Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Συστημάτων Μετάδοσης Πληροφορίας
και Τεχνολογίας Υλικών

Επέκταση πλατφόρμας κινητών αντιπροσώπων με χρήση δικτυακών υποδοχών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Θεόδωρος Ε. Αθαναηλέας

Επιβλέπων : Δήμητρα-Θεοδώρα Ι. Κακλαμάνη
Αν. Καθηγήτρια

Αθήνα, Ιούνιος 2005

Περίληψη

Σκοπός της συγκεκριμένης εργασίας είναι η επέκταση μιας ήδη υπάρχουσας πλατφόρμας κινητών αντιπροσώπων ώστε να γίνει δυνατή η επικοινωνία μεταξύ των συστατικών της αλλά και των αντιπροσώπων με διάφορα πρωτόκολλα. Η πλατφόρμα προσφέρει τις βασικές λειτουργίες για τη διαχείριση κινητών αντιπροσώπων, όπως η δυνατότητα φιλοξενίας αυτόνομων κινητών αντιπροσώπων, οι οποίοι έχουν δυνατότητα μετακίνησης, καθώς και δυνατότητα αναζήτησης και επικοινωνίας των κινητών αντιπροσώπων.

Στα πλαίσια της εργασίας προστέθηκε στην πλατφόρμα η δυνατότητα επικοινωνίας μέσω απλών δικτυακών υποδοχών (sockets), καθώς επίσης και ένας ευέλικτος μηχανισμός μεσισμικού που προσφέρει διαφανή επίτευξη επικοινωνίας μεταξύ των οντοτήτων που επικοινωνούν σε μια κατανεμημένη εφαρμογή, ανεξάρτητα από το πρωτόκολλο που χρησιμοποιείται. Τέλος, αναπτύχθηκε μία γραφική διεπαφή χρήστη για την διαχείριση της πλατφόρμας.

Λέξεις Κλειδιά

Κινητοί αντιπρόσωποι, κινητός κώδικας, sockets, Java, μεσισμικό, κατανεμημένη επεξεργασία, τεχνολογίες κατανεμημένων αντικειμένων.

Abstract

The objective of this thesis was the expansion of an already existing mobile agent platform in order to achieve multi-protocol communication between the platform's components and the agents. The platform offers the basic functionalities for mobile agents management, such as the ability to host agents that can move to a different location, as well as search and communication interfaces.

The platform was enriched with the capability of communicating through plain sockets and also with a flexible middleware mechanism that provides a transparent way of communication between the entities of a distributed application, independently of the protocol used. Lastly, a graphical user interface was created for the management of the platform.

Keywords

Mobile agents, mobile code, sockets, Java, middleware, distributed computing, distributed objects technology.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Συστημάτων Μετάδοσης Πληροφορίας
και Τεχνολογίας Υλικών

Επέκταση πλατφόρμας κινητών αντιπροσώπων με χρήση κατανεμημένων αντικειμένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Θεόδωρος Ε. Αθαναηλέας

Επιβλέπων : Δήμητρα-Θεοδώρα Ι. Κακλαμάνη
Αν. Καθηγήτρια

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28η Ιουνίου 2005.
Αθήνα, 2005

.....	.....	.....
Ιάκωβος Σ. Βενιέρης Καθηγητής	Δήμητρα-Θεοδώρα Ι. Κακλαμάνη Αν. Καθηγήτρια	Νικόλαος Ουζούνογλου Καθηγητής

.....
Θεόδωρος Ε. Αθαναηλέας

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περιεχόμενα

Κατάλογος Σχημάτων

1. Εισαγωγή

Τα τελευταία χρόνια η έρευνα στο πεδίο των πλατφορμών κινητών αντιπροσώπων έχει επικεντρωθεί στην εκμετάλλευση ήδη υπαρχόντων πλατφορμών ή στην ανάπτυξη νέων πλατφορμών για διαφορετικά περιβάλλοντα, όπως κινητές συσκευές, δικτυακούς εξυπηρετητές ή ακόμη και υπολογιστικά πλέγματα. Αυτές οι πλατφόρμες χρησιμοποιούν κοινά κανάλια επικοινωνίας, τα οποία προκύπτουν από τους περιορισμούς του περιβάλλοντος εκτέλεσης (π.χ. το HTTP για τις κινητές συσκευές). Πολλές αρχιτεκτονικές και γλώσσες επικοινωνίας μεταξύ αντιπροσώπων έχουν προταθεί για την επίτευξη αλληλεπίδρασης με άλλες πλατφόρμες, όμως οι περισσότερες πλατφόρμες σπάνια χρησιμοποιούν περισσότερα από ένα πρωτόκολλα επικοινωνίας. Αυτό βάζει κάποια εμπόδια στην τεχνολογία των κινητών πρακτόρων, λόγω της έλλειψης διαλειτουργικότητας μεταξύ πλατφορμών που τρέχουν πάνω σε διαφορετικά περιβάλλοντα.

Η συγκεκριμένη εργασία αποτελεί επέκταση μιας υλοποιημένης σε Java πλατφόρμας κινητών αντιπροσώπων, που έγινε στα πλαίσια προηγούμενης διπλωματικής εργασίας. Η πλατφόρμα αυτή προσφέρει τις βασικές υπηρεσίες που χρειάζονται για την ανάπτυξη εφαρμογών με χρήση κινητών αντιπροσώπων, όπως δυναμικό φόρτωμα αντιπροσώπων, υπηρεσίες κινητικότητας, επικοινωνίας και αναζήτησης. Για την επικοινωνία μεταξύ των οντοτήτων της πλατφόρμας και μεταξύ των πρακτόρων η υπάρχουσα υποδομή έκανε χρήση των Υπηρεσιών Διαδικτύου και ιδιαίτερα του πρωτοκόλλου SOAP (Simple Object Access Protocol). Σκοπός της παρούσας εργασίας είναι η ενσωμάτωση ευέλικτων μηχανισμών που επιτρέπουν την επικοινωνία μεταξύ των συστατικών της πλατφόρμας με διαφορετικά πρωτόκολλα και σε διαφορετικά κανάλια. Οι μηχανισμοί αυτοί είναι διαφανείς προς την πλατφόρμα, υπό την έννοια ότι ο τρόπος με τον οποίο μπορεί να επιτευχθεί επικοινωνία μεταξύ των οντοτήτων της είναι ο ίδιος ανεξάρτητα από το πρωτόκολλο που επιλέγεται κάθε φορά. Έτσι ο προγραμματιστής απαλλάσσεται από τον φόρτο του να γνωρίζει έναν ειδικό τρόπο επικοινωνίας για κάθε διαφορετικό πρωτόκολλο.

Ως υποκατάστατο του πρωτοκόλλου SOAP μπορούν να χρησιμοποιηθούν διάφορες άλλες τεχνολογίες κατανεμημένων αντικειμένων ή τεχνολογίες δικτυακού προγραμματισμού. Στη συγκεκριμένη εργασία έγινε χρήση των απλών δικτυακών υποδοχών (sockets). Σε επόμενο κεφάλαιο θα γίνει μια παρουσίαση των διαφόρων τεχνολογιών που μπορούν να χρησιμοποιηθούν για την επικοινωνία των οντοτήτων της πλατφόρμας.

Στα κεφάλαια που ακολουθούν θα γίνει μια εισαγωγή στην τεχνολογία κατανεμημένων αντικειμένων και παρουσίαση των σημαντικότερων τεχνολογιών που υπάρχουν διαθέσιμες. Επίσης, θα γίνει ξεχωριστή αναφορά στην τεχνολογία των κινητών πρακτόρων καθώς και παρουσίαση των τελευταίων εξελίξεων πάνω σε αυτήν την τεχνολογία. Στη συνέχεια θα περιγραφεί συνοπτικά η ήδη υπάρχουσα πλατφόρμα και θα γίνει αναλυτική περιγραφή της αρχιτεκτονικής μεσισιμικού που επιτρέπει την επικοινωνία μέσω πολλαπλών καναλιών και διαφορετικών πρωτοκόλλων. Τέλος, θα παρουσιαστεί το γραφικό περιβάλλον που αναπτύχθηκε για τη διαχείριση της πλατφόρμας και ένας οδηγός χρήστη, καθώς και ο οδηγός προγραμματιστή. Στο τελευταίο μέρος της εργασίας θα παρουσιαστεί η διεπαφή προγραμματισμού εφαρμογής (Application Programming Interface - API) της πλατφόρμας.

2. Δικτυακή Επικοινωνία

Ένα σύνολο υπολογιστών, οι οποίοι είναι συνδεδεμένοι μεταξύ τους με ένα φυσικό μέσο μεταφοράς, λέμε ότι αποτελούν ένα δίκτυο υπολογιστών. Η έννοια της επικοινωνίας μεταξύ των υπολογιστών είναι ένα από τα βασικά στοιχεία, πάνω στα οποία δομούνται οι κατανεμημένες υπολογιστικές αρχιτεκτονικές.

Η επικοινωνία μεταξύ των υπολογιστών πραγματοποιείται με την αποστολή πακέτων μεταξύ τους. Κάθε πακέτο περιέχει, εκτός από την πληροφορία που αποστέλλεται, τις διευθύνσεις του αποστολέα και του παραλήπτη, έτσι ώστε να είναι δυνατή η δρομολόγησή του μέσα στο δίκτυο. Στην περίπτωση που, το μέγεθος ενός πακέτου υπερβεί κάποιο καθορισμένο όριο, είναι δυνατός ο τεμαχισμός του σε μικρότερα κομμάτια, τα οποία επανασυντίθονται στον τελικό προορισμό.

Όπως σε κάθε περίπτωση επικοινωνίας, έτσι και στην επικοινωνία μεταξύ των υπολογιστών, είναι απαραίτητη η ύπαρξη κάποιων κανόνων, οι οποίοι ονομάζονται πρωτόκολλα. Ένα επικοινωνιακό πρωτόκολλο είναι στην ουσία ένα σύνολο από κανόνες, που περιγράφουν το πώς λαμβάνει χώρα η επικοινωνία σε ένα δίκτυο υπολογιστών και επιτρέπουν στα συστήματα υπολογιστών να ερμηνεύουν σωστά τα πακέτα, που λαμβάνουν από άλλα συστήματα.

Στη συνέχεια παρουσιάζονται κάποια βασικά χαρακτηριστικά και κάποιες βασικές έννοιες της δικτυακής επικοινωνίας.

2.1. Σύγχρονη και Ασύγχρονη Μετάδοση

Κατά τη διαδικασία αποστολής ενός πακέτου δεδομένων από μια διεύθυνση πηγής (source address) σε μια διεύθυνση προορισμού (destination address), διακρίνουμε δύο τρόπους μετάδοσης:

- Σύγχρονη μετάδοση: Κατά τη σύγχρονη μετάδοση πακέτων, ο αποστολέας ζητά επιβεβαίωση λήψης από τον κόμβο προορισμού κάθε πακέτου, που έστειλε, προτού αποστείλει το επόμενο πακέτο πληροφορίας.
- Ασύγχρονη μετάδοση: Κατά την ασύγχρονη μετάδοση, ο αποστολέας στέλνει πακέτα στο δίκτυο, χωρίς να περιμένει επιβεβαίωση λήψης για τα πακέτα αυτά. Όπως σε ένα ταχυδρομικό πακέτο, το ενδιαφέρον του αποστολέα για την λήψη ή μη, του πακέτου καθώς και οι ενέργειες, που εκτελούνται κατά τη λήψη του πακέτου, ποικίλουν. Υπάρχουν φορές, όπου ο αποστολέας δεν ενδιαφέρεται πότε ή εάν το πακέτο έφτασε στον προορισμό του. Σε άλλες περιπτώσεις ο αποστολέας επιζητεί την επιβεβαίωση λήψης του πακέτου, αλλά δεν χρειάζεται να λάβει την επιβεβαίωση, για να συνεχίσει να μεταδίδει πακέτα.

2.2. Οι έννοιες του Πελάτη, Εξυπηρετητή και Ομότιμης Οντότητας

Η λογική, στην οποία βασίζεται η κατανεμημένη επεξεργασία, είναι η υπόθεση ότι, μια εφαρμογή λογισμικού μπορεί να χωριστεί σε θεμελιώδεις συνιστώσες λογισμικού (modules), από τα οποία αποτελείται. Οι συνιστώσες αυτές δεν είναι απαραίτητο να εκτελούνται στον ίδιο χώρο μνήμης. Σύμφωνα με τη θεώρηση αυτή, προκειμένου να εκτελεστεί η εφαρμογή, θα πρέπει να υπάρχει επικοινωνία μεταξύ των κομματιών λογισμικού. Οι όροι "πελάτης", "εξυπηρετητής" και "ομότιμη οντότητα" χρησιμοποιούνται για να περιγράψουν τους ρόλους των συνιστωσών λογισμικού κατά τη διαδικασία εκτέλεσης της εφαρμογής. Οι συνιστώσες, οι οποίες ζητούν κάποια

υπηρεσία από άλλες συνιστώσες, ονομάζονται “πελάτες”, ενώ οι συνιστώσες οι οποίες προσφέρουν υπηρεσίες ονομάζονται “εξυπηρετητές”. Βέβαια, κατά την εκτέλεση της εφαρμογής, οι ρόλοι μπορούν να μεταβάλλονται. Για παράδειγμα, μια συνιστώσα λογισμικού μπορεί, σε κάποια περίπτωση, να λειτουργεί ως πελάτης ενώ, σε κάποια άλλη, ως εξυπηρετητής. Στην περίπτωση που, μια συνιστώσα λογισμικού δρα ταυτόχρονα ως πελάτης και ως εξυπηρετητής, αυτή χαρακτηρίζεται με τον όρο “ομότιμη οντότητα” (peer entity).

2.3. Τρόποι Δικτυακής Επικοινωνίας

2.3.1 Αποστολή Μηνυμάτων (Message Passing)

Ως μήνυμα ορίζουμε ένα πακέτο από δεδομένα με κατάλληλη ετικέτα, που φανερώνει την πληροφορία που περιέχει. Αυτό επιτρέπει, σε ένα ενδιάμεσο επίπεδο στον εξυπηρετητή να κατευθύνει το μήνυμα ή τα δεδομένα, που αυτό περιέχει, στον κατάλληλο αποδέκτη. Τα συστήματα μηνυμάτων μπορούν να λειτουργήσουν με βάση μια ασύγχρονη ή σύγχρονη αρχιτεκτονική. Λόγω του ότι η βασιζόμενη σε μηνύματα επικοινωνία είναι κατάλληλη για ενδιάμεση δρομολόγηση, τα χαρακτηριστικά αυτά μπορούν να συνδυαστούν, προκειμένου να παρέχουν ένα επίπεδο αφαίρεσης στην επικοινωνιακή υποδομή.

Κατά την ασύγχρονη λογική, τα μηνύματα αποθηκεύονται σε μια ουρά από τον εξυπηρετητή/δρομολογητή, από την οποία λαμβάνονται και διαβιβάζονται σε ένα ή περισσότερους λογικούς επεξεργαστές. Οι επεξεργαστές αυτοί μπορεί να μην αποκριθούν στα μηνύματα, ή να αποκριθούν απευθείας στον πελάτη. Παρόλα αυτά, προκειμένου να διατηρηθεί η αφαίρεση, μπορούν να στείλουν ένα μήνυμα πίσω στον εξυπηρετητή, μέσω μιας άλλης ουράς, το οποίο δρομολογείται πίσω στον πελάτη. Αντίθετα, κατά τη σύγχρονη μετάδοση μηνυμάτων ο εξυπηρετητής/δρομολογητής διαβιβάζει το μήνυμα στον επεξεργαστή, ο οποίος διαβιβάζει, με τη σειρά του, μια απόκριση στον εξυπηρετητή, για να επιστραφεί στον πελάτη. Υπάρχει ακόμη μια υβριδική κατάσταση λειτουργίας, στην οποία ο εξυπηρετητής συμπεριφέρεται ασύγχρονα, ενώ ο πελάτης συμπεριφέρεται σύγχρονα, επιτρέποντας στον εξυπηρετητή να έχει την αποδοτικότητα της ασύγχρονης λειτουργίας και στον πελάτη να επωφελείται από την απλότητα και την ασφάλεια της σύγχρονης επεξεργασίας.

2.3.2 Κλήση Απομακρυσμένων Διαδικασιών (Remote Procedure Call)

Η Κλήση Απομακρυσμένων Διαδικασιών (RPC) αποτελεί έναν δημοφιλή τρόπο ανάπτυξης εφαρμογών κατανεμημένης επεξεργασίας και, ουσιαστικά, σχετίζεται με την αλληλεπίδραση μεταξύ διαδικασιών, καθορίζοντας τους μηχανισμούς, που λαμβάνουν χώρα προκειμένου να κληθεί, μέσα από μια διεργασία, κάποια διαδικασία, η οποία εκτελείται σε ένα απομακρυσμένο σύστημα. Οι μηχανισμοί αυτοί θα πρέπει να φροντίσουν για τη σωστή μεταφορά των παραμέτρων κλήσης της απομακρυσμένης διαδικασίας, καθώς και για την επιστροφή των αποτελεσμάτων της στην καλούσα διεργασία (parameter marshalling/unmarshalling). Ουσιαστικά, σειριοποιούν τα δεδομένα κλήσης μιας συνάρτησης και τα μεταφέρουν στο απομακρυσμένο σύστημα, όπου και τα ανακατασκευάζουν. Η καλούσα διεργασία

αναμένει την επιστροφή από την κλήση της απομακρυσμένης μεθόδου, όπως ακριβώς συμβαίνει σε ένα πρόγραμμα δεδομενο-κεντρικού προγραμματισμού (data-oriented programming).

Σημαντικό ρόλο στην Απομακρυσμένη Κλήση Διαδικασιών διαδραματίζουν τα λεγόμενα στελέχη (stubs), τα οποία μπορούν να χαρακτηριστούν ως οι αντιπρόσωποι των απομακρυσμένων διαδικασιών στο σύστημα, στο οποίο πραγματοποιείται η κλήση. Χρησιμοποιώντας το στέλεχος, η κλήση της συνάρτησης πραγματοποιείται με τον ίδιο τρόπο, που θα πραγματοποιούταν, αν αυτή εκτελούταν τοπικά. Το στέλεχος συλλέγει τις παραμέτρους και τις τοποθετεί σε ένα μήνυμα. Η λειτουργία αυτή είναι γνωστή ως συγκέντρωση παραμέτρων (parameter marshalling). Κατόπιν, το μήνυμα μεταφέρεται σε ένα στέλεχος (stub), το οποίο βρίσκεται στο απομακρυσμένο περιβάλλον εκτέλεσης. Το στέλεχος αυτό διαχωρίζει τις παραμέτρους κλήσης και καλεί την απομακρυσμένη διεργασία με το γνωστό τρόπο κλήσης μιας συνάρτησης. Όταν ολοκληρωθεί η εκτέλεση της συνάρτησης, με αντίστοιχη λειτουργία των στελεχών, η τιμή που επιστρέφεται από την απομακρυσμένη διαδικασία, σειριοποιείται και μεταβιβάζεται στην καλούσα διεργασία.

2.3.3. Μεσισμικό (Middleware)

Η τεχνολογία μεσισμικού (middleware) παρεμβάλλει ένα λειτουργικό επίπεδο μεταξύ αυτών του πελάτη και του εξυπηρετητή, το οποίο παρέχει υπηρεσίες, όπως απόκρυψη ονομασιών και τοποθεσιών (location and alias resolution), πιστοποίηση (authentication) και σημασιολογία συναλλαγής (transaction semantics) [1]. Άλλες λειτουργίες, σχετιζόμενες με το μεσισμικό, περιλαμβάνουν το συγχρονισμό και τη μετάφραση μεταξύ διαφορετικών τρόπων αποθήκευσης και αναπαράστασης της πληροφορίας (translation between data formats). Με τη χρήση του μεσισμικού, οι πελάτες αλληλεπιδρούν με τον εξυπηρετητή σε ένα περισσότερο αφηρημένο επίπεδο, αντί να αλληλεπιδρούν απευθείας με ένα συγκεκριμένο εξυπηρετητή και/ή διεργασία. Οι διάφορες υπηρεσίες παρέχονται επίσης μέσα από αφηρημένα επίπεδα, μειώνοντας με τον τρόπο αυτό τη διάκριση μεταξύ των υπηρεσιών που παρέχονται από το μεσισμικό και τη λειτουργικότητα, που προστίθεται από τους εξυπηρετητές. Η παροχή των υπηρεσιών με αυτό τον αφηρημένο τρόπο επιτρέπει την ανάπτυξη εφαρμογών, βασιζόμενων σε τυποποιημένες προγραμματιστικές διεπαφές, χωρίς γνώση της τοποθεσίας (location transparency) ή της υλοποίησης της εξωτερικής λειτουργικότητας.

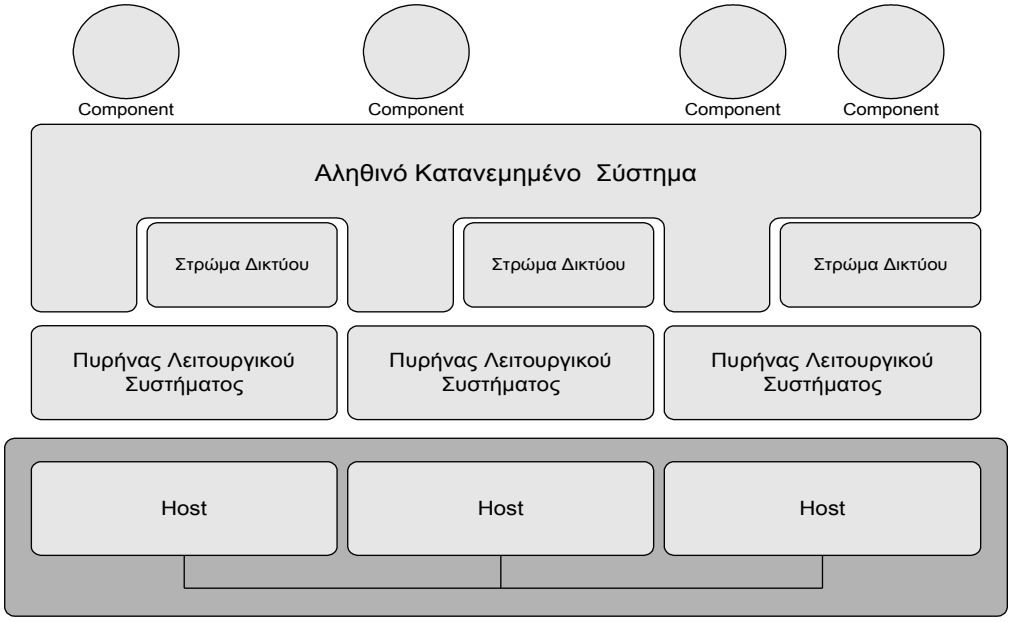
2.3.4 Κατανεμημένο Περιβάλλον

Τα κλασσικά κατανεμημένα συστήματα έχουν τη μορφή που φαίνεται στο Σχήμα 1. Τα συστήματα αυτής τη μορφής μπορούν να χωριστούν στα εξής επίπεδα:

- Το χαμηλότερο στρώμα, ακριβώς πάνω από το υλικό, αποτελείται από τον πυρήνα του λειτουργικού συστήματος. Ο πυρήνας του λειτουργικού συστήματος μπορεί να θεωρηθεί ως το στρώμα που παρέχει τις βασικές λειτουργίες του λειτουργικού συστήματος, όπως το σύστημα αρχείων, η διαχείριση μνήμης, και η εκτέλεση διεργασίας. Αυτό το στρώμα δεν παρέχει καμία υποστήριξη για την επικοινωνία ή τη διανομή.
- Το στρώμα δικτύου λειτουργικού συστήματος παρέχει τις αδιαφανείς υπηρεσίες επικοινωνίας. Η εφαρμογή που χρησιμοποιεί τις υπηρεσίες του στρώματος αυτού

απευθύνεται στον οικοδεσπότη (host) με τον οποίο θέλει να επικοινωνήσει χρησιμοποιώντας τη διεύθυνση/όνομα του. Παραδείγματος χάριν, οι υπηρεσίες υποδοχών (sockets) μπορούν να θεωρηθούν ότι ανήκουν στο στρώμα δικτύου, δεδομένου ότι για να δημιουργηθεί μια υποδοχή (socket) πρέπει να διευκρινίσει τον κόμβο του δικτύου με τον οποίο θέλει να επικοινωνήσει. Το στρώμα δικτύου χρησιμοποιεί τις υπηρεσίες που παρέχονται από τον πυρήνα, π.χ. διαχείριση μνήμης.

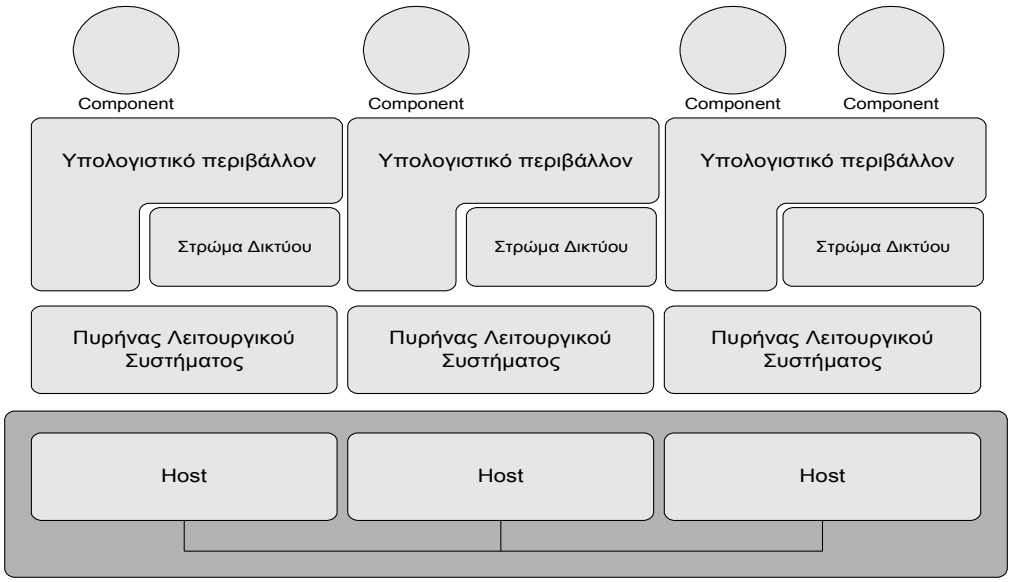
● Στο υψηλότερο επίπεδο υπάρχει το αληθινό κατανεμημένο στρώμα συστημάτων. Ένα αληθινό κατανεμημένο σύστημα υλοποιεί μια πλατφόρμα όπου τα συστατικά, που βρίσκονται επί των διαφορετικών τόπων ενός δικτύου, γίνονται αντιληπτά όπως τοπικά. Οι χρήστες των υπηρεσιών του αληθινού κατανεμημένου συστήματος δεν χρειάζεται να γνωρίζουν την ελλοχεύουσα δομή του δικτύου. Όταν καλείται μια υπηρεσία, δεν υπάρχει καμία ένδειξη για τον κόμβο του δικτύου που θα παράσχει πραγματικά την υπηρεσία, ούτε και για την παρουσία του δικτύου.



Σχήμα 1. Κλασσικό Κατανεμημένο Σύστημα.

2.3.5 Κινητός Κώδικας

Οι τεχνολογίες που υποστηρίζουν κινητό κώδικα έχουν μια διαφορετική προοπτική. Η δομή του ελλοχεύοντος δικτύου υπολογιστών δεν είναι κρυμμένη από τον προγραμματιστή. Όπως φαίνεται στο Σχήμα 2, το αληθινό κατανεμημένο σύστημα αντικαθίσταται από τα υπολογιστικά περιβάλλοντα, ένα νέο στρώμα πάνω από το στρώμα δικτύου σε κάθε οικοδεσπότη δικτύων (network host). Σε αντίθεση με το αληθινό κατανεμημένα σύστημα, το υπολογιστικό περιβάλλον (ΥΠ) διατηρεί τη "ταυτότητα" του οικοδεσπότη (host) όπου βρίσκεται. Ο σκοπός του ΥΠ είναι να παράσχει στις εφαρμογές την ικανότητα να επανεντοπίζουν δυναμικά τα συστατικά τους στους διαφορετικούς οικοδεσπότες (hosts). Έτσι τα ΥΠ χειρίζονται τον επανεντοπισμό του κώδικα, και ενδεχομένως της κατάστασης, των φιλοξενημένων (hosted) τμημάτων λογισμικού, απομακρύνοντας επιπρόσθετο φόρτο εργασίας τόσο από το στρώμα δικτύου όσο και από τον πυρήνα.



Σχήμα 2. Σύστημα Κινητού Κώδικα.

3. Τεχνολογίες Κατανεμημένων Αντικειμένων – Μεσισμικό

Τα τελευταία χρόνια υπήρξε ραγδαία αύξηση της επίδοσης των υπολογιστών, ενώ παράλληλα το κόστος της μετάδοσης δεδομένων μειώθηκε αισθητά. Με τη ταυτόχρονη άνοδος της ταχύτητας και της ποιότητας των δικτύων υπολογιστών, τόσο τοπικών όσο και μεγαλύτερης κλίμακας, οι συνθήκες αυτές οδήγησαν στη σκέψη για δημιουργία εφαρμογών οι οποίες θα ανταλλάσσουν δεδομένα πάνω από το δίκτυο. Λόγω της πολυπλοκότητας αυτών των εφαρμογών, είναι απαραίτητο να υπάρξει απλότητα και ευελιξία στην αρχιτεκτονική τους. Η ανάγκη αυτή οδήγησε στην ανάπτυξη κατανεμημένων εφαρμογών.

Οι αρχιτεκτονικές κατανεμημένων αντικειμένων χρησιμοποιούν ως βάση την ιδέα του μεσισμικού και ενθυλακώνουν δεδομένα μέσα στις λειτουργικές διεπαφές των αντικειμένων με τρόπο παρόμοιο με τις καλά σχεδιασμένες διαδικαστικές διεπαφές προγραμματισμού εφαρμογών όπου οι λεπτομέρειες της υλοποίησης αποκρύπτονται από τους χρήστες των αντικειμένων. Σε αντίθεση όμως με αυτές, οι αρχιτεκτονικές αντικειμένων περιορίζουν την πρόσβαση στην κλήση ορισμένων μεθόδων που ορίζονται για το αντικείμενο. Επιπρόσθετα, οι μέθοδοι καλούνται για τα αντικείμενα έμμεσα, μέσα από αναφορές σε αυτά, εξαλείφοντας έτσι την ανάγκη για δημιουργία τοπικών στιγμιότυπων των αντικειμένων.

3.1. Μεσισμικό

Με τον γενικό όρο μεσισμικό (middleware) αναφερόμαστε σε ένα κομμάτι λογισμικού που λειτουργεί σαν «κόλλα» μεταξύ δύο διαφορετικών και κατά τα άλλα ξεχωριστών εφαρμογών [1]. Λόγω της λειτουργίας του ως ενδιάμεσο στρώμα,το μεσισμικό έχει την ικανότητα να προσφέρει ένα σύνολο από δυνατότητες στις εφαρμογές τις οποίες συνδέει. Συχνά αναφέρεται και ως «σωλήνας» που μεταφέρει δεδομένα ανάμεσα σε δύο εφαρμογές ή ανάμεσα σε δύο τμήματα της ίδιας εφαρμογής. Υπάρχουν διάφορα είδη μεσισμικού ανάλογα με την προοριζόμενη εφαρμογή. Ενδεικτικά αναφέρεται ότι μπορεί να χρησιμοποιηθεί για την πρόσβαση σε βάσεις δεδομένων (τεχνολογία ODBC). Στην περίπτωση του κατανεμημένου μοντέλου, το μεσισμικό χρησιμοποιείται για την παροχή διαφάνειας μεταξύ των εφαρμογών. Για παράδειγμα, το ORB (Object Request Broker) χρησιμεύει ως ένα ενδιάμεσο στρώμα μεταξύ πελάτη και εξυπηρετητή και επιτρέπει σε έναν πελάτη να ζητήσει μια υπηρεσία από κάποιον εξυπηρετητή χωρίς να χρειάζεται αυτός να γνωρίζει τις λεπτομέρειες με τις οποίες ο οποιοσδήποτε εξυπηρετητής συνδέεται στο δίκτυο.

3.2. Java RMI

Το RMI (Remote Method Invocation) αποτελεί την πρόταση της Sun για τη σχεδίαση κατανεμημένων συστημάτων και έχει σχεδιαστεί ειδικά για τη λειτουργία σε περιβάλλον εφαρμογών Java. Το σύστημα του RMI προϋποθέτει το ομογενές περιβάλλον της Εικονικής Μηχανής της Java (Java Virtual Machine – JVM) και επομένως μπορεί να εκμεταλλευτεί το μοντέλο του αντικειμένου της πλατφόρμας της Java όποτε αυτό είναι δυνατό. Η αρχιτεκτονική του RMI εισάγει στη γλώσσα

προγραμματισμού Java κάποιες επιπλέον δυνατότητες όσον αφορά την ανάπτυξη κατανεμημένων εφαρμογών, οι σημαντικότερες από τις οποίες είναι οι εξής [2]:

- επιτρέπει τη διαφανή πρόσβαση σε απομακρυσμένα αντικείμενα που βρίσκονται σε διαφορετικές εικονικές μηχανές
- υποστηρίζει επανακλήσεις από servers σε applets
- ενσωματώνει το μοντέλο του κατανεμημένου αντικειμένου στην Java με φυσικό τρόπο, χωρίς να αλλοιώνει τα χαρακτηριστικά του κλασσικού μοντέλου αντικειμένου που υπάρχει στην Java.
- καθιστά εμφανείς τις διαφορές ανάμεσα στο μοντέλο του κατανεμημένου αντικειμένου και το μοντέλο του τοπικού αντικειμένου που υπάρχει στην πλατφόρμα της Java
- απλοποιεί την ανάπτυξη κατανεμημένων εφαρμογών με Java
- διατηρεί την ασφάλεια των τύπων που παρέχεται από το περιβάλλον εκτέλεσης της Java
- υποστηρίζει διάφορους τύπους αναφοράς για απομακρυσμένα αντικείμενα, όπως για παράδειγμα μη-παραμένουσες ή παραμένουσες αναφορές
- διατηρεί το ασφαλές περιβάλλον της πλατφόρμας της Java που παρέχουν οι διαχειριστές ασφάλειας και οι φορτωτές κλάσεων (class loaders).

3.2.1 Επισκόπηση Εφαρμογών RMI

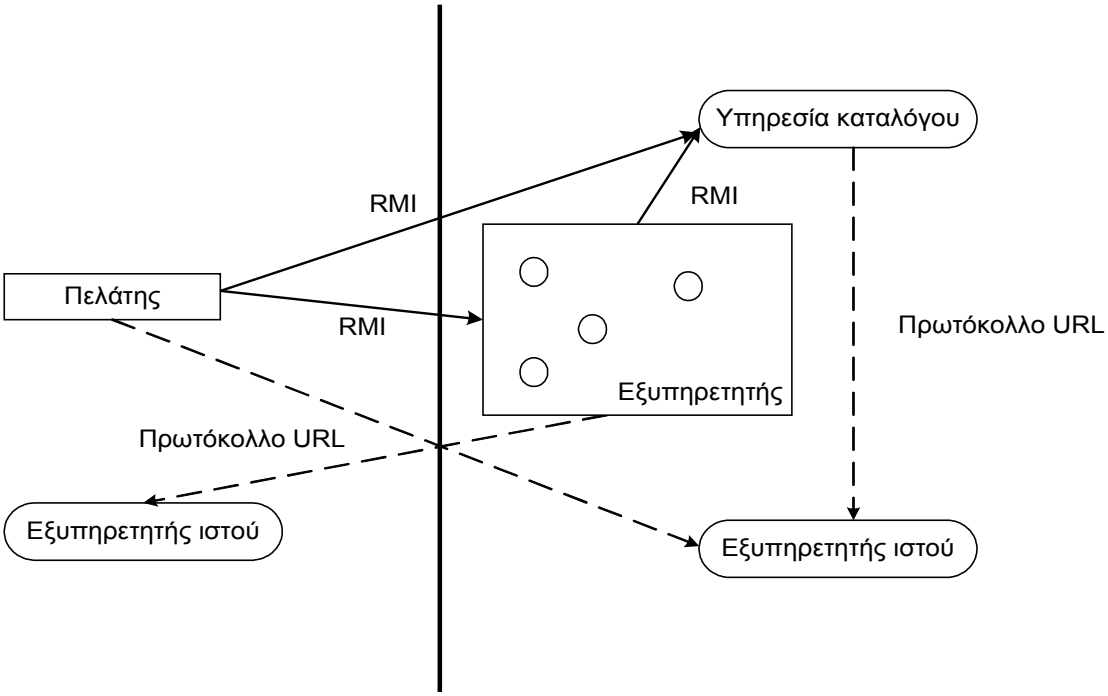
Οι εφαρμογές RMI αποτελούνται συνήθως από δύο διαφορετικά προγράμματα: έναν εξυπηρετητή και έναν πελάτη. Μια τυπική εφαρμογή εξυπηρετητή δημιουργεί κάποια απομακρυσμένα αντικείμενα, κάνει τις αναφορές σε αυτά προσβάσιμες και περιμένει από πελάτες να καλέσουν μεθόδους σε ένα ή περισσότερα από αυτά τα αντικείμενα. Από την άλλη, μια τυπική εφαρμογή πελάτη παίρνει κάποια απομακρυσμένη αναφορά σε ένα ή περισσότερα από τα απομακρυσμένα αντικείμενα στον εξυπηρετητή και καλεί μεθόδους σε αυτά. Το RMI προσφέρει το μηχανισμό με τον οποίο πελάτης και εξυπηρετητής επικοινωνούν και ανταλλάσσουν πληροφορία.

Οι εφαρμογές που χρησιμοποιούν το RMI έχουν τη δυνατότητα να [4]:

- εντοπίσουν απομακρυσμένα αντικείμενα: Οι εφαρμογές μπορούν να χρησιμοποιήσουν κάποιους μηχανισμούς για να αποκτήσουν αναφορές σε απομακρυσμένα αντικείμενα. Μια εφαρμογή μπορεί να καταγράψει (register) τα απομακρυσμένα της αντικείμενα σε μια απλή υπηρεσία ονομασίας που προσφέρει το RMI, το rmiregistry (υπηρεσία καταλόγου), ή ακόμη και να περνά ως παραμέτρους ή να επιστρέφει αναφορές σε αντικείμενα σαν μέρος της κανονικής της λειτουργίας.
- επικοινωνούν με απομακρυσμένα αντικείμενα: Οι λεπτομέρειες της επικοινωνίας μεταξύ απομακρυσμένων αντικειμένων χειρίζονται αποκλειστικά από το RMI. Από την πλευρά του προγραμματιστή, η απομακρυσμένη επικοινωνία μοιάζει με τη συνηθισμένη κλήση μεθόδων στην Java.
- φορτώνουν τον κώδικα αντικειμένων που περνιούνται ως παράμετροι: Επειδή το RMI επιτρέπει το πέρασμα απομακρυσμένων αντικειμένων κατά την κλήση μεθόδων, το RMI παρέχει τους απαραίτητους μηχανισμούς για το φόρτωμα του κώδικα ενός αντικειμένου, καθώς και για τη μετάδοση των δεδομένων του.

Στο Σχήμα 3 φαίνεται το παράδειγμα μιας υλοποιημένης με RMI κατανεμημένης εφαρμογής που χρησιμοποιεί την υπηρεσία καταλόγου για να αποκτήσει μια αναφορά σε κάποιο απομακρυσμένο αντικείμενο. Ο εξυπηρετητής χρησιμοποιεί την υπηρεσία καταλόγου για να συσχετίσει ένα όνομα με ένα απομακρυσμένο αντικείμενο. Ο πελάτης ψάχνει για το απομακρυσμένο αντικείμενο με το όνομα που αυτό έχει στην

υπηρεσία καταλόγου του εξυπηρετητή και στη συνέχεια καλεί κάποια μέθοδο σε αυτό.



Σχήμα 3. Παράδειγμα εφαρμογής RMI.

3.2.2 Αρχιτεκτονική του Συστήματος RMI

Η υλοποίηση του συστήματος του RMI αποτελείται ουσιαστικά από τρία επίπεδα [3]. Το πρώτο από αυτά είναι το στρώμα των στελεχών – σκελετών, με το οποίο έρχεται σε απευθείας επαφή ο προγραμματιστής. Το στρώμα αυτό παρεμβάλλεται στις κλήσεις που κάνει ο πελάτης σε μεθόδους κάποιου απομακρυσμένου αντικειμένου και επανακατευθύνει αυτές τις κλήσεις στην απομακρυσμένη υπηρεσία RMI. Το επόμενο στρώμα είναι το στρώμα απομακρυσμένης αναφοράς. Το στρώμα αυτό είναι υπεύθυνο για τη διερμηνεία και τη διαχείριση αναφορών που γίνονται από τους πελάτες σε αντικείμενα απομακρυσμένων υπηρεσιών. Αρχικά το στρώμα απομακρυσμένης αναφοράς ένωνε τους πελάτες με αντικείμενα απομακρυσμένων υπηρεσιών που γίνονταν διαθέσιμα μέσω κάποιου εξυπηρετητή. Η σύνδεση ήταν ένας σύνδεσμος ένα – προς – ένα (unicast). Στις τελευταίες εκδόσεις του Java SDK το στρώμα αυτό υποστηρίζει την ενεργοποίηση προσωρινά αδρανών απομακρυσμένων υπηρεσιών μέσω ενός χαρακτηριστικού που ονομάζεται απομακρυσμένη ενεργοποίηση αντικειμένου (remote object activation). Το τρίτο και τελευταίο ονομάζεται στρώμα μεταφοράς και βασίζεται σε συνδέσεις TCP/IP ανάμεσα σε υπολογιστές ενός δικτύου. Σχηματικά η αρχιτεκτονική του συστήματος RMI φαίνεται στο Σχήμα 4.

Ο λόγος για τον οποίο χρησιμοποιείται η παραπάνω διαστρωμάτωση είναι για την εύκολη αναβάθμιση ή αντικατάσταση κάποιου στρώματος χωρίς να επηρεάζονται τα υπόλοιπα στρώματα. Στη συνέχεια παρουσιάζονται αναλυτικότερα τα στρώματα της αρχιτεκτονικής του RMI.

3.2.2.1 Στελέχη και Σκελετοί

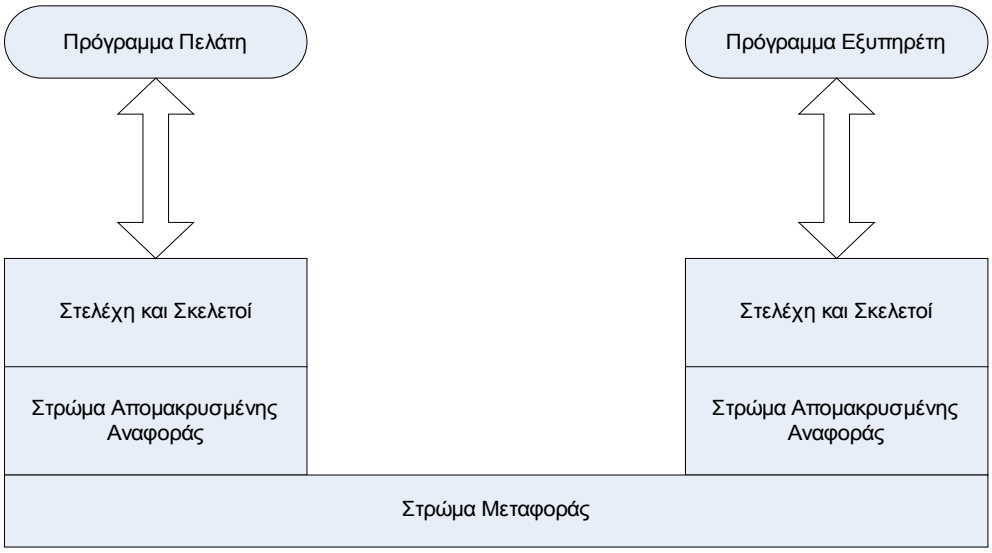
Για την επικοινωνία με απομακρυσμένα αντικείμενα το RMI χρησιμοποιεί τον μηχανισμό των στελεχών (stubs) και των σκελετών, που χρησιμοποιείται παραδοσιακά στα συστήματα RPC. Το στέλεχος λειτουργεί για ένα απομακρυσμένο αντικείμενο ως ο τοπικός αντιπρόσωπός του για τον πελάτη. Ο πελάτης καλεί μια μέθοδο στο τοπικό στέλεχος, το οποίο είναι υπεύθυνο να πραγματοποιήσει την κλήση της μεθόδου στο απομακρυσμένο αντικείμενο. Κάθε στέλεχος ενός απομακρυσμένου αντικειμένου υλοποιεί τις ίδιες διεπαφές που υλοποιεί και ένα απομακρυσμένο αντικείμενο. Όταν καλείται μια μέθοδος ενός στελέχους, συμβαίνουν τα ακόλουθα:

- αρχικοποιείται μια σύνδεση με την απομακρυσμένη εικονική μηχανή στην οποία υπάρχει το ζητούμενο απομακρυσμένο αντικείμενο
- γράφονται και μεταφέρονται οι παράμετροι (parameter marshalling) στην απομακρυσμένη εικονική μηχανή
- αναμένεται το αποτέλεσμα της κλήσης της μεθόδου
- διαβάζεται η επιστρεφόμενη τιμή ή η πιθανή εξαίρεση που δημιουργήθηκε
- επιστρέφεται το αποτέλεσμα στο αντικείμενο που έκανε αρχικά την κλήση της μεθόδου στο απομακρυσμένο αντικείμενο

Ο ρόλος του στελέχους είναι να κρύβει τη διαδικασία της σειριοποίησης των αντικειμένων, καθώς και τις λεπτομέρειες της επικοινωνίας σε επίπεδο δικτύου, ώστε να παρουσιάζεται στον προγραμματιστή ένας απλός μηχανισμός κλήσης μεθόδων σε απομακρυσμένα αντικείμενα.

Στην απομακρυσμένη εικονική μηχανή, σε κάθε απομακρυσμένο αντικείμενο μπορεί να αντιστοιχεί ένας σκελετός (skeleton). Ο σκελετός είναι υπεύθυνος για να μεταφέρει την κλήση στο δικό του απομακρυσμένο αντικείμενο. Όταν ένας σκελετός λάβει μια εισερχόμενη κλήση για απομακρυσμένη μέθοδο, εκτελεί τις ακόλουθες ενέργειες:

- διαβάζει (unmarshals) τις παραμέτρους για την απομακρυσμένη μέθοδο
- εκτελεί τη μέθοδο στην πραγματική υλοποίηση του απομακρυσμένου αντικειμένου
- γράφει και μεταδίδει (marshals) το αποτέλεσμα της μεθόδου (επιστρεφόμενη τιμή ή εξαίρεση) στο αντικείμενο που κάλεσε αρχικά την απομακρυσμένη μέθοδο



Σχήμα 4. Αρχιτεκτονική του Συστήματος RMI.

3.2.2.2 Στρώμα Απομακρυσμένης Αναφοράς

Το στρώμα απομακρυσμένης αναφοράς ορίζει και υποστηρίζει τον τρόπο με τον οποίο γίνεται η κλήση απομακρυσμένης μεθόδου σε μια σύνδεση RMI. Στην υλοποίηση του Java 2 SDK υποστηρίζεται η ενεργοποίηση απομακρυσμένων αντικειμένων. Όταν γίνεται μια κλήση μεθόδου στον αντιπρόσωπο ενός απομακρυσμένου αντικειμένου, το RMI καθορίζει αν η υλοποίηση του αντικειμένου στην απομακρυσμένη υπηρεσία είναι αδρανής. Αν όντως είναι αδρανής, το RMI θα στιγμιοποιήσει το αντικείμενο και θα επαναφέρει την κατάστασή του από κάποιο αρχείο σε κάποιον δίσκο. Στη συνέχεια, δημιουργείται μία από σημείο σε σημείο σύνδεση και πραγματοποιείται η διαδικασία απομακρυσμένης κλήσης της μεθόδου. Αν το απομακρυσμένο αντικείμενο δεν είναι αδρανές και έχει ήδη ενεργοποιηθεί, τότε δημιουργείται απευθείας η σύνδεση και εκτελείται η διαδικασία κλήσης που προαναφέρθηκε. Υπάρχουν επίσης και άλλοι τρόποι σύνδεσης, όπως για παράδειγμα η πολυεκπομπή, όπου ένας μοναδικός αντιπρόσωπος μπορεί να στείλει μια αίτηση μεθόδου σε πολλαπλές υλοποιήσεις ταυτόχρονα και να δεχτεί μόνο την πρώτη απάντηση, πράγμα που βελτιώνει το χρόνο απόκρισης και πιθανώς βελτιώνει τη διαθεσιμότητα.

3.2.2.3 Στρώμα Μεταφοράς

Το στρώμα μεταφοράς δημιουργεί τη σύνδεση μεταξύ εικονικών μηχανών. Όλες αυτές οι συνδέσεις είναι συνδέσεις δικτύου που χρησιμοποιούν το πρωτόκολλο TCP/IP και βασίζονται σε ρεύματα δεδομένων. Ακόμη και αν δύο εικονικές μηχανές τρέχουν στον ίδιο υπολογιστή, η σύνδεσή τους γίνεται και πάλι μέσω της στοίβας του πρωτοκόλλου TCP/IP. Πάνω από το TCP/IP, το RMI αρχικά χρησιμοποιούσε το πρωτόκολλο JRMP (Java Remote Method Protocol). Μετά την έκδοση 2 του Java SDK, το RMI αναβαθμίστηκε ώστε να χρησιμοποιεί το πρωτόκολλο IIOP (Internet Inter Orb Protocol) που δημιουργήθηκε από το OMG (Object Management Group).

3.3. MDA/OMA/CORBA

Ένας από τους πιο σημαντικούς στόχους της κατανεμημένης αντικειμενοστραφούς επεξεργασίας είναι ο συνδυασμός των τεχνολογιών αντικειμένων και των κατανεμημένων συστημάτων και η ολοκλήρωσή τους σε μία αρχιτεκτονική, με έμφαση στη διαλειτουργικότητα, όχι μόνο μεταξύ των ήδη υπαρχόντων υπολογιστικών πλατφορμών, αλλά και αυτών που πρόκειται να αναπτυχθούν στο μέλλον. Στα πλαίσια αυτά, ο OMG (Object Management Group) έχει ορίσει ένα σύνολο από προδιαγραφές για την επίτευξη αυτής της λειτουργικότητας ανεξαρτήτως πλατφόρμας υλοποίησης.

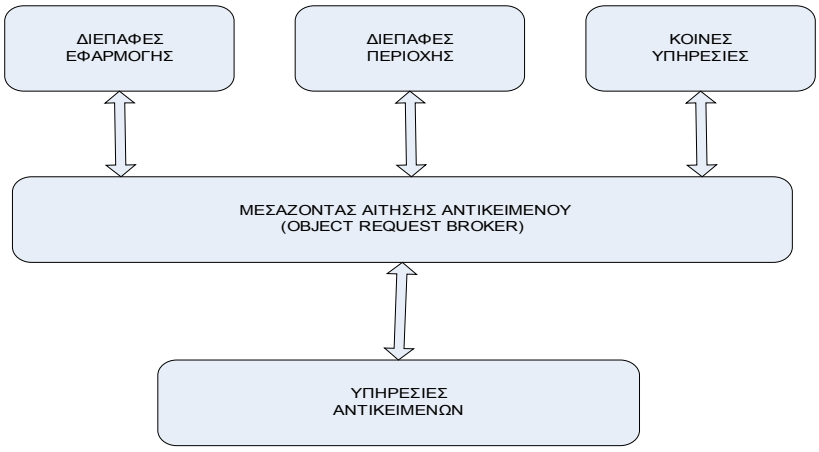
3.3.1 MDA

Η αρχιτεκτονική MDA (Model Driven Architecture) είναι ένας σχετικά καινούριος τρόπος για τη σχεδίαση προδιαγραφών και την ανάπτυξη εφαρμογών, ο οποίος είναι βασισμένος σε ένα μοντέλο ανεξάρτητο πλατφόρμας (platform-independent model – PIM). Μια ολοκληρωμένη MDA προδιαγραφή αποτελείται από ένα ανεξάρτητο

πλατφόρμας μοντέλο, μαζί με ένα ή περισσότερα σετ από μοντέλα ειδικά για κάθε πλατφόρμα (platform-specific models – PSMs), καθώς και από σετ με ορισμούς διεπαφών, τα οποία περιγράφουν τον τρόπο με τον οποίο το βασικό μοντέλο υλοποιείται σε διαφορετικές πλατφόρμες μεσισμικού. Μια ολοκληρωμένη MDA εφαρμογή αποτελείται από ένα PIM, ένα ή περισσότερα PSMs και από τις ολοκληρωμένες υλοποιήσεις για κάθε πλατφόρμα που θα υποστηρίζει η εφαρμογή. Η αρχιτεκτονική MDA επικεντρώνεται κυρίως στη λειτουργικότητα και τη συμπεριφορά μιας κατανεμημένης εφαρμογής ή ενός κατανεμημένου συστήματος, χωρίς να επηρεάζεται ουσιαστικά από την τεχνολογία ή τις τεχνολογίες με τις οποίες η εφαρμογή ή το σύστημα θα υλοποιηθεί. Με άλλα λόγια, η αρχιτεκτονική MDA διαχωρίζει τις λεπτομέρειες υλοποίησης από τις λειτουργίες που πρέπει να υποστηρίζονται. Αυτό σημαίνει ότι δεν είναι απαραίτητη η επανάληψη της διαδικασίας μοντελοποίησης της λειτουργικότητας μιας εφαρμογής ή ενός συστήματος κάθε φορά που χρειάζεται να χρησιμοποιηθεί μια νέα τεχνολογία. Σε αντίθεση με την κλασσική προσέγγιση, όπου οι αρχιτεκτονικές είναι άρρηκτα συνδεδεμένες με μια συγκεκριμένη τεχνολογία, στην περίπτωση της MDA η λειτουργικότητα και η συμπεριφορά μοντελοποιούνται μόνο μια φορά.

3.3.2 OMA

Η OMA (Object Management Architecture) [5,6] πρόκειται για μία αρχιτεκτονική που θέτει πρότυπα για τον ορισμό, τον τρόπο δημιουργίας και τους μηχανισμούς επικοινωνίας μεταξύ των αντικειμένων, στα πλαίσια ενός ανομοιογενούς κατανεμημένου περιβάλλοντος. Τα βασικά μοντέλα, που καθορίζονται στην αρχιτεκτονική αυτή είναι το Μοντέλο Αντικειμένου (Object Model), το οποίο καθορίζει την περιγραφή ενός αντικειμένου και το Μοντέλο Αναφοράς (Reference Model), το οποίο καθορίζει τις πιθανές αλληλεπιδράσεις μεταξύ των αντικειμένων. Σύμφωνα με το Μοντέλο Αντικειμένου, το αντικείμενο είναι μια οντότητα, η οποία χαρακτηρίζεται από κάποια συγκεκριμένη ταυτότητα και παρέχει υπηρεσίες σε εξωτερικές προς αυτή οντότητες (αντικείμενα). Κάθε αντικείμενο παρέχει τις υπηρεσίες του μέσα από διεπαφές. Το μοντέλο αναφοράς της αρχιτεκτονικής του OMA φαίνεται στο Σχήμα 5. Οι διεπαφές μεταξύ των αντικειμένων ορίζονται μέσω της γλώσσας καθορισμού διεπαφών (Interface Definition Language-IDL) του OMG, η οποία είναι σχεδιασμένη με προσανατολισμό την ανεξαρτησία από την πλατφόρμα και τη γλώσσα προγραμματισμού και η οποία παρουσιάζεται σε επόμενη παράγραφο.



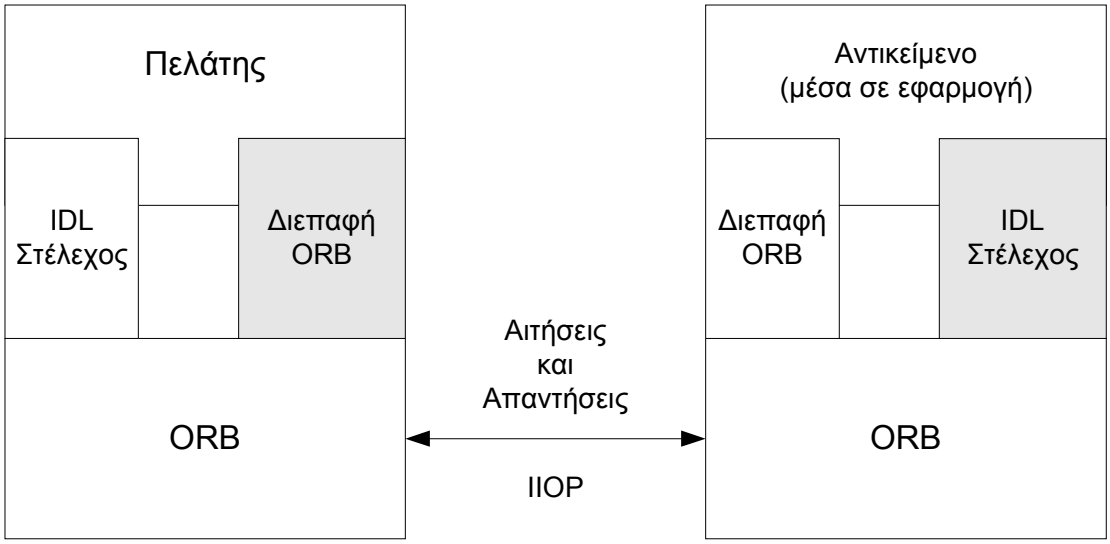
Σχήμα 5. Η αρχιτεκτονική OMA.

3.3.3 CORBA

Η CORBA (Common Object Request Broker Architecture) είναι μια αρχιτεκτονική η οποία βασίζεται στην ιδέα ενός κοινού διαύλου λογισμικού ή αντικειμένων, ο οποίος επιτρέπει τη συνεργασία κατανεμημένων αντικειμένων και προσφέρει μεγάλο εύρος υπηρεσιών στα αντικείμενα αυτά. Ο δίαυλος αυτός καλείται ORB (Object Request Broker). Ο κάθε κατασκευαστής κατασκευάζει επιμέρους κομμάτια του ORB, τα οποία εγκαθιστώνται τοπικά στον κάθε κόμβο και επικοινωνούν μεταξύ τους με χρήση των μηχανισμών χαμηλότερου επιπέδου (π.χ. με χρήση sockets), δημιουργώντας έτσι την ψευδαίσθηση του κοινού διαδρόμου. Κάθε εφαρμογή που στηρίζεται στην υποδομή που προσφέρει η CORBA χρησιμοποιεί το πρωτόκολλο IIOP για την επικοινωνία με άλλες εφαρμογές.

3.3.3.1 Αρχιτεκτονική της CORBA

Η βασική δομή της αρχιτεκτονικής CORBA μοιάζει αρκετά με την αρχιτεκτονική του συστήματος RMI. Και εδώ συναντώνται οι έννοιες του σκελετού και του στελέχους ως ενδιάμεσες οντότητες ανάμεσα στην εφαρμογή και το απομακρυσμένο αντικείμενο, με κάποιες όμως διαφορές. Η βασική αρχιτεκτονική της τεχνολογίας CORBA απεικονίζεται συνοπτικά στο Σχήμα 6.



Σχήμα 6. Αρχιτεκτονική τεχνολογίας CORBA.

Στις επόμενες παραγράφους θα παρουσιαστούν αναλυτικότερα κάποια από τα βασικότερα χαρακτηριστικά της αρχιτεκτονικής.

3.3.3.2 Γλώσσα IDL

Το χαρακτηριστικό της αντικειμενοστραφούς φιλοσοφίας που επιτρέπει τη διαφανή κατανομή αντικειμένων είναι η αρχή της ενθυλάκωσης. Ενσωματώνοντας την εσωτερική κατάσταση και το μηχανισμό των αντικειμένων μέσα σε κάποια όρια τα οποία ο πελάτης δεν επιτρέπεται να διαπεράσει, επιτυγχάνεται όχι μόνο ευλυγισία, αλλά και απλότητα στην αρχιτεκτονική, που δεν μπορεί να επιτευχθεί εύκολα με άλλο τρόπο. Για να υπάρχει η δυνατότητα να επικοινωνούν αυτά τα αντικείμενα με το εξωτερικό περιβάλλον, πρέπει να οριστούν κάποιες διεπαφές στο εξωτερικό αυτών

των ορίων. Αυτές οι διεπαφές χρειάζεται να είναι εύκολα κατανοητές και χρησιμοποιήσιμες από πελάτες και εξυπηρετητές, ανεξάρτητα από πλατφόρμα, λειτουργικό σύστημα, γλώσσα προγραμματισμού, σύνδεση δικτύου ή άλλα ιδιαίτερα χαρακτηριστικά.

Η Γλώσσα Ορισμού Διεπαφών (Interface Definition Language – IDL) του OMG είναι η γλώσσα που χρησιμοποιείται για την περιγραφή των διεπαφών που καλούν τα αντικείμενα των πελατών και παρέχουν οι υλοποιήσεις των αντικειμένων [7]. Ένας ορισμός διεπαφής που γράφεται με αυτήν τη γλώσσα περιγράφει ολοκληρωτικά τη διεπαφή και καθορίζει με πληρότητα κάθε παράμετρο όλων των λειτουργιών. Μια διεπαφή OMG IDL παρέχει την πληροφορία που χρειάζεται για την ανάπτυξη πελατών που χρησιμοποιούν τις λειτουργίες της διεπαφής. Οι πελάτες δεν γράφονται στη γλώσσα OMG IDL, αφού αυτή είναι μια καθαρά περιγραφική γλώσσα, αλλά σε άλλες γλώσσες για τις οποίες έχουν οριστεί αντιστοιχήσεις για τις έννοιες της γλώσσας OMG IDL. Υλοποιήσεις τέτοιων αντιστοιχήσεων υπάρχουν για όλες τις ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού, όπως C, C++, Java, Smalltalk, COBOL, Ada, Lisp, PL/1, Python και IDLscript, καθώς και για άλλες γλώσσες. Η αντιστοίχηση μιας έννοιας της OMG IDL με μια δομή της γλώσσας κάποιου πελάτη εξαρτάται από τα χαρακτηριστικά της γλώσσας του πελάτη και γίνεται από ειδικούς μεταγλωττιστές (IDL compilers). Για παράδειγμα, μια εξαίρεση στη γλώσσα OMG IDL μπορεί να αντιστοιχηθεί σε μια δομή σε μια γλώσσα που δεν υπάρχει η έννοια της εξαίρεσης, ή σε μια εξαίρεση σε γλώσσες που υπάρχει η συγκεκριμένη έννοια.

3.3.3.3 Μεσάζοντας Αίτησης Αντικειμένου (ORB)

Ο ORB (Object Request Broker) είναι το τμήμα της αρχιτεκτονικής CORBA που παρέχει τους μηχανισμούς με τους οποίους τα αντικείμενα διαφανώς στέλνουν και λαμβάνουν αιτήσεις και αποκρίσεις αντίστοιχα. Με αυτούς τους μηχανισμούς, ο ORB παρέχει διαλειτουργικότητα μεταξύ εφαρμογών σε διαφορετικές μηχανές σε ετερογενή κατανεμημένα περιβάλλοντα. Η διαλειτουργικότητα των ORB επεκτείνει τον παραπάνω ορισμό σε περιπτώσεις που τα αντικείμενα του πελάτη και του εξυπηρετητή αντιστοιχούν σε διαφορετικούς ORBs. Ο ORB είναι υπεύθυνος για όλους τους μηχανισμούς που χρειάζονται από μια αίτηση για την εύρεση της υλοποίησης του ζητούμενου αντικειμένου, για την προετοιμασία της υλοποίησης του αντικειμένου να δεχτεί την αίτηση, καθώς και για τη μετάδοση των δεδομένων που απαρτίζουν την αίτηση. Η διεπαφή που βλέπει ο πελάτης είναι εντελώς ανεξάρτητη από την τοποθεσία του αντικειμένου, τη γλώσσα προγραμματισμού με την οποία υλοποιήθηκε ή οποιοδήποτε άλλο χαρακτηριστικό που δεν υπάρχει στη διεπαφή του αντικειμένου.

3.3.3.4 Πρωτόκολλο IIOP

Το πρωτόκολλο IIOP (Internet Inter – ORB Protocol) αποτελεί εξειδίκευση του γενικότερου GIOP (General Inter – ORB Protocol). Το IIOP δουλεύει απευθείας πάνω από συνδέσεις που χρησιμοποιούν το πρωτόκολλο TCP/IP. Οι προδιαγραφές του πρωτοκόλλου GIOP αποτελούνται από τα εξής επιμέρους στοιχεία [7]:

- Ορισμός κοινής παράστασης δεδομένων (Common Data Representation – CDR definition). Ο CDR αποτελεί ένα συντακτικό απεικόνισης τύπων δεδομένων της OMG IDL σε μια χαμηλού επιπέδου παράσταση για μεταφορά μεταξύ ORBs.

● Μορφές Μηνυμάτων GIOP (GIOP Message Formats): τα GIOP μηνύματα ανταλλάσσονται μεταξύ ORBs για την εξυπηρετητήση αιτήσεων αντικειμένων, τον εντοπισμό υλοποιήσεων αντικειμένων και τη διαχείριση καναλιών επικοινωνίας.
● Υποθέσεις Μεταφοράς GIOP (GIOP Transport Assumptions): Οι προδιαγραφές του GIOP περιγράφουν γενικές υποθέσεις σχετικά με το στρώμα μεταφοράς οποιουδήποτε δικτύου που μπορεί πιθανώς να χρησιμοποιηθεί για τη μεταφορά GIOP μηνυμάτων. Οι προδιαγραφές αυτές περιγράφουν επίσης τον τρόπο με τον οποίο μπορεί να γίνει η διαχείριση των συνδέσεων, καθώς και κάποιους περιορισμούς σχετικά με τη σειρά των μηνυμάτων GIOP.

Το πρωτόκολλο IIOP προσθέτει στις προδιαγραφές ένα ακόμη στοιχείο:
● Μεταφορά Μηνυμάτων IOP Διαδικτύου (Internet IOP Message Transport): Η προδιαγραφή του IIOP περιγράφει τον τρόπο με τον οποίο οι ORBs εγκαθιστούν συνδέσεις TCP/IP και τις χρησιμοποιούν για τη μεταφορά μηνυμάτων GIOP.

Όπως προαναφέρθηκε, το IIOP δεν είναι ένα ξεχωριστό πρωτόκολλο, αλλά μια εξειδίκευση του GIOP για λειτουργία πάνω σε ένα συγκεκριμένο δίκτυο (διαδίκτυο). Το GIOP μπορεί να θεωρηθεί ως ένα σημείο αναφοράς για την ανάπτυξη μελλοντικών εξειδικεύσεων, παρόμοιων με το IIOP, για άλλα πρωτόκολλα.

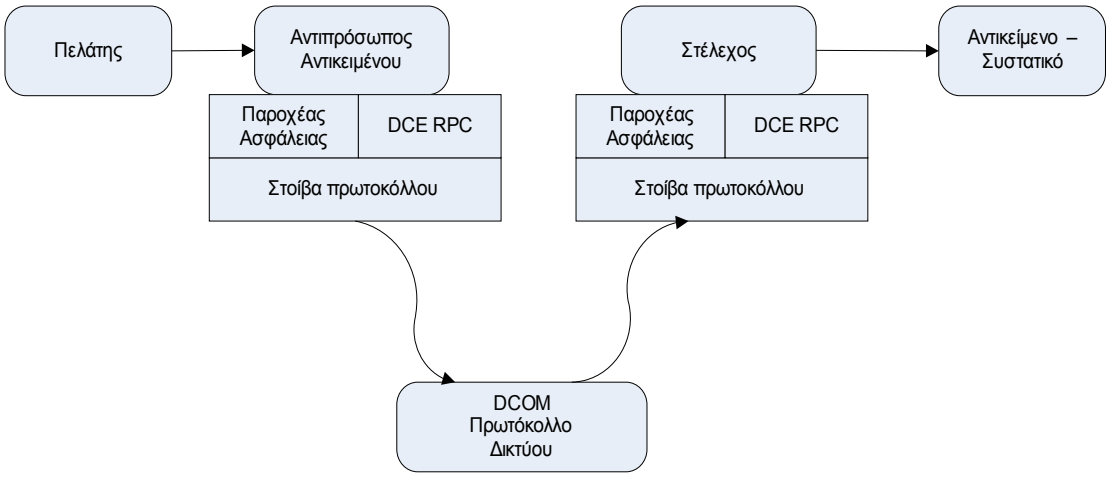
3.4. DCOM

Η DCOM (Distributed Component Object Model) είναι η απάντηση της Microsoft στην απαίτηση της βιομηχανίας για κατανεμημένα αντικείμενα λογισμικού. Αποτελεί επέκταση της τεχνολογίας COM (Component Object Model) και έχει ενσωματωθεί σε όλα τα λειτουργικά συστήματα που στηρίζονται στην πλατφόρμα Win32. Λόγω της ευρείας διάδοσης αυτών των συστημάτων πολλοί υποστηρίζουν ότι είναι πιθανό να έχει κάποιον σημαντικό μελλοντικό ρόλο στον τομέα της κατανεμημένης επεξεργασίας.

Η τεχνολογία COM επιτρέπει σε μια εφαρμογή να μπορεί να κατασκευαστεί από διάφορα δυαδικά συστατικά (αντικείμενα). Η τεχνολογία DCOM επεκτείνει την τεχνολογία COM στο ότι τα συστατικά αυτά μπορεί να βρίσκονται σε διαφορετικούς υπολογιστές. Επιπλέον, η DCOM βελτιώνει το μοντέλο ασφάλειας της τεχνολογίας COM και καθορίζει τη διαδικασία μεταφοράς δεδομένων μεταξύ υπολογιστών. Κάποια από τα σημαντικότερα χαρακτηριστικά της DCOM είναι τα εξής [8]:
● δυνατότητα χρησιμοποίησης ήδη υπαρχόντων αντικειμένων COM.
● ανεξαρτησία από πλατφόρμα: αντικείμενα που υλοποιούν την τεχνολογία DCOM και βρίσκονται σε διαφορετικά περιβάλλοντα, π.χ. Win32, Unix, Macintosh μπορούν να επικοινωνούν μεταξύ τους. Παρ' όλα αυτά η DCOM χρησιμοποιείται ευρέως μόνο σε πλατφόρμες Win32.
● ανεξαρτησία από γλώσσα προγραμματισμού: η Microsoft υποστηρίζει ότι η τεχνολογία COM (και κατά επέκταση και η DCOM) μπορεί να χρησιμοποιηθεί με οποιαδήποτε γλώσσα προγραμματισμού. Αυτό όμως είναι μόνο εν μέρει σωστό, αφού για μερικές γλώσσες (π.χ. Java) μόνο οι υλοποιήσεις από την Microsoft μπορούν να συνεργαστούν με την τεχνολογία COM.
● ανεξάρτητη από πρωτόκολλο μεταφοράς: Η DCOM μπορεί να δουλέψει πάνω από τα πρωτόκολλα TCP/IP, UDP, IPX/SPX, NetBIOS και AppleTalk.

3.4.1 Αρχιτεκτονική DCOM

Η αρχιτεκτονική DCOM απεικονίζεται στο Σχήμα 7.



Σχήμα 7. Αρχιτεκτονική DCOM.

Όπως φαίνεται και στο παραπάνω σχήμα, η αρχιτεκτονική αποτελείται από τα εξής βασικά στοιχεία:

- την υψηλού επιπέδου μεταφορά δεδομένων μεταξύ πελάτη και συστατικών, που στο σχήμα αντιστοιχεί στο ζεύγος αντιπρόσωπος – στέλεχος. Για το πέρασμα των παραμέτρων μεταξύ πελατών και εξυπηρετητών, χρησιμοποιείται η τεχνική της συγκέντρωσης/αποσυγκέντρωσης (marshalling/unmarshalling) που αναφέρθηκε σε προηγούμενη παράγραφο. Για να γίνει δυνατή η χρησιμοποίηση αυτής της τεχνικής χρησιμοποιείται και εδώ μια γλώσσα ορισμού διεπαφών (interface definition language – IDL).
- το συστατικό παροχής ασφάλειας. Ο μηχανισμός ασφάλειας στην τεχνολογία DCOM βασίζεται στο μοντέλο ασφάλειας που υπάρχει στην πλατφόρμα των Windows NT. Ακόμα και υλοποιήσεις της DCOM σε περιβάλλον Unix ή Macintosh χρησιμοποιούν το ίδιο μοντέλο ασφάλειας. Χαρακτηριστικό του συγκεκριμένου μοντέλου είναι ότι υποστηρίζει μηχανισμούς πολλαπλής αναγνώρισης και πιστοποίησης. Κεντρικό τμήμα της υποδομής ασφαλείας είναι ένας κατάλογος χρηστών, στον οποίο αποθηκεύονται πληροφορίες για την εξακρίβωση των στοιχείων ενός χρήστη.
- το συστατικό DCE RPC [9]. Αποτελεί ένα μηχανισμό απομακρυσμένης κλήσης διεργασίας (remote procedure call) που παρέχει πρωτόκολλα για τη μετατροπή των δομών δεδομένων και των παραμέτρων που υπάρχουν στη μνήμη σε πακέτα για μετάδοση μέσω ενός δικτύου.
- το πρωτόκολλο δικτύου. Αυτό το συστατικό έχει υλοποιηθεί έτσι ώστε να μπορεί η τεχνολογία DCOM να λειτουργήσει ανεξάρτητα από το πρωτόκολλο του υποκείμενου δικτύου. Όπως προαναφέρθηκε, τα πρωτόκολλα που υποστηρίζονται είναι τα TCP/IP, UDP, IPX/SPX, NetBIOS και AppleTalk.

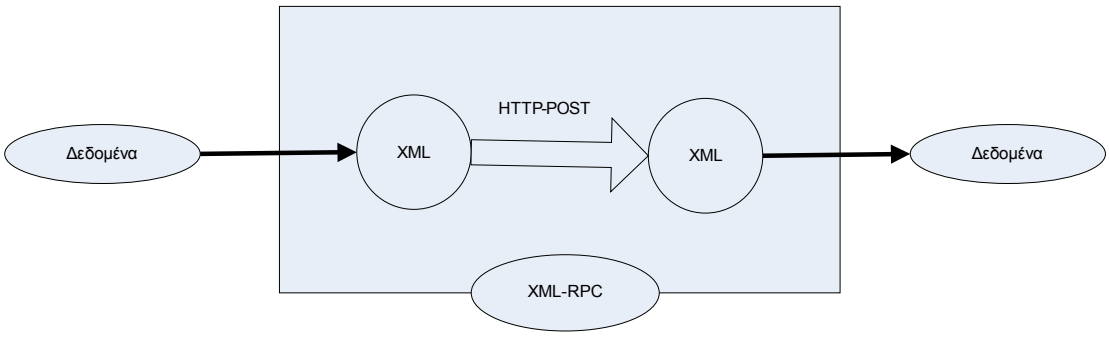
3.5. XML-RPC

Το XML-RPC αποτελεί ένα πρωτόκολλο απομακρυσμένης κλήσης διαδικασίας (Remote Procedure Calling) που λειτουργεί πάνω στο διαδίκτυο [10]. Αποτελεί περισσότερο τεχνολογία μεσισμικού παρά τεχνολογία κατανεμημένων αντικειμένων, αφού δεν παρέχει επικοινωνία μεταξύ αντικειμένων, αλλά επικοινωνία μεταξύ διαδικασιών. Ένα μήνυμα XML-RPC είναι μία αίτηση HTTP-POST. Σαν στρώμα

μεταφοράς για αυτά τα μηνύματα χρησιμοποιείται το HTTP, το οποίο μπορούν να χρησιμοποιήσουν όλες οι σύγχρονες γλώσσες προγραμματισμού. Το σώμα της αίτησης είναι γραμμένο σε XML. Αποτέλεσμα της αίτησης είναι η εκτέλεση κάποιας διαδικασίας στον εξυπηρετητή και επιστροφή του αποτελέσματος σε μήνυμα γραμμένο και πάλι σε XML. Αυτή είναι και η διαφορά του XML-RPC από τις τεχνολογίες κατανεμημένων αντικειμένων που προαναφέραμε. Ενώ στις τεχνολογίες RMI, CORBA και DCOM υποστηρίζεται επικοινωνία μεταξύ κατανεμημένων αντικειμένων, στο XML-RPC επιτρέπεται μόνο η κλήση μεθόδων των αντικειμένων. Αυτό σημαίνει στο κέντρο του προγραμματισμού δεν υπάρχουν πλέον τα ίδια τα αντικείμενα, αλλά οι υπηρεσίες που μπορούν αυτά να προσφέρουν.

Το σημαντικότερο ίσως πλεονέκτημα της τεχνολογίας XML-RPC είναι ότι επιτρέπει τη συνεργασία ανάμεσα σε ανόμοια συστήματα. Λόγω του ότι χρησιμοποιείται το HTTP σαν πρωτόκολλο μεταφοράς, έχουν ήδη δημιουργηθεί υλοποιήσεις των προδιαγραφών XML-RPC σε πολλές γλώσσες προγραμματισμού. Αυτό έχει ως αποτέλεσμα τη διαλειτουργικότητα εφαρμογών που έχουν υλοποιηθεί σε διαφορετικές γλώσσες, αλλά χρησιμοποιούν το XML-RPC για επικοινωνία. Ένα άλλο πλεονέκτημα της συγκεκριμένης τεχνολογίας είναι ότι χρησιμοποιεί σαν γλώσσα μηνυμάτων την XML, η οποία μπορεί να διαβαστεί με σχετική ευκολία και από τον άνθρωπο, κάνοντας την τεχνολογία XML-RPC πιο βολική στην ανάπτυξη εφαρμογών.

Ο τρόπος λειτουργίας του XML-RPC φαίνεται στο Σχήμα 8.



Σχήμα 8. Η τεχνολογία XML – RPC.

3.6. Υπηρεσίες Ιστού

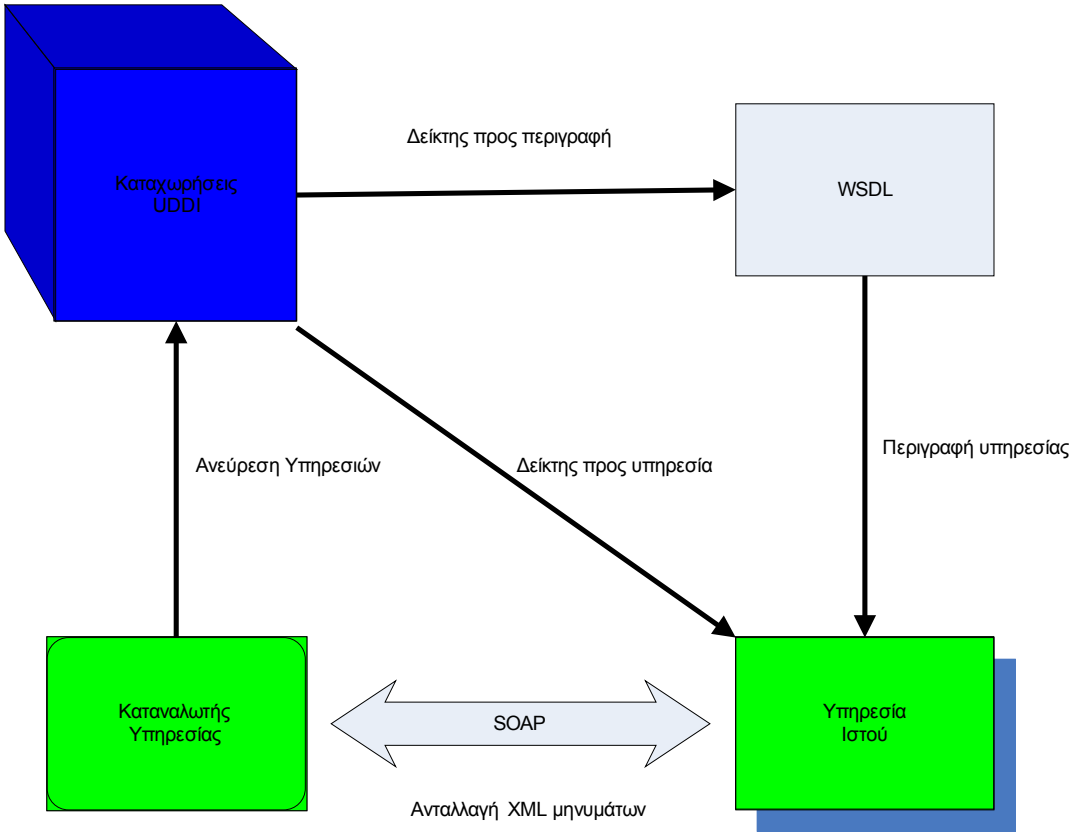
Μια υπηρεσία ιστού (Web Service) είναι μια εφαρμογή ή ένα συστατικό εφαρμογής υπηρεσίας λογισμικού, του οποίου οι διεπαφές μπορούν να περιγραφούν από καθορισμένες προδιαγραφές και υποστηρίζει απευθείας αλληλεπίδραση με άλλες εφαρμογές λογισμικού ή συστατικά εφαρμογών μέσω πρωτοκόλλων διαδικτύου [11]. Οι υπηρεσίες ιστού είναι δυνατόν να αναπτυχθούν χρησιμοποιώντας οποιαδήποτε γλώσσα προγραμματισμού και μπορούν να εγκατασταθούν σε οποιαδήποτε πλατφόρμα. Επίσης, μπορούν να επικοινωνούν μεταξύ τους γιατί όλες χρησιμοποιούν σαν γλώσσα επικοινωνίας την XML(Extensible Markup Language). Η XML χρησιμοποιείται τόσο για την περιγραφή των διεπαφών μιας υπηρεσίας, όσο και για την κωδικοποίηση των μηνυμάτων. Έτσι, οι υπηρεσίες ιστού επικοινωνούν με τη

βοήθεια συνηθισμένων πρωτοκόλλων ιστού χρησιμοποιώντας XML διεπαφές και XML μηνύματα, τα οποία μπορούν να διερμηνευτούν από οποιαδήποτε εφαρμογή.

Η χρήση της XML όμως δεν είναι από μόνη της ικανή να διασφαλίσει ανέξοδη επικοινωνία. Οι εφαρμογές χρειάζονται καθορισμένες μορφές και πρωτόκολλα που επιτρέπουν τη σωστή διερμηνεία της XML. Για αυτό το λόγο έχουν δημιουργηθεί τρεις βασικές τεχνολογίες που στηρίζονται στην XML και χρησιμοποιούνται ως προδιαγραφές για τις υπηρεσίες ιστού:

- το απλό πρωτόκολλο πρόσβασης αντικειμένου (Simple Object Access Protocol – SOAP)
- η γλώσσα περιγραφής υπηρεσιών ιστού (Web Services Description Language – WSDL)
- η καθολική περιγραφή, ανακάλυψη και ολοκλήρωση (Universal Description, Discovery and Integration – UDDI)

Στο Σχήμα 9 φαίνεται ο τρόπος που αλληλεπιδρούν οι τεχνολογίες αυτές για τη δημιουργία μιας εφαρμογής υπηρεσίας ιστού. Στις επόμενες παραγράφους γίνεται μια λεπτομερέστερη παρουσίαση των τριών αυτών τεχνολογιών.



Σχήμα 9. Δομή μιας εφαρμογής υπηρεσιών ιστού.

3.6.1 SOAP

Το SOAP είναι ένα επεκτάσιμο πρωτόκολλο για μηνύματα XML και αποτελεί τη βάση για τις υπηρεσίες ιστού. Το SOAP παρέχει έναν απλό και αξιόπιστο μηχανισμό που επιτρέπει σε μια εφαρμογή να στέλνει μηνύματα XML σε κάποια άλλη

εφαρμογή. Στη βάση του, το SOAP υποστηρίζει επικοινωνία μεταξύ δύο κόμβων. Ένα μήνυμα SOAP αποτελεί μια μονόδρομη μετάδοση από έναν αποστολέα σε ένα δέκτη, οπότε κάθε εφαρμογή μπορεί να συμμετέχει στη συνδιαλλαγή είτε ως αποστολέας, είτε ως δέκτης. Τα μηνύματα SOAP μπορούν να συνδυαστούν έτσι ώστε να υποστηρίζουν διάφορες συμπεριφορές επικοινωνίας, όπως αίτηση/απόκριση και γνωστοποίηση. Το πρωτόκολλο SOAP αποτελείται από τέσσερα βασικά συστατικά:

● Φάκελος SOAP (SOAP Envelope): Ο φάκελος SOAP παρέχει ένα μηχανισμό για την αναγνώριση των περιεχομένων ενός μηνύματος και για την επεξήγηση του τρόπου με τον οποίο πρέπει να γίνει η επεξεργασία του. Αποτελείται από την επικεφαλίδα SOAP και το σώμα SOAP. Η επικεφαλίδα SOAP παρέχει έναν επεκτάσιμο μηχανισμό για την παροχή πληροφορίας διευθυνσιοδότησης και ελέγχου για το μήνυμα. Το σώμα του μηνύματος περιέχει την ωφέλιμη πληροφορία (payload) που μεταφέρεται μέσα στο μήνυμα.

● Υποδομή για χρησιμοποίηση πρωτοκόλλων μεταφοράς (SOAP Transport Binding Framework): Αποτελεί μια υποδομή για την ανταλλαγή φακέλων SOAP χρησιμοποιώντας κάποιο υποκείμενο πρωτόκολλο, όπως για παράδειγμα το HTTP.

● Υποδομή σειριοποίησης (SOAP Serialization Framework): Οι υπηρεσίες μπορεί να σχεδιάζονται για εφαρμογή απευθείας στο φορτίο XML, αλλά είναι συνηθέστερο το φορτίο να αντιστοιχίζεται απευθείας σε τύπους δεδομένων της γλώσσας του υπολογιστή – παραλήπτη. Αυτό το βήμα ονομάζεται σειριοποίηση και υλοποιείται από τις περισσότερες εφαρμογές υπηρεσιών ιστού. Η σειριοποίηση είναι δυνατή ανεξάρτητα από το αν το φορτίο είναι ένα πραγματικό XML έγγραφο ή κωδικοποιημένο κατά SOAP, αφού και για τις δύο περιπτώσεις υπάρχει υποστήριξη από τις περισσότερες γλώσσες προγραμματισμού.

● Αναπαράσταση Απομακρυσμένης Κλήσης Διαδικασίας (SOAP RPC Representation): Η SOAP RPC αναπαράσταση ορίζει κάποιες προγραμματιστικές συμβάσεις που αντιπροσωπεύουν αιτήσεις και απαντήσεις απομακρυσμένων κλήσεων διεργασίας. Χρησιμοποιώντας το SOAP RPC, ο προγραμματιστής σχηματίζει μια αίτηση SOAP σαν μια κλήση σε μέθοδο με κάποια συγκεκριμένη δομή. Εδώ πρέπει να σημειωθεί ότι το SOAP υποστηρίζει και ένα χαλαρά συζευγμένο είδος επικοινωνίας μεταξύ δύο εφαρμογών. Ο αποστολέας μπορεί να στείλει ένα μήνυμα SOAP και να αφήσει τον δέκτη να αποφασίσει τον τρόπο με τον οποίο θα το χειριστεί. Έτσι, ο αποστολέας δεν χρειάζεται να γνωρίζει τίποτα για την υλοποίηση του δέκτη, πέρα από τη μορφή του μηνύματος και τη διεύθυνση πρόσβασης του δέκτη.

Πρόσφατα το πρωτόκολλο SOAP έχει αναβαθμιστεί ώστε να υποστηρίζεται και η μεταφορά συνημμένων δεδομένων. Αυτό σημαίνει ότι τα μηνύματα SOAP μπορούν να χρησιμοποιηθούν για μεταφορά πληροφορίας που δεν βρίσκεται σε μορφή XML, όπως για παράδειγμα αρχεία πολυμέσων. Για το σκοπό έχουν χρησιμοποιηθεί διάφοροι μηχανισμοί, όπως οι MIME [12], DIME και οι προδιαγραφές WS – Attachment [13].

3.6.2 WSDL

Η γλώσσα WSDL αποτελεί ένα λεξικό σε μορφή XML για την περιγραφή μιας υπηρεσίας ιστού. Ένα έγγραφο WSDL περιγράφει τη λειτουργικότητα που προσφέρει μία υπηρεσία ιστού, τον τρόπο που επικοινωνεί και τη διεύθυνση στην οποία αυτή είναι διαθέσιμη. Η WSDL προσφέρει ένα δομημένο μηχανισμό περιγραφής των

λειτουργιών που μπορεί να εκτελέσει μια υπηρεσία ιστού, τη μορφή των μηνυμάτων που υποστηρίζει και το σημείο πρόσβασης ενός στιγμιότυπου της υπηρεσίας. Τα εργαλεία ανάπτυξης SOAP μπορούν να χρησιμοποιήσουν μια WSDL περιγραφή για να δημιουργήσουν αυτόματα μια διεπαφή SOAP για μια εφαρμογή.

Μια περιγραφή WSDL ορίζει μια υπηρεσία ως μια συλλογή από τερματικά σημεία δικτύου (network endpoints) ή θύρες (ports). Κάθε θύρα ορίζεται αφηρημένα σαν ένας τύπος θύρας, ο οποίος υποστηρίζει ένα σύνολο λειτουργιών. Κάθε λειτουργία επεξεργάζεται ένα δεδομένο σύνολο μηνυμάτων. Ένας κανόνας αντιστοίχισης αντιστοιχεί έναν τύπο θύρας σε ένα συγκεκριμένο πρωτόκολλο και συγκεκριμένες μορφές δεδομένων. Μια θύρα στιγμιοποιεί έναν τύπο θύρας και μια αντιστοίχιση σε μια συγκεκριμένη διεύθυνση δικτύου.

Μια περιγραφή WSDL είναι ένα έγγραφο XML που περιέχει ένα σύνολο από ορισμούς. Μέσα σε κάθε τέτοιο έγγραφο υπάρχουν έξι βασικά στοιχεία:

- τύπος: το στοιχείο αυτό ορίζει τους τύπους δεδομένων που χρησιμοποιούνται μέσα στα μηνύματα.
- μήνυμα: ορίζει τη μορφή των μηνυμάτων, που χρησιμοποιούνται σαν δομές εισόδου ή εξόδου για τις διάφορες λειτουργίες.
- τύπος θύρας: ορίζει το σύνολο των λειτουργιών, καθώς και τη μορφή των μηνυμάτων που απαιτούνται ως είσοδοι και έξοδοι για την κάθε λειτουργία.
- αντιστοίχιση: αντιστοιχεί τις λειτουργίες και τα μηνύματα που ορίζονται στο στοιχείο τύπου θύρας με κάποιο συγκεκριμένο πρωτόκολλο και κάποια συγκεκριμένη προδιαγραφή τύπου δεδομένων.
- υπηρεσία: ορίζει ένα σύνολο από συσχετιζόμενες θύρες.
- θύρα: δημιουργεί μια σύνδεση ανάμεσα στο στοιχείο της αντιστοίχισης και την τοποθεσία (URL) ενός στιγμιότυπου μιας υπηρεσίας ιστού.

3.6.3 UDDI

Το UDDI αποτελεί ένα σύνολο από πρωτόκολλα και ένα δημόσιο κατάλογο για την καταχώρηση και την ανεύρεση υπηρεσιών σε πραγματικό χρόνο. Το ίδιο το UDDI είναι μια υπηρεσία ιστού που παρέχει μια λίστα με τις υπηρεσίες που υπάρχουν διαθέσιμες σε ένα δίκτυο. Το κύριο πλεονέκτημα που παρουσιάζει η χρήση του είναι η παροχή ενός μόνο σημείου αναφοράς για όλες τις υπηρεσίες σε ένα δίκτυο.

Όταν ένας προγραμματιστής αναζητά κάποια υπηρεσία ιστού, ψάχνει στον κατάλογο του UDDI για κάποια επιχείρηση που προσφέρει τον τύπο υπηρεσιών που επιθυμεί. Από τις καταχωρήσεις που υπάρχουν στον κατάλογο, μπορεί να αποκτήσει μια περιγραφή WSDL που περιγράφει τη διεπαφή της υπηρεσίας, καθώς και το σημείο πρόσβασης για την υπηρεσία. Επίσης, χρησιμοποιώντας την περιγραφή WSDL, ο προγραμματιστής μπορεί να κατασκευάσει μία SOAP διεπαφή πελάτη που μπορεί να επικοινωνεί με την υπηρεσία ιστού. Με τον ίδιο τρόπο ένα πρόγραμμα πελάτη μπορεί κατά την εκτέλεσή του να χρησιμοποιεί τον κατάλογο UDDI σαν εξυπηρετητή ονομάτων. Συγκεκριμένα, ο πελάτης μπορεί να ζητήσει από το UDDI να του παράσχει το σημείο πρόσβασης κάποιας υπηρεσίας με κάποιο δεδομένο όνομα ή κάποιου συγκεκριμένου είδους, και να χρησιμοποιήσει το URL της υπηρεσίας δυναμικά.

Σε αυτό το σημείο πρέπει να σημειωθεί ότι το UDDI είναι περιοχή καταχώρησης που μπορεί να χρησιμοποιηθεί για πολλαπλούς σκοπούς. Σε αυτό μπορεί να καταχωρηθεί οποιοδήποτε είδος υπηρεσίας, και όχι μόνο υπηρεσίες ιστού, αφού το ίδιο δεν απαιτεί οι υπηρεσίες που καταχωρούνται να υποστηρίζουν τις διεπαφές

SOAP ή να περιγράφονται χρησιμοποιώντας WSDL. Για την ακρίβεια, και οι τρεις τεχνολογίες που προαναφέρθηκαν μπορούν να λειτουργήσουν ανεξάρτητα η μία από την άλλη, αν και τα οφέλη από την ταυτόχρονη χρησιμοποίηση και των τριών είναι πολλά.

3.7. Κινητοί Πράκτορες

Ένας πράκτορας λογισμικού είναι μια προσανατολισμένη σε κάποιο σκοπό υπολογιστική οντότητα που μπορεί να αντιδρά σε ερεθίσματα του περιβάλλοντός της και η οποία έχει ένα μεγάλο βαθμό αυτονομίας στο να επιλέξει τις λειτουργίες που θα εκτελέσει για να επιτύχει αυτό το σκοπό. Για πρώτη φορά ο όρος κινητός πράκτορας εμφανίστηκε το 1994 με τη γλώσσα προγραμματισμού Telescript. Λόγω της σημασίας της συγκεκριμένης τεχνολογίας και της συνάφειας με την παρούσα εργασία γίνεται αναλυτική παρουσίασή της στο επόμενο κεφάλαιο.

3.8. Πλέγμα

Η έννοια του υπολογιστικού πλέγματος έχει αναδειχτεί τα τελευταία χρόνια σε ένα σημαντικό νέο πεδίο της επιστήμης των υπολογιστών, που συχνά διαχωρίζεται από τις υπόλοιπες κατανεμημένες τεχνολογίες. Ο λόγος για τη διάδοση της τεχνολογίας αυτής είναι οι προοπτικές που δίνει για κοινή χρήση πόρων σε μεγάλη κλίμακα και για καινοτόμες εφαρμογές, καθώς και ο προσανατολισμός της σε υπολογισμούς υψηλής απόδοσης [14]. Ένα υπολογιστικό πλέγμα αποτελείται από κατανεμημένες υποδομές λογισμικού και υλικού, όπως υπολογιστικοί πόροι, συστήματα αποθήκευσης, κατάλογοι, πόροι δικτύου και πιθανώς διάφορα επιστημονικά όργανα. Ένας πόρος μπορεί να είναι μεταξύ άλλων και μία λογική οντότητα, όπως ένα κατανεμημένο σύστημα αρχείων ή μία συστάδα υπολογιστών. Πρέπει να σημειωθεί ότι το υπολογιστικό πλέγμα δεν είναι μια κλασική τεχνολογία κατανεμημένων αντικειμένων όπως παρουσιάζονται σε αυτό το κεφάλαιο, με την έννοια ότι δεν περιορίζεται μόνο στην κατανομή αντικειμένων κώδικα, αλλά στοχεύει και στην κατανομή οποιουδήποτε άλλου αντικειμένου που παίρνει μέρος σε κάποιους υπολογισμούς, ακόμη και υλικού.

3.8.1 Αρχιτεκτονική Υπολογιστικού Πλέγματος

Η αρχιτεκτονική του περιβάλλοντος ενός πλέγματος ακολουθεί τη λογική της διαστρωμάτωσης. Συγκεκριμένα, κάθε αρχιτεκτονική αποτελείται από τρία στρώματα [15]:

- το φυσικό στρώμα (fabric layer): πρόκειται για το χαμηλότερο στην ιεραρχία στρώμα, το οποίο περιλαμβάνει όλα όσα πρέπει να ενώσει το πλέγμα. Περιλαμβάνει όλους τους πόρους που ανήκουν στο πλέγμα, βρίσκονται σε όλον τον κόσμο και ενώνονται μέσω του Internet. Αυτοί οι πόροι μπορεί να είναι είτε υλικές οντότητες (όπως επεξεργαστές, μνήμη, ηλεκτρονικές συσκευές ή δίκτυο), είτε λογισμικό (όπως συστατικά εφαρμογών, βάσεις δεδομένων), είτε ακόμη και λογικές οντότητες (π.χ. συστάδες υπολογιστών). Όλα τα πακέτα που παρέχουν βασικές υπηρεσίες σε τοπικές περιοχές επίσης ανήκουν σε αυτό το επίπεδο. Παραδείγματα τέτοιων υπηρεσιών είναι

οι τοπικοί διαχειριστές πόρων, οι κατανεμημένοι διαχειριστές συστημάτων, τα λειτουργικά συστήματα, καθώς και εργαλεία ασφάλειας. Βασικό χαρακτηριστικό του επιπέδου αυτού είναι η δυναμική φύση της σύστασής του, αφού οι πόροι που περιλαμβάνει μπορεί να μεταβάλλονται με το χρόνο.

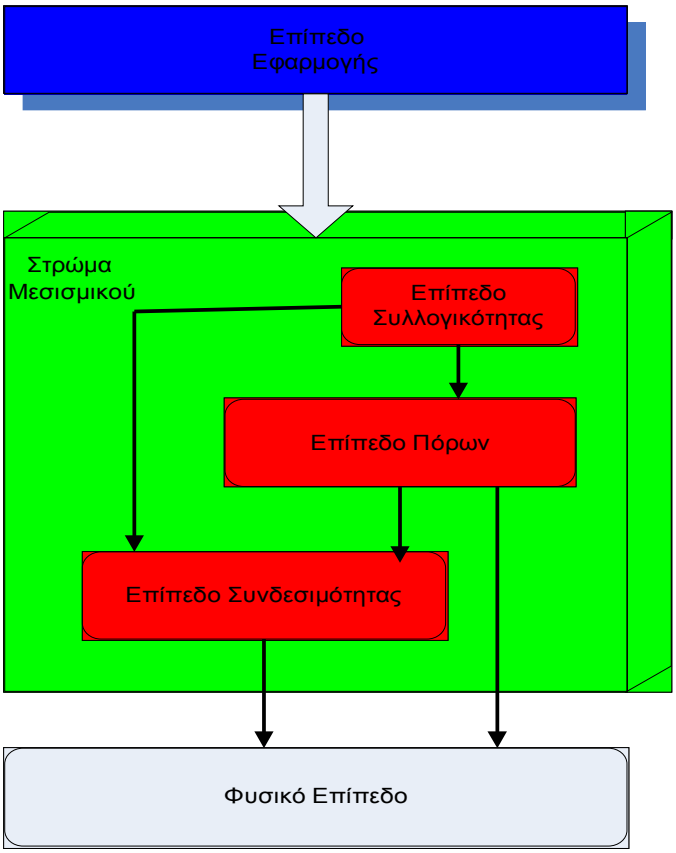
- στρώμα μεσισμικού (middleware level): το στρώμα αυτό περιλαμβάνει το λογισμικό που είναι υπεύθυνο για τη μεσολάβηση μεταξύ των πόρων και των διαχειριστών υψηλού επιπέδου. Τοποθετείται αμέσως μετά το φυσικό στρώμα, με σκοπό να αποκρύψει από τους τελικούς χρήστες του πλέγματος και τους προγραμματιστές εφαρμογών την πολυπλοκότητα του φυσικού στρώματος. Το πλεγματικό μεσισμικό λειτουργεί πάνω σε πλεγματικούς πόρους και τοπικούς διαχειριστές για την παροχή βασικών υπηρεσιών σε κατανεμημένες εφαρμογές. Επίσης, το στρώμα αυτό περιέχει βιβλιοθήκες και γλώσσες προσανατολισμένες στην υλοποίηση πλεγμάτων.
- στρώμα εφαρμογής (application level): αυτό είναι το στρώμα με το οποίο αλληλεπιδρούν οι τελικοί χρήστες. Περιλαμβάνει υψηλού επιπέδου υπηρεσίες που επιτρέπουν την ανάπτυξη πλεγματικών εφαρμογών, καθώς και εργαλεία ιστού που επιτρέπουν στους τελικούς χρήστες να χρησιμοποιήσουν το πλέγμα. Οποιαδήποτε εφαρμογή προσανατολισμένη στη λογική του πλέγματος ανήκει σε αυτό το επίπεδο.

3.8.2 Πλεγματικό Μεσισμικό

Το σημαντικότερο ίσως συστατικό μιας αρχιτεκτονικής πλέγματος είναι το μεσισμικό. Γενικά οι πλεγματικοί πόροι υποτίθεται ότι είναι κατανεμημένοι γεωγραφικά και ότι ανήκουν σε διαφορετικούς οργανισμούς, με διαφορετικές πολιτικές ο καθένας όσον αφορά την ασφάλεια, τη χρησιμοποίηση πόρων, τη συντήρηση πλατφόρμας και άλλα θέματα. Ένα τέτοιο περιβάλλον στηρίζεται κατά μεγάλο μέρος στην κατασκευή μιας υποδομής παροχής βασικών υπηρεσιών ικανής να εξομαλύνει τις όποιες αναντιστοιχίες υπάρχουν μεταξύ διαφορετικών μηχανών. Αυτόν ακριβώς το ρόλο καλείται να διαδραματίσει το μεσισμικό. Το μεσισμικό μπορεί επιπλέον να διαχωριστεί σε επιμέρους στρώματα [14], τα οποία από κάτω προς τα πάνω είναι τα εξής:

- στρώμα συνδεσιμότητας (connectivity layer): το συγκεκριμένο στρώμα ορίζει βασικά πρωτόκολλα επικοινωνίας και αυθεντικοποίησης που χρειάζονται για τις όποιες δικτυακές συνδιαλλαγές γίνονται στο πλέγμα. Τα πρωτόκολλα επικοινωνίας επιτρέπουν την ανταλλαγή δεδομένων μεταξύ πόρων στο φυσικό επίπεδο, ενώ τα πρωτόκολλα αυθεντικοποίησης κατασκευάζονται πάνω στους μηχανισμούς επικοινωνίας για την παροχή κρυπτογραφικά ασφαλών μηχανισμών για την εξακρίβωση της ταυτότητας χρηστών και πόρων.
- στρώμα πόρων (resource layer): το στρώμα αυτό χρησιμοποιεί τα πρωτόκολλα επικοινωνίας και αυθεντικοποίησης του προηγούμενου στρώματος για τον ορισμό νέων πρωτοκόλλων για την ασφαλή διαπραγμάτευση, αρχικοποίηση, παρακολούθηση και έλεγχο των διαμοιραζόμενων λειτουργιών ή των μεμονωμένων πόρων. Τα πρωτόκολλα του επιπέδου αυτού χρησιμοποιούν λειτουργίες του φυσικού επιπέδου για την πρόσβαση και τον έλεγχο τοπικών πόρων.
- συλλογικό στρώμα (collective layer): σε αντίθεση με το προηγούμενο στρώμα, το συλλογικό στρώμα περιέχει πρωτόκολλα και υπηρεσίες που δεν σχετίζονται με κάποιο συγκεκριμένο πόρο, αλλά είναι από τη φύση τους καθολικά και απευθύνονται σε αλληλεπιδράσεις μεταξύ ομάδων από πόρους. Αυτά τα πρωτόκολλα και οι αυτές οι υπηρεσίες χρησιμοποιούν και τα δύο προηγούμενα στρώματα του μεσισμικού.

Η συνολική μορφή της αρχιτεκτονικής του πλέγματος φαίνεται στο Σχήμα 10.



Σχήμα 10. Η γενική αρχιτεκτονική του υπολογιστικού πλέγματος.

3.8.3 OGSA - Globus Toolkit

Στις προηγούμενες παραγράφους έγινε περιγραφή της γενικής μορφής της αρχιτεκτονικής ενός πλέγματος. Πέρα από αυτό το γενικό σχήμα έχουν προταθεί και άλλες ειδικότερες αρχιτεκτονικές, η πιο διαδεδομένη από τις οποίες είναι η Open Grid Services Architecture (OGSA) [16]. Βασικό χαρακτηριστικό αυτής της αρχιτεκτονικής είναι ότι τα πάντα αναπαρίστανται από μια υπηρεσία, δηλαδή μια δικτυακά ενεργοποιημένη οντότητα που προσφέρει λειτουργίες μέσω ανταλλαγής μηνυμάτων. Οι υπολογιστικοί πόροι, οι αποθηκευτικοί πόροι, τα δίκτυα, τα προγράμματα, οι βάσεις δεδομένων και οτιδήποτε άλλο περιλαμβάνεται σε ένα πλέγμα, θεωρούνται υπηρεσίες, ή για την ακρίβεια, πλεγματικές υπηρεσίες, δηλαδή υπηρεσίες που ακολουθούν ένα σύνολο συμβάσεων. Οι συμβάσεις αυτές εκφράζονται με χρήση της γλώσσας WSDL που περιγράφτηκε στην παράγραφο 3.6.2 και προορίζονται για τη διαχείριση και την περιγραφή των χαρακτηριστικών της υπηρεσίας.

Αυτό το προσανατολισμένο στις υπηρεσίες μοντέλο καθιστά όλα τα συστατικά του περιβάλλοντος εικονικές οντότητες. Παρέχοντας ένα σύνολο από διεπαφές από τις οποίες υλοποιούνται όλες οι πλεγματικές υπηρεσίες, επιτυγχάνεται η κατασκευή ιεραρχικών υπηρεσιών που μπορούν να αντιμετωπίζονται ενιαία μέσω μιας λογικής αφηρημένων επιπέδων. Αυτή η εικονικοποίηση επιτρέπει την αντιστοίχιση πολλαπλών στιγμιοτύπων λογικών πόρων στον ίδιο φυσικό πόρο και γενικά τη

σύνθεση υπηρεσιών ανεξάρτητα από την υλοποίηση των πόρων του χαμηλότερου επιπέδου. Μια υλοποίηση που στηρίζεται στην αρχιτεκτονική OGSA είναι το Globus Toolkit [17,18]. Πρόκειται για ένα σύνολο open-source βιβλιοθηκών λογισμικού που υποστηρίζουν τη δημιουργία πλεγμάτων και πλεγματικών εφαρμογών. Το Globus Toolkit αντιμετωπίζει ζητήματα ασφάλειας, ανακάλυψης πληροφορίας, διαχείρισης πόρων και δεδομένων, επικοινωνίας, ανίχνευσης σφαλμάτων και φορητότητας.

3.9. Αναφορές

[1] Kevin Taylor, "What is Middleware", URL: http://java.about.com/b/a/099316.htm

[2] RMI Specification – URL: http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html

[3] Java On Line Training – RMI, URL: http://java.sun.com/developer/onlineTraining/rmi/RMI.html

[4] Java Tutorial – RMI

[5] Object Management Group et. al, "The Common Object Request Broker: Architecture and Specification, Version 2.1", August 1997.

[6] Object Management Group et al, "CORBAservices: Common Object Services Specification", July 1997.

[7] Common Object Request Broker Architecture (CORBA/IIOP) Specifications, v3.0, July 2002

[8] N. Brown, C. Kindel, "Distributed Component Object Model Protocol" Microsoft Corporation, January, 1998.

[9] R. Salz, DCE 1.2 Contents Overview, Open Group RFC 63.3, The Open Group, October 1996

[10] Dave Winer, XML-RPC Specification, UserLand Software, http://www.xmlrpc.com/spec.

[11] Web Services Architecture, W3C , URL: http://www.w3c.org/2000/ws/

[12] SOAP with Attachments binding, URL: http://www.w3.org/TR/SOAPattachments

[13] WS Attachment Specification, URL: http://www.106.ibm.com/developerworks/webservices/library/ws-attach.html

[14] Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," Int. Journal of High Performance Computing Applications, Vol. 15, No. 3, 2001, pp. 200–222.

[15] Baker, M., R. Buyya and D. Laforenza, "The Grid: International Efforts in Global Computing," International Conference on Advances in Infrastructure for Electronic Business, Science and Education on the Internet, Italy, 2000.

[16] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, The Physiology of the Grid An Open Grid Services Architecture for Distributed Systems Integration, in Open Grid Service Infrastructure WG, Global Grid Forum, 2002.

[17] The Globus Toolkit, URL: http://www-unix.globus.org/toolkit/

[18] Foster, I. and Kesselman, C. Globus: A Toolkit-Based Grid Architecture. In Foster, I. and Kesselman, C. eds. The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999, 259-278.

4. Κινητοί Αντιπρόσωποι

Με τον γενικό όρο «πράκτορας λογισμικού» [1] αναφερόμαστε σε μια αυτόνομη οντότητα λογισμικού, η οποία αυτοματοποιεί κάποιες δραστηριότητες που ένας άνθρωπος ή οποιαδήποτε διεργασία λογισμικού μπορεί να έχει μεταβιβάσει σε αυτήν. Κάθε πράκτορας λογισμικού έχει τη δική του πληροφορία κατάστασης, τη δική του συμπεριφορά καθώς και το δικό του νήμα ελέγχου και μπορεί να αλληλεπιδρά ελεύθερα με άλλες οντότητες με σκοπό την επίλυση κάποιου συγκεκριμένου προβλήματος.

4.1. Ιστορική Αναδρομή

Η έννοια του πράκτορα λογισμικού υπάρχει από τη δεκαετία του 1970, όταν οι Henry Baker και Carl Hewitt πρότειναν το μοντέλο του «ηθοποιού» (Actor model) [2]. Σε αυτό το μοντέλο, ο Hewitt πρότεινε την έννοια ενός αντικειμένου αυτόνομου και διαδραστικού που μπορεί να εκτελείται ταυτόχρονα με άλλα τέτοια όμοια αντικείμενα. Αυτό το μοντέλο αντικειμένου περιείχε ενσωματωμένη μια εσωτερική κατάσταση και μπορούσε να απαντά σε μηνύματα που προέρχονταν από άλλα όμοια αντικείμενα. Η έρευνα που ξεκίνησε τότε, και που ακόμη συνεχίζεται σήμερα, επικεντρωνόταν κυρίως στην αλληλεπίδραση και την επικοινωνία μεταξύ των αντιπροσώπων, τη διάσπαση και την κατανομή εργασιών, το συντονισμό και τη συνεργασία μεταξύ των πρακτόρων, την επίλυση συγκρούσεων και άλλα παρεμφερή θέματα.

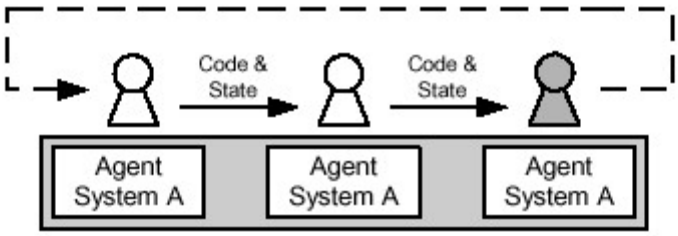
Από τις αρχές της δεκαετίας του 1990 άρχισε να γίνεται μια πιο εξειδικευμένη έρευνα στα διαφορετικά πιθανά είδη των πρακτόρων λογισμικού και γενικά σε αυτό που συχνά ονομάζεται «τυπολογία» των πρακτόρων [3]. Από αυτήν την περίοδο ξεκινάει και η κύρια εξέλιξη στην τεχνολογία των Κινητών Αντιπροσώπων. Με την ταυτόχρονη εμφάνιση κάποιων παρεμφερών εννοιών και τεχνολογιών, άρχισε να γίνεται διάκριση ανάμεσα σε Ευφυείς και Κινητούς Αντιπρόσωπους. Λίγο αργότερα εμφανίστηκαν και τα πρώτα συστήματα πολλαπλών πρακτόρων (Multi Agent Systems – MAS), τα οποία με τη σειρά τους χωρίζονται σε τρεις γενικές κατηγορίες: Συστήματα Κατανεμημένης Τεχνητής Νοημοσύνης (Distributed Artificial Intelligence - DAI), Συστήματα Παράλληλης Τεχνητής Νοημοσύνης (Parallel Artificial Intelligence – PAI) και Συστήματα Κατανεμημένης Επίλυσης Προβλημάτων (Distributed Problem Solving - DPS).

Την τελευταία δεκαετία έχουν γίνει αρκετές προσπάθειες για την ανάπτυξη συστημάτων κινητών πρακτόρων και τεχνολογιών που θα διευκολύνουν και θα κάνουν πιο αποδοτική τη χρήση τους. Το 1994 αναπτύχθηκε από την General Magic η γλώσσα TeleScript και στη συνέχεια άλλες διερμηνευόμενες γλώσσες, (scripting languages) όπως η Tool Command Language (TCL) και η SafeTCL που επέτρεπαν τη γρήγορη προτυποποίηση (prototyping) και παραγωγή μεταφέρσιμου κώδικα. Η TCL ήταν στην πραγματικότητα ένα πλήρες περιβάλλον εκτέλεσης κινητών πρακτόρων. Με την εμφάνιση της Java δόθηκε νέα ώθηση στην ανάπτυξη της τεχνολογίας των Κινητών Αντιπροσώπων. Η Java παρέχει εγγενώς πολλές δυνατότητες για κινητικότητα κώδικα, εξαιτίας της πρότυπης διαδικασίας σειριοποίησης αντικειμένων (object serialization) που εφαρμόζει καθώς και το δυναμικό φόρτωμα κώδικα κατά τη διάρκεια της εκτέλεσης ενός προγράμματος. Στη συνέχεια άρχισαν να εμφανίζονται και κάποιες πλατφόρμες που παρέχουν την απαραίτητη υποδομή για τη χρήση

κινητών πρακτόρων, οι σημαντικότερες από τις οποίες είναι η Odyssey της General Magic, η Aglets της IBM [4], η Grasshopper του ινστιτούτου IKV [5] καθώς και η Jade της Tilab [6] που εμφανίστηκε το τελευταίο διάστημα και από τις οποίες οι περισσότερες αναπτύχθηκαν σε Java.

4.2. Χαρακτηριστικά Κινητών Αντιπροσώπων και Συστημάτων Πολλαπλών Πρακτόρων

Όπως προαναφέρθηκε, ένας πράκτορας λογισμικού ορίζεται σαν ένα αντικείμενο που έχει τη δική του συμπεριφορά, κατάσταση, καθώς και τοποθεσία. Οι κινητοί πράκτορες έχουν την ιδιότητα να μεταναστεύουν από έναν υπολογιστή σε έναν άλλο και να συνεχίζουν εκεί την εκτέλεσή τους από το σημείο που σταμάτησαν (κινητικότητα). Κατά τη μετακίνηση αυτή εκτός από τον κώδικα ο πράκτορας παίρνει μαζί του και την πληροφορία κατάστασης (τιμές τοπικών μεταβλητών, ορίσματα και τιμές τοπικών μεταβλητών ρουτινών που έχουν κληθεί στο ίδιο νήμα εκτέλεσης και δεν έχουν τερματίσει, στοίβα εκτέλεσης κτλ). Ένα από τα πλέον βασικά χαρακτηριστικά τους είναι η αυτονομία, αφού από τη στιγμή που θα ξεκινήσει η εκτέλεσή τους μπορούν αυτόνομα να αποφασίσουν σε ποιες τοποθεσίες θα μεταναστεύσουν και ποιον κώδικα θα εκτελέσουν. Οι κινητοί πράκτορες αποτελούν μορφή κινητού κώδικα. Σε αντίθεση με τα κλασσικά παραδείγματα της απομακρυσμένης αξιολόγησης (remote evaluation) και του κατ' απαίτηση κώδικα (code on demand), οι κινητοί πράκτορες είναι ενεργοί υπό την έννοια ότι μπορούν να μεταναστεύσουν μεταξύ υπολογιστών οποιαδήποτε στιγμή κατά την εκτέλεσή τους. Αυτή η ιδιότητα τους καθιστά ένα ιδιαίτερα ισχυρό εργαλείο στην ανάπτυξη κατανεμημένων εφαρμογών πάνω σε ένα δίκτυο υπολογιστών.



Σχήμα 11: Το υπολογιστικό παράδειγμα του Κινητού Αντιπροσώπου.

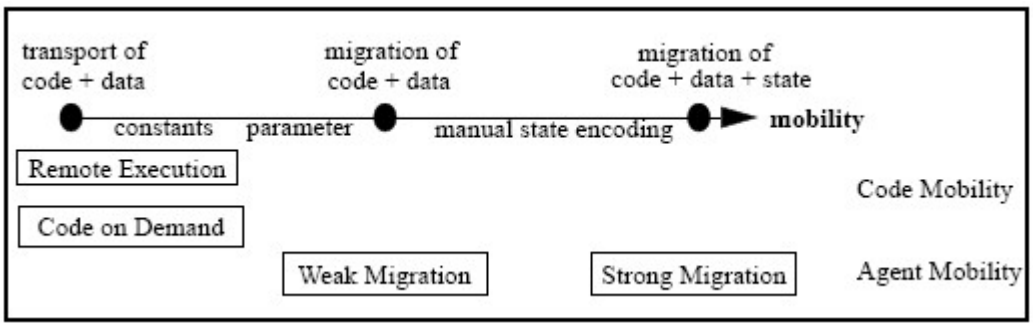
Εκτός από την κινητικότητα, την αυτονομία και την ενεργητικότητα, που αποτελούν τα βασικότερα χαρακτηριστικά που πρέπει να έχει ένας κινητός πράκτορας, υπάρχει και ένας αριθμός από άλλα στοιχεία που μπορεί να τον χαρακτηρίζουν [7], [8]:

- *διαδραστικότητα*: ένας κινητός πράκτορας πρέπει να έχει τη δυνατότητα να επικοινωνεί και να αλληλεπιδρά με το περιβάλλον του καθώς και με άλλους πράκτορες.
- *προσαρμοστικότητα*: οι κινητοί πράκτορες μπορούν να αποκρίνονται στο περιβάλλον τους ή σε άλλους πράκτορες και να προσαρμόζονται και οι ίδιοι κατάλληλα.

● *αντιπροσωπευτικότητα*: οι κινητοί πράκτορες μπορεί να ενεργούν εκ μέρους ή και προς όφελος κάποιου (ανθρώπου ή υπολογιστικής οντότητας). Για να είναι δυνατόν να γίνει κάτι τέτοιο οι κινητοί πράκτορες πρέπει να έχουν έναν ορισμένο βαθμό αυτονομίας.
● *πρωτοβουλία*: κάθε κινητός πράκτορας θα πρέπει να είναι προσανατολισμένος προς κάποιο συγκεκριμένο στόχο και να παίρνει κατάλληλες πρωτοβουλίες κάθε φορά που ανταποκρίνεται στο περιβάλλον του.
● *νοημοσύνη*: οι κινητοί αντιπρόσωποι μπορεί να έχουν νοημοσύνη βασισμένη σε κάποια γνώση, έτσι ώστε να γίνεται αποτελεσματικότερη η αλληλεπίδραση με το περιβάλλον τους.
● *συντονισμός*: οι κινητοί πράκτορες θα πρέπει να έχουν την ικανότητα να πραγματοποιούν μεταφορά δεδομένων που μοιράζονται μεταξύ των πρακτόρων ενός περιβάλλοντος με συγκεκριμένο τρόπο.
● *μάθηση*: αναφέρεται στην ικανότητα ενός κινητού πράκτορα να παίρνει πληροφορίες σχετικά με το περιβάλλον, οι οποίες θα του επιτρέψουν να προσαρμόσει ανάλογα τη συμπεριφορά του.
● *συνεργασία*: οι κινητοί πράκτορες πρέπει να συντονίζονται και να συνεργάζονται μεταξύ τους για την επίτευξη ενός κοινού στόχου.
● *ικανότητα ανάκαμψης*: ένας κινητός θα πρέπει να μπορεί να αντιμετωπίζει τα όποια σφάλματα εμφανίζονται κατά τη διάρκεια ζωής τους.

4.3. Μορφές Κινητικότητας

Οι κινητοί αντιπρόσωποι αποτελούν μια μορφή κινητού κώδικα. Διαφορετικοί βαθμοί κινητικότητας μπορούν να διακριθούν (σχήμα 12).



Σχήμα 12: Βαθμοί κινητικότητας.

Γενικά μπορούν να διακριθούν οι εξής μορφές κινητικότητας που σχετίζονται με τους κινητούς πράκτορες [9], [10]:

4.3.1 Απομακρυσμένη Εκτέλεση – Απομακρυσμένη Αξιολόγηση

Στην περίπτωση της Απομακρυσμένης Εκτέλεσης (Remote Execution), ο πράκτορας μεταφέρεται πριν την ενεργοποίησή του σε κάποιον απομακρυσμένο κόμβο, όπου και παραμένει ενεργός μέχρι τον τερματισμό του, π.χ. ένας πράκτορας που μεταφέρεται μόνο μία φορά. Η πληροφορία που μεταφέρεται περιλαμβάνει τον κώδικα του πράκτορα, καθώς και ένα σύνολο παραμέτρων. Όταν ενεργοποιηθεί, ο

ίδιος ο πράκτορας μπορεί να χρησιμοποιήσει το μηχανισμό της Απομακρυσμένης Εκτέλεσης για να ξεκινήσει την εκτέλεση άλλων πρακτόρων. Μια παρόμοια προσέγγιση αποτελεί η Απομακρυσμένη Αξιολόγηση (Remote Evaluation). Σε αυτήν την προσέγγιση, μία λειτουργία (π.χ. μία διαδικασία μαζί με τις παραμέτρους της) μεταφέρεται σε έναν απομακρυσμένο κόμβο, όπου και εκτελείται ολοκληρωτικά. Μετά την εκτέλεση του προγράμματος, ο απομακρυσμένος κόμβος επιστρέφει τα αποτελέσματα της λειτουργίας στον αρχικό κόμβο. Ο μηχανισμός της Απομακρυσμένης Αξιολόγησης μπορεί να εφαρμόζεται αναδρομικά, οπότε προκύπτει ως αποτέλεσμα ένα μοντέλο εκτέλεσης με δενδρική δομή.

4.3.2 Κώδικας Κατά Απαίτηση

Στην προηγούμενη περίπτωση ο προορισμός που πρέπει να μεταφερθεί ο πράκτορας καθορίζεται από την οντότητα που ξεκινάει την απομακρυσμένη εκτέλεση. Αντίθετα, στην περίπτωση του Κώδικα κατά Απαίτηση (Code on Demand), ο ίδιος ο προορισμός είναι αυτός που ξεκινάει τη μεταφορά του κώδικα του προγράμματος. Αν θεωρήσουμε το σχήμα του Κώδικα κατά Απαίτηση σε λειτουργία πελάτη/εξυπηρετητή, τότε προγράμματα που είναι αποθηκευμένα σε μηχανές – εξυπηρετητές μεταφέρονται στους πελάτες κατά απαίτηση (όπως για παράδειγμα συμβαίνει με τα Java Applets).

4.3.3 Ασθενής και Ισχυρή Μετανάστευση

Και τα δύο παραπάνω μοντέλα υποστηρίζουν κινητικότητα κώδικα και όχι κινητικότητα πράκτορα, αφού και στις δύο περιπτώσεις τα προγράμματα – πράκτορες μεταφέρονται πριν να αρχίσει η εκτέλεσή τους. Όπως προαναφέρθηκε όμως, για την μετανάστευση ενός πράκτορα πρέπει να μεταφερθεί στον προορισμό όχι μόνο ο κώδικας, αλλά και η κατάσταση του πράκτορα. Η κατάσταση του πράκτορα αποτελείται από την κατάσταση των δεδομένων (όπως για παράδειγμα οι καθολικές μεταβλητές και στατικές μεταβλητές κλάσεων) και την κατάσταση εκτέλεσης (όπως για παράδειγμα τοπικές μεταβλητές και παράμετροι καθώς και εκτελούμενα νήματα).

Ο υψηλότερος βαθμός κινητικότητας είναι η Ισχυρή Μετανάστευση [11]. Σε αυτό το μοντέλο, το υποκείμενο σύστημα παγιδεύει ολόκληρη την κατάσταση του πράκτορα (αποτελούμενη από δεδομένα και κατάσταση εκτέλεσης) και την μεταφέρει μαζί με τον κώδικα στην επόμενη τοποθεσία προορισμού. Όταν ο πράκτορας φτάσει στον προορισμό, η κατάστασή του επαναφέρεται αυτόματα. Από την πλευρά του προγραμματιστή το μοντέλο αυτό είναι πολύ ελκυστικό, καθώς η όλη διαδικασία της παγίδευσης της κατάστασης, της μεταφοράς και της επαναφοράς γίνεται με τελείως διαφανή τρόπο από το υποκείμενο σύστημα. Το πρόβλημα που προκύπτει είναι ότι για την παροχή αυτού του βαθμού διαφάνειας σε ετερογενή περιβάλλοντα, απαιτείται τουλάχιστον ένα καθολικό μοντέλο για την κατάσταση ενός πράκτορα, καθώς επίσης και ένα συντακτικό μεταφοράς για το συγκεκριμένο είδος πληροφορίας. Επιπλέον, ένα σύστημα πρακτόρων πρέπει να παρέχει λειτουργίες για την ανάκτηση και τη μεταφορά της κατάστασης ενός πράκτορα. Λίγες μόνο γλώσσες επιτρέπουν την ανάκτηση της κατάστασης εκτέλεσης σε τόσο υψηλό επίπεδο, όπως για παράδειγμα η Facile [12] και η Tycoon [13]. Τέλος, η πλήρης κατάσταση ενός πράκτορα (δεδομένα και κατάσταση εκτέλεσης) μπορεί να είναι αρκετά μεγάλη, όπως για παράδειγμα στην

περίπτωση των πολυνηματικών πρακτόρων, οπότε η ισχυρή μετανάστευση μπορεί να γίνει μια πολύ χρονοβόρα και ακριβή διαδικασία.

Όλες αυτές οι δυσκολίες έχουν οδηγήσει στο μοντέλο της ονομαζόμενης «Ασθενούς Μετανάστευσης», όπου μεταφέρεται μόνο η πληροφορία της κατάστασης δεδομένων και όχι και της κατάστασης εκτέλεσης. Το μέγεθος της πληροφορίας της μεταφερόμενης κατάστασης μπορεί να περιοριστεί ακόμη περισσότερο αφήνοντας τον προγραμματιστή να επιλέξει τις μεταβλητές που αποτελούν την κατάσταση του πράκτορα. Επιπλέον, ο προγραμματιστής πρέπει να παρέχει μια μέθοδο εκκίνησης που θα αποφασίζει με βάση την κωδικοποιημένη πληροφορία κατάστασης από που να συνεχίσει την εκτέλεση μετά από κάθε μετανάστευση. Αυτό το μοντέλο μειώνει δραστικά το μέγεθος της πληροφορίας που πρέπει να μεταφερθεί, αλλά βάζει μεγαλύτερο φόρτο στον προγραμματιστή και επιπλέον καθιστά τα προγράμματα – πράκτορες πολυπλοκότερα.

4.4. Πλεονεκτήματα και Εφαρμογές των Κινητών Πρακτόρων

Αναφορικά με τους κινητούς πράκτορες, αναφέρονται συνήθως τα εξής πλεονεκτήματα:

● Ασύγχρονη και αυτόνομη εκτέλεση διεργασιών: Μετά από την είσοδο ενός πράκτορα σε ένα κατανεμημένο υπολογιστικό σύστημα ο χρήστης μπορεί να εκτελέσει άλλες δραστηριότητες χωρίς να απαιτείται να αλληλεπιδρά με τον πράκτορα αυτό.

● Μείωση της δικτυακής κυκλοφορίας και της απαιτούμενης υπολογιστικής ισχύος στην μεριά του πελάτη: Καθώς ανταλλαγές δεδομένων πολύ μεγάλου όγκου γίνονται πλέον τοπικά από τους κόμβους που περιέχουν αυτήν την πληροφορία, οι υπολογιστές στην μεριά του εξυπηρετητή μπορούν να επικεντρωθούν στην εκτέλεση σχετικά απλών λειτουργιών. Αυτό το γεγονός επιτρέπει την υλοποίηση "λεπτών πελατών" (thin clients).

● Μείωση των χρονικών καθυστερήσεων λόγω δικτύου: Οι κινητοί πράκτορες, λόγω των κινητών και προσαρμοστικών ιδιοτήτων τους, προσφέρουν μεγάλες δυνατότητες για απόκριση σε πραγματικού χρόνου συστήματα, όπου η χρονική καθυστέρηση δεν μπορεί να γίνεται αποδεκτή και δημιουργεί προβλήματα.

● Αυξημένη ευρωστία: Η μείωση της εξάρτησης μιας κατανεμημένης εφαρμογής από την διαθεσιμότητα του δικτύου είναι ευπρόσδεκτη και λύνει ένα από τα βασικά προβλήματα που αντιμετωπίζουν οι αρχιτεκτονικές πελάτη-εξυπηρετητή. Στα συστήματα κινητών πρακτόρων επέρχεται αυτή η μείωση της εξάρτησης καθώς από την στιγμή που ο κώδικας του πελάτη βρίσκεται στον ίδιο κόμβο με αυτόν του εξυπηρετητή, η επικοινωνία είναι τοπική.

● Αυτοματοποίηση της επεξεργασίας κατανεμημένων υπολογισμών: Οι πράκτορες έχουν ενσωματωμένα δρομολόγια τα οποία καθορίζουν τι είδους δραστηριότητες πρέπει να εκτελέσουν και σε ποιους κόμβους με αποτέλεσμα να μην απαιτείται συνεχής αλληλεπίδραση με τον χρήστη.

● Αποκεντρωμένη και κατά τόπους επεξεργασία (έλεγχος και διαχείριση): Η κλωνοποίηση των πρακτόρων επιτρέπει την (αυτοματοποιημένη) κατανομή των υπολογιστικών συνιστωσών προηγούμενα συγκεντρωτικών προγραμμάτων.

● Ικανότητα εκτέλεσης σε ετερογενή περιβάλλοντα: οι κινητοί πράκτορες έχουν τη δυνατότητα να εκτελούνται σε περιβάλλοντα με διαφορετικές υποδομές.

● Ευελιξία: Η κατ'απαίτηση διανομή λογισμικού ή παροχή υπηρεσιών γίνεται εφικτή καθώς κινητοί πράκτορες μπορούν να "κατεβούν" στιγμιαία σε οποιοδήποτε κόμβο πελάτη ή εξυπηρετητή.
● Κλιμάκωση εφαρμογών: Λόγω της δυναμικής εκτέλεσης των προγραμμάτων πρακτόρων διευκολύνεται η ανάπτυξη περισσότερο κλιμακούμενων εφαρμογών.

Οι κινητοί πράκτορες, λόγω των πλεονεκτημάτων που προαναφέρθηκαν, βρίσκουν εφαρμογές σε διάφορους τομείς της τεχνολογίας. Κατ' αρχήν, οι κινητοί πράκτορες μπορούν να χρησιμοποιηθούν στη συλλογή δεδομένων πάνω σε ένα δίκτυο, π.χ. το διαδίκτυο. Για παράδειγμα, ένας κινητός πράκτορας μπορεί να σταλεί σε έναν εξυπηρετητή που υπάρχει μια μεγάλη βάση δεδομένων, να εκτελέσει τοπικά μια αναζήτηση και να επιστρέψει με το αποτέλεσμα στον υπολογιστή που ξεκίνησε η εκτέλεσή του. Επίσης, νέες προοπτικές ανοίγονται στον τομέα του ηλεκτρονικού εμπορίου. Για παράδειγμα, κάποιος αντιπρόσωπος μπορεί να αναλαμβάνει να ενημερώνει διαφορετικές ιστοσελίδες για τα νέα προϊόντα, χωρίς να είναι απαραίτητη η ύπαρξη κοινής βάσης δεδομένων ή άλλης υποδομής επικοινωνίας ανάμεσα στις ιστοσελίδες. Από την πλευρά του χρήστη, κάποιος αντιπρόσωπος μπορεί να αναλαμβάνει να αναζητήσει προϊόντα τα οποία ταιριάζουν με κάποια κριτήρια αναζήτησης, και να του παρουσιάσει τα αποτελέσματα, πραγματοποιώντας και κάποια σύγκριση μεταξύ τους. Οι πράκτορες που θα χρησιμοποιηθούν σε τέτοιου είδους εφαρμογές πρέπει να αξιόπιστοι, τόσο από άποψη ασφάλειας, όσο και ανοχής σε σφάλματα [14]. Επιπλέον, η ίδια η τεχνολογία των κινητών πρακτόρων οδηγεί σε ένα νέο είδος ηλεκτρονικού εμπορίου, το λεγόμενο κινητό εμπόριο (mobile commerce, m – commerce [15]).

Αποδοτική εφαρμογή των κινητών πρακτόρων μπορεί να γίνει και στην περίπτωση των κινητών χρηστών ενός δικτύου. Αυτό που χαρακτηρίζει αυτήν την κατηγορία χρηστών είναι η κοστοβόρα και ευαίσθητη σύνδεσή τους με το δίκτυο, καθώς και κατά κανόνα ο ελλιπής σε πόρους υπολογιστής που χρησιμοποιούν για τη σύνδεση. Για την υποστήριξη αυτών των κινητών χρηστών το παράδειγμα του κινητού πράκτορα μπορεί να χρησιμοποιηθεί, αφού ταιριάζει ιδανικά σε περιπτώσεις όπου δεν είναι δυνατή η συνεχόμενη και σταθερή σύνδεση. Προς αυτήν την κατεύθυνση έχουν γίνει προσπάθειες για την εκμετάλλευση του μοντέλου των κινητών πρακτόρων στην διαχείριση δικτύων κινητών συσκευών [16].

Η τεχνολογία των κινητών πρακτόρων μπορεί να χρησιμοποιηθεί με μεγάλη αποδοτικότητα για την επίλυση προβλημάτων απαιτητικών σε υπολογιστικούς πόρους, όπως για παράδειγμα προβλήματα Υπολογιστικού Ηλεκτρομαγνητισμού. Τέλος, σε κάποιες πρόσφατες ερευνητικές προσπάθειες [34, 35] έχει γίνει προσπάθεια για τη χρησιμοποίηση κινητών πρακτόρων στην ανάπτυξη πλεγματικών υποδομών.

4.5. Τελευταίες Εξελίξεις στην Τεχνολογία των Κινητών Πρακτόρων

Όπως προαναφέρθηκε, η τεχνολογία των κινητών πρακτόρων υπάρχει πολλά χρόνια, αλλά δεν έχει επιτευχθεί ακόμη η διάδοση και η ευρεία αποδοχή της από την κοινότητα της πληροφορικής. Αυτό συμβαίνει για το λόγο ότι υπάρχουν ακόμη βασικά θέματα και ζητήματα που λείπουν από τη συγκεκριμένη τεχνολογία για να μπορεί να θεωρηθεί πλήρως εκμεταλλεύσιμη. Ακόμα και σήμερα γίνονται μεγάλες ερευνητικές προσπάθειες για την ολοκλήρωση και διάδοση της τεχνολογίας των κινητών πρακτόρων. Στις παραγράφους που ακολουθούν θα παρουσιαστούν οι σημαντικότεροι τομείς στους οποίους γίνεται ερευνητική προσπάθεια ανά τον κόσμο σήμερα.

4.5.1 Υιοθέτηση Προτύπων

Όλοι οι κινητοί πράκτορες έχουν κάποια κοινά χαρακτηριστικά, τα οποία αναλύθηκαν σε προηγούμενη παράγραφο, όπως για παράδειγμα η αυτονομία, η ενεργητικότητα και η κοινωνικότητα. Όλα τα συστήματα πολλαπλών πρακτόρων μοιάζουν στο ότι πρέπει να παρέχουν στους πράκτορες την υποδομή που θα τους επιτρέπει να υλοποιούν τα παραπάνω χαρακτηριστικά. Πέρα όμως από τις όποιες ομοιότητες, μπορούν να υπάρξουν και πολλές διαφορές μεταξύ αυτών των συστημάτων. Αν δεν υπάρξει μια από άκρο σε άκρο διαλειτουργικότητα ανάμεσα στα διαφορετικά συστήματα, υπάρχει ο κίνδυνος η χρησιμότητα της τεχνολογίας αυτής να μείνει ανεκμετάλλευτη. Γι' αυτό το λόγο έχουν γίνει πολλές προσπάθειες για τη δημιουργία κάποιων προτύπων που θα επιτρέπουν τη διαλειτουργικότητα των συστημάτων κινητών πρακτόρων. Προς αυτήν την κατεύθυνση ιδρύθηκε και ο FIPA (Foundation for Intelligent Physical Agents), οργανισμός ο οποίος έχει ως σκοπό την δημιουργία standards για την διαλειτουργικότητα των συστημάτων κινητών πρακτόρων. Αυτά τα standards βρίσκονται διαθέσιμα στο [17]. Παλαιότερα είχαν προταθεί και άλλα πρότυπα από το OMG (Object Management Group) και τα οποία είναι γνωστά ως MASIF (Mobile Agent Systems Interoperability Facilities). Η σημερινή μορφή των προτύπων που προτείνει το OMG ονομάζεται Mobile Agent Facility Specification και βρίσκεται διαθέσιμη στο [18]. Για μια σύγκριση των δύο παραπάνω προτύπων παραπέμπουμε στο [19]. Σε αυτό το σημείο πρέπει να σημειωθεί ότι υπάρχει μία σύγχυση στην προγραμματιστική κοινότητα που ασχολείται με ανάπτυξη υποδομών πρακτόρων, η οποία οφείλεται στην μη κοινή αποδοχή κάποιων συγκεκριμένων προτύπων. Αυτή η μη αποδοχή θεωρείται από πολλούς ως ένας από τους βασικότερους λόγους που δεν έχει καθιερωθεί ακόμα η τεχνολογία των κινητών πρακτόρων και γι' αυτό το λόγο γίνονται σήμερα πολλές προσπάθειες για την καθιέρωση ενός κοινά αποδεκτού προτύπου.

4.5.2 Ασφάλεια

Η ασφάλεια αποτελεί ένα από τα πιο κρίσιμα ζητήματα όσον αφορά τα συστήματα κινητών πρακτόρων. Συχνά αναφέρεται [20] ως ο βασικός λόγος που τα συστήματα κινητών πρακτόρων δεν έχουν τύχει ακόμη ευρείας διάδοσης. Τα τελευταία χρόνια μεγάλο μέρος της έρευνας έχει αφιερωθεί στο συγκεκριμένο αντικείμενο. Ήδη από τα τέλη της προηγούμενης δεκαετίας έχουν αρχίσει οι προσπάθειες αντιμετώπισης προβλημάτων σχετικών με την ασφάλεια [21]. Ένα πρόβλημα που προέκυψε από την αρχή ήταν αυτό της επαλήθευσης της αυθεντικότητας. Αρχικά υπήρξε η ιδέα η αυθεντικοποίηση να γίνεται με βάση ψηφιακές υπογραφές που χρησιμοποιούν κάποιο δημόσιο κλειδί κρυπτογραφίας. Όμως ένας κινητός πράκτορας αλλάζει από τη στιγμή της έναρξης της εκτέλεσής του, πράγμα που κάνει δύσκολη την αυθεντικοποίηση των συνεχώς μεταβαλλόμενων στοιχείων του. Αυτό σημαίνει ότι θα πρέπει να υπάρχει η δυνατότητα για δυναμική δημιουργία κώδικα. Μία άλλη λύση είναι η συσσωρευτική κρυπτογραφία του πράκτορα από τον κάθε κόμβο από τον οποίο αυτός περνάει [22].

Γενικά ζητήματα ασφάλειας μπορούν να προκύψουν είτε όταν ένα κακόβουλο περιβάλλον εκτέλεσης απειλεί έναν πράκτορα, είτε όταν ένας κακόβουλος πράκτορας απειλεί ένα περιβάλλον εκτέλεσης ή έναν άλλο πράκτορα. Η πρώτη περίπτωση συμβαίνει όταν ένας υπολογιστής εξετάζει τον κώδικα ενός πράκτορα, προσπαθεί να μάθει τα δεδομένα που μεταφέρει ο πράκτορας και εκμεταλλεύεται αυτή τη γνώση στην διαπραγμάτευσή του με τον πράκτορα για να αποκτήσει το οποιοδήποτε

πλεονέκτημα ή όταν προσπαθεί να αλλοιώσει το αποτέλεσμα ενός υπολογισμού. Ήδη έχει προταθεί ένα μοντέλο για την προστασία του κινητού κώδικα από αυτού του είδους τα προβλήματα [23]. Η δεύτερη περίπτωση αποτελεί κλασσική περίπτωση προφύλαξης των υπολογιστών από κακόβουλο λογισμικό. Μια κοινή λύση στο πρόβλημα αποτελεί η υπογραφή του κώδικα για την εξακρίβωση του βαθμού εμπιστοσύνης προς την πηγή από την οποία προέρχεται ο κώδικας.

Επιπλέον, διάφορα άλλα μοντέλα για την ασφάλεια των συστημάτων κινητών πρακτόρων έχουν προταθεί σε πρόσφατες ερευνητικές προσπάθειες. Σε ένα από αυτά [24] τα μοντέλα θεωρείται ότι απειλές για την ασφάλεια μπορούν να γίνουν μόνο σε μία συσχέτιση που υπάρχει μεταξύ δύο οντοτήτων ενός συστήματος. Συσχέτιση μεταξύ δύο οντοτήτων υπάρχει όταν γίνεται οποιαδήποτε ανταλλαγή πληροφοριών μεταξύ τους. Αλλά επίθεση μπορεί να γίνει και σε μια οντότητα δημιουργώντας μια συσχέτιση με αυτήν. Για να γίνει λοιπόν αναγνώριση μιας επίθεσης πρέπει να αναγνωρίζονται όλες οι συσχετίσεις καθώς και οι οντότητες που παίρνουν μέρος σε αυτές. Γνωρίζοντας αυτές τις οντότητες μπορούν να κατηγοριοποιηθούν οι συσχετίσεις ως συγκεκριμένες μορφές αλληλεπιδράσεων. Επιπλέον, τα λειτουργικά συστατικά αυτών των οντοτήτων πρέπει να αναγνωρίζονται και να προστατεύονται κατάλληλα, αφού σε αυτά τα συστατικά μπορούν να εκδηλωθούν οι επιθέσεις. Αυτή η προστασία στα σύγχρονα συστήματα περιλαμβάνει χρήση κρυπτογραφίας με ασφαλή και αξιόπιστη διαχείριση και διανομή κλειδιού καθώς και εξουσιοδότηση δικαιωμάτων. Εδώ βέβαια υπάρχει ιδιαιτερότητα τόσο στην εξουσιοδότηση (delegation) όσο και στην επίτρεψη πρόσβασης (non-repudiation), καθώς οι κινητοί πράκτορες είναι δυνατόν να μετακινηθούν σε πολλά διαφορετικά μέρη κατά τον κύκλο ζωής τους, οπότε τα παραπάνω χαρακτηριστικά ασφάλειας μπορούν να πειραχτούν. Γι' αυτό το λόγο πρέπει να γίνεται χρήση κάποιας logging service που ελέγχει την ακεραιότητα αυτών των χαρακτηριστικών. Ακόμη, μια λύση για την επίτρεψη πρόσβασης μπορεί να είναι και η παροχή υπηρεσιών από τρίτους, που θα καταγράφουν τις όποιες λειτουργίες μπορεί να υπαγορεύουν την απαγόρευση πρόσβασης.

4.5.3 Ανοχή σε σφάλματα

Ένα από τα πρωταρχικά στοιχεία που απαιτείται για την περαιτέρω λειτουργικότητα και ανάπτυξη των συστημάτων κινητών πρακτόρων είναι η ανοχή σε σφάλματα. Αυτό σημαίνει ότι πρέπει να υπάρχει ένας τρόπος να προστατεύεται ένας πράκτορας από απώλειες, δηλαδή να εξασφαλίζεται ότι φτάνει στον προορισμό του. Συνήθως η προστασία από απώλειες γίνεται με δημιουργία πολλαπλών αντιγράφων του πράκτορα και κατάλληλη διαχείριση των αντιγράφων. Η κλασική προσέγγιση σε αυτό το πρόβλημα είναι η ενσωμάτωση του μηχανισμού αποτροπής απωλειών στο σύστημα των πρακτόρων (place-dependent αρχιτεκτονική). Υπάρχει όμως και η δυνατότητα το πρωτόκολλο που εξασφαλίζει την ανοχή σε σφάλματα να μετακινείται μαζί με τον πράκτορα (agent-dependent αρχιτεκτονική). Αυτό έχει το πλεονέκτημα ότι δεν χρειάζεται να τροποποιηθούν ήδη υπάρχουσες πλατφόρμες για την υποστήριξη ανοχής σφαλμάτων. Επιπλέον, σε περίπτωση που υιοθετηθεί η place-dependent αρχιτεκτονική, όλες οι πλατφόρμες που πιθανόν θα πρέπει να συνεργαστούν πρέπει να χρησιμοποιούν τον ίδιο μηχανισμό, πράγμα που προφανώς δημιουργεί προβλήματα συμβατότητας. Βέβαια η ενσωμάτωση του μηχανισμού στην πλατφόρμα έχει και κάποια πλεονεκτήματα, το σημαντικότερο των οποίων είναι μικρότερο μέγεθος πρακτόρων και επομένως βελτιωμένη απόδοση κατά τη μετάδοσή

τους.

Στην περίπτωση της agent-dependent προσέγγισης [25], ο μηχανισμός αποφυγής σφαλμάτων ταξιδεύει μαζί με τον πράκτορα. Μία μόνο υπόσταση του μηχανισμού υπάρχει σε κάθε πράκτορα, η οποία αρχικοποιείται στην πηγή του πράκτορα και τερματίζει στον προορισμό του. Ο συνδυασμός της υπόστασης αυτού του μηχανισμού με τον ίδιο τον πράκτορα δημιουργεί έναν κινητό πράκτορα ανεκτικό σε σφάλματα. Η πλατφόρμα βέβαια συνεχίζει να βλέπει τον πράκτορα σαν συνηθισμένο πράκτορα. Αυτό σημαίνει ότι η πλατφόρμα δεν χρειάζεται να αλλάξει, χρειάζεται όμως βέβαια να αλλάξει ο τρόπος που οι πράκτορες δημιουργούνται και κινούνται. Η agent-dependent αυτή προσέγγιση μπορεί να υλοποιηθεί αν δεν καθορίζεται μόνο ένα μέρος για μετακίνηση του πράκτορα, αλλά ένα σύνολο από μέρη στα οποία στέλνονται αντίγραφα του πράκτορα. Επιπλέον, αυτή η μετακίνηση σε πολλαπλούς προορισμούς μπορεί να γίνεται είτε αφού έχει διαπιστωθεί ότι έχει υπάρξει κάποιο σφάλμα, είτε από το ξεκίνημα της μετακίνησης.

Ένας επιπλέον διαχωρισμός που μπορεί να γίνει στις προσεγγίσεις για δημιουργία ανεκτικών σε σφάλματα πρακτόρων είναι σε αυτές που βασίζονται σε αντιγραφή πρακτόρων στο χώρο (spatial replication based – SRB) [26] και σε αυτές που βασίζονται σε αντιγραφή στο χρόνο (temporal replication based – TRB) [27]. Κατά την SRB προσέγγιση, ένας πράκτορας στέλνεται ταυτόχρονα σε ένα σύνολο προορισμών για τους υπολογισμούς της επόμενης φάσης. Για να εξασφαλιστεί όμως ότι ο πράκτορας θα εκτελεστεί μόνο μια φορά χρειάζονται πολύπλοκα πρωτόκολλα. Στην TRB προσέγγιση γίνεται αρχικά προσπάθεια να εκτελεστεί ο πράκτορας σε μια συγκεκριμένη τοποθεσία. Αν η εκτέλεση αποτύχει, ο πράκτορας στέλνεται σε άλλη τοποθεσία. Εδώ όμως υπάρχει το πρόβλημα ότι σε περίπτωση εσφαλμένης ανίχνευσης σφάλματος μπορεί να οδηγήσει στην ύπαρξη δύο ομοίων πρακτόρων, πράγμα που μπορεί να έχει ως αποτέλεσμα ακόμη και το διάβασμα λανθασμένων δεδομένων από άλλους πράκτορες.

4.5.4 Απλοποίηση και Διάδοση των Συστημάτων Κινητών Πρακτόρων

Υπάρχει σήμερα η άποψη ότι ένας από τους σημαντικότερους παράγοντες που εμποδίζουν την εξάπλωση της χρήσης της τεχνολογίας των κινητών πρακτόρων είναι η έλλειψη επαρκούς υποστήριξης για την ανάπτυξη εφαρμογών. Γι΄αυτό το λόγο γίνεται μια μεταφορά της έρευνας από το πεδίο της κινητικότητας και γενικά των θεμάτων που αφορούν τη λειτουργία των κινητών πρακτόρων σε θέματα που σχετίζονται με την απλοποίηση και την υποστήριξη χρήσης των κινητών πρακτόρων για υλοποίηση πραγματικών εφαρμογών.

Κάποιες προσπάθειες έχουν επικεντρωθεί στο να γίνουν οι εφαρμογές που θα χρησιμοποιούν τους πράκτορες το επίκεντρο της σχεδίασης και όχι οι ίδιοι οι πράκτορες. Με αυτήν την προσέγγιση η εφαρμογή και οι πράκτορες είναι δυνατόν να αλληλεπιδρούν απευθείας. Παρ' όλα αυτά, η ύπαρξη μιας πλατφόρμας είναι και πάλι απαραίτητη, έτσι ώστε να παρέχεται και ασφάλεια αλλά και η δυνατότητα στους πράκτορες να ψάχνουν και να αλληλεπιδρούν με άλλες υπηρεσίες. Η πλατφόρμα αυτή πρέπει να έχει κάποια χαρακτηριστικά, όπως

- ευλυγισία, δηλαδή κάθε εφαρμογή πρέπει να μπορεί να χρησιμοποιεί μόνο τα μέρη που χρειάζεται από την πλατφόρμα
- ευκολία στην ανάπτυξη εφαρμογών
- προστασία από σφάλματα
- αδιαφανής ενσωμάτωση στις εφαρμογές

Επιπλέον, εντείνονται οι έρευνες για δημιουργία μοντέλων και γλωσσών προδιαγραφών για κινητούς πράκτορες έτσι ώστε να αντιμετωπιστούν προβλήματα σχετικά με την ασφάλεια και την πολύπλοκη συμπεριφορά των πρακτόρων (όπως για παράδειγμα τη μετανάστευση) [28]. Αυτά τα μοντέλα και οι γλώσσες απαιτούν να έχουν κάποια χαρακτηριστικά, όπως παραμετροποίηση (customizability) και επαρκή περιγραφική δύναμη. Μια λύση για τα παραπάνω προβλήματα που διαθέτει αυτά τα χαρακτηριστικά είναι η FDT (Formal Description Technique).

4.5.5 Νέες Τεχνολογίες και Συστήματα Κινητών Πρακτόρων

Παρά την συνεχή αύξηση που παρατηρείται στην ταχύτητα των CPU, οι σύγχρονοι υπολογιστές δεν μπορούν να καλύψουν τις τεράστιες ανάγκες που υπάρχουν σε τομείς όπως η βιοπληροφορική. Τα υπολογιστικά μοντέλα που υπάρχουν σήμερα σε όλους τους τομείς της επιστήμης παρουσιάζουν μια εκθετική χρονική πολυπλοκότητα, ενώ και τα δεδομένα προς επεξεργασία αυξάνονται ραγδαία. Γι' αυτούς τους λόγους η ζήτηση για υπολογιστική ισχύ υπερσκελίζει την όποια αύξησή της. Ενώ όμως συμβαίνει αυτό, τεράστια ποσά υπολογιστικής ισχύος υπάρχουν αχρησιμοποίητα σε υπολογιστές σε όλο τον κόσμο. Ένας τρόπος για να γίνει εκμετάλλευση αυτής της αχρησιμοποίητης υπολογιστικής ισχύος είναι η σύνδεση των υπολογιστών αυτών σε ένα κατανεμημένο δίκτυο Peer-to-Peer .

Στο μοντέλο P2P δεν υπάρχει διάκριση ρόλων και κάθε κόμβος μπορεί είτε να ξεκινήσει επικοινωνία, είτε να δεχτεί αιτήσεις και να διαθέσει υπηρεσίες. Η λογική των εφαρμογών πλέον δεν είναι επικεντρωμένη πάνω σε κάποιον server, αλλά είναι κατανεμημένη σε όλους τους κόμβους του δικτύου. Κάθε κόμβος μπορεί να ψάχνει για άλλους και να μπαίνει στο δίκτυο ή να βγαίνει από αυτό κάθε στιγμή. Πρόκειται δηλαδή για ένα πλήρως κατανεμημένο σύστημα που προσφέρεται για υλοποίηση κατανεμημένων υπηρεσιών.

Το ιδανικό εργαλείο για την υλοποίηση αυτών των υπηρεσιών είναι οι κινητοί πράκτορες, αφού μπορούν να αντιδρούν με βάση τη διαθεσιμότητα πηγών ενώ μπορούν να εκτελούν διάφορες εργασίες χωρίς να χρειάζεται να υπάρχει εγκατεστημένο κάθε φορά κάποιο συγκεκριμένο για την εργασία αυτή λογισμικό. Τρία συστήματα που κάνουν χρήση κινητών πρακτόρων πάνω σε δίκτυα P2P είναι το POPCORN [29], το οποίο προορίζεται για εμπορικές συναλλαγές, το YACA [30], που προορίζεται για κατανεμημένη επίλυση υπολογιστικά απαιτητικών προβλημάτων και το PaCMan [31], το οποίο χρησιμοποιεί μια ιδιαίτερη προσέγγιση: τα δεδομένα που παίρνονται από κάποιον υπολογιστή σπάνε σε κομμάτια και μοιράζονται στους υπόλοιπους υπολογιστές που αποτελούν ένα cluster του συστήματος. Στη συνέχεια γίνεται χρήση πρακτόρων που μεταναστεύουν στους υπολογιστές που έχουν τα δεδομένα και εκεί τα επεξεργάζονται.

Το σημαντικότερο ίσως σύστημα κινητών πρακτόρων που χρησιμοποιεί την peer-to-peer αρχιτεκτονική επικοινωνίας είναι το JADE της TILAB [32]. Το περιβάλλον του JADE μπορεί να εξελίσσεται δυναμικά με κόμβους, οι οποίοι στη συγκεκριμένη πλατφόρμα ονομάζονται πράκτορες και οι οποίοι μπορούν να εμφανίζονται στο σύστημα με βάση τις ανάγκες και τις απαιτήσεις της κάθε εφαρμογής. Η επικοινωνία μεταξύ των κόμβων είναι απόλυτα συμμετρική με τον κάθε κόμβο να μπορεί είτε να δέχεται είτε να στέλνει αιτήσεις επικοινωνίας. Το JADE είναι βασισμένο στη Java και δεν απαιτεί κάποια συγκεκριμένη έκδοση για να εκτελεστεί, ενώ επιπλέον συγκεντρώνει τα περισσότερα από τα χαρακτηριστικά που προαναφέρθηκαν: διαλειτουργικότητα, ομοιομορφία, ευκολία στη χρήση καθώς και μια φιλοσοφία pay-

as-you-go που επιτρέπει στον προγραμματιστή να χρησιμοποιεί μόνο τα χαρακτηριστικά που χρειάζεται.

Επιπλέον, με τις πρόσφατες εξελίξεις στα ασύρματα δίκτυα (Wavelan, Bluetooth) και λόγω της ευρείας διάδοσης κινητών υπολογιστών διαφόρων ειδών (όπως φορητοί υπολογιστές, κινητά τηλέφωνα, PDAs), ένας συνεχώς αυξανόμενος αριθμός εφαρμογών γράφεται με σκοπό την εφαρμογή σε κινητούς υπολογιστές. Το μεγαλύτερο πρόβλημα που έχουν να αντιμετωπίσουν αυτές οι εφαρμογές είναι το εξαιρετικά ανομοιογενές και συνεχώς μεταβαλλόμενο περιβάλλον ως προς το λογισμικό, το υλικό και τα επίπεδα δικτύου, αφού οι συσκευές που εμπλέκονται περιέχουν εφαρμογές με μεγάλες διαφορές στο υλικό και τα συστήματα μεσισμικού και μπορεί να χρησιμοποιούν διαφορετικές δικτυακές υποδομές. Για την ανάπτυξη αυτών των εφαρμογών χρησιμοποιούνται νέα σενάρια που επιτρέπουν την προσαρμογή στις όποιες διαφορές και αλλαγές. Αυτά τα νέα σενάρια έχουν ως αποτέλεσμα τη μεταφορά της επιβάρυνσης κατά την ανάπτυξη συστημάτων κινητών πρακτόρων από τη διαδικασία της σχεδίασης στη διαδικασία εκτέλεσης [33].

4.6. Αναφορές

[1] J.M. Bradshaw, Software Agents, MIT Press, ISBN 0-262-52234-9, 1997

[2] Henry G. Baker and Carl Hewitt. Laws for communicating parallel processes. In IFIP Congress, pages 987--992. IFIP, August 1977.

[3] Hyacinth S. Nwana, Software Agents: An Overview, Knowledge Engineering Review, Vol. 11, No 3, pp.1-40, Sept 1996, Cambridge University Press

[4] IBM Aglets, Aglets home page: http://www.trl.ibm.com/aglets

[5] Grasshopper IKV++ Gmbh, Grasshopper home page: http://www.grasshopper.de

[6] Tilab Jade, Jade home page http://jade.tilab.com

[7] P. Morreale, Agents on the move, Spectrum, IEEE, Volume: 35 Issue: 4, April 1998, Page(s): 34 –41

[8] OMG, 2000, Agent technology, green paper, Technical Report ec#2000-03-01, Object Management Group, March 2000. URL: http://www.jamesodell.com/ec2000-08-01.pdf

[9] K. Rothermel, F. Hohl and N. Radouniklis, Mobile Agent Systems: What is missing? In: Invited Paper to the {IFIP} {WG} 6.1 International Working Conference on Distributed Applications and Interoperable Systems ({DAIS}'97)

[10] A. Fuggetta, G.P. Picco and G.Vigna ,Understanding Code Mobility, IEEE transactions on Software Engineering, 24, 5, 342 – 362, May 1998

[11] Ghezzi, C. and Vigna, G. (1997) Mobile Code Paradigms and Technologies: A Case Study, in Proceedings of the First International Workshop on Mobile Agents, MA'97 (eds. K. Rothermel and R. Popescu-Zeletin), Springer.

[12] Knabe, F. (1995) Language Support for Mobile Agents, PhD dissertation, School of Computer Science, Carnegie Mellon University.

[13] Matthiske, B., Matthes, F. and Schmidt, J. (1995) On migrating threads, in Proceedings of the Second International Workshop on Next Generation Information Technologies and Systems (Naharia, Israel, June 1995).

[14] Markus Straßer, Kurt Rothermel, Christian Maihöfer, Providing Reliable Agents for Electronic Commerce, URL: http://citeseer.ist.psu.edu/48504.html

[15] Brito, L., Neves, J., Moura, F., A Mobile-Agent Based Architecture for Virtual Enterprises, in Proceedings of the 2nd IFIP/MASSYVE Working Conference on Infrastructures for Virtual Enterprises (PRO-VE2000), Florianpolis, Brasil, 2000

[16] Akhil Sahai and Christine Morin, Mobile Agents for Enabling Mobile User Aware Applications, Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98), ACM Press, URL: http://citeseer.ist.psu.edu/article/sahai98mobile.html

[17] FIPA Specifications, URL: http://www.fipa.org/specs/

[18] OMG Mobile Agent Facility Specification, URL: http://www.omg.org/technology/documents/formal/mobile_agent_facility.htm

[19] Patricia Cuesta Rivalta, Mobile Agent Management, URL: http://citeseer.ist.psu.edu/rivalta00mobile.html

[20] Johansen, D.: Interview in: Milojicic, D.: Trend Wars: Mobile Agent Applications, IEEE Concurrency, pp 80-90, July – September, 1999

[21] Holger Peine and Torsten Stolpmann, The Architecture of the {Ara} Platform for Mobile Agents, First International Workshop on Mobile Agents {MA} 1997, URL: http://citeseer.ist.psu.edu/peine97architecture.html

[22] Young A., Yung M., Sliding Encryption: A Cryptographic Tool for Mobile Agents. (ed) Eli Biham, Proceedings of the 4th International Workshop on Fast Software Encryption, FSE'97, January 1997, LNCS 1419, Springer – Verlag, 1997

[23] Joy Algesheimer, Christian Cachin, Jan Camenisch and Gunter Karjoth, Cryptographic security for mobile code, URL: http://citeseer.ist.psu.edu/algesheimer03cryptographic.html

[24] Reiser, H. and Vogt, G., Security Requirements for Management Systems using Mobile Agents, Proceedings of the Fifth IEEE Symposium on Computers \& Communications, pages 160-165, URL: http://citeseer.ist.psu.edu/reiser00security.html

[25] Stefan Pleisch and Andre Schiper, FATOMAS - A Fault-Tolerant Mobile Agent System Based on the Agent-Dependent Approach, Proceedings of the IEEE Int. Conf. on Dependable Systems and Networks 2001, URL: http://citeseer.ist.psu.edu/article/pleisch01fatomas.html

[26] D. Johansen, K. Marzullo, F. B. Schneider, K. Jacobsen, and D. Zagorodnov, NAP: Practical fault-tolerance for itinerant computations. In Proc. of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS), Austin, Texas, June 1999

[27] L. Silva, V. Batista, and J. Silva. Fault-tolerant execution of mobile agents. In Proc. of the International Conference on Dependable Systems and Networks, pages 135–143, New York, June 2000.

[28] Yasuyuki Tahara, Pigeon : a Specification Language for Mobile Agent Applications, URL: http://citeseer.ist.psu.edu/712402.html

[29] Nisan, N., London, S., Regev, O., Camiel, N.: Globally distributed computation over the Internet - the POPCORN project. In: Sixth International World Wide Web Conference, Santa Clara, California USA (1997)

[30] Stefan Kleijkers, Floris Wiesman and Nico Roos, A Mobile Multi-Agent System for Distributed Computing, URL: http://citeseer.ist.psu.edu/kleijkers02mobile.html

[31] Evripidou, P., Samaras, G., Panayiotou, C., Pitoura, E.: The PaCMAn metacomputer: Parallel computing with Java mobile agents. In: Proceedings of 25th Euromicro Conference, In Press, Milan, Italy (1999)

[32] Tilab Jade White Paper, URL: http://sharon.cselt.it/projects/jade/papers/WhitePaperJADEEXP.pdf

[33] Stefanos Zachariadis and Cecilia Mascolo, Adaptable Mobile Applications through SATIN: Exploiting Logical Mobility, Mobile Computing Middleware, URL: http://citeseer.ist.psu.edu/686653.html

[34] Munehiro Fukuda, Yuichiro Tanaka, Naoya Suzuki, Lubomir F. Bic and Shinya Kobayashi, A Mobile-Agent-Based PC Grid, Autonomic Computing Workshop Fifth Annual International Workshop on Active Middleware Services (AMS'03) p. 142, URL: http://doi.ieeecomputersociety.org/10.1109/ACW.2003.1210214

[35] O. Rana and L. Moreau, Issues in Building Agent based Computational Grids, In Third Workshop of the UK Special Interest Group on Multi-Agent Systems (UKMAS), URL: http://citeseer.ist.psu.edu/523418.html

5. Υλοποίηση Πλατφόρμας Κινητών Αντιπροσώπων

Στο κεφάλαιο αυτό γίνεται η περιγραφή μιας ευέλικτης αρχιτεκτονικής που επιτρέπει την υιοθέτηση επιπλέον καναλιών επικοινωνίας σε μια πλατφόρμα κινητών αντιπροσώπων. Η πλατφόρμα αυτή σχεδιάστηκε και υλοποιήθηκε με τη βοήθεια της Java στα πλαίσια προηγούμενης διπλωματικής εργασίας [1] και βασίζεται στην τεχνολογία των υπηρεσιών ιστού (Web Services). Στην παρούσα εργασία έγινε αντικατάσταση των μηχανισμών επικοινωνίας της πλατφόρμας με πιο επιτηδευμένα συστατικά, κρατώντας παράλληλα τα βασικά στοιχεία της αρχικής πλατφόρμας ουσιαστικά αναλλοίωτα. Η αρχιτεκτονική που υλοποιήθηκε χρησιμοποιεί κάποια γνωστά σχεδιαστικά πρότυπα για την περιγραφή των στοιχείων που την αποτελούν. Σαν πρώτη εναλλακτική μορφή επικοινωνίας χρησιμοποιήθηκαν οι δικτυακοί υποδοχείς (sockets). Επίσης, σχεδιάστηκε μια γραφική διεπαφή χρήστη για τη διαχείριση της πλατφόρμας. Στις επόμενες παραγράφους γίνεται περιγραφή των βασικών χαρακτηριστικών της αρχικής πλατφόρμας και στη συνέχεια αναλυτική παρουσίαση της νέας αρχιτεκτονικής που επιτρέπει τη χρήση πολλαπλών πρωτοκόλλων επικοινωνίας. Επιπλέον, γίνεται μια σύγκριση μεταξύ των δύο τεχνολογιών επικοινωνίας με βάση κάποιες μετρήσεις. Το γραφικό περιβάλλον διαχείρισης της πλατφόρμας περιγράφεται στο Παράρτημα Α.

5.1. Εισαγωγή

Τα τελευταία χρόνια, οι ερευνητικές προσπάθειες πάνω στην τεχνολογία των υποδομών κινητών αντιπροσώπων έχουν επικεντρωθεί κατά βάση στη μεταφορά ήδη υπαρχόντων πλατφορμών σε νέα περιβάλλοντα ή στην ανάπτυξη νέων πλατφορμών για αυτά τα περιβάλλοντα, όπως για παράδειγμα κινητές συσκευές [3], εξυπηρετητές ιστού ή ακόμα και το πλέγμα. Αυτές οι πλατφόρμες χρησιμοποιούν συνηθισμένα κανάλια επικοινωνίας [4, 5], τα οποία συνήθως υπαγορεύονται από τους περιορισμούς και τις ιδιαιτερότητες του περιβάλλοντος εκτέλεσης (όπως για παράδειγμα το HTTP για τις κινητές συσκευές [6]). Αν και διάφορες αρχιτεκτονικές και γλώσσες επικοινωνίας πρακτόρων [7] έχουν προταθεί για την υποστήριξη της αλληλεπίδρασης με άλλες πλατφόρμες πρακτόρων, οι περισσότερες πλατφόρμες σπάνια χρησιμοποιούν περισσότερα από ένα κανάλι επικοινωνίας. Αυτό περιορίζει σε ένα βαθμό τις τεχνολογίες κινητών αντιπροσώπων, λόγω της έλλειψης διαλειτουργικότητας μεταξύ πλατφορμών που τρέχουν σε διαφορετικά περιβάλλοντα.

Η αρχιτεκτονική που αναπτύχθηκε στην παρούσα εργασία επιτρέπει την χρησιμοποίηση πολλαπλών πρωτοκόλλων επικοινωνίας σε μια πλατφόρμα, δίνοντας μια λύση στο πρόβλημα που περιγράφτηκε. Οι μηχανισμοί που επιτρέπουν την ύπαρξη και τη διαλειτουργικότητα πολλαπλών πρωτοκόλλων είναι τελείως διαφανείς στο βασικό σκελετό της πλατφόρμας, πράγμα που ελαχιστοποιεί το κόστος της μεταφοράς και της προσαρμογής υπαρχόντων εφαρμογών κινητών πρακτόρων στην νέα αρχιτεκτονική.

5.2. Περιγραφή της Βασικής Πλατφόρμας

Όπως προαναφέρθηκε, η αρχιτεκτονική που υλοποιήθηκε στην παρούσα διπλωματική εργασία αποτελεί επέκταση μιας ήδη υπάρχουσας πλατφόρμας κινητών

αντιπροσώπων, η οποία υλοποιήθηκε με τη βοήθεια της Java και η οποία κάνει χρήση των Υπηρεσιών Ιστού. Οι υπηρεσίες ιστού παρέχουν διάφορες συλλογές υπηρεσιών που περιέχονται μέσα σε αυτόνομες εφαρμογές και οι οποίες είναι προσβάσιμες μέσα σε ένα δίκτυο, ή ακόμη και το διαδίκτυο, μέσω τυποποιημένων XML μηνυμάτων. Η επικοινωνία μεταξύ των συστατικών της πλατφόρμας γίνεται χρησιμοποιώντας το πρωτόκολλο SOAP (Simpe Object Access Protocol) και συγκεκριμένα το SOAP – RPC. Με χρήση αυτού του πρωτοκόλλου, ένα αντικείμενο μπορεί να καλέσει μια μέθοδο ενός αντικειμένου που βρίσκεται σε απομακρυσμένο υπολογιστή και να πάρει το παραγόμενο αποτέλεσμα όταν η μέθοδος επιστρέψει.

5.2.1 Συστατικά Πλατφόρμας

Η πλατφόρμα έχει σχεδιαστεί σύμφωνα με τις προδιαγραφές OMG MASIF [2]. Για την υλοποίηση υιοθετήθηκαν οι έννοιες της περιοχής (Region), του πρακτορείου (Agency) και της τοποθεσίας (Place), των οποίων ο ρόλος περιγράφεται στη συνέχεια.

Η περιοχή χρησιμοποιείται για την παροχή υπηρεσιών καταλόγου. Κατά τη δημιουργία, μετανάστευση, τερματισμό λειτουργίας και άλλα σημαντικά γεγονότα στον κύκλο ζωής ενός πράκτορα, η περιοχή ενημερώνεται από τα πρακτορεία που είναι καταχωρημένα σε αυτήν, ώστε να μπορεί να παρακολουθεί όλες τις αλλαγές που γίνονται. Επιπλέον, ένα πρακτορείο ή ένας πράκτορας μπορεί να επικοινωνεί με την περιοχή στέλνοντας αναζητήσεις, με σκοπό την απόκτηση πληροφοριών για άλλους πράκτορες ή πρακτορεία.

Το βασικό συστατικό της πλατφόρμας είναι το πρακτορείο. Τα πρακτορεία παρέχουν στους πράκτορες ένα περιβάλλον εκτέλεσης και ένα μέσο επικοινωνίας με άλλους πράκτορες και άλλα πρακτορεία. Τα αντικείμενα – πρακτορεία παρέχουν το πραγματικό περιβάλλον εκτέλεσης, τόσο για στατικούς, όσο και για κινητούς πράκτορες. Οι βασικές λειτουργίες ενός πρακτορείου είναι η δημιουργία, η αποθήκευση και η μετακίνηση των πρακτόρων, καθώς και η παροχή μηχανισμών για την επικοινωνία των πρακτόρων με άλλα συστατικά της πλατφόρμας.

Κάθε πρακτορείο περιέχει μία ή περισσότερες τοποθεσίες. Η έννοια της τοποθεσίας χρησιμοποιείται για την ομαδοποίηση των πρακτόρων. Μια τοποθεσία ομαδοποιεί τη λειτουργικότητα μέσα σε ένα πρακτορείο, εμπεριέχοντας συγκεκριμένες δυνατότητες και συγκεκριμένους περιορισμούς, βοηθώντας με αυτόν τον τρόπο τον διαχειριστή της πλατφόρμας αλλά και τον προγραμματιστή να διαχωρίζει διαφορετικούς πράκτορες.

Ένα μεγάλο και σημαντικό μέρος της πλατφόρμας είναι οι εργάτες (workers), οι οποίοι είναι υπεύθυνοι για την εκτέλεση συγκεκριμένων εργασιών, όπως την δημιουργία κάποιας αίτησης για τη δημιουργία νέας τοποθεσίας σε ένα πρακτορείο ή την καταχώρηση ενός πρακτορείου σε μια περιοχή. Υπάρχουν δύο κατηγορίες εργατών: αυτοί που χρησιμοποιούνται για επικοινωνία μεταξύ πρακτορείων και αυτοί που χρησιμοποιούνται για επικοινωνία μεταξύ πρακτορείου και περιοχής και αντίστροφα. Οι εργάτες αποτελούν ένα ενδιάμεσο στρώμα μεταξύ της πλατφόρμας και του καναλιού επικοινωνίας, καθιστώντας δυνατή την επέκταση του μηχανισμού επικοινωνίας της πλατφόρμας με περισσότερα επικοινωνιακά πρωτόκολλα.

5.2.2 Κινητικότητα

Οι αντιπρόσωποι σε αυτή την πλατφόρμα διαχωρίζονται σε δύο κύριες κατηγορίες: τους κινητούς και τους στατικούς. Ενώ οι τελευταίοι παραμένουν στην

αντιπροσωπεία που τους δημιούργησε μέχρι το τέλος της εκτέλεσής τους, οι πρώτοι έχουν την ικανότητα να μεταναστεύουν σε απομακρυσμένες αντιπροσωπείες αυτόνομα ή να τους δοθεί εντολή γι' αυτό. Αυτή η μεταφορά επιτυγχάνεται και από τις δύο αντιπροσωπείες με διαπραγμάτευση και εάν είναι δυνατή η μεταφορά τότε την πραγματοποιούν. Κάθε αντιπρόσωπος (όπως και κάθε πρόγραμμα) αποτελείται από δύο μέρη: την πραγματική εντολή στην μηχανή και το τρέχον αποτύπωμα στη μνήμη. Και τα δύο αυτά μέρη μεταφέρονται στην απομακρυσμένη αντιπροσωπεία ώστε να καταστεί δυνατή η επαναδημιουργία του αντιπροσώπου.

5.2.3 Μηχανισμοί Επικοινωνίας

Η επικοινωνία μεταξύ των αντιπροσώπων ακολουθεί τρία διαφορετικά μοντέλα. Το πρώτο αφορά απλές σύγχρονες κλήσεις. Η κλήση σε ένα απομακρυσμένο αντιπρόσωπο επιστρέφει όταν το αποτέλεσμα είναι έτοιμο. Το κύριο πλεονέκτημα αυτού του μηχανισμού είναι ότι μετά την κλήση το αποτέλεσμα είναι πάντα έτοιμο αλλά ο αντιπρόσωπος μπλοκάρεται μέχρι να γίνει αυτό. Η δεύτερη μέθοδος είναι η ασύγχρονη. Στην περίπτωση αυτή ο αντιπρόσωπος δημιουργεί μια κλήση με τρόπο όμοιο με τη σύγχρονη κλήση, με τη διαφορά ότι η εκτέλεση του αντιπροσώπου μπλοκάρεται σε αυτή την περίπτωση. Έπειτα ο αντιπρόσωπος μπορεί να τσεκάρει περιοδικά αν το αποτέλεσμα της κλήσης είναι έτοιμο. Τέλος, η πλατφόρμα παρέχει τον τύπο κλήσης call-back-call όπου ο αντιπρόσωπος όχι μόνο μορφοποιεί την κλήση αλλά δηλώνει και τη μέθοδο που θα καλεστεί όταν ανακτήσει το αποτέλεσμα. Η εκτέλεση του αντιπροσώπου δε μπλοκάρεται και σε αυτή την περίπτωση. Από την πλευρά του προγραμματιστή ο αντιπρόσωπος απλά στιγμιοτυποποιεί κάποια από τις κλάσεις που επεκτείνουν μια κλάση Call και την περνάει ως όρισμα στη μέθοδο makeCall() της κλάσης ενθυλάκωσης (wrapper) του αντιπροσώπου.

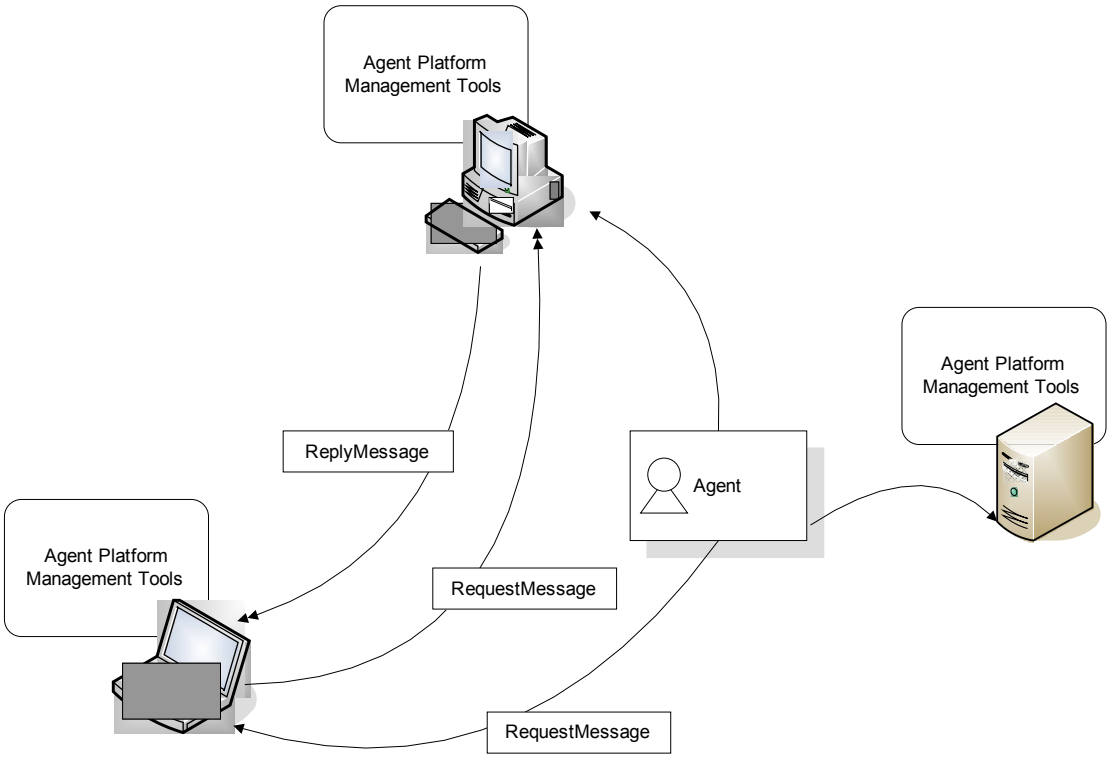
5.3. Η Socket Πλατφόρμα

5.3.1 Sockets

Όπως είναι γνωστό, ένας δικτυακός υποδοχέας (socket) είναι το ένα άκρο μιας αμφίδρομης επικοινωνιακής σύνδεσης μεταξύ δύο προγραμμάτων που εκτελούνται σε ένα δίκτυο. Για κάθε έναν υποδοχέα υπάρχει μια αντιστοίχιση με έναν αριθμό θύρας, έτσι ώστε το πρωτόκολλο επικοινωνίας να αναγνωρίζει την εφαρμογή στην οποία στέλνονται τα δεδομένα. Για τον προγραμματισμό με sockets μπορεί γενικά να γίνει επιλογή μεταξύ δύο πρωτοκόλλων: του User Datagam Protocol (UDP) [8] και του Transmission Control Protocol (TCP) [9]. Το UDP είναι ένα αναξιόπιστο πρωτόκολλο, αφού δεν παρέχει καμμία εγγύηση ότι τα πακέτα που εστάλησαν θα ληφθούν από τον υποδοχέα – παραλήπτη με την σωστή σειρά. Το TCP από την άλλη είναι ένα αξιόπιστο πρωτόκολλο – τα πακέτα που έστειλε ο αποστολέας είναι εγγυημένο ότι θα ληφθούν από τον υποδοχέα – παραλήπτη στην ίδια σειρά με την οποία εστάλησαν. Είναι προφανές ότι για μια πλατφόρμα κινητών πρακτόρων, όπου πρέπει μεταξύ των άλλων να υπάρχει αξιοπιστία κατά τη μετανάστευση των πρακτόρων, μόνο η χρήση των υποδοχέων TCP μπορεί να γίνει αποδεκτή.

5.3.2 Αρχιτεκτονική Επικοινωνίας μέσω Sockets

Ο τρόπος με τον οποίο γίνεται η επικοινωνία με χρήση δικτυακών υποδοχών φαίνεται στο Σχήμα 13:



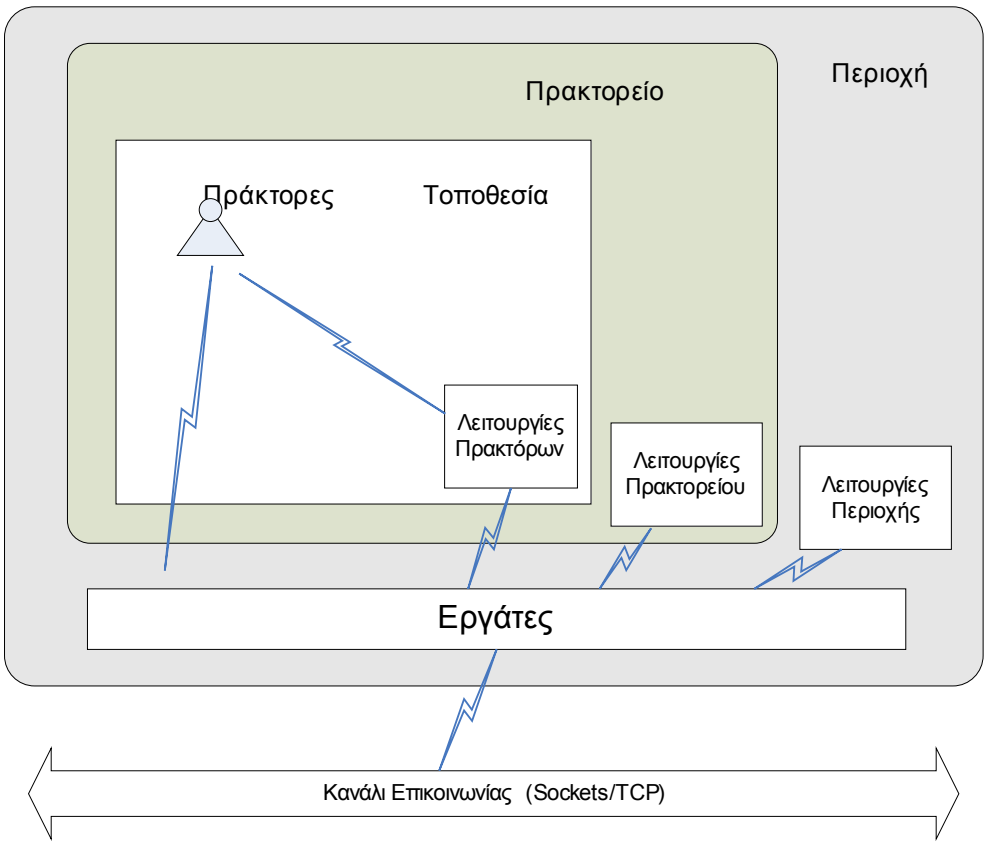
Σχήμα 13. Επικοινωνία μέσω δικτυακών υποδοχών.

Η αρχική πλατφόρμα, η οποία είχε υλοποιηθεί με βάση τις υπηρεσίες ιστού, προσφέρει κάποιες λειτουργίες με τη μορφή υπηρεσιών, τις οποίες μπορούν να χρησιμοποιούν τα διάφορα συστατικά της μέσω του διαδικτύου. Το πρωτόκολλο πάνω στο οποίο στηρίζεται αυτή η μορφή επικοινωνίας είναι το SOAP – RPC, πράγμα που σημαίνει ότι όλες οι συναλλαγές γίνονται με βάση το μοντέλο της απομακρυσμένης κλήσης διαδικασίας. Εφόσον στην παρούσα αρχιτεκτονική χρησιμοποιούμε sockets, είναι αναπόφευκτη η χρήση του μοντέλου πελάτη – εξυπηρετητή. Η επικοινωνία μέσω sockets γίνεται μέσω μιας προσέγγισης προσανατολισμένης στην ανταλλαγή μηνυμάτων. Όταν ένα συστατικό της πλατφόρμας επιθυμεί να ζητήσει κάποια λειτουργία από κάποιο άλλο συστατικό, λειτουργεί σαν πελάτης και στέλνει στο απομακρυσμένο συστατικό ένα μήνυμα για την περιγραφή της λειτουργίας που χρειάζεται αυτό να εκτελέσει. Το μήνυμα αυτό είναι ένα αντικείμενο της κλάσης RequestMessage, όπως φαίνεται και στο παραπάνω σχήμα, η οποία περιέχει όλες τις πληροφορίες που χρειάζονται για την επικοινωνία. Η κλάση RequestMessage περιέχει πεδία που καθορίζουν τη μέθοδο που πρέπει να κληθεί στον παραλήπτη, καθώς και τα ορίσματα αυτής της μεθόδου. Όταν η πλευρά του εξυπηρετητή δεχτεί ένα τέτοιο μήνυμα, εκτελεί την κατάλληλη μέθοδο,η οποία ανήκει σε αντικείμενο που τρέχει στην ίδια εικονική μηχανή, και στέλνει το αποτέλεσμα στο πελάτη μέσω ενός άλλου μηνύματος. Αυτού του είδους το μήνυμα είναι αντικείμενο της κλάσης ReplyMessage, η οποία περιέχει ένα πεδίο για την αποθήκευση του αποτελέσματος της κλήσης της μεθόδου. Η συναλλαγή που μόλις

περιγράφτηκε επιτυγχάνεται χρησιμοποιώντας το χαρακτηριστικό της σειριακοποίησης αντικειμένων που προσφέρει η Java, σε συνδυασμό με streams εισόδου και εξόδου αντικειμένων.

5.3.3 Η αρχιτεκτονική της Socket Πλατφόρμας

Όπως προαναφέρθηκε, η πλατφόρμα ακολουθεί τα πρότυπα των προδιαγραφών OMG MASIF. Τα συστατικά στοιχεία της είναι η περιοχή, το πρακτορείο και η τοποθεσία. Ένα επιπλέον στοιχείο που έχει εισαχθεί στην αρχιτεκτονική της πλατφόρμας είναι η έννοια του εργάτη. Οι εργάτες είναι οντότητες που λειτουργούν ως ενδιάμεσο στρώμα μεταξύ πλατφόρμας και καναλιού επικοινωνίας. Ο τρόπος που τα παραπάνω συστατικά αλληλεπιδρούν φαίνεται στο Σχήμα 14.



Σχήμα 14. Αρχιτεκτονική Socket Πλατφόρμας.

Είναι προφανές ότι για κάθε διαφορετικό κανάλι επικοινωνίας που θα χρησιμοποιηθεί πρέπει να υλοποιηθούν καινούριες κλάσεις εργατών, οι οποίες θα στηρίζονται στα χαρακτηριστικά του πρωτοκόλλου του καναλιού. Αυτό σημαίνει ότι οι κλάσεις των εργατών που χρησιμοποιούνται για την επικοινωνία μέσω καναλιού HTTP με το πρωτόκολλο SOAP θα είναι διαφορετικές από τις κλάσεις των εργατών που χρησιμοποιούνται για την επικοινωνία μέσω sockets με το πρωτόκολλο TCP. Με μια πρώτη ματιά αυτό φαίνεται να δημιουργεί πρόβλημα στη διαλειτουργικότητα των πλατφορμών που υλοποιούνται πάνω σε διαφορετικό πρωτόκολλο επικοινωνίας. Χρησιμοποιώντας όμως ένα επιπλέον στρώμα μεσισμικού, είναι δυνατόν να

επιτευχθεί η επιθυμητή διαλειτουργικότητα, όπως περιγράφεται σε επόμενη παράγραφο.

5.3.4 Η πλευρά του εξυπηρετητή – The server side

Οι οντότητες της πλατφόρμας που πρέπει να παρέχουν κάποιες λειτουργίες σε απομακρυσμένα αντικείμενα, όπως είναι για παράδειγμα οι πράκτορες, είναι η περιοχή και το πρακτορείο. Στην πλευρά αυτών των συστατικών υλοποιήθηκε ένας εξυπηρετητής που δέχεται αιτήσεις από απομακρυσμένα αντικείμενα και εκτελεί την κατάλληλη λειτουργία που προσφέρει το κάθε συστατικό. Για την επίτευξη μιας λειτουργικής πλατφόρμας χρησιμοποιώντας sockets, έγινε χρήση ενός κλασσικού πολυνηματικού εξυπηρετητή, που περιέχει ένα στιγμιότυπο της κλάσης Agency ή της κλάσης Region, ανάλογα με την οντότητα για την οποία προορίζεται. Ο κάθε εξυπηρετητής δημιουργεί ένα αντικείμενο της κλάσης ServerSocket και περιμένει για αιτήσεις επικοινωνίας. Κάθε φορά που γίνεται μια αίτηση επικοινωνίας, δημιουργείται ένα νέο νήμα, το οποίο διαβάζει μέσα από το stream εισόδου του socket ένα σειριοποιημένο αντικείμενο της κλάσης RequestMessage. Ανάλογα με το είδος του μηνύματος, ο εξυπηρετητής επιλέγει την κατάλληλη μέθοδο του συστατικού που πρέπει να κληθεί. Το αποτέλεσμα που επιστρέφεται από τη μέθοδο αποθηκεύεται σε ένα αντικείμενο της κλάσης ReplyMessage, το οποίο με τη σειρά του στέλνεται πίσω στον πελάτη σειριοποιημένο. Φυσικά ο εξυπηρετητής μιας περιοχής έχει διαφορετική υλοποίηση από τον εξυπηρετητή ενός πρακτορείου, αφού οι μέθοδοι που παρέχει η κάθε οντότητα είναι διαφορετικές. Πάντως οι μορφές των RequestMessage και ReplyMessage είναι οι ίδιες ανεξάρτητα από το είδος του εξυπηρετητή για τον οποίο προορίζονται. Εδώ πρέπει να σημειωθεί ότι ενώ στην περίπτωση της SOAP πλατφόρμας η αναγνώριση του κάθε συστατικού της γίνεται μέσω ενός URL, σε μια Socket πλατφόρμα για την αναγνώριση ενός εξυπηρετητή απαιτείται μόνο η διεύθυνση του υπολογιστή στον οποίο εκτελείται ο εξυπηρετητής και ο αριθμός της θύρας με την οποία αυτός έχει συνδεθεί.

5.3.5 Η πλευρά του πελάτη – The client side

Η πλευρά του πελάτη στην Socket Πλατφόρμα υλοποιείται μέσω της διεπαφής των εργατών. Η διεπαφή των εργατών επιτυγχάνει καταρχήν την μεγιστοποίηση της επαναχρησιμοποίησης κώδικα, αλλά κατά κύριο λόγο προσφέρει ένα διαφανή τρόπο επικοινωνίας μεταξύ των οντοτήτων των πλατφορμών ανεξάρτητα από το πρωτόκολλο που χρησιμοποιεί η κάθε πλατφόρμα. Αυτό σημαίνει ότι χρησιμοποιώντας αυτή τη διεπαφή μαζί με ένα επιπλέον στρώμα μεσισμικού που θα περιγραφεί σε επόμενη παράγραφο, γίνεται δυνατή η επικοινωνία μεταξύ πλατφορμών που χρησιμοποιούν είτε SOAP, είτε sockets, είτε οποιοδήποτε άλλο μελλοντικά υποστηριζόμενο πρωτόκολλο (όπως RMI ή CORBA), και μάλιστα με ομοιόμορφο τρόπο.

Οι εργάτες είναι τα αντικείμενα που δημιουργούν τη σύνδεση socket σε έναν απομακρυσμένο υπολογιστή. Κάθε φορά που υπάρχει ανάγκη για αλληλεπίδραση με κάποιο συστατικό μιας απομακρυσμένης πλατφόρμας, δημιουργείται ένας εργάτης. Ο εργάτης δημιουργεί το κατάλληλο στιγμιότυπο της κλάσης RequestMessage, προωθεί το μήνυμα στον εξυπηρετητή και αναστέλλει τη λειτουργία του μέχρι να ληφθεί ένα αντικείμενο της κλάσης ReplyMessage από το stream εισόδου του socket.

Για να κάνει μια τέτοια προσπάθεια σύνδεσης, ένας εργάτης χρειάζεται μόνο τη διεύθυνση του υπολογιστή στον οποίο εκτελείται η περιοχή ή το πρακτορείο και τον αριθμό της θύρας στην οποία είναι συνδεδεμένος ο αντίστοιχος εξυπηρετητής. Εκτός βέβαια από τα στοιχεία αυτά που χρειάζονται μόνο για την επικοινωνία, ένας εργάτης χρειάζεται και τα όποια ορίσματα απαιτεί η μέθοδος που θα κληθεί στην πλευρά του εξυπηρετητή, τα οποία τοποθετεί μέσα στο αντικείμενο της κλάσης RequestMessage που στέλνει σε αυτόν.

5.4. Η Ανεξάρτητη Πρωτοκόλλου Πλατφόρμα

Ένας από τους βασικότερους σκοπούς της εργασίας ήταν η επίτευξη απόλυτης διαφάνειας στην αλληλεπίδραση μεταξύ των συνιστωσών των πλατφορμών, ανεξάρτητα από το χρησιμοποιούμενο πρωτόκολλο. Με άλλα λόγια, όταν μια προσπάθεια επικοινωνίας γίνεται προς μια οντότητα της πλατφόρμας που λειτουργεί είτε με βάση το SOAP, είτε με βάση τα sockets, είτε οποιοδήποτε άλλο πρωτόκολλο, η προσέγγιση πρέπει να είναι σε όλες τις περιπτώσεις η ίδια.

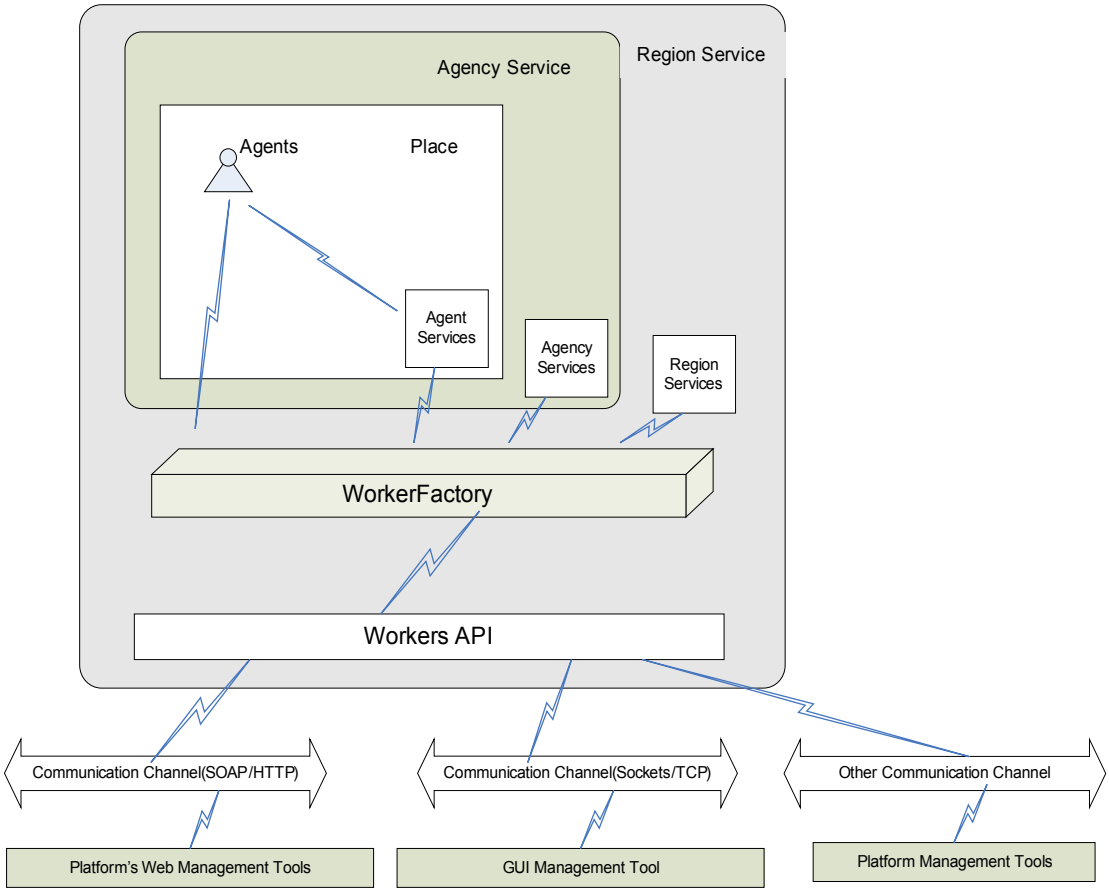
Χρησιμοποιώντας μόνο το ενδιάμεσο στρώμα των εργατών όπως παρουσιάστηκε προηγουμένως, δεν είναι δυνατή η παροχή αυτής της διαφάνειας. Αυτό συμβαίνει γιατί οι εργάτες υλοποιούνται για κάθε πρωτόκολλο ξεχωριστά και για κάθε διαφορετικό πρωτόκολλο υπάρχει ένα διαφορετικό σύνολο από εργάτες.

5.4.1 Η Αρχιτεκτονική της Ανεξάρτητης Πρωτοκόλλου Πλατφόρμας

Για την επίτευξη της ανεξαρτησίας της πλατφόρμας από το υποκείμενο πρωτόκολλο επικοινωνίας, καταρχήν σε κάθε συστατικό της πλατφόρμας εισάγαμε μια νέα κλάση Identifier. Τα αντικείμενα της κλάσης αυτής περιέχουν πληροφορία για το χρησιμοποιούμενο πρωτόκολλο, καθώς και για όλα τα υπόλοιπα στοιχεία που κατασκευάζουν το αναγνωριστικό που χρειάζεται για την επικοινωνία με το συγκεκριμένο συστατικό. Το αναγνωριστικό αυτό εξαρτάται από το υποκείμενο πρωτόκολλο (για παράδειγμα σε μια socket πλατφόρμα απαιτείται η διεύθυνση δικτύου του υπολογιστή στον οποίο εκτελείται η οντότητα, καθώς και ο αριθμός θύρας στην οποία είναι συνδεδεμένος ο αντίστοιχος εξυπηρετητής).

Η σημαντικότερη όμως αλλαγή που χρειάζεται για την επίτευξη της ανεξαρτησίας από το υποκείμενο πρωτόκολλο ήταν η εύρεση ενός τρόπου για την δυναμική επιλογή του σωστού είδους εργάτη που χρειάζεται για το κάθε πρωτόκολλο. Αυτό σημαίνει ότι κάθε φορά που χρειάζεται να γίνει μια προσπάθεια επικοινωνίας, το συστατικό – πελάτης, το οποίο διαθέτει μόνο το αντικείμενο Identifier του συστατικού – εξυπηρετητή και τις παραμέτρους που θα περαστούν ως ορίσματα στην απομακρυσμένη μέθοδο, θα πρέπει να είναι σε θέση να χρησιμοποιήσει το σωστό είδος εργάτη σύμφωνα με τα δεδομένα που διαθέτει και με τρόπο διαφανή.

Για την επίτευξη των παραπάνω καταλήξαμε στην αρχιτεκτονική που απεικονίζεται στο Σχήμα 15.

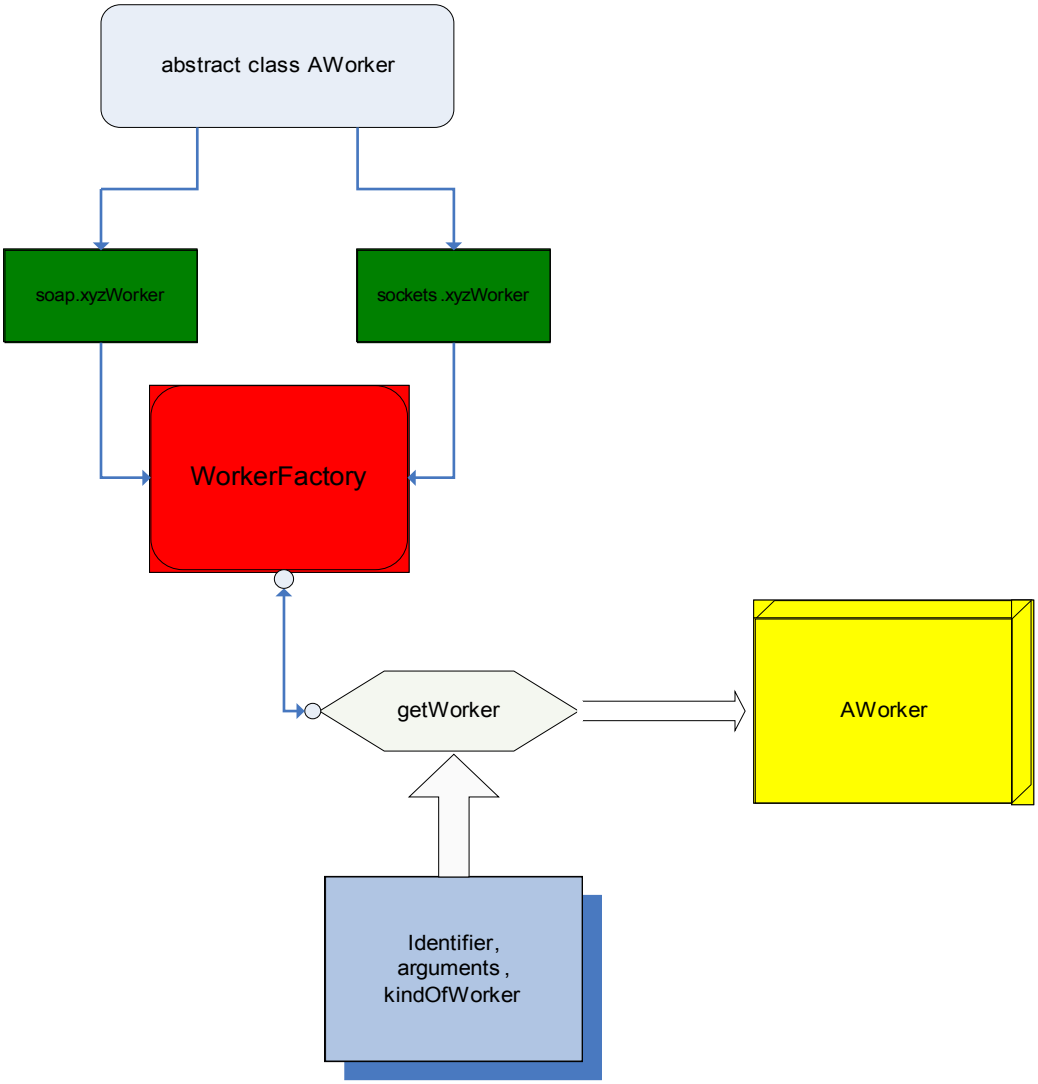


Σχήμα 15. Αρχιτεκτονική της Ανεξάρτητης Πρωτοκόλλου Πλατφόρμας.

Όπως φαίνεται στο παραπάνω σχήμα, το στρώμα των εργατών έχει καταρχήν επεκταθεί έτσι ώστε να μπορεί να χρησιμοποιήσει περισσότερα του ενός πρωτόκολλα επικοινωνίας. Όπως επίσης φαίνεται στο παραπάνω σχήμα, έχει εισαχθεί στην αρχική αρχιτεκτονική της πλατφόρμας (Σχήμα 14) ένα νέο στρώμα μεσισμικού. Αυτό το στρώμα τοποθετείται ανάμεσα στις λογικές συνιστώσες της πλατφόρμας και το στρώμα των εργατών και αποτελείται από ένα αντικείμενο της κλάσης WorkerFactory, η οποία περιγράφεται στην επόμενη παράγραφο με λεπτομέρεια.

5.4.2 Η κλάση WorkerFactory

Το διάγραμμα της κλάσης WorkerFactory φαίνεται στο Σχήμα 16.



Σχήμα 16. Τρόπος λειτουργίας του αντικειμένου WorkerFactory.

Στο παραπάνω σχήμα, AWorker είναι η αφηρημένη κλάση από την οποία κληρονομούν όλοι οι SOAP εργάτες και όλοι οι Socket εργάτες. Η κλάση AWorker υλοποιεί ή ορίζει πεδία και μεθόδους που αποτελούν κοινά χαρακτηριστικά όλων των κλάσεων των εργατών. Κάθε κλάση εργάτη απαιτείται να επεκτείνει την αφηρημένη κλάση AWorker. Η κλάση WorkerFactory είναι μια κλάση που αποφασίζει ποια από αυτές τις υποκλάσεις να επιστρέψει, με βάση τα ορίσματα που της δίνονται. Περιέχει μια μέθοδο getWorker, η οποία δέχεται τα εξής ορίσματα: ορίσματα για την αναγνώριση της διεύθυνσης και του πρωτοκόλλου του παραλήπτη, τα ορίσματα που χρειάζεται η κλήση της ζητούμενης απομακρυσμένης μεθόδου, καθώς και το είδος του εργάτη που πρέπει να επιστραφεί. Η μέθοδος αυτή επιστρέφει το κατάλληλο στιγμιότυπο της κλάσης AWorker. Πρέπει να σημειωθεί ότι το ποιο στιγμιότυπο θα επιστραφεί δεν έχει κάποια ιδιαίτερη σημασία για τον προγραμματιστή, αφού όλα τα στιγμιότυπα έχουν τις ίδιες ακριβώς μεθόδους, αλλά διαφορετικές υλοποιήσεις. Η απόφαση για την επιλογή του σωστού στιγμιότυπου γίνεται αποκλειστικά από ένα αντικείμενο της κλάσης WorkerFactory με βάση το πρωτόκολλο της οντότητας – παραλήπτη και το είδος του εργάτη που απαιτείται. Τα υπόλοιπα ορίσματα που

περνιούνται στη μέθοδο getWorker χρειάζονται μόνο για τη στιγμιοποίηση του εργάτη.

5.5. Σύγκριση μεταξύ SOAP και Socket Πλατφορμών

Στις επόμενες παραγράφους θα παρουσιαστούν οι διαφορές που υπάρχουν μεταξύ των δύο πλατφορμών, με βάση τόσο θεωρητικά δεδομένα, όσο και πειραματικές μετρήσεις.

5.5.1 Sockets vs SOAP

Η πλατφόρμα που κάνει χρήση του πρωτοκόλλου SOAP για την επικοινωνία λειτουργεί με βάση το μοντέλο των υπηρεσιών ιστού. Αυτό σημαίνει ότι εκμεταλλεύεται όλες τις ευκολίες και τα κατανεμημένα χαρακτηριστικά που παρέχουν οι υπηρεσίες ιστού για την ανάπτυξη κατανεμημένων εφαρμογών. Εκτός όμως από αυτές τις ευκολίες που προσφέρουν οι υπηρεσίες ιστού, υπάρχουν ορισμένα μειονεκτήματα που μπορούν να προκαλέσουν προβλήματα σε ορισμένες περιστάσεις.

Καταρχήν, οι υπηρεσίες ιστού βασίζονται στο πρωτόκολλο HTTP, πράγμα που σημαίνει ότι κάνουν χρήση της θύρας 80. Αυτή η θύρα όμως χρησιμοποιείται επίσης από κάθε άλλο πρόγραμμα που χρησιμοποιεί το πρωτόκολλο HTTP. Αυτό έχει ως επακόλουθο το γεγονός ότι οι προφυλάξεις ασφαλείας μπορούν να γίνουν περίπλοκες. Εδώ πρέπει να σημειωθεί ότι το SOAP δεν καθιστά το πρωτόκολλο HTTP λιγότερο ασφαλές από ότι ήταν πάντα, αλλά το χρησιμοποιεί με αρκετά πιο εξεζητημένο και πολύπλοκο τρόπο από ότι συνήθως, οπότε απαιτείται και ένα πιο εξεζητημένο σε σχέση με το συνηθισμένο μοντέλο ασφαλείας για την κίνηση HTTP.

Χρησιμοποιώντας απλά sockets, το ζήτημα της ασφάλειας μπορεί να απλοποιηθεί σημαντικά. Κάθε συστατικό της πλατφόρμας μπορεί να χρησιμοποιεί οποιαδήποτε θύρα, πράγμα που προφανώς σημαίνει ότι δεν χρειάζεται να χρησιμοποιεί τη θύρα του από κοινού με κάποια άλλη εφαρμογή. Αυτό καθιστά τους ελέγχους ασφαλείας πολύ απλούστερους. Επιπλέον, η χρήση Ασφαλούς Στρώματος Υποδοχέων (Secure Socket Layer – SSL) μπορεί να εισαχθεί εύκολα στην οποιαδήποτε εφαρμογή.

Το σημαντικότερο ίσως μειονέκτημα της χρήσης του πρωτοκόλλου SOAP είναι η χαμηλές επιδόσεις λόγω των απαιτήσεων για ανάλυση και μεταφορά μηνυμάτων XML. Η χρήση της XML επιβαρύνει το σύστημα με σημαντικό φορτίο. Αν και η δομή της XML είναι σχετικά απλή, οι απαιτήσεις σε μνήμη και υπολογιστική ισχύ είναι τεράστιες σε σύγκριση με άλλες δυνατές λύσεις. Η δημιουργία και η ανάλυση κειμένων XML είναι μια χρονοβόρα διαδικασία, η οποία μάλιστα απαιτεί μεγάλη ποσότητα μνήμης σε σχέση με τα πραγματικά δεδομένα που περιέχουν τα κείμενα αυτά. Επιπλέον, πρέπει να λαμβάνεται υπόψη ότι τα κείμενα XML πρέπει να μεταφερθούν μέσω ενός δικτύου. Εφόσον σε ένα μήνυμα XML η πραγματική πληροφορία είναι μόνο ένα μικρό μέρος των συνολικών δεδομένων που απαρτίζουν το μήνυμα, το αποτέλεσμα είναι η σπατάλη εύρους ζώνης του δικτύου.

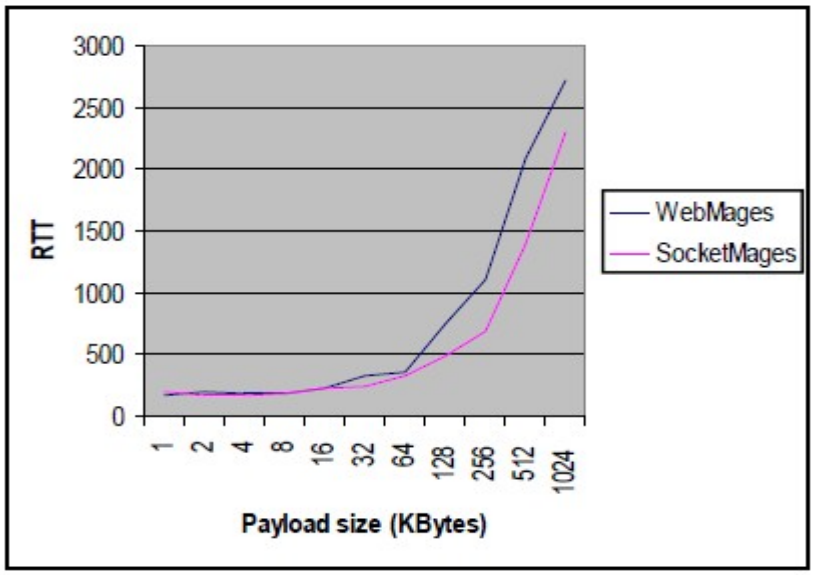
Από την άλλη πλευρά, χρησιμοποιώντας απλά sockets επιτυγχάνουμε σημαντικό κέρδος στην απόδοση του συστήματος. Ο προγραμματισμός σε sockets είναι γενικά ευέλικτος και δίνει στον προγραμματιστή τη δυνατότητα υλοποίησης μιας εφαρμογής με τέτοιο τρόπο ώστε μόνο η απολύτως απαραίτητη πληροφορία να μεταφέρεται πάνω σε ένα δίκτυο, πράγμα που οδηγεί τόσο σε μεγαλύτερη ταχύτητα επικοινωνίας, όσο και στην εξοικονόμηση εύρους ζώνης. Για την υλοποίηση μιας κατανεμημένης

εφαρμογής, συνήθως υιοθετείται η προσέγγιση ανταλλαγής μηνυμάτων. Τα μηνύματα που ανταλλάσσονται μπορούν να περιέχουν μόνο τα απολύτως απαραίτητα δεδομένα, πράγμα που συμβάλει στην αποδοτικότητα της εφαρμογής. Αντίθετα, οι εφαρμογές που στηρίζονται στο πρωτόκολλο SOAP πρέπει να μεταφέρουν ολόκληρες σελίδες ιστού σε κάθε επικοινωνία.

Το trade-off της χρήσης sockets για την υλοποίηση μιας κατανεμημένης εφαρμογής, όπως είναι για παράδειγμα μια πλατφόρμα κινητών πρακτόρων, είναι ότι τόσο η πλευρά του εξυπηρετητή όσο και του πελάτη πρέπει να παρέχουν μηχανισμούς για να καταστήσουν τα μεταφερόμενα δεδομένα χρήσιμα. Τα sockets παρέχουν μηχανισμούς μόνο για την βασική δικτυακή επικοινωνία μεταξύ εφαρμογών, οπότε δεν προσφέρουν καμμία ευκολία. Αυτό κάνει την υλοποίηση μιας εφαρμογής πολύ πιο περίπλοκη.

5.5.2 Σύγκριση μεταξύ των δύο Καναλιών Επικοινωνίας της Πλατφόρμας

Σε αυτό το σημείο κρίνεται σκόπιμη η σύγκριση της απόδοσης των δύο διαφορετικών καναλιών επικοινωνίας της πλατφόρμας, με βάση κάποιες πειραματικές μετρήσεις. Για το σκοπό αυτό δημιουργήθηκε ένας απλός κινητός πράκτορας. Ο πράκτορας αυτός μετακινείται ανάμεσα σε δύο πρακτορεία που φιλοξενούνται σε διαφορετικούς υπολογιστές. Κατά τη δημιουργία, ο πράκτορας δημιουργεί έναν πίνακα από bytes που αντιπροσωπεύει το φορτίο, μεταναστεύει στο απομακρυσμένο πρακτορείο και επιστρέφει στο πρακτορείο που δημιουργήθηκε. Ο πίνακας των bytes χρησιμεύει ως ένας τρόπος για τον έλεγχο του μεγέθους του πράκτορα στη μνήμη. Για την αξιολόγηση της απόδοσης των δύο πλατφορμών χρησιμοποιείται ως μέτρο σύγκρισης ο χρόνος διαδρομής μετ' επιστροφής (Round Trip Time – RTT), δηλαδή ο χρόνος για την μετακίνηση του πράκτορα στο απομακρυσμένο πρακτορείο και για την επαναφορά του στο πρακτορείο που αρχικά δημιουργήθηκε. Αυτό το σενάριο επαναλαμβάνεται αρκετές φορές για διάφορα μεγέθη φορτίων και ο χρόνος RTT μετριέται για κάθε διαφορετικό μέγεθος φορτίου. Τα αποτελέσματα των μετρήσεων φαίνονται στο Σχήμα 17.



Σχήμα 17. Συγκριτικά αποτελέσματα μεταξύ των δύο καναλιών επικοινωνίας της πλατφόρμας.

Από το παραπάνω σχήμα φαίνεται ότι για μικρά μεγέθη φορτίου (1-64 Kbytes), ο μέσος χρόνος RTT για τα δύο διαφορετικά κανάλια επικοινωνίας της πλατφόρμας είναι περίπου ο ίδιος. Για μεγαλύτερα όμως μεγέθη φορτίου, το κανάλι που χρησιμοποιεί τα sockets είναι αρκετά ταχύτερο από το κανάλι που βασίζεται στο πρωτόκολλο SOAP. Αυτό είναι αναμενόμενο, αφού, όπως αναφέρθηκε και στην προηγούμενη παράγραφο, χρειάζεται σημαντικός χρόνος για την ανάλυση μεγάλων κειμένων XML.

5.6. Συμπεράσματα και Δυνατές Μελλοντικές Επεκτάσεις

Στις προηγούμενες παραγράφους παρουσιάστηκε μια αρχιτεκτονική που επιτρέπει σε μια πλατφόρμα κινητών πρακτόρων να λειτουργεί πάνω σε διαφορετικά υποκείμενα πρωτόκολλα επικοινωνίας. Η δυνατότητα μιας πλατφόρμας να υποστηρίζει πολλαπλά πρωτόκολλα είναι από τα πλέον βασικά χαρακτηριστικά για την παροχή διαλειτουργικότητας μεταξύ πλατφορμών που λειτουργούν μέσα σε διαφορετικά περιβάλλοντα. Η συγκεκριμένη αρχιτεκτονική ορίζει κάποια τερματικά σημεία στα οποία μπορούν να προστεθούν νέα στοιχεία για το χειρισμό επιπλέον καναλιών, πράγμα που επιτρέπει την περαιτέρω επέκταση της πλατφόρμας. Το σημαντικότερο χαρακτηριστικό της είναι ότι δεν απαιτεί σημαντικές μετατροπές στην κύρια υλοποίηση της πλατφόρμας. Η συγκεκριμένη αρχιτεκτονική επιτρέπει επιπλέον την διαφανή χρήση των διαφόρων μηχανισμών επικοινωνίας, παρέχοντας στον προγραμματιστή σημαντική απλότητα. Τέλος, από μετρήσεις που έγιναν χρησιμοποιώντας ως κανάλια επικοινωνίας το SOAP και τα sockets, βγήκε το συμπέρασμα ότι χρησιμοποιώντας τα sockets επιτυγχάνουμε καλύτερη απόδοση της πλατφόρμας.

Η βασική επέκταση που μπορεί να γίνει στην αρχιτεκτονική είναι η υλοποίηση επιπλέον κλάσεων εργατών για την υποστήριξη και άλλων πρωτοκόλλων επικοινωνίας, όπως RMI ή CORBA. Σημειώνεται ότι η υλοποίηση αυτών των κλάσεων και η επέκταση της κλάσης του WorkerFactory είναι αρκετά για την υποστήριξη οποιουδήποτε νέου πρωτοκόλλου. Όσον αφορά την ίδια την πλατφόρμα, το βασικό πρόβλημα που πρέπει να επιλυθεί είναι αυτό της ασφάλειας. Το θέμα της ασφάλειας περιλαμβάνει κατά κύριο λόγο το πρόβλημα της αυθεντικοποίησης των αντιπροσώπων και το πρόβλημα της υποκλοπής τους από τρίτους. Ένα δεύτερο στοιχείο επέκτασης όσο αφορά την πλατφόρμα είναι η υλοποίησή της με βάση τις προδιαγραφές FIPA [7].

5.7. Αναφορές

[1] I. E. Foukarakis, A. I. Kostaridis, C. G. Biniaris, D. I. Kaklamani and I. S. Venieris, "Implementation of a Mobile Agent Platform based on Web Services", *Proceedings of the Fifth International Workshop (Mata 2003, Marrakech, Morocco)* on Mobile Agents for Telecommunication Applications, *pp. 190 - 199.*

[2] OMG MASIF, "Mobile Agent System Interoperability Facility (MASIF) Specification", URL: ftp://ftp.omn.org/pub/docs/orbos/97-10-05.pdf

[3] J. Cao, D.C.K. Tse, and A. T.S. Chan, "PDAgent: A Platform for Developing and Deploying Mobile Agent-Enabled Applications for Wireless Devices", in Proc. 2004 International Conference on Parallel Processing (ICPP'04), August 15 - 18, 2004, Montreal, Quebec, Canada, pp. 510-517

[4] JADE mobile agent platform, URL: http://jade.tilab.com

[5] M. Aleksy, A. Korthaus and Martin Schader, “CARLA – A CORBA-based Architecture for Lightweight Agents”, in Proc. IEEE/WIC International Conference on Intelligent Agent Technology, October 13 - 17, 2003, Halifax, Canada, p. 111.

[6] LEAP Lightweight Extensible Agent Platform, URL: http://leap.crm-paris.com/

[7] FIPA 97 Part 2 Version 2.0: Agent Communication Language Specification, URL: http://www.fipa.org/specs/fipa00003

[8] RFC 768, User Datagram Protocol, URL: http://www.rfc-editor.org/rfc/rfc768.txt

[9] RFC 793, Transmission Control Protocol, URL: http://www.rfc-editor.org/rfc/rfc793.txt

Παράρτημα Α – Οδηγός Χρήστη

Σε αυτό το σημείο παρουσιάζεται η γραφική διεπαφή που υλοποιήθηκε για τη διαχείριση των διαφόρων συστατικών της socket πλατφόρμας. Γίνεται αναλυτική περιγραφή του τρόπου με τον οποίο μπορεί να γίνει χρήση της διεπαφής για τη διαχείριση περιοχών και πρακτορείων και τη δημιουργία και διαχείριση των πρακτόρων.

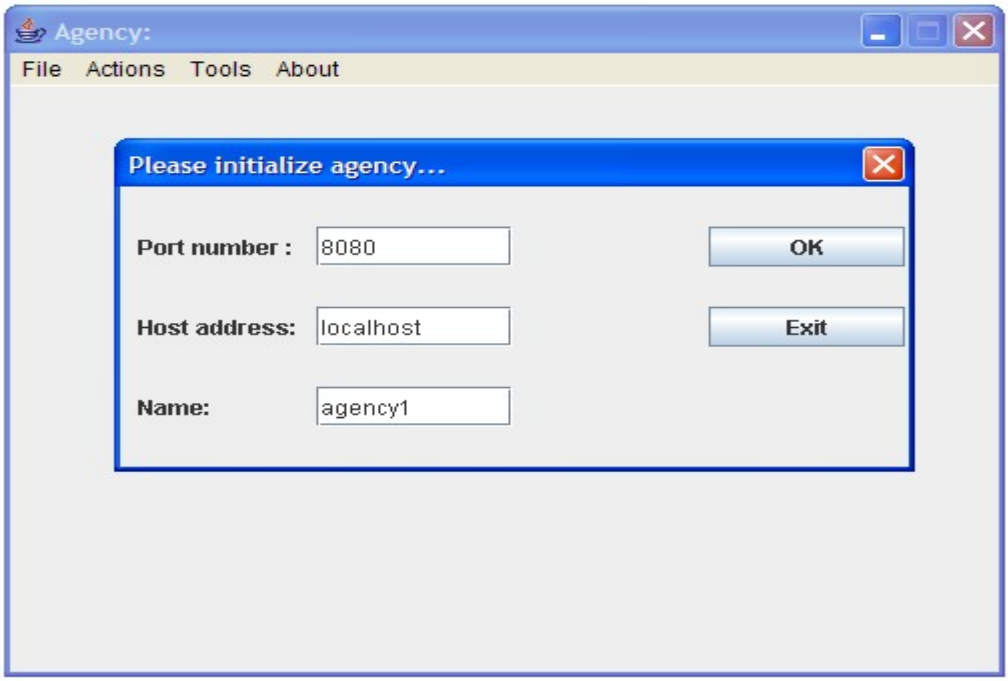
Α.1. Οδηγός εγκατάστασης και απαιτήσεις πλατφόρμας.

Για την υλοποίηση της πλατφόρμας χρησιμοποιήθηκε η έκδοση 1.5.0 του JDK. Οι μόνες απαιτήσεις για την εκτέλεση της πλατφόρμας είναι η εγκατάσταση της έκδοσης 1.5.0 του JRE και η μεταβλητή ClassPath του συστήματος να περιέχει τη διαδρομή στην οποία βρίσκεται το αρχείο log4j-1.2.8.jar. Η πλατφόρμα δεν χρειάζεται κάποια ιδιαίτερη εγκατάσταση για να λειτουργήσει. Αρκεί η αντιγραφή των πακέτων που περιέχουν τις κλάσεις της πλατφόρμας στον δίσκο. Η εκκίνηση ενός πρακτορείου γίνεται με την εκτέλεση του αρχείο start_agency.bat, ενώ η εκκίνηση μιας περιοχής γίνεται με την εκτέλεση του αρχείου start_region.bat.

Α.2. Εργαλεία διαχείρισης.

Α.2.1 Διαχείριση Πρακτορείου.

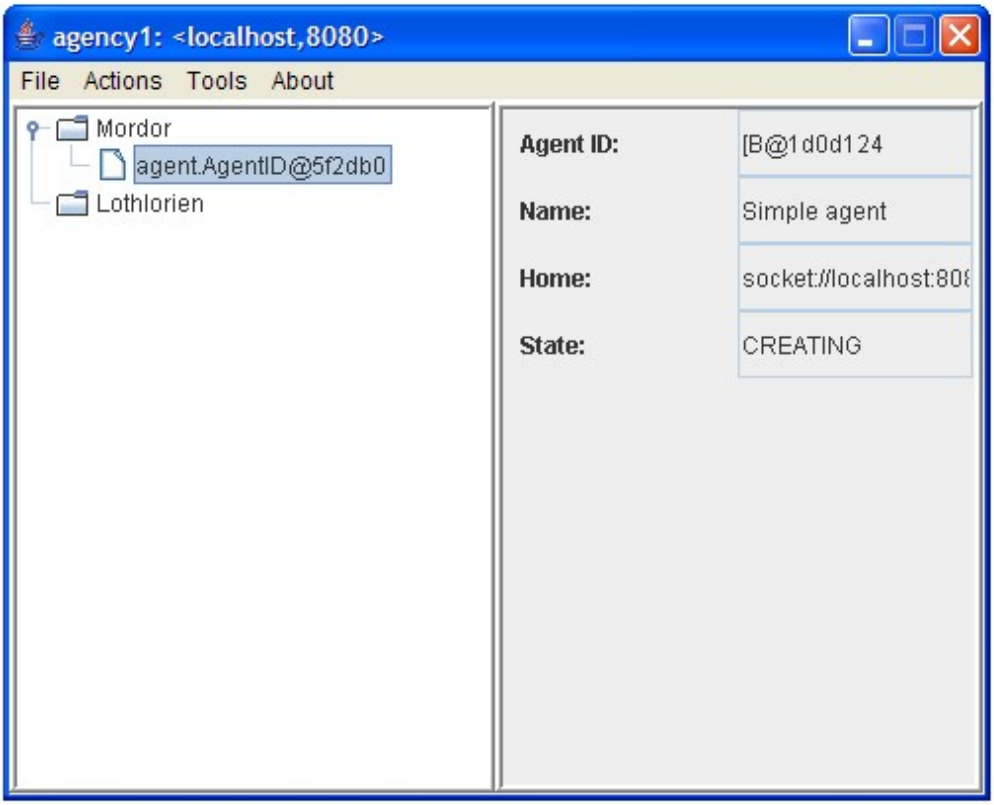
Αμέσως μετά την εκτέλεση του αρχείου start_agency.bat εμφανίζεται στην οθόνη το παράθυρο επιλογής που φαίνεται στο Σχήμα 18.



Σχήμα 18. Παράθυρο αρχικοποίησης πρακτορείου.

Σε αυτό το σημείο γίνεται η αρχικοποίηση των παραμέτρων port number, host address και name του πρακτορείου. Στο πεδίο port number πρέπει να εισαχθεί ο αριθμός της θύρας στην οποία επιθυμούμε να συνδεθεί ο εξυπηρετητής του πρακτορείου. Στη θέση host address εισάγουμε την IP διεύθυνση του υπολογιστή στον οποίο τρέχει το πρακτορείο και στη θέση name το επιθυμητό όνομα του πρακτορείου. Προσοχή απαιτείται στην τοποθέτηση της σωστής διεύθυνσης IP, αφού σε περίπτωση που δώσουμε λάθος τιμή, το πρακτορείο δεν θα μπορεί να επικοινωνήσει με τις άλλες συνιστώσες της πλατφόρμας. Οι default τιμές για τις τρεις αυτές παραμέτρους είναι 8080, localhost και agency1 αντίστοιχα.

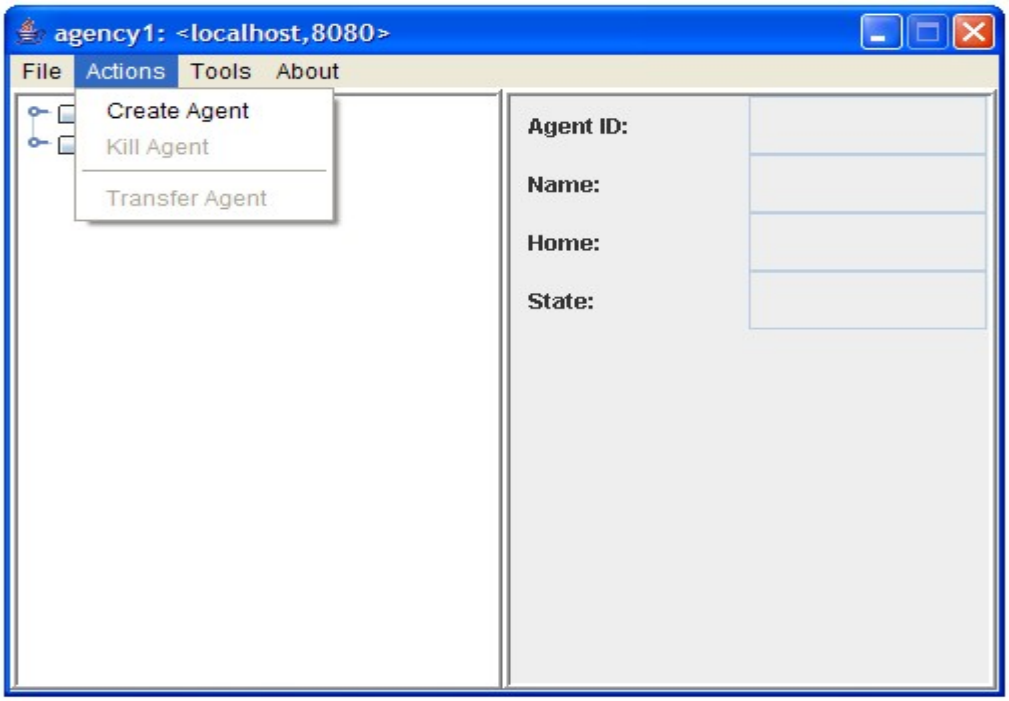
Αφού εισάγουμε τις επιθυμητές τιμές των παραμέτρων που απαιτούνται για την ορθή λειτουργία του πρακτορείου επιλέγουμε το κουμπί “OK”, οπότε εμφανίζεται το παράθυρο του Σχήματος 19.



Σχήμα 19. Παράθυρο διαχείρισης πρακτορείου.

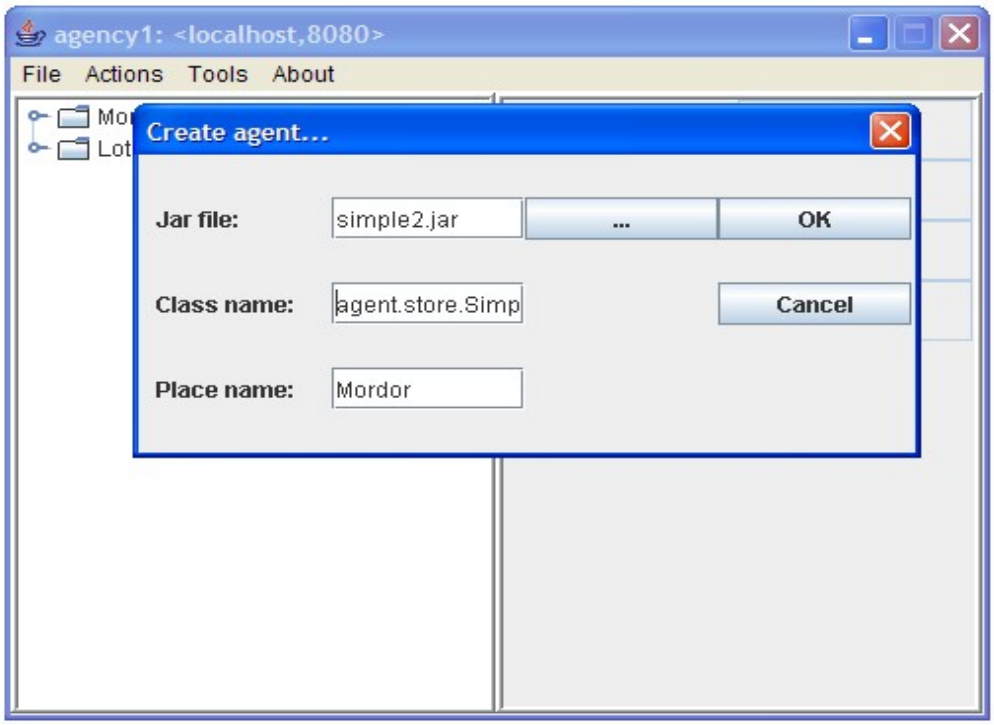
Το παράθυρο αυτό αποτελεί το παράθυρο διαχείρισης του πρακτορείου. Χωρίζεται σε δύο βασικά μέρη. Στα αριστερά υπάρχει μία δενδρική δομή που αναπαριστά τη δομή του πρακτορείου, ενώ στην δεξιά πλευρά υπάρχει ένας πίνακας που παρουσιάζει κάποια στοιχεία του πράκτορα που έχει επιλεγεί στο δένδρο.

Οι βασικές λειτουργίες που μπορεί να εκτελέσει ένα πρακτορείο γίνονται διαθέσιμες μέσω του μενού του παραθύρου. Στο Σχήμα 20 φαίνεται το μενού για τη διαχείριση των πρακτόρων.



Σχήμα 20. Μενού διαχείρισης πρακτόρων.

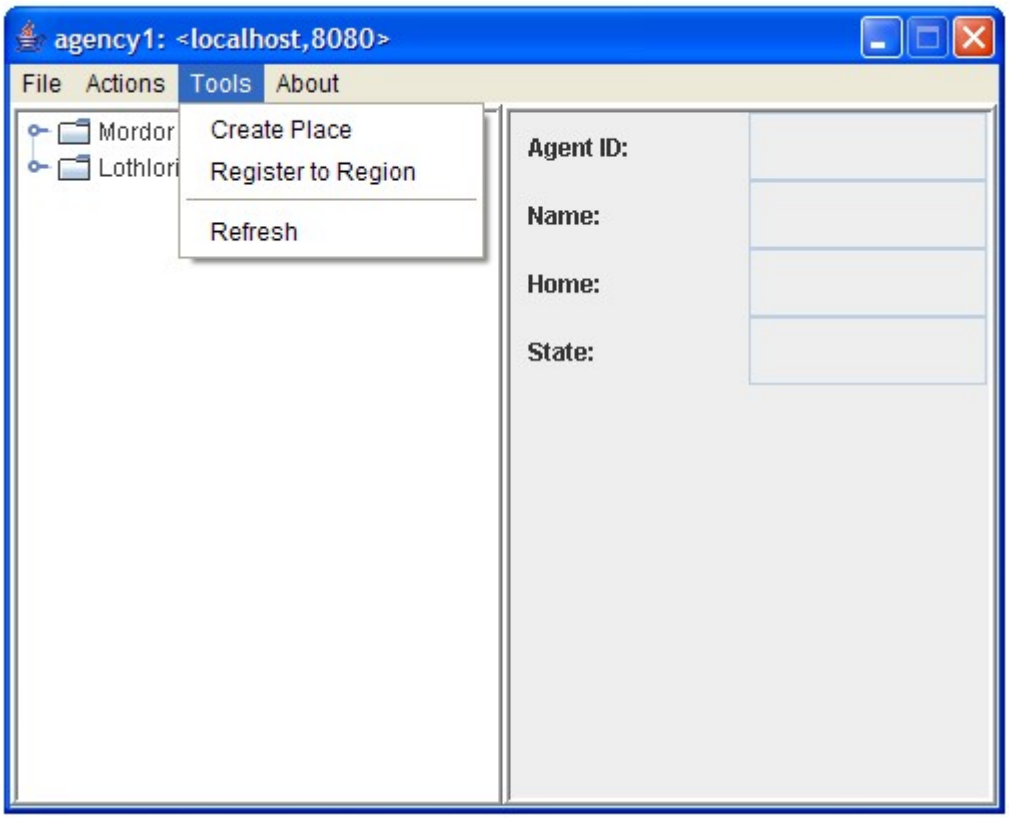
Το μενού περιλαμβάνει τις λειτουργίες δημιουργία πράκτορα, τερματισμός πράκτορα και μεταφορά πράκτορα. Οι επιλογές Kill Agent και Transfer Agent ενεργοποιούνται μόνο όταν έχει επιλεγεί από το δένδρο ένας πράκτορας. Για τη δημιουργία ενός πράκτορα επιλέγουμε το Create Agent, οπότε εμφανίζεται το μενού του Σχήματος 21.



Σχήμα 21. Παράθυρο δημιουργίας πράκτορα.

Εδώ χρειάζεται λίγη προσοχή στις παραμέτρους που πρέπει να εισάγουμε στο προηγούμενο παράθυρο για τη σωστή δημιουργία ενός πράκτορα. Η πρώτη παράμετρος αντιστοιχεί στο jar αρχείο που περιέχει το .class αρχείο του πράκτορα που επιθυμούμε να φορτώσουμε. Στο αρχείο αυτό πρέπει να υπάρχει όλο το μονοπάτι στο οποίο βρίσκεται το .jar αρχείο στο δίσκο και το ίδιο το αρχείο πρέπει να τοποθετηθεί στη σχετική θέση gr\ntua\itsmod\thathan\agent\store. Επιπλέον, στο πεδίο Class Name πρέπει να τοποθετηθεί ολόκληρη η διαδρομή μέσα στα πακέτα που ανήκει η κλάση του πράκτορα. Για παράδειγμα, έστω ότι θέλουμε να φορτώσουμε έναν πράκτορα του οποίου το αρχείο .class υπάρχει στο σχετικό μονοπάτι δίσκου agent/store/SimpleAgent.class. Σε αυτήν την περίπτωση, πρέπει αρχικά να δημιουργήσουμε ένα .jar αρχείο με την εξής εντολή:
jar -cf simple2.jar agent\store\SimpleAgent.class
Στη συνέχεια πρέπει να μεταφέρουμε το αρχείο αυτό στη θέση που αναφέραμε πιο πάνω.
Στο πεδίο Class Name πρέπει να εισάγουμε το κείμενο agent.store.SimleAgent.class. Δεν πρέπει να ξεχνάμε να εισάγουμε το πλήρες όνομα του πακέτου στο οποίο ανήκει η κλάση, καθώς και το .class στο τέλος.
Στο πεδίο Place Name εισάγουμε την τοποθεσία του πρακτορείου στην οποία θέλουμε να ανήκει ο πράκτορας.

Στο μενού Tools υπάρχουν κάποια εργαλεία για τη διαχείριση θεμάτων που αφορούν το ίδιο το πρακτορείο. Το μενού Tools φαίνεται στο Σχήμα 22.



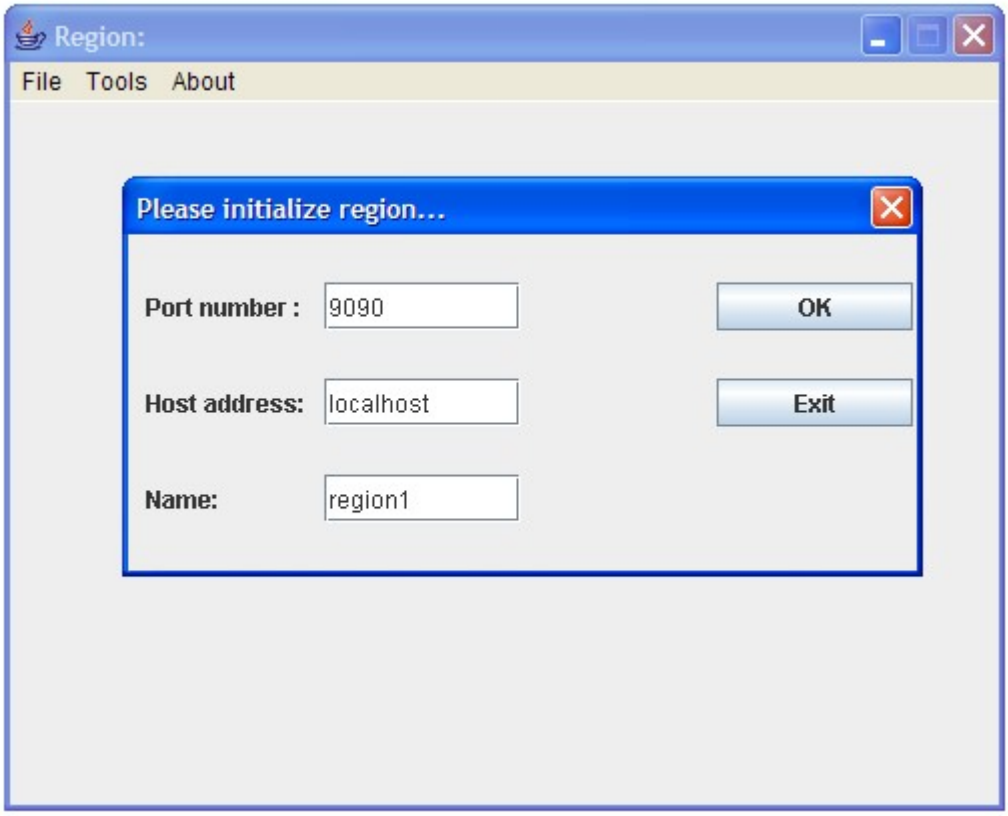
Σχήμα 22. Μενού εργαλείων για τη διαχείριση του πρακτορείου.

Το μενού αυτό περιλαμβάνει επιλογές για δημιουργία νέας τοποθεσίας, για καταχώρηση σε μια περιοχή και για ανανέωση του δένδρου που αναπαριστά τη δομή του πρακτορείου και βρίσκεται στα αριστερά του παραθύρου. Επιλέγοντας Create

Place εμφανίζεται ένα παράθυρου εισόδου στο οποίο απλώς εισάγουμε το όνομα της τοποθεσίας που θέλουμε να δημιουργήσουμε. Αν επιλέξουμε Register to Region, θα εμφανιστεί ένα παράθυρο που ζητάει τη διεύθυνση και τη θύρα της περιοχής στην οποία θέλουμε να καταχωρήσουμε το πρακτορείο.

Α.2.2 Διαχείριση Περιοχής.

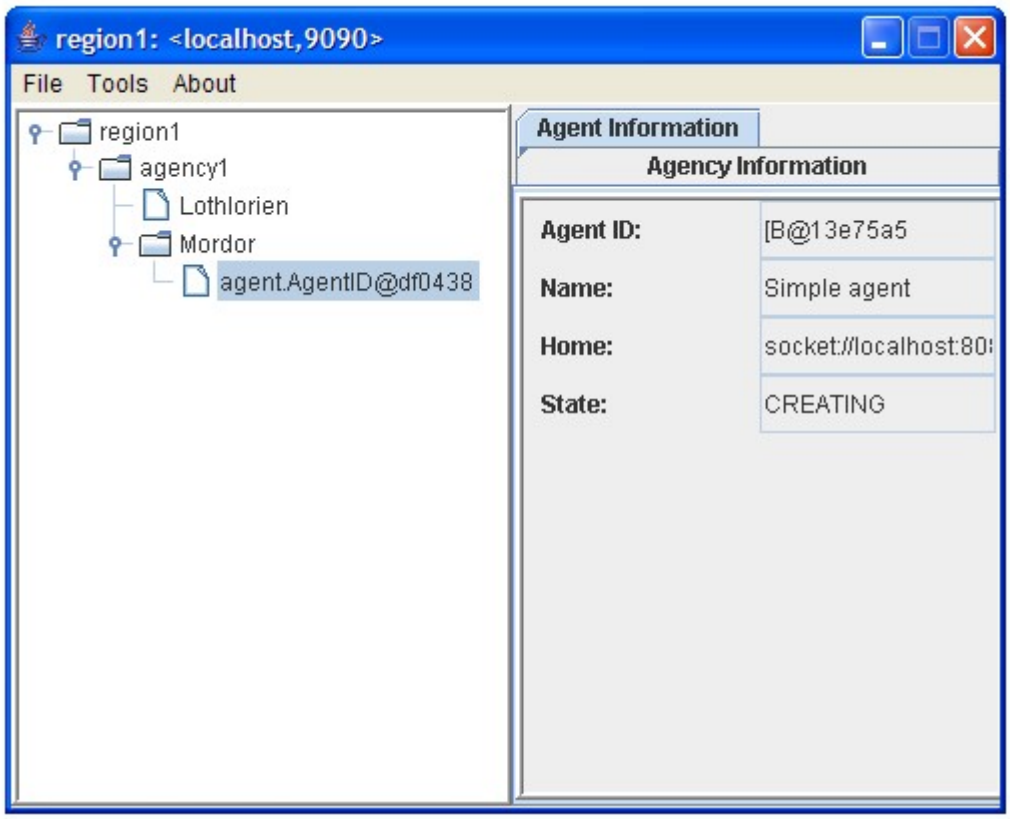
Εκτελώντας το αρχείο start_region.bat εμφανίζεται στην οθόνη του υπολογιστή το παράθυρο αρχικοποίησης που φαίνεται στο Σχήμα 23.



Σχήμα 23. Παράθυρο αρχικοποίησης περιοχής.

Το παράθυρο αυτό είναι τελείως ανάλογο με το παράθυρο αρχικοποίησης πρακτορείου που φαίνεται στο Σχήμα 18. Οι default τιμές για αριθμό θύρας, διεύθυνση IP και όνομα είναι αντίστοιχα 9090, localhost και region1. Σε αυτό το σημείο πρέπει να σημειωθεί ότι πρέπει φυσικά να προσέχουμε να μην χρησιμοποιούμε τον ίδιο αριθμό θύρας σε οντότητες της πλατφόρμας που τρέχουν στον ίδιο υπολογιστή, καθώς θα υπάρξει σύγκρουση.

Αφού ξεκινήσει η εκτέλεση μιας περιοχής, μπορούμε να χρησιμοποιούμε τις επιλογές που παρέχει το παράθυρο διαχείρισης των πρακτορείων για να καταχωρούμε τα πρακτορεία στην περιοχή. Στο επόμενο σχήμα (Σχήμα 24) φαίνεται η μορφή που μπορεί να έχει το παράθυρο διαχείρισης της περιοχής κατά τη διάρκεια εκτέλεσής της.



Σχήμα 24. Παράθυρο διαχείρισης περιοχής.

Το παράθυρο διαχείρισης της περιοχής επίσης χωρίζεται σε δύο βασικά μέρη. Στα αριστερά υπάρχει και πάλι μια δενδρική δομή που οπτικοποιεί τη δομή της περιοχής. Ρίζα του δένδρου είναι η περιοχή, κλαδιά της περιοχής είναι τα καταχωρημένα πρακτορεία, κλαδιά των πρακτορείων είναι οι τοποθεσίες τους και κλαδιά των τοποθεσιών είναι οι πράκτορες που υπάρχουν σε αυτές. Στην δεξιά πλευρά εμφανίζονται πληροφορίες για τον επιλεγμένο κόμβο της δενδρικής δομής, διατεταγμένες σε μορφή καρτέλας. Στην καρτέλα με τίτλο Agent Information υπάρχουν πληροφορίες για τον επιλεγμένο πράκτορα και στην καρτέλα με τίτλο Agency Information υπάρχουν πληροφορίες για το επιλεγμένο πρακτορείο. Στο παραπάνω σχήμα φαίνεται η περίπτωση που έχει επιλεγεί στο δένδρο ένας πράκτορας. Το μενού Tools παρέχει επιλογές για καταχώρηση νέου πρακτορείου (Register Agency), για αφαίρεση κάποιου ήδη καταχωρημένου πρακτορείου (Unregister Agency), καθώς και για ανανέωση του δένδρου στα αριστερά του παραθύρου.

Παράρτημα Β – Οδηγός Προγραμματιστή

Σε αυτό το κεφάλαιο γίνεται περιγραφή του τρόπου ανάπτυξης κινητών πρακτόρων και των δυνατοτήτων τους, μαζί με παραδείγματα. Για την εκτέλεση των παραδειγμάτων απαιτείται να υπάρχει σε λειτουργία ένα τουλάχιστον πρακτορείο, το οποίο παρέχει το περιβάλλον εκτέλεσης για έναν πράκτορα, ενώ σε παραδείγματα που εμπλέκονται και υπηρεσίες καταλόγου απαιτείται και η λειτουργία μιας περιοχής.

Β.1. Αντιπρόσωποι

Προγραμματιστικά, οι αντιπρόσωποι είναι προγράμματα Java, αποτελούμενα από μία ή περισσότερες κλάσεις. Ο κάθε αντιπρόσωπος είναι απόγονος μιας από τις κλάσεις StationaryAgent και MobileAgent.

Η πρώτη απόφαση κατά την υλοποίηση ενός αντιπροσώπου είναι το ποια από τις παραπάνω κλάσεις θα επεκτείνει. Η πρώτη προορίζεται για αντιπροσώπους οι οποίοι κατά τη διάρκεια της ζωής τους δεν πρόκειται να μετακινηθούν από την αντιπροσωπεία στην οποία δημιουργήθηκαν. Η δεύτερη χρησιμοποιείται για αντιπροσώπους που ενδέχεται να μετακινηθούν κατά τη διάρκεια της ζωής τους.

Και οι δύο τύποι αντιπροσώπων είναι απόγονοι της κλάσης Agent. Στην κλάση αυτή ορίζονται τρεις μέθοδοι: η init, η start και η destroy. Ο κύκλος ζωής ενός αντιπροσώπου ακολουθεί την παραπάνω σειρά, δηλαδή πρώτα καλείται η init, μετά η start, και τέλος η destroy. Κάθε αντιπρόσωπος μπορεί να υλοποιήσει μία, μερικές ή όλες αυτές τις μεθόδους. Ο σκοπός ύπαρξης της κάθε μίας είναι συγκεκριμένος. Η init υπάρχει για να μπορεί να γίνει κάποια αρχικοποίηση του αντιπροσώπου. Αν και η αρχικοποίηση μπορεί να γίνει και στον constructor, είναι προτιμότερο να πραγματοποιείται στην init. Η start περιέχει το «χρήσιμο» κώδικα του αντιπροσώπου, τον κώδικα δηλαδή που αναλαμβάνει να εκτελέσει την εργασία για την οποία έχει δημιουργηθεί ο αντιπρόσωπος. Τέλος, η destroy καλείται αμέσως μόλις τελειώσει η start, και χρησιμοποιείται για οποιαδήποτε ανάγκη υπάρχει στο τέλος της εκτέλεσης του αντιπροσώπου (π.χ. αποδέσμευση μνήμης). Μια σημαντική λεπτομέρεια είναι ότι μετά το τέλος της destroy, ο αντιπρόσωπος συνεχίζει να υπάρχει στην αντιπροσωπεία, με τη διαφορά ότι τώρα θεωρείται ανενεργός.

Ακολουθεί ένα απλό παράδειγμα αντιπροσώπου:

```
import gr.ntua.ifouk.agent.MobileAgent;

public class HelloAgent extends MobileAgent {
      public HelloAgent() {
      }

      public void start() {
            System.out.println("Hello Agent World!!!");
      }
}
```

Ο παραπάνω αντιπρόσωπος είναι της κατηγορίας MobileAgent, δηλαδή έχει τη δυνατότητα να μετακινηθεί σε άλλη αντιπροσωπία. Υλοποιεί μόνο τη start, η οποία τυπώνει το μήνυμα “Hello Agent World!!!”.

Εκτός από τις παραπάνω μεθόδους, ο αντιπρόσωπος μπορεί να υλοποιεί και τη μέθοδο getName. Η αξία αυτής της μεθόδου είναι ότι επιστρέφει ένα όνομα με το οποίο μπορούμε να ξεχωρίσουμε τον αντιπρόσωπο στο εργαλείο διαχείρισης.

Β.2. Πληροφορίες Αντιπροσώπων

Κάθε αντιπρόσωπος έχει πρόσβαση σε κάποιες πληροφορίες. Οι πληροφορίες αυτές είναι αποθηκευμένες σε ένα αντικείμενο της κλάσης AgentInfo, και ο κάθε αντιπρόσωπος μπορεί να πάρει μια αναφορά στο αντικείμενο αυτό χρησιμοποιώντας την μέθοδο getInfo. Η μέθοδος αυτή υλοποιείται στην κλάση Agent, με αποτέλεσμα να είναι διαθέσιμη σε όλους τους τύπους των αντιπροσώπων.

Οι πληροφορίες που περιέχει το αντικείμενο αυτό, καθώς και η μέθοδος που πρέπει να κληθεί για να αποκτήσουμε πρόσβαση σε αυτές φαίνεται στον πίνακα 9.1.

Μέθοδος	Τύπος	Περιγραφή
getName	String	Επιστρέφει το όνομα του αντιπροσώπου
getId	AgentID	Επιστρέφει ένα αντικείμενο που αναπαριστά ένα αναγνωριστικό
getType	int	Επιστρέφει τον τύπο του αντιπροσώπου (παίρνει τις τιμές AgentInfo.UNKNOWN, AgentInfo.MOBILE, AgentInfo.STATIONARY)
getLastHost	String	Επιστρέφει τη διεύθυνση της τελευταίας αντιπροσωπείας στην οποία βρέθηκε ο αντιπρόσωπος
getLastPlace	String	Επιστρέφει το προηγούμενο μέρος στο οποίο βρέθηκε ο αντιπρόσωπος
getHomeHost	String	Επιστρέφει τη διεύθυνση της αντιπροσωπείας στην οποία δημιουργήθηκε ο αντιπρόσωπος
getHomePlace	String	Επιστρέφει το όνομα του μέρους που δημιουργήθηκε ο αντιπρόσωπος
isActive	boolean	Επιστρέφει true αν ο αντιπρόσωπος είναι ενεργός ή false αν είναι ανενεργός.

Πίνακας 1. Πληροφορίες διαθέσιμες από το αντικείμενο AgentInfo.

Β.3. Μετακίνηση

Οι αντιπρόσωποι που είναι απόγονοι της κλάσης MobileAgent έχουν την ικανότητα να μετακινηθούν σε κάποια άλλη αντιπροσωπεία. Σε κάθε μετακίνηση πραγματοποιείται μια σειρά από διαδικασίες. Το πρώτο πράγμα που συμβαίνει είναι ότι η αντιπροσωπεία «παγώνει» την εκτέλεση του νήματος του αντιπροσώπου. Στη συνέχεια καλείται η μέθοδος beforeMove. Η μέθοδος αυτή έχει δηλωθεί στην κλάση MobileAgent, και όλοι οι κινητοί αντιπρόσωποι μπορούν να την υλοποιήσουν, με σκοπό να εκτελέσουν όποια διαδικασία θέλουν αμέσως πριν μετακινηθούν στη νέα αντιπροσωπεία (π.χ. αποδέσμευση τοπικών πόρων). Αμέσως μετά πραγματοποιείται η μετακίνηση. Μετά την μετακίνηση, στη νέα αντιπροσωπεία, καλείται η μέθοδος afterMove. Όπως και η beforeMove, έτσι και η afterMove είναι δηλωμένη στην

κλάση MobileAgent. Σκοπός της είναι να παρέχει έναν τρόπο για την εκτέλεση κώδικα αρχικοποίησης αμέσως μετά τη μετακίνηση του αντιπροσώπου. Τέλος, καλείται και πάλι η start. Προσοχή πρέπει να δοθεί στο γεγονός ότι ο αντιπρόσωπος δε συνεχίζει την εκτέλεσή του από το σημείο στο οποίο βρισκόταν πριν τη μετακίνηση, αλλά ξαναξεκινά με τη start.

Η μετακίνηση του αντιπροσώπου μπορεί να γίνει με δύο τρόπους. Ο πρώτος είναι όποτε επιθυμεί κάποιος χρήστης, να χρησιμοποιήσει το εργαλείο διαχείρισης της αντιπροσωπείας, να επιλέξει τον αντιπρόσωπο, και δίνοντας τη διεύθυνση της νέας αντιπροσωπείας να προκαλέσει τη μετακίνηση. Ο δεύτερος τρόπος είναι προγραμματιστικά, μέσα από τον ίδιο τον αντιπρόσωπο, καλώντας τη μέθοδο move. Η μέθοδος αυτή παίρνει δύο ορίσματα, τη διεύθυνση της αντιπροσωπείας στην οποία θέλει να μετακινηθεί ο αντιπρόσωπος, καθώς και το μέρος στο οποίο επιθυμεί να μεταβεί. Μόλις καλείται αυτή η μέθοδος, ειδοποιείται η τοπική αντιπροσωπεία ότι ο αντιπρόσωπος επιθυμεί να μετακινηθεί, και ξεκινά η διαδικασία μετανάστευσης.

Β.4. Επικοινωνία

Β.4.1 Επικοινωνία μεταξύ αντιπροσώπων

Η επικοινωνία ανάμεσα στους αντιπροσώπους είναι ένα ιδιαίτερα σημαντικό ζήτημα. Είναι ο τρόπος με τον οποίο ανεξάρτητες οντότητες (αντιπρόσωποι) μπορούν να συντονιστούν και να ανταλλάξουν πληροφορίες. Παρόλο που αυτή η διαδικασία είναι τόσο σημαντική, η πολυπλοκότητά της προγραμματιστικά δεν είναι πολύ μεγάλη.

Για να επικοινωνήσει ένας αντιπρόσωπος με κάποιον άλλο αντιπρόσωπο υπάρχουν τρεις διαφορετικοί τρόποι. Ο πρώτος είναι η σύγχρονη επικοινωνία, κατά την οποία ο αντιπρόσωπος που καλεί κάποιον άλλο αντιπρόσωπο περιμένει το αποτέλεσμα, και μετά συνεχίζει την εκτέλεσή του. Ο δεύτερος τρόπος είναι η ασύγχρονη επικοινωνία. Ο αντιπρόσωπος πραγματοποιεί την κλήση, αλλά συνεχίζει την εκτέλεσή του. Περιοδικά, μπορεί να ελέγχει αν έχει έρθει η απάντηση. Ο τελευταίος τρόπος είναι η επικοινωνία με επανάκληση. Ο αντιπρόσωπος πραγματοποιεί την κλήση και συνεχίζει την εκτέλεσή του. Η διαφορά από την ασύγχρονη επικοινωνία είναι ότι τώρα ο αντιπρόσωπος δηλώνει μια μέθοδό του η οποία θα κληθεί όταν επιστρέψει το αποτέλεσμα. Ένα UML διάγραμμα των τριών κλάσεων που υλοποιούν τις παραπάνω μεθόδους φαίνεται στο σχήμα 6.7. Η κλάση SynchCall υλοποιεί τη σύγχρονη επικοινωνία, η AsynchCall υλοποιεί την ασύγχρονη και η CallBackCall την επικοινωνία με επανάκληση. Για να πραγματοποιήσει μια κλήση ο αντιπρόσωπος, πρέπει να γνωρίζει τη διεύθυνση της αντιπροσωπείας που βρίσκεται ο αντιπρόσωπος του οποίου μέθοδο θέλει να καλέσει, καθώς και το αναγνωριστικό του απομακρυσμένου αντιπροσώπου (AgentID). Η κλήση πραγματοποιείται καλώντας τη μέθοδο makeCall, την οποία κληρονομεί από την κλάση Agent. Η μέθοδος αυτή παίρνει τρία ορίσματα: τη διεύθυνση της αντιπροσωπείας στην οποία βρίσκεται ο απομακρυσμένος αντιπρόσωπος, το αναγνωριστικό του αντιπροσώπου και το αντικείμενο που περιέχει τα δεδομένα της κλήσης. Και στις τρεις περιπτώσεις επικοινωνίας επιστρέφεται ένα αντικείμενο της κλάσης CallResponse. Το αντικείμενο αυτό αναπαριστά την απάντηση στη κλήση. Οι κυριότερες μέθοδοί του είναι η hasFinished, η οποία επιστρέφει true αν έχει έρθει η απάντηση από τον απομακρυσμένο αντιπρόσωπο ή false αλλιώς, και η getResponse, η οποία επιστρέφει ένα αντικείμενο Object που περιέχει την απάντηση (αν αυτή έχει έρθει). Ο

αντιπρόσωπος με το κατάλληλο casting μπορεί να χρησιμοποιήσει αυτή την απάντηση.

Β.4.2 Επικοινωνία μεταξύ αντιπροσώπων και άλλων συστατικών

Εκτός από την επικοινωνία με άλλους αντιπροσώπους, ένας αντιπρόσωπος ενδέχεται κατά τη διάρκεια ζωής του να χρειαστεί να επικοινωνήσει και με άλλες οντότητες της πλατφόρμας, δηλαδή με πρακτορεία και περιοχές. Για παράδειγμα, ένας πράκτορας – συντονιστής ενός κατανεμημένου υπολογισμού χρειάζεται να βρει τα πρακτορεία στα οποία μπορεί να στείλει πράκτορες – εργάτες σε μια δεδομένη στιγμή. Για να αποκτήσει μια τέτοια πληροφορία, πρέπει να ζητήσει από την περιοχή μια λίστα με τα πρακτορεία που είναι καταχωρημένα σε αυτήν. Αυτό μπορεί να γίνει με τις ακόλουθες γραμμές κώδικα:

```
Identifier regionID = new Identifier(Identifier.SOCKET,
                                     147.102.232.155,8080);
Worker worker = WorkerFactory.getWorker(regionID,null,
                                    WorkerFactory.LIST_AGENCIES);
while(!worker.isResultReady())
     ;
Vector agencies = (Vector)worker.getResult();
```

Αρχικά δημιουργείται ένα αντικείμενο της κλάσης Identifier που λειτουργεί ως διεύθυνση της περιοχής με την οποία θα επικοινωνήσει ο πράκτορας. Η κλάση αυτή παίρνει ως πρώτο όρισμα το πρωτόκολλο που χρησιμοποιεί το αντικείμενο – στόχος, ως δεύτερο όρισμα την IP διεύθυνση του υπολογιστή στον οποίο τρέχει το αντικείμενο και ως τρίτο όρισμα τη θύρα με την οποία έχει συνδεθεί. Στη συνέχεια δημιουργείται ένας εργάτης μέσω της μεθόδου getWorker της κλάσης WorkerFactory. Η μέθοδος αυτή παίρνει ως πρώτο όρισμα το αναγνωριστικό του αντικειμένου – στόχου, ως δεύτερο όρισμα τις παραμέτρους για τη μέθοδο που θα κληθεί στο απομακρυσμένο αντικείμενο και ως τρίτο όρισμα έναν ακέραιο που υποδηλώνει τον τύπο του εργάτη που θα επιστραφεί από τη μέθοδο. Τέλος, καλώντας απλώς τη μέθοδο getResult του εργάτη, παίρνουμε το αποτέλεσμα που επιστρέφει η μέθοδος του απομακρυσμένου αντικειμένου, προσέχοντας να κάνουμε το σωστό casting δεδομένων ανάλογα με τον τύπο δεδομένων που επιστρέφει ο κάθε εργάτης. Η μέθοδος που καλείται απομακρυσμένα στο παραπάνω παράδειγμα επιστρέφει ένα αντικείμενο της κλάσης Vector, οπότε γίνεται και το ανάλογο casting.

Γενικά, κάθε φορά που ένας πράκτορας πρέπει να αλληλεπιδράσει με ένα πρακτορείο ή μια περιοχή, κάνουμε χρήση της κλάσης WorkerFactory. Συγκεκριμένα, καλώντας τη στατική μέθοδο getWorker της κλάσης WorkerFactory, ένας πράκτορας μπορεί να δημιουργήσει έναν εργάτη που με τη σειρά του θα εκτελέσει κάποια λειτουργία σε ένα απομακρυσμένο συστατικό. Καλώντας στη συνέχεια τη μέθοδο getResult του εργάτη που δημιουργήσαμε προηγουμένως, παίρνουμε το αποτέλεσμα της απομακρυσμένης μεθόδου.

Οι πλήρεις λεπτομέρειες σχετικά με το είδος των εργατών που έχουν υλοποιηθεί αυτή τη στιγμή στην πλατφόρμα και που μπορούν να χρησιμοποιηθούν για την επικοινωνία πρακτόρων με τις οντότητες της πλατφόρμας υπάρχουν στο Παράρτημα Γ – API Πλατφόρμας.

Β.5. Υπηρεσίες καταλόγου

Μια απορία που προκύπτει από τα παραπάνω είναι πως μπορεί ένας αντιπρόσωπος να εντοπίσει κάποιον άλλο αντιπρόσωπο έτσι ώστε να μπορέσει να επικοινωνήσει μαζί του. Η λύση σε αυτό το πρόβλημα είναι οι υπηρεσίες καταλόγου που προσφέρει η region. Ο αντιπρόσωπος μπορεί να χρησιμοποιήσει τη μέθοδο search για αυτή την εργασία. Η μέθοδος αυτή δέχεται σαν όρισμα ένα αντικείμενο της κλάσης AgentFilter, το οποίο περιέχει τα κριτήρια αναζήτησης, και επιστρέφει ένα αντικείμενο της κλάσης Result, η οποία περιέχει πληροφορίες σχετικά με το σύνολο των αντιπροσώπων που ταίριαζαν στα κριτήρια. Τα κριτήρια της αναζήτησης μπορούν να τεθούν στην AgentFilter μέσω της μεθόδου της setConstraint. Η μέθοδος αυτή δέχεται δύο ορίσματα. Το πρώτο είναι το κριτήριο αναζήτησης και το δεύτερο η τιμή την οποία πρέπει να ταιριάξει ο μηχανισμός αναζήτησης. Αυτή τη στιγμή υπάρχει δυνατότητα για τρία κριτήρια : το όνομα του αντιπροσώπου, το αναγνωριστικό του και τη διεύθυνση της αντιπροσωπείας που δημιουργήθηκε. Το όνομα που χρησιμοποιείται για κάθε κριτήριο είναι “agent.name”, “agent.id” και “agent.home” αντίστοιχα. Στο παρακάτω παράδειγμα φαίνεται ο κώδικας για την αναζήτηση ενός αντιπροσώπου, του οποίου το αναγνωριστικό το γνωρίζουμε ήδη, όπως και το όνομά του.

```
//Δημιουργία του φίλτρου αναζήτησης
AgentFilter filter = new AgentFilter();
//Πρώτος περιορισμός: όνομα
filter.setConstraint("agent.name", "Joe Doe");
//Δεύτερος περιορισμός: αναγνωριστικό
//Υποθέτουμε ότι το γνωρίζουμε ήδη και βρίσκεται στη μεταβλητή
//myTargetId
filter.setConstraint("agent.id", myTargetId.getId());

//Αναζήτηση
Result res = search(filter);
if(res == null)
      //Δεν βρέθηκε αντιπρόσωπος
      System.out.println("No results found");
else {
      //Εκτύπωση των αποτελεσμάτων
      Vector v = res.getEntries();
      Enumeration e = v.elements();
      while(e.hasMoreElements()) {
            ResultEntry re = (ResultEntry) e.nextElement();
            System.out.println(re.getAgency() + "\t" + re.getId() +
"\t" + re.getPlace());
      }
}
```

Παράρτημα Γ – API Πλατφόρμας

Γ.1. Package gr.ntua.itsmod.thathan.agency

Class Agency

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agency.Agency
```

All Implemented Interfaces:

IAgency

```
public class Agency
extends java.lang.Object
implements IAgency
```

Field Summary

static java.lang.String	**DEFAULT_NAME**
static java.lang.String	**DEFAULT_PATH**
static java.lang.String	**DEFAULT_PLACE**
static int	**DEFAULT_PROTOCOL**

Constructor Summary

Agency()	

Method Summary

void	**createAgent**(java.lang.String jarFile, java.lang.String className, java.lang.String placeName)
void	**createPlace**(java.lang.String name)
void	**destroy**()
byte[]	**getAgentFile**(java.lang.String jarFile)

AgencyInfo	**getInfo**() Returns the agency's information.
java.lang.String	**getName**() Returns the name of the agency.
java.util.Properties	**getProperties**() Returns the properties of the agency.
java.lang.String	**getProperty**(java.lang.String name) Returns the value of a property.
java.lang.String	**getProperty**(java.lang.String name, java.lang.String dflt) Returns the value of a property or dflt if the property is not set.
java.util.Hashtable	**getStructure**()
SystemInfo	**getSystemInfo**() Returns information about the system the agency is running.
void	**init**(java.util.Properties params)
void	**init**(java.lang.String host)
void	**init**(java.lang.String host, java.lang.String name)
void	**init**(java.lang.String host, java.lang.String name, int protocol)
void	**killAgent**(AgentID id)
java.util.Vector	**listAllAvailableAgencies**()
void	**registerToRegion**(Identifier regionID)
void	**removeAgent**(AgentID id)
void	**removePlace**(java.lang.String name)
void	**requestMove**()
void	**requestRetrieve**(AgentID id, Identifier target_id, java.lang.String place)

	This function is used from remote hosts to request from this agency an agent.
void	**requestTransfer**(AgentID id, Identifier target_id, java.lang.String place) Method used from remote objects to request to transfer a specific agent.
java.lang.String	**retrieveAgent**(Identifier host_id, java.lang.String jarFile, java.lang.String placeName, java.lang.String data)
Result	**search**(AgentFilter filter)
java.lang.String	**sendMessage**(AgentID id, java.lang.String methodName, java.lang.String[] arguments)
void	**setInfo**(AgencyInfo info) Sets the agency's information.
void	**setProperties**(java.util.Properties props) Sets the properties of the agent.
void	**setProperty**(java.lang.String name, java.lang.String value) Sets a property.
void	**start**()
void	**stop**()
void	**unregisterFromRegion**()

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencyInfo

java.lang.Object
  └ **gr.ntua.itsmod.thathan.agency.AgencyInfo**

All Implemented Interfaces:

java.io.Serializable

---

public class **AgencyInfo**
extends java.lang.Object

implements java.io.Serializable

See Also:

Serialized Form

Constructor Summary

AgencyInfo()
AgencyInfo(Identifier id, Identifier region_id, java.lang.String name)
AgencyInfo(Identifier id, java.lang.String name)

Method Summary

Identifier	**getId**()
boolean	**getInitialized**()
java.lang.String	**getName**()
Identifier	**getRegionID**()
boolean	**getRegistered**()
boolean	**isInitialized**()
boolean	**isRegistered**()
void	**setId**(Identifier id)
void	**setInitialized**(boolean initialized)
void	**setName**(java.lang.String name)
void	**setRegionID**(Identifier region_id)
void	**setRegistered**(boolean registered)

java.lang.String	**toString**()

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class AgencyUtil

java.lang.Object
  └ **gr.ntua.itsmod.thathan.agency.AgencyUtil**

public class **AgencyUtil**
extends java.lang.Object

Constructor Summary

AgencyUtil()	

Method Summary

static java.lang.String	**agentToBytes**(AgentBox box)
static byte[]	**getBytesFromFile**(java.io.File file) Returns the contents of the file in a byte array
static java.lang.String	**objectToString**(java.lang.Object obj)
static void	**saveFromURL**(java.net.URL url, java.lang.String destPath)
static java.lang.Object[]	**StringsToObjects**(java.lang.String[] arg)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgentBox

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agency.AgentBox
```

All Implemented Interfaces:

java.io.Serializable

```
public class AgentBox
extends java.lang.Object
implements java.io.Serializable
```

See Also:

Serialized Form

Constructor Summary

AgentBox()
AgentBox(IAgency agency, IAgent agent, AgentID id, AgentInfo info, AgentClassLoader loader)

Method Summary

void	**createAgent**(java.lang.String classFile, java.lang.String placeName)
IAgency	**getAgency**() Returns a reference to the agency the agent is running.
IAgent	**getAgent**() Returns the agent.
AgentID	**getId**() Returns the agent's id.
AgentInfo	**getInfo**() Returns the agent's info
AgentClassLoader	**getLoader**()
State	**getState**() Returns the agent's state object.
AgentThread	**getThread**()
java.lang.ThreadGroup	**getThreads**() Returns the thread group that is responsible for running the agent.
void	**go**()

	Starts the agent.
java.util.Vector	**listAllAvailableAgencies**()
void	**log**(java.lang.Object o)
CallResponse	**makeCall**(Identifier agency_id, AgentID id, Call call) Makes a call to a remote agent.
void	**moveAgent**(Identifier agency_id, java.lang.String place) Moves the agent to a remote agency or another place.
Result	**search**(AgentFilter filter) Searches for agents or agencies matching the criteria described in filter
void	**setAgency**(IAgency agency) Sets a reference to the agency the agency is running.
void	**setAgent**(IAgent agent)
void	**setId**(AgentID id) Sets the agent's id.
void	**setInfo**(AgentInfo info) Sets the agent's info.
void	**setLoader**(AgentClassLoader loader)
void	**setState**(State state)
void	**setThread**(AgentThread thread)
void	**setThreads**(java.lang.ThreadGroup threads)
java.lang.String	**toString**()

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class AgentClassLoader

```
java.lang.Object
  └java.lang.ClassLoader
      └java.security.SecureClassLoader
          └gr.ntua.itsmod.thathan.agency.AgentClassLoader
```

```
public class AgentClassLoader
extends java.security.SecureClassLoader
```

Constructor Summary

AgentClassLoader(java.lang.ClassLoader parent, java.lang.String jarFile)

Method Summary

java.lang.String	**getJarFile**()

Methods inherited from class java.lang.ClassLoader

clearAssertionStatus, getParent, getResource, getResourceAsStream, getResources, getSystemClassLoader, getSystemResource, getSystemResourceAsStream, getSystemResources, loadClass, setClassAssertionStatus, setDefaultAssertionStatus, setPackageAssertionStatus

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgentThread

```
java.lang.Object
  └java.lang.Thread
      └gr.ntua.itsmod.thathan.agency.AgentThread
```

All Implemented Interfaces:

java.lang.Runnable

```
public class AgentThread
extends java.lang.Thread
```

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread
java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler

Field Summary

Fields inherited from class java.lang.Thread
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

AgentThread(java.lang.ThreadGroup group, AgentBox wrapper)	

Method Summary

void	**pleaseStop**()
void	**run**()
void	**setMoved**(boolean val)

Methods inherited from class java.lang.Thread
activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Interface IAgency

All Known Implementing Classes:

Agency

```
public interface IAgency
```

Method Summary

void	**createAgent**(java.lang.String jarFile, java.lang.String className, java.lang.String placeName)
void	**createPlace**(java.lang.String name)
void	**destroy**()
AgencyInfo	**getInfo**() Returns the agency's information.
java.lang.String	**getName**() Returns the name of the agency.
java.util.Properties	**getProperties**() Returns the properties of the agency.
java.lang.String	**getProperty**(java.lang.String name) Returns the value of a property.
java.lang.String	**getProperty**(java.lang.String name, java.lang.String dflt) Returns the value of a property or dflt if the property is not set.
SystemInfo	**getSystemInfo**() Returns information about the system the agency is running.
void	**init**(java.util.Properties params)
void	**init**(java.lang.String host)
void	**init**(java.lang.String host, java.lang.String name, int protocol)
void	**killAgent**(AgentID id)

java.util.Vector	**listAllAvailableAgencies**()
void	**registerToRegion**(Identifier region_id)
void	**removeAgent**(AgentID id)
void	**removePlace**(java.lang.String name)
void	**requestMove**()
void	**requestRetrieve**(AgentID id, Identifier host_id, java.lang.String place)
void	**requestTransfer**(AgentID id, Identifier host_id, java.lang.String place)
java.lang.String	**retrieveAgent**(Identifier host_id, java.lang.String className, java.lang.String placeName, java.lang.String data)
Result	**search**(AgentFilter filter)
java.lang.String	**sendMessage**(AgentID id, java.lang.String methodName, java.lang.String[] arguments)
void	**setInfo**(AgencyInfo info) Sets the agency's information.
void	**setProperties**(java.util.Properties props) Sets the properties of the agent.
void	**setProperty**(java.lang.String name, java.lang.String value) Sets a property.
void	**start**()
void	**stop**()
void	**unregisterFromRegion**()

Class Identifier

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agency.Identifier
```

All Implemented Interfaces:

java.io.Serializable

```
public class Identifier
extends java.lang.Object
implements java.io.Serializable
```

This class provides the data required in order to form the address of an entity of the platform.

See Also:

Serialized Form

Field Summary

static int	**RMI**
static int	**SOCKET**
static int	**WEB_SERVICES_SOAP**

Constructor Summary

Identifier()
Identifier(int protocol, java.lang.String host, int port)
Identifier(int protocol, java.lang.String host, int port, java.lang.String path)

Method Summary

java.lang.String	**getAddress**()
java.lang.String	**getHost**()
java.lang.String	**getPath**()

int	**getPort**()
int	**getProtocol**()
void	**setHost**(java.lang.String host)
void	**setPath**(java.lang.String path)
void	**setPort**(int port)
void	**setProtocol**(int protocol)
java.lang.String	**toString**()

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class InvalidProtocolException

java.lang.Object
  └ java.lang.Throwable
    └ java.lang.Exception
      └ gr.ntua.itsmod.thathan.agent.AgentException
        └

gr.ntua.itsmod.thathan.agency.InvalidProtocolException

All Implemented Interfaces:

java.io.Serializable

public class **InvalidProtocolException**
extends AgentException

See Also:

Serialized Form

Constructor Summary

InvalidProtocolException(java.lang.String msg)	

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class ObjectInputStreamWithLoader

```
java.lang.Object
  └java.io.InputStream
      └java.io.ObjectInputStream
          └gr.ntua.itsmod.thathan.agency.ObjectInputStreamWithLoader
```

All Implemented Interfaces:

java.io.Closeable, java.io.DataInput, java.io.ObjectInput, java.io.ObjectStreamConstants

```
public class ObjectInputStreamWithLoader
extends java.io.ObjectInputStream
```

Nested Class Summary

Nested classes/interfaces inherited from class java.io.ObjectInputStream
java.io.ObjectInputStream.GetField

Field Summary

Fields inherited from interface java.io.ObjectStreamConstants
baseWireHandle, PROTOCOL_VERSION_1, PROTOCOL_VERSION_2, SC_BLOCK_DATA, SC_ENUM, SC_EXTERNALIZABLE, SC_SERIALIZABLE, SC_WRITE_METHOD, STREAM_MAGIC, STREAM_VERSION, SUBCLASS_IMPLEMENTATION_PERMISSION, SUBSTITUTION_PERMISSION, TC_ARRAY, TC_BASE, TC_BLOCKDATA, TC_BLOCKDATALONG, TC_CLASS,

TC_CLASSDESC, TC_ENDBLOCKDATA, TC_ENUM, TC_EXCEPTION, TC_LONGSTRING, TC_MAX, TC_NULL, TC_OBJECT, TC_PROXYCLASSDESC, TC_REFERENCE, TC_RESET, TC_STRING

Constructor Summary

ObjectInputStreamWithLoader(java.io.InputStream inputstream, java.lang.ClassLoader classloader)

Method Summary

Methods inherited from class java.io.ObjectInputStream

available, close, defaultReadObject, read, read, readBoolean, readByte, readChar, readDouble, readFields, readFloat, readFully, readFully, readInt, readLine, readLong, readObject, readShort, readUnshared, readUnsignedByte, readUnsignedShort, readUTF, registerValidation, skipBytes

Methods inherited from class java.io.InputStream

mark, markSupported, read, reset, skip

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface java.io.ObjectInput

read, skip

Class PlaceExistsException

java.lang.Object
  └ java.lang.Throwable

java.lang.Exception
gr.ntua.itsmod.thathan.agent.AgentException
gr.ntua.itsmod.thathan.agency.PlaceExistsException

All Implemented Interfaces:

java.io.Serializable

```
public class PlaceExistsException
extends AgentException
```

See Also:

Serialized Form

Constructor Summary

PlaceExistsException(java.lang.String msg)	

Method Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class RegistrationErrorException

java.lang.Object
java.lang.Throwable
java.lang.Exception
gr.ntua.itsmod.thathan.agent.AgentException
gr.ntua.itsmod.thathan.agency.RegistrationErrorException

All Implemented Interfaces:

java.io.Serializable

```
public class RegistrationErrorException
extends AgentException
```

See Also:

Serialized Form

Constructor Summary

RegistrationErrorException(java.lang.String msg)	

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class SystemInfo

java.lang.Object
  └ **gr.ntua.itsmod.thathan.agency.SystemInfo**

All Implemented Interfaces:

java.io.Serializable

---

public class **SystemInfo**
extends java.lang.Object
implements java.io.Serializable

See Also:

Serialized Form

Constructor Summary

SystemInfo()	

Method Summary

java.lang.String	**getJavaVersion**()
java.lang.String	**getOSName**()
java.lang.String	**getOSVersion**()

java.lang.String	**getVMVersion**()
void	**setJavaVersion**(java.lang.String javaVersion)
void	**setOSName**(java.lang.String oSName)
void	**setOSVersion**(java.lang.String oSVersion)
void	**setVMVersion**(java.lang.String vMVersion)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class TAgentBox

java.lang.Object
  └ **gr.ntua.itsmod.thathan.agency.TAgentBox**

All Implemented Interfaces:

java.io.Serializable

```
public class TAgentBox
extends java.lang.Object
implements java.io.Serializable
```

See Also:

Serialized Form

Constructor Summary

TAgentBox()	
TAgentBox(AgentID id, AgentInfo info)	

Method Summary

AgentID	**getId**()
AgentInfo	**getInfo**()

State	**getState**()
void	**setId**(AgentID id)
void	**setInfo**(AgentInfo info)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Γ.2. Package gr.ntua.itsmod.thathan.agent

Class Agent

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agent.Agent
```

All Implemented Interfaces:

IAgent, java.io.Serializable

Direct Known Subclasses:

MobileAgent, StationaryAgent

```
public abstract class Agent
extends java.lang.Object
implements IAgent
```

See Also:

Serialized Form

Constructor Summary

Agent()	

Method Summary

void	**createAgent**(java.lang.String className, java.lang.String placeName)
abstract void	**destroy**() Destroys the agent.
AgentID	**getAgentID**()

AgentInfo	**getInfo**() Returns the agent's information object.
java.lang.String	**getName**() Returns the name of the agent.
java.util.Properties	**getProperties**() Returns the properties of the agent.
java.lang.String	**getProperty**(java.lang.String name) Returns the value of a property.
java.lang.String	**getProperty**(java.lang.String name, java.lang.String dflt) Returns the value of a property or dflt if the property is not set.
int	**getType**()
AgentBox	**getWrapper**() Returns the wrapper
abstract void	**init**() Initializes the agent
CallResponse	**makeCall**(Identifier agency_id, AgentID id, Call call)
Result	**search**(AgentFilter filter)
void	**setInfo**(AgentInfo info) Sets the agent's info
void	**setProperties**(java.util.Properties props) Sets the properties of the agent.
void	**setProperty**(java.lang.String name, java.lang.String value) Sets a property.
void	**setWrapper**(AgentBox wrapper) Sets the wrapper
abstract void	**start**() Starts the agent.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgentException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └gr.ntua.itsmod.thathan.agent.AgentException
```

All Implemented Interfaces:

java.io.Serializable

Direct Known Subclasses:

AlreadyRegisteredException, InvalidProtocolException, InvalidStateException, MoveException, PlaceExistsException, RegionInitializationException, RegistrationErrorException, ResultNotReadyException

---

```
public class AgentException
extends java.lang.Exception
```

Generic exception for this platform. Every exception in this platform is a subclass of this class.

See Also:

Serialized Form

Constructor Summary

AgentException(java.lang.String msg)	

Method Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class AgentID

```
java.lang.Object
  └gr.ntua.itsmod.thathan.agent.AgentID
```

All Implemented Interfaces:

java.io.Serializable

public class **AgentID**
extends java.lang.Object
implements java.io.Serializable

See Also:

Serialized Form

Constructor Summary

AgentID()	
AgentID(AgentID agentID)	
AgentID(java.lang.String id)	

Method Summary

java.lang.String	**getId**()
void	**setId**(java.lang.String id)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgentInfo

java.lang.Object
  └ **gr.ntua.itsmod.thathan.agent.AgentInfo**

All Implemented Interfaces:

java.io.Serializable

public class **AgentInfo**
extends java.lang.Object
implements java.io.Serializable

See Also:

Serialized Form

Field Summary

static int	**MOBILE**

static int	**STATIONARY**
static int	**UNKNOWN**

Constructor Summary

AgentInfo()	

Method Summary

java.lang.String	**getHomeHost**()
java.lang.String	**getHomePlace**()
AgentID	**getId**()
java.lang.String	**getLastHost**()
java.lang.String	**getLastPlace**()
java.lang.String	**getName**()
int	**getType**()
boolean	**isActive**()
void	**setActive**(boolean active)
void	**setHomeHost**(java.lang.String homeHost)
void	**setHomePlace**(java.lang.String homePlace)
void	**setId**(AgentID id)
void	**setLastHost**(java.lang.String lastHost)

void	**setLastPlace**(java.lang.String lastPlace)
void	**setName**(java.lang.String name)
void	**setType**(int type)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Interface IAgent

All Superinterfaces:
java.io.Serializable

All Known Subinterfaces:
IMobileAgent, IStationaryAgent

All Known Implementing Classes:
Agent, MobileAgent, SimpleAgent, StationaryAgent

```
public interface IAgent
extends java.io.Serializable
```

Method Summary

void	**destroy**() Destroys the agent.
AgentInfo	**getInfo**() Returns the agent's information object.
java.lang.String	**getName**() Returns the name of the agent.
java.util.Properties	**getProperties**() Returns the properties of the agent.
java.lang.String	**getProperty**(java.lang.String name) Returns the value of a property.
java.lang.String	**getProperty**(java.lang.String name, java.lang.String dflt) Returns the value of a property or dflt if the property is not set.
AgentBox	**getWrapper**() Returns the wrapper

void	**init**() Initializes the agent
void	**setInfo**(AgentInfo info) Sets the agent's info
void	**setProperties**(java.util.Properties props) Sets the properties of the agent.
void	**setProperty**(java.lang.String name, java.lang.String value) Sets a property.
void	**setWrapper**(AgentBox wrapper) Sets the wrapper
void	**start**() Starts the agent.

Interface IMobileAgent

All Superinterfaces:

IAgent, java.io.Serializable

All Known Implementing Classes:

MobileAgent, SimpleAgent

```
public interface IMobileAgent
extends IAgent
```

Marker interface for stationary agents. The purpose of this interface is to provide an additional way of finding the type of an agent, this time using reflection.

Method Summary

Methods inherited from interface gr.ntua.itsmod.thathan.agent.IAgent
destroy, getInfo, getName, getProperties, getProperty, getProperty, getWrapper, init, setInfo, setProperties, setProperty, setWrapper, start

Class InvalidStateException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └gr.ntua.itsmod.thathan.agent.AgentException
              └gr.ntua.itsmod.thathan.agent.InvalidStateException
```

All Implemented Interfaces:

java.io.Serializable

```
public class InvalidStateException
```

extends AgentException

This exception is thrown when an invalid state is attempted to be assigned to an agent.

See Also:

Serialized Form

Constructor Summary

InvalidStateException(int state)	

Method Summary

int	**getState**()

Methods inherited from class java.lang.Throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Interface IStationaryAgent

All Superinterfaces:

IAgent, java.io.Serializable

All Known Implementing Classes:

StationaryAgent

public interface **IStationaryAgent**
extends IAgent

Marker interface for stationary agents. The purpose of this interface is to provide an additional way of finding the type of an agent, this time using reflection.

Method Summary

Methods inherited from interface gr.ntua.itsmod.thathan.agent.IAgent
destroy, getInfo, getName, getProperties, getProperty, getProperty, getWrapper, init, setInfo, setProperties, setProperty, setWrapper, start

Class MobileAgent

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agent.Agent
      └ gr.ntua.itsmod.thathan.agent.MobileAgent
```

All Implemented Interfaces:

IAgent, IMobileAgent, java.io.Serializable

Direct Known Subclasses:

SimpleAgent

```
public abstract class MobileAgent
extends Agent
implements IMobileAgent
```

See Also:

Serialized Form

Constructor Summary

MobileAgent()	

Method Summary

abstract void	**afterMove**()
abstract void	**beforeMove**()
int	**getType**()
void	**move**(Identifier agency_id, java.lang.String place)

Methods inherited from class gr.ntua.itsmod.thathan.agent.Agent
createAgent, destroy, getAgentID, getInfo, getName, getProperties, getProperty, getProperty, getWrapper, init, makeCall, search, setInfo, setProperties, setProperty, setWrapper, start

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface gr.ntua.itsmod.thathan.agent.IAgent
destroy, getInfo, getName, getProperties, getProperty, getProperty, getWrapper, init, setInfo, setProperties, setProperty, setWrapper, start

Class MoveException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └gr.ntua.itsmod.thathan.agent.AgentException
              └gr.ntua.itsmod.thathan.agent.MoveException
```

All Implemented Interfaces:

java.io.Serializable

```
public class MoveException
extends AgentException
```

This exception is thrown when an agent fails to move.

See Also:

Serialized Form

Constructor Summary

MoveException(java.lang.String msg)	

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class State

java.lang.Object
  └ **gr.ntua.itsmod.thathan.agent.State**

All Implemented Interfaces:

java.io.Serializable

---

public class **State**
extends java.lang.Object
implements java.io.Serializable

See Also:

Serialized Form

Field Summary

static int	**CREATED**
static int	**CREATING**
static int	**DESTROYED**
static int	**DESTROYING**
static int	**INITIALIZED**
static int	**INITIALIZING**
static int	**NONE**
static int	**STOPED**
static int	**STOPING**
static int	**WORKING**

Constructor Summary

State()	

State(int state)	
State(State state)	

Method Summary

int	**getState**()
void	**setState**(int state)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class StationaryAgent

java.lang.Object
  └ gr.ntua.itsmod.thathan.agent.Agent
    └ **gr.ntua.itsmod.thathan.agent.StationaryAgent**

All Implemented Interfaces:

IAgent, IStationaryAgent, java.io.Serializable

```
public abstract class StationaryAgent
extends Agent
implements IStationaryAgent
```

Abstract class for stationary agents. Subclass this class if you want to create a new stationary agent.

See Also:

Serialized Form

Constructor Summary

StationaryAgent()	

Method Summary

int	**getType**()

Methods inherited from class gr.ntua.itsmod.thathan.agent.Agent

createAgent, destroy, getAgentID, getInfo, getName, getProperties, getProperty, getProperty, getWrapper, init, makeCall, search, setInfo, setProperties, setProperty, setWrapper, start

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface gr.ntua.itsmod.thathan.agent.IAgent

destroy, getInfo, getName, getProperties, getProperty, getProperty, getWrapper, init, setInfo, setProperties, setProperty, setWrapper, start

Γ.3. Package gr.ntua.itsmod.thathan.agent.helpers

Class AFilter

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agent.helpers.AFilter
```

All Implemented Interfaces:

IFilter, java.io.Serializable

```
public abstract class AFilter
extends java.lang.Object
implements IFilter
```

See Also:

Serialized Form

Constructor Summary

AFilter()	

Method Summary

java.lang.String	**getConstraint**()

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface gr.ntua.itsmod.thathan.agent.helpers.IFilter
getConstraint

Class AgentFilter

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agent.helpers.Filter
      └ gr.ntua.itsmod.thathan.agent.helpers.AgentFilter
```

All Implemented Interfaces:

IFilter, java.io.Serializable

```
public class AgentFilter
extends Filter
```

This class defines a search filter for agents.

See Also:

Serialized Form

Field Summary

static java.lang.String	**CLASS**
static java.lang.String	**HOME**
static java.lang.String	**LOCATION**
static java.lang.String	**NAME**
static java.lang.String	**STATE**
static java.lang.String	**TYPE**

Constructor Summary

AgentFilter()	

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.agent.helpers.Filter

getConstraint_id, getConstraint, getConstraints, setConstraint, setConstraint, setConstraints

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class Filter

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agent.helpers.Filter
```

All Implemented Interfaces:

IFilter, java.io.Serializable

Direct Known Subclasses:

AgentFilter

```
public abstract class Filter
extends java.lang.Object
implements IFilter
```

See Also:

Serialized Form

Constructor Summary

Filter()	

Method Summary

Identifier	**getConstraint_id**(java.lang.String name)

java.lang.String	**getConstraint**(java.lang.String name)
java.util.Hashtable	**getConstraints**()
void	**setConstraint**(java.lang.String name, Identifier id)
void	**setConstraint**(java.lang.String name, java.lang.String value)
void	**setConstraints**(java.util.Hashtable constraints)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Interface IFilter

All Superinterfaces:

java.io.Serializable

All Known Implementing Classes:

AFilter, AgentFilter, Filter

```
public interface IFilter
extends java.io.Serializable
```

Method Summary

java.lang.String	**getConstraint**(java.lang.String name)

Class Result

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agent.helpers.Result
```

All Implemented Interfaces:

java.io.Serializable

```
public class Result
extends java.lang.Object
implements java.io.Serializable
```

See Also:

Serialized Form

Constructor Summary

Result()	

Method Summary

void	**addEntry**(ResultEntry entry)
java.util.Vector	**getEntries**()
void	**removeEntry**(ResultEntry entry)
void	**setEntries**(java.util.Vector entries)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class ResultEntry

java.lang.Object
  └ **gr.ntua.itsmod.thathan.agent.helpers.ResultEntry**

All Implemented Interfaces:

java.io.Serializable

public class **ResultEntry**
extends java.lang.Object
implements java.io.Serializable

See Also:

Serialized Form

Constructor Summary

ResultEntry()
ResultEntry(AgentID id, Identifier agency_id, java.lang.String place)

Method Summary

Identifier	**getAgency**()
AgentID	**getId**()
java.lang.String	**getPlace**()
void	**setAgency**(Identifier agency_id)
void	**setId**(AgentID id)
void	**setPlace**(java.lang.String place)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Γ.4. Package gr.ntua.itsmod.thathan.agent.rpc

Class AsynchCall

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agent.rpc.Call
      └ gr.ntua.itsmod.thathan.agent.rpc.AsynchCall
```

All Implemented Interfaces:

ICall, java.io.Serializable

```
public class AsynchCall
extends Call
```

Object representing asynchronous calls to remote agents.

See Also:

Serialized Form

Constructor Summary

AsynchCall()
AsynchCall(java.lang.String methodName, java.lang.Object[] args)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.agent.rpc.Call
getArgs, getMethodName, setArgs, setMethodName

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class Call

java.lang.Object
  └ **gr.ntua.itsmod.thathan.agent.rpc.Call**

All Implemented Interfaces:

ICall, java.io.Serializable

Direct Known Subclasses:

AsynchCall, CallBackCall, SynchCall

```
public abstract class Call
extends java.lang.Object
implements ICall
```

Simple implementation of the `ICall` interface. The subclasses of this abstract class are used to represent the three methods that the agent platform uses for agent-to-agent communication.

See Also:

Serialized Form

Constructor Summary

Call()
Call(Call other)
Call(java.lang.String methodName, java.lang.Object[] args)

Method Summary

java.lang.Object[]	**getArgs**()

java.lang.String	**getMethodName**()
void	**setArgs**(java.lang.Object[] args)
void	**setMethodName**(java.lang.String methodName)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class CallBackCall

java.lang.Object
  └ gr.ntua.itsmod.thathan.agent.rpc.Call
    └ **gr.ntua.itsmod.thathan.agent.rpc.CallBackCall**

All Implemented Interfaces:

ICall, java.io.Serializable

---

public class **CallBackCall**
extends Call

Object representing call back calls to remote agents.

See Also:

Serialized Form

Constructor Summary

CallBackCall()
CallBackCall(java.lang.String methodName, java.lang.Object[] args, java.lang.String callBackMethod)

Method Summary

java.lang.String	**getCallBackMethod**()
void	**setCallBackMethod**(java.lang.String callBackMethod)

Methods inherited from class gr.ntua.itsmod.thathan.agent.rpc.Call
getArgs, getMethodName, setArgs, setMethodName

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class CallResponse

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agent.rpc.CallResponse
```

All Implemented Interfaces:

java.io.Serializable

```
public final class CallResponse
extends java.lang.Object
implements java.io.Serializable
```

See Also:

Serialized Form

Constructor Summary

CallResponse(Identifier agency_id, AgentID agent_id, Call call)	
CallResponse(Identifier agency_id, AgentID agent_id, IAgent agent, CallBackCall call)	

Method Summary

java.lang.Object	**getResponse**()
boolean	**hasFinished**()

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Interface ICall

All Superinterfaces:

java.io.Serializable

All Known Implementing Classes:

AsynchCall, Call, CallBackCall, SynchCall

```
public interface ICall
extends java.io.Serializable
```

Method Summary

java.lang.Object[]	**getArgs**()
java.lang.String	**getMethodName**()
void	**setArgs**(java.lang.Object[] args)
void	**setMethodName**(java.lang.String methodName)

Class SynchCall

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.agent.rpc.Call
      └ gr.ntua.itsmod.thathan.agent.rpc.SynchCall
```

All Implemented Interfaces:

ICall, java.io.Serializable

```
public class SynchCall
extends Call
```

Object representing synchronous calls to remote agents.

See Also:

Serialized Form

Constructor Summary

SynchCall()
SynchCall(java.lang.String methodName, java.lang.Object[] args)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.agent.rpc.Call
getArgs, getMethodName, setArgs, setMethodName

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Γ.5. Package gr.ntua.itsmod.thathan.communication

Interface IWorker

All Known Implementing Classes:

AgencyInfoWorker, AgencyInfoWorker, AgencyInitWorker, AgencyInitWorker, AgencyNameWorker, AgencyNameWorker, AgencyRegisterToRegionWorker, AgencyRegisterToRegionWorker, AgencyStructureWorker, AgencyStructureWorker, AgencySystemInfoWorker, AgencySystemInfoWorker, CreateAgentWorker, CreateAgentWorker, CreateAgentWorker, CreateAgentWorker, CreatePlaceWorker, CreatePlaceWorker, KillAgentWorker, KillAgentWorker, ListAgenciesWorker, ListAgenciesWorker, MoveAgentWorker, MoveAgentWorker, NewPlaceWorker, NewPlaceWorker, RegionInfoWorker, RegionInfoWorker, RegionStructureWorker, RegionStructureWorker, RegionSystemInfoWorker, RegionSystemInfoWorker, RegisterAgencyWorker, RegisterAgencyWorker, RequestTransferWorker, RequestTransferWorker, RetrieveAgentWorker, RetrieveAgentWorker, RetrieveJarFileWorker, SearchWorker, SearchWorker, SendMessageWorker, SendMessageWorker, SocketWorker, Worker

```
public interface IWorker
```

Main interface for Workers. Workers are small classes that perform specific remote tasks.

Method Summary

boolean	**generatedError**()
java.lang.Object	**getResult**()
boolean	**isResultReady**()

Class ResultNotReadyException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └gr.ntua.itsmod.thathan.agent.AgentException
              └
```

gr.ntua.itsmod.thathan.communication.ResultNotReadyException

All Implemented Interfaces:

java.io.Serializable

```
public class ResultNotReadyException
extends AgentException
```

See Also:

Serialized Form

Constructor Summary

ResultNotReadyException(java.lang.String msg)

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class Worker

```
java.lang.Object
  └gr.ntua.itsmod.thathan.communication.Worker
```

All Implemented Interfaces:

IWorker

Direct Known Subclasses:

AgencyInfoWorker, AgencyInitWorker, AgencyNameWorker, AgencyRegisterToRegionWorker, AgencyStructureWorker, AgencySystemInfoWorker, CreateAgentWorker, CreateAgentWorker, CreatePlaceWorker, KillAgentWorker, ListAgenciesWorker, MoveAgentWorker, NewPlaceWorker, RegionInfoWorker, RegionStructureWorker, RegionSystemInfoWorker, RegisterAgencyWorker,

RequestTransferWorker, RetrieveAgentWorker, SearchWorker, SendMessageWorker, SocketWorker

```
public abstract class Worker
extends java.lang.Object
implements IWorker
```

Abstract class that every Worker must extend.

Constructor Summary

Worker(java.lang.Object target)	

Method Summary

boolean	**generatedError**()
java.lang.Object	**getResult**()
boolean	**isResultReady**()

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class WorkerFactory

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.WorkerFactory
```

```
public class WorkerFactory
extends java.lang.Object
```

Field Summary

static int	**AGENCY_CREATE_AGENT**
static int	**AGENCY_CREATE_PLACE**
static int	**AGENCY_INFO**

static int	**AGENCY_INIT**
static int	**AGENCY_NAME**
static int	**AGENCY_REGISTER_TO_REGION**
static int	**AGENCY_STRUCTURE**
static int	**AGENCY_SYSTEM_INFO**
static int	**KILL_AGENT**
static int	**LIST_AGENCIES**
static int	**MOVE_AGENT**
static int	**REGION_CREATE_AGENT**
static int	**REGION_INFO**
static int	**REGION_NEW_PLACE**
static int	**REGION_STRUCTURE**
static int	**REGION_SYSTEM_INFO**
static int	**REGISTER_AGENCY**
static int	**REQUEST_TRANSFER**
static int	**RETRIEVE_AGENT**
static int	**RETRIEVE_JARFILE**
static int	**SEARCH**
static int	**SEND_MESSAGE**

Constructor Summary

WorkerFactory()	

Method Summary

static Worker	**getWorker**(Identifier target, java.lang.Object[] args, int kindOfWorker) Returns the right kind of Worker according to protocol and the kind of worker needed.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Γ.6. Package gr.ntua.itsmod.thathan.communication.soap.agency

Class AgencyInfoWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └

gr.ntua.itsmod.thathan.communication.soap.agency.AgencyInfoWorker

All Implemented Interfaces:

IWorker

public class **AgencyInfoWorker**
extends Worker

Constructor Summary

AgencyInfoWorker(java.net.URL url)	

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker

generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencyInitWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
```

gr.ntua.itsmod.thathan.communication.soap.agency.AgencyInitWorker

All Implemented Interfaces:

IWorker

```
public class AgencyInitWorker
extends Worker
```

Constructor Summary

AgencyInitWorker(java.net.URL url, java.lang.String hostAddr)	
AgencyInitWorker(java.net.URL url, java.lang.String hostAddr, java.lang.String name)	

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencyNameWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
```

gr.ntua.itsmod.thathan.communication.soap.agency.AgencyNameWorker

All Implemented Interfaces:

IWorker

public class **AgencyNameWorker**
extends Worker

Constructor Summary

AgencyNameWorker(java.net.URL url)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencyRegisterToRegionWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └

gr.ntua.itsmod.thathan.communication.soap.agency.AgencyRegisterToRegionWorker

All Implemented Interfaces:

IWorker

public class **AgencyRegisterToRegionWorker**
extends Worker

Constructor Summary

AgencyRegisterToRegionWorker(java.net.URL url, Identifier regionID)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencyStructureWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
```

gr.ntua.itsmod.thathan.communication.soap.agency.AgencyStructureWorker

All Implemented Interfaces:

IWorker

```
public class AgencyStructureWorker
extends Worker
```

Constructor Summary

AgencyStructureWorker(java.net.URL url)	

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencySystemInfoWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
```

gr.ntua.itsmod.thathan.communication.soap.agency.AgencySystemInfoWorker

All Implemented Interfaces:

IWorker

```
public class AgencySystemInfoWorker
extends Worker
```

Constructor Summary

AgencySystemInfoWorker(java.net.URL url)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker

generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class CreateAgentWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
```

gr.ntua.itsmod.thathan.communication.soap.agency.CreateAgentWorker

All Implemented Interfaces:

IWorker

```
public class CreateAgentWorker
extends Worker
```

Constructor Summary

CreateAgentWorker(java.net.URL url, java.lang.String jarFile, java.lang.String classname, java.lang.String placeName)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class CreatePlaceWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └

gr.ntua.itsmod.thathan.communication.soap.agency.CreatePlaceWorker

All Implemented Interfaces:

IWorker

public class **CreatePlaceWorker**
extends Worker

Constructor Summary

CreatePlaceWorker(java.net.URL url, java.lang.String name)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class KillAgentWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └

gr.ntua.itsmod.thathan.communication.soap.agency.KillAgentWorker

All Implemented Interfaces:

IWorker

public class **KillAgentWorker**
extends Worker

Constructor Summary

KillAgentWorker(java.net.URL url, AgentID id)	

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RequestTransferWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └

gr.ntua.itsmod.thathan.communication.soap.agency.RequestTransferWorker

All Implemented Interfaces:

IWorker

public class **RequestTransferWorker**
extends Worker

Constructor Summary

RequestTransferWorker(java.net.URL url, AgentID id, Identifier host, java.lang.String place)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RetrieveAgentWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └

gr.ntua.itsmod.thathan.communication.soap.agency.RetrieveAgentWorker

All Implemented Interfaces:

IWorker

public class **RetrieveAgentWorker**
extends Worker

Constructor Summary

RetrieveAgentWorker(java.net.URL url, Identifier host, java.lang.String jarFile, java.lang.String placeName, java.lang.String data)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class SendMessageWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
```

gr.ntua.itsmod.thathan.communication.soap.agency.SendMessageWorker

All Implemented Interfaces:

IWorker

```
public class SendMessageWorker
extends Worker
```

Constructor Summary

SendMessageWorker(java.net.URL url, AgentID id, Call call)	

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Γ.7. Package gr.ntua.itsmod.thathan.communication.soap.region

Class CreateAgentWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
```

gr.ntua.itsmod.thathan.communication.soap.region.CreateAgentWorker

All Implemented Interfaces:

IWorker

```
public class CreateAgentWorker
extends Worker
```

Constructor Summary

CreateAgentWorker(java.net.URL url, Identifier agency, java.lang.String place, TAgentBox info)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class ListAgenciesWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
```

gr.ntua.itsmod.thathan.communication.soap.region.ListAgenciesWorker

All Implemented Interfaces:

IWorker

```
public class ListAgenciesWorker
extends Worker
```

Constructor Summary

ListAgenciesWorker(java.net.URL url)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker

generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class MoveAgentWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └

gr.ntua.itsmod.thathan.communication.soap.region.MoveAgentWorker

All Implemented Interfaces:

IWorker

public class **MoveAgentWorker**
extends Worker

Constructor Summary

MoveAgentWorker(java.net.URL url, Identifier origin, java.lang.String origin_place, Identifier my_target, java.lang.String target_place)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker

generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class NewPlaceWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └

gr.ntua.itsmod.thathan.communication.soap.region.NewPlaceWorker

All Implemented Interfaces:

IWorker

public class **NewPlaceWorker**
extends Worker

Constructor Summary

NewPlaceWorker(java.net.URL url, Identifier agency, java.lang.String name)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegionInfoWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
```

gr.ntua.itsmod.thathan.communication.soap.region.RegionInfoWorker

All Implemented Interfaces:

IWorker

```
public class RegionInfoWorker
extends Worker
```

Constructor Summary

RegionInfoWorker(java.net.URL url)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegionStructureWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
```

gr.ntua.itsmod.thathan.communication.soap.region.RegionStructureWorker

All Implemented Interfaces:

IWorker

```
public class RegionStructureWorker
extends Worker
```

Constructor Summary

RegionStructureWorker(java.net.URL url)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegionSystemInfoWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
```

gr.ntua.itsmod.thathan.communication.soap.region.RegionSystemInfoWorker

All Implemented Interfaces:

IWorker

```
public class RegionSystemInfoWorker
extends Worker
```

Constructor Summary

RegionSystemInfoWorker(java.net.URL url)	

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegisterAgencyWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └
gr.ntua.itsmod.thathan.communication.soap.region.RegisterAgencyWorker
```

All Implemented Interfaces:

IWorker

```
public class RegisterAgencyWorker
extends Worker
```

Constructor Summary

RegisterAgencyWorker(java.net.URL url, Identifier agency)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class SearchWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.soap.region.SearchWorker
```

All Implemented Interfaces:

IWorker

```
public class SearchWorker
extends Worker
```

Constructor Summary

SearchWorker(java.net.URL url, AgentFilter filter)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Γ.8. Package gr.ntua.itsmod.thathan.communication.sockets

Class AgencyCommunicationRequest

java.lang.Object
  ∟
gr.ntua.itsmod.thathan.communication.sockets.AgencyCommunicationRequest

All Implemented Interfaces:

java.lang.Runnable

```
public class AgencyCommunicationRequest
extends java.lang.Object
implements java.lang.Runnable
```

Constructor Summary

AgencyCommunicationRequest(java.net.Socket sock, Agency agency)	

Method Summary

void	**processRequest**()
void	**run**()

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencyServer

java.lang.Object
  └ **gr.ntua.itsmod.thathan.communication.sockets.AgencyServer**

public class **AgencyServer**
extends java.lang.Object

Field Summary

static int	**DEFAULT_PORT**

Constructor Summary

AgencyServer(int port, Agency agency)	

Method Summary

static void	**main**(java.lang.String[] args)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegionCommunicationRequest

java.lang.Object
  └
gr.ntua.itsmod.thathan.communication.sockets.RegionCommunicationRequest

All Implemented Interfaces:

java.lang.Runnable

public class **RegionCommunicationRequest**
extends java.lang.Object
implements java.lang.Runnable

Constructor Summary

RegionCommunicationRequest(java.net.Socket sock, Region region)	

Method Summary

void	**processRequest**()
void	**run**()

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegionServer

java.lang.Object
  └ **gr.ntua.itsmod.thathan.communication.sockets.RegionServer**

public class **RegionServer**
extends java.lang.Object

Field Summary

static int	**DEFAULT_PORT**

Constructor Summary

RegionServer(int port, Region region)	

Method Summary

static void	**main**(java.lang.String[] args)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class SocketWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
```

All Implemented Interfaces:

IWorker

Direct Known Subclasses:

AgencyInfoWorker, AgencyInitWorker, AgencyNameWorker, AgencyRegisterToRegionWorker, AgencyStructureWorker, AgencySystemInfoWorker, CreateAgentWorker, CreateAgentWorker, CreatePlaceWorker, KillAgentWorker, ListAgenciesWorker, MoveAgentWorker, NewPlaceWorker, RegionInfoWorker, RegionStructureWorker, RegionSystemInfoWorker, RegisterAgencyWorker, RequestTransferWorker, RetrieveAgentWorker, RetrieveJarFileWorker, SearchWorker, SendMessageWorker

```
public class SocketWorker
extends Worker
```

Constructor Summary

SocketWorker(Identifier id)	

Method Summary

void	**communicate**()

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker

generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Γ.9. Package gr.ntua.itsmod.thathan.communication.sockets.agency

Class AgencyInfoWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
```

└

gr.ntua.itsmod.thathan.communication.sockets.agency.AgencyInfoWorker

All Implemented Interfaces:

IWorker

public class **AgencyInfoWorker**
extends SocketWorker

Constructor Summary

AgencyInfoWorker(Identifier id)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker

communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker

generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencyInitWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
      └

gr.ntua.itsmod.thathan.communication.sockets.agency.AgencyInitWorker

All Implemented Interfaces:

IWorker

public class **AgencyInitWorker**
extends SocketWorker

Constructor Summary

AgencyInitWorker(Identifier id, java.lang.String name)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencyNameWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
      └

gr.ntua.itsmod.thathan.communication.sockets.agency.AgencyNameWorker

All Implemented Interfaces:

IWorker

public class **AgencyNameWorker**
extends SocketWorker

Constructor Summary

AgencyNameWorker(Identifier id)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencyRegisterToRegionWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
          └
```

gr.ntua.itsmod.thathan.communication.sockets.agency.AgencyRegisterToRegionWorker

All Implemented Interfaces:

IWorker

```
public class AgencyRegisterToRegionWorker
extends SocketWorker
```

Constructor Summary

AgencyRegisterToRegionWorker(Identifier id, Identifier region_id)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencyStructureWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
          └
```

gr.ntua.itsmod.thathan.communication.sockets.agency.AgencyStructureWorker

All Implemented Interfaces:

IWorker

```
public class AgencyStructureWorker
extends SocketWorker
```

Constructor Summary

AgencyStructureWorker(Identifier id)	

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AgencySystemInfoWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
      └

gr.ntua.itsmod.thathan.communication.sockets.agency.AgencySystemInfoWorker

All Implemented Interfaces:

IWorker

public class **AgencySystemInfoWorker**
extends SocketWorker

Constructor Summary

AgencySystemInfoWorker(Identifier id)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker

communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker

generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class CreateAgentWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker

└ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
└

gr.ntua.itsmod.thathan.communication.sockets.agency.CreateAgentWorker

All Implemented Interfaces:

IWorker

public class **CreateAgentWorker**
extends SocketWorker

Constructor Summary

CreateAgentWorker(Identifier id, java.lang.String jarFile, java.lang.String className, java.lang.String placeName)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker

communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker

generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class CreatePlaceWorker

java.lang.Object
└ gr.ntua.itsmod.thathan.communication.Worker
└ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
└

gr.ntua.itsmod.thathan.communication.sockets.agency.CreatePlaceWorker

All Implemented Interfaces:

IWorker

public class **CreatePlaceWorker**
extends SocketWorker

Constructor Summary

CreatePlaceWorker(Identifier id, java.lang.String name)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker

communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker

generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class KillAgentWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
      └

gr.ntua.itsmod.thathan.communication.sockets.agency.KillAgentWorker

All Implemented Interfaces:

IWorker

public class **KillAgentWorker**
extends SocketWorker

Constructor Summary

KillAgentWorker(Identifier agency_id, AgentID agent_id)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RequestTransferWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
          └
```

gr.ntua.itsmod.thathan.communication.sockets.agency.RequestTransferWorker

All Implemented Interfaces:

IWorker

```
public class RequestTransferWorker
extends SocketWorker
```

Constructor Summary

RequestTransferWorker(Identifier agency_id, AgentID agent_id, Identifier target_agency_id, java.lang.String place)	

Method Summary

Methods inherited from class

gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RetrieveAgentWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
          └
```

gr.ntua.itsmod.thathan.communication.sockets.agency.RetrieveAgentWorker

All Implemented Interfaces:

IWorker

```
public class RetrieveAgentWorker
extends SocketWorker
```

Constructor Summary

RetrieveAgentWorker(Identifier id, Identifier host_id, java.lang.String jarFile, java.lang.String placeName, java.lang.String data)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RetrieveJarFileWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
          └
```

gr.ntua.itsmod.thathan.communication.sockets.agency.RetrieveJarFileWorker

All Implemented Interfaces:

IWorker

```
public class RetrieveJarFileWorker
extends SocketWorker
```

Constructor Summary

RetrieveJarFileWorker(Identifier id, java.lang.String jarFile)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class SendMessageWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
      └

gr.ntua.itsmod.thathan.communication.sockets.agency.SendMessageWorker

All Implemented Interfaces:

IWorker

public class **SendMessageWorker**
extends SocketWorker

Constructor Summary

SendMessageWorker(Identifier id, AgentID agent_id, Call call)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Γ.10. Package gr.ntua.itsmod.thathan.communication.sockets.message

Interface IMessage

All Superinterfaces:

java.io.Serializable

All Known Implementing Classes:

Message, ReplyMessage, RequestMessage

```
public interface IMessage
extends java.io.Serializable
```

Method Summary

void	**addArg**(java.lang.Object Obj)
java.util.Vector	**getArgs**()
int	**getMessageType**()
void	**setMessageType**(int messageType)

Class Message

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.sockets.message.Message
```

All Implemented Interfaces:

IMessage, java.io.Serializable

Direct Known Subclasses:

ReplyMessage, RequestMessage

```
public class Message
extends java.lang.Object
implements IMessage
```

See Also:

Serialized Form

Field Summary

static int	**AGENCY_INFO**
static int	**AGENCY_INIT**
static int	**AGENCY_NAME**
static int	**AGENCY_REGISTER_TO_REGION**
static int	**AGENCY_STRUCTURE**
static int	**AGENCY_SYSTEM_INFO**
static int	**CREATE_AGENT**
static int	**CREATE_PLACE**

static int	KILL_AGENT
static int	LIST_AGENCIES
int	messageType
static int	MOVE_AGENT
static int	NEW_PLACE
java.util.Vector	params
static int	REGION_CREATE_AGENT
static int	REGION_INFO
static int	REGION_STRUCTURE
static int	REGION_SYSTEM_INFO
static int	REGISTER_AGENCY
static int	REQUEST_TRANSFER
static int	RETRIEVE_AGENT
static int	RETRIEVE_JAR
static int	SEARCH
static int	SEND_MESSAGE
static int	UNKNOWN

Constructor Summary

Message()	
Message(int messageType)	

Method Summary

void	**addArg**(java.lang.Object obj)
void	**addArg**(java.util.Vector params)
java.util.Vector	**getArgs**()
int	**getMessageType**()
void	**setMessageType**(int messageType)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class ReplyMessage

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.sockets.message.Message
    └

gr.ntua.itsmod.thathan.communication.sockets.message.ReplyMessage

All Implemented Interfaces:

IMessage, java.io.Serializable

public class **ReplyMessage**
extends Message

See Also:

Serialized Form

Field Summary

Fields inherited from class

gr.ntua.itsmod.thathan.communication.sockets.message.Message
AGENCY_INFO, AGENCY_INIT, AGENCY_NAME, AGENCY_REGISTER_TO_REGION, AGENCY_STRUCTURE, AGENCY_SYSTEM_INFO, CREATE_AGENT, CREATE_PLACE, KILL_AGENT, LIST_AGENCIES, messageType, MOVE_AGENT, NEW_PLACE, params, REGION_CREATE_AGENT, REGION_INFO, REGION_STRUCTURE, REGION_SYSTEM_INFO, REGISTER_AGENCY, REQUEST_TRANSFER, RETRIEVE_AGENT, RETRIEVE_JAR, SEARCH, SEND_MESSAGE, UNKNOWN

Constructor Summary

ReplyMessage()
ReplyMessage(int messageType)
ReplyMessage(int messageType, java.io.Serializable obj)

Method Summary

java.lang.Object	**getReplyObject**()
void	**setReplyObject**(java.io.Serializable obj)

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.message.Message
addArg, addArg, getArgs, getMessageType, setMessageType

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RequestMessage

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.sockets.message.Message

└

gr.ntua.itsmod.thathan.communication.sockets.message.RequestMessage

All Implemented Interfaces:

IMessage, java.io.Serializable

```
public class RequestMessage
extends Message
```

See Also:

Serialized Form

Field Summary

Fields inherited from class gr.ntua.itsmod.thathan.communication.sockets.message.Message

AGENCY_INFO, AGENCY_INIT, AGENCY_NAME, AGENCY_REGISTER_TO_REGION, AGENCY_STRUCTURE, AGENCY_SYSTEM_INFO, CREATE_AGENT, CREATE_PLACE, KILL_AGENT, LIST_AGENCIES, messageType, MOVE_AGENT, NEW_PLACE, params, REGION_CREATE_AGENT, REGION_INFO, REGION_STRUCTURE, REGION_SYSTEM_INFO, REGISTER_AGENCY, REQUEST_TRANSFER, RETRIEVE_AGENT, RETRIEVE_JAR, SEARCH, SEND_MESSAGE, UNKNOWN

Constructor Summary

RequestMessage()	
RequestMessage(int messageType)	

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.message.Message

addArg, addArg, getArgs, getMessageType, setMessageType

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Γ.11. Package gr.ntua.itsmod.thathan.communication.sockets.region

Class CreateAgentWorker

```
java.lang.Object
  └gr.ntua.itsmod.thathan.communication.Worker
      └gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
          └
```

gr.ntua.itsmod.thathan.communication.sockets.region.CreateAgentWorker

All Implemented Interfaces:

IWorker

```
public class CreateAgentWorker
extends SocketWorker
```

Constructor Summary

CreateAgentWorker(Identifier region_id, Identifier agency_id, java.lang.String place, TAgentBox info)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class ListAgenciesWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
          └
```

gr.ntua.itsmod.thathan.communication.sockets.region.ListAgenciesWorker

All Implemented Interfaces:

IWorker

```
public class ListAgenciesWorker
extends SocketWorker
```

Constructor Summary

ListAgenciesWorker(Identifier region_id)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class MoveAgentWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
```

└

gr.ntua.itsmod.thathan.communication.sockets.region.MoveAgentWorker

All Implemented Interfaces:

IWorker

public class **MoveAgentWorker**
extends SocketWorker

Constructor Summary

MoveAgentWorker(Identifier region_id, AgentID id, Identifier origin_id, java.lang.String origin_place, Identifier target_id, java.lang.String target_place)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class NewPlaceWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
      └

gr.ntua.itsmod.thathan.communication.sockets.region.NewPlaceWorker

All Implemented Interfaces:

IWorker

public class **NewPlaceWorker**
extends SocketWorker

Constructor Summary

NewPlaceWorker(Identifier region_id, Identifier agency_id, java.lang.String name)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker

communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker

generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegionInfoWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
      └
gr.ntua.itsmod.thathan.communication.sockets.region.RegionInfoWorker

All Implemented Interfaces:

IWorker

public class **RegionInfoWorker**
extends SocketWorker

Constructor Summary

RegionInfoWorker(Identifier region_id)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegionStructureWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
          └
```

gr.ntua.itsmod.thathan.communication.sockets.region.RegionStructureWorker

All Implemented Interfaces:

IWorker

```
public class RegionStructureWorker
extends SocketWorker
```

Constructor Summary

RegionStructureWorker(Identifier region_id)	

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker

communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegionSystemInfoWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
          └
```

gr.ntua.itsmod.thathan.communication.sockets.region.RegionSystemInfoWorker

All Implemented Interfaces:

IWorker

```
public class RegionSystemInfoWorker
extends SocketWorker
```

Constructor Summary

RegionSystemInfoWorker(Identifier region_id)	

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegisterAgencyWorker

java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
    └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
      └

gr.ntua.itsmod.thathan.communication.sockets.region.RegisterAgencyWorker

All Implemented Interfaces:

IWorker

public class **RegisterAgencyWorker**
extends SocketWorker

Constructor Summary

RegisterAgencyWorker(Identifier region_id, Identifier agency_id)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class SearchWorker

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.communication.Worker
      └ gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
          └
```

gr.ntua.itsmod.thathan.communication.sockets.region.SearchWorker

All Implemented Interfaces:

IWorker

```
public class SearchWorker
extends SocketWorker
```

Constructor Summary

SearchWorker(Identifier region_id, AgentFilter filter)

Method Summary

Methods inherited from class gr.ntua.itsmod.thathan.communication.sockets.SocketWorker
communicate

Methods inherited from class gr.ntua.itsmod.thathan.communication.Worker
generatedError, getResult, isResultReady

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Γ.12. Package gr.ntua.itsmod.thathan.region

Class AgencyBox

```
java.lang.Object
```

└ **gr.ntua.itsmod.thathan.region.AgencyBox**

All Implemented Interfaces:

java.io.Serializable

```
public class AgencyBox
extends java.lang.Object
implements java.io.Serializable
```

See Also:

Serialized Form

Constructor Summary

AgencyBox()
AgencyBox(AgencyBox other)
AgencyBox(Identifier identifier, AgencyInfo info, java.util.Hashtable places)
AgencyBox(java.lang.String host, int port, AgencyInfo info, java.util.Hashtable places)

Method Summary

java.lang.String	**getHost**()
AgencyInfo	**getInfo**()
java.util.Hashtable	**getPlaces**()
int	**getPort**()
void	**setHost**(java.lang.String host)
void	**setInfo**(AgencyInfo info)
void	**setPlaces**(java.util.Hashtable places)
void	**setPort**(int port)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class AlreadyRegisteredException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └gr.ntua.itsmod.thathan.agent.AgentException
              └
```

gr.ntua.itsmod.thathan.region.AlreadyRegisteredException

All Implemented Interfaces:

java.io.Serializable

```
public class AlreadyRegisteredException
extends AgentException
```

See Also:

Serialized Form

Constructor Summary

AlreadyRegisteredException(java.lang.String msg)	

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Interface IRegion

All Known Implementing Classes:

Region

```
public interface IRegion
```

Method Summary

void	**createAgent**(Identifier agency_id, java.lang.String place, TAgentBox info)
void	**createPlace**(Identifier agency_id, java.lang.String name)
void	**destroy**()
RegionInfo	**getInfo**()
java.lang.String	**getName**()
java.util.Properties	**getProperties**()
java.lang.String	**getProperty**(java.lang.String name)
java.lang.String	**getProperty**(java.lang.String name, java.lang.String dflt)
java.util.Hashtable	**getStructure**()
SystemInfo	**getSystemInfo**()
void	**init**(java.lang.String host)
void	**init**(java.lang.String host, java.lang.String name, int protocol)
java.util.Vector	**listAllAvailableAgencies**()
void	**move**(AgentID id, Identifier origin_id, java.lang.String origin_place, Identifier target_id, java.lang.String target_place)
void	**registerAgency**(Identifier id)
void	**removeAgent**(Identifier agency_id, AgentID id)
Result	**search**(AgentFilter filter)

void	**setProperties**(java.util.Properties props)
void	**setProperty**(java.lang.String name, java.lang.String value)
void	**start**()
void	**stop**()
void	**unregisterAgency**(Identifier id)

Class Region

java.lang.Object
  └ **gr.ntua.itsmod.thathan.region.Region**

All Implemented Interfaces:

IRegion

public class **Region**
extends java.lang.Object
implements IRegion

Field Summary

static java.lang.String	**DEFAULT_NAME**
static int	**DEFAULT_PROTOCOL**

Constructor Summary

Region()
Region(java.lang.String host, java.lang.String name)

Method Summary

void	**createAgent**(Identifier agency_id, java.lang.String place, TAgentBox info)
void	**createPlace**(Identifier agency_id,

	java.lang.String name)
void	**destroy**()
RegionInfo	**getInfo**()
java.lang.String	**getName**()
java.util.Properties	**getProperties**()
java.lang.String	**getProperty**(java.lang.String name)
java.lang.String	**getProperty**(java.lang.String name, java.lang.String dflt)
java.util.Hashtable	**getStructure**()
SystemInfo	**getSystemInfo**()
void	**init**(java.lang.String host)
void	**init**(java.lang.String host, java.lang.String name, int protocol)
java.util.Vector	**listAllAvailableAgencies**()
void	**move**(AgentID id, Identifier origin_id, java.lang.String origin_place, Identifier target_id, java.lang.String target_place)
void	**registerAgency**(Identifier id)
void	**removeAgent**(Identifier agency_id, AgentID id)
Result	**search**(AgentFilter filter)
void	**setProperties**(java.util.Properties props)
void	**setProperty**(java.lang.String name, java.lang.String value)

void	**start**()
void	**stop**()
void	**unregisterAgency**(Identifier id)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegionInfo

java.lang.Object
  └ **gr.ntua.itsmod.thathan.region.RegionInfo**

All Implemented Interfaces:

java.io.Serializable

public class **RegionInfo**
extends java.lang.Object
implements java.io.Serializable

See Also:

Serialized Form

Constructor Summary

RegionInfo()
RegionInfo(RegionInfo other)
RegionInfo(java.lang.String host, int port, java.lang.String name)
RegionInfo(java.lang.String host, java.lang.String name)

Method Summary

Identifier	**getId**()
java.lang.String	**getName**()

void	**setId**(Identifier id)
void	**setName**(java.lang.String name)
java.lang.String	**toString**()

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class RegionInitializationException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └gr.ntua.itsmod.thathan.agent.AgentException
              └
```

gr.ntua.itsmod.thathan.region.RegionInitializationException

All Implemented Interfaces:

java.io.Serializable

```
public class RegionInitializationException
extends AgentException
```

See Also:

Serialized Form

Constructor Summary

RegionInitializationException(java.lang.String msg)	

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage,

getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Γ.13. Package gr.ntua.itsmod.thathan.sockets.management

Class AgencyInitDialog

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Dialog
                  └javax.swing.JDialog
                      └
```

gr.ntua.itsmod.thathan.sockets.management.AgencyInitDialog

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.event.WindowListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class AgencyInitDialog
extends javax.swing.JDialog
implements java.awt.event.ActionListener,
java.awt.event.WindowListener
```

See Also:

Serialized Form

Field Summary

AgencyManager	**owner**

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

AgencyInitDialog(AgencyManager owner)	

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent evt)
void	**windowActivated**(java.awt.event.WindowEvent evt)
void	**windowClosed**(java.awt.event.WindowEvent evt)
void	**windowClosing**(java.awt.event.WindowEvent evt)
void	**windowDeactivated**(java.awt.event.WindowEvent evt)
void	**windowDeiconified**(java.awt.event.WindowEvent evt)
void	**windowIconified**(java.awt.event.WindowEvent evt)
void	**windowOpened**(java.awt.event.WindowEvent evt)

Class AgencyManager

java.lang.Object

```
java.awt.Component
  java.awt.Container
    java.awt.Window
      java.awt.Frame
        javax.swing.JFrame
          
```

gr.ntua.itsmod.thathan.sockets.management.AgencyManager

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class AgencyManager
extends javax.swing.JFrame
implements java.awt.event.ActionListener
```

See Also:

Serialized Form

Field Summary

Agency	**agency**
AgentPane	**agentpane**
static int	**DEFAULT_PORT**
java.lang.String	**hostAddr**
AgencyMenuBar	**menubar**
java.lang.String	**name**
javax.swing.JPanel	**pane**
int	**port**
AgentID	**selectedID**
AgencyServer	**server**
AgencyTreePane	**treepane**

Fields inherited from class javax.swing.JFrame
EXIT_ON_CLOSE

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

AgencyManager()	

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent evt)
static void	**main**(java.lang.String[] args)

Class AgencyMenuBar

```
java.lang.Object
  └java.awt.MenuComponent
      └java.awt.MenuBar
          └gr.ntua.itsmod.thathan.sockets.management.AgencyMenuBar
```

All Implemented Interfaces:

java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

```
public class AgencyMenuBar
extends java.awt.MenuBar
```

See Also:

Serialized Form

Field Summary

java.awt.Menu	**about**
java.awt.MenuItem	**aboutItem**
java.awt.Menu	**actions**
java.awt.MenuItem	**createAgent**
java.awt.MenuItem	**createPlace**
java.awt.Menu	**file**
java.awt.MenuItem	**killAgent**
java.awt.event.ActionListener	**listener**
java.awt.MenuItem	**quit**
java.awt.MenuItem	**refresh**
java.awt.MenuItem	**registerToRegion**
java.awt.Menu	**tools**
java.awt.MenuItem	**transferAgent**

Constructor Summary

AgencyMenuBar(java.awt.event.ActionListener listener)

Method Summary

Methods inherited from class java.awt.MenuBar

add, addNotify, countMenus, deleteShortcut, getAccessibleContext, getHelpMenu, getMenu, getMenuCount, getShortcutMenuItem, remove, remove, removeNotify, setHelpMenu, shortcuts

Methods inherited from class java.awt.MenuComponent

dispatchEvent, getFont, getName, getParent, getPeer, postEvent, setFont, setName, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.awt.MenuContainer

getFont, postEvent

Class AgencyPane

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └
```

gr.ntua.itsmod.thathan.sockets.management.AgencyPane

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

---

public class **AgencyPane**

extends javax.swing.JPanel

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent
javax.swing.JComponent.AccessibleJComponent

Field Summary

Fields inherited from class javax.swing.JComponent
TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

AgencyPane()	

Method Summary

void	**display**(AgencyBox box)
void	**setAddress**(java.lang.String address)
void	**setName**(java.lang.String name)
void	**setPort**(int port)

Class AgencyRegisterDialog

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Dialog
                  └javax.swing.JDialog
                      └
```

gr.ntua.itsmod.thathan.sockets.management.AgencyRegisterDialog

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class AgencyRegisterDialog
extends javax.swing.JDialog
implements java.awt.event.ActionListener
```

See Also:

Serialized Form

Field Summary

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

AgencyRegisterDialog(AgencyManager owner)

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent evt)

Class AgencyTreePane

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JScrollPane
                  └
```

gr.ntua.itsmod.thathan.sockets.management.AgencyTreePane

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.event.TreeSelectionListener, javax.swing.ScrollPaneConstants

```
public class AgencyTreePane
extends javax.swing.JScrollPane
implements javax.swing.event.TreeSelectionListener
```

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent
javax.swing.JComponent.AccessibleJComponent

Field Summary

AgencyManager	**owner**

Fields inherited from class javax.swing.JComponent

TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.ScrollPaneConstants

COLUMN_HEADER, HORIZONTAL_SCROLLBAR, HORIZONTAL_SCROLLBAR_ALWAYS, HORIZONTAL_SCROLLBAR_AS_NEEDED, HORIZONTAL_SCROLLBAR_NEVER, HORIZONTAL_SCROLLBAR_POLICY, LOWER_LEADING_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_TRAILING_CORNER, ROW_HEADER, UPPER_LEADING_CORNER, UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, UPPER_TRAILING_CORNER, VERTICAL_SCROLLBAR, VERTICAL_SCROLLBAR_ALWAYS, VERTICAL_SCROLLBAR_AS_NEEDED, VERTICAL_SCROLLBAR_NEVER, VERTICAL_SCROLLBAR_POLICY, VIEWPORT

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

AgencyTreePane(AgencyManager owner)	

Method Summary

void	**createTree**(java.util.Hashtable structure)

void	**setAgentPane**(AgentPane agentpane)
void	**valueChanged**(javax.swing.event.TreeSelectionEvent evt)

Class AgentMutableTreeNode

```
java.lang.Object
  └javax.swing.tree.DefaultMutableTreeNode
      └
```

gr.ntua.itsmod.thathan.sockets.management.AgentMutableTreeNode

All Implemented Interfaces:

java.io.Serializable, java.lang.Cloneable, javax.swing.tree.MutableTreeNode, javax.swing.tree.TreeNode

---

```
public class AgentMutableTreeNode
extends javax.swing.tree.DefaultMutableTreeNode
```

See Also:

Serialized Form

Field Summary

Fields inherited from class javax.swing.tree.DefaultMutableTreeNode
EMPTY_ENUMERATION

Constructor Summary

AgentMutableTreeNode()
AgentMutableTreeNode(java.lang.Object arg0)
AgentMutableTreeNode(java.lang.Object arg0, boolean arg1)

Method Summary

java.lang.String	**getName**()

void	**setName**(java.lang.String name)
java.lang.String	**toString**()

Methods inherited from class javax.swing.tree.DefaultMutableTreeNode

add, breadthFirstEnumeration, children, clone, depthFirstEnumeration, getAllowsChildren, getChildAfter, getChildAt, getChildBefore, getChildCount, getDepth, getFirstChild, getFirstLeaf, getIndex, getLastChild, getLastLeaf, getLeafCount, getLevel, getNextLeaf, getNextNode, getNextSibling, getParent, getPath, getPreviousLeaf, getPreviousNode, getPreviousSibling, getRoot, getSharedAncestor, getSiblingCount, getUserObject, getUserObjectPath, insert, isLeaf, isNodeAncestor, isNodeChild, isNodeDescendant, isNodeRelated, isNodeSibling, isRoot, pathFromAncestorEnumeration, postorderEnumeration, preorderEnumeration, remove, remove, removeAllChildren, removeFromParent, setAllowsChildren, setParent, setUserObject

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class AgentPane

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └
```

gr.ntua.itsmod.thathan.sockets.management.AgentPane

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

```
public class AgentPane
extends javax.swing.JPanel
```

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent

javax.swing.JComponent.AccessibleJComponent

Field Summary

Fields inherited from class javax.swing.JComponent

TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

AgentPane()	

Method Summary

void	**display**(TAgentBox box)
void	**setHome**(java.lang.String home)
void	**setId**(java.lang.String id)
void	**setName**(java.lang.String name)

void	**setState**(int state)

Class CreateAgentDialog

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Dialog
                  └javax.swing.JDialog
                      └
```

gr.ntua.itsmod.thathan.sockets.management.CreateAgentDialog

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

```
public class CreateAgentDialog
extends javax.swing.JDialog
implements java.awt.event.ActionListener
```

See Also:

Serialized Form

Field Summary

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants

DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

CreateAgentDialog(AgencyManager owner)	

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent evt)

Class InputException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └gr.ntua.itsmod.thathan.sockets.management.InputException
```

All Implemented Interfaces:

java.io.Serializable

```
public class InputException
extends java.lang.Exception
```

See Also:

Serialized Form

Constructor Summary

InputException()	

Method Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class JarFilter

```
java.lang.Object
  └javax.swing.filechooser.FileFilter
      └gr.ntua.itsmod.thathan.sockets.management.JarFilter
```

```
public class JarFilter
extends javax.swing.filechooser.FileFilter
```

Constructor Summary

JarFilter()	

Method Summary

boolean	**accept**(java.io.File file)
java.lang.String	**getDescription**()

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class RegionInitDialog

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Dialog
                  └javax.swing.JDialog
                      └
```

gr.ntua.itsmod.thathan.sockets.management.RegionInitDialog

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.event.WindowListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

```
public class RegionInitDialog
extends javax.swing.JDialog
implements java.awt.event.ActionListener,
java.awt.event.WindowListener
```

See Also:

Serialized Form

Field Summary

RegionManager	**owner**

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

RegionInitDialog(RegionManager owner)	

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent evt)
void	**windowActivated**(java.awt.event.WindowEvent evt)
void	**windowClosed**(java.awt.event.WindowEvent evt)
void	**windowClosing**(java.awt.event.WindowEvent evt)
void	**windowDeactivated**(java.awt.event.WindowEvent evt)
void	**windowDeiconified**(java.awt.event.WindowEvent evt)

void	**windowIconified**(java.awt.event.WindowEvent evt)
void	**windowOpened**(java.awt.event.WindowEvent evt)

Class RegionManager

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └
```

gr.ntua.itsmod.thathan.sockets.management.RegionManager

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class RegionManager
extends javax.swing.JFrame
implements java.awt.event.ActionListener
```

See Also:

Serialized Form

Field Summary

static int	**DEFAULT_PORT**
java.lang.String	**hostAddr**
RegionMenuBar	**menubar**
java.lang.String	**name**
javax.swing.JPanel	**pane**
int	**port**
Region	**region**
RegionTabbedPane	**regionPane**
RegionServer	**server**

RegionTreePane	**treepane**

Fields inherited from class javax.swing.JFrame
EXIT_ON_CLOSE

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

RegionManager()	

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent evt)
static void	**main**(java.lang.String[] args)

Class RegionMenuBar

```
java.lang.Object
  └java.awt.MenuComponent
      └java.awt.MenuBar
          └gr.ntua.itsmod.thathan.sockets.management.RegionMenuBar
```

All Implemented Interfaces:

java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

```
public class RegionMenuBar
extends java.awt.MenuBar
```

See Also:

Serialized Form

Field Summary

java.awt.Menu	**about**
java.awt.MenuItem	**aboutItem**
java.awt.Menu	**file**
java.awt.event.ActionListener	**listener**
java.awt.MenuItem	**quit**
java.awt.MenuItem	**refresh**
java.awt.MenuItem	**registerAgency**
java.awt.Menu	**tools**
java.awt.MenuItem	**unregisterAgency**

Constructor Summary

RegionMenuBar(java.awt.event.ActionListener listener)

Class RegionRegisterAgencyDialog

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Dialog
                  └javax.swing.JDialog
                      └
```

gr.ntua.itsmod.thathan.sockets.management.RegionRegisterAgencyDialog

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class RegionRegisterAgencyDialog
extends javax.swing.JDialog
implements java.awt.event.ActionListener
```

See Also:

Serialized Form

Field Summary

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants

DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

RegionRegisterAgencyDialog(RegionManager owner)

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent evt)

Class RegionTabbedPane

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JTabbedPane
                  └
```

gr.ntua.itsmod.thathan.sockets.management.RegionTabbedPane

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.SwingConstants

```
public class RegionTabbedPane
extends javax.swing.JTabbedPane
```

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent
javax.swing.JComponent.AccessibleJComponent

Field Summary

Fields inherited from class javax.swing.JTabbedPane
SCROLL_TAB_LAYOUT, WRAP_TAB_LAYOUT

Fields inherited from class javax.swing.JComponent

TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION,
WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED,
WHEN_IN_FOCUSED_WINDOW

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT,
TOP_ALIGNMENT

Fields inherited from interface javax.swing.SwingConstants

BOTTOM, CENTER, EAST, HORIZONTAL, LEADING, LEFT, NEXT, NORTH,
NORTH_EAST, NORTH_WEST, PREVIOUS, RIGHT, SOUTH, SOUTH_EAST,
SOUTH_WEST, TOP, TRAILING, VERTICAL, WEST

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

RegionTabbedPane()	

Class RegionTreePane

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JScrollPane
                  └
```

gr.ntua.itsmod.thathan.sockets.management.RegionTreePane

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.event.TreeSelectionListener, javax.swing.ScrollPaneConstants

```
public class RegionTreePane
extends javax.swing.JScrollPane
implements javax.swing.event.TreeSelectionListener
```

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent

javax.swing.JComponent.AccessibleJComponent

Field Summary

Fields inherited from class javax.swing.JComponent

TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION,
WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED,
WHEN_IN_FOCUSED_WINDOW

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT,
TOP_ALIGNMENT

Fields inherited from interface javax.swing.ScrollPaneConstants

COLUMN_HEADER, HORIZONTAL_SCROLLBAR, HORIZONTAL_SCROLLBAR_ALWAYS,
HORIZONTAL_SCROLLBAR_AS_NEEDED, HORIZONTAL_SCROLLBAR_NEVER,
HORIZONTAL_SCROLLBAR_POLICY, LOWER_LEADING_CORNER, LOWER_LEFT_CORNER,
LOWER_RIGHT_CORNER, LOWER_TRAILING_CORNER, ROW_HEADER,
UPPER_LEADING_CORNER, UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER,
UPPER_TRAILING_CORNER, VERTICAL_SCROLLBAR, VERTICAL_SCROLLBAR_ALWAYS,
VERTICAL_SCROLLBAR_AS_NEEDED, VERTICAL_SCROLLBAR_NEVER,
VERTICAL_SCROLLBAR_POLICY, VIEWPORT

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

RegionTreePane(java.util.Hashtable hashtable, Region region, RegionManager owner)

Method Summary

void	**createTree**(java.util.Hashtable structure)
void	**setAgencyPane**(AgencyPane agencypane)
void	**setAgentPane**(AgentPane agentpane)
void	**valueChanged**(javax.swing.event.TreeSelectionEvent evt)

Class RegionUnregisterAgencyDialog

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Dialog
                  └javax.swing.JDialog
                      └
```

gr.ntua.itsmod.thathan.sockets.management.RegionUnregisterAgencyDialog

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class RegionUnregisterAgencyDialog
extends javax.swing.JDialog
implements java.awt.event.ActionListener
```

See Also:

Serialized Form

Field Summary

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT,

TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

RegionUnregisterAgencyDialog(RegionManager owner)	

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent evt)

Class RetrieveAgentDialog

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Dialog
                  └javax.swing.JDialog
                      └
```

gr.ntua.itsmod.thathan.sockets.management.RetrieveAgentDialog

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class RetrieveAgentDialog
extends javax.swing.JDialog
implements java.awt.event.ActionListener
```

See Also:

Serialized Form

Field Summary

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

RetrieveAgentDialog(AgencyManager owner)	

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent evt)

Class TransferAgentDialog

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Dialog
                  └javax.swing.JDialog
                      └
```

gr.ntua.itsmod.thathan.sockets.management.TransferAgentDialog

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener,

javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

```
public class TransferAgentDialog
extends javax.swing.JDialog
implements java.awt.event.ActionListener
```

See Also:

Serialized Form

Field Summary

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants

DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, EXIT_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

TransferAgentDialog(AgencyManager owner)	

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent evt)

Γ.14. Package gr.ntua.itsmod.thathan.utils

Class JarResource

```
java.lang.Object
  └ gr.ntua.itsmod.thathan.utils.JarResource
```

```
public class JarResource
```

extends java.lang.Object

Constructor Summary

JarResource(java.lang.String jarFileName) Creates a JarResource.	

Method Summary

byte[]	**getResource**(java.lang.String name) Extracts a jar resource as a blob.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

JarResource

public **JarResource**(java.lang.String jarFileName)

Creates a JarResource. It extracts all resources from a Jar into an internal hashtable, keyed by resource names.

Parameters:

jarFileName - a jar or zip file

Method Detail

getResource

public byte[] **getResource**(java.lang.String name)

Extracts a jar resource as a blob.

Parameters:

name - a resource name.