οι εφαρμογές αυτές να προσαρμόζουν αυτόματα τις διεπαφές, το περιεχόμενο και τις λειτουργίες τους προς τους τελικούς χρήστες. Για παράδειγμα, ο τρόπος παρουσίασης των προϊόντων σε καταλόγους να προσαρμόζεται με βάση διάφορους περιορισμούς και δεδομένα που ισχύουν τη δεδομένη χρονική στιγμή (π.χ. διαστάσεις οθόνης υπολογιστή ή κινητού τηλεφώνου, ταχύτητα σύνδεσης χρήστη, προτιμήσεις για συγκεκριμένες κατηγορίες προϊόντων, γεωγραφική τοποθεσία του χρήστη ή θέση στο δίκτυο της κινητής τηλεφωνίας).

Στη βιβλιογραφία αναφέρονται διάφορα μοντέλα για την υλοποίηση προσαρμοστικών προτύπων, μερικά από τα οποία είναι ειδικά προσαρμοσμένα στο πεδίο του ηλεκτρονικού εμπορίου. Τα πρώτα μοντέλα βασίστηκαν σε τεχνολογίες πρακτόρων λογισμικού (agents) [1]. Απώτερος σκοπός τους ήταν η εκμάθηση των συνηθειών των καταναλωτών και η προσαρμογή του περιεχομένου των αγορών σε αυτές. Καθώς άλλαζε δυναμικά η συμπεριφορά του χρήστη και οι προτιμήσεις του, θα έπρεπε ο πράκτορας λογισμικού του καταστήματος να προσαρμόζει το ίδιο δυναμικά την πληροφορία που εμφανίζεται στον κατάλογο των προϊόντων, για παράδειγμα να υπολογίζει αυτόματα την έκπτωση ή την τιμή [2]. Το πρόβλημα με την

τεχνολογία των πρακτόρων λογισμικού ήταν κυρίως η ετερογένεια μεταξύ των διαφόρων προτύπων. Δημιουργήθηκαν διάφορα πρότυπα, όπου το καθένα απαιτούσε την εφαρμογή ενός ξεχωριστού πρωτοκόλλου προσαρμοστικότητας. Επιπλέον ο ρόλος των προτύπων που βασίζονται στους πράκτορες λογισμικού agent δεν ήταν ξεκάθαρος, πως έπρεπε να προσαρμοστούν σε σύγχρονες τεχνολογίες με εκτέλεση από την πλευρά του εξυπηρέτη. Επιπλέον η εφαρμογή αυτών των πρωτοκόλλων προσαρμοστικότητας (όπου αναφέρονται στη χρήση πρακτόρων λογισμικού) δεν μπορούσαν να ταιριάξουν με τις σύγχρονες τεχνολογίες με εκτέλεση από την πλευρά του εξυπηρέτη (server-side) για την ανάπτυξη εφαρμογών ηλεκτρονικού εμπορίου [3].

Επιπλέον, η σύγχρονη μορφή του ηλεκτρονικού εμπορίου έχει καθιερώσει νέα επιχειρησιακά μοντέλα στα πλαίσια της παγκοσμιοποίησης, αλλά πάνω σε μία κοινή πλατφόρμα που είναι το διαδίκτυο (internet). Το διαδίκτυο φέρνει πλέον πιο κοντά τον καταναλωτή στον πάροχο των προϊόντων ή υπηρεσιών και μέσα από οποιαδήποτε συσκευή (υπολογιστής, κινητό τηλέφωνο, ταμπλέτες κ.λ.π.) και μέσα από οποιοδήποτε μέσω επικοινωνίας [τοπικά δίκτυα, ενσύρματα/ασύρματα ευρυζωνικά δίκτυα (broadband), WiFi/3G/4G κ.ο.κ.]. Για το λόγο αυτό η παραδοσιακή μορφή του ηλεκτρονικού εμπορίου έχει εξελιχθεί σε μία νέα γενιά εμπορίου μέσω έξυπνων τηλεφώνων (smart phones) ή κινητών τηλεφώνων που καλείται *κινητό ηλεκτρονικό εμπόριο* (m-commerce). Ο χρήστης παραμένει ο πυρήνας των σύγχρονων επιχειρησιακών μοντέλων (τα οποία διατηρούν στοιχεία όπως προφίλ, ιστορικό αγορών, προτιμήσεις) αλλά επιπλέον λαμβάνονται υπ’ όψιν οι διαστάσεις της σύγχρονης εποχής όπως η δραστηριότητα του χρήστη (π.χ. εργασία, διασκέδαση, ταξίδι κ.λ.π.), η τοποθεσία του (π.χ. στο γραφείο, στο σπίτι), κινητικότητα (mobility) και μέσο κινητικότητας (πεζός, με αυτοκίνητο, σε τρένο,

μετρό ή λεωφορείο), διαθέσιμη συσκευή και τα χαρακτηριστικά της [3].

Ο συνδυασμός όλων αυτών των διαστάσεων ορίζει ένα σύνολο από περιβάλλουσες συνθήκες (context) και αποτελεί τον πυρήνα του ορισμού προτύπων προσαρμοστικότητας και ανάπτυξης τέτοιων εφαρμογών. Ένα πρότυπο προσαρμοστικότητας το οποίο περιλαμβάνει στοιχεία της περιβάλλουσας πληροφορίας καλείται *πρότυπο με επίγνωση της περιβάλλουσας πληροφορίας*. Σκοπός των προτύπων με επίγνωση της περιβάλλουσας πληροφορίας είναι η ανάπτυξη συστημάτων/αρχιτεκτονικών που προσαρμόζουν τις εφαρμογές/υπηρεσίες με σημαντική ευελιξία: οι παραπάνω διαστάσεις του *Ποιός, Πού, Πότε, Πώς* ορίζουν το πλαίσιο της προσαρμοστικότητας είτε εξετάζοντας μεμονωμένα κάθε διάσταση, είτε σε συνδυασμένη εφαρμογή τους επάνω στην εμφάνιση (presentation), περιεχόμενο (content), και λειτουργικότητα (operation) της εφαρμογής ηλεκτρονικού εμπορίου [3].

Στην παρούσα εργασία διερευνούμε τα διάφορα μοντέλα με επίγνωση της περιβάλλουσας πληροφορίας καθώς και σχετικές αρχιτεκτονικές που έχουν δοκιμαστεί στο πεδίο των εφαρμογών κινητού ηλεκτρονικού εμπορίου. Η αρχιτεκτονική βασίζεται στο προσανατολισμένο σε απόψεις υπόδειγμα προγραμματισμού (aspect-oriented paradigm - AOP) για εφαρμογές των οποίων τις λειτουργίες και περιεχόμενο ελέγχει κυρίως ένας εξυπηρετητής (server-side) αλλά αξιοποιεί συγχρόνως τεχνολογίες ανάπτυξης εφαρμογών ιστού (web applications) που επιτρέπουν ευελιξία στη διεπαφή του χρήστη (π.χ. Ajax). Η συνδυασμένη εφαρμογή αυτών των τεχνολογιών συνθέτει μία αρχιτεκτονική προσαρμοστικότητας με βάση την περιβάλλουσα κατάσταση σε οποιαδήποτε εφαρμογή κινητού ηλεκτρονικού εμπορίου [3].

Το υπόλοιπο της παρούσας πτυχιακής έχει διαρθρωθεί ως ακολούθως: Στο Κεφάλαιο 2 γίνεται επισκόπηση της βιβλιογραφίας στην εφαρμογή

μοντέλων με επίγνωση της περιβάλλουσας πληροφορίας στο χώρο του ηλεκτρονικού εμπορίου καθώς και τεχνολογιών που μπορούν να αξιοποιηθούν ως προς την υλοποίηση τέτοιων μοντέλων. Το Κεφάλαιο 3 παρουσιάζει την προτεινόμενη αρχιτεκτονική προσαρμοστικότητας αναλύοντας τις επί μέρους συνιστώσες και τεχνικές. Το Κεφάλαιο 4 αξιολογεί τη λύση που προτείνεται[3] σε σχέση με το αν επιτυγχάνει τα πλεονεκτήματα που αναφέρονται στην εισαγωγή, και συγκριτικά με άλλες λύσεις. Το Κεφάλαιο 5 ολοκληρώνει την εργασία με συμπεράσματα και μελλοντικές επεκτάσεις.

2. Επισκόπηση Βιβλιογραφίας

2.1. Το υπόδειγμα της υπολογιστικής με επίγνωση της περιβάλλουσας κατάστασης (Context-aware computing paradigm)

Οι τεχνικές που αξιοποιούν και προσαρμόζονται δυναμικά, με βάση περιβάλλουσα πληροφορία, εντάσσονται σε ένα γενικότερο πλαίσιο που ονομάζεται *υπολογιστική με επίγνωση της περιβάλλουσας κατάστασης* (context-aware computing) [4]. Η υπολογιστική με επίγνωση της περιβάλλουσας κατάστασης αποτελεί μία περιοχή της επιστήμης των υπολογιστών για φορητές συσκευές και δίκτυα (mobile computing) μέσα στο οποίο οι εφαρμογές ανακαλύπτουν και αξιοποιούν με τον καλύτερο δυνατό τρόπο την περιβάλλουσα πληροφορία (π.χ. τοποθεσία, ώρα της ημέρας, χρήστες και συσκευές εντός εμβέλειας και δραστηριότητες του χρήστη). Η συγκεκριμένη περιοχή έχει εγκαθιδρυθεί εδώ και μία δεκαετία περίπου και έχει πλέον εντάξει στο ενεργητικό της εκατοντάδες παραδείγματα εφαρμογών με επίγνωση της περιβάλλουσας κατάστασης – πολύ λίγα όμως εξελίχθηκαν σε μεγαλύτερες υποδομές που να χρησιμοποιούνται ευρέως από πολλούς χρήστες στην καθημερινότητα τους [4].

2.2. Κατηγορίες περιβάλλουσας κατάστασης

Πράγματι, οι άνθρωποι χρησιμοποιούν μία ευρεία γκάμα φορητών συσκευών με διαφορετικά λειτουργικά συστήματα, που τους δίνουν τη δυνατότητα να έχουν πρόσβαση σε προσωπικά ή επαγγελματικά στοιχεία σε οποιονδήποτε τόπο και σε οποιαδήποτε στιγμή. Κατά το πρότυπο της υπολογιστικής με επίγνωση της περιβάλλουσας κατάστασης ο όρος *περιβάλλουσα κατάσταση* αντικατοπτρίζει το συνδυασμό ορισμένων συνθηκών και παραμέτρων στο περιβάλλον,

που είτε καθορίζει τη συμπεριφορά της εφαρμογής ή μέσα στο οποίο (περιβάλλον) συμβαίνει ένα γεγονός (event) και ενδιαφέρει άμεσα το χρήστη. Για την περιγραφή ενός τέτοιου συνδυασμού συνθηκών έχει προταθεί στη βιβλιογραφία ο ορισμός των παρακάτω κατηγοριών [5]:

1) *Υπολογιστική κατάσταση (computing context)*, όπως η συνδεσιμότητα, το κόστος επικοινωνίας, διαθέσιμο bandwidth, διαθέσιμοι πόροι εντός εμβέλειας (π.χ. εκτυπωτές, οθόνες, σταθμοί εργασίας).
2) *Κατάσταση χρήστη (user context)*, όπως προφίλ, τοποθεσία, χρήστες σε εμβέλεια, κοινωνική κατάσταση.
3) *Κατάσταση φυσικού περιβάλλοντος (physical context)*, όπως φωτισμός, επίπεδα θορύβου, συνθήκες κίνησης και θερμοκρασία.
4) *Χρονική κατάσταση (time context),* όπως ώρα της ημέρας, εβδομάδα, μήνα και εποχή του χρόνου.
5) *Ιστορικό των παραπάνω καταστάσεων (context history),* η οποία περιλαμβάνει την καταγραφή και ανάκτηση δεδομένων για την υπολογιστική κατάσταση του χρήστη και του φυσικού περιβάλλοντος στο πέρασμα του χρόνου.

Επιπλέον, ο συνδυασμός των παραπάνω συνθηκών δίνει τη δυνατότητα για την εξαγωγή πρόσθετων κανόνων για μία ευρύτερη πληροφόρηση σχετικά με την κατάσταση του χρήστη. Για παράδειγμα, έχοντας πρόσβαση στην ατζέντα του χρήστη και γνωρίζοντας την τρέχουσα ώρα και την τρέχουσα τοποθεσία του χρήστη μπορούμε να συνάγουμε τη δραστηριότητα του (π.χ. βρίσκεται στο γραφείο, έχει συνάντηση, βρίσκεται σε αίθουσα αναμονής στο αεροδρόμιο).
Μία άλλη διάκριση των καταστάσεων είναι στο κατά πόσο επηρεάζουν τη συμπεριφορά της εφαρμογής, βάσει της οποίας διάκρισης κατατάσσονται σε ενεργές και παθητικές. Για παράδειγμα σε μία εφαρμογή Προώθησης Κλήσεων, η πληροφορία για την τοποθεσία του

χρήστη χρησιμοποιείται ενεργά στην εφαρμογή. Έστω ότι αναπτύσσουμε μια εφαρμογή όπου θέλουμε να προωθήσουμε την κλήση σε ένα από τα τηλέφωνα κάποιου ατόμου, ο οποίος έχει ένα τηλέφωνο στο γραφείο, ένα στο σπίτι και ένα κινητό, οπότε θα πρέπει να χρησιμοποιηθεί η τοποθεσία του χρήστη για να διαπιστώσουμε που βρίσκεται, ώστε να χρησιμοποιηθεί αυτή η πληροφορία ως μέρος του κανόνα για να αποφασίσουμε που θα δρομολογηθεί η κλήση. Π.χ. αν είναι αυτή τη στιγμή σε κίνηση, η κλήση θα δρομολογηθεί στο κινητό τηλέφωνο. Η πληροφορία αυτή για την τοποθεσία του χρήστη είναι μία δυναμική πληροφορία, δηλαδή συνέχεια αλλάζει και συνέχεια ενημερώνει την εφαρμογή ώστε να μπορέσουμε να την αξιοποιήσουμε.

Αντίθετα σε μία εφαρμογή Ενεργού Χάρτη, η τοποθεσία του χρήστη χρησιμοποιείται παθητικά [5]. Αυτό σημαίνει ότι δεν υπάρχει κάποιος κανόνας που να περιλαμβάνει τη θέση του χρήστη: απλά εντοπίζουμε τη θέση του χρήστη και την τοποθετούμε πάνω στο χάρτη. Με άλλα λόγια, ανά πάσα στιγμή μπορούμε να ζητήσουμε την τοποθεσία του χρήστη και στη συνέχεια να εμφανίσουμε το στίγμα πάνω στο χάρτη.

Συνοψίζοντας, η τοποθεσία του χρήστη στην εφαρμογή προώθησης κλήσης επηρεάζει τη συμπεριφορά της εφαρμογής, ενώ στην εφαρμογή του ενεργού χάρτη η τοποθεσία του χρήστη δεν επηρεάζει τη συμπεριφορά.

2.3 Υπολογισμός και ενημέρωση των καταστάσεων

2.3.1 Ανίχνευση τοποθεσίας

Η τρέχουσα τοποθεσία ενός αντικειμένου ή ενός ανθρώπου ορίζεται ανάλογα με τον τρόπο που ορίζεται η τοπολογία του συστήματος και των εργαλείων που χρησιμοποιούνται για τον προσδιορισμό ενός σημείου στο χώρο. Για παράδειγμα, ένα σημείο στο φυσικό χώρο ορίζεται από γεωγραφικές συντεταγμένες, οι οποίες αναφέρονται από

συστήματα ανίχνευσης συντεταγμένων (Global Positioning System-GPS), ενώ στη συνέχεια είναι δυνατόν να απεικονίζονται με γραφικό τρόπο οι τρέχουσες θέσεις των αντικειμένων ή να υπάρχουν ειδοποιήσεις (π.χ. με γραφικό ή ακουστικό τρόπο) για τη μετακίνηση ενός αντικειμένου ή ανθρώπου. Εναλλακτικά, στη διαδικασία μετακίνησης του συνδρομητή κινητής τηλεφωνίας από ένα κελί δικτύου κινητής τηλεφωνίας τρίτης γενιάς (3G) σε άλλο χρησιμοποιείται η μέθοδος τριγωνισμού (triangulation). Η μέθοδος αυτή λαμβάνει υπ' όψιν την απόσταση της συσκευής από 3 διαφορετικά κελιά και με βάση τις γωνίες που σχηματίζονται στο ιδεατό τρίγωνο υπολογίζεται το σημείο που βρίσκεται η συσκευή [6].

Σε εσωτερικούς χώρους, χρησιμοποιούνται σύγχρονης μορφής ανιχνευτές (π.χ. RFID, badge controllers, Bluetooth detection), Εναλλακτικά, κάμερες με συστήματα επεξεργασίας εικόνας (image processing) και ανίχνευσης κίνησης (motion detection) καταγράφουν την κίνηση μίας οντότητας σε ένα κτίριο, εργοστάσιο κ.λπ. και μεταφέρουν την εικόνα σε μία συσκευή καταγραφής ή σε απομακρυσμένες συσκευές αναπαραγωγής πολυμέσων (ακόμη και σε ένα κινητό τηλέφωνο) [4].

2.3.2 Υπολογισμός χρόνου και χρονοπρογραμματισμός άλλων καταστάσεων

Ο χρόνος είναι βασική διάσταση για τη χρονική ταξινόμηση των αλλαγών στις διάφορες καταστάσεις καθώς μπορεί να εφαρμοστεί κάποιου είδους προτεραιότητα. Αυτόματα μπαίνει μια χρονική ταξινόμηση επειδή υπάρχει αυτή η διάσταση του χρόνου. Ο χρόνος παίζει ρόλο στην προσαρμοστικότητα με την έννοια ότι μπορεί να αποφασιστεί η αλλαγή της συμπεριφοράς στην εφαρμογή.

Ένα παράδειγμα είναι η αποθήκευση των διαφορετικών τοποθεσιών στις οποίες μετακινείται μία οντότητα, συνοδευόμενες με το σχετικό χρονόσημο. Αυτό επιτρέπει την εξαγωγή επιπλέον κανόνων όπως [4]:

- Υπολογισμός της επόμενης εργασίας στο ημερολόγιο του χρήστη με βάση την ώρα της ημέρας, τρέχουσα ημέρα στην εβδομάδα ή μήνα κ.λπ.
- Υπολογισμός του επόμενου διαθέσιμου καταστήματος με βάση την ώρα της ημέρας τρέχουσα ημέρα στην εβδομάδα ή μήνα, κατάσταση (ανοικτό/κλειστό) κ.λπ.

Για τον υπολογισμό αυτών των αφηρημένων καταστάσεων απαιτείται η εφαρμογή μεθόδων τεχνητής νοημοσύνης [7]. Εφαρμόζουμε κάποιους κανόνες βάσει της *συλλογιστικής κατά περίπτωση* (case based reasoning) ώστε να βγάλουμε κάποιο αποτέλεσμα. Π.χ. κάποιος αυτή την στιγμή κοιμάται οπότε θα υπολογίσουμε αν η ώρα είναι 23:00 μ.μ. και το ημερολόγιό του είναι κενό, τότε μπορούμε να συμπεράνουμε ότι υπάρχει σημαντική πιθανότητα να κοιμάται.

Όλες οι περιπτώσεις-με βάση τις μεθόδους συλλογισμού έχουν από κοινού την ακόλουθη διαδικασία:

- Ανάκτηση παρόμοιων περιπτώσεων συγκρίνοντας την υπόθεση με τη βιβλιοθήκη των υποθέσεων του παρελθόντος.
- Επαναχρησιμοποίηση των περιπτώσεων που ανακτώνται από το παρελθόν ώστε να λυθεί το τρέχον πρόβλημα.
- Αναθεωρεί και προσαρμόζει την προτεινόμενη λύση, εάν χρειάζεται.
- Διατηρεί την τελική λύση στο πλαίσιο μιας νέας υπόθεσης.

Υπάρχει μια ποικιλία διαφορετικών μεθόδων για την οργάνωση, την ανάκτηση, την αξιοποίηση και τη γνώση από παλαιότερες περιπτώσεις. Η ανάκτηση μιας υπόθεσης αρχίζει με μια (ενδεχομένως μερική) περιγραφή του προβλήματος και τελειώνει όταν μια καλύτερη αντίστοιχη υπόθεση έχει βρεθεί. Οι δευτερεύουσες εργασίες περιλαμβάνουν:

- Προσδιορισμός μιας σειράς από σχετικές περιγραφές προβλημάτων.
- Ταίριασμα της υπόθεσης και της επιστροφής ενός συνόλου παρόμοιων περιπτώσεων (δεδομένου κάποιου ορίου ομοιότητας κάποιου είδους) και
- Επιλογή της καλύτερης περίπτωσης από το σύνολο των περιπτώσεων που επέστρεψε.

2.3.3 Άλλοι ανιχνευτές

Άλλες συσκευές ανίχνευσης μετρούν και ενημερώνουν για αλλαγές στο διαθέσιμο εύρος ζώνης (bandwidth) του δικτύου, προσανατολισμός της συσκευής, ανιχνευτές φωτός, θορύβου, θερμοκρασίας κ.ά.

Οι ανιχνευτές αυτοί είτε τοποθετούνται επάνω στη συσκευή του χρήστη (π.χ. υπολογιστή, κινητό τηλέφωνο) ή τοποθετούνται στο φυσικό χώρο ή στην υποδομή (π.χ. δίκτυο) και παρέχουν τα δεδομένα μέσω ενδιάμεσων εφαρμογών(middleware). Για παράδειγμα, κάποιος μπορεί να παρακολουθεί αλλαγές στη θερμοκρασία του διαμερίσματός του μέσω μίας υπηρεσίας (web service) που έχει πρόσβαση στο συγκεκριμένο ανιχνευτή [4].

2.4 Αρχιτεκτονικές με επίγνωση της περιβάλλουσας κατάστασης

2.4.1 Κεντρικοποιημένη αρχιτεκτονική

Σε αυτό το μοντέλο [8] υπάρχει ένας εξυπηρέτης για κάθε κατάσταση με σκοπό την παροχή ενημερώσεων προς τρίτους όταν υπάρχουν αλλαγές στη συγκεκριμένη κατηγορία (π.χ. τοποθεσία). Βασική συνιστώσα της αρχιτεκτονικής είναι το "context widget", μία συνιστώσα που επιτρέπει σε εφαρμογές να έχουν πρόσβαση σε πληροφορίες για μία συνθήκη (π.χ. έναν ανιχνευτή θερμοκρασίας που

να παρέχει ενημερώσεις για αλλαγές στη θερμοκρασία ενός δωματίου). Η κατάσταση του κάθε widget ορίζεται από τις τιμές ενός συνόλου παραμέτρων. Επιπλέον το κάθε widget υλοποιεί ένα σύνολο από συναρτήσεις callback που ενημερώνουν για οποιαδήποτε αλλαγή στις τιμές των προαναφερθεισών παραμέτρων.

Η συνιστώσα λαμβάνει δεδομένα σε πρωτογενή μορφή για τη συγκεκριμένη κατάσταση και τα μεταφέρει δυναμικά είτε σε μεταφραστές (interpreters) είτε σε εξυπηρέτες (servers) για τη δημιουργία νέων συναθροίσεων των δεδομένων (aggregation). Οι μεταφραστές και οι εξυπηρέτες υλοποιούν συγκεκριμένες προγραμματιστικές διεπαφές (APIs) έτσι ώστε τρίτες εφαρμογές να μπορούν να έχουν πρόσβαση σε αυτές τις δομές. Το ευρύτερα χρησιμοποιούμενο πρωτόκολλο για αυτού του είδους την επικοινωνία είναι το HTTP, ενώ η XML (eXtensible Markup Language) είναι η τυπική γλώσσα ανταλλαγής δεδομένων.

2.4.2 Κατανεμημένη αρχιτεκτονική

Σε μία κατανεμημένη αρχιτεκτονική [9], οι εξυπηρέτες αναλαμβάνουν να συγκεντρώσουν τα δεδομένα για την τρέχουσα κατάσταση μίας ή περισσότερων οντοτήτων από widgets που βρίσκονται σε διαφορετικές τοποθεσίες.

Το πλεονέκτημα να χρησιμοποιείται κατανεμημένη αρχιτεκτονική είναι ότι συγκεντρώνεται πληροφορία από διαφορετικές πηγές, με δυνατότητα διασταύρωσής της, αλλά και επεξεργασίας της και εξατομίκευσής της ανάλογα με τον χρήστη και τις προτιμήσεις του.(π.χ. μπορούν να συγκεντρωθούν στοιχεία για άτομα που βρίσκονται στην περιοχή κάποιου μουσείου και να προωθηθούν πληροφορίες για τα εκθέματα, αλλά αυτό να γίνει στη γλώσσα επιλογής του κάθε χρήστη) Μάλιστα η πληροφορία συλλέγεται από ένα ενδιάμεσο επίπεδο, το οποίο διαθέτει μια πιο γενική γλώσσα επικοινωνίας με τους διάφορους κόμβους από τους οποίους

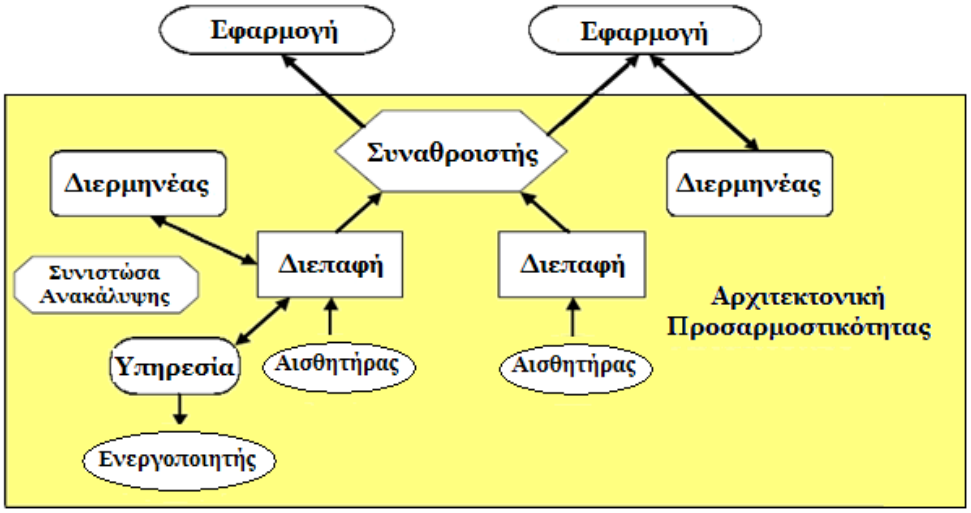
προέρχεται η πληροφορία. Ταυτόχρονα με αυτήν την αρχιτεκτονική επιτυγχάνεται η άμεση επεκτασιμότητα χωρίς να επηρεαστεί καθόλου η γενικότερη αρχιτεκτονική (π.χ. μπορούμε να προσθέσουμε όσους αισθητήρες χρειαζόμαστε χωρίς καμία επιβάρυνση στο σύστημα) Αυτό είναι ένα πρόσθετο πλεονέκτημα έναντι της μη κατανεμημένης, στην οποία απαιτούνται συνεχώς αλλαγές με την κάθε προσθήκη αισθητήτα, δεδομένου ότι η άμεση επικοινωνία με κάθε αισθητήρα είναι υποχρεωτική.

Στις εφαρμογές με επίγνωση περιβάλλουσας κατάστασης, η κατανεμημένη αρχιτεκτονική δίνει τη δυνατότητα οι κόμβοι να γνωρίζουν τη θέση άλλων κοντινών κόμβων, επιτρέποντας έτσι την αποτελεσματική γεωγραφική δρομολόγηση. Δημιουργείται όμως κι ένα βασικό μειονέκτημα κι αυτό είναι το ότι οι πληροφορίες φτάνουν στην εφαρμογή με καθυστέρηση. Στη χειρότερη περίπτωση, η καθυστέρηση αυτή ισούται με το διάστημα σταθμοσκόπησης (polling), επηρεάζεται δηλαδή η περιοδικότητα με την οποία διαβάζουμε την εφαρμογή και στις κρίσιμες εφαρμογές, αν υπάρχει καθυστέρηση, επηρεάζεται η συμπεριφορά της εφαρμογής. Για παράδειγμα, αν έχουμε ιατρικούς αισθητήρες που λαμβάνουν δεδομένα από ασθενείς, αν υπάρξει καθυστέρηση και δεν ληφθεί έγκαιρα ένα σήμα, μπορεί να επηρεαστεί η ζωή κάποιου ασθενούς

Ένα επιπλέον μειονέκτημα είναι ότι δημιουργείται παραπάνω κόστος γιατί πρέπει να διατηρηθεί η ίδια πληροφορία για κάθε widget σε διαφορετικούς κόμβους, καθώς μπορούν να δημιουργηθούν και άλλα προβλήματα όπως προβλήματα επικοινωνίας μεταξύ των κόμβων με αποτέλεσμα να μην υπάρχει πρόσβαση στην πληροφορία.

2.5 Ανάπτυξη context-Aware εφαρμογών: βιβλιοθήκες, εργαλειοθήκες

Το πλαίσιο που παρουσιάζεται [10] είναι ένα από τα πρώτα πλαίσια υλοποίησης εφαρμογών με επίγνωση της περιβάλλουσας κατάστασης που διαθέτει και ολοκληρωμένο θεωρητικό υπόβαθρο. Η **Error! Reference source not found.** 1 απεικονίζει τις οντότητες αυτού του πλαισίου.



Εικόνα 1: Εργαλειοθήκη δημιουργίας εφαρμογών με επίγνωση της περιβάλλουσας κατάστασης: γενική αρχιτεκτονική

2.5.1 Συνιστώσες

Οι συνιστώσες (widgets) [10] εσωκλείουν τη διαχείριση των συσκευών εισόδου (π.χ. ανιχνευτές κατάστασης) και είναι αυτές που παρέχουν την κατάλληλη διεπαφή στους προγραμματιστές των εφαρμογών για την ενσωμάτωσή τους στις διάφορες εφαρμογές. Οποιεσδήποτε αλλαγές σε μια γραφική συνιστώσα και στον τρόπο διαχείρισης μίας συσκευής δεν έχουν αντίκτυπο στις υπόλοιπες ενότητες της εφαρμογής. Ειδικά οι γραφικές συνιστώσες διαχειρίζονται την αλληλεπίδραση χρήστη – συσκευής, και ενημερώνουν την εφαρμογή για ενέργειες που έχουν προκύψει ως αποτέλεσμα της

αλληλεπίδρασης. Ως βασική συνιστώσα των εφαρμογών η κάθε γραφική συνιστώσα λειτουργεί ανεξάρτητα από τις υπόλοιπες ενότητες του λογισμικού, εξασφαλίζοντας την επαναχρησιμοποίηση τους σε επόμενες εφαρμογές.

2.5.2 Διερμηνείς περιβάλλουσας πληροφορίας

Οι διερμηνείς (interpreters) [10] αποτελούν τις επόμενες συνιστώσες μίας εφαρμογής με επίγνωση της περιβάλλουσας κατάστασης και είναι υπεύθυνες για το φιλτράρισμα των πληροφοριών που λαμβάνουν από τις συσκευές εισόδου (ανιχνευτές κατάστασης) μέσω των γραφικών διεπαφών (widgets) τους. Οι μεταφραστές λαμβάνουν δεδομένα από μία ή περισσότερες διεπαφές ή άλλες πηγές ταυτόχρονα, με σκοπό το σχηματισμό νέων δεδομένων που απλοποιούν ή ενοποιούν την πληροφορία που οι ίδιοι φιλτράρουν. Για παράδειγμα, ο μεταφραστής τοποθεσίας λαμβάνει γεωγραφικές συντεταγμένες που αντικατοπτρίζουν την κίνηση ενός οχήματος από ένα δορυφόρο. Το επόμενο βήμα για το μεταφραστή είναι να μετατρέψει τις συντεταγμένες σε σημείο απεικόνισης πάνω σε ένα χάρτη (π.χ. οδός σε συγκεκριμένη περιοχή μίας πόλης).

2.5.3 Συναθροιστές

Οι συναθροιστές (aggregators) [10] λαμβάνουν δεδομένα από διαφορετικές πηγές (π.χ. widgets) με σκοπό τη σύνθεσή τους σε μία νέα δομή δεδομένων που έχουν μία λογική συνάφεια. Η σύνθεση πληροφοριών από επί μέρους καταστάσεις (π.χ. χρόνος, τοποθεσία, δραστηριότητα) είναι χρήσιμη γιατί παρέχει στην εφαρμογή μία ενιαία κατάσταση για μία οντότητα άμεσου ενδιαφέροντος για μία εφαρμογή. Για παράδειγμα, μία τέτοια ευρύτερη οντότητα μπορεί να είναι οι εργαζόμενοι σε μία εταιρία ή οι εργαζόμενοι που μετακινούνται με οχήματα και εκτελούν εξωτερικές εργασίες κ.ά.

2.5.4 Υπηρεσίες

Οι υπηρεσίες (services) [10] που έχουν επίδραση στις καταστάσεις που ενδιαφέρουν μία εφαρμογή είναι συνιστώσες ανάλογες με αυτές των γραφικών συνιστωσών με αντίθετο όμως ρόλο. Οι υπηρεσίες δε διαβάζουν την κατάσταση μίας συσκευής εισόδου, αντίθετα τη διαμορφώνουν ή την τροποποιούν λαμβάνοντας εντολές από το περιβάλλον της εφαρμογής. Σε ορισμένες αρχιτεκτονικές, οι δυνατότητες των υπηρεσιών αυτών συγχωνεύονται με τις λειτουργίες των γραφικών συνιστωσών.

2.5.5 Συνιστώσες ανακάλυψης

Επιπλέον στην παραπάνω αρχιτεκτονική ορίζονται συνιστώσες (discoverers) που είναι υπεύθυνες για την ανακάλυψη, αποθήκευση και ανάκτηση διαφόρων τύπων συνιστωσών όπως αυτές που αναφέρθηκαν ανωτέρω [10]. Έτσι, widgets, διερμηνείς, συναθροιστές και υπηρεσίες που ανακαλύπτονται σε ένα συγκεκριμένο περιβάλλον αποθηκεύονται σε ένα ευρετήριο για την ενσωμάτωση τους σε διάφορες εφαρμογές. Οι συνιστώσες διαχείρισης του ευρετηρίου αποδίδουν σε κάθε τύπο συνιστώσας περιβάλλουσας κατάστασης μία μοναδική ταυτότητα και μία περιγραφή όσον αφορά τον εξειδικευμένο ρόλο που μπορεί να παρέχουν σε μία εφαρμογή (πχ widgets ενημερώνουν για την κίνηση σε ένα κτίριο Α, και κάποια άλλα widgets ενημερώνουν για την κίνηση σε ένα κτίριο Β).

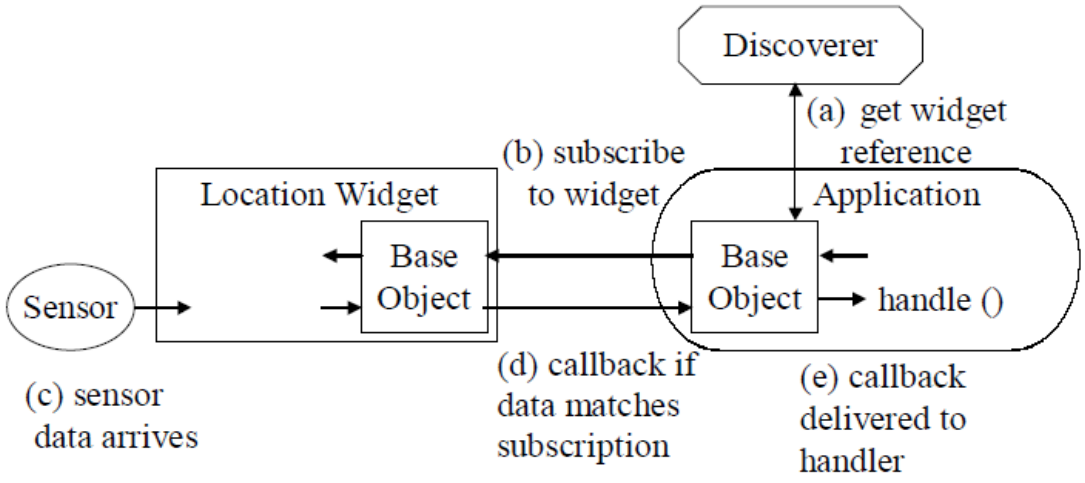
2.5.6 Μηχανισμοί αλληλεπίδρασης

Οι δύο κύριοι μηχανισμοί αλληλεπίδρασης μεταξύ των context συνιστωσών στα πλαίσια μίας εφαρμογής είναι οι εξής [10]:

1. ειδοποίησης σχετικά με μία ενέργεια ή αλλαγή κατάστασης (notification)

2. ερώτηση προς τη διεπαφή μίας συνιστώσας για τη λήψη πληροφορίας (query).

Η συνιστώσα (widget) δηλώνει ότι παρακολουθεί την κατάσταση ενός αισθητήρα ο οποίος μόλις αλλάξει κατάσταση στέλνει μία ειδοποίηση(callback). Η συνιστώσα έχει αυτόν τον μηχανισμό που ειδοποιείται γι' αυτό το callback, γι' αυτή την αλλαγή κατάστασης του αισθητήρα, και με βάση αυτό μπορεί μετά να εκτελέσει μία ενέργεια και να αλλάξει και αυτό κατάσταση ή να προβεί σε μία άλλη ενέργεια.
Η Εικόνα 2 απεικονίζει ένα παράδειγμα ειδοποίησης (callback) αλλαγής στην κατάσταση ενός ανιχνευτή κατάστασης (π.χ. καταγράφει την κίνηση σε ένα δωμάτιο).
Η ειδοποίηση μεταφέρεται από τη συνιστώσα προς την εφαρμογή που έχει δηλώσει ότι θέλει να λαμβάνει ειδοποιήσεις από το συγκεκριμένο αισθητήρα.



Εικόνα 2: Παράδειγμα κοινοποίησης αλλαγής στην κατάσταση «τοποθεσία» [10]

Οι δύο παραπάνω μηχανισμοί επιτρέπουν στις συνιστώσες επίγνωσης περιβάλλουσας κατάστασης που χρησιμοποιούνται να παρέχουν ενημέρωση για τις καταστάσεις που ενδιαφέρουν μία ή περισσότερες εφαρμογές ταυτόχρονα[10]. Δηλαδή υπάρχουν κάποιες εφαρμογές που κάνουν τη δουλειά τους ανεξάρτητα η μία από την άλλη (δεν

γνωρίζει η μία την παρουσία της άλλης) αλλά παίρνουν πληροφορία από την ίδια συνιστώσα επίγνωσης περιβάλλουσας κατάστασης. Επιπρόσθετα, η επικοινωνία των εφαρμογών με τις συνιστώσες γίνεται μέσω πρωτοκόλλων ή γλωσσών προγραμματισμού που έχουν ευρύτερη αποδοχή ή αποτελούν πλέον πρότυπο υλοποίησης τέτοιων υπηρεσιών (π.χ. HTTP, XML). Αυτό διευκολύνει σημαντικά όσους υλοποιούν ενότητες λογισμικού που αποσκοπούν στην επικοινωνία με άλλες ενότητες στα πλαίσια που ορίζει η αρχιτεκτονική μίας εφαρμογής (π.χ. Εικόνα 1) χωρίς να προβληματίζονται για το μηχανισμό επικοινωνίας.

2.5.7 Context Toolkit

Η διαμόρφωση ενός θεωρητικού πλαισίου ορισμού των βασικών οντοτήτων μίας εφαρμογής με επίγνωση της περιβάλλουσας κατάστασης επιτρέπει κατόπιν την απεικόνιση τους σε μία βιβλιοθήκη υλοποίησης όπως αυτή που περιγράφεται στο [10]. Η συγκεκριμένη βιβλιοθήκη έχει υλοποιηθεί σε Java, και χρησιμοποιεί τα πρότυπα HTTP και XML για την επικοινωνία μεταξύ των αντικειμένων. Η βιβλιοθήκη έχει χρησιμοποιηθεί σε ένα πλήθος εφαρμογών (π.χ. Active-Badge Call Forwarding, Mobile Tour Guide, Context-Aware Mailing List, κ.ά.) και μία ευρεία γκάμα συσκευών (π.χ. φορητοί υπολογιστές, κινητά τηλέφωνα) και ανιχνευτών κατάστασης . Από την άλλη πλευρά, βασικό μειονέκτημα αυτής της βιβλιοθήκης είναι η έλλειψη ικανότητας κλιμάκωσης (scalability) της αρχιτεκτονικής, όταν ο αριθμός των συνιστωσών, που πρέπει να χρησιμοποιηθεί, αυξάνεται.

2.6 Υπηρεσίες Κινητού Εμπορίου: διαστάσεις κινητικότητας και προσαρμοστικότητας

Το [11] ορίζει μία νέα κατεύθυνση στην υλοποίηση υπηρεσιών ηλεκτρονικού εμπορίου που προσαρμόζονται σε συνθήκες του

περιβάλλοντος. Οι υπηρεσίες αυτές αποκαλούνται Mobile and Context-Aware Ecommerce Services (MCACSs) και οι σχετικές μεθοδολογίες ανάπτυξης τέτοιων υπηρεσιών αποσκοπούν στο να προσδιορίσουν τα παρακάτω:

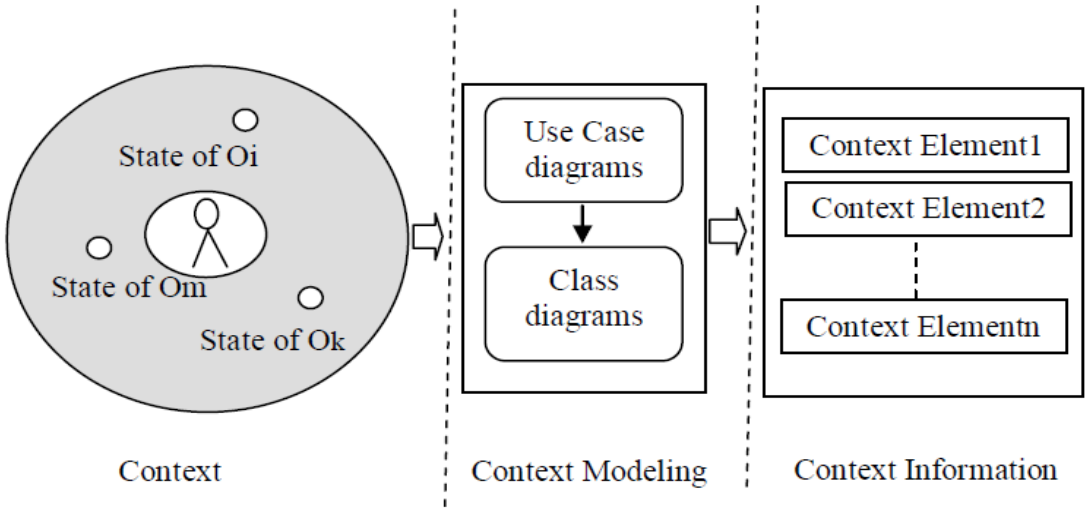
1. τις ενότητες των υπηρεσιών που θα προσαρμόζονται στις περιβάλλουσες συνθήκες (context) και ποιες ενότητες δεν θα τις λαμβάνουν υπόψη
2. τα κριτήρια προσαρμογής (adaptation criteria) στις περιβάλλουσες συνθήκες και σχεδιασμός των ανιχνευτών προσαρμογής (adaptation sensors)
3. τις πολιτικές προσαρμογής (adaptation policy) και τους αλγορίθμους υλοποίησης των πολιτικών αυτών και
4. την ενσωμάτωση λειτουργιών κινητικότητας (mobility features) και προσαρμοστικότητας (adaptation features) στις τυπικές υπηρεσίες ηλεκτρονικού εμπορίου.

Προσαρμοστικότητα συσκευής

Μία άλλη διάσταση που τίθεται στο [11] σχετικά με την υλοποίηση των MCACSs είναι η διεπαφή της κάθε συσκευής μέσω της οποίας ο κάθε χρήστης έχει πρόσβαση στην MCACS εφαρμογή. Και αυτό γιατί οι παράμετροι της διεπαφής όπως η διάσταση της οθόνης, ο τρόπος αλληλεπίδρασης του χρήστη (π.χ. πληκτρολόγιο, οθόνη αφής, κίνηση στην οθόνη χωρίς ποντίκι) θέτουν διάφορους περιορισμούς που πρέπει να ληφθούν υπόψη ως ένα σύνολο από ξεχωριστές καταστάσεις (device context). Οι εναλλακτικοί τρόποι αλληλεπίδρασης (modalities) προσαρμόζονται επίσης στα δεδομένα άλλων καταστάσεων (για παράδειγμα όταν κάποιος οδηγεί μπορεί να κάνει χρήση μίας συσκευής μόνο μέσω φωνητικής αναγνώρισης).

Στο [12] ορίζεται μία μεθοδολογία που να λαμβάνει υπόψη και τη προσαρμογή της διεπαφής (presentation) των φορητών συσκευών,

πέρα από την προσαρμογή του περιεχομένου (content) και των υπηρεσιών ηλεκτρονικού εμπορίου (services). Η μεθοδολογία βασίζεται στην αναπαράσταση της οντολογίας σε ένα πρώτο αφηρημένο επίπεδο (Εικόνα 3), όπου ορίζονται οι οντότητες Oi, …, Om που δραστηριοποιούνται στο συγκεκριμένο πλαίσιο (π.χ. χρήστες, συνδρομητές). Η κάθε οντότητα χαρακτηρίζεται από μία αφηρημένη κατάσταση.



Εικόνα 3: Αναπαράσταση καταστάσεων και των ιδιοτήτων τους ως μέρος μίας UML οντολογίας [14]

Στο επόμενο στάδιο, ορίζονται διαγράμματα περίπτωσης χρήσης (use cases) που ορίζουν τις διαδικασίες οι οποίες εκτελούνται στο συγκεκριμένο σύστημα, λαμβάνοντας υπ' όψιν τις καταστάσεις των οντοτήτων. Οι διαδικασίες αυτές εκτελούνται σε συνάρτηση συγκεκριμένων συνθηκών που ορίζουν το Πλαίσιο (Context) εκτέλεσης των εργασιών. Ταυτόχρονα, ορίζονται διαγράμματα κλάσεων (class diagrams) που ορίζουν κατάλληλες κλάσεις που μπορούν να αποθηκεύσουν τις ιδιότητες των οντοτήτων, αλλά και τα στοιχεία που αφορούν την περιβάλλουσα κατάστασή τους σε μία αντικειμενοστραφή αναπαράσταση. Οι κλάσεις αυτές λειτουργούν σε ένα συνολικό πλαίσιο από πληροφορίες για την υπηρεσία. Το πλαίσιο αυτό αποτελείται από ένα σύνολο καταστάσεων που παρέχουν

συνολική πληροφορία για την κατάσταση μίας οντότητας (τη χρονική στιγμή t, μετακινείται στη τοποθεσία l, ή/και κάνει την εργασία a, με τη συσκευή d, που έχει διαστάσεις οθόνης WxH, διαθέσιμη χωρητικότητα στο δίκτυο b, άλλες δυνατότητες κ.λπ.).

Το πλαίσιο των παραγόντων αυτών καθορίζει τις δυνατότητες προσαρμοστικότητας εφαρμογών κινητού εμπορίου (mobile commerce). Επιπρόσθετα, συμβάλει στην ανάλυση και σχεδιασμό των υπηρεσιών που να λαμβάνεται υπόψη η συσχέτιση τους με τους διάφορους παράγοντες και τις ιδιότητες τους (context analysis). Για παράδειγμα, η οθόνη της φορητής συσκευής μπορεί να χρησιμοποιεί διαφορετικά χρώματα για την προβολή των προϊόντων ανάλογα με το αν ο χρήστης το χρησιμοποιεί μέρα ή νύχτα, αν χρησιμοποιεί φωνή ή ποντίκι για επιλογές κ.λπ.

Επιπλέον, η μεθοδολογία επεκτείνει την τυπική αναπαράσταση των κλάσεων σε UML έτσι ώστε να μπορούν οι σχεδιαστές να σχεδιάζουν μέσα σε διαγράμματα κλάσεων και συσχετίσεων μεταξύ τους, τη χρήση δυναμικών πληροφοριών που προκύπτουν από την αναπαράσταση των πληροφοριών περιβάλλουσας κατάστασης. Για το σκοπό αυτό η προτεινόμενη μεθοδολογία επεκτείνει την αναπαράσταση UML με επιπλέον συμβολισμούς για τα παρακάτω:

1. Δυναμικότητα πληροφοριών για μία περιβάλλουσα κατάσταση (context information dynamicity) και διακρίνεται σε στατική και δυναμική πληροφορία
2. Τον τρόπο απόκτησης πληροφορίας για μία κατάσταση και διακρίνεται σε:
    - πληροφορία που έχει ληφθεί μέσω αισθητήρα (παρέχεται αυτόματα από μία συσκευή εισόδου)
    - πληροφορία που έχει ανακτηθεί από μία συσκευή εισόδου, συνήθως κατόπιν παρέμβασης του χρήστη ή ανάγνωσης αρχείου

- πληροφορία που έχει παραχθεί ως αποτέλεσμα επεξεργασίας ή κανόνων σε διαθέσιμη πληροφορία.

3. Μετα-δεδομένα όπως πηγή, χρονική στιγμή (timestamp), βαθμός βεβαιότητας για την τιμή, συχνότητα ανάκτησης, διάστημα ισχύος, μονάδα μέτρησης.
4. Ιστορικότητα τιμών συγκεκριμένων παραμέτρων.
5. Την πιθανότητα ότι ο τύπος μίας συγκεκριμένης οντότητας μπορεί να αλλάξει.

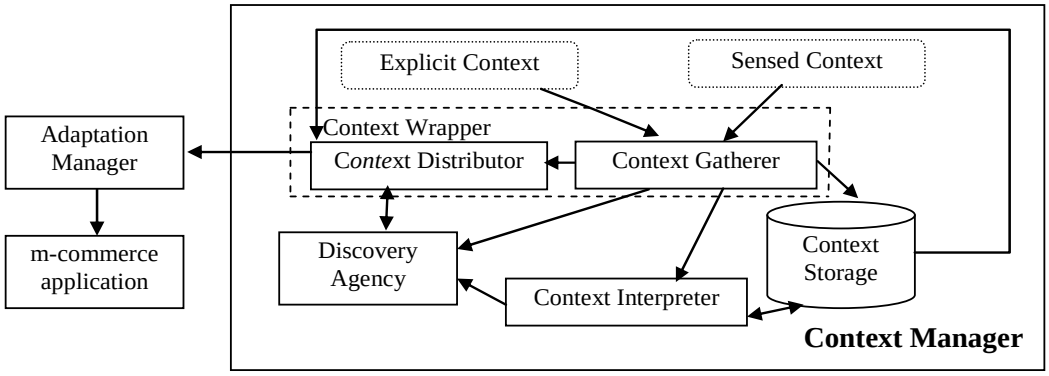
2.7 Υλοποίηση στρατηγικών προσαρμοστικότητας με εφαρμογή τεχνικών Aspect-Oriented Programming

Μία επιπλέον διάσταση που τίθεται στο [11] είναι ότι οι λειτουργίες κινητικότητας και προσαρμοστικότητας πρέπει να υλοποιούνται και να συντηρούνται ξεχωριστά από τις βασικές λειτουργίες. Η δυνατότητα αυτή μπορεί να υποστηριχθεί μέσω των δομών που παρέχει η προσέγγιση του Aspect-oriented programming [12]. Ο προγραμματισμός με απόψεις δίνει τη δυνατότητα σχεδιασμού των παραπάνω λειτουργιών ως *διατέμνοντα ενδιαφέροντα* (*cross-cutting concerns)*, δηλαδή οι λειτουργίες προσαρμοστικότητας και κινητικότητας μπορούν να υλοποιηθούν ως απόψεις που ενσωματώνονται στον κώδικα των βασικών λειτουργιών με τη χρήση ειδικών *aspect weavers*.

Μια τέτοια τεχνική προτείνεται από τους: Μπένου και Βασιλάκης [15] για τη διαχείριση των λειτουργικών προσαρμογής ως *διατέμνοντα ενδιαφέροντα*. Για την ενσωμάτωση της τεχνικής AOP στην υλοποίηση προσαρμοστικών υπηρεσιών ηλεκτρονικού εμπορίου, προτείνουν μία αρχιτεκτονική διαχείρισης της περιβάλλουσας κατάστασης (*Context Manager*). Η αρχιτεκτονική αυτή παρουσιάζεται στην Εικόνα 4, με τις βασικές συνιστώσες για τη συλλογή πληροφοριών περιβάλλουσας κατάστασης, την επεξεργασία και την κατανομή τους (π.χ. Εικόνα 2).

Οι βασικές λειτουργίες μιας εφαρμογής αφορούν την κύρια συμπεριφορά της και δεν εμπλέκονται με άλλες λειτουργίες που αφορούν την προσαρμοστικότητα και την κινητικότητά της εφαρμογής. Αυτό γίνεται για να μπορούν οι βασικές λειτουργίες να είναι επεκτάσιμες και συντηρήσιμες όσον αφορά την συμπεριφορά τους χωρίς να επηρεάζονται από άλλα χαρακτηριστικά.

Βασική αρχή του διατέμνοντα ενδιαφέροντα (cross-cutting concerns) είναι ότι αν θέλουμε να υλοποιήσουμε τις λειτουργίες που αφορούν την κύρια συμπεριφορά μιας εφαρμογής αυτή η βασική λειτουργία δεν εμπλέκεται με άλλα πράγματα που επηρεάζουν όχι μόνο αυτή την λειτουργία αλλά και άλλες λειτουργίες. Όλα τα υπόλοιπα τα υλοποιούμε ξεχωριστά ώστε αυτή η βασική λειτουργία να είναι επεκτάσιμη όσον αφορά τη συμπεριφορά της και χωρίς να επηρεάζεται από άλλα χαρακτηριστικά. Π.χ. αποφασίζουμε να βάλουμε ασφάλεια μέσα σε μια εφαρμογή. Υπό αυτές τις συνθήκες πρέπει σε κάθε λειτουργία να ελέγχουμε την ασφάλεια και όταν μετά θέλουμε να επεκτείνουμε αυτή τη λειτουργία είμαστε υποχρεωμένοι να λαμβάνουμε υπ’ όψιν κάθε φορά τον τρόπο λειτουργίας της ασφάλειας. Επίσης σε περίπτωση αναβάθμισης του συστήματος και εφαρμογής μιας νέας τεχνικής θα πρέπει να πάμε μέσα σε αυτές τις μεθόδους και να κάνουμε αλλαγές. Σε αντίθετη περίπτωση όμως, αν υλοποιηθούν ξεχωριστά, μπορεί ο κώδικας να είναι επεκτάσιμος και συντηρήσιμος. Με τον τρόπο αυτό διασφαλίζεται η ικανότητα κλιμάκωσης των λειτουργιών.



Εικόνα 4: Context Manager: αρχιτεκτονική και βασικές συνιστώσες [3]

2.7.1 Διαδικασία προσαρμογής

Σε σχέση με τη διαδικασία προσαρμογής θα πρέπει να απαντηθούν τα ακόλουθα ερωτήματα:

1. *"Πότε θα γίνει η προσαρμογή"*: σύμφωνα με τους συγγραφείς [3,4], στη περίπτωση αυτή υπάρχουν 2 επιλογές: η προσαρμογή να γίνει στατικά ή δυναμικά. Οι εφαρμογές κινητού ηλεκτρονικού εμπορίου υλοποιούνται συνήθως σε πλατφόρμα web, το οποίο σημαίνει ότι χρησιμοποιείται ο μηχανισμός αίτησης-απάντησης του πρωτοκόλλου HTTP. Η απάντηση του εξυπηρέτη στα αιτήματα του εξυπηρετούμενο HTTP είναι δυναμική, συνεπώς η καλύτερη επιλογή είναι η *δυναμική προσαρμογή* διότι είναι στη φύση των εφαρμογών που υλοποιούνται σε web, να μην μπορούν να γίνουν με στατική εφαρμογή, με την έννοια ότι υπάρχει πληροφορία που δεν είναι γνωστή εκ των προτέρων π.χ. ποιες είναι οι διαστάσεις στην οθόνη του υπολογιστή του χρήστη.
2. *"Ποιές θα είναι οι εργασίες προσαρμογής" [3,4]:*

   I. *Προσαρμογή περιεχομένου* (content adaptation): ανάλογα με την περιβάλλουσα κατάσταση καθορίζονται τα δεδομένα που θα χρησιμοποιηθούν και θα παρουσιαστούν στο χρήστη, αναφορικά με τα στοιχεία δεδομένων που θα

χρησιμοποιηθούν (π.χ. χρήση στοιχείων κειμένου έναντι στοιχείων εικόνας).

II. *προσαρμογή λειτουργιών* (operation adaptation): η περιβάλλουσα πληροφορία φιλτράρει τις λειτουργίες της εφαρμογής που είναι διαθέσιμες ανά σελίδα.

III. *προσαρμογή παρουσίασης* (presentation adaptation): βάσει της περιβάλλουσας κατάστασης καθορίζονται επίσης οι τιμές για ιδιότητες παρουσίασης (χρώμα, μέγεθος, γραμματοσειρά, θέση κ.λπ.) της διεπαφής της εφαρμογής. Για παράδειγμα, σε φορητές συσκευές θα πρέπει να χρησιμοποιηθεί μεγαλύτερη γραμματοσειρά για τις επιλογές που εμφανίζονται στην οθόνη.

3. *"Πού θα γίνει η προσαρμογή"*: σύγχρονες τεχνολογίες στην ανάπτυξη web εφαρμογών, π.χ. η τεχνολογία Ajax, επιτρέπουν τη δυναμική ενημέρωση τμημάτων μίας σελίδας. Επιλέγοντας το μοντέλο της κεντρικής διαχείρισης, όλες οι πολιτικές προσαρμογής καθορίζονται στο server. Η αρχιτεκτονική αυτή ενσωματώνει κώδικα που εσωκλείει την υλοποίηση της προσαρμογής (hard-coded) ή υλοποιεί παραμετροποιημένες μεθόδους και κλάσεις που προσαρμόζονται στις ιδιότητες της περιβάλλουσας πληροφορίας και με βάση τις τιμές της περιβάλλουσας πληροφορίας καθορίζονται οι λειτουργίες της προσαρμογής μέσα σε αυτές τις μεθόδους. Είναι σαφές ότι η δεύτερη περίπτωση είναι πιο ευέλικτη διότι επιτρέπει οι πολιτικές προσαρμογής να αποθηκεύονται σε ξεχωριστά αρχεία ή βάσεις δεδομένων και διαβάζονται δυναμικά από την εφαρμογή έτσι ώστε να εφαρμοστούν.

4. *"Πώς θα γίνει η προσαρμογή":* Οι κλάσεις που υλοποιούν τη στρατηγική της προσαρμογής αναμειγνύονται δυναμικά με τις κλάσεις που υλοποιούν το λογισμικό της εφαρμογής και μέσω της τεχνικής του Aspect-Oriented Programming.

2.7.2 Aspect-Oriented Programming (AOP)

Οι γλώσσες που υλοποιούν αυτή την τεχνική εφαρμόζουν δομές όπως *σημείο συνένωσης, σημείο τομής, συμβουλή, και απόψεις* [3,12,14]. Επιπλέον σύγχρονα εργαλεία AOP (π.χ. Java Spring Framework) επιτρέπουν την υλοποίηση των παραπάνω δομών χρησιμοποιώντας *αντικατοπτρισμό της υπολογιστικής* (*computational reflection*) [15]. Συνοπτικά, οι δομές του Aspect-Oriented Programming είναι οι εξής:

1. ***Σημείο συνένωσης (Join Point)*:** Ένα *σημείο συνένωσης* αντιπροσωπεύει ένα σημείο που διατέμνει (*cross-cutting concern)* το κύριο πρόγραμμα [15]. Αυτά τα σημεία μπορεί να είναι κλήσεις ή εκτελέσεις μεθόδων (method calls), χειριστές εξαιρέσεων (exception handlers), ανάκτηση ή αλλαγή τιμής πεδίου (field get/set), και άλλα [3,14].
2. ***Σημείο τομής (Pointcut)*:** να *σημείο τομής* συνθέτει μία ομάδα από σημεία συνένωσης και κατευθύνει τον aspect weaver για το πως θα «ταιριάξει» (match) το κάθε σημείο συνένωσης κατά την εκτέλεση του προγράμματος [3,14].
3. ***Συμβουλή (Advice):*** όταν θα «ταιριάξει» ένα σημείο συνένωσης ενός σημείου τομής κατά την εκτέλεση, μία *συμβουλή* ορίζει ένα κομμάτι κώδικα που θα εκτελεστεί για το συγκεκριμένο σημείο τομής. Ο κώδικας αυτός μπορεί να εκτελεστεί σε τρεις διαφορετικές θέσεις: *before*, *after* και *around.*
4. ***Απόψεις (Aspects):*** είναι οι πιο γενικές δομές που ορίζονται στο ίδιο επίπεδο και με τον ίδιο τρόπο που σχηματίζονται οι κλάσεις και ενσωματώνουν όλες τις παραπάνω δομές.
5. ***Δηλώσεις μεταξύ τύπων (Inter-type declarations):*** μία άλλη δυνατότητα του AOP είναι να μεταβάλει τον ορισμό των κλάσεων και των κληρονομικών ιεραρχιών (inheritance hierarchies) έξω από τον αρχικό ορισμό μίας κλάσης. Για παράδειγμα δημιουργούνται

απόψεις που προσθέτουν μεθόδους, πεδία σε μια ήδη υπάρχουσα κλάση ή να υλοποιήσει νέα interfaces.

Πράγματι οι δομές που προσφέρει η τεχνική του Aspect-Oriented Programming δίνουν την προοπτική να δημιουργηθούν λειτουργίες προσαρμοστικότητας στα πλαίσια του κινητού ηλεκτρονικού εμπορίου. Οι διαδικασίες που καθορίζουν την παρουσίαση, το περιεχόμενο και τις λειτουργίες στην εφαρμογή είναι σημαντικά cross-cutting concerns στην εφαρμογή, και συνεπώς ορίζουν τα αντίστοιχα σημεία συνένωσης και σημεία τομής για την εκτέλεση λειτουργιών προσαρμοστικότητας. Οι λειτουργίες αυτές θα προστεθούν ως συμβουλές πριν ή μετά την κλήση των μεθόδων (*before or after συμβουλές*), ή με την αντικατάσταση ολόκληρης της μεθόδου (*around συμβουλή*). Μέσα στον κώδικα των συμβουλών υπάρχει πρόσβαση σε αντικείμενα που επιστρέφονται από τα αντίστοιχα σημεία συνένωσης της βασικής εφαρμογής και η ανάλογη επεξεργασία τους επηρεάζει την παρουσίαση, περιεχόμενο ή συμπεριφορά της εφαρμογής. Στη δυνατότητα αυτή του AOP θα βασιστούμε για να δημιουργήσουμε μία αρχιτεκτονική προσαρμοστικότητας για τις εφαρμογές κινητού ηλεκτρονικού εμπορίου.

3. Προτεινόμενη Αρχιτεκτονική

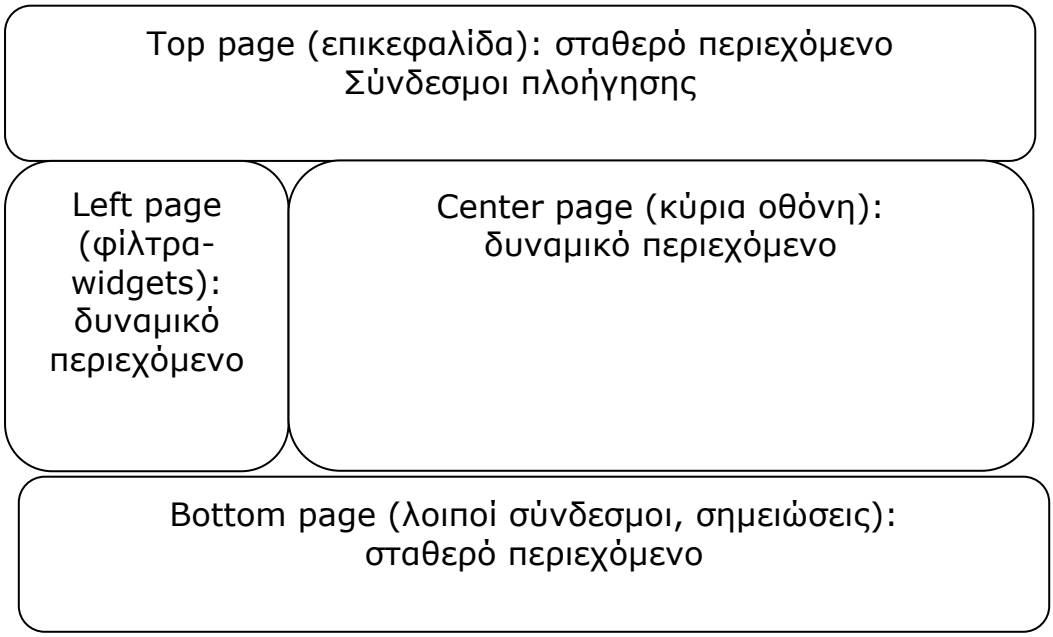
3.1 Εφαρμογή κινητού εμπορίου

Πριν αναφερθούμε στις λεπτομέρειες υλοποίησης της προτεινόμενης αρχιτεκτονικής θα πρέπει να κάνουμε την απαραίτητη εισαγωγή στην εφαρμογή κινητού εμπορίου που χρησιμοποιείται στην εργασία αυτή. Η εφαρμογή που έχει υλοποιηθεί παρέχει υπηρεσίες αναζήτησης εστιατορίων και φαρμακείων στους χρήστες είτε μέσω υπολογιστή είτε κινητού τηλεφώνου. Στη δεύτερη περίπτωση η εφαρμογή αναμένεται να χρησιμοποιηθεί και από μία μερίδα χρηστών που βρίσκονται ήδη καθ' οδόν προς εξεύρεση εστιατορίου/φαρμακείου είτε με τα πόδια είτε με το αυτοκίνητο τους. Επομένως λαμβάνεται υπόψη η διαθεσιμότητα του κάθε σημείου ενδιαφέροντος τη δεδομένη χρονική στιγμή καθώς και η τοποθεσία.

Η Εικόνα 5 απεικονίζει τα μέρη από τα οποία αποτελείται η γραφική διεπαφή της εφαρμογής που εμφανίζεται στον υπολογιστή ή το κινητό τηλέφωνο. Καθώς πρόκειται για εφαρμογή ιστού με δυναμικό περιεχόμενο, η εφαρμογή είναι προσπελάσιμη μέσω ενός προγράμματος περιήγησης (browser). Τα σύγχρονα κινητά έχουν πρόσβαση στο διαδίκτυο και διαθέτουν ένα πρόγραμμα περιήγησης.

Η κάθε σελίδα html της γραφικής διεπαφής χωρίζεται σε τέσσερα κύρια μέρη: το πάνω μέρος περιλαμβάνει κάποιες επικεφαλίδες και τους απαραίτητους συνδέσμους πλοήγησης (π.χ. αρχική σελίδα, επικοινωνία, βοήθεια). Το κάτω μέρος μπορεί να περιλαμβάνει επιπλέον συνδέσμους ή βασικές πληροφορίες για την εφαρμογή (π.χ. ιδιοκτησία, δικαιώματα χρήσης κ.λπ). Το πάνω και το κάτω μέρος εμφανίζουν σταθερό περιεχόμενο σε όλες τις σελίδες τις οποίες επισκέπτεται ο χρήστης.

Αντίθετα, το αριστερό και το δεξιό (κεντρικό) μέρος της σελίδας εμφανίζουν δυναμικό περιεχόμενο και αποτελούν τον κορμό των βασικών λειτουργιών της εφαρμογής και της εφαρμογής πολιτικών προσαρμοστικότητας σε αυτές.



Εικόνα 5: Web εφαρμογή: διαμόρφωση σελίδας

Ο Πίνακας 1 σημειώνει την αντιστοιχία των βασικών σελίδων (λειτουργιών) της εφαρμογής και του δυναμικού περιεχομένου στο αριστερό και κεντρικό μέρος της εκάστοτε σελίδας. Για παράδειγμα, το αριστερό μέρος εμφανίζει δύο σημαντικά widgets για τη συλλογή context πληροφορίας: αυτά είναι οι προτιμήσεις του χρήστη (π.χ. κατηγορίες εστιατορίων) και ο τρόπος μετακίνησης του (π.χ. πεζός ή με αυτοκίνητο).

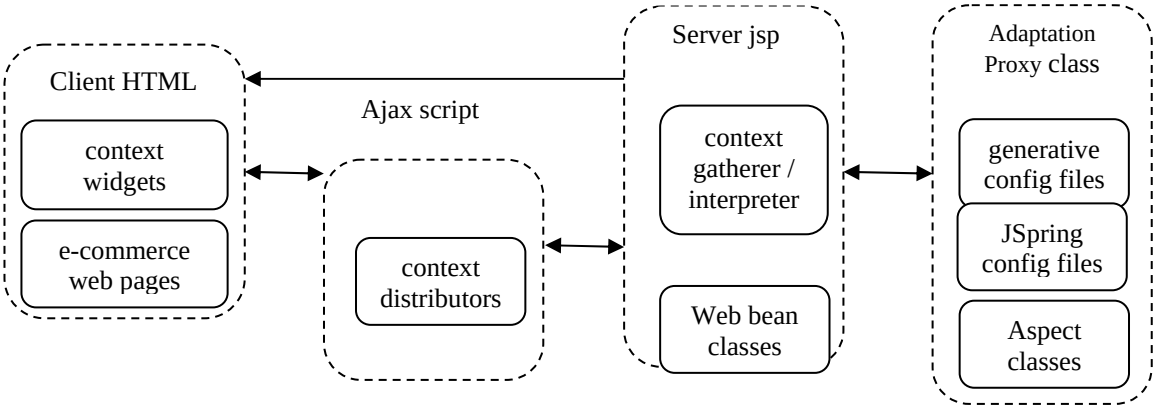
Λειτουργία ανά σελίδα	Left page	Center page
Εισαγωγική σελίδα	Κενή	Επιλογή «εστιατόρια» ή «φαρμακεία»
Σελίδα εισόδου χρήστη	Κενή	Φόρμα εισαγωγής ονόματος χρήστη και κωδικού πρόσβασης
Σελίδα εγγραφής νέου χρήστη	Κενή	Φόρμα καταχώρισης στοιχείων νέου χρήστη
Σελίδα αναζήτησης εστιατορίων	Widget A: Κατηγορίες εστιατορίων Widget B: Μετακίνηση: «πεζός», «αυτοκίνητο»	Εμφάνιση πίνακα με επιλογές διαθέσιμων εστιατορίων που ταιριάζουν στα κριτήρια αναζήτησης αριστερά
Σελίδα αναζήτησης φαρμακείων	Widget B: Μετακίνηση: «πεζός», «αυτοκίνητο»	Εμφάνιση πίνακα με επιλογές διαθέσιμων φαρμακείων που ταιριάζουν στα κριτήρια αναζήτησης αριστερά

Πίνακας 1: αντιστοιχία λειτουργιών με δυναμικό περιεχόμενο αριστερού και κεντρικού μέρους σελίδας

Οι επιλογές σε αυτά τα widgets θα φιλτράρουν το περιεχόμενο στο κεντρικό μέρος της σελίδας. Επιπλέον, οι επιλογές αυτές αποθηκεύονται στον εξυπηρέτη του συστήματος και χρησιμοποιούνται ως ένα αυτόματο φίλτρο (π.χ. με περιοδικότητα ορισμένων δευτερολέπτων ή λεπτών) για την δυναμική προσαρμογή του κεντρικού μέρους της σελίδας.

3.2 Αρχιτεκτονική Εφαρμογής

Η Εικόνα 6 απεικονίζει την προτεινόμενη αρχιτεκτονική του συστήματος προσαρμοστικότητας εφαρμογών ηλεκτρονικού εμπορίου που προτείνουμε. Η αρχιτεκτονική αυτή βασίζεται στην αρχιτεκτονική του Context Manager [3, 4] που παρουσιάσαμε στην προηγούμενη ενότητα (Εικόνα 4).



Εικόνα 6: Προτεινόμενη Αρχιτεκτονική

Όπως βλέπουμε στην εικόνα 6 τα context widgets και οι e-commerce web pages βρίσκονται από την πλευρά του client και καλύπτουν τις βασικές λειτουργίες του και μέσω των Ajax script σελίδων στέλνουν αιτήματα στις σελίδες JSP οι οποίες επιστρέφουν το αίτημα στον εξυπηρετούμενο.

Η Adaptation Proxy Class είναι η κλάση που καλείται από το αρχείο JSP και αυτή η κλάση διαβάζει τους aspect κανόνες από το xml αρχείο και αποφασίζει μετά ποια κλάση aspect θα καλέσει.

Η Adaptation Proxy Class αποφασίζει για το πλαίσιο προσαρμοστικότητας. Εκεί καλεί το xml αρχείο με το jsp configuration στο οποίο μέσα είναι όλοι οι κανόνες προσαρμοστικότητας. Άρα ο Adaptation Proxy ενορχηστρώνει τη διασύνδεση ανάμεσα στους κανόνες xml της διαμόρφωσης JSP και τις κλάσεις aspect, εκτελεί δηλαδή τη δυναμική διασύνδεση ανάμεσά τους. Φορτώνει δυναμικά στην μνήμη τις κλάσεις aspect που ταιριάζουν στους κανόνες AOP, οι οποίοι περιγράφονται στο αρχείο XML.

Η αρχιτεκτονική που προτείνουμε συνδυάζει βασικές έννοιες της Context Manager [3,14] αρχιτεκτονικής με τις διαθέσιμες τεχνολογίες υλοποίησης εφαρμογών κινητού (ηλεκτρονικού) εμπορίου. Οι τεχνολογίες αυτές συνδυάζουν την ανάπτυξη σελίδων HTML που το

περιεχόμενο τους προσαρμόζεται δυναμικά με τη συνδρομή της τεχνολογίας Ajax. Οι σελίδες html παράγονται δυναμικά από την εκτέλεση scripts JSP στον εξυπηρέτη εφαρμογής.

Οι ανωτέρω τεχνολογίες καλούνται να υλοποιήσουν το πλαίσιο προσαρμοστικότητας των εφαρμογών κινητού (ηλεκτρονικού) εμπορίου αξιοποιώντας παράλληλα την τεχνολογία Aspect-Oriented Programming. Όπως αναφέρεται και στην προηγούμενη ενότητα οι εφαρμογές αποτελούνται από τρία σκέλη: παρουσίαση, περιεχόμενο, λειτουργίες. Σκεπτόμενοι ότι η αρχιτεκτονική λαμβάνει υπόψη ότι η υλοποίηση γίνεται σε Java (π.χ. Ajax scripts, server-side JSP scripts), οι κλάσεις aspect θα πρέπει να χρησιμοποιήσουν παρόμοια γλώσσα υλοποίησης.

Μία AOP γλώσσα ευρύτερης αποδοχής είναι η AspectJ [15]. Επιπλέον, ένα νέο πλαίσιο έχει δημιουργηθεί για server-side aspect-oriented programming που ονομάζεται JSpring Framework [16]. Το τελευταίο είναι ιδανικό για επεμβάσεις σε εφαρμογές που εφαρμόζουν το πρότυπο MVC (Model-View-Control) στον σχεδιασμό τους. Συνεπώς το JSpring Framework ταιριάζει στις ανάγκες προσαρμοστικότητας της παρουσίασης (view), περιεχόμενο (model), λειτουργίες (control) μίας εφαρμογής ηλεκτρονικού εμπορίου.

Έτσι, μία ιστοσελίδα που δημιουργείται στατικά ή δυναμικά και εμφανίζεται στο χρήστη (client html) αποτελείται από δύο συνιστώσες: τα context widgets και τα βασικά συστατικά μίας σελίδας ιστού που έχει δημιουργηθεί από μία εφαρμογή που ακολουθεί το πρότυπο MVC. Τα widgets λαμβάνουν όλη την απαραίτητη περιβάλλουσα πληροφορία που αφορά το περιβάλλον της συσκευής εξυπηρετούμενου και το μεταφέρει μέσω του Ajax script (περιβάλλουσας πληροφορίας) στο κατάλληλο JSP script του εξυπηρέτη που είναι και ο συλλέκτης περιβάλλουσας πληροφορίας. Οι περιβάλλουσες συνθήκες που ενδιαφέρουν τον εξυπηρέτη περιλαμβάνουν:

1. Οθόνη (διαστάσεις) συσκευής
2. Δίκτυο (χωρητικότητα)
3. Προφίλ και προτιμήσεις χρήστη που είτε αποθηκεύονται σε μία βάση είτε εισάγονται μέσω μίας φόρμας αλληλεπίδρασης με τον χρήστη
4. Κίνηση /τοποθεσία χρήστη
5. Χρονική στιγμή

Το Ajax script συγκεντρώνει την απαραίτητη πληροφορία είτε μέσω client παραμέτρων είτε από events της αλληλεπίδρασης του χρήστη με τα context widgets. Ερμηνεύοντας τις συνθήκες που παράγονται και λαμβάνοντας υπόψη την τρέχουσα κατάσταση κατά την εκτέλεση της εφαρμογής αποφασίζει σε ποιο JSP script πρέπει να μεταφέρει το αίτημα από την client σελίδα (client request) μαζί με την περιβάλλουσα πληροφορία (context distributor). Μεταφέροντας το αίτημα εκτέλεσης μίας διεργασίας και συγχρόνως το πλαίσιο προσαρμοστικότητας που πρέπει να λάβει υπόψη της η διεργασία, το Ajax script περιμένει κατόπιν μία απάντηση. Η απάντηση αυτή θεωρείται προσαρμοσμένη στα δεδομένα της εφαρμογής αλλά και στα δεδομένα του περιβάλλοντος.

Ο εξυπηρέτης της εφαρμογής αναλαμβάνει την ανταπόκριση στα αιτήματα των πελατών απαντώντας με το ανάλογο html περιεχόμενο και περιβάλλοντάς το με τους κατάλληλους κανόνες προσαρμογής των MVC συνιστωσών της σελίδας. Το κάθε jsp script περιέχει μία ενότητα συλλογής περιβάλλουσας πληροφορίας (context gatherer). Αυτό γίνεται μεταφράζοντας τις παραμέτρους εισόδου κατά την υποβολή του αιτήματος από τον εξυπηρετούμενο ή ανακτώντας τιμές για αυτές τις παραμέτρους από μία βάση δεδομένων. Η επόμενη ενότητα αφορά τη διαδικασία προσαρμογής. Οι κλάσεις (beans) της εφαρμογής καθορίζουν την παρουσίαση, περιεχόμενο και λειτουργίες της δομής html που σχηματίζεται ως απάντηση. Επιπλέον, ο adaptation proxy

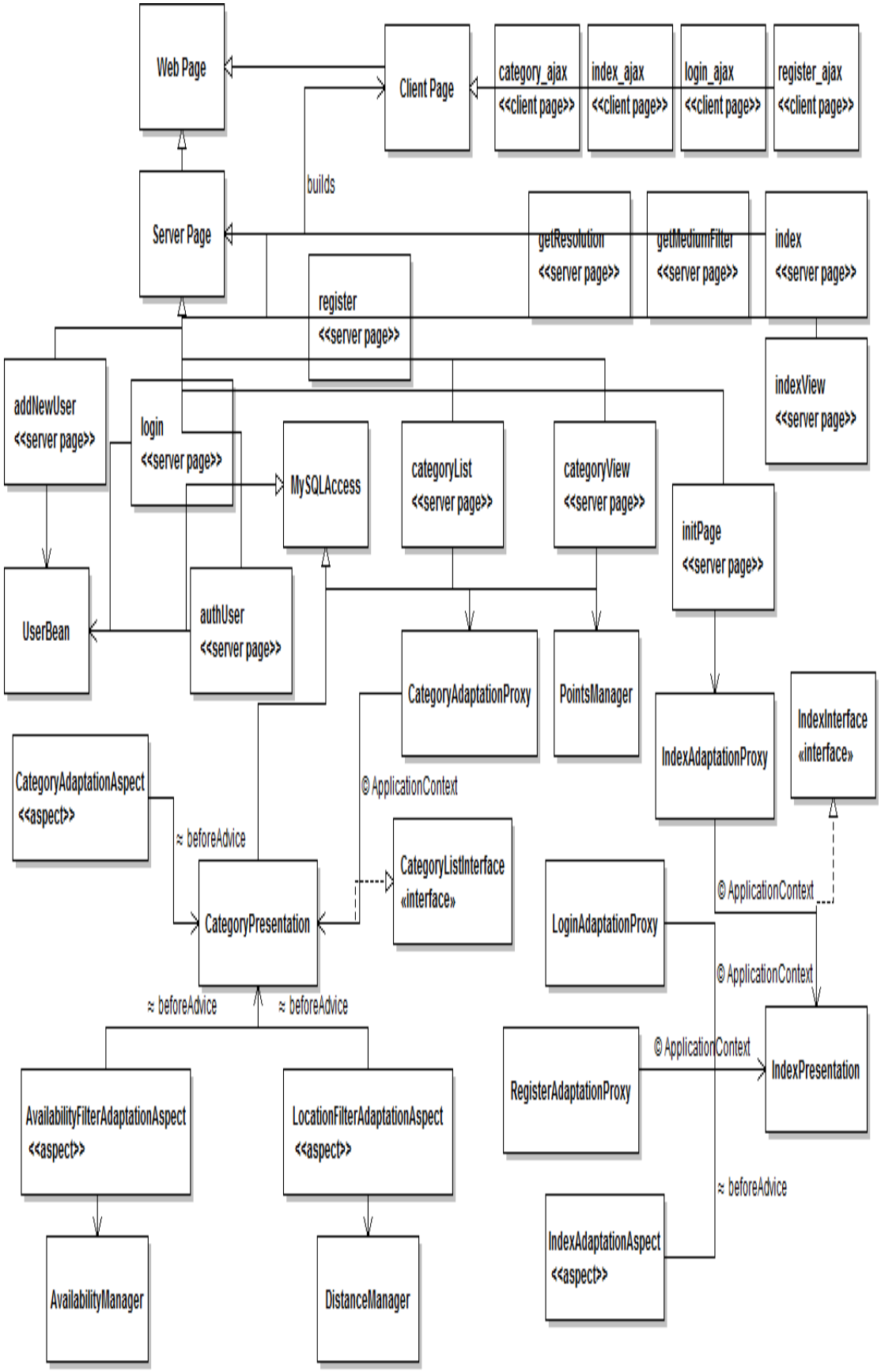
καλείται να φιλτράρει και να προσαρμόσει μία ή περισσότερες από τις δομές αυτές, επομένως το αποτέλεσμα που επιστρέφεται στον client είναι διαφορετικό από αυτό που είχε αρχικά εκτιμηθεί.
Ο adaptation proxy εκτελεί το ρόλο του βασιζόμενος σε τρεις συνιστώσες:

1. *generative configuration* αρχεία: περιέχουν κανόνες που καθορίζουν τον τρόπο παρουσίασης ορισμένων στοιχείων μίας σελίδας (π.χ. cascade stylesheet rules). Οι κανόνες αυτοί είναι απλοί και άμεσα εκτελέσιμοι. Για παράδειγμα αν τα cascade *stylesheet rules* αντιγραφούν σε μία σελίδα καθορίζουν αμέσως τον τρόπο παρουσίασης της σελίδας.
2. *JSpring XML files*: εκμεταλλευόμενοι το μηχανισμό του υπολογιστικού κατοπτρισμού (computational reflection) ορίζονται xml κανόνες που ορίζουν τη δυναμική αντιστοιχία κλάσεων και μεθόδων τους σε aspects και advices. Ουσιαστικά οι κανόνες αυτοί αντιπροσωπεύουν σημεία τομής (pointcuts) και ορίζουν πρότυπα (patterns) για το ποιες μεθόδους της συγκεκριμένης κλάσης το beforeAdvice, afterAdvice, ή aroundAdvice μπορεί να χειριστεί.

Η Εικόνα 7 παρουσιάζει το διάγραμμα κλάσεων της αρχιτεκτονικής. Το διάγραμμα UML επεκτείνει τη βασική ορολογία της UML γλώσσας για να αντικατοπτρίσει τη χρήση των aspect-oriented programming λειτουργιών. Οι κλάσεις με τον συμβολισμό «server page» αντιπροσωπεύουν τα jsp scripts που υλοποιούν τις προδιαγραφές της εφαρμογής του ηλεκτρονικού εμπορίου και καθοδηγούν την εφαρμογή των πολιτικών προσαρμοστικότητας (context collection, distribution). Τα jsp scripts θεωρούνται υπο-κλάσεις της γενικής κλάσης Server Page.
Επιπλέον, οι κλάσεις με τον συμβολισμό «client page» συνθέτουν τα Ajax scripts που συλλέγουν τις περιβάλλουσες πληροφορίες και τις

μεταφράζουν σε πολιτικές προσαρμοστικότητας που εκτελούνται από τα JSP scripts. Συγχρόνως, ένα Ajax script μεταφέρει το δυναμικό περιεχόμενο που επιστρέφει το εκάστοτε JSP script και το προσαρμόζει στη σελίδα που προβάλλεται στο χρήστη. Τα ajax scripts θεωρούνται υπο-κλάσεις της γενικής κλάσης Client Page.

Τα jsp scripts αλληλεπιδρούν με Java Beans, δηλαδή κλάσεις που δρομολογούν την παρουσίαση, περιεχόμενο και λειτουργίες που θα εμφανιστούν στην μπροστινή σελίδα της εφαρμογής. Επιπλέον, έχοντας συλλέξει το κατάλληλο σύνολο περιβάλλουσας πληροφορίας, αυτό μεταφέρεται στις κλάσεις εκείνες που πραγματοποιούν τις πολιτικές προσαρμοστικότητας στις μεθόδους των Java Beans. Η εκάστοτε AdaptationProxy κλάση διαπερνά το σύνολο της περιβάλλουσας πληροφορίας (© ApplicationContext) στις κλάσεις που καθορίζουν την παρουσίαση, περιεχόμενο και λειτουργίες ανά σελίδα.



Εικόνα 7: Διάγραμμα Κλάσεων

Ταυτόχρονα aspect classes («aspect») παρακολουθούν το πέρασμα του © *ApplicationContext* στις παραπάνω κλάσεις και ορίζουν σημεία τομής (pointcuts) για την δυναμική επέμβαση των ενσωματωμένων συμβουλών(advices) στα Java Beans. Αυτό παριστάνεται με τον συμβολισμό:

≈ beforeAdvice.

3.3 Επιμέρους διαγράμματα κλάσεων

Η Εικόνα 8 και η Εικόνα 9 αντίστοιχα απομονώνουν από το σφαιρικό διάγραμμα κλάσεων τις ενότητες που αφορούν την ενσωμάτωση της *aspect-oriented* λογικής στην υλοποίηση των πολιτικών προσαρμοστικότητας. Στην πρώτη περίπτωση ορίζονται τρεις διαφορετικές *AdaptationProxy* κλάσεις να χειριστούν την είσοδο του χρήστη στο σύστημα: εισαγωγική σελίδα, σελίδα ταυτοποίησης ή καταχώρισης νέου χρήστη. Οι κλάσεις αυτές προσαρμόζουν το MVC μοντέλο της σελίδας που εμφανίζεται στον χρήστη με βάση την περιβάλλουσα πληροφορία. Στη προκειμένη περίπτωση το context που λαμβάνεται υπόψη περιλαμβάνει τα εμφανιζόμενα στον Πίνακα 2:

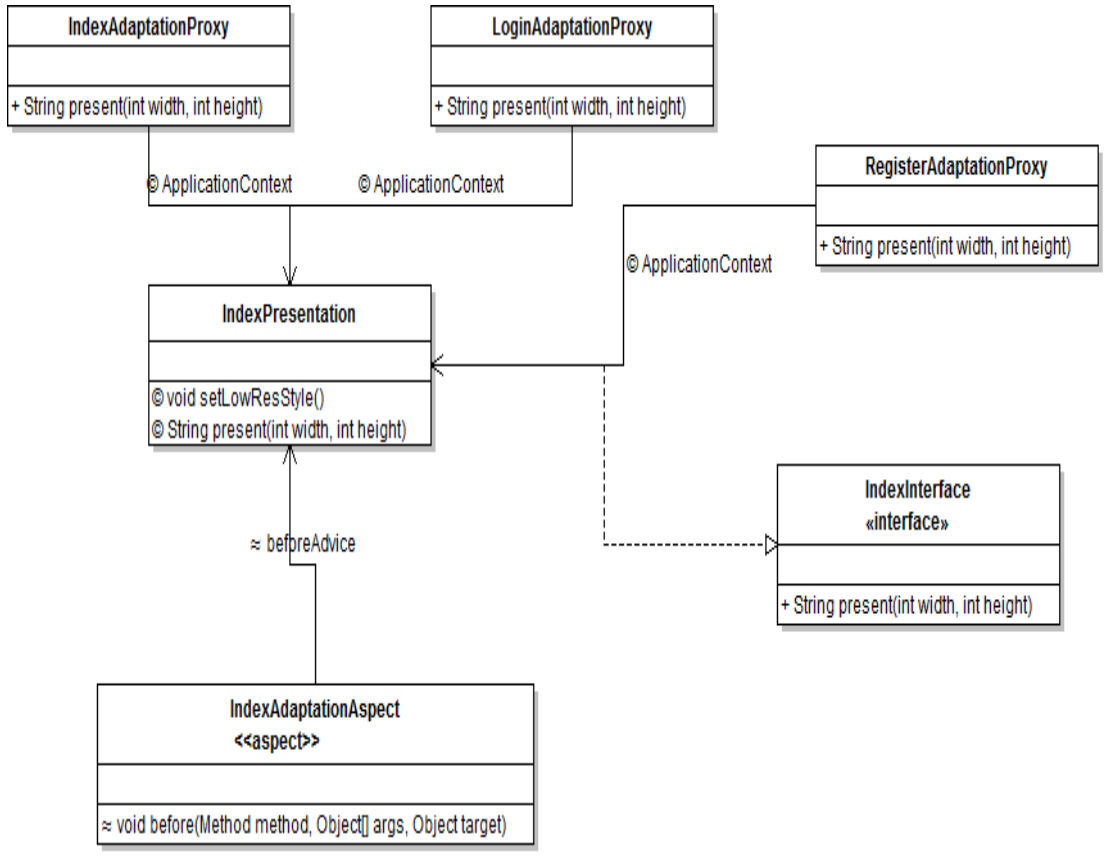
Context parameters	**Adaptation class A**	**Adaptation class B**
Διαστάσεις Οθόνης width, height	Υψηλή ανάλυση οθόνης (desktop/laptop PC)	Χαμηλή ανάλυση οθόνης (φορητή συσκευή)
	Cascade style sheet rules για υψηλή ανάλυση οθόνης	Cascade style sheet rules για χαμηλή ανάλυση οθόνης
Προφίλ χρήστη	Συχνός επισκέπτης: προσπέλαση οθόνη ταυτοποίησης	Πρώτη επίσκεψη: άμεση πρόσβαση στην οθόνη ταυτοποίησης ή καταχώρισης στοιχείων νέου χρήστη

Πίνακας 2: Context παράμετροι – κλάσεις προσαρμοστικότητας

Ανάλογα με τις διαστάσεις της οθόνης της συσκευής του χρήστη αποφασίζονται οι κανόνες προσαρμογής της παρουσίασης της σελίδας του χρήστη. Η κλάση *IndexAdaptationAspect* περιβάλλει την κλάση *IndexPresentation* για να επέμβει στον προσδιορισμό των κανόνων προσαρμοστικότητας με βάση την περιβάλλουσα πληροφορία (συμβολίζεται με: © *ApplicationContext*). Αυτό υλοποιείται στη μέθοδο

public void before(Method method, Object[] args, Object target)

που εκτελείται πριν τη μέθοδο *present*. Αυτό επιτρέπει τον καθορισμό της κλάσης προσαρμοστικότητας και αναλαμβάνει κατόπιν η μέθοδος *present* της κλάσης IndexPresentation να υλοποιήσει τους κανόνες της αντίστοιχης κλάσης προσαρμοστικότητας.



Εικόνα 8: Index Presentation Class Sub-Diagram: methods and pointcuts

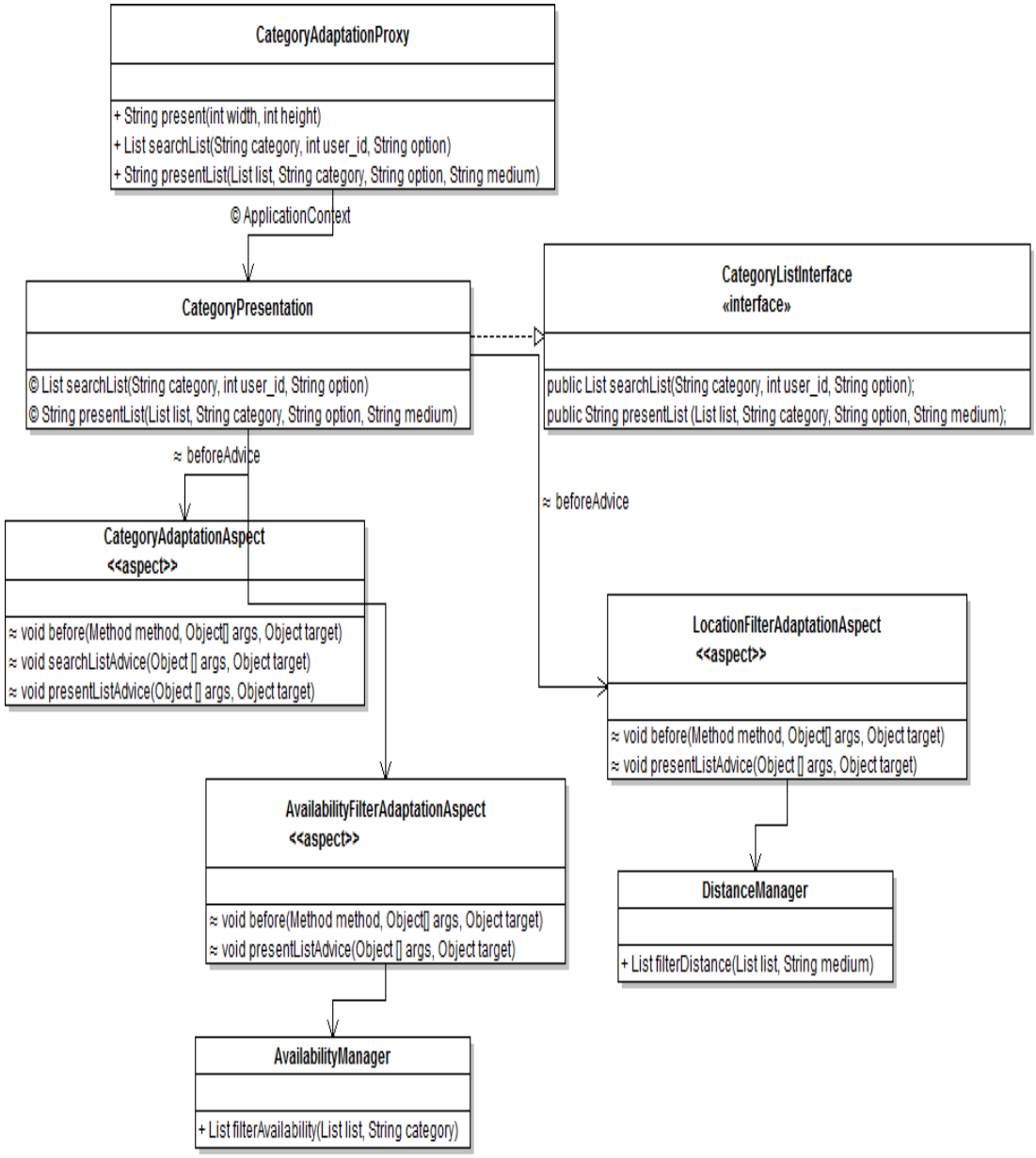
Η μέθοδος *beforeAdvice* υλοποιεί μία δυναμική πολιτική προσαρμοστικότητας στην μέθοδο *present* κατευθύνοντας την παρουσίαση της εφαρμογής. Η μέθοδος *present* εμφανίζεται από κοινού σε όλες τις κλάσεις *AdaptationProxy* μέσω της *κληρονομικότητας* καθώς ο σχεδιασμός της web εφαρμογής είναι τέτοιος έτσι ώστε να υπάρχουν κοινοί κανόνες για τη διαμόρφωση της κάθε σελίδας και το περιεχόμενο τους.

Οι κανόνες διαμόρφωσης της παρουσίασης της κάθε σελίδας υλοποιούνται στη μέθοδο *present* και ορίζονται σε ένα αρχείο διαμόρφωσης (cfg) όπως αυτά στο Παράρτημα Α'. Τα αρχεία "HighResIndex.cfg", "LowResIndex.cfg" περιέχουν κανόνες για τη δημιουργία του cascade stylesheet περιεχομένου για τη κάθε σελίδα. Το πρώτο αρχείο προσαρμόζει το περιεχόμενο του cascade stylesheet σε οθόνες υψηλής ανάλυσης (π.χ. υπολογιστή) και το δεύτερο αρχείο προσαρμόζει το περιεχόμενο σε οθόνες χαμηλής ανάλυσης (π.χ. κινητό τηλέφωνο). Αυτό αποτελεί ένα μόνο παράδειγμα υλοποίησης γεννητριών (generative) περιεχομένου για την προσαρμογή της παρουσίασης που λαμβάνει υπόψη δύο γενικευμένες κατηγορίες διαστάσεων οθόνης (υψηλή, χαμηλή). Θα μπορούσε εναλλακτικά να υπάρχει ένα γενικευμένο αρχείο με κανόνες προσαρμογής των διαστάσεων και των λοιπών στοιχείων των περιεχομένων (π.χ. πίνακες, συνιστώσες html, διαστάσεις div, μέγεθος γραμματοσειράς κ.ά.) και μία αντίστοιχη γεννήτρια cascade stylesheet περιεχομένου.

Παρακάτω θα εξηγήσουμε τον τρόπο με τον οποίο ενεργοποιείται η μέθοδος beforeAdvice όπως και κάθε άλλη aspect advice.

Το δεύτερο μέρος των διαγραμμάτων κλάσεων περιλαμβάνει τις aspect κλάσεις που προσαρμόζουν την παρουσίαση, περιεχόμενο και λειτουργίες των οθονών αναζήτησης (π.χ. εστιατορίων, φαρμακείων). Πέρα από τη μέθοδο *present* πάνω στην οποία εφαρμόζεται η

γενικευμένη πολιτική προσαρμογής όλης της σελίδας, η κλάση *CategoryPresentation* καλείται στις σελίδες αναζήτησης για να μεταφέρει στη σελίδα τα αποτελέσματα της αναζήτησης για τα συγκεκριμένα κριτήρια που έρχονται από το χρήστη (client) και κατόπιν να τα απεικονίσει σε μία συγκεκριμένη δομή (πίνακα) στο κύριο μέρος της σελίδας.



Εικόνα 9: Category Presentation Class Sub-Diagram: methods and pointcuts

Στις δύο αυτές μεθόδους αναζήτησης (*searchList*) και προβολής των αποτελεσμάτων (*presentList*) της κλάσης εφαρμόζονται μέσω του JSpring AOP μηχανισμού τρεις κύριες aspect κλάσεις:

1. CategoryAdaptationAspect: σχηματίζει τα κριτήρια αναζήτησης στη βάση δεδομένων με βάση περιβάλλουσα πληροφορία (προφίλ χρήστη, επιλογές χρήστη), ορίζει τη βασική δομή εμφάνισης του περιεχομένου των αποτελεσμάτων.
2. AvailabilityFilterAdaptationAspect: εφαρμόζει επιπλέον κριτήρια αναζήτησης στη βάση δεδομένων που να λαμβάνουν υπόψη τις παραμέτρους διαθεσιμότητας του κάθε σημείου αναζήτησης ως προς τη δεδομένη χρονική στιγμή. Για παράδειγμα, τα αποτελέσματα ταξινομούνται για την εύκολη διάκριση τους σε ανοιχτά και κλειστά εστιατόρια/φαρμακεία. Επιπρόσθετα, προσαρμόζει τη βασική δομή εμφάνισης του περιεχομένου των αποτελεσμάτων ώστε να εμφανίζεται και η πληροφορία της τοποθεσίας του κάθε σημείου αναζήτησης.
3. LocationFilterAdaptationAspect: εφαρμόζει επιπλέον κριτήρια αναζήτησης στη βάση δεδομένων που να λαμβάνουν υπόψη τις παραμέτρους τοποθεσίας και μετακίνησης του χρήστη κατά το φιλτράρισμα των αποτελεσμάτων. Προσαρμόζει τη βασική δομή εμφάνισης του περιεχομένου των αποτελεσμάτων ώστε να εμφανίζεται και η πληροφορία της τοποθεσίας του κάθε σημείου αναζήτησης.

Οι πολιτικές προσαρμοστικότητας που εφαρμόζονται επάνω στις δύο μεθόδους *searchList*, *presentList* σύμφωνα με περιβάλλουσες πληροφορίες για προτιμήσεις χρήστη, διαθεσιμότητα, τοποθεσία απεικονίζονται στον Πίνακα 3.

Context parameters	**searchList προσαρμογή**	**presentList προσαρμογή**
Προτιμήσεις χρήστη (εστιατόρια ή φαρμακεία, κατηγορίες εστιατορίων στη 1η περίπτωση)	Σχηματισμός SQL query αναζήτησης εστιατορίων ή φαρμακείων- ενσωμάτωση κατηγορίας εστιατορίων στην αναζήτηση για την 1η περίπτωση	Προβολή εστιατορίων ή φαρμακείων σε πίνακα. Προβολή κατηγορίας εστιατορίων στην 1η περίπτωση.
Διαθεσιμότητα εστιατορίων ή φαρμακείων την τρέχουσα χρονική στιγμή/ περιοδική αναζήτηση (κάθε 3')	ενσωμάτωση κριτηρίου διαθεσιμότητας στην αναζήτηση εστιατορίων ή φαρμακείων	Επέκταση προβολής των στοιχείων εστιατορίων ή φαρμακείων με την πληροφορία της διαθεσιμότητας. Ταξινόμηση σε Ανοιχτά/Κλειστά.
Τοποθεσία εστιατορίων ή φαρμακείων σε σχέση με την τοποθεσία του χρήστη. Υπολογισμός απόστασης με βάση τον τρόπο μετακίνησης του χρήστη - περιοδική αναζήτηση (κάθε 3')	ενσωμάτωση κριτηρίου απόστασης στην αναζήτηση εστιατορίων ή φαρμακείων	Επέκταση προβολής των στοιχείων εστιατορίων ή φαρμακείων με την πληροφορία της απόστασης. Εμφανίζονται αυτά που είναι σε απόσταση < 1χλμ αν ο χρήστης είναι πεζός ή αυτά που είναι σε απόσταση < 5χλμ αν ο χρήστης είναι με το αυτοκίνητο.

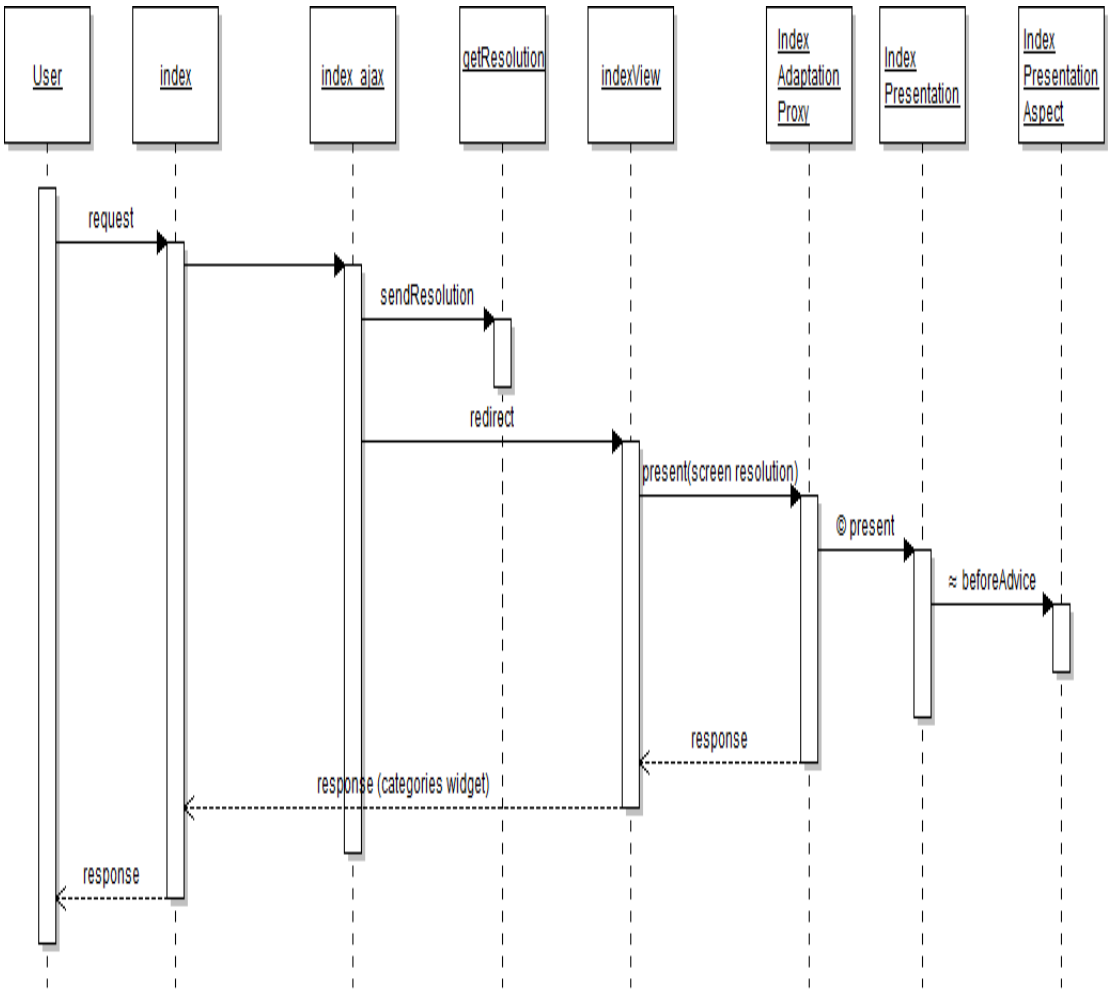
Πίνακας 3: Context παράμετροι για προτιμήσεις χρήστη, διαθεσιμότητα, τοποθεσία

3.4 Δυναμική εκτέλεση κανόνων προσαρμοστικότητας

Στην ενότητα αυτή αναλύουμε τον τρόπο με τον οποίο οι συνιστώσες της προτεινόμενης αρχιτεκτονικής (Εικόνα 6) αλληλεπιδρούν ανά λειτουργία για την προσαρμογή τους με βάση τις περιβάλλουσες πληροφορίες.

Εισαγωγή στο σύστημα

Η αλληλεπίδραση όλων των συνιστωσών κατά την εισαγωγή του χρήστη στο σύστημα απεικονίζεται στην Εικόνα 10.



Εικόνα 10: Index Sequence Diagram

Τα βήματα που εκτελούνται στην εισαγωγή στο σύστημα είναι τα παρακάτω:

1. Ο χρήστης επισκέπτεται την εισαγωγική σελίδα της εφαρμογής (index.jsp)
2. Το ενσωματωμένο javascript που εφαρμόζει την τεχνολογία ajax (index_ajax.js) λαμβάνει τις διαστάσεις της οθόνης στη συσκευή του χρήστη και τις στέλνει στο jsp script του εξυπηρέτη (getResolution.jsp) για τη καταγραφή αυτής της πληροφορίας.
3. Κατόπιν καλείται ο εξυπηρέτης της εφαρμογής (indexView.jsp) να προσαρμόσει την παρουσίαση της σελίδας με βάση την προσχεδιασμένη δομή (Εικόνα 5) και την περιβάλλουσα πληροφορία (διαστάσεις οθόνης).
4. Ο εξυπηρέτης της εφαρμογής (indexView.jsp) καλεί μέσω του αντικειμένου *IndexAdaptationProxy* το JavaBean *IndexPresentation.* Συγκεκριμένα καλεί τη μέθοδο present που επιστρέφει το περιεχόμενο της σελίδας προς το χρήστη.
5. Στην κλήση *IndexPresentation:present(width, height)* μέσα στο αντικείμενο *IndexAdaptationProxy* ενεργοποιείται ο **J**Spring AOP μηχανισμός όπως εξηγείται παρακάτω:

```
public String present(int width, int height) {
   ApplicationContext appContext = new
        FileSystemXmlApplicationContext("classpath:./springconfi
        g.xml");
IndexInterface indexInterface = (IndexInterface)
     appContext.getBean("proxyBean");
return indexInterface.present(width, height);
}
```

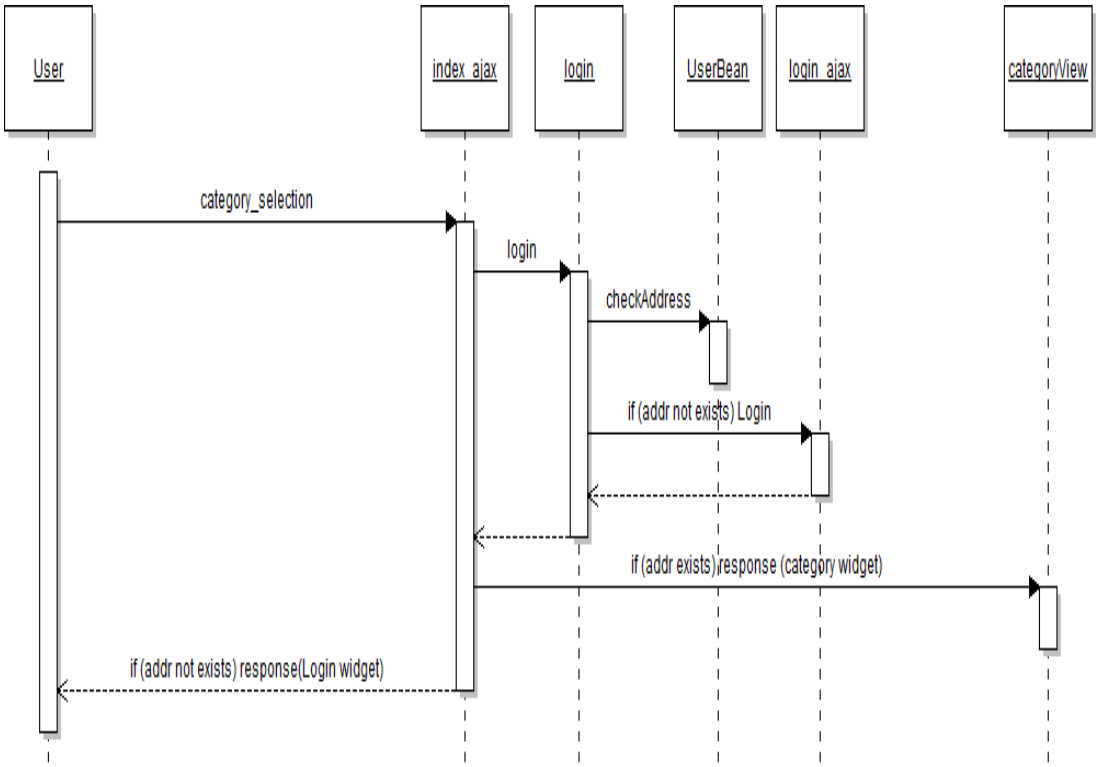
6. Σύμφωνα με το ApplicationContext που ορίζεται στο xml αρχείο *SpringConfig.xml* ορίζονται *σημεία συνένωσης/σημεία τομής* στην κλήση της μεθόδου και ορίζονται επίσης ποιες aspect κλάσεις (IndexPresentation*Aspect*) υλοποιούν κάποιο before/after/around advice.
7. Η μέθοδος beforeAdvice της *IndexPresentationAspect* κλάσης μεταφράζει τις παραμέτρους των διαστάσεων της οθόνης και ορίζει στο αντικείμενο *IndexPresentation* αν πρέπει να χρησιμοποιήσει το αρχείο κανόνων δημιουργίας cascade stylesheet για υψηλή ανάλυσης οθόνη ή για οθόνη χαμηλής ανάλυσης.

Επιλογή κατηγορίας

Στην πρώτη οθόνη που εμφανίζεται στη συσκευή του χρήστη συλλέγονται οι επόμενες περιβάλλουσες πληροφορίες που θα καθορίσουν την προσαρμογή των οθονών (λειτουργιών) που ακολουθούν. Σύμφωνα με το διάγραμμα στην Εικόνα 11 συλλέγονται τα παρακάτω στοιχεία:

1. Προτίμηση του χρήστη για εστιατόρια ή φαρμακεία (category_selection: ενεργοποίηση επιλογής στην συνιστώσα που εμφανίζεται στο κύριο μέρος της πρώτης σελίδας).
2. IP διεύθυνση (address) της διασύνδεσης της συσκευής του χρήστη με την εφαρμογή
3. Το index_ajax.js (διερμηνέας περιβάλλουσας πληροφορίας) στέλνει την προτίμηση του χρήστη και την IP διεύθυνση στον εξυπηρέτη (login.jsp).
4. Το script JSP αποφασίζει με τη βοήθεια του αντίστοιχου bean (UserBean: checkAddress) αν ο χρήστης είναι συχνός επισκέπτης οπότε προωθείται αμέσως στη φόρμα αναζήτησης με βάση την προτίμηση του ή αν εμφανίζεται για πρώτη φορά στη

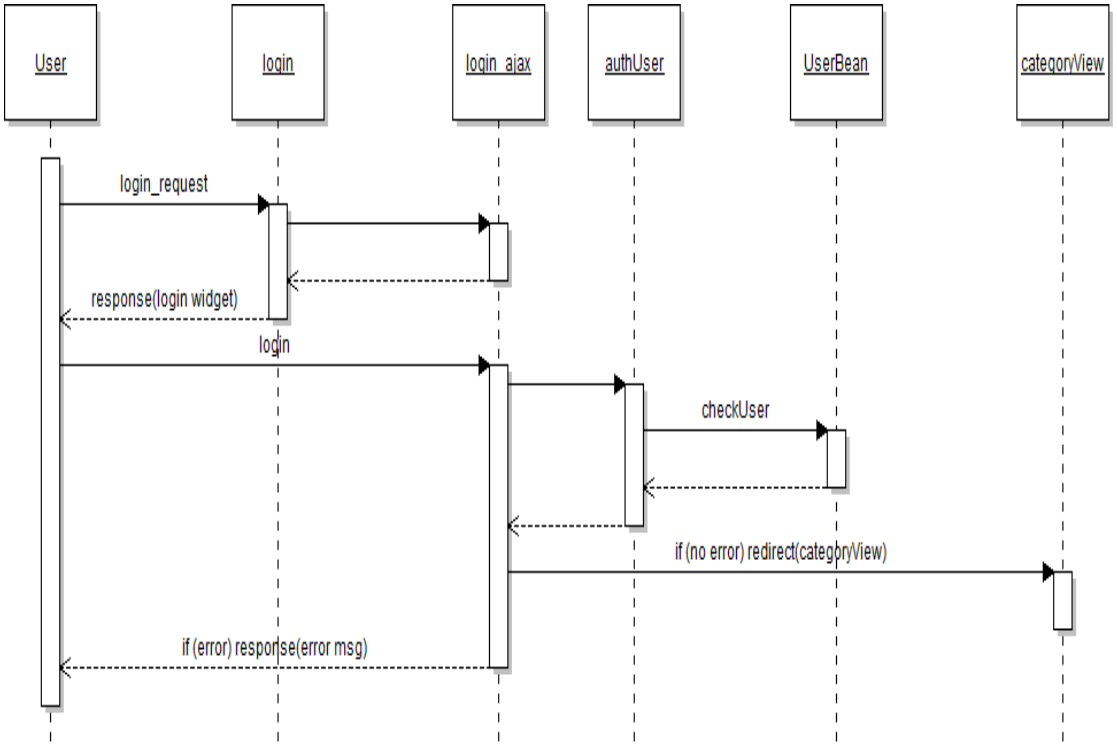
σελίδα θα πρέπει να προωθηθεί στη σελίδα εισαγωγής χρήστη (login.jsp).



Εικόνα 11: Category Selection Sequence Diagram

Εισαγωγή χρήστη

Οι νέοι χρήστες πρέπει να αναγνωριστούν από το σύστημα μέσα από μία φόρμα καταχώρησης ονόματος χρήστη και κωδικού πρόσβασης (login.jsp). Το login_ajax μεσολαβεί για την επικύρωση της αναγνώρισης των στοιχείων του χρήστη από τον εξυπηρέτη (authUser.jsp) ή για τη δυναμική εμφάνιση στην οθόνη του αντίστοιχου μηνύματος λάθους σε περίπτωση που παρουσιαστεί σφάλμα στην αναγνώριση. Κατά την ορθή επικύρωση των στοιχείων του χρήστη, ο έλεγχος μεταφέρεται (από το login_ajax.js) στην οθόνη αναζήτησης (categoryView.jsp).



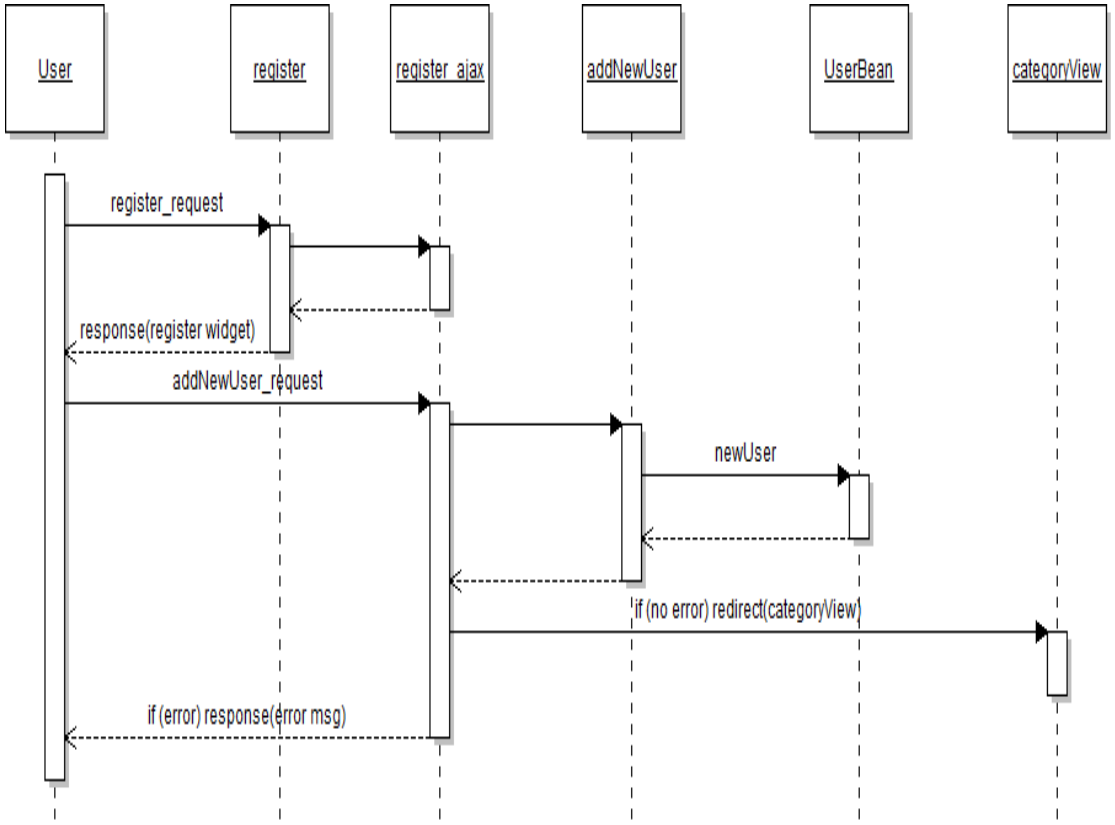
Εικόνα 12: Login Sequence Diagram

Όπως αναφέρθηκε στη προηγούμενη ενότητα, ορίζεται και για τη λειτουργία αυτή μία AdaptationProxy κλάση (*LoginAdaptationProxy*) για τη δυναμική δημιουργία του cascade stylesheet περιεχομένου της φόρμας εισαγωγής. Και αυτή η οντότητα επαναχρησιμοποιεί το ApplicationContext (springconfig.xml) που εφαρμόζεται επάνω στη μέθοδο *present*.

Καταχώρηση νέου χρήστη

Οι χρήστες που δεν έχουν αναγνωριστεί από το σύστημα, τους δίνεται η δυνατότητα εγγραφής τους μέσα από μία φόρμα καταχώρησης των βασικών τους στοιχείων πέρα από το όνομα χρήστη και τον κωδικό πρόσβασης (register.jsp). Το register_ajax μεσολαβεί για την επικύρωση των στοιχείων του χρήστη πριν αυτά καταλήξουν στον εξυπηρέτη (addNewUser.jsp) για την αποθήκευση τους στη βάση δεδομένων. Το register_ajax μεσολαβεί επίσης για τη δυναμική εμφάνιση στην οθόνη του αντίστοιχου μηνύματος λάθους σε

περίπτωση που παρουσιαστεί σφάλμα στην καταχώριση (π.χ. το όνομα χρήστη ήδη υπάρχει). Κατά την ορθή επικύρωση και αποθήκευση των στοιχείων του χρήστη, η ροή μεταφέρεται στην οθόνη αναζήτησης (categoryView.jsp).



Εικόνα 13: Register Sequence Diagram

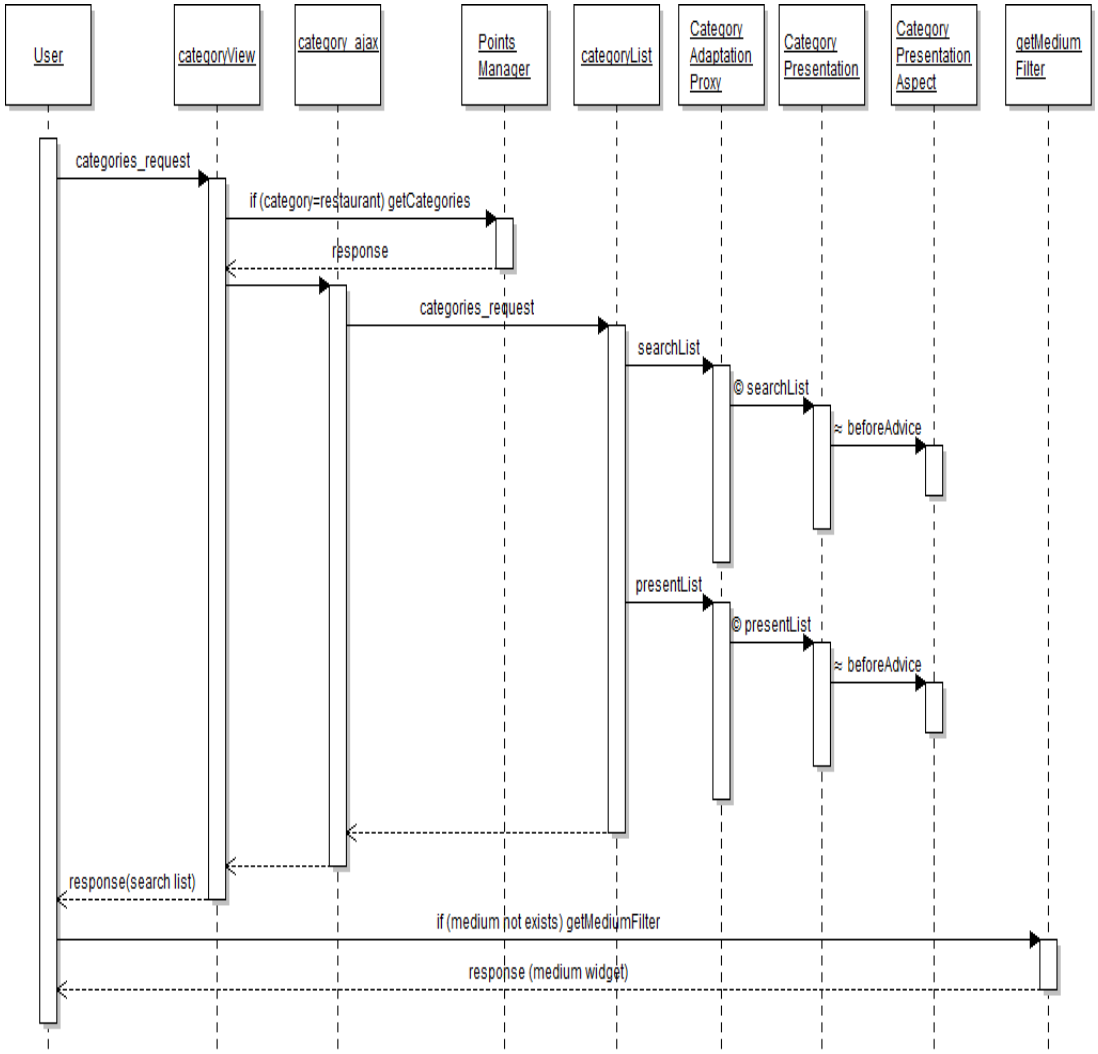
Οθόνη φίλτρων αναζήτησης και εμφάνισης αποτελεσμάτων

Η οθόνη αυτή είναι η κύρια οθόνη που χρησιμοποιεί ο χρήστης για την πλοήγηση του στα διαθέσιμα εστιατόρια ή φαρμακεία. Όπως αναφέρθηκε παραπάνω (Πίνακας 3) η οθόνη αυτή προσαρμόζει το περιεχόμενο και την εμφάνιση των αποτελεσμάτων τους σύμφωνα με τα δεδομένα που συλλέγονται από τα φίλτρα αναζήτησης (Εικόνα 14):

1. Εξετάζοντας τις προτιμήσεις των χρηστών η σελίδα εμφανίζει δυναμικά στο αριστερό μέρος τις κατηγορίες των εστιατορίων (αν ο χρήστης έχει επιλέξει εστιατόρια).

2. Η επιλογή μίας κατηγορίας εστιατορίων είναι άλλη μία προτίμηση που χειρίζεται το category_ajax και μεταφέρει στον εξυπηρέτη (categoryList.jsp). Στην περίπτωση των φαρμακείων δεν υπάρχει αυτή η επιλογή.
3. Το επιλεγμένο jsp script του εξυπηρέτη καλεί μέσω του *CategoryAdaptationProxy* δύο μεθόδους: α) για την συλλογή του κατάλληλου περιεχομένου από τη βάση δεδομένων (*searchList*) και β) για την προβολή του περιεχομένου (*presentList*) στο κεντρικό μέρος της σελίδας αναζήτησης.
4. Στην προκειμένη περίπτωση των δύο μεθόδων το JSpring πλαίσιο υλοποίησης της aspect κλάσης *CategoryAdaptationAspect* βασίζεται στο αρχείο categoryListconfig.xml. Το αρχείο περιέχει τους κανόνες που ορίζουν τα αντίστοιχα σημεία τομής και συμβουλές των δύο παραπάνω μεθόδων.
5. Η μέθοδος beforeAdvice που εφαρμόζεται στη μέθοδο searchList λαμβάνει υπόψη τις προτιμήσεις του χρήστη και τις επιλογές του στην αντίστοιχη συνιστώσα με τις διαθέσιμες κατηγορίες για να ορίσει το κατάλληλο sql query που θα εκτελέσει κατόπιν η μέθοδος searchList.
6. Λαμβάνεται υπόψη αν προϋπάρχουν στη βάση δεδομένων προτιμήσεις του χρήστη για συγκεκριμένα εστιατόρια ή φαρμακεία.
7. Κατόπιν ελέγχεται αν ο χρήστης έχει επιλέξει συγκεκριμένη κατηγορία εστιατορίου στη συγκεκριμένη διεπαφή (και μόνο για τη συγκεκριμένη επιλογή) διαφορετικά θεωρούνται όλες οι κατηγορίες.
8. Η μέθοδος beforeAdvice που εφαρμόζεται στη μέθοδο presentList λαμβάνει υπόψη τον αριθμό των στηλών που περιλαμβάνονται στις πλειάδες των αποτελεσμάτων για να κατευθύνει τον σχηματισμό του πίνακα πάνω στον οποίο προβάλλονται τα αποτελέσματα.

9. Αμέσως μετά την εμφάνιση των πρώτων διαθέσιμων αποτελεσμάτων εμφανίζεται ένα επιπλέον φίλτρο για την μετακίνηση του χρήστη με αυτοκίνητο ή αν είναι πεζός. Το φίλτρο παραμένει σταθερό και ενεργοποιείται αυτόματα κάθε τρία λεπτά προσομοιώνοντας τις δυνατότητες ενός συστήματος πλοήγησης.

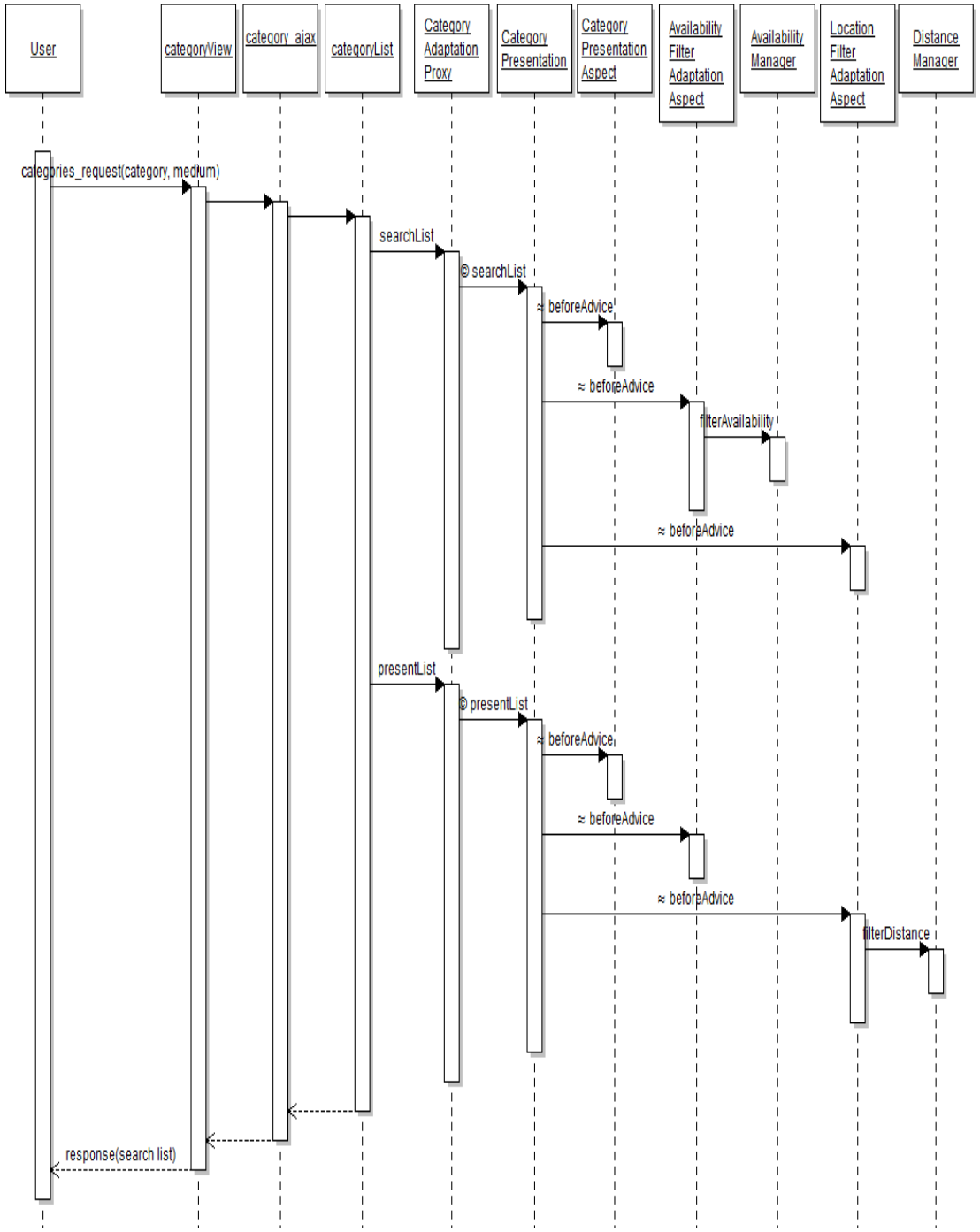


Εικόνα 14: Search Sequence Diagram

Το διάγραμμα στην Εικόνα 15 είναι επέκταση του παραπάνω διαγράμματος για την απεικόνιση της εφαρμογής του φίλτρου διαθεσιμότητας και γεωγραφικής απόστασης.

Δύο διαφορετικές aspect κλάσεις καλούνται να διαχειριστούν την προσαρμογή των μεθόδων searchList και presentList με βάση αυτά τα δύο κριτήρια.

1. Για τη πρώτη μέθοδο η κλάση AvailabilityFilterAdaptationAspect εξετάζει τις ημέρες και ώρες διαθεσιμότητας του εκάστοτε εστιατορίου ή φαρμακείου σε σχέση με την τρέχουσα ώρα.
2. Για τη δεύτερη μέθοδο η ίδια κλάση χρησιμοποιεί την προηγούμενη σύγκριση για να προβάλει την πληροφορία δίπλα σε κάθε εστιατόριο ή φαρμακείο για τον αν είναι ανοιχτό ή κλειστό τη δεδομένη χρονική στιγμή
3. Για την πρώτη μέθοδο η κλάση LocationFilterAdaptationAspect εξετάζει τη γεωγραφική απόσταση του χρήστη σε σχέση με την τοποθεσία του υποψήφιου σημείου αναζήτησης. Η απόσταση υπολογίζεται εκείνη τη στιγμή σύμφωνα με συγκεκριμένο αλγόριθμο (*Spherical Earth Model*) που λαμβάνει υπόψη το γεωγραφικό πλάτος (latitude) και μήκος (longitude) του κάθε σημείου και επιστρέφει την απόσταση σε χιλιόμετρα ή άλλη μονάδα μέτρησης απόστασης.
4. Για την περίπτωση του χρήστη λαμβάνεται υπόψη μία σταθερή τιμή γεωγραφικού πλάτους και μήκους (π.χ. το κέντρο της Σπάρτης) αλλά θα μπορούσε να είναι η τιμή που επιστρέφει μία GPS συσκευή και να αποθηκεύεται στο προφίλ του.
5. Για τη δεύτερη μέθοδο η παραπάνω κλάση χρησιμοποιεί την υπολογισθείσα απόσταση για να απεικονίσει ή όχι το σημείο αναζήτησης στον πίνακα αποτελεσμάτων. Για παράδειγμα θα πρέπει να είναι σε απόσταση μικρότερη του ενός χιλιομέτρου αν ο χρήστης είναι πεζός διαφορετικά θα πρέπει να είναι σε απόσταση μικρότερη των πέντε χιλιομέτρων αν είναι με το αυτοκίνητο.



Εικόνα 15: Search Sequence Diagram: presentation, content, operation crosscuts

4. Σύγκριση με άλλες Αρχιτεκτονικές

Η αρχιτεκτονική προσαρμοστικότητας εφαρμογών κινητού εμπορίου που προτείνεται σε αυτή την εργασία έχει επηρεαστεί σε μεγάλο βαθμό από την αρχιτεκτονική που προτείνεται από τους κ. Μπένου και κ. Βασιλάκη [3, 14]. Συγκρίνοντας τις δύο αρχιτεκτονικές (Εικόνα 4, Εικόνα 6) κοινή αναφορά είναι ότι οι ενότητες της υλοποίησης πολιτικών προσαρμοστικότητας εφαρμόζουν Aspect-Oriented Programming. Επιπλέον, και στις δύο περιπτώσεις προσαρμοστικότητας εφαρμόζεται ένα request/response μοντέλο, δηλαδή η προσαρμογή εκτελείται αφού ο εξυπηρέτης λάβει σχετικό αίτημα από τον client. Η αρχιτεκτονική του Context Manager προτείνει ένα επιπλέον μοντέλο (push) [3, 14] για την αυτόματη πρωτοβουλία του εξυπηρέτη να εκτελέσει ενέργειες προσαρμοστικότητας ως απόκριση σε δυναμικές καταστάσεις (events).

Η κύρια διαφορά ανάμεσα στις δύο αρχιτεκτονικές είναι ότι ο Context Manager βασίζεται σε μία λίστα από πολιτικές προσαρμοστικότητας οι οποίες αποθηκεύονται ως αρχεία με κανόνες της μορφής *Condition, Action* (πχ. *If* condition *x* is met *then* action *y* is triggered). Οι συνθήκες αφορούν περιβάλλουσα πληροφορία (context) και οι ενέργειες προσαρμόζουν τα τρία σκέλη της εφαρμογής: παρουσίαση, περιεχόμενο, λειτουργίες. Το μοντέλο αυτό προϋποθέτει την υλοποίηση ενός *Code Generator* για την μετατροπή των CA κανόνων σε aspect κώδικα είτε στατικά (πριν την εκτέλεση της εφαρμογής) ή δυναμικά.

Αντίθετα στην αρχιτεκτονική που προτείνεται σε αυτή την εργασία οι πολιτικές προσαρμοστικότητας έχουν ήδη υλοποιηθεί σε aspect κλάσεις. Ωστόσο, υπάρχουν αρχεία παραμετροποίησης που επιτρέπουν την ευέλικτη χρήση τους ή ακόμα και αντικατάσταση τους από εναλλακτικές aspect κλάσεις ακόμα και σε χρόνο εκτέλεσης. Αυτό

είναι εφικτό με τη χρήση των μηχανισμών του JSpring (computational reflections).

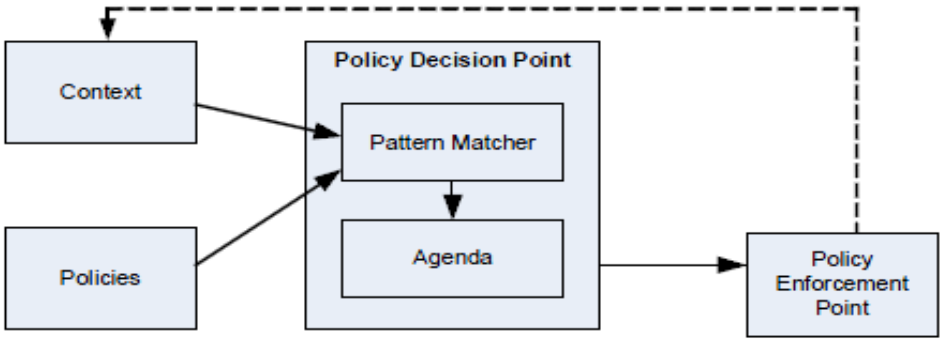
Στην αρχιτεκτονική, MyCAMPUS Agents [17], από το πανεπιστήμιο του Carnegie Mellon, πράκτορες λογισμικού(agent) είναι αυτόνομα προγράμματα που συλλέγουν πληροφορίες για τους χρήστες και τους προτείνουν επιλογές στα πλαίσια μίας εφαρμογής. Για τη συλλογή πληροφοριών από το περιβάλλον οι πράκτορες λογισμικού στηρίζονται επίσης σε επικοινωνία με υπηρεσίες *web*. Σε μία παρόμοια εφαρμογή που έχει υλοποιηθεί ("restaurant concierge") οι πράκτορες λογισμικού προτείνουν στους χρήστες εστιατόρια λαμβάνοντας υπόψη τις προτιμήσεις τους, *πόσο χρόνο έχουν στη διάθεση τους*, τη τοποθεσία και τον καιρό. Για παράδειγμα, αν έξω βρέχει θα προτιμηθεί η κοντινότερη απόσταση ή κάποιο μέρος στο χώρο του πανεπιστημίου.

Επιπλέον στοιχείο στην αρχιτεκτονική αυτή είναι η συλλογή όλης της περιβάλλουσας πληροφορίας για ένα χρήστη σε μία ενιαία δομή (e-wallet) που λειτουργεί ως ένα ευρετήριο όλων αυτών των στοιχείων (π.χ. ημερολόγιο, τωρινή τοποθεσία, προτιμήσεις ανά κατηγορία, υπηρεσίες ως συνδρομητής, κλπ.). Τα δεδομένα στο ευρετήριο ενημερώνονται με την χρήση υπηρεσιών web(π.χ. GPS/WiFi tracking, weather web service). Οι πράκτορες λογισμικού ανακτούν σε χρόνο εκτέλεσης πληροφορίες από αυτό το ευρετήριο και τις χρησιμοποιούν για να προσαρμόσουν τις προσφερόμενες υπηρεσίες αναζήτησης (π.χ. εστιατορίων).

Ο ορισμός ενός ιδεατού ευρετηρίου *context* πληροφοριών είναι σημαντικός για την χρήση μίας συγκεκριμένης οντολογίας στην αρχιτεκτονική. Ωστόσο, δεν προκύπτει αν το μοντέλο των πρακτόρων λογισμικού είναι όσο αποτελεσματικό όσο η AOP στην εφαρμογή πολιτικών προσαρμοστικότητας σε διαφορετικού είδους εφαρμογές.

Μία άλλη αρχιτεκτονική προσαρμοστικότητας [18] για web εφαρμογές βασίζεται επίσης στη χρήση ενός ευρετηρίου πολιτικών και εφαρμογής

τους στην δυναμική προσαρμογή του περιεχομένου ή των λειτουργιών λαμβάνοντας υπόψη πληροφορίες για τη τωρινή κατάσταση του χρήστη, τη συσκευή, προτιμήσεις κ.ά. Όπως περιγράφεται στην Εικόνα 16, οι πολιτικές μεταφράζονται από την κεντρική μονάδα αποφάσεων (*Policy Decision Point*) και εφαρμόζονται επάνω σε περιβάλλουσα πληροφορία από τον *Pattern Matcher*. Η τελευταία ενότητα αποθηκεύει τις πολιτικές που ταιριάζουν σε ένα ευρετήριο (*Agenda*) και τις εκτελεί σειριακά. Οι ενέργειες που επιβάλλονται από κάθε πολιτική που ενεργοποιείται, εκτελούνται από την ενότητα *Policy Enforcement Point* (PEP) και συμβάλλουν στη τροποποίηση της συμπεριφοράς της εφαρμογής όπως παραπάνω.



Εικόνα 16: Βασικές συνιστώσες αρχιτεκτονικής προσαρμοστικότητας [18]

Η παραπάνω αρχιτεκτονική έχει πολλές ομοιότητες με τον **J**Spring μηχανισμό εφαρμογής του AOP. Το ApplicationContext (xml) αρχείο μέσα στο οποίο ορίζουμε τις aspect κλάσεις και τις συμβουλές/σημεία τομής που εφαρμόζονται επάνω σε μεθόδους συγκεκριμένων κλάσεων παρομοιάζεται με την ενότητα Pattern Matcher. Οι συμβουλές(advices) που ταιριάζουν σε ένα συγκεκριμένο περιεχόμενο(context) εκτελούνται σειριακά ενώ το περιεχόμενο των συμβουλών(advices) ορίζουν κατανεμημένα *policy enforcement points* σε αντίθεση με τη παραπάνω αρχιτεκτονική που ορίζει ένα κεντρικό PEP.

5. Συμπεράσματα – Μελλοντικές επεκτάσεις

Η εφαρμογή που χρησιμοποιήθηκε ως μελέτη χρηστικότητας της προτεινόμενης αρχιτεκτονικής επιτρέπει την αναζήτηση εστιατορίων και φαρμακείων λαμβάνοντας υπόψη τις επιλογές του χρήστη (π.χ. κατηγορία, τοποθεσία, μέσο μετακίνησης κλπ). Η εφαρμογή δοκιμάστηκε σε περιβάλλον υπολογιστή και κινητού τηλεφώνου επιδεικνύοντας προσαρμοστικότητα στα τρία επίπεδα που είχαν αναφερθεί: παρουσίαση, περιεχόμενο και λειτουργίες. Επιπλέον, επιδεικνύει τη δυναμική των τεχνολογιών υπηρεσιών web στη συλλογή και αξιοποίηση πληροφορίας περιεχομένου σε οποιοδήποτε πεδίο εφαρμογών.

Πολύ περισσότερο, η τεχνολογία Aspect-Oriented Programming συνέβαλλε στην διαχωρισμό των *context* εννοιών που ορίζουν το περιβάλλον προσαρμογής της εφαρμογής από τις κύριες λειτουργίες της. Οι έννοιες αυτές υλοποιούνται ως ανεξάρτητες και επεκτάσιμες οντότητες για την προσαρμογή της συμπεριφοράς της εφαρμογής σε συνδυασμό καταστάσεων που αφορούν τον χρήστη (π.χ. προτιμήσεις, διαθέσιμη συσκευή, δίκτυο, τοποθεσία). Ο AOP μηχανισμός που υλοποιήσαμε στην εφαρμογή αυτή (*Java Spring Framework*) λειτούργησε καταλυτικά για την υλοποίηση παραμετροποιημένων πολιτικών προσαρμοστικότητας στη πλευρά του εξυπηρέτη και ανεξάρτητα από τη συσκευή ή το δίκτυο από το οποίο ο χρήστης προσπελαύνει την εφαρμογή. Πράγματι κατά την εκτέλεση της εφαρμογής και με τη χρήση των πολιτικών αυτών φάνηκε ότι ο *JSpring* μηχανισμός λειτούργησε το ίδιο αποτελεσματικά όσο και ο *AspectJ* μεταγλωττιστής που χρησιμοποιείται ευρύτερα για την υλοποίηση AOP εφαρμογών.

Ο τρόπος με τον οποίο αξιοποιήσαμε την AOP τεχνολογία στα πλαίσια του Java Spring χωράει βέβαια σημαντικές επεκτάσεις. Μία από αυτές είναι η υλοποίηση ενός Aspect Code Generator παρόμοιο με αυτό που

ορίζεται στον Context Manager [3,14]. Αυτό θα επιτρέψει σε τελικούς χρήστες που δεν έχουν γνώσεις (AOP) προγραμματισμού να ορίσουν εκτελέσιμες πολιτικές προσαρμοστικότητας, για παράδειγμα στη μορφή Condition-Action. Αυτό θα πλησιάσει την αφηρημένη μορφή μίας αρχιτεκτονικής που υποστηρίζει την προσαρμοστικότητα όπως αυτή στην Εικόνα 7.

Επιπλέον, είναι χρήσιμο να οριστεί μία οντολογία για τον ορισμό των διαφορετικών κατηγοριών περιβάλλουσας πληροφορίας (context information). Θα επιτρέψει έτσι να δημιουργούνται ιδεατές δομές τέτοιων πληροφοριών ή να επιτρέψει την ταξινόμηση και ευκολότερο εντοπισμό context υπηρεσιών σε μία ευρύτερα κατανεμημένη αρχιτεκτονική (distributed context servers). Η γλώσσα (οντολογία) αυτή μπορεί να ενσωματωθεί στον Aspect Code Generator για τη δημιουργία των aspect κλάσεων.

Βιβλιογραφία

1. Francisco J. García, Fabio Paternò and Ana B. Gil. An Adaptive e-Commerce System Definition. In *Adaptive Hypermedia and Adaptive Web-Based Systems,* Lecture Notes in Computer Science, 2006, Volume 2347/2006, pages 505-509.
2. Filippo Menczer, Alvaro E. Monge, W. Nick Street. *A*daptive Assistants for Customized E-Shopping. *IEEE Intelligent Systems*, vol. 17, no. 6, pp. 12-19, Nov/Dec. 2002.
3. Benou Poulcheria, Vassilakis Costas. An Aspect-Oriented Approach for Adaptation of m-Commerce Applications. University of Peloponnese, Department of Computer Science and Technology.
4. Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report, Dartmouth TR2000-381.
5. Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pages 85-90, Santa Cruz, California, December 1994. IEEE Computer Society Press.
6. Gonzalo Camarillo, Miguel-Angel García-Martín. The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds. Wiley, 2004. SBN-10: 0470871563, pages 406.
7. Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to context than location. In *Proceedings of Workshop on Interactive Applications of Mobile Computing (IMC'98)*, Rostock, Germany, November 1998. Neuer Hochschulschriftverlag.

8. Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceeding of the CHI 99 Conference on Human Factors in Computing Systems: The CHI is the Limit*, pages 434-441, Pittsburgh, PA, May 1999. ACM Press.
9. Anind K. Dey, Daniel Salber, Masayasu Futakawa, and Gregory D. Abowd. An architecture to support context-aware applications. Technical Report GIT-GVU-99-23, Georgia Institute of Technology, College of Computing, June 1999.
10. Anind K. Dey and Gregory D. Abowd. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *In Journal Human-Computer Interaction*, Volume 16, Issue 2, December 2001.
11. Costas Vassilakis, George Lepouras, Spiros Skiadopoulos. Mobile and Context-Aware e-Commerce: Issues, Challenges and Research Directions. In Journal of Electronic Commerce, 2008.
12. R. Laddad. AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications Co, 2003, ISBN:1930110936.
13. P. Tarasewich, T. Bhimdi, and M. Dideles. Testing Visual Notification Cues on a Mobile Device. *Proceedings of CHI 2004*, Vienna, Austria, 1562.
14. Benou Poulcheria, Vassilakis Costas. The Conceptual Model of Context for Mobile Commerce Applications. *Journal of Electronic Commerce Research*, Vol. 10, Issue 2, June 2010.
15. AspectJ, http://eclipse.org/aspectj/
16. Java Spring Framework, http://www.springsource.org/
17. Norman M. Sadeh, Ting-chak Chan, Linh Van, Ohbyung Kwon and Kazuaki Takizawa. *Creating an Open Agent Environment for Context-aware M-Commerce*. *In Agentcities: Challenges in*

Open Agent Environments, 2003, pp: 152-158, Springer Verlag.

18. E. Rukzio, S. Siorpaes, O. Falke, H. Hussmann. Policy Based Adaptive Services for Mobile Commerce. The Second IEEE International Workshop on Mobile Commerce and Services, 2005 (WMCS '05), pages 183 – 192.

Παράρτημα Α’:
Αρχεία παραμετροποίησης πολιτικών προσαρμοστικότητας

Springconfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"


xsi:schemaLocation="http://www.springframework.org/schema/beans
s       http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
">


<!-- Bean configuration -->
  <bean                                                id="proxyBean"
class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
      <value>ecommerce.adaptation.IndexInterface</value>
    </property>
    <property name="target">
      <ref local="beanTarget" />
    </property>
```

```
    <property name="interceptorNames">
      <list>
        <value>theTracingBeforeAdvisor</value>
      </list>
    </property>
  </bean>

  <!-- Bean Classes -->

  <bean                                                id="beanTarget"
class="ecommerce.adaptation.IndexPresentation" />
  <!-- Advisor pointcut definition for before advice -->
  <bean id="theTracingBeforeAdvisor"

class="org.springframework.aop.support.RegexpMethodPointcutAdvis
or">
    <property name="advice">
      <ref local="theTracingBeforeAdvice" />
    </property>
    <property name="pattern">
      <value>.*present.*</value>
    </property>
  </bean>

  <!-- Advisor pointcut definition for after advice -->

  <!-- Advice classes -->
  <bean                                        id="theTracingBeforeAdvice"
class="ecommerce.adaptation.aspect.IndexAdaptationAspect" />
</beans>
```

CategoryListConfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"


xsi:schemaLocation="http://www.springframework.org/schema/bean
s        http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
         http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
">


<!-- Bean configuration -->
  <bean                                                id="proxyBean"
class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
      <value>ecommerce.adaptation.CategoryListInterface</value>
    </property>
    <property name="target">
      <ref local="beanTarget" />
    </property>
    <property name="interceptorNames">
      <list>
        <value>theTracingBeforeAdvisor</value>
        <value>theTracingBeforeAdvisor2</value>
        <value>theTracingBeforeAdvisor3</value>
      </list>
```

```
    </property>
  </bean>

  <!-- Bean Classes -->

  <bean                                                    id="beanTarget"
class="ecommerce.adaptation.CategoryPresentation" />
  <!-- Advisor pointcut definition for before advice -->
  <bean id="theTracingBeforeAdvisor"

class="org.springframework.aop.support.RegexpMethodPointcutAdvis
or">
    <property name="advice">
      <ref local="theTracingBeforeAdvice" />
    </property>
    <property name="patterns">
       <list>
      <value>.*searchList.*</value>
      <value>.*presentList.*</value>
      </list>
    </property>
  </bean>

<bean id="theTracingBeforeAdvisor2"

class="org.springframework.aop.support.RegexpMethodPointcutAdvis
or">
    <property name="advice">
      <ref local="theTracingBeforeAdvice2" />
    </property>
    <property name="pattern">
```

```
      <value>.*presentList.*</value>
    </property>
  </bean>

  <bean id="theTracingBeforeAdvisor3"

class="org.springframework.aop.support.RegexpMethodPointcutAdvis
or">
    <property name="advice">
      <ref local="theTracingBeforeAdvice3" />
    </property>
    <property name="pattern">
      <value>.*presentList.*</value>
    </property>
  </bean>

  <!-- Advisor pointcut definition for Before advice -->

  <!-- Advice classes -->
  <bean                                    id="theTracingBeforeAdvice"
class="ecommerce.adaptation.aspect.CategoryAdaptationAspect" />

  <bean                                  id="theTracingBeforeAdvice2"
class="ecommerce.adaptation.aspect.LocationFilterAdaptationAspect"
/>

  <bean                                  id="theTracingBeforeAdvice3"
class="ecommerce.adaptation.aspect.AvailabilityFilterAdaptationAspe
ct" />
```

```
</beans>
```

HighResIndex.cfg

```
<STYLE type="text/css">
h2.h2class {}

div.headclass {  background-image: url(./img/top.png); color:white }

div.topclass {  background-image: url(./img/top.png); color:white }
div.leftclass {position:absolute; top:140px;

              width:130px;
              height:100%;
              background-image: url(./img/pic2.jpg)
             }

div.centerclass { background-image: url(./img/pic2.jpg);
               position:absolute; left:140px; top:140px;
               width: 89%;
               height: 600px;
              }

div.bottomclass {background-image: url(./img/top.png); color:white;
                 position:absolute; top: 600px;
                 width: 100%;
                 height: 40px;
    }


tr.trclass1 {background-image: url(./img/td1.png)}

tr.trclass2 {background-image: url(./img/td2.png)}
```

```
td.tdclass1 {width:100px}

td.lastname {width:200px}
td.address {width:300px}

a.menuclass { color: white }

p.errorclass { color: red }


input.rbclass {}

button.btclass {}

<div.divres {overflow:auto; height:556px; width:564px;}

table.tblres {border: 1}
tr.trres {background-image: url(./img/td1.png)}

</style>
```

LowResIndex.cfg

```
<STYLE type="text/css">
h2.h2class {font-size: 32px;}

div.headclass {  background-image: url(./img/top.png); color:white;
```

```
            font-size: 32px;
             height: 140px;
             width: 100%;
            }

div.topclass {  background-image: url(./img/top.png); color:white;
            font-size: 24px;
             width: 785px;
            position:absolute; top:140px;
            }

div.leftclass {position:absolute; top:180px;
            font-size: 24px;
            width:160px;
            height: 765px;
            background-image: url(./img/left.png)
           }

div.centerclass { background-image: url(./img/center.png);
               position:absolute; left:170px; top:180px;
               font-size: 28px;
               width: 620px;
               height: 765px;
           }

div.bottomclass {background-image: url(./img/top.png); color:white;
              position:absolute; top: 950px;
              width: 100%;
              height: 40px;
              font-size: 32px
   }
```

```
tr.trclass1 {background-image: url(./img/td1.png)}

tr.trclass2 {background-image: url(./img/td2.png)}

td.tdclass1 {width:100px}

a.menuclass { color: white }

p.errorclass { color: red }

input.rbclass {}

button.btclass {}

<div.divres {overflow:auto; height:556px; width:564px;}

table.tblres {border: 1px; }
tr.trres {background-image: url(./img/td1.png); border:1px}


</style>
```

CategoryAdaptationAspect.java

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

import ecommerce.adaptation.CategoryPresentation;
import java.lang.reflect.Method;
import java.util.List;
import org.springframework.aop.MethodBeforeAdvice;

/**
 *
 * @author home
 */


public class CategoryAdaptationAspect implements MethodBeforeAdvice {


  public void before(Method method, Object[] args, Object target) throws Throwable {

        if (method.getName().equals("searchList"))
           searchListAdvice(args, target);


        else if (method.getName().equals("presentList"))
```

```
            presentListAdvice(args, target);


    }

   void searchListAdvice(Object [] args, Object target) {
         String category = args[0].toString();
             int user_id = (Integer) args[1];
            String option= args[2].toString();

             String sql= null;

             CategoryPresentation instance= ( CategoryPresentation) target;

         if (category.equals("restaurant")) {

              if (option== null) {


                  sql= "select r.name, r.street_name, r.postcode, g.city, rc.category, g.geographic_latitude, g.geographic_longtitude ";
                  sql = sql + " from restaurants r, preference_restaurant pr, restaurant_category rc, gps g ";
                  sql = sql + " where r.id = pr.restaurant_id and r.category = rc.id ";
                  sql = sql + " and r.gps_fk = g.id and pr.user_id = " + user_id;
                  sql = sql + " order by rc.category, g.city";
              }
```

```
            else if (option.equals("-1")) {
              sql= "select r.name, r.street_name, r.postcode, g.city,
rc.category, g.geographic_latitude, g.geographic_longtitude ";
               sql = sql + " from restaurants r, restaurant_category rc,
gps g ";
               sql = sql + " where r.category = rc.id ";
               sql = sql + " and r.gps_fk = g.id";
                sql = sql + " order by rc.category, g.city";
             }
            else {
              sql= "select r.name, r.street_name, r.postcode, g.city,
rc.category, g.geographic_latitude, g.geographic_longtitude ";
               sql = sql + " from restaurants r, restaurant_category rc,
gps g ";
               sql = sql + " where r.category = rc.id ";
               sql = sql + " and r.gps_fk = g.id and r.category = " +
option;
                sql = sql + " order by rc.category, g.city";
             }


       }

       else  if (category.equals("medshop")) {


          if (option== null) {

            sql= "select m.owner, m.street_name, m.postcode, g.city,
g.geographic_latitude, g.geographic_longtitude ";
```

```
          sql = sql + " from medshops m, preference_medshop pm,
gps g ";
          sql = sql + " where m.id = pm.medshop_id ";
          sql = sql + " and m.gps_fk = g.id and pm.user_id = " +
user_id;
          sql = sql + " order by g.city ";
         }
         else if (option.equals("-1")) {
             sql= "select m.owner, m.street_name, m.postcode,
g.city, g.geographic_latitude, g.geographic_longtitude ";
          sql = sql + " from medshops m, gps g ";
          sql = sql + " where m.gps_fk = g.id ";
          sql = sql + " order by g.city ";
         }



      }
       instance.setQueryString(sql);
  }

  void presentListAdvice(Object [] args, Object target) {
     String category=args[1].toString();
    CategoryPresentation instance= ( CategoryPresentation) target;
    List list= (List) args[0];
     int length=0;

     if ((list != null) && (list.size() >0)) {
         String [] data= (String []) list.get(0);
         length= data.length;
     }
```

```
        instance.setDataLength(length);

        if (category.equals("restaurant"))
        instance.hasCategory(true);
        else
            instance.hasCategory(false);
    }

}
```