ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εξαγωγή Χαρακτηριστικών από Οντολογίες: Εφαρμογή σε Πρωτόκολλα Εύρεσης Υπηρεσιών

Γιώργος Γρ. Νταρλαδήμας

Υπεύθυνος: Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2005

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εξαγωγή Χαρακτηριστικών από Οντολογίες: Εφαρμογή σε Πρωτόκολλα Εύρεσης Υπηρεσιών

ΓΙΩΡΓΟΣ ΓΡ. ΝΤΑΡΛΑΔΗΜΑΣ

Α. Μ. 2002129

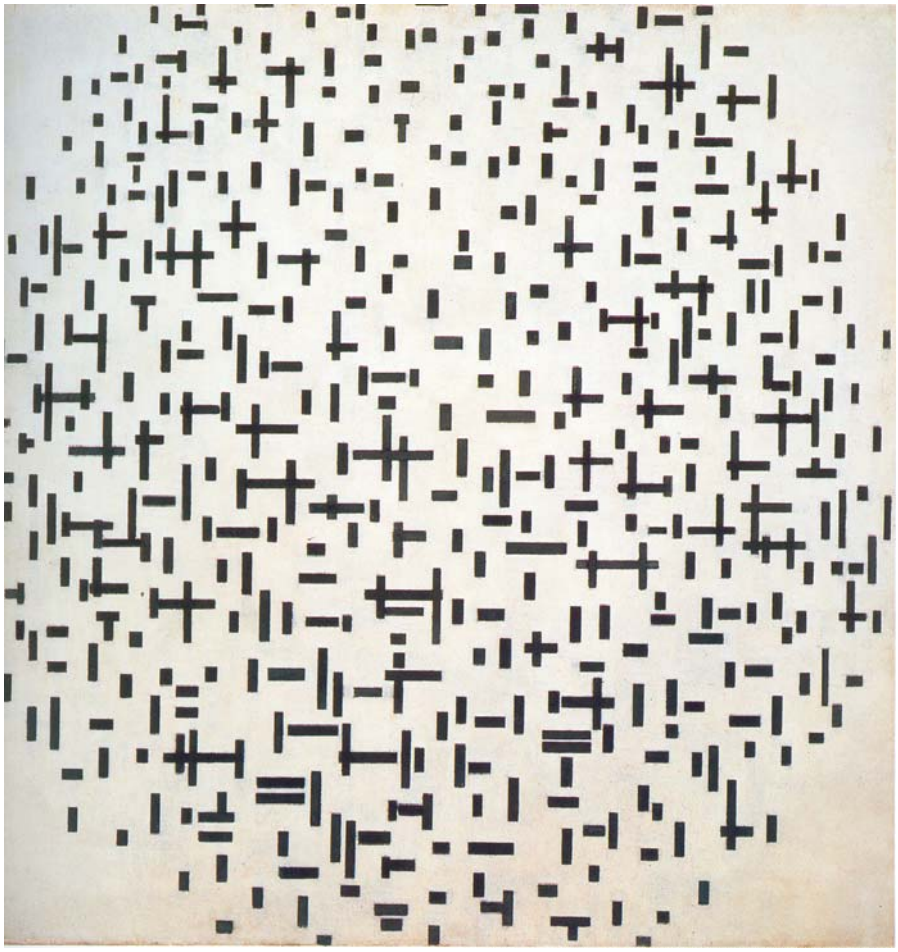
ΕΠΙΒΛΕΠΟΝΤΕΣ:

Επίκουρος Καθηγητής ΕΚΠΑ, **ΕΥΣΤΑΘΙΟΣ ΧΑΤΖΗΕΥΘΥΜΙΑΔΗΣ**

Υποψήφιος Διδάκτωρ ΕΚΠΑ, **ΧΡΗΣΤΟΣ ΑΝΑΓΝΩΣΤΟΠΟΥΛΟΣ**

Έτσι είναι αν έτσι νομίζετε

Luigi Pirantello



Θα ήθελα να ευχαριστήσω πρωτίστως το Χρήστο και τον κο Χατζηευθυμιάδη, χωρίς τη συμβολή των οποίων δεν θα ήταν δυνατή η πραγματοποίηση αυτής της εργασίας. Επιπλέον, ευγνωμοσύνη χρωστώ στο Λουκά Σαμαρά και τον Ιάννη Ξενάκη. Τέλος, ειδική μνεία πρέπει να γίνει στο Nokia Research Center (Helsinki, Finland) και την παραλία του Αγ. Νικολάου στη Φολέγανδρο.

Η παρούσα εργασία είναι αφιερωμένη στη Στέλλα.

ΙΔΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Σ' ένα σύγχρονο περιβάλλον διάχυτου υπολογισμού παρουσιάζεται πληθώρα υπηρεσιών με διαφορετικά πρωτόκολλα Εύρεσης Υπηρεσιών (Jini, UPnP, Salutation, Bluetooth SDP). Καθεμιά από αυτές τις υπηρεσίες διαθέτει διαφορετική αρχιτεκτονική, αλλά κάποιες φορές παρόμοιο σκοπό. Από την άλλη μεριά, οι χρήστες έχουν παραπλήσιες ανάγκες και συχνά η αίτηση τους στο τοπικό δίκτυο για μια συγκεκριμένη εφαρμογή αποτυγχάνει, είτε γιατί δεν υπάρχει η συγκεκριμένη υπηρεσία στο τοπικό δίκτυο είτε επειδή υπάρχει κάποια παρόμοια, αλλά για ποικίλους λόγους δεν επιτεύχθηκε το λεγόμενο *service matching*. Συνεπώς, επιτακτική προβάλλει η ανάγκη από τη μια για *διαλειτουργικότητα* μεταξύ δικτύων και από την άλλη για εντοπισμό της «πλησιέστερης» υπηρεσίας στο ίδιο δίκτυο.

Η προαναφερθείσα αποτυχία μπορεί ενδεχομένως να αντιμετωπιστεί με δύο τρόπους:

- αναζήτηση της ζητηθείσας υπηρεσίας σε διαφορετικό δίκτυο,
- εύρεση παρεμφερούς υπηρεσίας στο ίδιο δίκτυο.

Στα πλαίσια της παρούσας διπλωματικής εργασίας για την πρώτη περίπτωση, προτείνεται ένα είδος γεφύρωσης των διαφορετικών δικτύων με τη βοήθεια της οντότητας HyperReggie, που περιγράφεται διεξοδικά στο κεφάλαιο 6. Όσον αφορά τη δεύτερη περίπτωση, επιχειρείται μια διαφορετική προσέγγιση. Αυτό σημαίνει ότι όταν δεν είναι δυνατή η εύρεση της ζητούμενης υπηρεσίας, αντί να επιστρέφεται ένα αρνητικό μήνυμα στο χρήστη, επιστρέφεται μια λίστα από παρεμφερείς υπηρεσίες. Μέσω της οντολογίας, που περιγράφει τα χαρακτηριστικά των υπηρεσιών είναι δυνατόν να βρεθούν οι πιο «συγγενικές» υπηρεσίες, χρησιμοποιώντας παράλληλα διαδικασία μέτρησης σημασιολογικής και λεξικογραφικής ομοιότητας που βασίζεται σε αλγορίθμους αντιστοίχισης και στην θεωρία ασάφειας, όπως παρουσιάζονται στα κεφάλαια 3 και 4. Επιπρόσθετα, στο χρήστη προτείνεται και ένα σύνολο από υπηρεσίες (*suggested services*), οι οποίες μπορούν να συνδυαστούν ή να συνεργαστούν κατάλληλα με τη ζητηθείσα υπηρεσία.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Σημασιολογικός Ιστός, Διαχείριση Οντολογιών, Θεωρία Ασάφειας

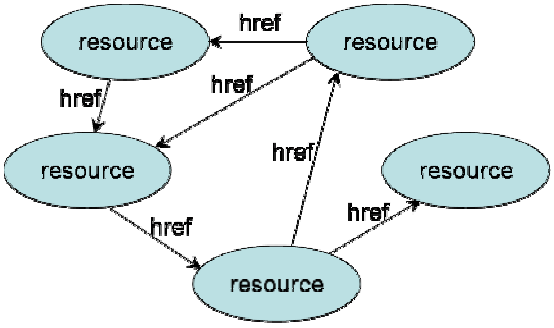
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Σημασιολογική Ομοιότητα, Σημασιολογικός Ιστός, Οντολογία, Πρωτόκολλο Εύρεσης Υπηρεσιών, Θεωρία Ασάφειας

ΠΕΡΙΕΧΟΜΕΝΑ

1 ΕΙΣΑΓΩΓΗ ΣΤΟ ΣΗΜΑΣΙΟΛΟΓΙΚΟ ΙΣΤΟ

Το μεγαλύτερο τμήμα του Παγκόσμιου Ιστού (*World Wide Web*), που έχει αναπτυχθεί έως τώρα, έχει σχεδιαστεί ως μια συλλογή από αρχεία, τα οποία προορίζονται να διαβάζονται αποκλειστικά από ανθρώπους και όχι σαν καθαρή πληροφορία, την οποία θα μπορούσαν να επεξεργάζονται απευθείας προγράμματα υπολογιστών. Οι σημερινοί υπολογιστές έχουν – χωρίς αμφιβολία – τη δυνατότητα ν' αναλύσουν ιστοσελίδες προκειμένου ν' αντιληφθούν τη δομή, π.χ. αυτός ο σύνδεσμος οδηγεί σε μια άλλη σελίδα, αλλά δεν είναι σε θέση ν' αναγνωρίσουν τη σημασιολογία, π.χ. αυτός ο σύνδεσμος οδηγεί στην αρχική σελίδα της μηχανής αναζήτησης *Google*.

Στα σχήματα 1 και 2 φαίνεται ο τρόπος λειτουργίας του σημερινού Ιστού. Ο Παγκόσμιος Ιστός αποτελεί μια κοινή πύλη για εφαρμογές που είναι προσβάσιμες μέσω αρχείων, των λεγόμενων ιστοσελίδων, και παράλληλα μια πλατφόρμα για εφαρμογές πολυμέσων. Ουσιαστικά, πρόκειται για μια ψηφιακή βιβλιοθήκη που αποτελείται από ιστοσελίδες, οι οποίες διασυνδέονται μεταξύ τους. Οι άνθρωποι που ασχολούνται με την ανάπτυξη λογισμικού, πραγματοποιούν την αναπαράσταση της γνώσης και το σχεδιασμό, ενώ οι χρήστες καλούνται να ερμηνεύσουν την πληροφορία. Τέλος, οι υπολογιστές είναι υπεύθυνοι μόνο για την παρουσίαση, μια που δεν έχουν τη δυνατότητα να επεξεργαστούν την πληροφορία.

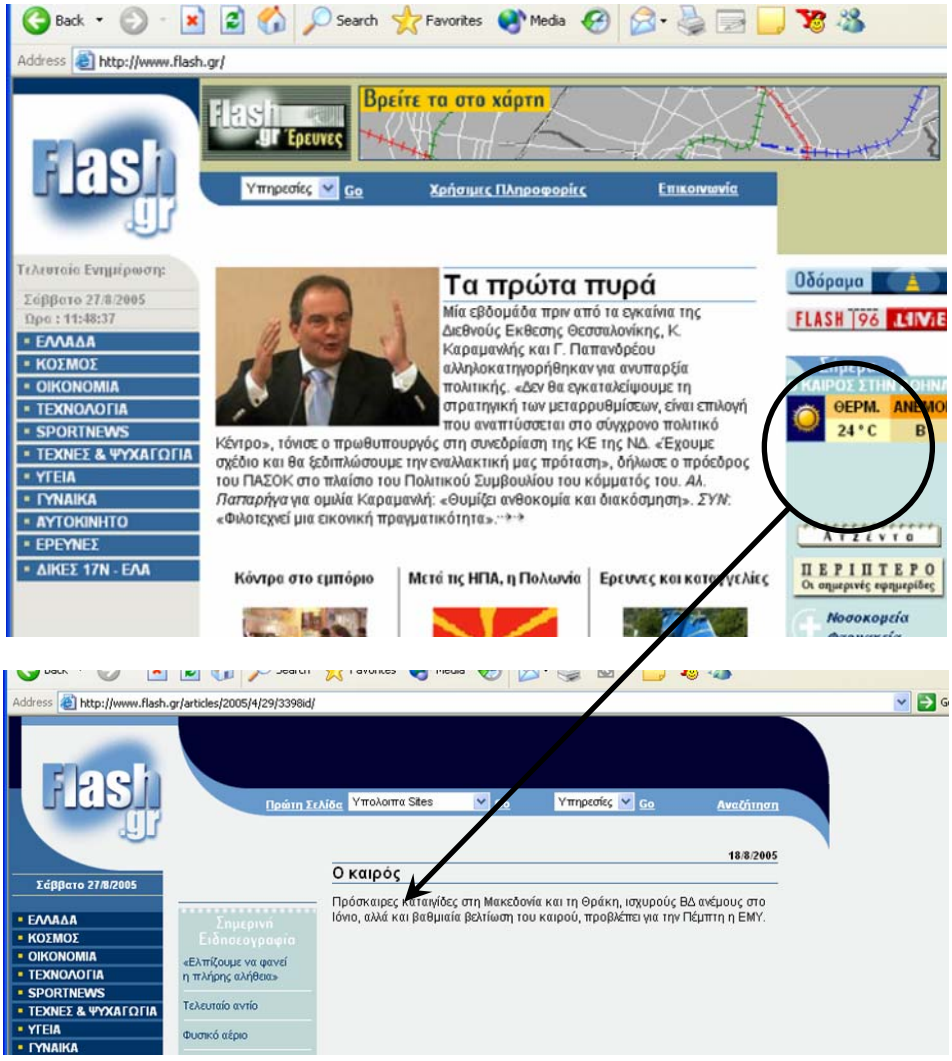


Εικόνα 1: Τρόπος διασύνδεσης των ιστοσελίδων του Παγκόσμιου Ιστού

Είναι φανερό πως με τη σημερινή μορφή ο Παγκόσμιος Ιστός δεν μπορεί να δώσει λύση σε συγκεκριμένες περιπτώσεις, όπου απαιτείται βαθύτερο υπόβαθρο γνώσης. Παραδείγματος χάριν, τέτοια περίπτωση αποτελεί το ερώτημα «ποιο ζώο εκτός από τη νυχτερίδα και το δελφίνι χρησιμοποιεί sonar». Μια άλλη περίπτωση είναι η ανάκτηση γνώσης από αποθήκες δεδομένων, όσον αφορά π.χ. ταξιδιωτικές πληροφορίες ή τιμές για προϊόντα και υπηρεσίες. Τέλος, άλλες εφαρμογές που θα είναι εφικτές στο Σημασιολογικό Ιστό αποτελούν η σημασιολογική αναζήτηση (π.χ. εύρεση δρομολογίων

πλοίων στην Ελλάδα), η χρήση ευφυών πρακτόρων για εύρεση πληροφοριών (π.χ. αναζήτηση στο ηλεκτρονικό κατάστημα *e-shop* συγκεκριμένου προϊόντος με τη χαμηλότερη τιμή) και σημασιολογικές πύλες.

Το πρόβλημα που εμφανίζει ο Παγκόσμιος Ιστός με την υπάρχουσα μορφή εστιάζεται στο γεγονός ότι σε μια τυπική ιστοσελίδα η *Markup* πληροφορία περιέχει χαρακτηριστικά παρουσίασης (π.χ. μέγεθος γραμματοσειράς) και συνδέσεις με σχετικό περιεχόμενο. Συνεπώς για να είναι προσβάσιμο το σημασιολογικό περιεχόμενο όχι μόνο από τους ανθρώπους αλλά και από τους υπολογιστές, επιτακτική προβάλλει η ανάγκη για προσθήκη σημασιολογικού *Markup*, δηλαδή σχολιασμών στις ιστοσελίδες. Η σημασιολογία επιτυγχάνεται με τη χρήση οντολογιών, για τις οποίες εκτενής αναφορά γίνεται στο επόμενο κεφάλαιο.



Εικόνα 2: Παράδειγμα υπερ-συνδέσμου ιστοσελίδων

Η μεταμόρφωση του Ιστού σε Σημασιολογικό Ιστό μπορεί να επιτευχθεί εάν οι υπάρχουσες ιστοσελίδες επεκταθούν, ενσωματώνοντας δεδομένα που απευθύνονται σε υπολογιστές. Μ' αυτόν τον τρόπο, το περιεχόμενο του Ιστού είναι δυνατόν να δομηθεί, χρησιμοποιώντας υπερ-συνδέσμους για τους ορισμούς των όρων-κλειδί και κανόνες με σκοπό τον συμπερασμό για αυτούς τους όρους. Ο Σημασιολογικός Ιστός – σύμφωνα με τον εμπνευστή του, Tim Berners-Lee [1] – δεν πρέπει να θεωρείται ως ένας ξεχωριστός Ιστός, αλλά μια επέκταση του ήδη υπάρχοντος Ιστού, στην οποία η πληροφορία διαθέτει καλά ορισμένο νόημα, αναβαθμίζοντας τη συνεργασία ανθρώπων και υπολογιστών.

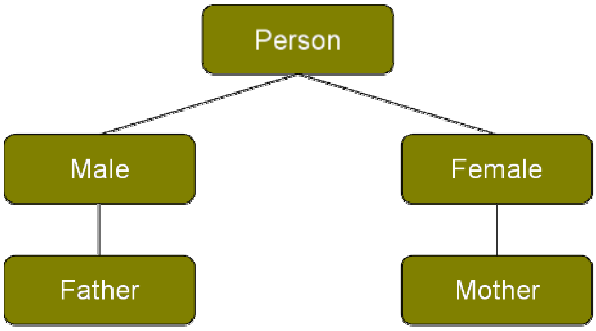
Η θεμελιώδης ιδιότητα του Παγκόσμιου Ιστού είναι η καθολικότητά του. Συνεπώς, η τεχνολογία αυτή δεν θα έπρεπε να εξαρτάται από τις διάφορες κουλτούρες, γλώσσες ή άλλα ξεχωριστά χαρακτηριστικά της πληροφορίας, παρόλο που η γνώση μπορεί να παρουσιάζεται σε διαφορετικές εκφάνσεις. Η πρόκληση του Σημασιολογικού Ιστού έγκειται στη δημιουργία μιας γλώσσας που μπορεί να εκφράσει τόσο τα δεδομένα όσο και τους κανόνες για απόφαση αλλά που συνάμα επιτρέπει την εφαρμογή κανόνων στον Ιστό από οποιοδήποτε σύστημα αναπαράστασης γνώσης. Το βήμα που απαιτείται πριν την εγκαθίδρυση της κοινότητας του Σημασιολογικού Ιστού είναι η προσθήκη λογικής, της δυνατότητας δηλαδή να χρησιμοποιούνται κανόνες έτσι ώστε να πραγματοποιούνται συμπερασμοί.

Σαν παράδειγμα μπορούμε να θεωρήσουμε την υποβολή της ερώτησης σε μια μηχανή αναζήτησης: «*Πόσες γραμμές τραίνου υπάρχουν στη Μεγάλη Βρετανία;*». Στο σημερινό διαδίκτυο, η μηχανή αναζήτησης θα επιστρέψει αρκετές σελίδες, αλλά η απάντηση ενδέχεται να μην υπάρχει σε καμιά από αυτές. Αντίθετα, στον Σημασιολογικό Ιστό οι απαντήσεις μπορεί να έχουν ως εξής:

- σύμφωνα με το [2] οι γραμμές υπολογίζονται σε πάνω από 2000,
- υπάρχει μια βάση δεδομένων που παρέχει τέτοιου είδους πληροφορίες, αλλά απαιτεί κωδικό πρόσβασης,
- υπάρχει μια υπηρεσία διαδικτύου, αλλά κοστίζει 3 στερλίνες,
- μπορεί να βρεθεί η απάντηση, αλλά θα χρειαστούν μερικές ώρες.

2 ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ ΟΝΤΟΛΟΓΙΩΝ

Όπως είναι γνωστό, η γνώση αποτελεί ένα σύνολο από δεδομένα με σημασιολογικό περιεχόμενο και οι *οντολογίες* είναι το εργαλείο που χρησιμοποιείται, προκειμένου ν' αναπαρασταθεί η γνώση ενός συγκεκριμένου πεδίου ενδιαφέροντος. Στην ουσία, μια οντολογία περιγράφει τις κλάσεις που εμπλέκονται στο συγκεκριμένο πεδίο και παράλληλα τις αλληλεπιδράσεις που υπάρχουν μεταξύ αυτών των κλάσεων (Εικόνα 3). Όσον αφορά τον ορισμό της οντολογίας, στο πεδίο της φιλοσοφίας σύμφωνα με τον Αριστοτέλη [3], πρόκειται για την επιστήμη της ύπαρξης. Επιπρόσθετα, στο πεδίο της τεχνητής νοημοσύνης, κατά τον Gruber [4], η οντολογία αποτελείται από τις ρητές προδιαγραφές της αντίληψης για τον κόσμο, ενώ κατά τον Borst [5] αφορά την τυπική προδιαγραφή μιας κοινής αντίληψης για τον κόσμο.



Εικόνα 3: Παράδειγμα οντολογίας

Τα βασικά στοιχεία που χαρακτηρίζουν μια οντολογία είναι τα ακόλουθα:

- *κλάσεις,*
- *σχέσεις,*
- *αξιώματα και*
- *στιγμιότυπα.*

Οι κλάσεις περιλαμβάνουν οτιδήποτε για το οποίο μπορεί να διατυπωθεί μια πρόταση, ενώ οι σχέσεις αναπαριστούν τις αλληλεπιδράσεις μεταξύ των κλάσεων του πεδίου ορισμού. Τα αξιώματα περιγράφουν προτάσεις που είναι πάντα αληθείς και τα στιγμιότυπα είναι συγκεκριμένα αντικείμενα των κλάσεων.

Υπάρχουν πολλές κατηγορίες γλωσσών αναπαράστασης οντολογιών, όπως οι παραδοσιακές γλώσσες, οι λεγόμενες *web-based* γλώσσες και οι γλώσσες που αναπτύχθηκαν με στόχο ν' αναπαραστήσουν συγκεκριμένες οντολογίες και να χρησιμοποιηθούν σε πολύ εξειδικευμένες εφαρμογές. Στην πρώτη κατηγορία ανήκουν η κατηγορηματική *λογική πρώτης τάξης*, η *λογική πλαισίων*, η *περιγραφική λογική*, και

μερικά παραδείγματα είναι οι *Carin*, *Flogic*, *Loom*, *OCML* και *Ontolingua*. Η σύνταξη των γλωσσών της δεύτερης κατηγορίας, που θα απασχολήσει περισσότερο την παρούσα εργασία, βασίζεται στην *XML* και τέλος ορισμένα παραδείγματα της τρίτης κατηγορίας γλωσσών αποτελούν οι *CycL*, *GRAIL* και *NKRL*. Οι διαφορές των γλωσσών αναπαράστασης οντολογιών συνίστανται:

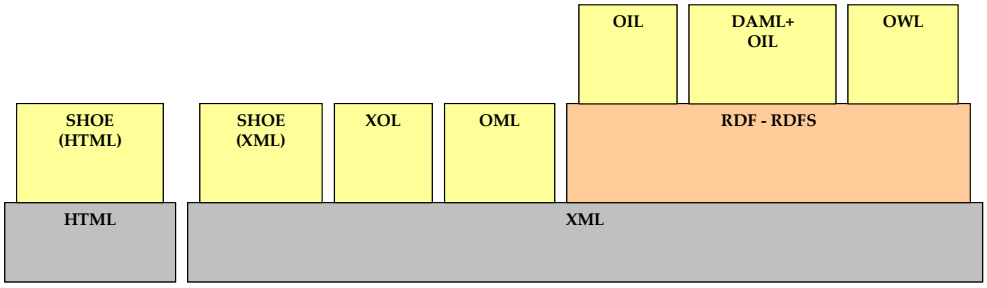
- στη σύνταξη,
- την ορολογία,
- την εκφραστικότητα (κάτι που μπορεί να εκφραστεί σε μια γλώσσα, ενδέχεται να μην διατυπώνεται σε μια άλλη) και
- τη σημασιολογία (η ίδια δήλωση μπορεί να σημαίνει διαφορετικά πράγματα σε διαφορετικές γλώσσες).

Στα πλαίσια της παρούσας εργασίας, θα γίνει μια επιμέρους αναφορά στις *web-based* γλώσσες και θα ακολουθήσει διεξοδική περιγραφή της συγκεκριμένης γλώσσας καθώς και του εργαλείου που χρησιμοποιήθηκε για τους σκοπούς της εργασίας.

Οι κυριότερες *web-based* γλώσσες είναι οι εξής:

- SHOE,
- XOL,
- OML/KML,
- RDFS,
- DAML,
- DAML+OIL,
- OIL και
- OWL [6]

Όπως φαίνεται και στην εικόνα 4, η SHOE μπορεί να βασίζεται τόσο στην HTML όσο και την XML, οι XOL και OML στην XML ενώ οι OIL, DAML+OIL και OWL αποτελούν επέκταση της RDFS.



Εικόνα 4: Γλώσσες αναπαράστασης οντολογιών

Όσον αφορά την XML, πρόκειται για μοντέλο δεδομένων χαμηλού επιπέδου, το οποίο δεν μπορεί να χρησιμοποιηθεί για τη δημιουργία οντολογιών εξειδικευμένου τύπου. Επίσης δεν μπορεί να στηρίξει τις διαμοιραζόμενες πηγές στο διαδικτύο και δε διαθέτει μηχανισμό συμπερασμού.

Η RDF αποτελεί ένα τυποποιημένο μοντέλο δεδομένων για το οποίο διατίθενται *parsers* και *APIs*. Αρχικά προοριζόταν από την W3C για αναπαράσταση δεδομένων και ενδείκνυται για τη δημιουργία εκτεταμένων λεξικών. Επιπροσθέτως, στους περιορισμούς της RDF συγκαταλέγονται η αδυναμία σημασιολογικής αναπαράστασης της πληροφορίας και η έλλειψη μηχανής συμπερασματολογίας.

Η DAML+OIL δημιουργήθηκε από την DARPA σε συνεργασία με την Ευρωπαϊκή Επιτροπή για *Agent Markup Languages*. Βασίζεται στην RDFS, αποτελεί πρόγονο της OWL και έχει εφαρμοστεί στην κατασκευή πολλών οντολογιών.

Όσον αφορά την OWL, πρόκειται για την πιο πρόσφατη εξέλιξη στο χώρο των τυποποιημένων γλωσσών από την W3C. Διαθέτει πλουσιότερο σύνολο από τελεστές και βασίζεται σ' ένα λογικό μοντέλο, επιτρέποντας έτσι τόσο τον ορισμό όσο και την περιγραφή των εννοιών. Οι πιο σύνθετες έννοιες μπορούν να προκύψουν χρησιμοποιώντας τους ορισμούς των απλούστερων εννοιών. Επίσης, το λογικό μοντέλο επιτρέπει τη χρήση ενός *μηχανισμού συμπερασμού*, ο οποίος εξετάζει αν όλοι οι ορισμοί είναι συνεπείς. Μ' αυτόν τον τρόπο βοηθά στη διατήρηση της ιεραρχίας, γεγονός που είναι εξαιρετικά χρήσιμο σε περιπτώσεις, όπου κάποιες κλάσεις έχουν παραπάνω από έναν πατέρα.

Η OWL μπορεί να διακριθεί, ως προς την εκφραστικότητα, σε τρεις υπο-γλώσσες:

- OWL-Lite,
- OWL-DL και
- OWL-Full.

Η OWL-Lite είναι η λιγότερο εκφραστική υπο-γλώσσα και αποτελεί την πιο απλή, συντακτικά, υπο-γλώσσα. Χρησιμοποιείται σε περιπτώσεις όπου απαιτείται η δημιουργία απλών ιεραρχιών κλάσεων με στοιχειώδεις περιορισμούς. Η OWL-DL είναι πολύ πιο εκφραστική από την OWL-Lite και βασίζεται στη περιγραφική λογική. Η περιγραφική λογική αποτελεί ένα αποφασίσιμο τμήμα της λογικής πρώτης τάξης, δηλαδή οι αλγόριθμοί της τερματίζουν σε πεπερασμένο χρόνο. Επομένως, με τη βοήθεια της DL, είναι δυνατός ο έλεγχος, για τυχόν ασυνέπειες, μιας οντολογίας που συμμορφώνεται με τις επιταγές της OWL-DL και παράλληλα ο αυτόματος υπολογισμός της ταξινομημένης ιεραρχίας. Για τους σκοπούς αυτής της διπλωματικής εργασίας, έχει επιλεγεί η χρήση της OWL-DL.

Τέλος, η OWL-Full είναι η πιο εκφραστική υπο-γλώσσα από τις τρεις και συνηθίζεται να εφαρμόζεται ιδίως σε περιπτώσεις όπου η υψηλή εκφραστικότητα τείνει να θεωρείται πιο σημαντική και από την εγγύηση της αποφασισιμότητας ή της υπολογιστικής πληρότητας μιας γλώσσας. Γι' αυτό το λόγο δεν είναι δυνατή η εκτέλεση αυτόματου συμπερασμού σε οντολογίες τέτοιου τύπου.

Μερικά από τα εργαλεία επεξεργασίας οντολογιών που διατίθενται είναι τα ακόλουθα:

- KAON, [6]
- OilEd, [7]
- Ontolingua, [8]
- OntoSaurus, [9]
- OntoEdit, [10]
- Protégé 2000, [11]
- WebOnto [12] και
- WebODE [13]

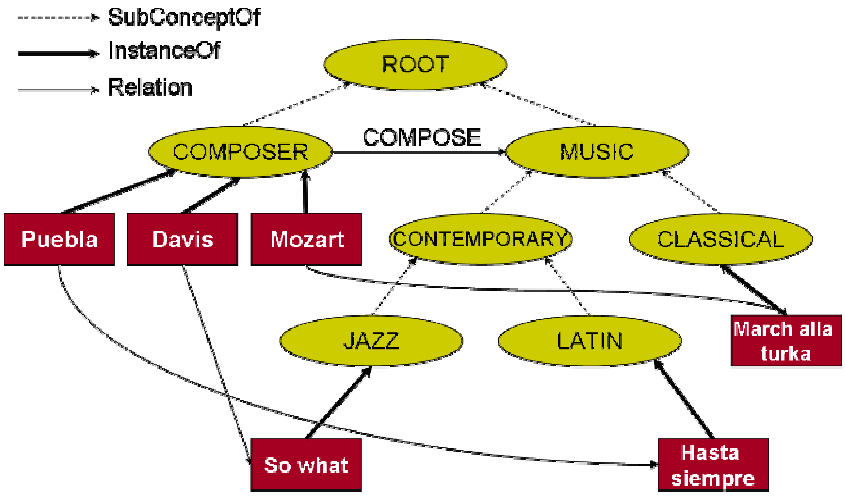
3 ΟΝΤΟΛΟΓΙΚΗ ΣΗΜΑΣΙΟΛΟΓΙΚΗ ΟΜΟΙΟΤΗΤΑ

Σ' αυτό το κεφάλαιο παρουσιάζεται το μαθηματικό μοντέλο, που υιοθετήθηκε έτσι ώστε να υπολογιστεί η ομοιότητα μεταξύ των κλάσεων και των στιγμιοτύπων και βασίστηκε στο [14]. Ως παράδειγμα θα χρησιμοποιηθεί η οντολογία και το αντίστοιχο μοντέλο μετα-δεδομένων που φαίνονται στην εικόνα 5. Ο όρος μετα-δεδομένα χρησιμοποιείται ως συνώνυμο του συνόλου των στιγμιοτύπων που υπάρχουν στην οντολογία. Προκειμένου να περιγραφεί η οντολογία, θα θεωρηθεί η ακόλουθη δομή:

$$O := \{C, \mathcal{P}, \mathcal{A}, \mathcal{H}^C, prop\}, \text{ Εξ. (1)}$$

που αποτελείται από:

- δύο ξένα σύνολα C και $\mathcal{P}$, των οποίων τα στοιχεία καλούνται κλάσεις και σχέσεις αντίστοιχα,
- μια ιεραρχία κλάσεων $\mathcal{H}^C$, η οποία αφορά μια κατευθυντική, μεταβατική σχέση ($\mathcal{H}^C \subseteq C \times C$), ($\mathcal{H}^C(C_1, C_2)$ σημαίνει ότι το C_1 είναι υπο-κλάση του C_2),
- μια συνάρτηση $prop$, η οποία συνδέει τις κλάσεις με τρόπο διαφορετικό από αυτόν που προκύπτει από το δέντρο ταξινόμησης.



Εικόνα 5: Πρότυπο Οντολογίας

Σύμφωνα με τη δομή που περιγράφηκε παραπάνω υπάρχουν τα σύνολα:

C:: {COMPOSER, MUSIC, CONTEMPORARY, CLASSICAL, JAZZ, LATIN},

$\mathcal{P}$: {COMPOSE}.

Αντίστοιχα ορίζεται η δομή των μετα-δεδομένων: $\mathcal{MD}$:= {O, I, $inst$, $instr$}, που αποτελείται από την οντολογία O, το σύνολο I των στιγμιοτύπων, τη συνάρτηση $inst$: C

$\rightarrow 2^{I}$ που σχετίζεται με το λεγόμενο *concept instantiation* και τη συνάρτηση $instr$: $\mathcal{P} \rightarrow 2^{III}$ που αφορά το *relation instantiation*.

Επομένως, σύμφωνα με τη δομή των μετα-δεδομένων μπορούν να διατυπωθούν οι εξής προτάσεις:

- I = {Puebla, Davis, Mozart, So what, Hasta Siempre, March Alla Turka},
- $inst$ (Puebla) = COMPOSER,
- $inst$ (Davis) = COMPOSER,
- $inst$ (Mozart) = COMPOSER,
- $inst$ (So what) = JAZZ,
- $inst$ (Hasta siempre) = LATIN,
- $inst$ (March alla turka) = CLASSICAL,
- COMPOSE(Puebla, Hasta siempre),
- COMPOSE(Davis, So what) και
- COMPOSE(Mozart, March alla turka).

Οι τρεις τελευταίες προτάσεις αφορούν τη συνάρτηση $instr$, δηλαδή τις σχέσεις που υπάρχουν μεταξύ των στιγμιοτύπων.

Προκειμένου να ομαδοποιηθούν με οποιοδήποτε τρόπο τα αντικείμενα, απαιτείται κάποιου είδους μέτρηση ομοιότητας μεταξύ αυτών των αντικειμένων. Στη συγκεκριμένη περίπτωση, τα αντικείμενα περιγράφονται μέσω των μετα-δεδομένων που βασίζονται στην οντολογία. Ο υπολογισμός της συνολικής ομοιότητας συνίσταται σε δύο επιμέρους μετρήσεις: μέτρηση *ομοιότητας ταξινομίας* και μέτρηση *ομοιότητας σχέσεων*.

Η *ομοιότητα ταξινομίας* προκύπτει από την ομοιότητα δύο στιγμιοτύπων με βάση τις αντίστοιχες κλάσεις και τις θέσεις τους στην ιεραρχία $\mathcal{H}^{C}$. Προκειμένου να υπολογιστεί αυτή η ομοιότητα, απαιτείται ο ορισμός και υπολογισμός της έννοιας *Upwards Cotopy* ($\mathcal{UC}$), για την οποία ισχύει $\mathcal{UC}(C_i, \mathcal{H}^{C})$:= $\{C_j \in C \mid \mathcal{H}^{C}(C_i, C_j) \wedge C_j = C_i\}$. Ουσιαστικά για να υπολογιστεί το $\mathcal{UC}$, προσμετρώνται όλες οι κλάσεις που βρίσκονται στη διαδρομή από τη συγκεκριμένη κλάση ως τη ρίζα του δέντρου ταξινόμησης, συμπεριλαμβανομένης και της ίδιας της κλάσης.

Παίρνοντας υπόψιν τον ορισμό του $\mathcal{UC}$, αντίστοιχα μπορεί να οριστεί και η έννοια του *Concept Match* ($\mathcal{CM}$), το οποίο αφορά το βαθμό ομοιότητας μεταξύ δύο κλάσεων. Ισχύει:

$$CM(C_1, C_2) := \frac{\left|(UC(C_1, H^C)) \cap (UC(C_2, H^C))\right|}{\left|(UC(C_1, H^C)) \cup (UC(C_2, H^C))\right|}, \text{ Εξ.(2)}$$

Η ομοιότητα ταξινομίας (*Taxonomy Similarity*) $\mathcal{TS}$ μεταξύ των αντίστοιχων στιγμιοτύπων ορίζεται ως εξής:

$$TS(I_1, I_2) = \begin{cases} 1 & if I_1 = I_2 \\ \dfrac{CM(C(I_1), C(I_2))}{2} & otherwise \end{cases}, \text{ Εξ.(3)}$$

Χρησιμοποιώντας την οντολογία του σχήματος 5 για το *Upwards Cotopy* ισχύουν οι ακόλουθες προτάσεις:

($\mathcal{UC}$({JAZZ}, $\mathcal{H}^C$)) = {JAZZ, CONTEMPORARY, MUSIC, ROOT},

($\mathcal{UC}$({LATIN}, $\mathcal{H}^C$)) = {LATIN, CONTEMPORARY, MUSIC, ROOT}

και για το Concept Match:

($\mathcal{CM}$ ({JAZZ}, {LATIN})) = 0.6

Επομένως, η ομοιότητα ταξινομίας μεταξύ των *So what*, *Hasta siempre* είναι:

($\mathcal{TS}$({So what}, {Hasta siempre}))= 0.3.

Ο αλγόριθμος που περιγράφηκε παραπάνω βασίζεται στην υπόθεση ότι εάν δύο στιγμιότυπα έχουν την ίδια σχέση με ένα τρίτο, είναι περισσότερο όμοια από δύο άλλα που έχουν σχέσεις με τελείως διαφορετικά στιγμιότυπα. Συνεπώς, η ομοιότητα δύο στιγμιοτύπων εξαρτάται από την ομοιότητα των στιγμιοτύπων με τα οποία συνδέονται.

Το τελευταίο σύνολο ορισμών αφορά τον υπολογισμό ομοιότητας μεταξύ σχέσεων. Θεωρώντας τα στιγμιότυπα I_1, $I_2 \in I$, από τον ορισμό της οντολογίας, είναι γνωστό ότι υπάρχει ένα σύνολο από σχέσεις P_1, που επιτρέπει στο I_1 να είναι πεδίο ορισμού, πεδίο τιμών ή και τα δύο σε μια σχέση (ανάλογα για το I_2 υπάρχει το P_2). Όσον αφορά την ομοιότητα σχέσεων ενδιαφέρον εμφανίζει η τομή $P_{CO} = P_1 \cap P_2$ των P_1, P_2, διότι οι διαφορές μεταξύ των P_1, P_2 καθορίζονται από τις ταξινομικές σχέσεις, οι οποίες έχουν ήδη ληφθεί υπόψιν στον υπολογισμό της ομοιότητας ταξινομίας. Το σύνολο σχέσεων P_{CO} συνίσταται σε σχέσεις που έχουν τα I_1, I_2 ως *range* ($P_{CO\text{-}I}$) και σχέσεις που έχουν τα I_1, I_2 ως *domain* ($P_{CO\text{-}O}$). Δοθείσης της δομής O := {C, $\mathcal{P}$, $\mathcal{A}$, $\mathcal{H}^C$, *prop*} που παρουσιάστηκε παραπάνω και θεωρώντας τα στιγμιότυπα I_1, $I_2 \in I$, είναι δυνατόν να διατυπωθούν οι ακόλουθες προτάσεις:

- $H^{trans} := \{(a,b) : \exists a_1 ... a_n \in C : H^C(a, a_1) ... H^C(a_n, b))\}$,
- $P_{CO-I_i}(I_i) := \left\{R : R \in P \wedge \left((C(I_i), range(R)) \in H^{trans}\right)\right\}$,

- $P_{CO-O_i}(I_i) := \{R : R \in P \wedge ((C(I_i), domain(R)) \in H^{trans})\}$,
- $P_{CO-I}(I_i, I_j) := P_{CO-I_i}(I_i) \cap P_{CO-I}(I_j)$ και
- $P_{CO-O}(I_i, I_j) := P_{CO-O_i}(I_i) \cap P_{CO-O}(I_j)$

Πρέπει να σημειωθεί ότι για μια δεδομένη οντολογία με μια σχέση P_X υπάρχει ελάχιστη θετική ομοιότητα μεταξύ δύο στιγμιοτύπων, τα οποία είναι είτε η πηγή (*source*) είτε ο προορισμός (*target*) μιας σχέσης στιγμιοτύπων – $MinSim_{s(P_X)}$ και $MinSim_{t(P_X)}$ αντίστοιχα. Αγνοώντας το γεγονός αυτό, είναι δυνατόν ν' αυξηθεί η ομοιότητα δύο στιγμιοτύπων που έχουν σχέσεις με τα «πιο διαφορετικά» στιγμιότυπα, συγκρινόμενα με δύο στιγμιότυπα που δεν ορίζουν αυτή τη σχέση. Για κάθε σχέση $P_n \in P_{CO-O}$ και κάθε στιγμιότυπο I_1 υπάρχει ένα σύνολο από σχέσεις στιγμιοτύπων $P_n(I_i, I_x)$. Το σύνολο I_x αυτών των στιγμιοτύπων καλείται σύνολο των συσχετιζόμενων στιγμιοτύπων (A_S) και αναπαρίσταται ως εξής:

$$A_S(P, I) := \{I_x : I_x \in \mathrm{I} \wedge \mathrm{P}(\mathrm{I}, \mathrm{I}_x)\}, \text{ Εξ.(4)}$$

Προκειμένου να συγκριθούν τα στιγμιότυπα I_1, I_2 όσον αφορά τη σχέση P_n, αρκεί να συγκριθούν τα $A_s(P_n, I_1)$ και $A_s(P_n, I_1)$. Αυτό μπορεί να συμβεί χρησιμοποιώντας τον παρακάτω τύπο, που δίνει την ομοιότητα για μια σχέση:

$$OR(I_1, I_2, P) = \begin{cases} MinSim_{t(P)} & if A_s(P, I_1) = \emptyset \vee A_s(P, I_2) = \emptyset \\ \left(\frac{\sum_{(a \in A_s(P, I_1))} \max\{sim(a, b) | b \in A_s(P, I_2)\}}{|A_s(P, I_1)|} \right) & if |A_s(P, I_1) \geq |A_s(P, I_2)| \\ \left(\frac{\sum_{(a \in A_s(P, I_2))} \max\{sim(a, b) | b \in A_s(P, I_1)\}}{|A_s(P, I_2)|} \right) & otherwise \end{cases}, \text{ Εξ.(5)}$$

Τελικά, τα αποτελέσματα για όλες τις $P_n \in P_{CO-O}$ και $P_n \in P_{CO-I}$ συνδυάζονται υπολογίζοντας τον αριθμητικό τους μέσο:

$$RS(I_1, I_2) := \frac{\sum_{p \in P_{CO-I}} OR(I_1, I_2, p) + \sum_{p \in P_{CO-O}} OR(I_1, I_2, p)}{|P_{CO-I}| + |P_{CO-O}|}, \text{ Εξ.(6)}$$

Το μοναδικό πρόβλημα έγκειται στην αναδρομική φύση του υπολογισμού ομοιότητας που ενδέχεται να οδηγήσει σε άπειρους κύκλους. Η λύση δίνεται θέτοντας ένα μέγιστο βάθος για την αναδρομή.

Προκειμένου να γίνει περισσότερο αντιληπτή η χρήση των τύπων, ας υποτεθεί ότι συγκρίνονται τα στιγμιότυπα *Davis*, *Mozart* της γνωστής οντολογίας του σχήματος 1. Το

σύνολο των κοινών σχέσεων περιέχει μόνο τη σχέση *COMPOSE* (εξάλλου είναι η μοναδική σχέση στην οντολογία). Το στιγμιότυπο *Davis* συνδέεται μέσω της σχέσης *COMPOSE* με το *So what* και αντίστοιχα το *Mozart* με το *March alla turka*. Αν υποτεθεί ότι το μέγιστο βάθος αναδρομής είναι 1, τότε η ομοιότητα σχέσης είναι 0.2. Άρα η ολική ομοιότητα είναι για α = 0.7,όπου α είναι βαθμός ταξινομίας,

sim(Davis, Mozart) = α *$\mathcal{IS}$(Mozart, Davis) + (1-α)*$\mathcal{RS}$(Davis, Mozart) = 0.7*0.5 + 0.2*0.3 = 0.35 + 0.06 = 0.41.

4 ΛΕΞΙΚΟΓΡΑΦΙΚΗ ΟΜΟΙΟΤΗΤΑ ΜΕ ΧΡΗΣΗ ΘΕΩΡΙΑΣ ΑΣΑΦΕΙΑΣ

Σ' αυτό το κεφάλαιο παρουσιάζεται μια προσέγγιση μέτρησης λεξικογραφικής ομοιότητας, η οποία επιχειρείται με τη βοήθεια μιας μεθόδου ταξινόμησης κειμένου που χρησιμοποιεί τους τελεστές της Θεωρίας Ασάφειας [15], [16].

4.1 Ασαφής Σχέση Όρου-Κατηγορίας

Θεωρούνται δύο πεπερασμένα διακριτά σύνολα: ένα σύνολο όρων **T** = {t_1, t_2, …, t_n} και ένα σύνολο κατηγοριών **C** = {c_1, c_2, …, c_m}, καθώς επίσης και ένα σύνολο που αποτελείται από n έγγραφα και τις αντίστοιχες κατηγορίες ταξινόμησης **D** = {<d_1, $c(d_1)$>, <d_2, $c(d_2)$>, …, <d_n, $c(d_n)$>}. Ξεκινώντας από τον αλγόριθμο του Rocchio [17], μετράται η ομοιότητα μεταξύ ενός *εγγράφου* και μιας *κατηγορίας*, χρησιμοποιώντας τον συντελεστή συνημιτόνου. Στη συνέχεια αναπτύσσεται μια ασαφής σχέση μεταξύ όρων και κατηγοριών, από όπου προκύπτει ένα σύνολο με τα ποσοστά που αντιπροσωπεύουν το κατά πόσο μια λέξη ανήκει σε μια κατηγορία. Σύμφωνα με αυτήν τη σχέση, υπολογίζεται η ομοιότητα ενός εγγράφου και λεξικογραφικής κατηγορίας, χρησιμοποιώντας τους τελεστές τομής και ένωσης της Θεωρίας Ασάφειας. Η ομοιότητα αυτή αντιπροσωπεύει το ποσοστό για το οποίο ένα έγγραφο ανήκει σε μια κατηγορία.

Επιπρόσθετα, ορίζεται μια ασαφής δυαδική σχέση προκειμένου να περιγραφεί η συσχέτιση μεταξύ όρου και κατηγορίας: R: **T** × **C** → [0,1], όπου το R(t_i, c_j) υποδηλώνει το βαθμό συγγένειας του όρου t_i και της κατηγορίας c_j.

Παράλληλα, το κάθε έγγραφο αναπαρίσταται από το σύνολο: **d** = {<t_1, w_1>, <t_1, w_1>, …, <t_m, w_m>}, όπου w_j είναι η συχνότητα εμφάνισης του όρου t_j στο συγκεκριμένο έγγραφο. Επομένως, δοθέντος ενός συνόλου από αρχεία **D**, ο βαθμός της συσχέτισης R(t_i, c_j), που συμβολίζεται ως $\mu_R(t_i, c_j)$ υπολογίζεται σύμφωνα με τον τύπο:

$$\mu_R(t_i, c_j) = \frac{\sum_{\{w_i \in d_k \wedge d_k \in D \wedge c(d_k)=c_j\}} w_i}{\sum_{\{w_i \in d_k \wedge d_k \in D\}} w_i} \quad \text{, Εξ.(6)}$$

Συνεπώς, το κατά πόσο ένας όρος t_i ανήκει σε μια κατηγορία c_j ισούται με τη μονάδα, αν ο συγκεκριμένος όρος εμφανίζεται μόνο σε αρχεία που ομαδοποιούνται ως κατηγορία c_j. Αντίστροφα, αν ο όρος εμφανίζεται σε έγγραφα διαφορετικών κατηγοριών, τότε υπάρχει μια «ποινή» για τον όρο, έτσι ώστε η τιμή της $\mu_R(t_i, c_j)$ για κάθε κατηγορία να παραμένει χαμηλή. Ουσιαστικά, η τιμή της $\mu_R(t_i, c_j)$ αποτελεί την κατανομή του όρου t_i στις διάφορες κατηγορίες.

Στους πίνακες 1, 2, 3 απεικονίζεται ένα τυπικό παράδειγμα των όσων περιγράφηκαν παραπάνω. Ας υποτεθεί ότι τα αρχεία d_1 και d_2 ανήκουν στη κατηγορία c_1 ενώ τα d_3 και d_4 ανήκουν στη κατηγορία c_2. Στον πίνακα 1 φαίνεται η κατανομή του κάθε όρου σε κάθε έγγραφο. Λαμβάνοντας υπόψιν αυτήν την πληροφορία, ο πίνακας 2 δείχνει τη συχνότητα εμφάνισης του κάθε όρου όσον αφορά τις δύο κατηγορίες. Τέλος, στον πίνακα 3 απεικονίζονται οι τιμές της $\mu_R(t_i, c_j)$, δηλαδή το ποσοστό που ένας όρος ανήκει σε μια κατηγορία.

Doc	Term						Cat
	t_1	t_2	t_3	t_4	t_5	t_6	
d_1	2	1	2				c_1
d_2	3	2				1	c_1
d_3			1	2	3		c_2
d_4				3	1	1	c_2

Πίνακας 1: Συχνότητα εμφάνισης όρου ανά έγγραφο

Term	Category	
	c_1	c_2
t_1	5	0
t_2	3	0
t_3	2	1
t_4	0	5
t_5	0	4
t_6	1	1

Πίνακας 2: Συχνότητα εμφάνισης όρου ανά κατηγορία

Term	Category	
	c_1	c_2
t_1	1	0
t_2	1	0
t_3	0.67	0.33
t_4	0	1
t_5	0	1
t_6	0.5	0.5

Πίνακας 3: Τιμές της $\mu_R(t_i, c_j)$

4.2 Ασαφής Μέτρηση Λεξικογραφικής Ομοιότητας

Εφόσον οι τιμές της συνάρτησης $\mu_R(t_i, c_j)$ έχουν υπολογιστεί, απαιτείται μια μέθοδος μέτρησης ομοιότητας μεταξύ ενός εγγράφου και των λεξικογραφικών κατηγοριών, που αναπαρίστανται μέσω των τιμών της $\mu_R(t_i, c_j)$ όσον αφορά τους όρους που ανήκουν στην ίδια κατηγορία. Η μέθοδος υλοποιείται με τη βοήθεια των τελεστών της Θεωρίας Ασάφειας.

Ας θεωρηθεί το έγγραφο **d** = {<t_1, $\mu_d(t_1)$>, <t_2, $\mu_d(t_2)$>, …, <t_n, $\mu_d(t_m)$>}, όπου η τιμή $\mu_d(t_i)$ αντιπροσωπεύει το κατά πόσο ο όρος t_i ανήκει στο **d**. Δοθείσης της σχέσης R(T,C) η ομοιότητα μεταξύ ενός εγγράφου d και μιας κατηγορίας c_j υπολογίζεται από τον τύπο:

$$sim(d,c_j)=\frac{\sum_{t\in d}\mu_R(t,c_j)\otimes\mu_d(t)}{\sum_{t\in d}\mu_R(t,c_j)\oplus\mu_d(t)}$$, Εξ. (7)

όπου τα $\otimes$ και $\oplus$ υποδηλώνουν την ασαφή τομή και ένωση αντίστοιχα. Στον πίνακα 4 συνοψίζονται τα συνηθέστερα ζεύγη τομή και ένωσης, για τα οποία από εδώ και πέρα θα χρησιμοποιούνται οι συμβολισμοί *t-norm* και *t-conorm* αντίστοιχα.

t-norm t(x,y)	t-conorm s(x,y)
Einstein Product: $\frac{x\cdot y}{2-[x+y-(x\cdot y)]}$	Einstein Sum: $\frac{x\cdot y}{[1+x\cdot y]}$
Algebraic Product: $x\cdot y$	Algebraic Sum: $x+y-x\cdot y$
Hamacher Product: $\frac{x\cdot y}{x+y-(x\cdot y)}$	Hamacher Sum: $\frac{x+y-2xy}{1-(x\cdot y)}$
Minimum: $\min\{x\cdot y\}$	Maximum: $\max\{x\cdot y\}$
Drastic Product: $\min\{x\cdot y\}$ *if* $\max\{x,y\}=1$; 0 *if* $x,y<1$	Drastic Sum: $\max\{x\cdot y\}$ *if* $\min\{x,y\}=0$; 1 *if* $x,y>0$
Bounded Difference: $\max\{0,x+y-1\}$	Bounded Difference: $\min\{1,x+y\}$

Πίνακας 4:Τυπικά ζεύγη τελεστών τομής και ένωσης

Υπάρχουν δύο εναλλακτικές όσον αφορά τον υπολογισμό του $\mu_d(t_i)$. Σύμφωνα με την πρώτη, το έγγραφο που πρόκειται να κατηγοριοποιηθεί αναπαρίσταται ως ένα δυαδικό διάνυσμα, στο οποίο το $\mu_d(t_i)$ παίρνει την μοναδιαία τιμή για κάθε όρο που εμφανίζεται τουλάχιστον μια φορά στο έγγραφο, ενώ ισούται με 0 αν ο συγκεκριμένος όρος δεν υπάρχει στο έγγραφο. Εφόσον μόνο οι όροι που εμφανίζονται στο d λαμβάνονται υπόψιν στον υπολογισμό της ασαφούς ομοιότητας, θα ισχύει: $\mu_d(t)=1,\ \forall t\in D$. Κατά τη δεύτερη εναλλακτική, στη $\mu_d(t_i)$ ανατίθεται μια πιο εκλέπτυσμένη τιμή, που καθορίζεται βάσει της σχετικής συχνότητας εμφάνισης του όρου t στο έγγραφο d. Συγκεκριμένα, η

$\mu_d(t_i)$ που παρουσιάζει τη μεγαλύτερη τιμή συχνότητας εμφάνισης, τίθεται ίση με 1. Τελικά, οι τιμές της $\mu_d(t_i)$ υπολογίζονται σύμφωνα με τη σχέση:

$$\mu_d(t_i) = \frac{w_i}{\max_{t \in d}\{w\}}$$, Εξ. (8)

Με βάση την υλοποίηση που έγινε για τον υπολογισμό της ασαφούς μέτρησης της λεξικογραφικής ομοιότητας, που ερμηνεύει τον βαθμό συγγένειας ενός εγγράφου σε μία κατηγορία, λαμβάνονται τα παρακάτω αποτελέσματα. Δοθείσης της ερώτησης εύρεσης της υπηρεσίας που περιγράφεται από το έγγραφο *Implicit-Printing* = {File (2),Printer(2), Wireless(4), Nearest(4)}, (η παρένθεση δηλώνει τον αριθμό εμφάνισης του κάθε όρου) λαμβάνεται η μονάδα συγγένειας με τις κατηγορίες PeripheralManagement, DataManagement, και LocationBasedService, μεσώ της μήτρας *min-max* και της μήτρας *algebraic* (Πίνακας 4).

Αποτελέσματα υλοποίησης

T-Norm {min-max}

>**Implicit-Printing**

Implicit-Printing query has Fuzzy Membership in **PeripheralManagement** is 0.42879663688912245

Implicit-Printing query has Fuzzy Membership in **DataManagement** is 0.14176470588235293

Implicit-Printing query has Fuzzy Membership in **LocationBasedService** is 0.6042245989304813

T-Norm {Algebraic}

>**Implicit-Printing**

Implicit-Printing query has Fuzzy Membership in **PeripheralManagement** is 0.3113093802748975

Implicit-Printing query has Fuzzy Membership in **DataManagement** is 0.10756062767475034

Implicit-Printing query has Fuzzy Membership in **LocationBasedService** is 0.5129996217374858

5 ΠΡΩΤΟΚΟΛΛΑ ΕΥΡΕΣΗΣ ΥΠΗΡΕΣΙΩΝ

Οι τεχνολογίες για εύρεση υπηρεσιών (SLP, Salutation, UPnP, Bluetooth SDP, Jini) παρέχουν το πλαίσιο μέσα στο οποίο μπορούν να βρεθούν λύσεις για διάφορα προβλήματα που αφορούν εφαρμογές *client/server*, όπως είναι η διαλειτουργικότητα. Λέγοντας πλαίσιο εννοούμε ένα σύνολο από πρωτόκολλα, τα οποία επιτρέπουν την ανάπτυξη δυναμικών εφαρμογών *client/server*. Κατά τη διάρκεια του σχεδιασμού τέτοιων εφαρμογών, ανακύπτουν καίρια ερωτήματα, όπως:

- τί τύπου υπηρεσίες είναι διαθέσιμες,
- πού «βρίσκονται» αυτές οι υπηρεσίες,
- τί χρειάζεται ένας *client* προκειμένου να χρησιμοποιήσει μια υπηρεσία και
- ποιό πρωτόκολλο χρησιμοποιείται μεταξύ *client* και *service*;

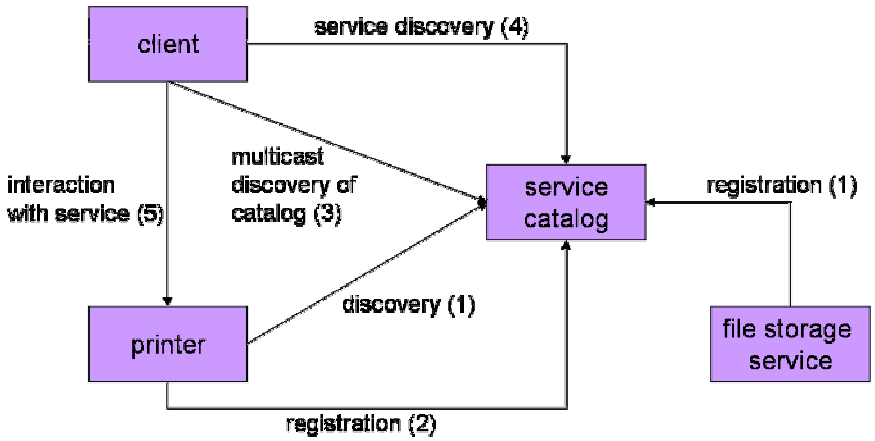
Το μοντέλο που ακολουθείται είναι η δυναμική αλληλεπίδραση μεταξύ *clients* και *services*. Οι *clients* αναζητούν τις *υπηρεσίες* που χρειάζονται, βασιζόμενοι στον τύπο της υπηρεσίας (π.χ. εκτύπωση αρχείου, αποθήκευση αρχείου) ή σε ιδιαίτερα χαρακτηριστικά που αφορούν τον κατασκευαστή, το κόστος της υπηρεσίας, τα οποία συγκροτούν αυτό που ονομάζεται *service profile*. Οι υπηρεσίες ανακοινώνουν την εμφάνισή τους όταν εισέρχονται στο δίκτυο και (αν είναι δυνατόν) την αποχώρησή τους όταν εξέρχονται από αυτό. Ειδικοί κατάλογοι αποθηκεύουν στοιχεία που αφορούν τις διαθέσιμες υπηρεσίες, ενώ άλλες οντότητες τύπου *garbage collector* είναι επιφορτισμένες με την εκκαθάριση του συστήματος, διαγράφοντας υπηρεσίες, οι οποίες δεν διατίθενται πλέον στο δίκτυο.

Επιπρόσθετα, οι τεχνολογίες εύρεσης υπηρεσιών [17] τυποποιούν τα περιβάλλοντα όπου αναπτύσσονται εφαρμογές *client/server* και οι υλοποιήσεις των πρωτοκόλλων για service discovery παρέχουν εργαλεία λογισμικού, τα οποία κάνουν την αλληλεπίδραση μεταξύ *clients* και *services* πολύ πιο δυναμική, απ’ ότι είναι στα σημερινά συστήματα.

Σε όλα τα πλαίσια που προσφέρουν εύρεση υπηρεσιών υπάρχουν οι οντότητες *client* και *service*, που παρέχουν συγκεκριμένη λειτουργικότητα. Οι *clients* αναζητούν απευθείας τις απαιτούμενες υπηρεσίες ή συμβουλευόμενοι τους σχετικούς καταλόγους, όπου διατηρούνται οι διαθέσιμες υπηρεσίες.

Μια απόπειρα αναζήτησης εν γένει ταξινομεί τις υπηρεσίες, ανάλογα με τον τύπο τους και προαιρετικά λαμβάνει χαρακτηριστικά όπως ο κατασκευαστής ή ο σειριακός αριθμός της υπηρεσίας. Ο *client* χρειάζεται λίγη πληροφορία για το περιβάλλον του· μπορεί να εντοπίσει υπηρεσίες δυναμικά με ελάχιστο *configuration*. Ο τυπικός τρόπος για την υποστήριξη της αναζήτησης υπηρεσίας είναι *multicast*. Δηλαδή, όταν μια υπηρεσία

«εισέρχεται» σ' ένα δίκτυο εκτελεί το λεγόμενο *service advertisement*, είτε απευθείας στους *clients* είτε στους καταλόγους υπηρεσιών, προκειμένου να «διαφημίσει» μια υπηρεσία. Στην εικόνα 6 απεικονίζεται ένα κλασικό παράδειγμα εύρεσης υπηρεσίας. Μια υπηρεσία αποθήκευσης αρχείων, γνωρίζοντας τη θέση ενός καταλόγου υπηρεσιών, καταχωρείται. Παράλληλα, μια υπηρεσία εκτύπωσης ανακαλύπτει δυναμικά τον κατάλογο (χρησιμοποιώντας *multicast*) και επίσης καταχωρείται. Στη συνέχεια ένας πελάτης που χρειάζεται εκτυπωτή, βρίσκει τον κατάλογο, εντοπίζει την υπηρεσία εκτύπωσης και τελικά αλληλεπιδρά με την υπηρεσία.



Εικόνα 6: Παράδειγμα εύρεσης υπηρεσιών

Συμπερασματικά, οι τεχνολογίες για εύρεση υπηρεσιών προσφέρουν τα περισσότερα τμήματα, που απαιτούνται, προκειμένου να δομηθούν δυναμικά κατανεμημένα συστήματα επικοινωνιών, γεγονός που βοηθά στο σχεδιασμό και σε θέματα διαχείρισης των εφαρμογών *client*/*server*. Ακόμα αυτές οι τεχνολογίες επιτρέπουν στις υπηρεσίες που εισάγονται στο δίκτυο να διαμορφωθούν και να χρησιμοποιηθούν με ελάχιστη ανθρώπινη παρέμβαση. Σ' ένα δίκτυο που βασίζεται σε τέτοιου είδους τεχνολογίες οτιδήποτε μπορεί να θεωρηθεί *client* ή *service*· οι *clients* χρειάζονται «κάτι» και οι υπηρεσίες το προσφέρουν. Για παράδειγμα, ένας επεξεργαστής κειμένου μπορεί να θεωρηθεί ένας τυπικός *client* ως προς έναν εκτυπωτή, ενώ αντίστοιχα σ' αυτήν την περίπτωση ο εκτυπωτής παίζει το ρόλο ενός *service*. Ακόμα μια ψηφιακή κάμερα μπορεί να παίξει το ρόλο του *service*, παρέχοντας φωτογραφίες, αλλά μπορεί παράλληλα να ζητήσει ως *client* τη βοήθεια κάποιου μηχανισμού φωτισμού, αν μια φωτογραφία κριθεί υποφωτισμένη. Ουσιαστικά, οι τεχνολογίες για *service discovery* θέτουν ζητήματα όπως «δεν ξέρω που βρίσκεσαι» ή «δεν ξέρω πως να μιλήσω μαζί

σου». Το ζητούμενο είναι η επικοινωνία μεταξύ των «καταναλωτικών» clients και των διαθέσιμων υπηρεσιών.

Ένα πρόβλημα που συναντούν οι χρήστες με όλο και αυξανόμενη συχνότητα σχετίζεται με την εγκατάσταση, προσαρμογή και διαχείριση των περιφερειακών συσκευών. Πρόκειται για ένα κλασικό *client*/*server* πρόβλημα, το οποίο εντείνεται όταν σ' ένα δικτυακό περιβάλλον συνυπάρχουν φορητοί υπολογιστές, εκτυπωτές, σαρωτές, ασύρματες συσκευές, εξωτερικοί σκληροί δίσκοι, ψηφιακές κάμερες και άλλο παρεμφερές υλικό. Είναι προφανές ότι άπειροι χρήστες υπολογιστών μάλλον θα απογοητευτούν στην προσπάθειά τους να εξασφαλίσουν τη σωστή λειτουργία και αλληλεπίδραση ενός τέτοιου πλήθους συσκευών. Για παράδειγμα, η διαδικασία αντικατάστασης μιας παλιάς συσκευής από μια καινούρια στο δίκτυο περιλαμβάνει «φυσική» εγκατάσταση, αφαίρεση των *drivers* της παλιάς συσκευής, έλεγχο για την καταλληλότητα των νέων *drivers*, εγκατάσταση τους και μια σειρά από άλλες λειτουργίες. Η διαδικασία αυτή συνήθως καταλήγει σε δυσλειτουργική συνύπαρξη με τις ήδη υπάρχουσες συσκευές.

Στην περίπτωση που η χρήση των κινητών συσκευών αυξάνεται, το πρόβλημα της εγκατάστασης περιφερειακών εμφανίζεται σοβαρότερο, μια που η δυναμική συμπεριφορά γίνεται πλέον ο κανόνας. Επιπρόσθετα, όλο και περισσότεροι χρήστες καταφεύγουν στην αγορά πολλών διαφορετικών κινητών συσκευών, αφού δεν υπάρχει μία που να καλύπτει όλες τις ανάγκες. Τα κινητά τηλέφωνα ενδείκνυνται για φωνητική συνομιλία, αλλά το περιορισμένο μέγεθός τους δυσχεραίνει τη χρήση *email* και *web browsing*. Από την άλλη μεριά, οι φορητοί υπολογιστές έχουν σαφώς μεγαλύτερες δυνατότητες, αλλά παρουσιάζουν δυσκολία στη μεταφορά. Τα *palmtops* αποτελούν μια ενδιάμεση κατάσταση. Οι περισσότερες από αυτές τις συσκευές δεν μπορούν να υποστηρίξουν μεγάλη γκάμα περιφερειακών και γι' αυτό το λόγο υποχρεωτικά στηρίζονται σε γειτονικές συσκευές για λειτουργίες όπως αποθήκευση, εκτύπωση, πρόσβαση στο δίκτυο με υψηλές ταχύτητες. Βέβαια, κάποιες από αυτές τις υπηρεσίες μπορούν να προσφερθούν από τον ίδιο τον εξοπλισμό του χρήστη – για παράδειγμα ένα κυψελωτό τηλέφωνο μπορεί να εξασφαλίσει σύνδεση με το δίκτυο σ' έναν υπολογιστή τύπου *palmtop* – αλλά σε πολλές περιπτώσεις αυτές οι υπηρεσίες παρέχονται από μια τρίτη οντότητα.

Συχνά η αλληλεπίδραση μεταξύ της κινητής συσκευής και του απαιτούμενου περιφερειακού χρειάζεται να γίνεται στιγμιαία. Έτσι, στην περίπτωση που ένας *πράκτορας* επιθυμεί να τυπώσει ένα έγγραφο, που είναι αποθηκευμένο σ' ένα *palmtop* χρησιμοποιώντας τον εκτυπωτή δικτύου, πρέπει να βρεθεί ο τρόπος ώστε να

συνεργαστούν οι δύο συσκευές χωρίς να δαπανηθεί σημαντικός χρόνος. Μια παρόμοια κατάσταση θα παρουσιαστεί σ' ένα μελλοντικό καφέ, όπου διατίθενται υπηρεσίες όπως εκτύπωση, αποθήκευση, πρόσβαση στο δίκτυο με υψηλές ταχύτητες. Ένα τέτοιο περιβάλλον φαντάζει ως τεχνολογική ουτοπία, αλλά γρήγορα αμβλύνεται ο ενθουσιασμός ενός υποτιθέμενου χρήστη, όταν ανακαλύψει ότι είναι υποχρεωμένος να «κατεβάσει» και να εγκαταστήσει τους κατάλληλους *drivers* για τη συγκεκριμένη συσκευή. Επιπλέον, το να πρέπει να διαπιστώσει ποιες υπηρεσίες είναι διαθέσιμες αποτελεί μια δυσάρεστη κατάσταση.

Εν γένει οι άνθρωποι είναι πρόθυμοι να δουλέψουν παράλληλα σε διαφορετικά δίκτυα, αλλά απογοητεύονται αν κάτι τέτοιο αποδειχθεί επίπονο. Η τεχνολογία του *service discovery* επιτρέπει στις υπηρεσίες να γνωστοποιηθούν με σχετική ευκολία και οι *clients* έχουν τη δυνατότητα να ξεκινήσουν να εργάζονται, αφού οι διεπαφές μεταξύ *clients* και *services* είναι τυποποιημένες.

Παρότι σε χαμηλό επίπεδο οι διάφορες τεχνολογίες για εύρεση υπηρεσιών έχουν διαφορετικές σχεδιαστικές φιλοσοφίες, είναι γεγονός ότι σε υψηλότερο επίπεδο παρουσιάζουν κοινά χαρακτηριστικά, τα οποία περιγράφονται εν συντομία παρακάτω.

- ***Discovery of services***: οι ζητούμενες υπηρεσίες μπορούν να εντοπιστούν από τους *clients*, έχοντας ελάχιστη γνώση όσον αφορά τη δομή του δικτύου. Οι *clients* μπορούν ν' αναζητήσουν τις υπηρεσίες σύμφωνα με τον τύπο τους ή άλλα χαρακτηριστικά. Η αναζήτηση ποικίλει ανάλογα με την εκάστοτε τεχνολογία.
- ***Service "subtyping"***: οι *clients* ενδέχεται περιστασιακά να ενδιαφερθούν για μια πολυ εξειδικευμένη υπηρεσία, όπως είναι για παράδειγμα η εκτύπωση ενός διαφημιστικού εντύπου από έναν έγχρωμο εκτυπωτή υψηλής ανάλυσης. Το *Service "subtyping"* επιτρέπει τη χρήση τόσο των στοιχειωδών λειτουργιών ενός συνηθισμένου εκτυπωτή, όσο και κάποιων πιο «εξεζητημένων». Ένας *client* μπορεί να είναι όσο συγκεκριμένος επιθυμεί κι έτσι ν' ανακαλύψει την εφαρμογή με τα απαιτούμενα χαρακτηριστικά.
- ***Service insertion and advertisement***: όταν μια υπηρεσία «εισέρχεται» σ'ένα δίκτυο απαιτεί ελάχιστη ανθρώπινη διαμόρφωση και διαφημίζει τη διαθεσιμότητά της, είτε κατευθείαν στους *clients* είτε σε *servers*, οι οποίοι διατηρούν λίστες με τις υπάρχουσες υπηρεσίες. Αντίστοιχα, όταν μια υπηρεσία παύει να ισχύει, ενημερώνει το δίκτυο για την αποχώρησή της. Μια πρωταρχική διαφορά μεταξύ των τεχνολογιών για εύρεση υπηρεσιών

και των τεχνολογιών που βασίζονται σε στατική πληροφορία είναι πως οι πρώτες παρέχουν δυναμική ενημέρωση των καταλόγων με τις υπάρχουσες υπηρεσίες στο δίκτυο. Αντίθετα, υπηρεσίες όπως το DNS ή το DHCP βασίζονται σε στατικές βάσεις δεδομένων, στις οποίες έχουν πρόσβαση μόνο οι διαχειριστές. Επιπρόσθετα, οι τεχνολογίες για εύρεση υπηρεσιών προσφέρουν πολύ πιο ισχυρές δυνατότητες αναζήτησης απ' ότι τα στατικά σχήματα.

- ***Service browsing***: οι *clients* έχουν τη δυνατότητα να πλοηγηθούν στις λίστες των διαθέσιμων υπηρεσιών. Η όλη διαδικασία μπορεί να διεξαχθεί με τη βοήθεια ένος γραφικού περιβάλλοντος, όπου ο χρήστης μπορεί να επιλέξει τις εφαρμογές που τον ενδιαφέρουν.
- ***Catalogs of available services***: μερικές τεχνολογίες για εύρεση υπηρεσιών όπως το UPnP, είναι εξ ορισμού *peer-to-peer*, επιτρέποντας σε *clients* και *services* την απευθείας επικοινωνία μεταξύ τους, δηλαδή χωρίς τη μεσολάβηση κάποιου *communication broker*. Άλλες παρεμφερείς τεχνολογίες, όπως το Jini, συστήνουν καταλόγους με τις διαθέσιμες υπηρεσίες, ενώ το SLP μπορεί να λειτουργήσει τόσο με τη βοήθεια καταλόγων όσο και χωρίς αυτούς. Στα πρωτόκολλα για εύρεση υπηρεσιών που υποστηρίζουν τη χρήση καταλόγων οι υπηρεσίες καταχωρούνται στις λίστες και οι *clients* αντλούν απο εκεί τις απαιτούμενες πληροφορίες για τις υπηρεσίες που τους ενδιαφέρουν. Από τη μια μεριά, οι κατάλογοι μπορούν να μειώσουν δραματικά την *multicast* κίνηση, αφού απαλλάσσουν τους *clients* από τον κόπο να ανακαλύψουν μόνοι τους τις υπηρεσίες, πλημμυρίζοντας το δίκτυο με αιτήσεις. Από την άλλη μεριά οι κατάλογοι επιβαρύνουν το δικτύο με μια επιπλέον οντότητα, η οποία χρήζει ειδικής διαχείρισης. Τέλος, οι κατάλογοι παρέχουν εύρεση υπηρεσιών και πέρα από την επιφάνεια που καλύπτει η ακτίνα του *multicast*, επιτρέποντας την προσθήκη πολλών τομέων στην περιοχή που εφαρμόζεται το εύρεση υπηρεσιών .
- ***Eventing***: το *eventing* αφορά την ασύγχρονη ειδοποίηση για σημαντικές αλλαγές στο δίκτυο, όπως η εμφάνιση μιας καινούριας υπηρεσίας, ο τερματισμός μιας παλαιότερης ή η αλλαγή της κατάστασης μιας υπάρχουσας, π.χ. η έλλειψη χαρτιού σ' έναν εκτυπωτή. Το *eventing* κάνει πιο εύκολη τη ζωή του χρήστη, εξαλείφοντας την ανάγκη για

παρακολούθηση και ανακάλυψη των διαφόρων αλλαγών, που σχετίζονται με τις υπηρεσίες που εμφανίζονται στο δίκτυο.

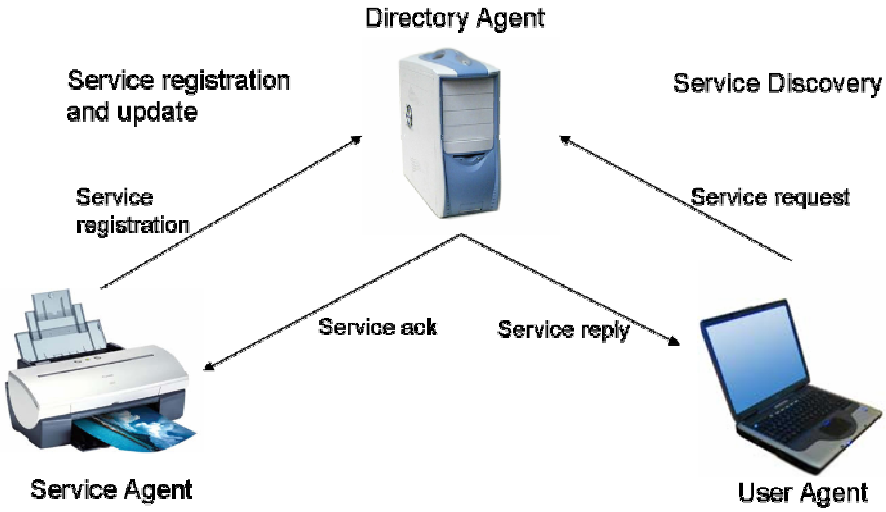
- ***Garbage collection***: στο πλαίσιο του εύρεση υπηρεσιών περιέχεται ένας μηχανισμός, ο οποίος αναλαμβάνει τη διαγραφή από τις λίστες των υπηρεσιών που έχουν λήξει. Χωρίς τον *garbage collector*, πέρα από τη σπατάλη όσον αφορά την αποθήκευση των υπηρεσιών στους καταλόγους, εμφανίζονται πρόσθετα προβλήματα όταν ένας *client* επιχειρεί να χρησιμοποιήσει μη υπάρχουσα υπηρεσία ή όταν μια υπηρεσία συνεχίζει να εκτελεί λειτουργίες για λογαριασμό κάποιων *clients*, οι οποίοι δεν υφίστανται πλέον στο δίκτυο. Το μοντέλο που εφαρμόζεται είναι απλό: σε κάθε υπηρεσία ανατίθεται ένα συγκεκριμένο χρονικό διάστημα, το λεγόμενο *lease*, μετά την πάροδο του οποίου η υπηρεσία παύει να υφίσταται. Προκειμένου να συνεχιστεί η παρουσία της στο δίκτυο, πρέπει να ζητηθεί η παράταση της λήξης της προθεσμίας και να ανανεωθεί ο χρόνος παραμονής. Η διαδικασία αυτή ονομάζεται *lease renewal*.

Έχει αναπτυχθεί ένα πλήθος τεχνολογιών για εύρεση υπηρεσιών, οι κυριότερες από τις οποίες είναι το SLP, UPnP, *Bluetooth SDP*, *Salutation* και *Jini* [18]. Στη συνέχεια θα περιγραφούν με συντομία τα τέσσερα πρώτα πρωτόκολλα, ενώ το *Jini* θα παρουσιαστεί πιο διεξοδικά.

5.1 Το Πρωτόκολλο Εύρεσης Υπηρεσιών SLP

Το SLP είναι ένα τυποποιημένο πρωτόκολλο που αναπτύχθηκε από την IETF και συγκεκριμένα από το IETF Svrloc working group. Σχεδιάστηκε ειδικά για TCP/IP δίκτυα και μπορεί να επεκταθεί, καλύπτοντας έτσι μεγαλύτερα δίκτυα. Είναι ανεξάρτητο από γλώσσες προγραμματισμού, παρόλο που τα APIs έχουν οριστεί σε C και Java. Επίσης το SLP φιλοδοξεί να αποτελέσει ένα πρότυπο ανεξάρτητο από τις προσεγγίσεις των διαφόρων κατασκευαστών. Η αρχιτεκτονική του SLP συνίσταται σε τρεις βασικές οντότητες: *User Agents* (UAs), *Service Agents* (SAs), *Directory Agents* (DAs). Οι *User Agents* αναζητούν τις ζητούμενες υπηρεσίες για λογαριασμό των *clients*· λέγοντας *client* εννοούμε κάποιο χρήστη ή μια εφαρμογή. Οι *Service Agents* διαφημίζουν τη διαθεσιμότητα των υπηρεσιών, είτε κατευθείαν στους *User Agents* είτε στους *Directory Agents*, αν υπάρχει ένας τέτοιος τουλάχιστον. Επιπλέον, οι SAs πληροφορούν το δίκτυο σχετικά με τη «θέση» και τα χαρακτηριστικά των υπηρεσιών. Οι DAs συλλέγουν τις

διευθύνσεις των υπηρεσιών καθώς και άλλες σχετικές πληροφορίες, τις αποθηκεύουν στις βάσεις δεδομένων που διαθέτουν και απαντούν σε κάθε αίτηση που δέχονται από τους UAs. Οι DAs αποθηκεύουν μόνο τις απαιτούμενες πληροφορίες για την επικοινωνία με την υπηρεσία – όχι τμήματα κώδικα. Οι πληροφορίες σχετικά με τον εντοπισμό μιας υπηρεσίας κωδικοποιούνται σε URLs.



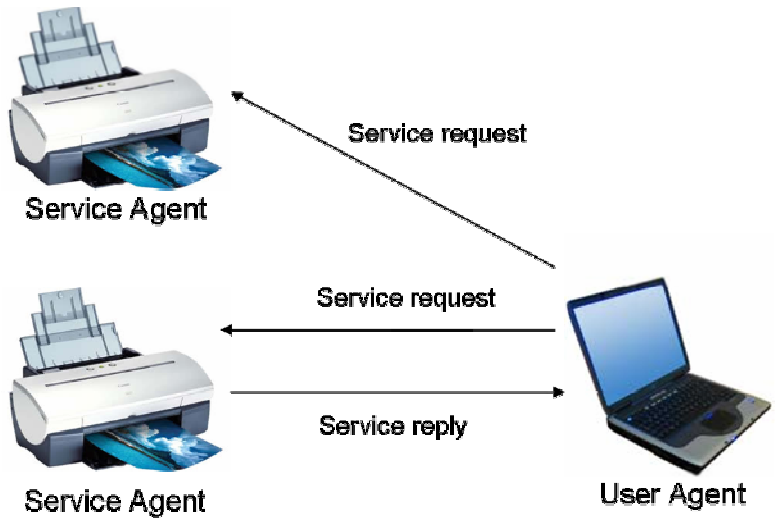
Εικόνα 7: Οι *SLP Agents* και οι αλληλεπιδράσεις μεταξύ τους για *service discovery* και *registration*

Η εικόνα 7 απεικονίζει την αλληλεπίδραση μεταξύ των τριών προαναφερθέντων οντοτήτων. Όταν μια καινούρια υπηρεσία συνδέεται στο δίκτυο, ο SA επικοινωνεί με τον DA, προκειμένου να γνωστοποιήσει την ύπαρξή της (*Service Registration*). Όταν κάποιος χρήστης επιθυμεί μια συγκεκριμένη υπηρεσία, ο UA ζητά από τον SA ενημέρωση σχετικα με τις διαθέσιμες υπηρεσίες στο δίκτυο. Αφού βρεθεί η διεύθυνση και τα υπόλοιπα χαρακτηριστικά της επιθυμητής υπηρεσίας, ο χρήστης μπορεί πλέον να χρησιμοποιήσει την υπηρεσία. Πριν ένας *client* (UA ή SA) έρθει σ' επαφή με τον DA, πρέπει αρχικά ν' ανακαλύψει την ύπαρξη ενός DA (*DA discovery*). Για να επιτευχθεί αυτή η διαδικασία υπάρχουν τρεις μέθοδοι: το στατικό (*static*), το ενεργό (*active*) και το παθητικό (*passive*) *DA discovery*.

Σύμφωνα με το *static DA discovery*, οι *SLP agents* ανακτούν τη διεύθυνση του DA μέσω του DHCP. Οι *DHCP servers* είναι υπεύθυνοι για τη διανομή των διευθύνσεων των DAs στους *hosts*, που κάνουν τις αντίστοιχες αιτήσεις. Στο *active DA discovery* οι UAs και SAs στέλνουν αιτήσεις για υπηρεσίες στην *SLP multicast group address* (π.χ. 239.255.255.253). Κάποιος DA που «ακούει» σ' αυτήν τη διεύθυνση, λαμβάνει τελικά μια αίτηση υπηρεσίας και απαντά απευθείας (με *unicast* τρόπο) στον αιτούντα *agent.*

Τέλος, στην περίπτωση του passive DA discovery, οι DAs δημοσιοποιούν περιοδικά με multicast τρόπο τις υπηρεσίες τους. Μ' αυτόν τον τρόπο, UAs και SAs ενημερώνονται για τις διευθύνσεις των υπηρεσίων και είναι σε θέση να επικοινωνήσουν με τους DAs πάλι μέσω *unicast*.

Είναι σημαντικό ν' αναφερθεί πως ο ρόλος των DAs είναι προαιρετικός και ότι εν γένει χρησιμοποιούνται σε μεγάλα δίκτυα με πολλές υπηρεσίες, όπου επιτακτική εμφανίζεται η ανάγκη για κατηγοριοποίηση των υπηρεσιών σε διαφορετικά πεδία. Σε μικρότερα δίκτυα, όπως το δίκτυο του σπιτιού ή του αυτοκινήτου, ενδείκνυται η εφαρμογή του SLP χωρίς τη χρήση DAs. Ανάλογα με την ύπαρξη ή όχι των DAs, το SLP διαθέτει διαφορετικούς τρόπους λειτουργίας. Έτσι όταν υπάρχει ένας DA στο δίκτυο, όπως φαίνεται και στην εικόνα 7, συλλέγει όλη τη σχετική πληροφορία που προσφέρουν οι SAs. Έπειτα οι UAs στέλνουν αιτήσεις υπηρεσιών στους DAs και λαμβάνουν την επιθυμητή πληροφορία. Στην περίπτωση απουσίας DAs, όπως φαίνεται και στην εικόνα 8, οι UAs στέλνουν επανειλημμένα τις αιτήσεις υπηρεσιών στη *SLP multicast* διεύθυνση. Όλοι οι SAs ακούνε αυτές τις αιτήσεις και αν διαθέτουν τις ζητούμενες υπηρεσίες, στέλνουν unicast απαντήσεις στους UAs. Επιπροσθέτως οι SAs υπενθυμίζουν την παρουσία τους, έτσι ώστε να είναι σε θέση οι UAs να ενημερωθούν για νέες υπηρεσίες.



Εικόνα 8: Εύρεση υπηρεσιών χωρίς DA

Η διαδικασία γνωστοποίησης των υπηρεσιών επιτυγχάνεται με τη βοήθεια ενός *Service URL* και ενός *Service Template*. Το *Service URL* περιέχει την IP διεύθυνση της υπηρεσίας, τον αριθμό θύρας (*port number*) και το σχετικό μονοπάτι (*path*), ενώ το *Service Template* καθορίζει τα χαρακτηριστικά της υπηρεσίας και τις εξ ορισμού τιμές.

Για παράδειγμα το *Service Template* για έναν εκτυπωτή θα μπορούσε να έχει την εξής μορφή:

```
service:printer://lj4050.uoa.gr:1020/queue1
scopes = uoa, di, administrator
printer-name = lj4050
printer-model = HP LJ4050 N
printer-location = Room 0409
color-supported = false
pages-per-minute = 9
sides-supported = one-sided, two-sided
```

Η πρώτη έκδοση του SLP έχει υλοποιηθεί σε διάφορα εμπορικά προϊόντα, όπως το *JetSend* της *Hewlett-Packard*, το οποίο υποστηρίζει εκτυπωτές, ψηφιακές κάμερες, σαρωτές, *projectors* και τις PDA πλατφόρμες *Windows* CE και *Palm* [19]. Η δεύτερη έκδοση αναμένεται να έχει ανάλογη εξέλιξη, ενώ έχει ήδη περιληφθεί στο *Web JetAdmin* της *Hewlett-Packard* και στο *Solaris 8*.

5.2 Το Πρωτόκολλο Εύρεσης Υπηρεσιών Salutation

Το *Salutation* αποτελεί μια άλλη προσέγγιση όσον αφορά την εύρεση υπηρεσιών, η οποία αναπτύχθηκε από το *Salutation Consortium* [20]. Η αρχιτεκτονική του Salutation περιλαμβάνει τους *Salutation Managers* (SLMs), οι οποίοι παίζουν το ρόλο των *service brokers*. Τα χαρακτηριστικά των υπηρεσίων καταχωρούνται με τη βοήθεια των SLMs ενώ οι *clients* απευθύνονται επίσης στους SLMs, προκειμένου ν' αναζητήσουν την υπηρεσία που επιθυμούν. Αφού βρούνε την εν λόγω υπηρεσία, μπορούν να την χρησιμοποιήσουν.

Το *Salutation* θεωρείται μια μάλλον παγιωμένη προσέγγιση με μερικές εμπορικές εφαρμογές από τις IBM και Axis. Χαρακτηριστικό παράδειγμα αποτελεί το *NuOffice* της IBM ενώ μελλοντικά σχέδια περιλαμβάνουν εφαρμογές για Palm OS και Windows CE.

5.3 Το Πρωτόκολλο Εύρεσης Υπηρεσιών UPnP

Το UPnP [21] αποτελείται από ένα σύνολο πρωτοκόλλων, που προορίζονται για *εύρεση υπηρεσιών*, ενώ δεν χρησιμοποιείται η μεταφορά κώδικα (όπως συμβαίνει για παράδειγμα στο Jini). Αναπτύχθηκε από το *Universal Plug and Play Consortium* της *Microsoft* και θα μπορούσε να πει κάποιος ότι ουσιαστικά πρόκειται για την εξέλιξη του *Plug and Play* της *Microsoft*, με την έννοια ότι τώρα οι συσκευές «επικοινωνούν» πάνω

από ένα TCP/IP δικτύο. Το UPnP σχεδιάστηκε με σκοπό την εφαρμογή του σε μικρά δίκτυα, όπου ενεργοποιούνται peer-to-peer μηχανισμοί για «αυτο-διαμόρφωση» των συσκευών, έυρεση και έλεγχο των υπηρεσιών.

Οι συσκευές και οι υπηρεσίες έχουν περιγραφεί σε XML ενώ στις προδιαγραφές περιλαμβάνονται τα πρωτόκολλα που αφορούν τοπική «αυτο-διαμόρφωση», εύρεση υπηρεσιών, αλληλεπίδραση *client/service* και *eventing* (Auto-IP, SSDP, SOAP, GENA). Στην τρέχουσα έκδοση το UPnP δε διαθέτει κάποιον τρόπο καταχώρησης υπηρεσιών, όπως συμβαίνει στο SLP με τους DAs ή στο *Jini* με το *lookup service* (που θα περιγραφεί παρακάτω), αλλά οι *clients* έρχονται απευθείας σ' επαφή με τις υπηρεσίες. Η εύρεση υπηρεσιών υλοποιείται μέσω του SSDP, το οποίο χρησιμοποιεί το HTTP πάνω από UDP και είναι σχεδιασμένο ειδικά για IP δίκτυα. Συμπερασματικά, το UPnP ενδείκνυται για περιβάλλοντα περιορισμένης έκτασης, όπως αυτό του σπιτιού ή του γραφείου, όπου το πλήθος των συσκευών είναι μικρό και η ανάγκη για καταχώρηση των υπηρεσιών θεωρείται αμελητέα.

5.4 Το Πρωτόκολλο Εύρεσης Υπηρεσιών Bluetooth SDP

Το *Bluetooth* [22] είναι μια νέα τεχνολογία μικρής εμβέλειας για ασύρματη μετάδοση. Οι συσκευές *Bluetooth* ψάχνουν στο περιβάλλον τους (σε ακτίνα 10 μέτρων από τον πομπό) για άλλες συσκευές, που προσφέρουν ή αναζητούν υπηρεσίες. Το Bluetooth SDP έχει βασιστεί στην πλατφόρμα *Piano* της *Motorola*, αλλά έχει τροποποιηθεί κατάλληλα ώστε να ταιριάζει με το δυναμικό χαρακτήρα των ad hoc επικοινωνιών. Πρόκειται για έναν απλό μηχανισμό που υλοποιεί *service discovery*, χρησιμοποιείται για τον εντοπισμό συσκευών *Bluetooth* και διευκολύνει την προσφορά υπηρεσιών. Κάθε συσκευή *Bluetooth* διατηρεί λίστα εγγραφών, όπου περιέχονται τα χαρακτηριστικά των διαθέσιμων υπηρεσιών, π.χ. τύπος υπηρεσίας, απαιτούμενα πρωτόκολλα για την επικοινωνία.

Επιπλέον, το Bluetooth SDP δεν περιλαμβάνει λειτουργία για πρόσβαση υπηρεσιών. Αφού εντοπιστεί μια υπηρεσία μέσω του SDP στη συνέχεια η επιλογή, πρόσβαση και χρήση της υπηρεσίας γίνεται από άλλους μηχανισμούς, όπως είναι το SLP και το Salutation.

5.5 Το Πρωτόκολλο Εύρεσης Υπηρεσιών Jini

Το *Jini*^{TM} [23] είναι μια τεχνολογία για εύρεση υπηρεσιών που έχει βασιστεί σε *Java* και έχει αναπτυχθεί από την *Sun Microsystems*. Το *Jini* θέτει το ζήτημα του πως μπορούν

οι συσκευές να συνδεθούν μεταξύ τους, σχηματίζοντας ένα απλό ad hoc δίκτυο (το λεγόμενο *Jini "community"*) και πως μπορούν να παρέχουν υπηρεσίες η μια στην άλλη. Το *Jini* αποτελείται από μια συγκεκριμένη αρχιτεκτονική [24] και ένα προγραμματιστικό μοντέλο. Επίσης λόγω της ανεξάρτητης – από πλατφόρμες – φύσης της *Java*, το *Jini* βασίζεται στη μεταφορά κώδικα για να ελέγχει τις υπηρεσίες.

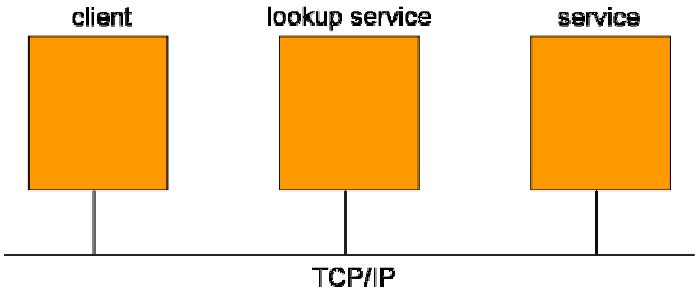
Ουσιαστικά, πρόκειται για ένα κατανεμημένο υπολογιστικό περιβάλλον, το οποίο φιλοδοξεί να προσφέρει το λεγόμενο "*network plug and play*". Μια συσκευή ή μια υπηρεσία συνδέεται στο δίκτυο, ανακοινώνει την παρουσία της και οι ενδιαφερόμενοι *clients* μπορούν να την εντοπίσουν. Το *Jini* μπορεί να χρησιμοποιηθεί σε διάφορα θέματα κινητού υπολογισμού, όπου κάποιες υπηρεσίες συνδέονται ενδεχομένως για μικρό χρονικό διάστημα, αλλά μπορεί να εφαρμοστεί και σε δίκτυα όπου υπάρχει σχετικός βαθμός μεταβλητότητας.

Για παράδειγμα:

- ένας καινούριος εκτυπωτής συνδέεται σ' ένα δίκτυο και ανακοινώνει την παρουσία του καθώς επίσης και τα χαρακτηριστικά του. Ένας *client* μπορεί να χρησιμοποιήσει τον εκτυπωτή, χωρίς να απαιτείται ειδική διαμόρφωση,
- μια ψηφιακή κάμερα συνδέεται στο δίκτυο και προσφέρει στο χρήστη όχι μόνο τη δυνατότητα για λήψη φωτογραφιών, αλλά επίσης και τη δυνατότητα να γνωρίζει αν υπάρχουν στο δίκτυο εκτυπωτές, προκειμένου να τις τυπώσει,
- ένα αρχείο για configuration, που μπορεί να αντιγραφεί και να τροποποιηθεί σε ξεχωριστά μηχανήματα, είναι δυνατόν να περιέχεται σε μια υπηρεσία ενός συγκεκριμένου μηχανήματος, μειώνοντας έτσι το κόστος συντήρησης,
- νέες δυνατότητες που αφορούν υπάρχουσες υπηρεσίες του δικτύου μπορούν να εγκατασταθούν, χωρίς να διακόψουν τη λειτουργία αυτών των υπηρεσιών και χωρίς την ανάγκη για αναδιαμόρφωση των *clients*,
- οι υπηρεσίες μπορούν να ανακοινώνουν τυχόν μεταβολές, όσον αφορά την κατάστασή τους και οι οντότητες με διαχειριστικό χαρακτήρα αναλαμβάνουν να παρακολουθούν αυτές τις αλλαγές και να τις επισημαίνουν.

Στην πραγματικότητα ένα σύστημα *Jini* συνίσταται σε *clients* και *services*, που επικοινωνούν μεταξύ τους χρησιμοποιώντας τα πρωτόκολλα του *Jini*. Συχνά ένα τέτοιο σύστημα αποτελείται από εφαρμογές *Java*, οι οποίες επικοινωνούν με τη βοήθεια του

μηχανισμού RMI. Παρότι το *Jini* είναι εξ ολοκλήρου γραμμένο σε *Java*, οι *clients* και οι υπηρεσίες μπορούν να περιέχουν τμήματα κώδικα άλλης γλώσσας προγραμματισμού. Το *Jini* απλά παρέχει το *middleware* επίπεδο, πάνω στο οποίο συνδέονται *clients* και *services*, που ενδεχομένως διαθέτουν διαφορετικό υπόβαθρο.



Εικόνα 9: Τα βασικά συστατικά του *Jini*

Το *Jini* αποτελεί μια από τις πολλές αρχιτεκτονικές για κατανεμημένα συστήματα, όπως είναι τα CORBA και DCOM. Αυτό που διαφοροποιεί το *Jini* είναι το γεγονός ότι βασίζεται στη Java και ως εκ τούτου κληρονομεί τα χαρακτηριστικά της. Υπάρχουν κι αλλά *Java frameworks* της Sun, όπως τα EJBs που εφαρμόζονται κυρίως σε *business logic servers*, ενώ το *Jini* ενδείκνυται για υποστήριξη αυτών των υπηρεσιών με έναν "*plug and play*" τρόπο.

Ένα σύστημα *Jini*, όπως απεικονίζεται και στην εικόνα 9, περιέχει τρία βασικά συστατικά:

- ***service***, υπηρεσία όπως για παράδειγμα ένας εκτυπωτής, μια τοστιέρα κτλ.,
- ***client***, που χρησιμοποιεί την υπηρεσία και
- ***lookup service (service locator)***, που λειτουργεί σαν *broker* και αποτελεί την ενδιάμεση οντότητα αναζήτησης μεταξύ *clients* και *services*.

Ένα τέταρτο συστατικό θα μπορούσε να θεωρηθεί το ***network***, το δίκτυο δηλαδή που συνδέει τα τρία προαναφερθέντα και γενικά τρέχει πάνω από TCP/IP (οι προδιαγραφές του *Jini* είναι ανεξάρτητες από το πρωτόκολλο δικτύου, αλλά η τρέχουσα υλοποίησή του αφορά το TCP/IP).

Ένα ιδιαίτερο γνώρισμα, που χαρακτηρίζει το *Jini* είναι η δυνατότητα της μεταφοράς κώδικα μεταξύ των τριών βασικών οντοτήτων, το οποίο επιτυγχάνεται αν τα διάφορα αντικείμενα χρησιμοποιηθούν με τέτοιο τρόπο, ώστε να είναι *serializable*, να μπορούν να μετακινηθούν στο δίκτυο και να αποθηκευτούν σε ξεχωριστά μηχανήματα, καθένα από τα οποία διαθέτει ένα JVM. Η όλη διαδικασία υλοποιείται με τη χρήση *sockets* της

Java. Επιπλέον, τα αντικείμενα σ' ένα JVM ενδέχεται να χρειαστούν την ενεργοποίηση μεθόδων που ανήκουν σε άλλο JVM, το οποίο μπορεί να υλοποιηθεί μέσω του RMI, αν και δεν απαιτείται από το Jini μια που διατίθενται κι άλλοι τρόποι.

5.5.1 Lookup Service

Σε κάθε *lookup service* υπάρχει το λεγόμενο *lookup table* – ουσιαστικά πρόκειται για μια βάση δεδομένων – όπου καταχωρούνται όλες οι υπηρεσίες που διατίθενται στο δίκτυο (κάτι αντίστοιχο με το DA του SLP). Αυτό που αξίζει να σημειωθεί είναι ότι στο *lookup table* εκτός από τους δείκτες στις υπηρεσίες, μπορεί να αποθηκευτεί και *java-based* κώδικας για τις υπηρεσίες. Αυτό σημαίνει πως οι υπηρεσίες έχουν τη δυνατότητα ν' «ανεβάσουν» μια διεπαφή, *drivers* και άλλα σχετικά προγράμματα, που βοηθούν το χρήστη να προσπελάσει/υλοποιήσει την υπηρεσία.

5.5.2 Service Registration

Κάθε υπηρεσία είναι μια οντότητα, που συνήθως ορίζεται από ένα *Java interface* και συνήθως αναγνωρίζεται μέσω αυτού του *interface*. Μια υπηρεσία μπορεί να υλοποιηθεί με πολλούς τρόπους, ανάλογα με τον κατασκευαστή. Αυτό που διαφοροποιείται από τη μια υλοποίηση στην άλλη είναι τα διαφορετικά αντικείμενα που χρησιμοποιούνται και ανήκουν σε διαφορετικές κλάσεις.

Η κάθε υπηρεσία δημιουργείται από το λεγόμενο *service provider*, ο οποίος παίζει μια σειρά από ρόλους:

- δημιουργεί τα αντικείμενα που υλοποιούν την υπηρεσία*,*
- καταχωρεί το ένα από αυτά, *service object*, στις *lookup services*. Το *service object* είναι το μόνο ορατό τμήμα της υπηρεσίας στο δίκτυο και είναι αυτό που «κατεβάζουν» οι *clients* και
- παίζει το ρόλο του *server*, εκτελώντας ποικίλα καθήκοντα, όπως το να κρατά την υπηρεσία «ζωντανή».

Προκειμένου να καταχωρηθεί ένα *service object* σε μια *lookup service*, πρέπει πρώτα ο *service provider* να εντοπίσει μια *lookup service*, πράγμα που μπορεί να επιτευχθεί με δύο τρόπους: αν η θέση της είναι γνωστή, τότε μπορεί να χρησιμοποιηθεί unicast TCP για απευθείας σύνδεση. Διαφορετικά, ο *service provider* είναι υποχρεωμένος να κάνει multicast UDP αιτήσεις, στις οποίες απαντούν οι *lookup services*. Οι *lookup services* «ακούνε» για unicast και multicast αιτήσεις στη θύρα 4160 κι όταν φτάσει μια από αυτές

τις αιτήσεις, ένα αντικείμενο στέλνεται πίσω στον server. Αυτό το αντικείμενο είναι γνωστό ως *registrar*, λειτουργεί ως proxy για τη *lookup service* και τρέχει στο τοπικό JVM της υπηρεσίας. Όλες οι αιτήσεις που επιθυμεί να κάνει ένας *service provider* στη *lookup service*, γίνονται μέσω αυτού του *proxy registrar*, χρησιμοποιώντας οποιοδήποτε κατάλληλο πρωτόκολλο, αλλά στην πράξη το πιο διαδεδομένο είναι το RMI.

Σε μορφή ψευτοκώδικα μια υπηρεσία μπορεί να είναι ως εξής:

prepare *for discovery*

discover *a lookup service*

create *information about a service*

export *a service*

renew *leasing periodically*

5.5.3 Client lookup

Ο client από την άλλη πλευρά προσπαθεί ν' αποκτήσει ένα αντίγραφο της υπηρεσίας που τον ενδιαφέρει στο JVM του, ακολουθώντας το μηχανισμό που περιγράψαμε προηγουμένως για το *service registration*. Η διαδικασία είναι πανομοιότυπη μέχρι του σημείου, όπου ζητά την αντιγραφή του *service object* στο τοπικό JVM. Μ' αυτόν τον τρόπο υπάρχουν τελικά δύο αντίγραφα του *service object*, ένα στον *client* και ένα στη *lookup service*. Στη συνέχεια με τη βοήθεια του κώδικα του *service object*, είναι δυνατή η απευθείας πρόσβαση του *client* στην υπηρεσία, χρησιμοποιώντας το *interface* που είναι γνωστό μόνο στον *client*. Αυτή η μετακίνηση του κώδικα αντικαθιστά την προ-εγκατάσταση των *drivers* στον *client*.

Σε μορφή ψευτοκώδικα ένας *client* θα μπορούσε να είναι κάπως έτσι:

prepare *for discovery*

discover *a lookup service*

prepare *a template for lookup search*

lookup *a service*

call *the service*

Συμπερασματικά, τα τρία συστατικά ενός συστήματος Jini (*clients, services, service locators*) μπορούν να υλοποιηθούν σε *Java* και να «τρέχουν» οπουδήποτε στο δίκτυο στα τοπικά JVMs. Εναλλακτικά, μπορούν να περιέχουν τμήματα κώδικα σε JNI ή ακόμα και να κάνουν εξωτερικές κλήσεις σε άλλες εφαρμογές. Κάθεμιά από αυτές τις εφαρμογές τρέχει σε ξεχωριστό JVM, παρόλο που θα μπορούσε να τρέχει και στο ίδιο μηχάνημα. Κατά τη διάρκεια της εκτέλεσης χρειάζεται η πρόσβαση σε κάποιες *Java*

κλάσεις και το κάθε συστατικό χρησιμοποιεί τη μεταβλητή περιβάλλοντος *classpath*, προκειμένου να εντοπίσει τις απαιτούμενες *Java* κλάσεις.

5.5.4 Entry Objects/Entry Class

Όταν ένας *service provider* καταχωρεί μια υπηρεσία, στην ουσία τοποθετεί ένα αντίγραφο του *service object* στη *lookup service*, ενώ μπορεί προαιρετικά να καταχωρήσει ένα σετ από χαρακτηριστικά σχετικά με το *service object*. Μ' αυτόν τον τρόπο σε κάθε *service locator* υπάρχει μια σειρά εγγραφών που περιέχουν το *instance* μιας κλάσης καθώς επίσης και τα ιδιαίτερα χαρακτηριστικά της υπηρεσίας. Για παράδειγμα έστω ότι διατίθεται ένα σετ από επεξεργαστές αρχείων (*file editors*), καθένας από τους οποίους μπορεί να χειριστεί διαφορετικού τύπου αρχεία. Ένας πιθανός *client* μπορεί ν' αναζητήσει τον κατάλληλο επεξεργαστή με δύο τρόπους:

- Ρωτώντας για την ύπαρξη μιας συγκεκριμένης κλάσης, όπως *imageEditor,*
- Ρωτώντας για την ύπαρξη μιας γενικής κλάσης (*editor*), δίνοντας επιπλέον την πληροφορία που αφορά τον σχετικό τύπο αρχείων.

Ο τύπος της αναζήτησης εξαρτάται από το είδος της πληροφορίας που διαθέτουν οι *clients*. Το *Jini* είναι σε θέση να χειριστεί και τις δύο προαναφερθείσες περιπτώσεις. Στην πρώτη απλά εντοπίζει ένα *class object*, όπως το *ImageEditor.class*, ενώ στη δεύτερη εντοπίζει ένα *superclass object* και επιτρέπει την επιπλέον πληροφορία να δοθεί στην αίτηση, προσθέτοντας κάποια άλλα αντικείμενα.

Η κλάση *Entry* επιτρέπει στις υπηρεσίες να διαφημίσουν τις δυνατότητές τους με πολλούς τρόπους. Για παράδειγμα, αν υποτεθεί ότι ένας *file editor* μπορεί να επεξεργαστεί αρχεία κειμένου και φωτογραφίες, τότε αρκεί να δημιουργήσει το *service object*, που υλοποιεί τη διεπαφή *Editor* καθώς επίσης και ένα *Entry* object, το οποίο αναφέρει ότι ο συγκεκριμένος επεξεργαστής αρχείων μπορεί να εργαστεί μόνο πάνω σε αρχεία κειμένου ή φωτογραφίες.

5.5.5 Leases

Στο Jini, κάθε υπηρεσία διατίθεται στο δίκτυο για συγκεκριμένο χρονικό διάστημα, το οποίο επιτυγχάνεται μέσω του μηχανισμού leasing. Μ' αυτόν τον τρόπο ένα αντίγραφο της υπηρεσίας παραμένει αποθηκευμένο στη *lookup service* για ορισμένο διάστημα του χρόνου, προκειμένου να χρησιμοποιηθεί όταν ζητηθεί από κάποιο χρήστη. Η κάθε

υπηρεσία ζητά το προαναφερθέν χρονικό διάστημα με τη βοήθεια της μεθόδου *register* της κλάσης *ServiceRegistrar* και υπάρχουν δυο συνηθισμένες τιμές χρόνου:

- Lease.ANY: η υπηρεσία αφήνει τη *lookup service* ν' αποφασίσει για το χρονικό διάστημα
- Lease.FOREVER: η υπηρεσία ζητά την καταχώρησή της για πάντα

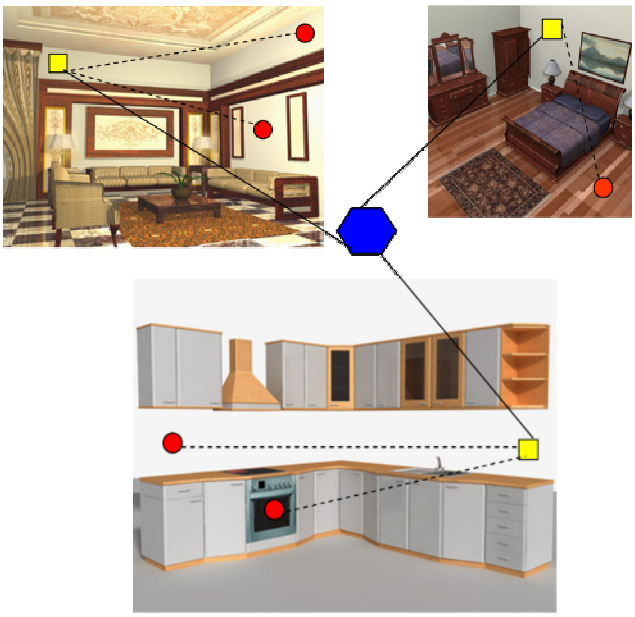
Η υπηρεσία μπορεί ν' ανανεώσει το χρόνο ύπαρξής της καλώντας τη μέθοδο *renew* της κλάσης *LeaseRenewalManager*.

6 ΠΡΟΤΕΙΝΟΜΕΝΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗ *HYPER-REGGIE* ΓΙΑ ΤΟ *JINI*

6.1 Περιγραφή και Σενάρια

Το *Jini*™ αποτελεί μια αρχιτεκτονική που βασίζεται στη *Java*™ και επιτρέπει το δυναμικό έλεγχο των συσκευών και των υπηρεσιών που υπάρχουν στο δίκτυο. Μια κοινότητα *Jini* (*Jini* community) εγκαθίσταται γύρω από μία ή περισσότερες *lookup services*, οι οποίες διαθέτουν τις υπηρεσίες στην κοινότητα και απαντούν στις αιτήσεις των πελατών. Ουσιαστικά το *Jini* επιτυγχάνει το λεγόμενο *spontaneous networking*, το οποίο βασίζεται σ' ένα μηχανισμό *leasing*, που σημαίνει ότι κάθε υπηρεσία προσφέρεται για συγκεκριμένο χρονικό διάστημα και έπειτα παύει να υφίσταται. Αν ο πάροχος της υπηρεσίας δεν ζητήσει την ανανέωση του χρόνου της, η *lookup service* αυτόματα διακόπτει την υπηρεσία.

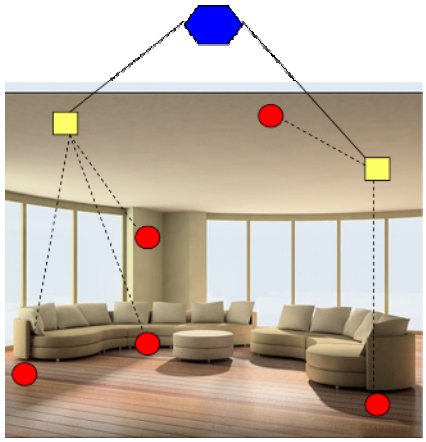
Ας θεωρήσουμε δύο σενάρια που αφορούν περιβάλλοντα περιορισμένης εμβέλειας (οικία, γραφείο, αεροδρόμιο), τα οποία βασίζονται σε διαφορετικές μορφές αρχιτεκτονικής. Σύμφωνα με το πρώτο από αυτά, που αναφέρεται σε οικιακό περιβάλλον, θεωρούμε ότι σε κάθε δωμάτιο του σπιτιού υπάρχει ένα ξεχωριστό δίκτυο. Κάθε τέτοιο δίκτυο διαθέτει το δικό του *reggie* στο οποίο είναι καταχωρημένο ένα σύνολο από υπηρεσίες. Αυτή η μορφή αρχιτεκτονικής εστιάζει στη χωρική κατανομή των υπηρεσίων, αφού σ' ένα σπίτι οι ανάγκες των χρηστών ταξινομούνται ανάλογα με τους χώρους του σπιτιού.



Εικόνα 10: Εφαρμογή *Jini* σε οικιακό περιβάλλον

Στο παραπάνω σχήμα απεικονίζεται συμβολικά ένα οικιακό περιβάλλον που αποτελείται από μια κουζίνα, έναν καθιστικό χώρο και μια κρεβατοκάμαρα. Κάθε δωμάτιο διαθέτει το δικό του *reggie* (κίτρινο τετράγωνο) και ένα σύνολο από υπηρεσίες όπως φωτισμός, κλιματισμός, τηλεόραση, καφετιέρα, ηλεκτρικός φούρνος (κόκκινοι κύκλοι).

Το δεύτερο σενάριο εξετάζει την περίπτωση διαφορετικών δικτύων που συνυπάρχουν στον ίδιο χώρο (*spatial or conceptual space*). Ένα παράδειγμα αυτής της ιδέας είναι ο συνεδριακός χώρος ενός αεροδρομίου, ο οποίος φαίνεται στο παρακάτω σχήμα.



Εικόνα 11: Εφαρμογή *Jini* σε συνεδριακό χώρο αεροδρομίου

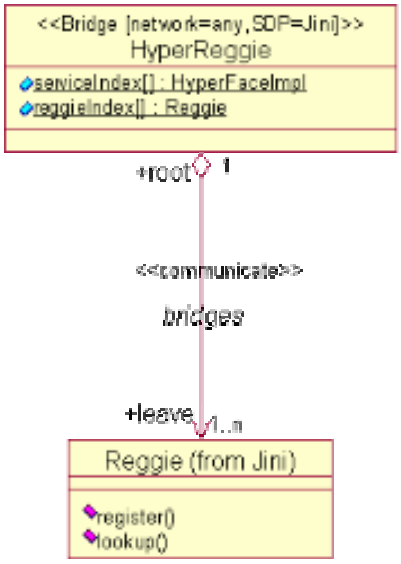
Ας υποθέσουμε ότι συνυπάρχουν δύο δίκτυα, κάθενα από τα οποία περιλαμβάνει ένα *reggie* και ένα πλήθος από υπηρεσίες. Στο πρώτο δίκτυο ως μια ενδεχόμενη υπηρεσία θα μπορούσε να προσφερθεί η ενημέρωση σχετικά με τις πτήσεις που λαμβάνουν χώρα στο αεροδρόμιο, περιλαμβάνοντας πληροφορίες όπως ώρα άφιξης, ώρα αναχώρησης, προορισμός και χρονική καθυστέρηση. Αντίστοιχα, στο δεύτερο δίκτυο μια πιθανή υπηρεσία θα μπορούσε να θεωρηθεί η οργάνωση των πρακτικών του συνεδρίου.

Το ζήτουμενο και στα δύο σενάρια είναι η επίτευξη της λεγόμενης «μεταπομπής» (*hand-over*) μεταξύ μιας υπηρεσίας του ενός δικτύου και μιας υπηρεσίας του άλλου δικτύου. Για παράδειγμα στο πρώτο σενάριο υποθέτουμε ότι ο χρήστης ξυπνάει το πρωί και επιθυμεί να θέσει αμέσως σε λειτουργία την καφετιέρα, που βρίσκεται στην κουζίνα, χωρίς όμως να χρειάζεται να πάει ως εκεί. Το πρόβλημα έγκειται στο γεγονός ότι η καφετιέρα δεν είναι καταχωρημένη στο *reggie* της κρεβατοκάμαρας. Αντίστοιχα, στο δεύτερο σενάριο επιθυμητή θα ήταν η ενημέρωση των συνέδρων σχετικά με τις πτήσεις, χωρίς να απαιτείται η μετακίνηση από το συνεδριακό χώρο στον πίνακα ανακοινώσεων του αεροδρομίου.

Προκειμένου ν' αντιμετωπιστούν τέτοιου είδους καταστάσεις, η προσέγγιση που προτείνεται προϋποθέτει τη σύσταση μια νέας οντότητας υψηλότερου επιπέδου, που από εδώ και στο εξής θα αποκαλούμε *HyperReggie* (μπλε εξάγωνο στα προηγούμενα σχήματα). Η οντότητα αυτή είναι ενήμερη για όλα τα *reggies* που υπάρχουν σε μια ευρύτερη περιοχή που περιλαμβάνει περισσότερα από ένα δίκτυο. Έτσι όταν ένας *client* ζητά μια υπηρεσία που «ανήκει» σε διαφορετικό δίκτυο, είναι σε θέση να την «κατεβάσει», με τη βοήθεια του *HyperReggie*. Θα μπορούσαμε να υποθέσουμε ότι η οντότητα του *HyperReggie* συμπεριφέρεται σαν ένα είδος *gateway* μεταξύ διαφορετικών δικτύων.

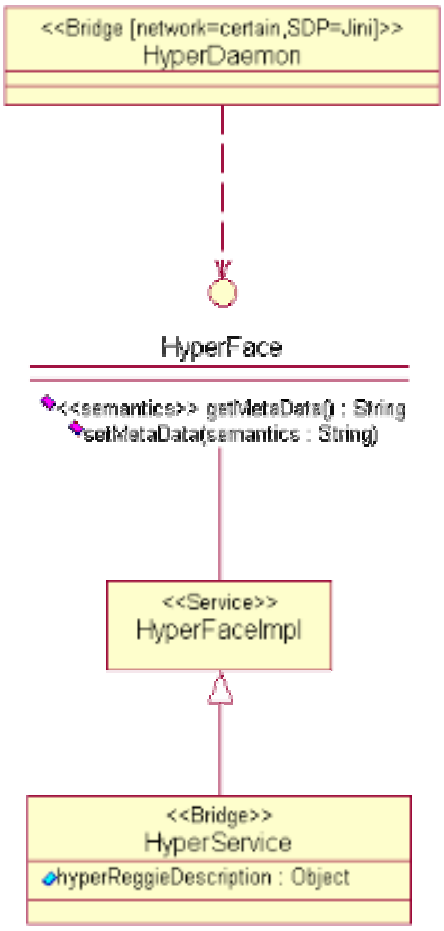
6.2 Ανάλυση με χρήση UML διαγραμμάτων

Σ' αυτήν την ενότητα γίνεται προσπάθεια μοντελοποίησης της ιεραρχικής δομής που περιγράφηκε παραπάνω με τη βοήθεια UML διαγραμμάτων. Αυτή η δομή περιλαμβάνει τις εξής οντότητες: *HyperReggie, reggies, services, clients*.



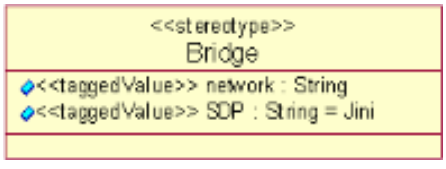
Διάγραμμα 1: Ιεραρχική δομή των *Reggies*

Υποθέτουμε την ύπαρξη έως και n δικτύων, σε κάθενα από τα οποία ως πρωτόκολλο για *εύρεση υπηρεσιών* χρησιμοποιείται το Jini. Στο διάγραμμα 1 απεικονίζεται η σύνδεση μεταξύ του HyperReggie και ενός πλήθους από reggies.



Διάγραμμα 2: Η υπηρεσία HyperDaemon

Στο διάγραμμα 2 απεικονίζεται η υλοποιηση μιας ειδικής υπηρεσίας, HyperDaemon, η οποία επισκέπτεται αέναα το κάθε reggie των δικτύων του Jini. Η υπηρεσία αυτή εκτελεί καθολικές ερωτήσεις από το κάθε reggie για το ποιες υπηρεσίες είναι uploaded. Για κάθε μια υπηρεσία μέσω της διεπαφής getMetaData(), συλλέγει τις πληροφορίες που βρίσκονται στην δομή Entries. Με αυτά τα μετα-δεδομένα της υπηρεσίας και η πληροφορία του εκάστοτε πλαισίου στέλνεται στον HyperReggie. Ο τελευταίος κατασκευάζει τα μεταδεδομένα σε στιγμιότυπα της οντολογίας *ServiceOntology* (Κεφ. 8) που διατηρείται συνέχεια ενημερωμένη από τον HyperDaemon. Το στερεότυπο Bridge ερμηνεύει τις κλάσεις του τύπου που περιέχουν πληροφορία πλαισίου, όπως με τι πρωτόκολλο *εύρεσης υπηρεσιών* επικοινωνούν οι υπηρεσίες καθώς και ποιο είναι το δίκτυο επικοινωνίας τους.

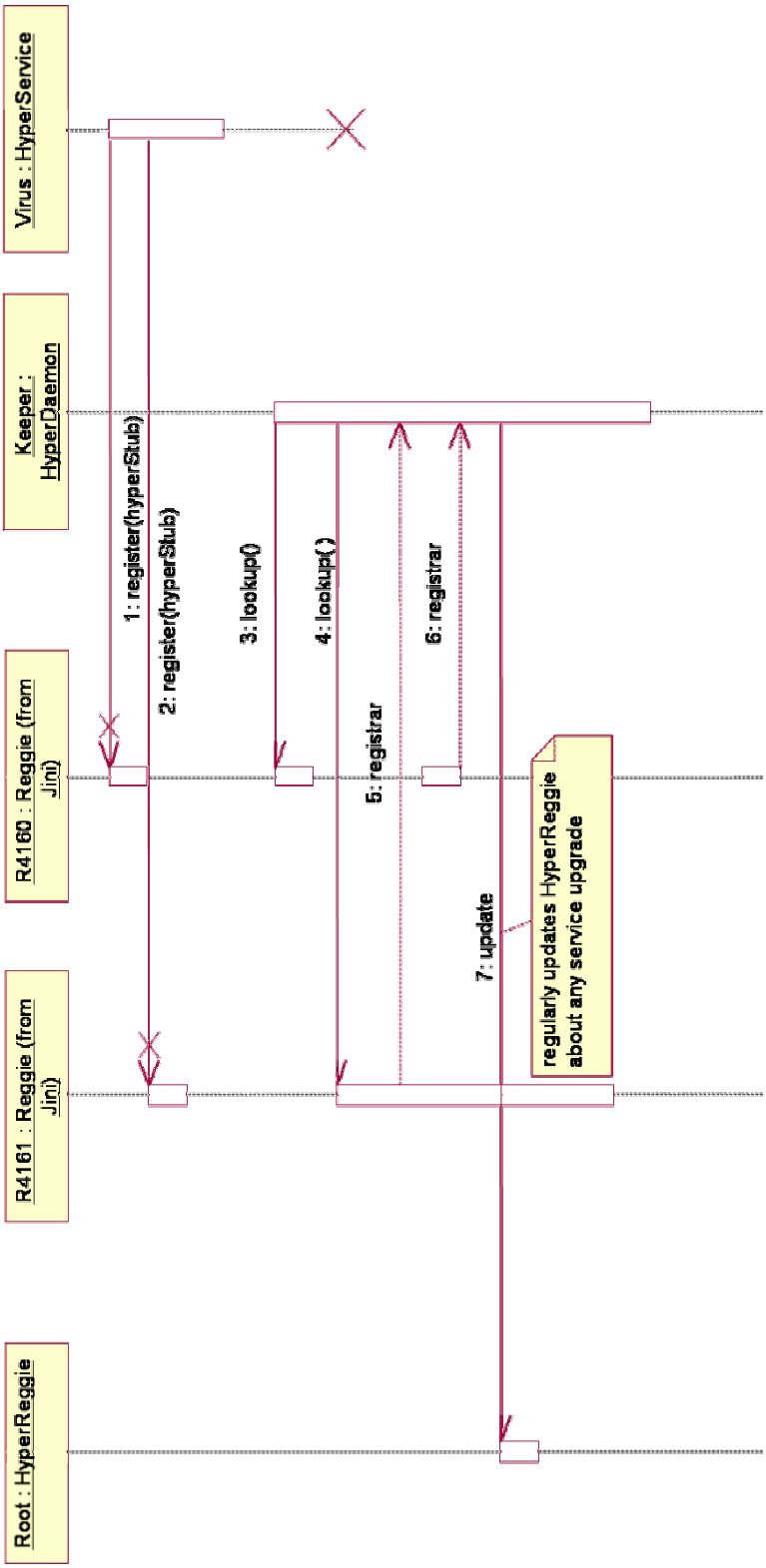


Διάγραμμα 3: Το στερεότυπο Bridge

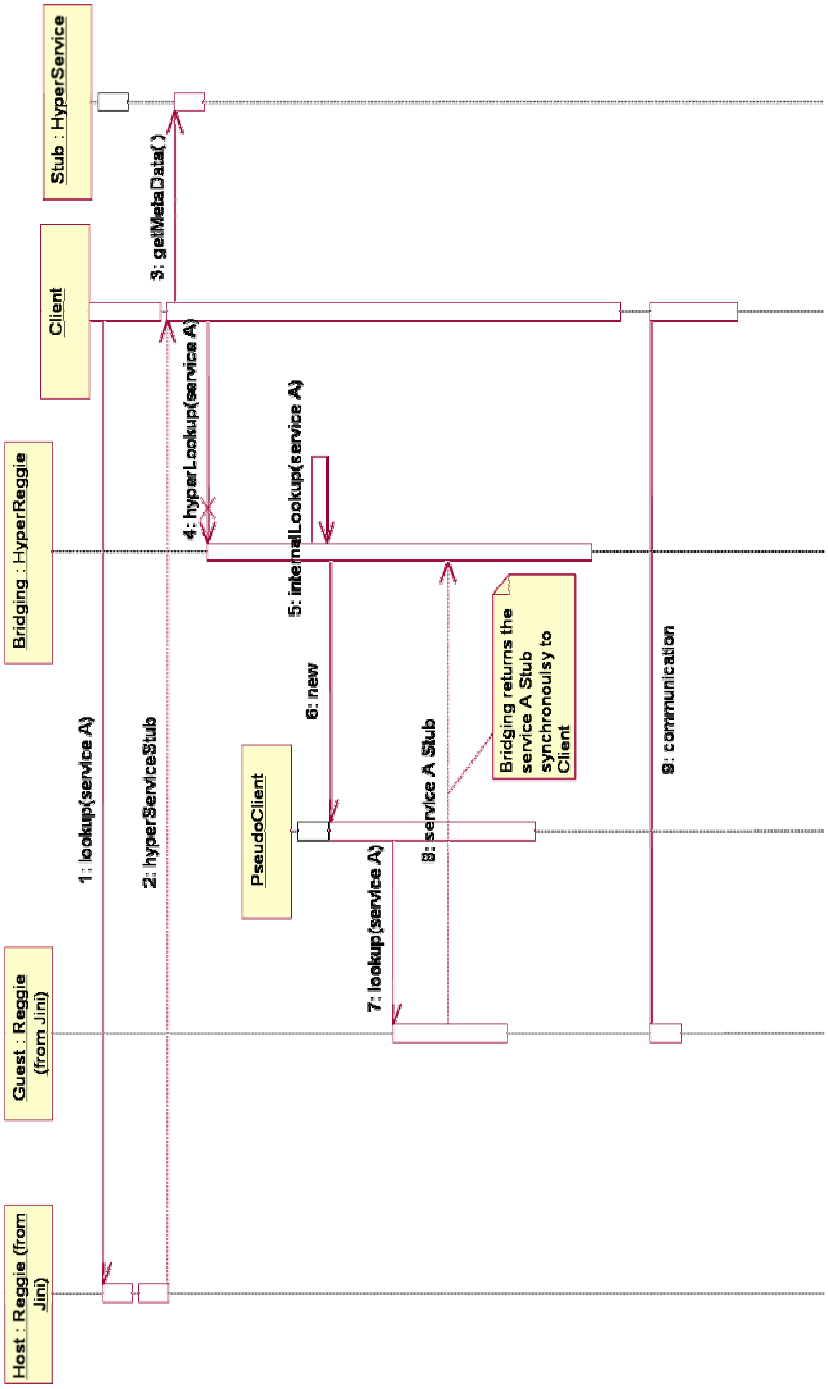
Το διάγραμμα 4 ορίζει την αρχικοποίηση του HyperDaemon και τον τρόπο που η οντολογία ServiceOntology κατασκευάζεται και ενημερώνεται στον HyperReggie.
Το διάγραμμα 5 ορίζει ένα σενάριο για την επιλογή μιας υπηρεσίας από άλλο δίκτυο Jini όπου επιστρέφεται μια παρόμοια με την ζητηθείσα υπηρεσία. Αρχικά ο client αναζητά την υπηρεσία *ServiceA*. Η υπηρεσία αυτή όμως δεν διατίθεται στο reggie όπου γίνεται η αίτηση. Παρόλα αυτά ο reggie δεν απαντά αρνητικά και επιστρέφει στον client μια ειδική υπηρεσία η οποία γνωρίζει την τοποθεσία του HyperReggie. Η ειδική αυτή υπηρεσία προωθεί την ίδια αίτηση στον HyperReggie με σκοπό την αναζήτηση της ServiceA σε άλλα reggies. Στην συνέχεια ο HyperReggie επιλέγει μεταξύ των δύο:

- αφού αποφανθεί ότι υπάρχει αλλού η *ServiceA* δημιουργεί μια επίπλαστη αίτηση για την όμοια υπηρεσία και επιστρέφει τον proxy στον client,
- αφού αποφανθεί ότι δεν υπάρχει αλλού η *ServiceA* τότε υπολογίζει την περισσότερο όμοια υπηρεσία σύμφωνα με την διαδικασία που θα περιγραφεί σε επόμενο κεφάλαιο, δημιουργεί μια αίτηση για την όμοια υπηρεσία και επιστρέφει τον proxy στον client. Τέλος ο HyperReggie συνιστά ένα σύνολο από συνεργαζόμενες με την όμοια υπηρεσία.

Με αυτό το τρόπο ο client γνωρίζει ότι μπορεί να έχει πρόσβαση και σε άλλες υπηρεσίες εκτός του δικτύου που είναι συνδεδεμένος.



Διάγραμμα 4: Αρχικοποίηση HyperDaemon και ενημέρωση της ServiceOntology



Διάγραμμα 5: Παράδειγμα αναζήτησης υπηρεσίας

7 ΠΕΡΙΓΡΑΦΙΚΗ ΛΟΓΙΚΗ

Δεδομένης της υψηλής συχνότητας εμφάνισης των DLs [25] στην κοινότητα του Σημασιολογικού Ιστού, θεμιτή θεωρείται η υπόθεση ότι το σχήμα κλάσεων της ζητούμενης οντολογίας δύναται να εκφραστεί με τη βοήθεια της DL. Ως γνωστό, η DL αποτελεί ένα υποσύνολο της Λογικής Πρώτης Ταξης, στο οποίο δεν υποστηρίζονται free μεταβλητές και v-απλές σχέσεις (n-ary relationships). Μια DL οντολογία συνίσταται σε απλές ή και σύνθετες κλάσεις (πίνακας 5) και εμφανίζει περιορισμούς. Οι κλάσεις ταξινομούνται σε πρωταρχικές (ορισμένες από τον μοντελιστή), που εμπλέκονται δηλαδή μόνο σε ικανές συνθήκες και ορισμένες (που συνεπάγονται από μια μηχανή συμπερασμού), που διαθέτουν τουλάχιστον μια ικανή και αναγκαία συνθήκη. Επιπρόσθετα, υπάρχει η έννοια των συσχετίσεων, που ορίζουν τις συσχετίσεις μεταξύ δύο κλάσεων ή μεταξύ μιας κλάσης και ενός τύπου δεδομένων. Συνδυάζοντας κλάσεις και συσχετίσεις, είναι δυνατή η δημιουργία ιεραρχικών δομών, που αποτελούνται από is-a σχέσεις, όπου οι γενικότερες έννοιες εμπεριέχουν τις πιο εξειδικευμένες. Στην OWL-DL ορίζονται το πεδίο τιμών και το πεδίο ορισμού για μια συγκεκριμένη συσχέτιση. Όταν αμφότερα τα πεδίο τιμών και πεδίο ορισμού μιας συσχέτισης αποτελούν κλάσεις, τότε η εν λόγω συσχέτιση καλείται ObjectProperty. Στην περίπτωση όμως που το πεδίο τιμών μπορεί να είναι λεξικογραφικό (π.χ. ακέραιος, συμβολοσειρά), τότε η αντίστοιχη συσχέτιση ονομάζεται DatatypeProperty.

Η μεγάλη δημοτικότητα των οντολογιών που βασίζονται σε DL έγκειται στο γεγονός ότι οι ανάλογες DL μηχανές συμπερασμού προσφέρουν αποτελεσματικές υπηρεσίες, όσον αφορά τους αποθηκευμένους ισχυρισμούς. Το πιο σημαντικό ζητούμενο είναι η απόφαση για το αν μια κλάση είναι *συνεπής* ή όχι και αν μια κλάση είναι πιο γενική από κάποια άλλη. Τέλος, μια άλλη υπηρεσία που παρέχεται από ένα DL-system είναι η απόφαση για το κατά πόσο ένα σύνολο από ισχυρισμούς είναι λογικά σωστό (λογικό επακόλουθο), που σημαίνει ότι οι συγκεκριμένοι ισχυρισμοί (στιγμιότυπα) αποτελούν στιγμιότυπα των περιγραφών των κλάσεων της δοθείσης οντολογίας.

OWL syntax	DL syntax	Example	Description
owl:intersectionOf	C Π D	Supervisor Π Male	All Supervisors that are Male
owl:unionOf	C∪D	Supervisor ∪ Manager	Anything that is either Supervisor or

			Employee
owl:allValuesFrom	∀R.C	∀ boss.Male	All bosses must be of type Male
owl:someValuesFrom	∃R.C	∃hasSon.Male	At least one of the sons must be of type Male
owl:value	∃R.{o}	∃hasLocation.Athens	The location property must have the value Athens
owl:minCardinality	≥n R.C	≥1supervises.Employee	A Supervisor supervises at least one employee
owl:maxCardinality	≤n R.C	≤1hasManager.Manager	An Employee has at most one Manager
owl:Cardinality	=n R.C	= 1 isDependentOn.Employee	A Person is dependent on exactly one Employee
owl:subClassOf	C⊆D	Manager ⊆ Employee	A Manager is a kind of Employee
owl:disjointClass	C¬D	Department_Manager ¬ Technical_Manager	Someone cannot be a Department_Manager and a Technical_Manager at the same time
	∃R.(CΠ∃P.D)≡ ∃(RΠP).D	∃drives.(Car Π ∃type.Sport) ≡∃drivesΠtype.Sport	Anyone who drives a sport car

Πίνακας 5: Τα κυριότερα OWL-DL αξιώματα (C, D: κλάσεις - R, S: συσχετίσεις)

8 ΟΝΤΟΛΟΓΙΑ ΥΠΗΡΕΣΙΩΝ - ΠΡΟΤΕΙΝΟΜΕΝΩΝ ΥΠΗΡΕΣΙΩΝ

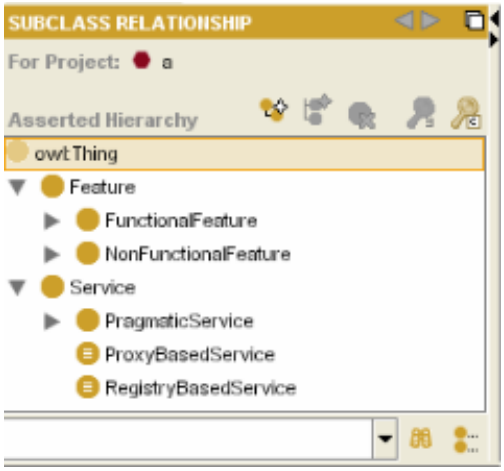
Σ' αυτό το κεφάλαιο περιγράφεται αναλυτικά η οντολογία που δημιουργήθηκε προκειμένου να εξυπηρετηθούν οι σκοποί της παρούσας διπλωματικής εργασίας. Όπως αναφέρθηκε και σε προηγούμενο κεφάλαιο, η οντολογία αυτή χρησιμοποιείται από τον HyperReggie σε περίπτωση που δεν καταστεί εφικτός ο εντοπισμός του *service* που ζητήθηκε από κάποιον *client*. Ουσιαστικά, μέσω της οντολογίας επιχειρείται η εύρεση μιας άλλης παρεμφερούς υπηρεσίας και μ' αυτόν τον τρόπο ο HyperReggie έχει τη δυνατότητα να προτείνει την παραπλήσια υπηρεσία.

8.1 Περιγραφή ιεραρχίας Υπο-Κλάσεων

Η συγκεκριμένη οντολογία αφορά ένα σύνολο από *services*, ορισμένα χαρακτηριστικά τους καθώς επίσης και συσχετίσεις μεταξύ τους. Θα πρέπει να σημειωθεί ότι για την κατασκευή της οντολογίας χρησιμοποιήθηκε το εργαλείο *Protégé* του Πανεπιστημίου του *Stanford* [26]. Όπως φαίνεται και στην εικόνα 12, η αρχική ιεραρχία (*asserted hierarchy*) της οντολογίας *ServiceOntology* περιλαμβάνει δύο κλάσεις: *Feature* και *Service*. Η πρώτη σχετίζεται με τα χαρακτηριστικά των *services* ενώ η δεύτερη περιλαμβάνει τις ίδιες τις υπηρεσίες, ενώ πρέπει να σημειωθεί ότι οι δύο *κλάσεις* έχουν οριστεί ως ξένες μεταξύ τους.

Εικόνα 12: Οι κλάσεις *Feature* και *Service*

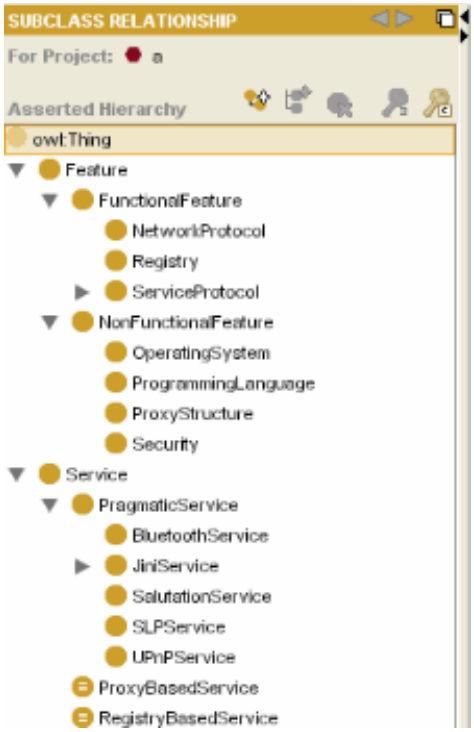
Κάτω από την κλάση *Feature* υπάρχουν δύο υπο-κλάσεις: *FunctionalFeature* και *NonFunctionalFeature*, ενώ κάτω από την κλάση *Service* βρίσκονται οι υπο-κλάσεις *PragmaticService*, *ProxyBasedService*, *RegistryBasedService* (Εικόνα 13). Η κλάση *FunctionalFeature* αφορά τα λεγόμενα «λειτουργικά» χαρακτηριστικά μιας υπηρεσίας, ενώ η *NonFunctionalFeature* σχετίζεται με τα «μη λειτουργικά», τα οποία θα παρουσιαστούν στη συνέχεια.



Εικόνα 13: Οι υπο-κλάσεις *FunctionalFeature*, *NonFunctionalFeature* και *PragmaticService, ProxyBasedService*, *RegistryBasedService*

Ως υπο-κλάσεις του *FunctionalFeature* θεωρούνται τα *NetworkProtocol* και *ServiceProtocol* (Εικόνα 14), με τα οποία εννοούνται ο τύπος δικτύου και το πρωτόκολλο που χρησιμοποιεί η υπηρεσία, που είναι ξένα μεταξύ τους. Αντίστοιχα, ως υπο-κλάσεις του *NonFunctionalFeature* ορίζονται τα επίσης ξένες κλάσεις:

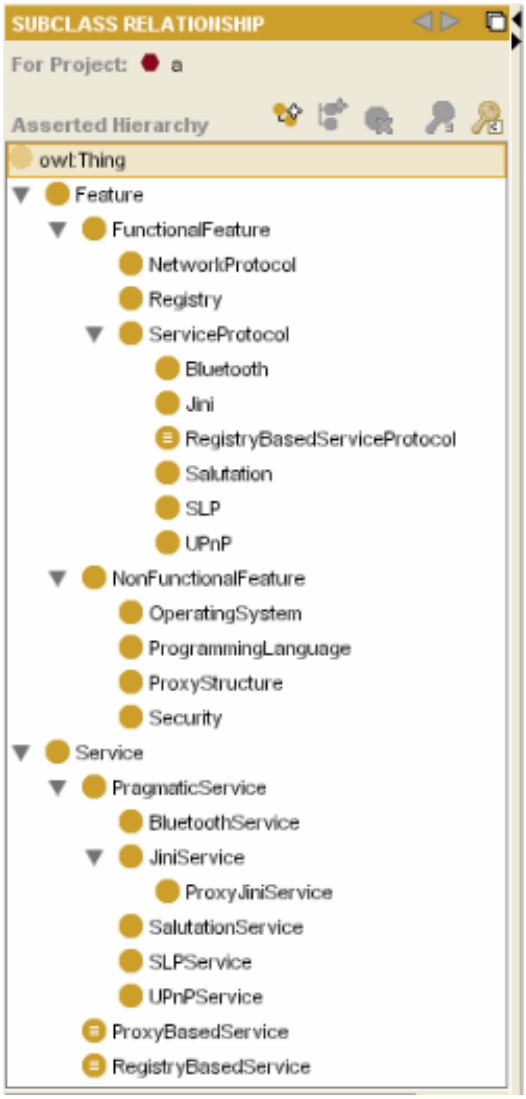
- *OperatingSystem*, (λειτουργικό σύστημα),
- *ProgrammingLanguage*, (γλώσσα προγραμματισμού),
- *ProxyStructure* (αν είναι *proxy-based* ή όχι) και
- *Security (*βαθμός και/ή τύπος ασφάλειας*)*.



Εικόνα 14: Οι υπο-κλάσεις των κλάσεων *FunctionalFeature*, *NonFunctionalFeature* και *PragmaticService*

Ομοίως, όπως επίσης φαίνεται στην εικόνα 14, οι ξένες υπο-κλάσεις του *PragmaticService* είναι οι εξής:

- *BluetoothService* (υπηρεσία υλοποιημένη με βάση το πρωτόκολλο *Bluetooth*),
- *JiniService* (υπηρεσία υλοποιημένη με βάση το πρωτόκολλο *Jini*),
- *SalutationService* (υπηρεσία υλοποιημένη με βάση το πρωτόκολλο *Salutation*),
- *SLPService* (υπηρεσία υλοποιημένη με βάση το πρωτόκολλο *SLP*) και
- *UPnPService* (υπηρεσία υλοποιημένη με βάση το πρωτόκολλο *UPnP*).



Εικόνα 15: Η πλήρης ιεραρχία υπο-κλάσεων της οντολογίας των υπηρεσιών

Τέλος, όπως απεικονίζεται στην εικόνα 15, κάτω από την κλάση *ServiceProtocol*, υπάρχουν οι υπο-κλάσεις:

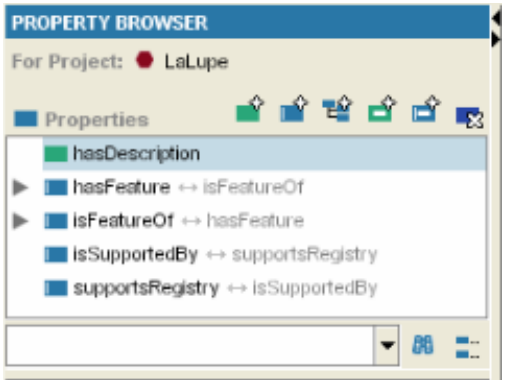
- *Bluetooth*,

- *Jini*,
- *RegistryBasedServiceProtocol,*
- *Salutation*,
- *SLP* και
- *UPnP*, τα οποία είναι εξ ορισμού ξένα.

8.2 Περιγραφή των Συσχετίσεων

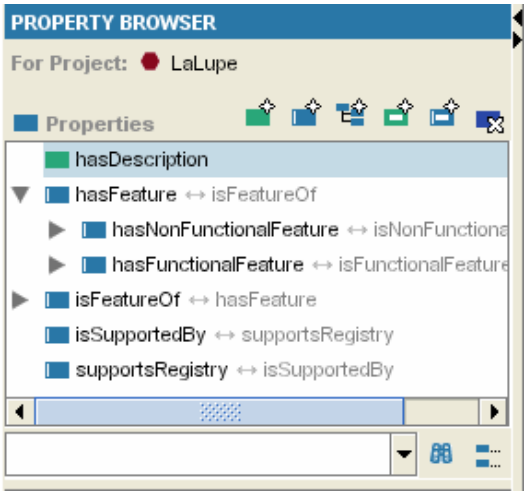
Σ' αυτήν την παράγραφο, περιγράφονται οι συσχετίσεις που έχουν συμπεριληφθεί στην οντολογία. Όπως φαίνεται και στην εικόνα 16, υπάρχουν τρεις βασικές συσχετίσεις: *hasDescription*, *hasFeature* και *supportsRegistry*. Αντίστοιχα υπάρχουν οι αντίστροφες συσχετίσεις αυτών των συναρτήσεων. Για παράδειγμα υπάρχει το *isFeatureOf*, το οποίο όμως αποτελεί την αντίστροφη συσχέτιση του *hasFeature*. Ουσιαστικά, τα *hasFeature* και *isFeatureOf* αποτελούν κοινές συσχετίσεις μεταξύ δύο στιγμιοτύπων, με τη προϋπόθεση ότι το πεδίο ορισμού του *hasFeature* ταυτίζεται με το πεδίο τιμών του *isFeatureOf* και το αντίστροφο. Θα πρέπει να σημειωθεί και μια ειδοποιός διαφορά μεταξύ της *hasDescription* και των υπολοίπων ιδιοτήτων: η *hasDescription* έχει οριστεί ως *datatype property*, ενώ οι υπόλοιπες είναι τύπου *object property*. Η διαφορά έγκειται στο γεγονός ότι οι *hasFeature* και *supportsRegistry* εμφανίζονται αποκλειστικά μεταξύ των στιγμιοτύπων, ενώ το *hasDescription* μπορεί να έχει ως πεδίο τιμών κάποιο *RDF literal* ή *XML Schema Datatype value*.

Η συσχέτιση *hasDescription* αφορά τη σχέση μιας υπηρεσίας με μια συγκεκριμένη περιγραφή. Η συσχέτιση *hasFeature* συνδέει μια υπηρεσία με κάποιο από τα χαρακτηριστικά που αναφέρθηκαν παραπάνω. Αντίστοιχα, το *isFeatureOf* συσχετίζει ένα χαρακτηριστικό με μια υπηρεσία. Τέλος, το *supportsRegistry* δηλώνει εάν μια υπηρεσία υποστηρίζει *registry*, δηλαδή αν η υπηρεσία καταχωρείται σε κάποια λίστα, μέσω της οποίας θ' αναζητηθεί από κάποιον *client*.



Εικόνα 16: Οι συσχετίσεις *hasDescription* και *hasFeature*

Στην εικόνα 17 φαίνονται οι υπο-συσχετίσεις του *hasFeature (*αντίστοιχα και του *isFeatureOf)*, που απαρτίζονται από τα: *hasNonFunctionalFeature* και hasFunctionalFeature, τα οποία έχουν ως πεδίο τιμών τα σύνολα των «μη-λειτουργικών» και «λειτουργικών» χαρακτηριστικών αντίστοιχα, που περιγράφηκαν παραπάνω.



Εικόνα 17: Οι υπο-συσχετίσεις της συσχέτισης *hasFeature*

Πρέπει να επισημανθεί ότι το πεδίο ορισμού και το πεδίο τιμών ορίζονται ρητά για κάθε συσχέτιση. Παραδείγματος χαριν, στην εικόνα 18 φαίνεται ότι το πεδίο ορισμού της συσχέτισης *hasFeature* είναι τύπου *service,* ενώ το πεδίο τιμών είναι τύπου *feature*.

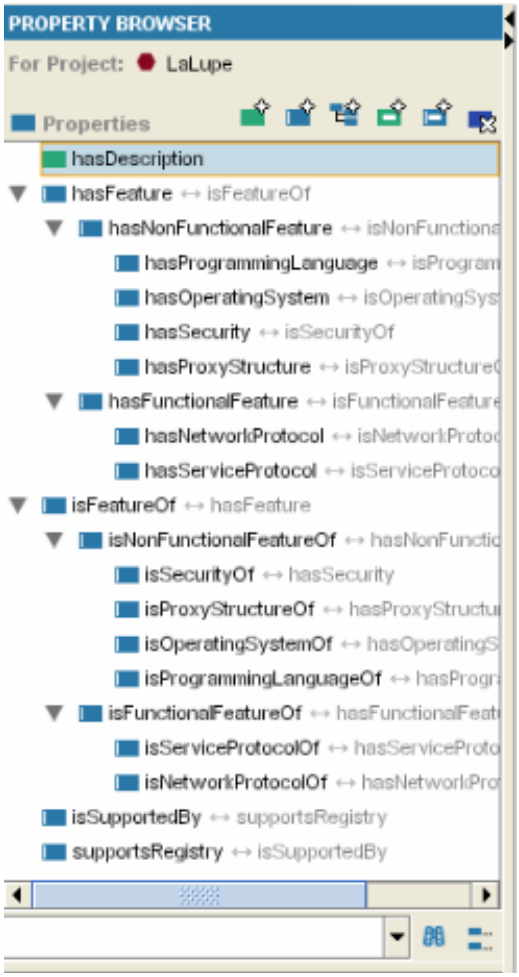
Εικόνα 18: Οι τιμές των πεδίων *Domain* και *Range* για τη σχέση *hasFeature*

Ομοίως, στην εικόνα 19 απεικονίζονται οι αντίστοιχοι περιορισμοί για τη συσχέτιση *isFeatureOf* και επαληθεύεται η χαρακτηριστική ιδιότητα των αντίστροφων συσχετίσεων που αναφέρθηκε παραπάνω.

Εικόνα 19: Οι τιμές των πεδίων *Domain* και *Range* για τη σχέση *isFeatureOf*

Στη συνέχεια, σύμφωνα με την εικόνα 20, έχουν οριστεί οι υπο-συσχετίσεις των *hasNonFunctionalFeature*, *hasFunctionalFeature*. Συνεπώς, όσον αφορά την *hasNonFunctionalFeature* υπάρχουν οι:

- *hasProgrammingLanguage*,
- *hasOperatingSystem,*
- *hasSecurity,*
- *hasProxyStructure,*



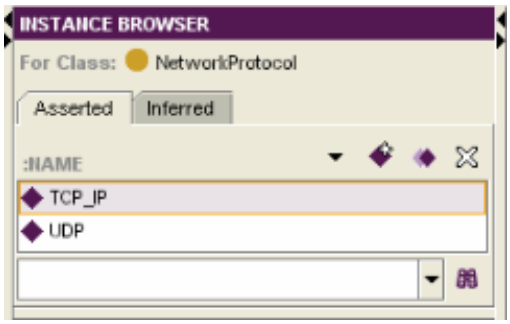
Εικόνα 20: Οι υπο-συσχετίσεις των *hasNonFunctionalFeature* και *hasFunctionalFeature*

ενώ η *hasFunctionalFeature* διάθετει τις ακόλουθες υπο-συσχετίσεις:

- *hasNetworkProtocol*
- *hasServiceProtocol*

8.3 Περιγραφή των Στιγμιοτύπων

Η παράγραφος αυτή εστιάζει στη σκιαγράφηση των στιγμιοτύπων που έχουν εισαχθεί στην οντολογία. Αρχικά, θα παρουσιαστούν τα στιγμιότυπα, που ανήκουν στην κατηγορία των «λειτουργικών» χαρακτηριστικών (δηλαδή της κλάσης *FunctionalFeature*). Συγκεκριμένα, όπως δείχνει και η εικόνα 21, υπάρχουν δύο στιγμιότυπα, *TCP_IP* και *UDP*, της υπο-κλάσης *NetworkProtocol*, δηλαδή της κατηγορίας των πρωτοκόλλων δικτύου.



Εικόνα 21: Τα στιγμιότυπα *TCP_IP* και *UDP* της κλάσης *NetworkProtocol*

Σύμφωνα με την εικόνα 22, το *TCP_IP* αποτελεί feature των:

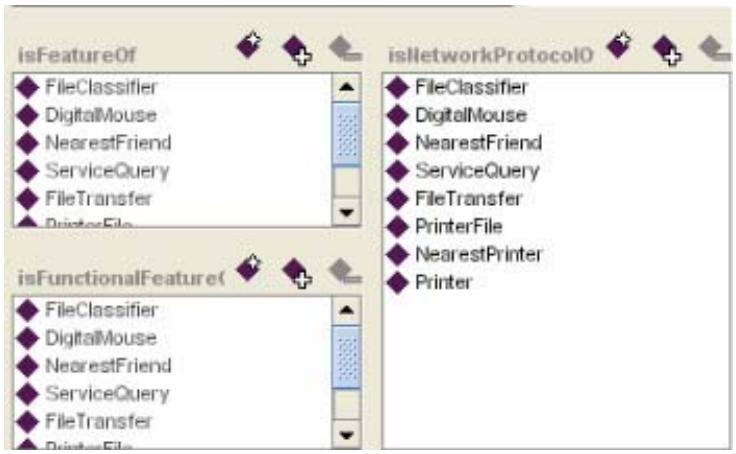
- *FileClassifier*,
- *DigitalMouse*,
- *NearestFriend*,
- *ServiceQuery*,
- *FileTransfer*,
- *PrinterFile*,
- *Printer*, τα οποία είναι υπαρκτές *υπηρεσίες*, που έχουν εισαχθεί επίσης ως στιγμιότυπα στην οντολογία και θα περιγραφούν παρακάτω.

Επίσης, το *TCP_IP* συνδέεται μέσω της συσχέτισης *isFunctionalFeature* με τα στιγμιότυπα:

- *FileClassifier*,
- *DigitalMouse*,
- *NearestFriend*,
- *ServiceQuery*,
- *FileTransfer*,
- *PrinterFile*,
- *Printer*.

Τέλος, αποτελεί το πρωτόκολλο δικτύου των:

- *FileClassifier*,
- *DigitalMouse*,
- *NearestFriend*,
- *ServiceQuery*,
- *FileTransfer*,
- *PrinterFile*,
- *Printer*.



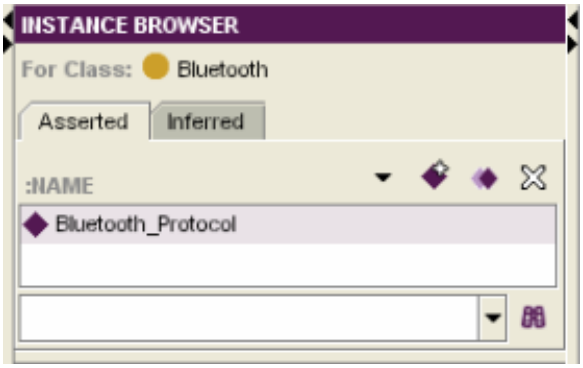
Εικόνα 22: Οι συσχετίσεις του στιγμιότυπου *TCP_IP*

Ομοίως, στην εικόνα 23, απεικονίζονται οι συσχετίσεις του *UDP* με άλλα στιγμιότυπα. Συνεπώς, το UDP αποτελεί:

- *feature* της υπηρεσίας *MobilePrinter*,
- *«λειτουργικό» feature* της υπηρεσίας *MobilePrinter* και
- *το πρωτόκολλο δικτύου* της υπηρεσίας *MobilePrinter*.

Εικόνα 23: Οι συσχετίσεις του στιγμιότυπου *UDP*

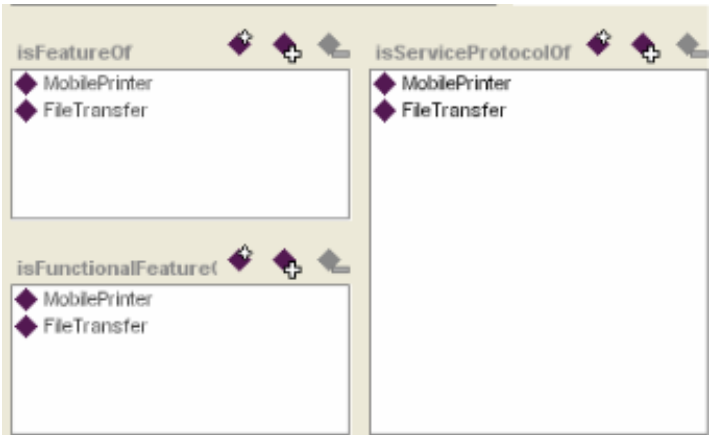
Όσον αφορά την υπο-κλάση *ServiceProtocol*, δηλαδή την κατηγορία πρωτοκόλλων εύρεσης υπηρεσιών, υπάρχει ένα στιγμιότυπο της κλάσης *Bluetooth*, το οποίο ονομάζεται *Bluetooth_Protocol* (εικόνα 24).



Εικόνα 24: Το στιγμιότυπο *Bluetooth_Protocol*

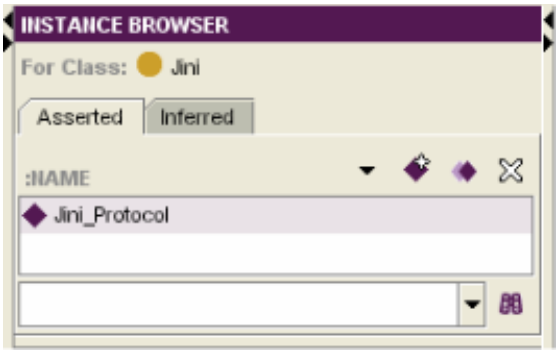
Στην επόμενη εικόνα (εικόνα 25) διακρίνονται οι συσχετίσεις στις οποίες εμπλέκεται το στιγμιότυπο *Bluetooth_Protocol*. Ως εκ τούτου, το εν λόγω στιγμιότυπο είναι:

- *feature* των *MobilePrinter* και *Printer,*
- «λειτουργικό» *feature* των *MobilePrinter* και *Printer*,
- πρωτόκολλο εύρεσης υπηρεσιών των *MobilePrinter* και *Printer.*



Εικόνα 25: Οι συσχετίσεις του στιγμιότυπου *Bluetooth_Protocol*

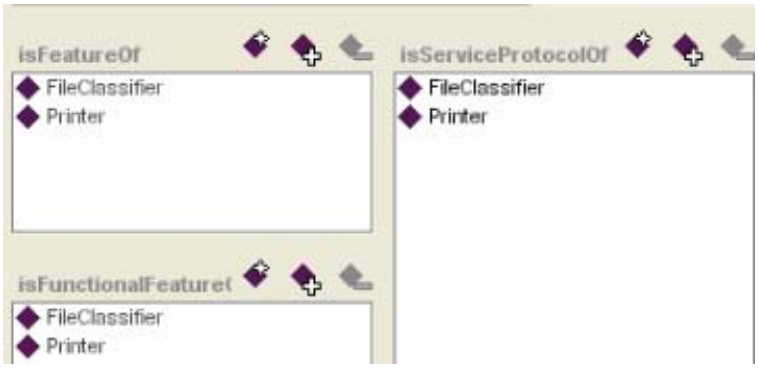
Ομοίως, υπάρχει το στιγμιότυπο *Jini_Protocol* της κλάσης *Jini* (εικόνα 26).



Εικόνα 26: Το στιγμιότυπο *Jini_Protocol*

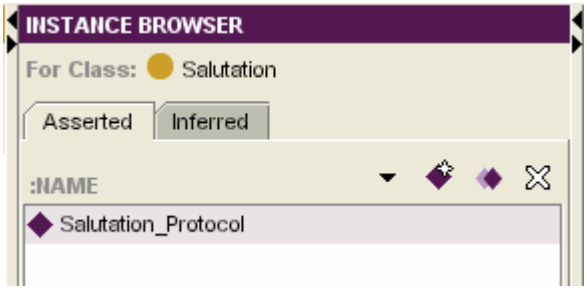
Ανάλογα ορίζονται και οι σχέσεις του *Jini_Protocol*, το οποίο σύμφωνα με την εικόνα 27, είναι:

- *feature* των *FileClassifier* και *Printer,*
- «λειτουργικό» *feature* των *FileClassifier* και *Printer*,
- πρωτόκολλο εύρεσης υπηρεσιών των *FileClassifier* και *Printer.*



Εικόνα 27: Οι συσχετίσεις του στιγμιότυπου *Jini_Protocol*

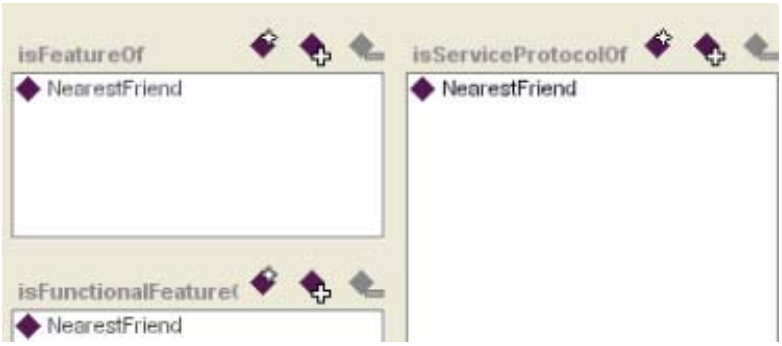
Ακριβώς με τον ίδιο τρόπο ορίζεται το στιγμιότυπο *Salutation_Protocol* της κλάσης *Salutation* (εικόνα 28).



Εικόνα 28: Το στιγμιότυπο *Salutation_Protocol*

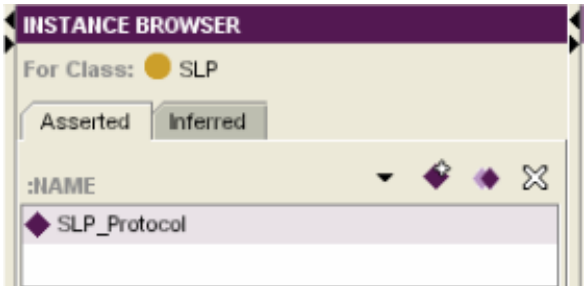
Οι συσχετίσεις του *Salutation_Protocol* φαίνονται στην εικόνα 29, σύμφωνα με την οποία το συγκεκριμένο στιγμιότυπο αποτελεί:

- *feature* του *NearestFrined*,
- «λειτουργικό» *feature* του *NearestFriend*,
- πρωτόκολλο εύρεσης υπηρεσιών του *NearestFriend*.



Εικόνα 29: Οι συσχετίσεις του στιγμιότυπου *Salutation_Protocol*

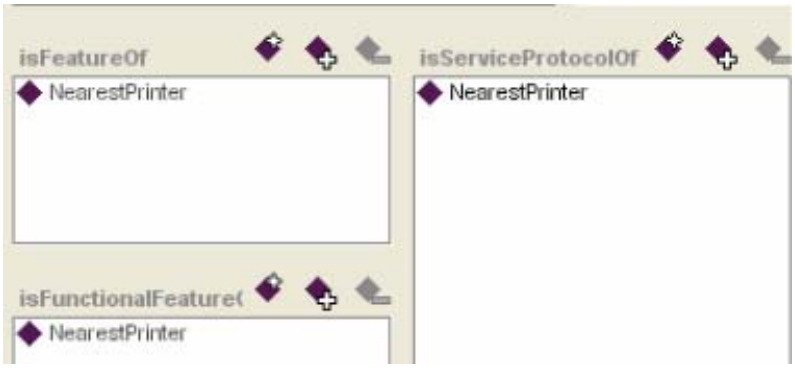
Ανάλογα ορίζεται το στιγμιότυπο *SLP_Protocol* της κλάσης *SLP* (εικόνα 30).



Εικόνα 30: Το στιγμιότυπο *SLP_Protocol*

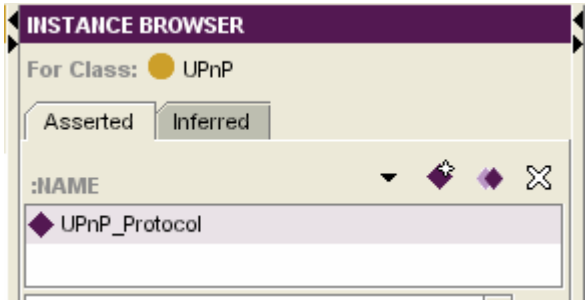
Το στιγμιότυπο *SLP_Protocol*, όπως δείχνει η εικόνα 31 είναι:

- *feature* του *NearestPrinter*,
- «λειτουργικό» *feature* του *NearestPrinter*,
- πρωτόκολλο εύρεσης υπηρεσιών του *NearestPrinter*.



Εικόνα 31: Οι συσχετίσεις του στιγμιότυπου *SLP_Protocol*

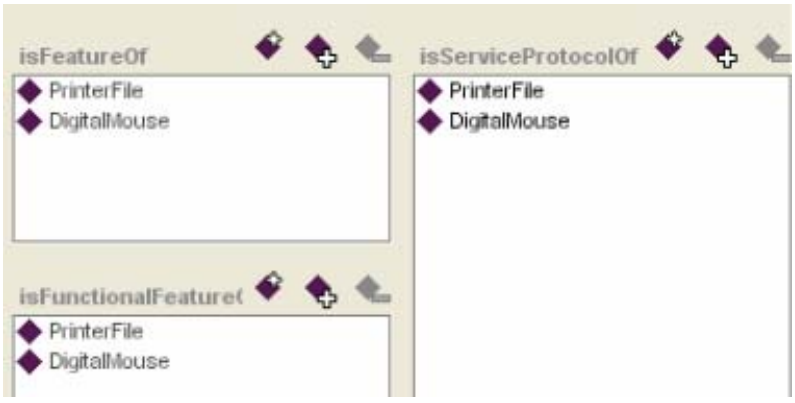
Τέλος, όσον αφορά τα πρωτόκολλα εύρεσης υπηρεσιών υπάρχει και το στιγμιότυπο *UPnP_Protocol* της κλάσης *UPnP* (εικόνα 32).



Εικόνα 32: Το στιγμιότυπο *UPnP_Protocol*

Αντίστοιχα οι σχέσεις του *UPnP_Protocol* απεικονίζονται στην εικόνα 33, σύμφωνα με την οποία το UPnP_Protocol είναι:

- *feature* των *DigitalMouse* και *PrinterFile*,
- «λειτουργικό» *feature* των *DigitalMouse* και *PrinterFile*,
- πρωτόκολλο εύρεσης υπηρεσιών των *DigitalMouse* και *PrinterFile*.

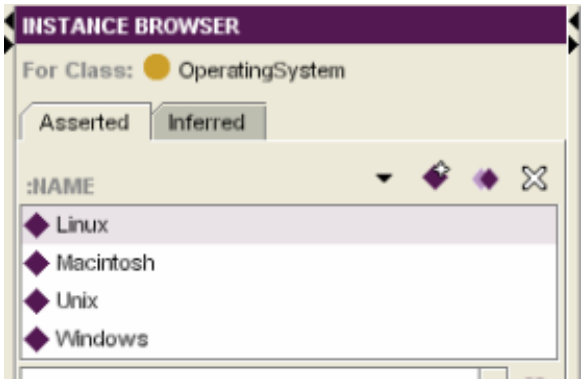


Εικόνα 33: Οι συσχετίσεις του στιγμιότυπου *UPnP_Protocol*

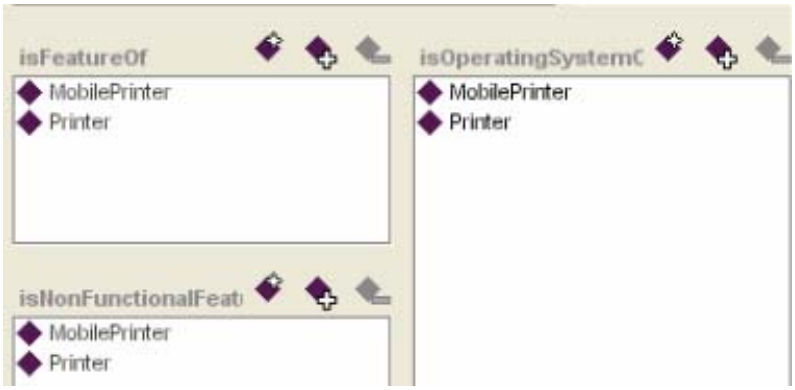
Στη συνέχεια παρουσιάζονται στιγμιότυπα όσον αφορά την κατηγορία των «μη-λειτουργικών» χαρακτηριστικών, δηλαδή της κλάσης *NonFunctionalFeature*. Επομένως για την κλάση *OperatingSystem* υπάρχουν τέσσερα στιγμιότυπα: *Linux*, *Macintosh*, *Unix* και *Windows* (εικόνα 34).

Όσον αφορά το στιγμιότυπο *Linux*, όπως δείχνει και η εικόνα 35, πρόκειται για:

- *feature* των *MobilePrinter* και *Printer*,
- μη «λειτουργικό» *feature* των *MobilePrinter* και *Printer*,
- πρωτόκολλο εύρεσης υπηρεσιών των *MobilePrinter* και *Printer*.



Εικόνα 34: Τα στιγμιότυπα της κλάσης *NonFunctionalFeature*



Εικόνα 35: Οι συσχετίσεις του στιγμιότυπου *Linux*

Στη συνέχεια, όπως φαίνεται στην εικόνα 36, το στιγμιότυπο *Macintosh* είναι *feature* των:

- *PrinterFile*,
- *Printer*,
- *FileTransfer*,
- *NearestPrinter*,
- *FileClassifier*.

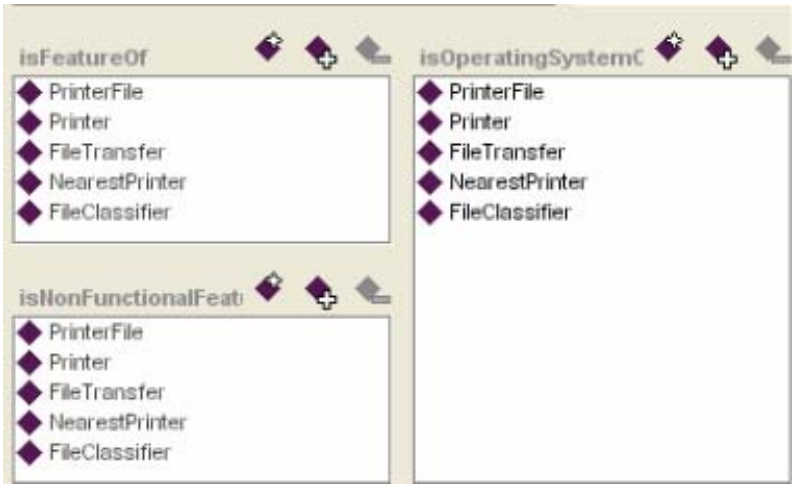
Επίσης, το *Macintosh* συνδέεται μέσω της συσχέτισης *isNonFunctionalFeature* με τα στιγμιότυπα:

- *PrinterFile*,
- *Printer*,
- *FileTransfer*,
- *NearestPrinter*,
- *FileClassifier*.

Τέλος, αποτελεί το λειτουργικό σύστημα που χρησιμοποιούν τα:

- *PrinterFile*,

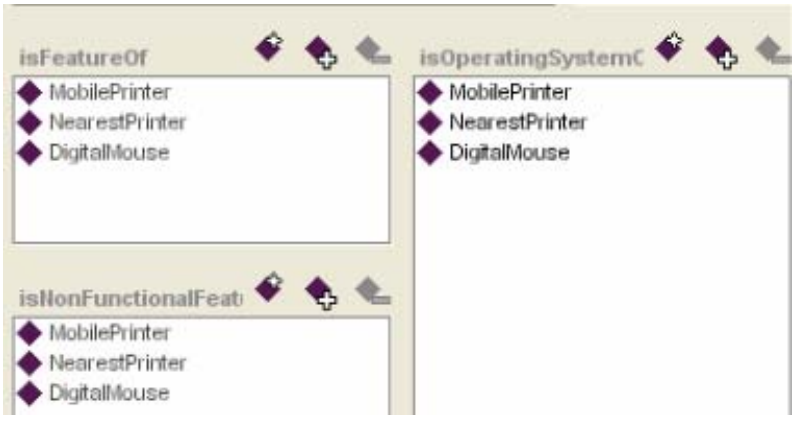
- *Printer*,
- *FileTransfer*,
- *NearestPrinter*,
- *FileClassifier*.



Εικόνα 36: Οι συσχετίσεις του στιγμιότυπου *Macintosh*

Όμοια, το στιγμιότυπο Unix αποτελεί:

- *feature* των *MobilePrinter*, *NearestPrinter* και *DigitalMouse*,
- μη «λειτουργικό» *feature* των *MobilePrinter*, *NearestPrinter* και *DigitalMouse*,
- λειτουργικό σύστημα των *MobilePrinter*, *NearestPrinter* και *DigitalMouse* (εικόνα 37).



Εικόνα 37: Οι συσχετίσεις του στιγμιότυπου *Unix*

Τέλος, σύμφωνα με την εικόνα 38, το στιγμιότυπο *Windows* είναι:
feature των:

- *MobilePrinter*,

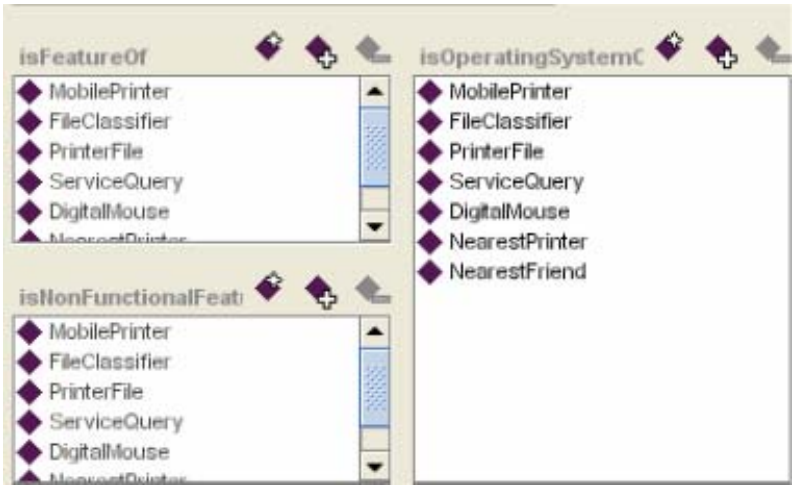
- *FileClassifier*,
- *PrinterFile*,
- *ServiceQuery*,
- *DigitalMouse*,
- *NearestPrinter,*
- *NearestFriend.*

Ακόμα, το στιγμιότυπο *Windows* συνδέεται μέσω της συσχέτισης *isNonFunctionalFeature* με τα στιγμιότυπα:

- *MobilePrinter*,
- *FileClassifier*,
- *PrinterFile*,
- *ServiceQuery*,
- *DigitalMouse*,
- *NearestPrinter,*
- *NearestFriend.*

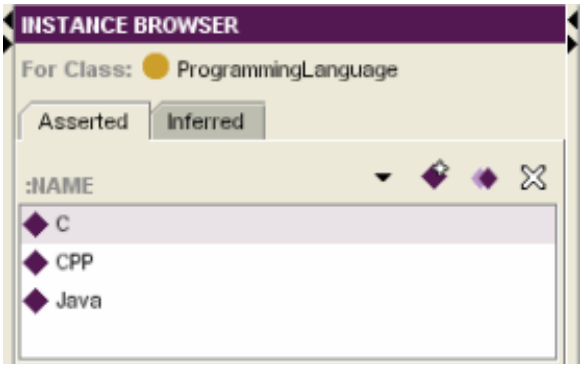
Επίσης, αποτελεί το λειτουργικό σύστημα που χρησιμοποιούν τα:

- *MobilePrinter*,
- *FileClassifier*,
- *PrinterFile*,
- *ServiceQuery*,
- *DigitalMouse*,
- *NearestPrinter,*
- *NearestFriend.*



Εικόνα 38: Οι συσχετίσεις του στιγμιότυπου *Windows*

Το επόμενο σύνολο στιγμιοτύπων αφορά την κλάση *ProgrammingLanguage*, όπου έχουν οριστεί τα στιγμιότυπα: *C*, *CPP*, *Java* (εικόνα 39).



Εικόνα 39: Τα στιγμιότυπα της κλάσης *ProgrammingLanguage*

Συγκεκριμένα, το στιγμιότυπο *C* διαθέτει τις σχέσεις που φαίνονται στην εικόνα 40. Συνεπώς, αποτελεί:

- *feature* του *PrinterFile*,
- μη «λειτουργικό» *feature* του *PrinterFile*,
- γλώσσα προγραμματισμού στην οποία είναι υλοποιημένο το *PrinterFile*.

Εικόνα 40: Οι συσχετίσεις του στιγμιότυπου *C*

Παρόμοια, σύμφωνα με την εικόνα 41, το στιγμιότυπο *CPP (*δηλαδή *C++)* είναι *feature* των:

- *DigitalMouse*,
- *FileTransfer*,
- *NearestFriend*,
- *NearestPrinter*,

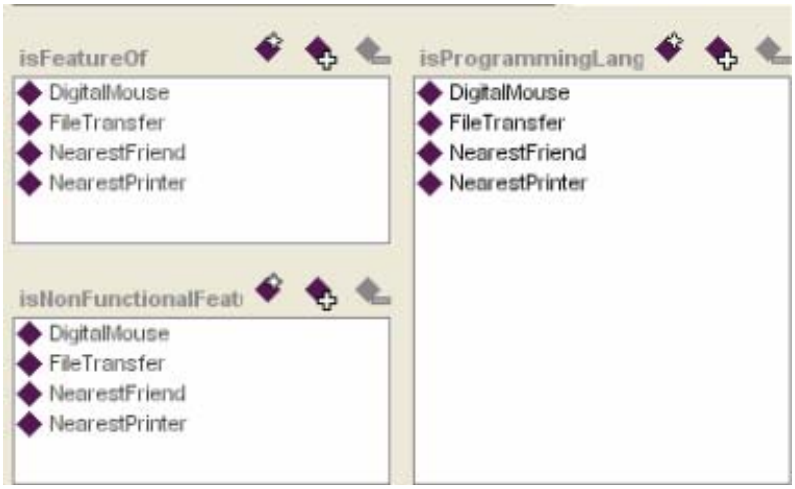
«μη-λειτουργικό» χαρακτηριστικό των:

- *DigitalMouse*,

- *FileTransfer*,
- *NearestFriend*,
- *NearestPrinter*,

και γλώσσα προγραμματισμού που έχει χρησιμοποιηθεί για την υλοποιήση των:

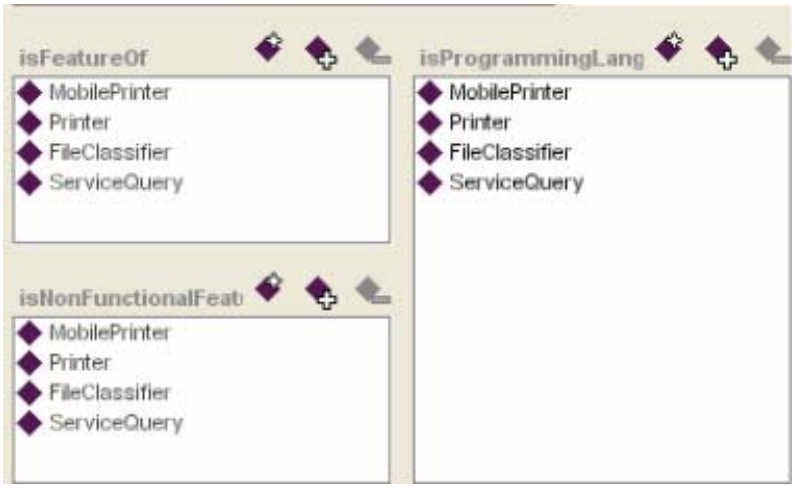
- *DigitalMouse*,
- *FileTransfer*,
- *NearestFriend*,
- *NearestPrinter*.



Εικόνα 41: Οι συσχετίσεις του στιγμιότυπου *CPP*

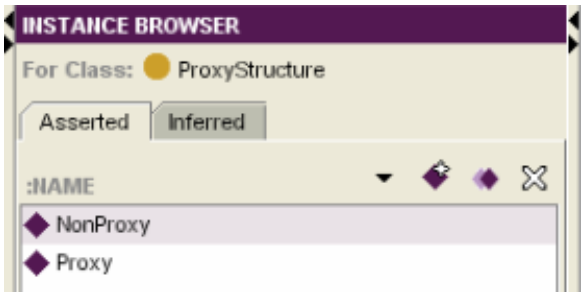
Τέλος, στην κατηγορία των γλωσσών προγραμματισμού υπάρχει και το στιγμιότυπο *Java*, το οποίο είναι:

- *feature* των *MobilePrinter, Printer, FileClassifier, ServiceQuery*,
- μη «λειτουργικό» *feature* των *MobilePrinter, Printer, FileClassifier, ServiceQuery*,
- γλώσσα υλοποίησης των *MobilePrinter, Printer, FileClassifier, ServiceQuery* (εικόνα 42).



Εικόνα 42: Οι συσχετίσεις του στιγμιότυπου *Java*

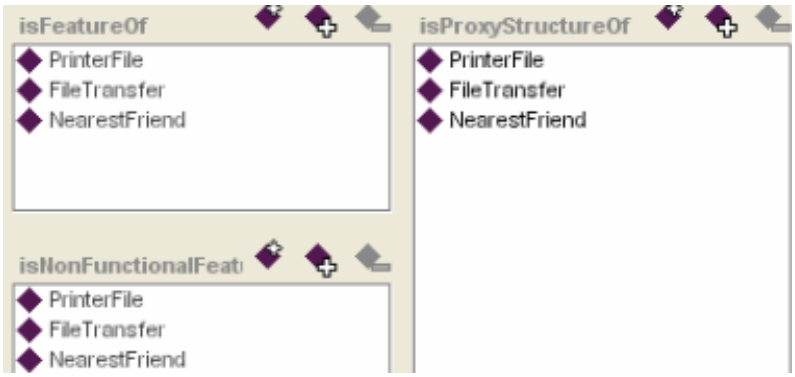
Το επόμενο σύνολο στιγμιοτύπων ανήκει στην κλάση *ProxyStructure*, που καθορίζει αν μια υπηρεσία είναι *ProxyBased* ή όχι. Έτσι , υπάρχουν δύο στιγμιότυπα: *Proxy* και *NonProxy* (εικόνα 43).



Εικόνα 43: Τα στιγμιότυπα της κλάσης *ProxyStructure*

Το στιγμιότυπο *NonProxy*, σύμφωνα με την εικόνα 44 αποτελεί:

- *feature* των *PrinterFile, FileTransfer, NearestFriend*,
- μη «λειτουργικό» *feature* των *PrinterFile, FileTransfer, NearestFriend*,
- στιγμιότυπο της κλάσης *ProxyStructure* για τα *PrinterFile, FileTransfer, NearestFriend* (δηλαδή τα συγκεκριμένα *services* δεν είναι proxy-based).



Εικόνα 44: Οι συσχετίσεις του στιγμιότυπου *NonProxy*

Αντίστοιχα, το στιγμιότυπο *Proxy*, όπως δείχνει η εικόνα 45, είναι *feature* των:

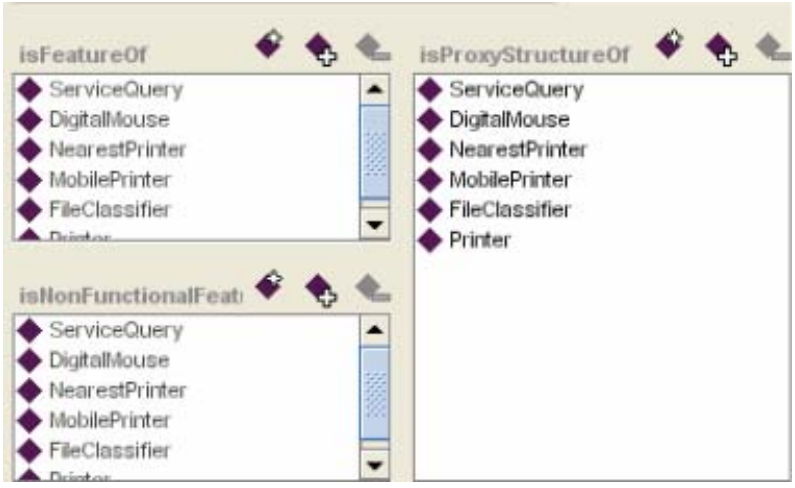
- *ServiceQuery*,
- *DigitalMouse*,
- *NearestPrinter*,
- *MobilePrinter,*
- *FileClassifier*
- *Printer*,

«μη-λειτουργικό» χαρακτηριστικό των:

- *ServiceQuery*,
- *DigitalMouse*,
- *NearestPrinter*,
- *MobilePrinter,*
- *FileClassifier*
- *Printer*,

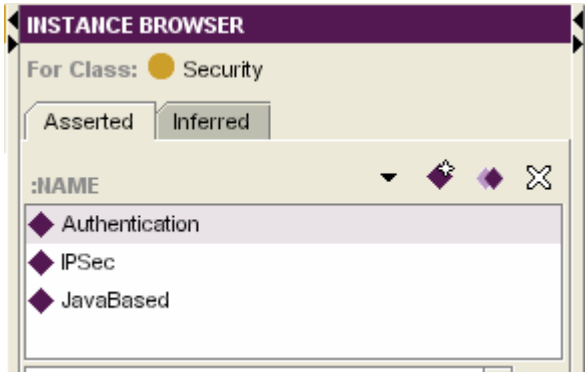
και στιγμιότυπο της κλάσης *ProxyStructure* για τα:

- *ServiceQuery*,
- *DigitalMouse*,
- *NearestPrinter*,
- *MobilePrinter,*
- *FileClassifier*
- *Printer* (δηλαδή τα συγκεκριμένα *services* είναι *proxy-based*).



Εικόνα 45: Οι συσχετίσεις του στιγμιότυπου *Proxy*

Η τελευταία κατηγορία «μη-λειτουργικών» χαρακτηριστικών είναι η κλάση *Security*, όπου έχουν εισαχθεί τρία στιγμιότυπα: *Authentication*, *IPSec*, *JavaBased* (εικόνα 46).



Εικόνα 46: Τα στιγμιότυπα της κλάσης *Security*

Το στιγμιότυπο *Authentication*, σύμφωνα με την εικόνα 47, είναι:

- *feature* του *NearestFriend*,
- μη «λειτουργικό» *feature* του *NearestFriend*,
- τύπος ασφάλειας για το *NearestFriend*.

Εικόνα 47: Οι συσχετίσεις του στιγμιότυπου *Authentication*

Στη συνέχεια το στιγμιότυπο IPSec αποτελεί:

- *feature* του *DigitalMouse*,
- μη «λειτουργικό» *feature* του *DigitalMouse*,
- τύπο ασφάλειας για το *DigitalMouse* (εικόνα 48).

Εικόνα 48: Οι συσχετίσεις του στιγμιότυπου *IPSec*

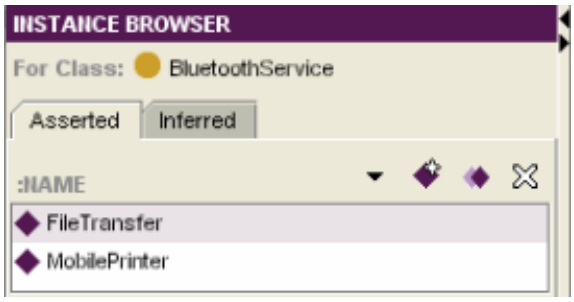
Τέλος, το στιγμιότυπο *JavaBased* είναι:

- *feature* του *ServiceQuery*,
- μη «λειτουργικό» *feature* του *ServiceQuery*,
- τύπος ασφάλειας για το *ServiceQuery* (εικόνα 49).

Εικόνα 49: Οι συσχετίσεις του στιγμιότυπου *JavaBased*

Το τελευταίο σύνολο στιγμιοτύπων αφορά τις ίδιες τις υπηρεσίες, που έχουν εισαχθεί ως *στιγμιότυπα* των υπο-κλάσεων *BluetoothService*, *JiniService*, *SalutationService*, *SLPService* και *UpnPService*.

Ξεκινώντας από την υπο-κλάση *BluetoothService*, υπάρχουν δύο στιγμιότυπα: *FileTransfer* και *MobilePrinter* (εικόνα 50).



Εικόνα 50: Τα στιγμιότυπα *FileTransfer* και *MobilePrinter*

Στην εικόνα 51 απεικονίζονται οι συσχετίσεις του στιγμιότυπου *FileTransfer*. Έτσι το *FileTransfer* έχει τα εξής *features*:

- *CPP*,
- *Macintosh,*
- *NonProxy,*
- *TCP_IP,*
- *Bluetooth_Protocol,*
- από τα οποία τα «λειτουργικά» χαρακτηριστικά είναι:
- *TCP_IP,*

- *Bluetooth_Protocol,*

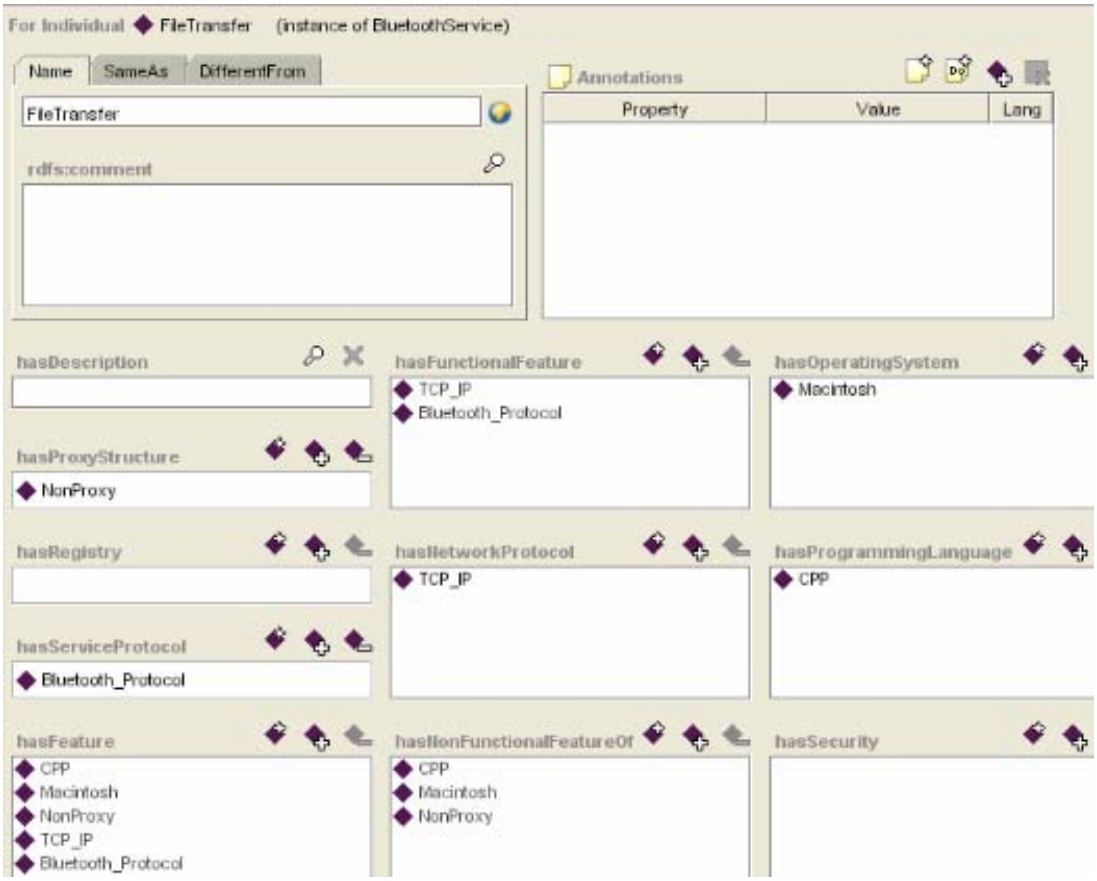
και τα «μη-λειτουργικά» χαρακτηριστικά:

- *CPP,*
- *Macintosh,*
- *NonProxy.*

Επιπρόσθετα, το *FileTransfer* δεν είναι *proxy-based* (καθότι η σχέση *hasProxyStructure* έχει ως *Range* το στιγμιότυπο *NonProxy*), διαθέτει πρωτόκολλο εύρεσης υπηρεσιών το *Bluetooth_Protocol* και πρωτόκολλο δικτύου το *TCP_IP*, είναι υλοποιημένο σε *C++* και χρησιμοποιεί *Macintosh* ως λειτουργικό σύστημα.

Ανάλογα στην εικόνα 52, φαίνονται οι συσχετίσεις του στιγμιότυπου *MobilePrinter*. Συνεπώς το εν λόγω στιγμιότυπο έχει τα εξής *features*:

- *Java*,
- *Windows,*
- *Linux,*
- *Unix,*
- *UDP,*
- *Proxy,*
- *Bluetooth_Protocol,*



Εικόνα 51: Οι συσχετίσεις του στιγμιότυπου *FileTransfer*

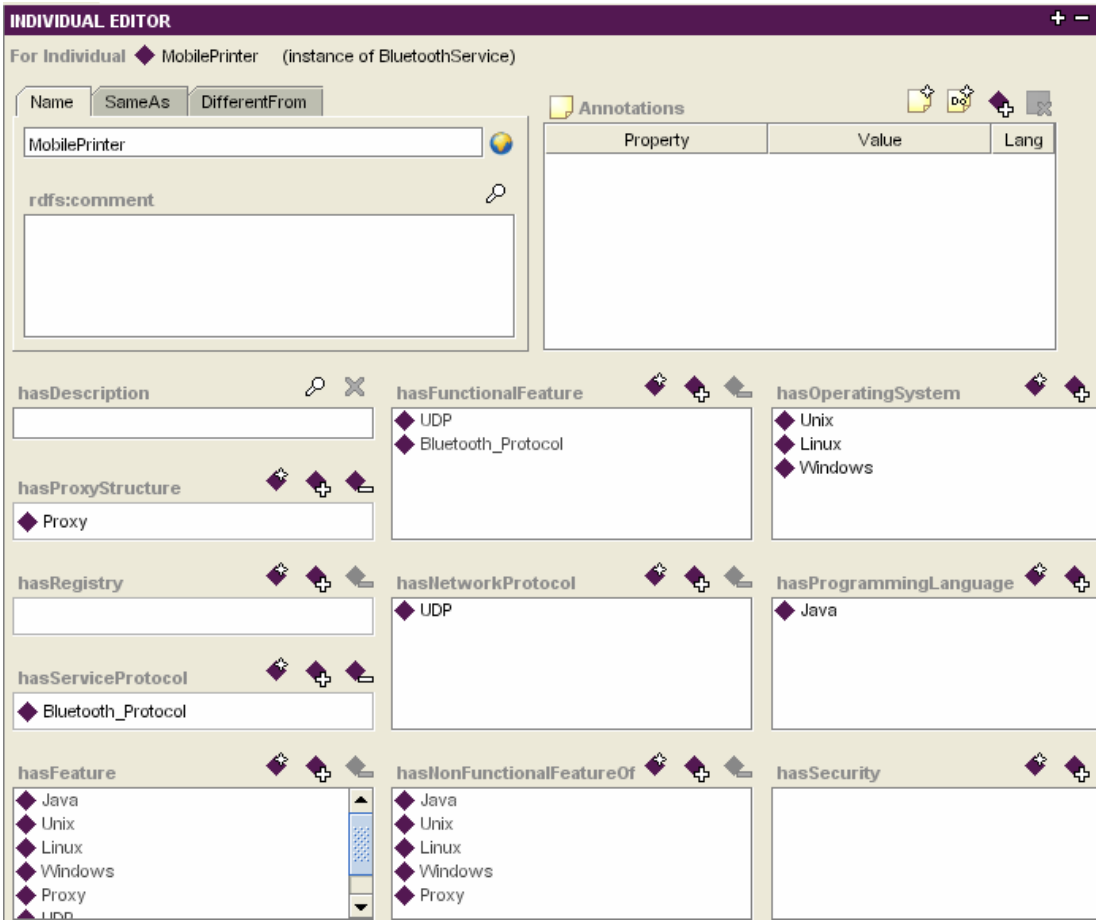
από τα οποία τα «λειτουργικά» χαρακτηριστικά είναι:

- *UDP,*
- *Bluetooth_Protocol,*

και τα «μη-λειτουργικά» χαρακτηριστικά:

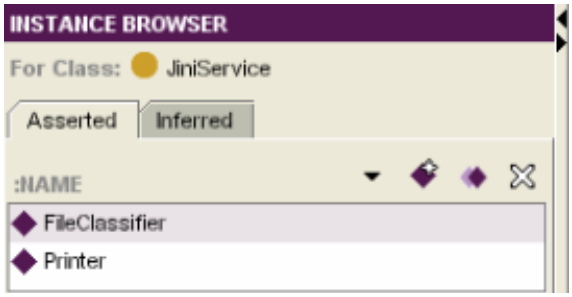
- *Java*,
- *Windows,*
- *Linux,*
- *Unix,*
- *Proxy.*

Επιπρόσθετα, το *MobilePrinter* είναι *proxy-based* (καθότι η σχέση *hasProxyStructure* έχει ως *Range* το στιγμιότυπο *Proxy*), διαθέτει πρωτόκολλο εύρεσης υπηρεσιών το *Bluetooth_Protocol* και πρωτόκολλο δικτύου το *UDP*, είναι υλοποιημένο σε *Java* και χρησιμοποιεί *Linux, Unix* ή *Windows* ως λειτουργικό σύστημα.



Εικόνα 52: Οι συσχετίσεις του στιγμιότυπου *MobilePrinter*

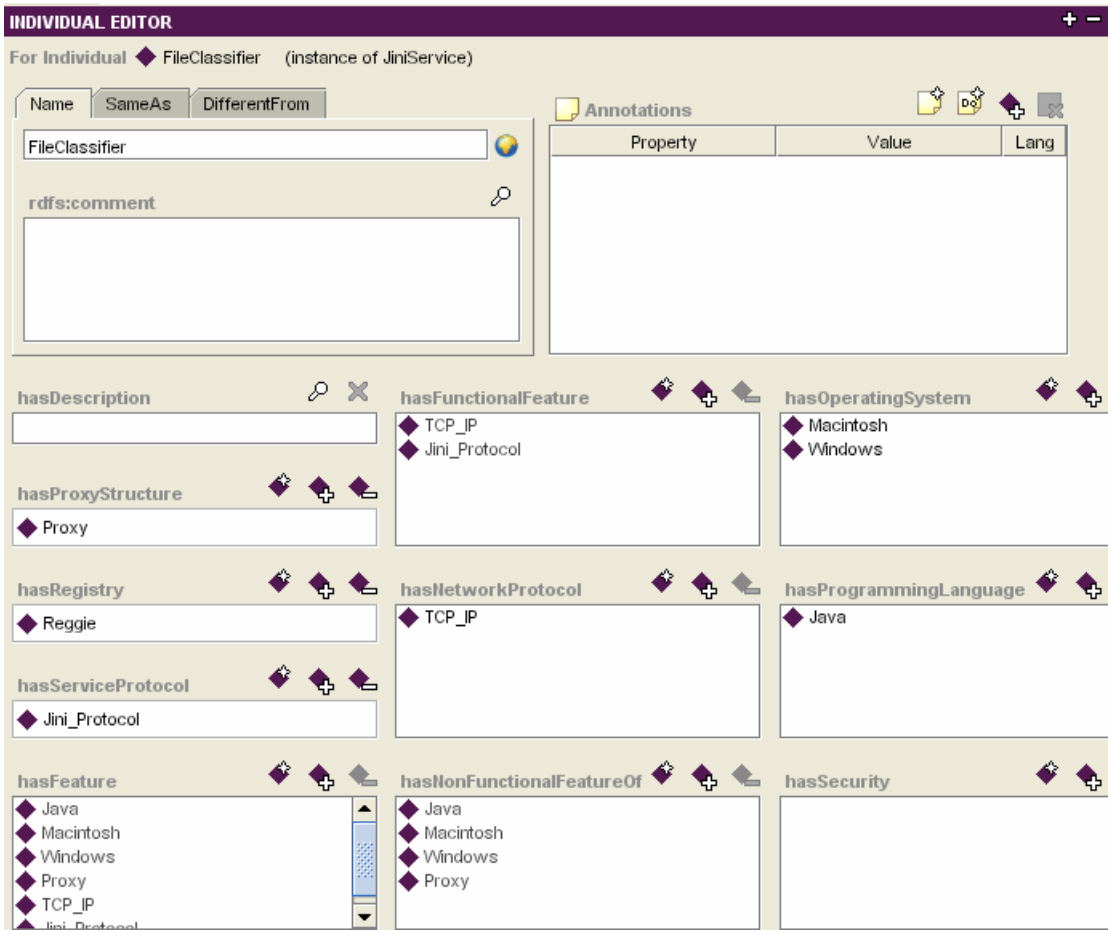
Στη συνέχεια, όσον αφορά τη υπο-κλάση *JiniService*, υπάρχουν τα στιγμιότυπα *FileClassifier*, *Printer*, όπως φαίνεται στην εικόνα 53.



Εικόνα 53: Τα στιγμιότυπα *FileClassifier* και *Printer*

Το *FileClassifier* εμπλέκεται στις σχέσεις που απεικονίζονται στην εικόνα 54. Επομένως, το *FileClassifier* έχει τα εξής *features*:

- *Java*,
- *Macintosh,*
- *Windows,*
- *Proxy,*
- *TCP_IP*
- *Jini_Protocol,*



Εικόνα 54: Οι συσχετίσεις του στιγμιότυπου *FileClassifier*

από τα οποία τα «λειτουργικά» χαρακτηριστικά είναι:

- *TCP_IP,*
- *Jini_Protocol,*

και τα «μη-λειτουργικά» χαρακτηριστικά:

- *Java,*
- *Macintosh,*
- *Windows,*
- *Proxy.*

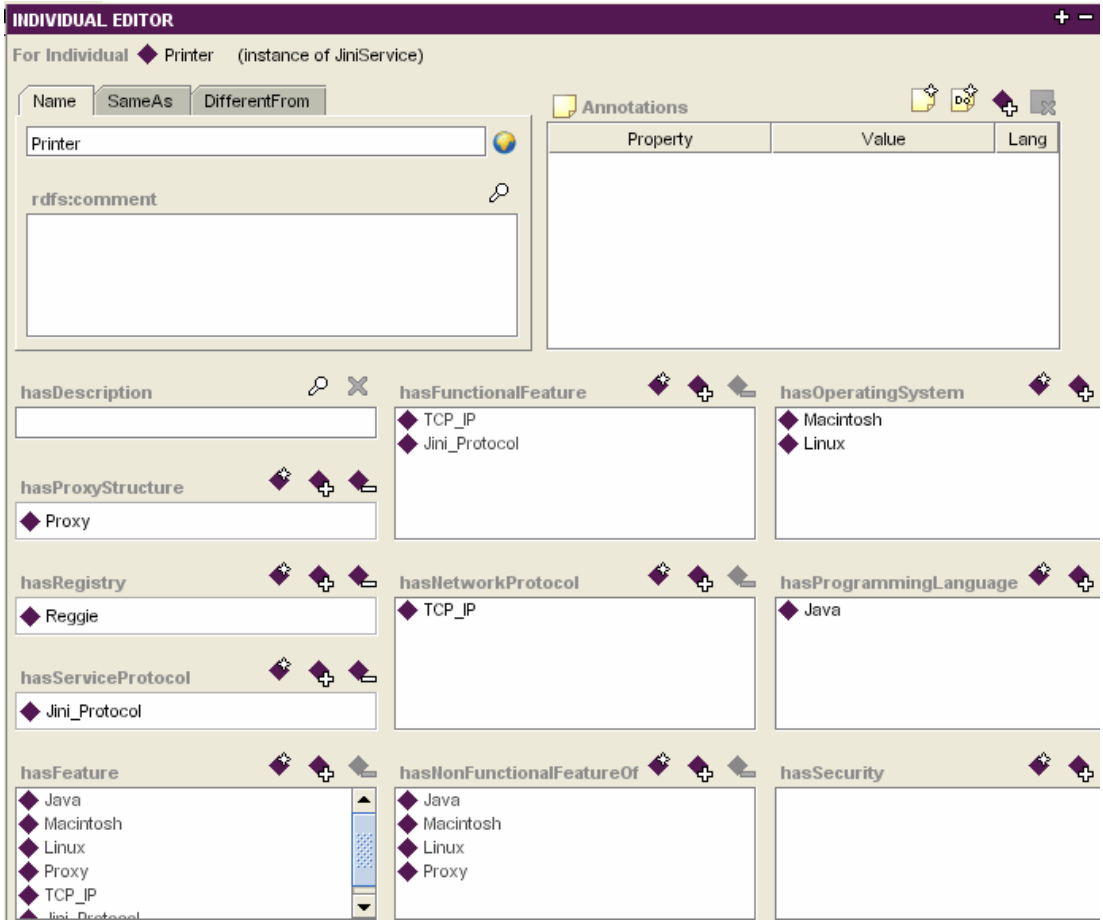
Επιπρόσθετα, το *FileClassifier* είναι *proxy-based* (καθότι η σχέση *hasProxyStructure* έχει ως *Range* το στιγμιότυπο *Proxy*), διαθέτει πρωτόκολλο εύρεσης υπηρεσιών το *Jini_Protocol* και πρωτόκολλο δικτύου το *TCP_IP*, είναι υλοποιημένο σε *Java* και χρησιμοποιεί *Macintosh* ή *Windows* ως λειτουργικό σύστημα.

Ομοίως, το *Printer*, σύμφωνα με την εικόνα 55, διαθέτει:

τα εξής *features*:

- *Java*,
- *Macintosh,*

- *Linux,*
- *Proxy,*
- *TCP_IP*
- *Jini_Protocol,*



Εικόνα 55: Οι συσχετίσεις του στιγμιότυπου *Printer*

από τα οποία τα «λειτουργικά» χαρακτηριστικά είναι τα:

- *TCP_IP,*
- *Jini_Protocol,*

και τα «μη-λειτουργικά» χαρακτηριστικά:

- *Java,*
- *Macintosh,*
- *Linux,*
- *Proxy.*

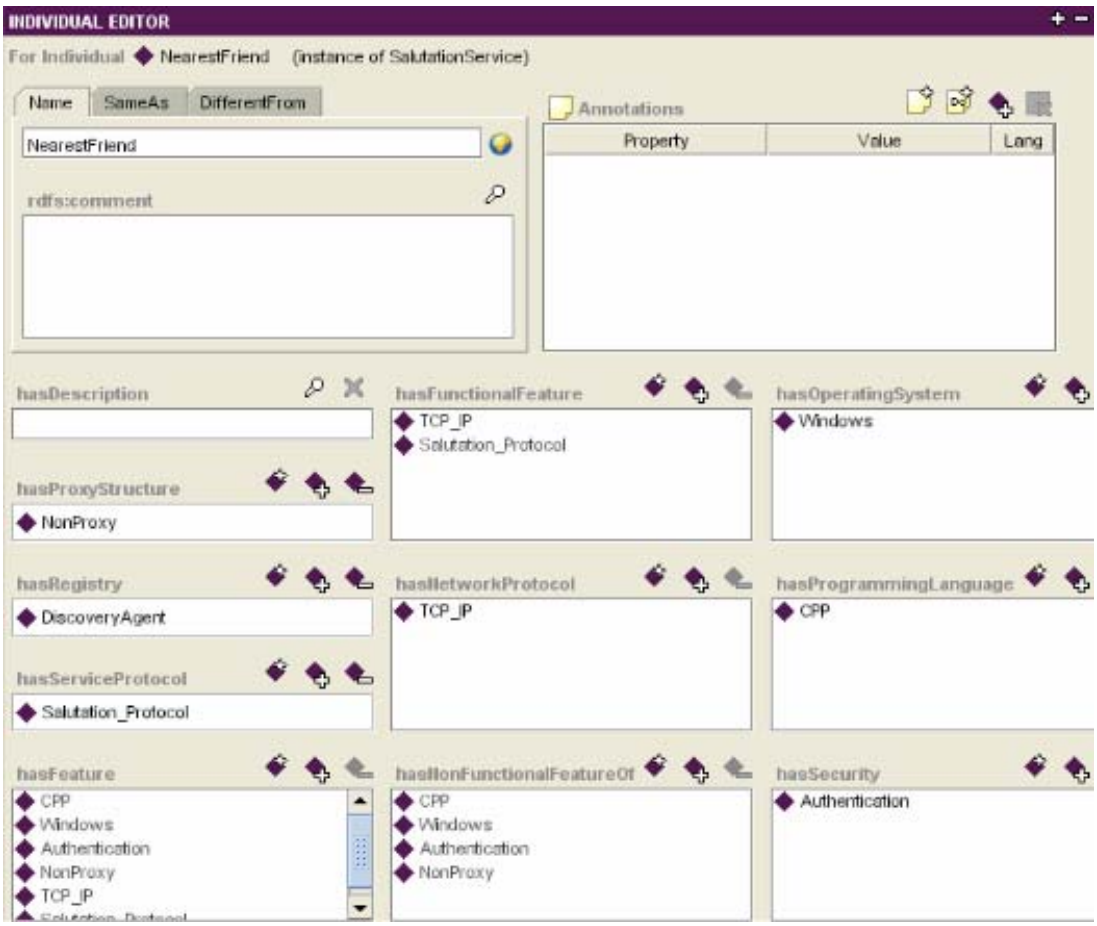
Επιπρόσθετα, το *Printer* είναι *proxy-based* (καθότι η σχέση *hasProxyStructure* έχει ως *Range* το στιγμιότυπο *Proxy*), διαθέτει πρωτόκολλο εύρεσης υπηρεσιών το *Jini_Protocol* και πρωτόκολλο δικτύου το *TCP_IP*, είναι υλοποιημένο σε *Java* και χρησιμοποιεί *Macintosh* ή *Linux* ως λειτουργικό σύστημα.

Η επόμενη υπο-κλάση υπαρκτών υπηρεσιών (*pragmatic services*) είναι η *SalutationService*, της οποίας έχει οριστεί ένα στιγμιότυπο, το *NearestFriend* (εικόνα 56).

Εικόνα 56: Το στιγμιότυπο *NearestFriend*

Το στιγμιότυπο *NearestFrined*, όπως απεικονίζεται και στην εικόνα 57, έχει τα εξής *features*:

- *CPP*,
- *Windows,*
- *Authentication,*
- *NonProxy,*
- *TCP_IP*
- *Salutation_Protocol,*



Εικόνα 57: Οι συσχετίσεις του στιγμιότυπου *NearestFriend*

από τα οποία τα «λειτουργικά» χαρακτηριστικά είναι τα:

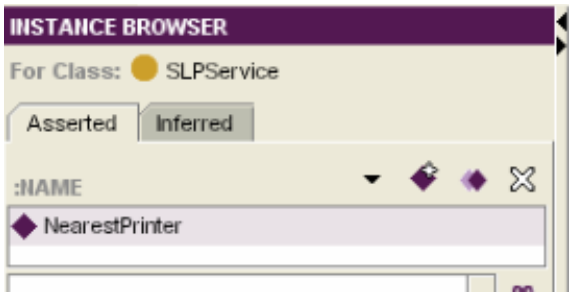
- *TCP_IP,*
- *Salutation_Protocol,*

και τα «μη-λειτουργικά» χαρακτηριστικά:

- *CPP,*
- *Windows,*
- *Authentication,*
- *NonProxy.*

Επιπρόσθετα, το *NearestFriend* δεν είναι *proxy-based* (καθότι η σχέση *hasProxyStructure* έχει ως *Range* το στιγμιότυπο Non*Proxy*), διαθέτει πρωτόκολλο εύρεσης υπηρεσιών το *Salutation_Protocol* και πρωτόκολλο δικτύου το *TCP_IP*, είναι υλοποιημένο σε *C++* και χρησιμοποιεί *Windows* ως λειτουργικό σύστημα. Όσον αφορά το συγκεκριμένο στιγμιότυπο, υπάρχει μια επιπλέον πληροφορία που σχετίζεται με τον τύπο ασφάλειας (η σχέση *hasSecurity* έχει ως *Range* την τιμή *Authentication*).

Το επόμενο στιγμιότυπο αφορά την κλάση *SLPService* και ονομάζεται *NearestPrinter* (εικόνα 58).



Εικόνα 58: Το στιγμιότυπο *NearestPrinter*

Σύμφωνα με την εικόνα 59, το στιγμιότυπο *NearestPrinter* διαθέτει τα ακόλουθα *features*:

- *CPP*,
- *Unix,*
- *Macintosh,*
- *Windows,*
- *IPSec,*
- *Proxy,*
- *TCP_IP,*
- *SLP_Protocol,*

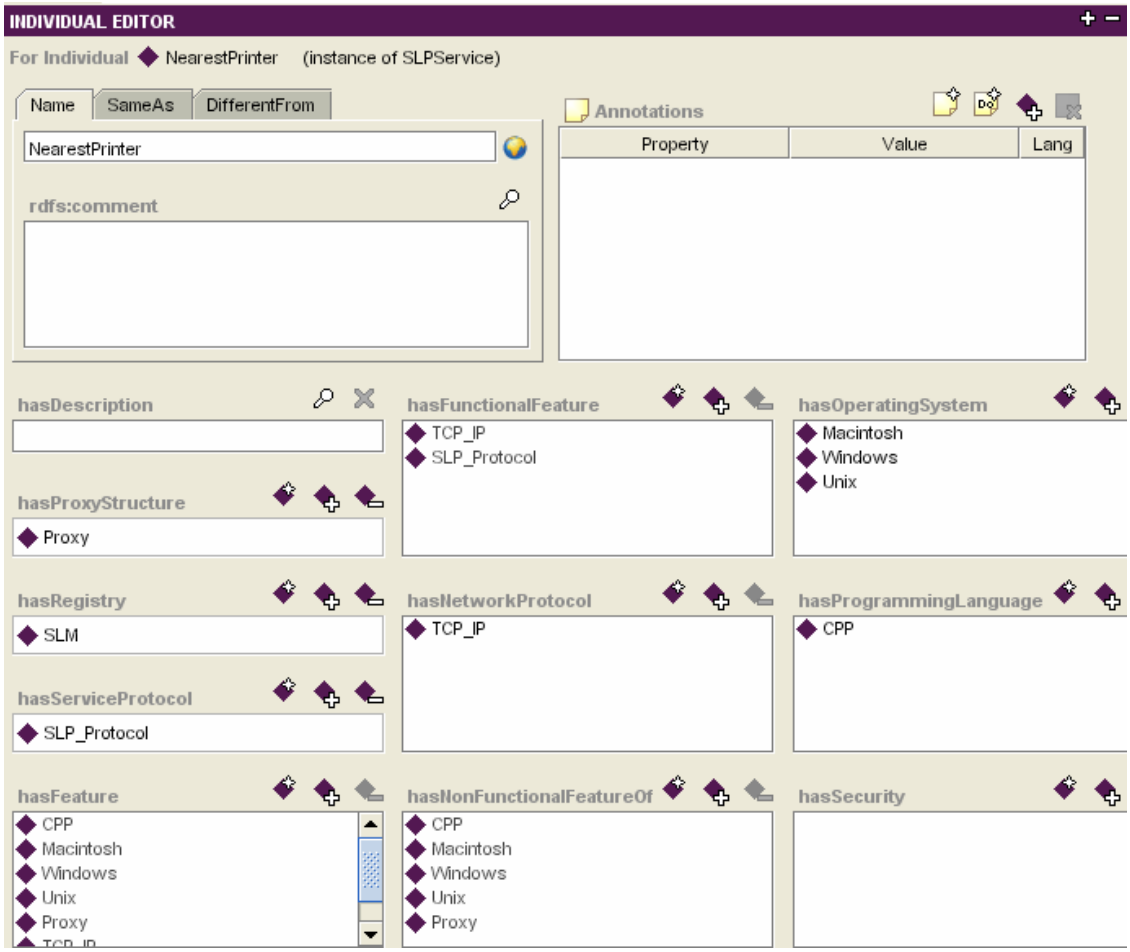
Από τα οποία τα «λειτουργικά» χαρακτηριστικά είναι τα:

- *TCP_IP,*
- *SLP_Protocol,*

και τα «μη-λειτουργικά» χαρακτηριστικά:

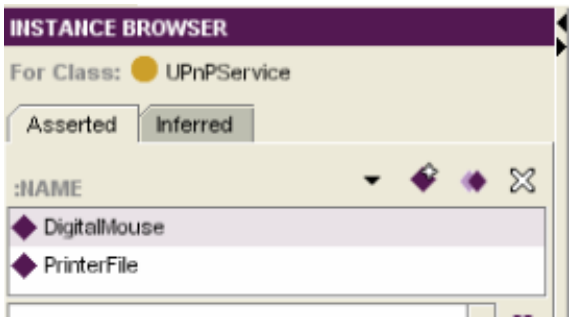
- *CPP,*
- *Unix,*
- *Macintosh,*
- *Windows,*
- *IPSec,*
- *Proxy.*

Ακόμα το *NearestPrinter* είναι *proxy-based* (καθότι η σχέση *hasProxyStructure* έχει ως *Range* το στιγμιότυπο *Proxy*), διαθέτει πρωτόκολλο εύρεσης υπηρεσιών το *SLP_Protocol* και πρωτόκολλο δικτύου το *TCP_IP*, είναι υλοποιημένο σε *C++* και χρησιμοποιεί *Windows, Macintosh* ή *Unix* ως λειτουργικό σύστημα. Όσον αφορά το εν λόγω στιγμιότυπο, η σχέση *hasSecurity* έχει ως πεδίο τιμών την τιμή *IPSec*.



Εικόνα 59: Οι συσχετίσεις του στιγμιότυπου *NearestPrinter*

Η τελευταία υπο-κλάση υπαρκτών υπηρεσιών είναι η *UPnPService*, που διαθέτει δύο στιγμιότυπα: *DigitalMouse*, *PrinterFile* (εικόνα 60).

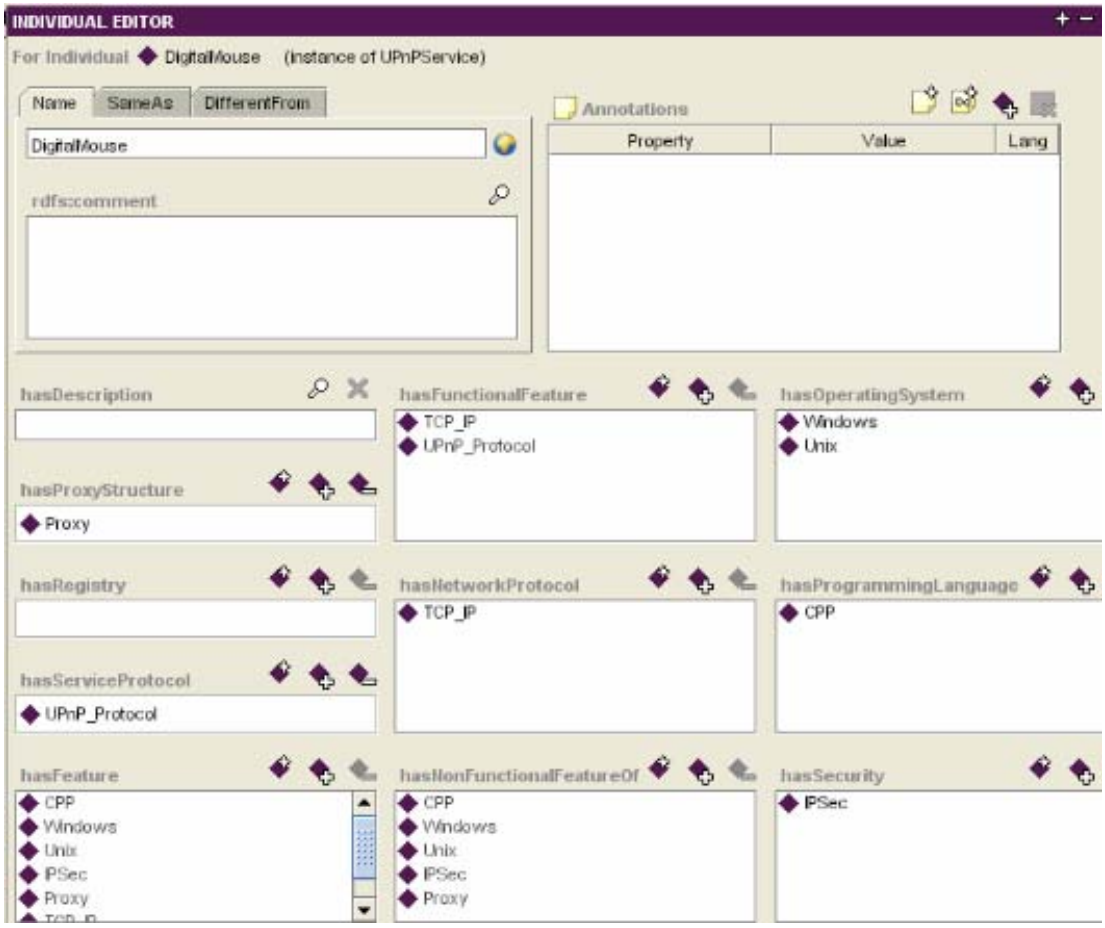


Εικόνα 60: Τα στιγμιότυπα *DigitalMouse* και *PrinterFile*

Όπως είναι φανερό από την εικόνα 61, το *DigitalMouse* έχει τα εξής *features*:

- *C*,
- *Macintosh,*

- *Windows,*
- *JavaBased,*
- *NonProxy,*
- *TCP_IP,*
- *UPnP_Protocol,*



Εικόνα 61: Οι συσχετίσεις του στιγμιότυπου *DigitalMouse*

από τα οποία τα «λειτουργικά» χαρακτηριστικά είναι τα:

- *TCP_IP,*
- *UPnP_Protocol,*

και τα «μη-λειτουργικά» χαρακτηριστικά:

- *C,*
- *Macintosh,*
- *Windows,*
- *JavaBased,*
- *NonProxy.*

Επίσης το *DigitalMouse* δεν είναι *proxy-based* (καθότι η σχέση *hasProxyStructure* έχει ως *Range* το στιγμιότυπο Non*Proxy*), διαθέτει πρωτόκολλο εύρεσης υπηρεσιών το *UPnP_Protocol* και πρωτόκολλο δικτύου το *TCP_IP*, είναι υλοποιημένο σε *C* και χρησιμοποιεί *Windows* ή *Macintosh* ως λειτουργικό σύστημα. Τέλος, το εν λόγω στιγμιότυπο έχει ως πεδίο τιμών της σχέσης *hasSecurity,* την τιμή *IPSec*.

Το *PrinterFile*, σύμφωνα και την εικόνα 62, έχει τα εξής *features*:

- *CPP*,
- *Windows,*
- *Unix,*
- *JavaBased,*
- *Proxy,*
- *TCP_IP,*
- *UPnP_Protocol,*

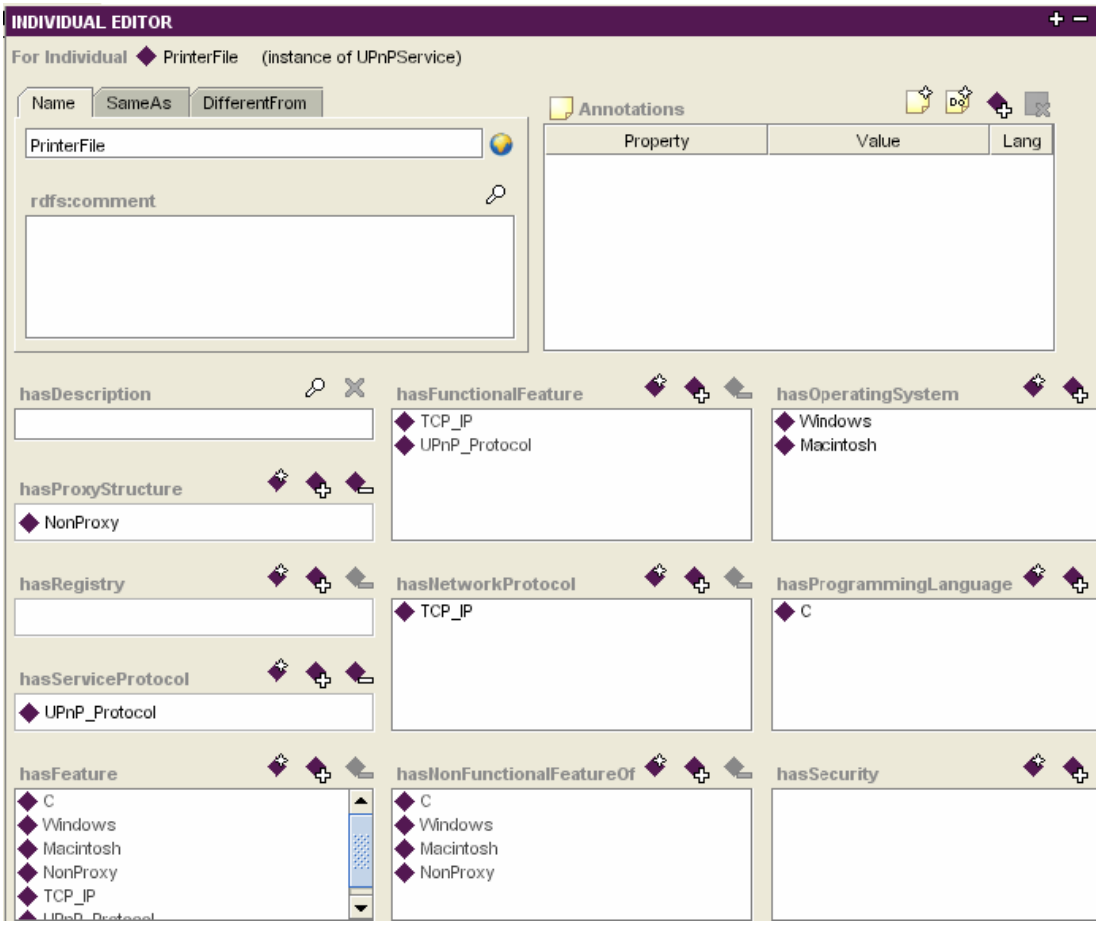
από τα οποία τα «λειτουργικά» χαρακτηριστικά είναι τα:

- *TCP_IP,*
- *UPnP_Protocol,*

και τα «μη-λειτουργικά» χαρακτηριστικά:

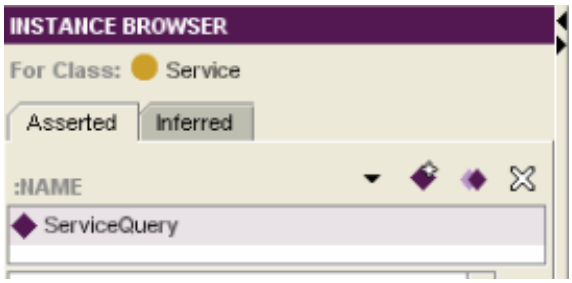
- *CPP,*
- *Windows,*
- *Unix,*
- *Proxy.*

Τέλος, το *PrinterFile* είναι *proxy-based* (καθότι η σχέση *hasProxyStructure* έχει ως *Range* το στιγμιότυπο *Proxy*), διαθέτει πρωτόκολλο εύρεσης υπηρεσιών το *UPnP_Protocol* και πρωτόκολλο δικτύου το *TCP_IP*, είναι υλοποιημένο σε *C++* και χρησιμοποιεί *Windows* ή *Unix* ως λειτουργικό σύστημα. Το συγκεκριμένο στιγμιότυπο έχει ως πεδίο τιμών της σχέσης *hasSecurity,* την τιμή *JavaBased*.



Εικόνα 62: Οι συσχετίσεις του στιγμιότυπου *PrinterFile*

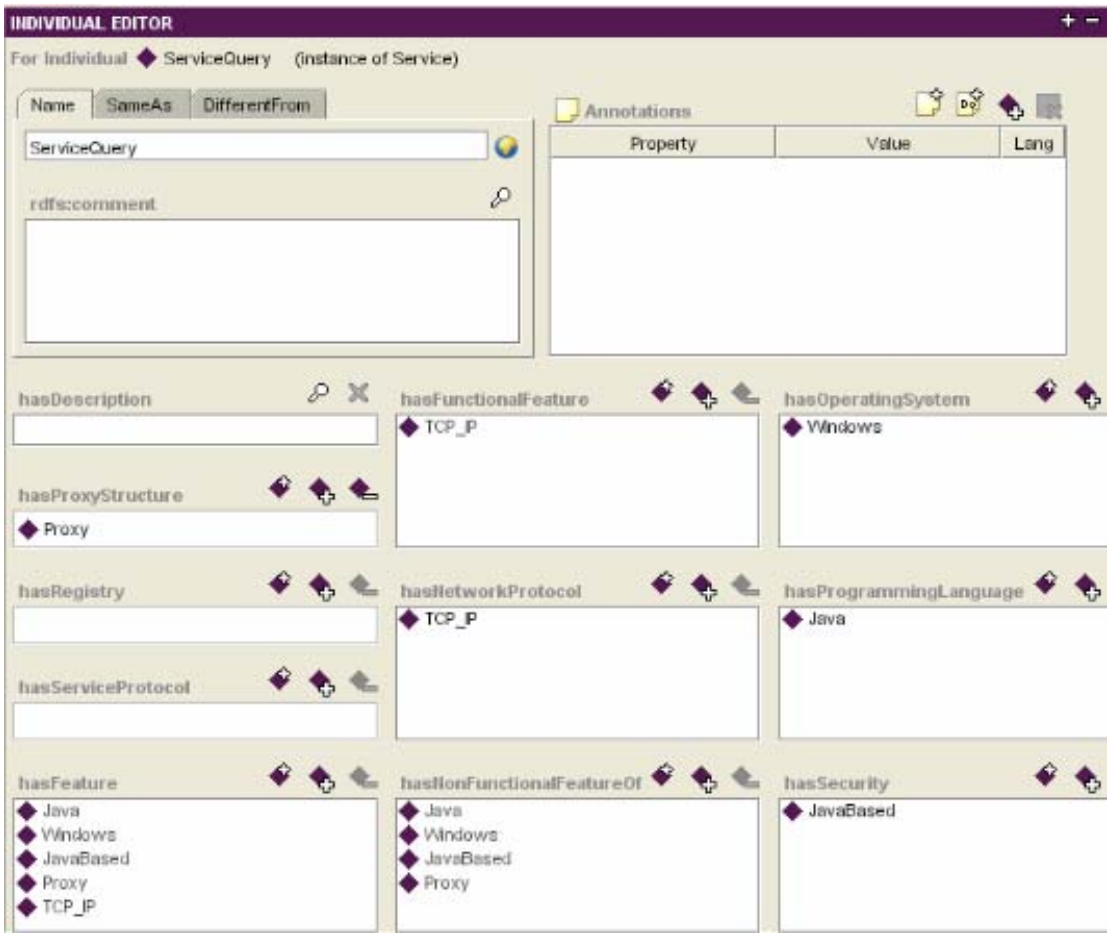
Τέλος, στην οντολογία υπάρχει και το στιγμιότυπο που αφορά την ερώτηση για την υποτιθέμενη υπηρεσία (*ServiceQuery*), που ζητείται από κάποιον χρήστη (εικόνα 63).



Εικόνα 63: Το στιγμιότυπο της ζητούμενης υπηρεσίας ServiceQuery

Το *ServiceQuery*, σύμφωνα και την εικόνα 64, έχει τα εξής *features*:

- *Java*,
- *Windows,*
- *JavaBased,*
- *Proxy,*
- *TCP_IP.*



Εικόνα 64: Οι συσχετίσεις του στιγμιότυπου *ServiceQuery*

από τα οποία το «λειτουργικό» χαρακτηριστικό είναι το:

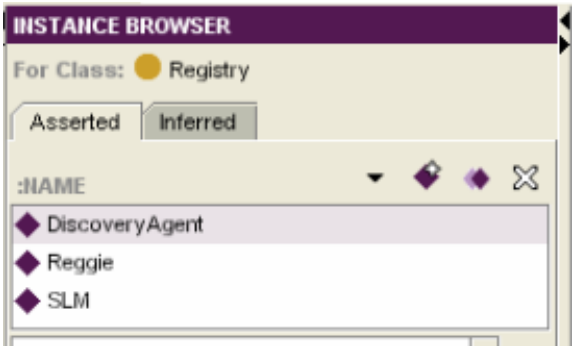
- *TCP_IP,*

και τα «μη-λειτουργικά» χαρακτηριστικά:

- *Java,*
- *Windows,,*
- *Proxy.*

Ακόμα, το *ServiceQuery* είναι *proxy-based* (καθότι η σχέση *hasProxyStructure* έχει ως *Range* το στιγμιότυπο *Proxy*), πρωτόκολλο δικτύου το *TCP_IP*, είναι υλοποιημένο σε *Java* και χρησιμοποιεί *Windows* ως λειτουργικό σύστημα. Το συγκεκριμένο στιγμιότυπο έχει ως πεδίο τιμών της σχέσης *hasSecurity,* την τιμή *JavaBased*.

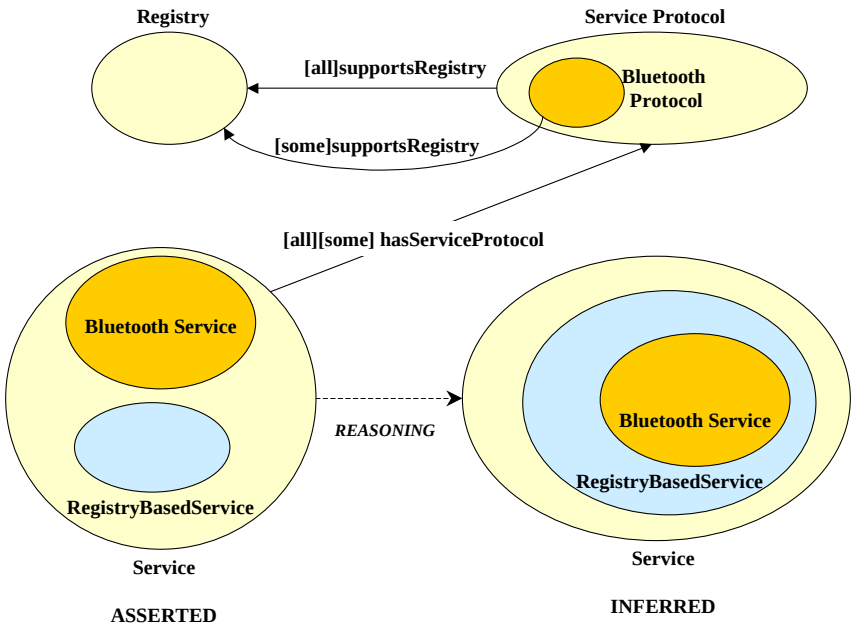
Τέλος, στην υπο-κλάση Registry υπάρχουν τρία στιγμιότυπα: *DiscoveryAgent*, *Reggie* και *SLM* (εικόνα 65).



Εικόνα 65: Τα στιγμιότυπα *DiscoveryAgent*, *Reggie*, *SLM*

8.4 Περιγραφή της Ιεραρχίας μετά από Συμπερασμό Πρώτης Τάξης

Σύμφωνα με την εικόνα 66, η κλάση *ServiceProtocol* περιορίζεται από μια καθολική σχέση ∀*supportsRegistry*. Αυτό σημαίνει ότι οποτεδήποτε ένα πρωτόκολλο υπηρεσίας υποστηρίζει την τεχνολογία *Registry* τότε πρέπει να λαμβάνει τιμές από τα στιγμιότυπα αυτής της *κλάσης* (*Registry*). Ο καθολικός περιορισμός (∀) δεν αναφέρεται στην τουλάχιστον μία τιμή αλλά στο γεγονός ότι όταν ένα πρωτόκολλο έχει κάποιο *Registry* πρέπει να είναι καθορισμένου τύπου.



Εικόνα 66: Συμπερασμός και εμπλουτισμός οντολογίας με σημασιολογία μετά τη χρήση μηχανισμού συμπερασμού

Με την χρήση του τελεστή συμπερασμού θα ταξινομήσουμε την υπηρεσία *Jini* ως μια (*RegistryBasedServiceProtocol*) υπηρεσία που χρησιμοποιεί την τεχνολογία *Registry* επειδή το πρωτόκολλο που χρησιμοποιεί είναι αυτής της τεχνολογίας. Στην

πραγματικότητα το πρωτόκολλό της ταξινομείται ως *RegistryBasedServiceProtocol* και λόγο του υπαρξιακού ποσοδείκτη στην σχέση ∃ *hasServiceProtocol* η ταξινόμηση είναι *RegistryBasedServiceProtocol*. Αυτή η ταξινόμηση επιτυγχάνεται λόγω του Open World Assumption (OWA) [27] που υποστηρίζεται από την DL. Αναφορικά η τεχνική OWA φέρνει βάσεις γνώσης στο συμπερασμό προτάσεων μόνον εφόσον αυτές οι προτάσεις ικανοποιούν όλες τις ικανές και αναγκαίες συνθήκες για να γίνουν αποδεκτές από τις εκάστοτε βάσεις γνώσης. Με βάση την OWA κάτι που δεν αποδεικνύεται *αληθές* δεν υπονοείται ότι είναι *ψευδές*. Σε αντίθεση λειτουργεί ο σχεσιακός λογισμός ο οποίος υποστηρίζει τη τεχνική Close World Assumption (CWA). Με βάση την CWA οτιδήποτε δεν αποδεικνύεται ως *αληθές* τότε υπονοείται ότι είναι *ψευδές*.

Η βάση γνώσης KB υποστηρίζει τις κάτωθι αληθείς προτάσεις:

H |= ServiceProtocol = FunctionalFeature Π ∀**supportsRegistry.Registry**

H |=Jini ⊆ **ServiceProtocol** Π ∃**supportsRegistry.Registry** (εδώ πραγματοποιείται το closure axiom)

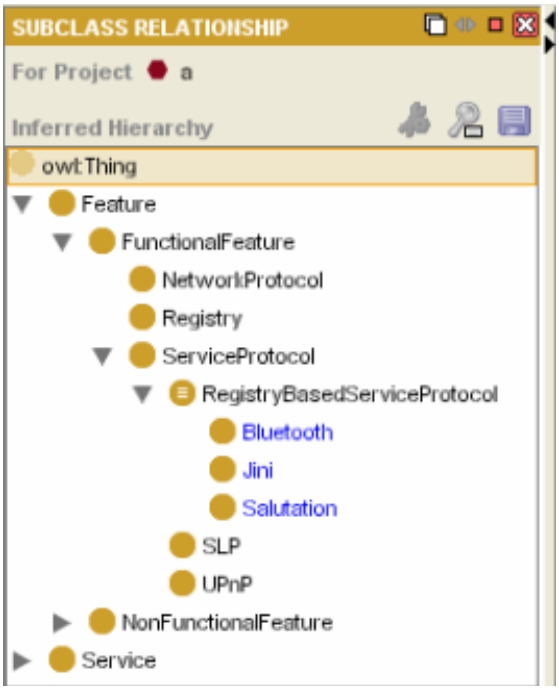
H |= RegistryBasedServiceProtocol ≡

ServiceProtocol Π ∃**supportsRegistry.Registry**

Συμπεραίνουμε ότι (Εικόνα 67):

H |= Jini ⊆ **RegistryBasedServiceProtocol**,

που σημαίνει ότι το πρωτόκολλο *Jini* ταξινομείται κατάλληλα εφόσον η τεχνική OWA είναι σε θέση να αποφανθεί ότι η περιγραφή του *Jini* παρουσιάζεται ως ικανή και αναγκαία συνθήκη για να θεωρηθεί ως πρωτόκολλο που υποστηρίζει την τεχνική *Registry*. Το ίδιο συμβαίνει με τα πρωτόκολλα *Bluetooth* και *Salutation*.



Εικόνα 67: Τμήμα της ιεραρχίας κλάσεων μετά τη χρήση μηχανισμού συμπερασμού

Έστω επίσης ότι η KB υποστηρίζει τα εξής:

H |= RegistryBasedService ≡ Service Π

∃hasServiceProtocol.(ServiceProtocol Π**.(∃supportsRegistry.Registry))**

H |= ProxyJiniService ⊆ JiniService Π **∃hasProxyStructure.ProxyStructure**

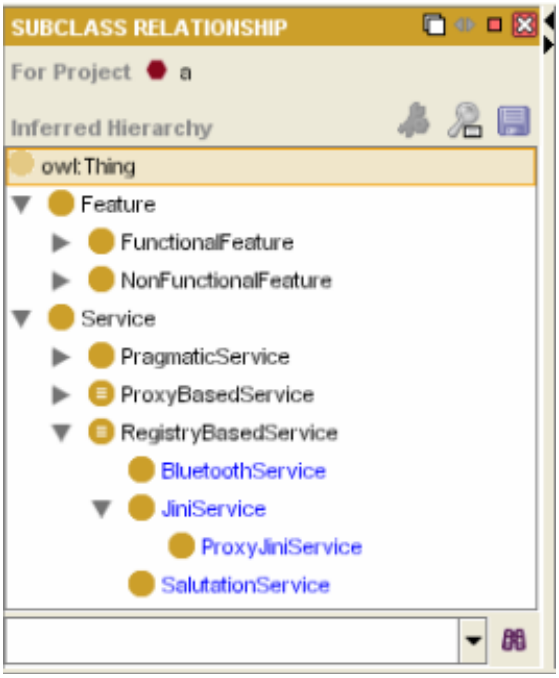
H |= JiniService Π **∃hasServiceProtocol.Jini**

Όπως αποδεικνύεται και από την εικόνα 68, συμπεραίνουμε ότι:

H |= JiniService ⊆ RegistryBasedServiceProtocol

H |= ProxyJiniService ⊆ ProxyBasedService

H |= ProxyJiniService ⊆ RegistryBasedServiceProtocol.



Εικόνα 68: Τμήμα της ιεραρχίας κλάσεων μετά τη χρήση μηχανισμού συμπερασμού

Τέλος, στις εικόνες 69, 70 απεικονίζεται η οντολογία με την αρχική μορφή καθώς και με τη μορφή που παίρνει αφότου χρησιμοποιηθεί ο μηχανισμός συμπερασμού. Αξίζει να παρατηρηθούν οι διαφορές που εντοπίζονται, όσον αφορά τις κλάσεις *Jini*, *Bluetooth* και *Salutation* καθώς και *BluetoothService*.

Σημειώνεται ότι:

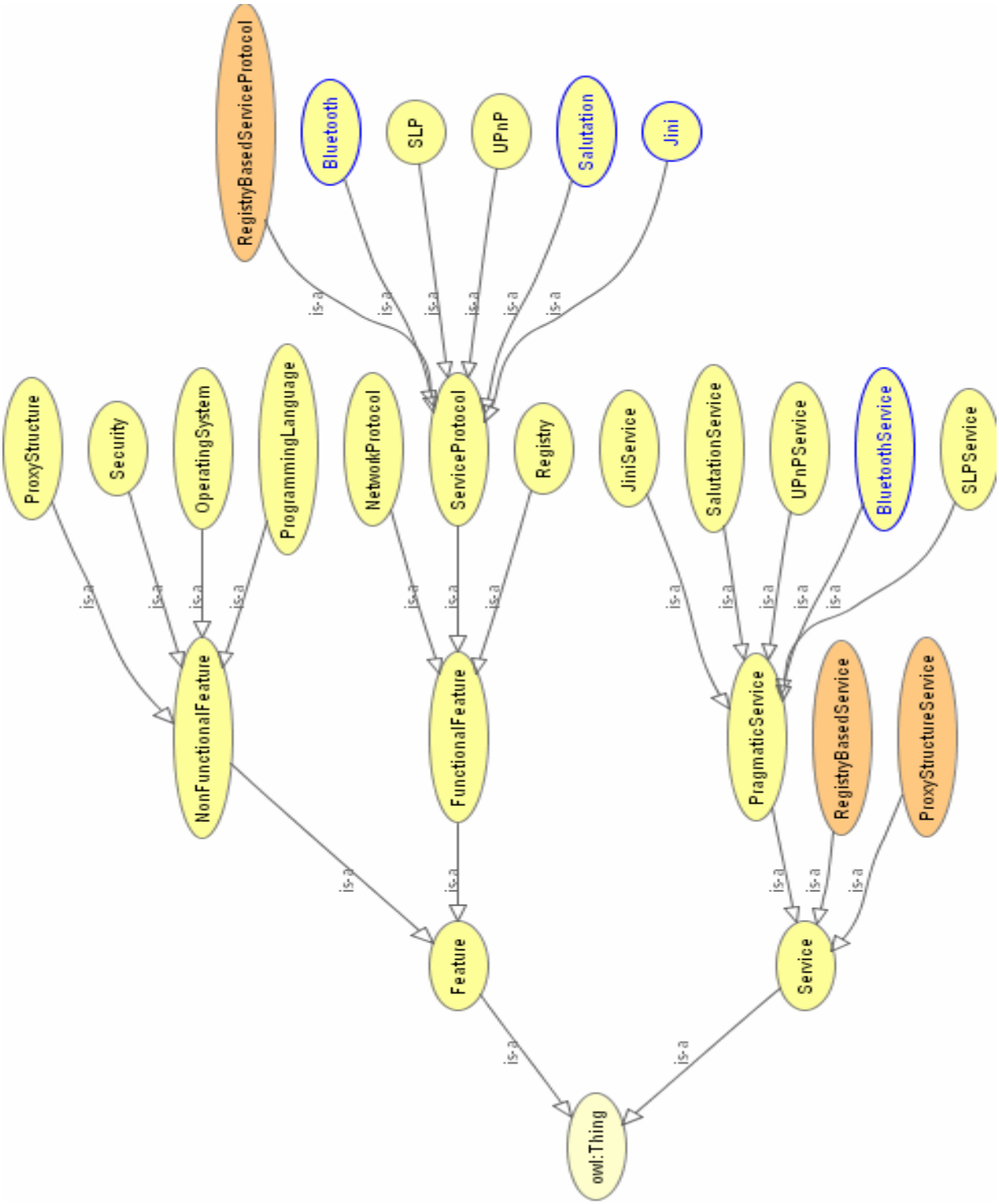
H |= RegistryBasedService ≡

ServiceΠ∃hasServiceProtocol.(ServiceProtocolΠ.(∃supportsRegistry.Registry))

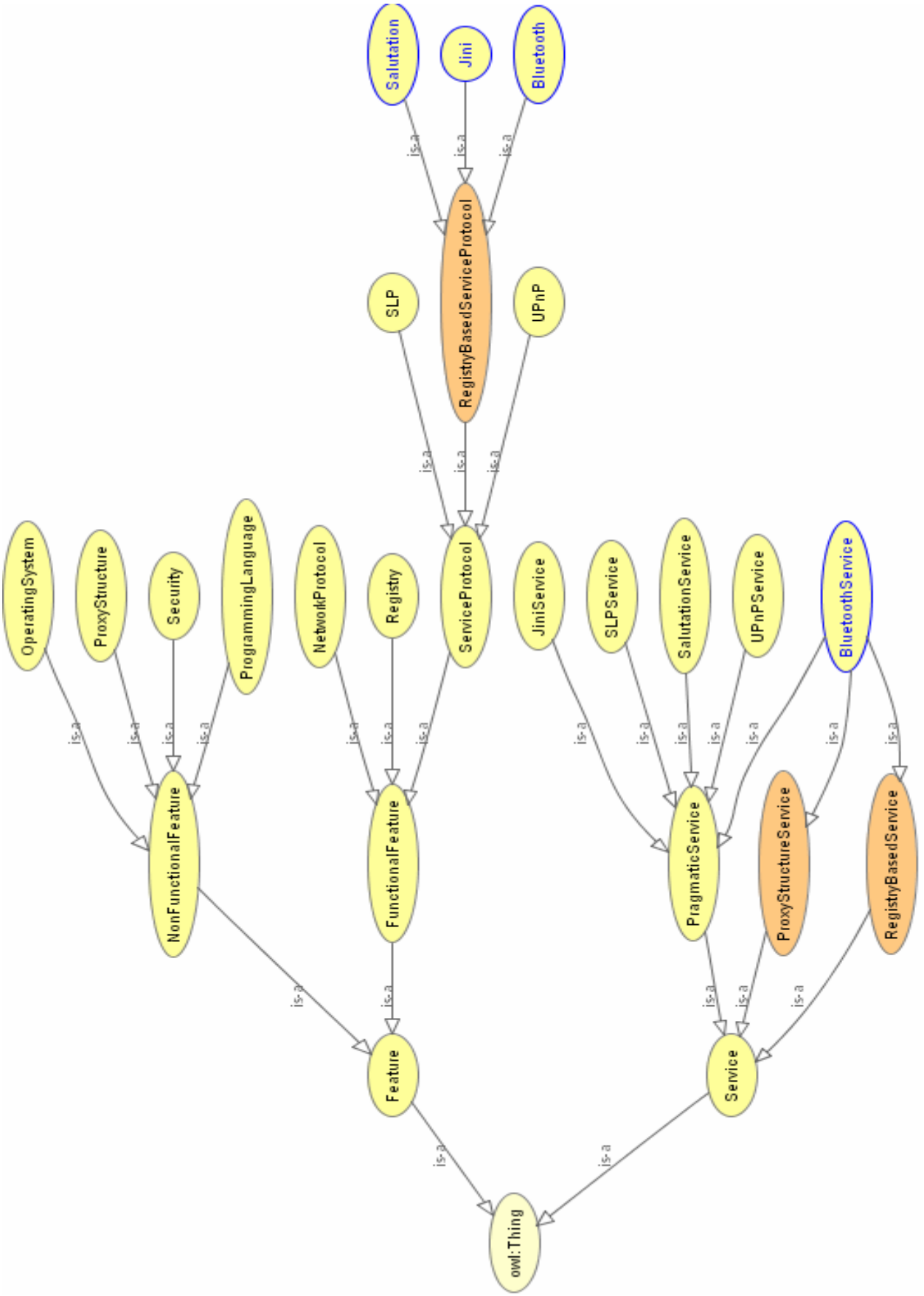
↔

H|=RegistryBasedService≡

ServiceΠ∃hasServiceProtocol.○(∃supportsRegistry.Registry))



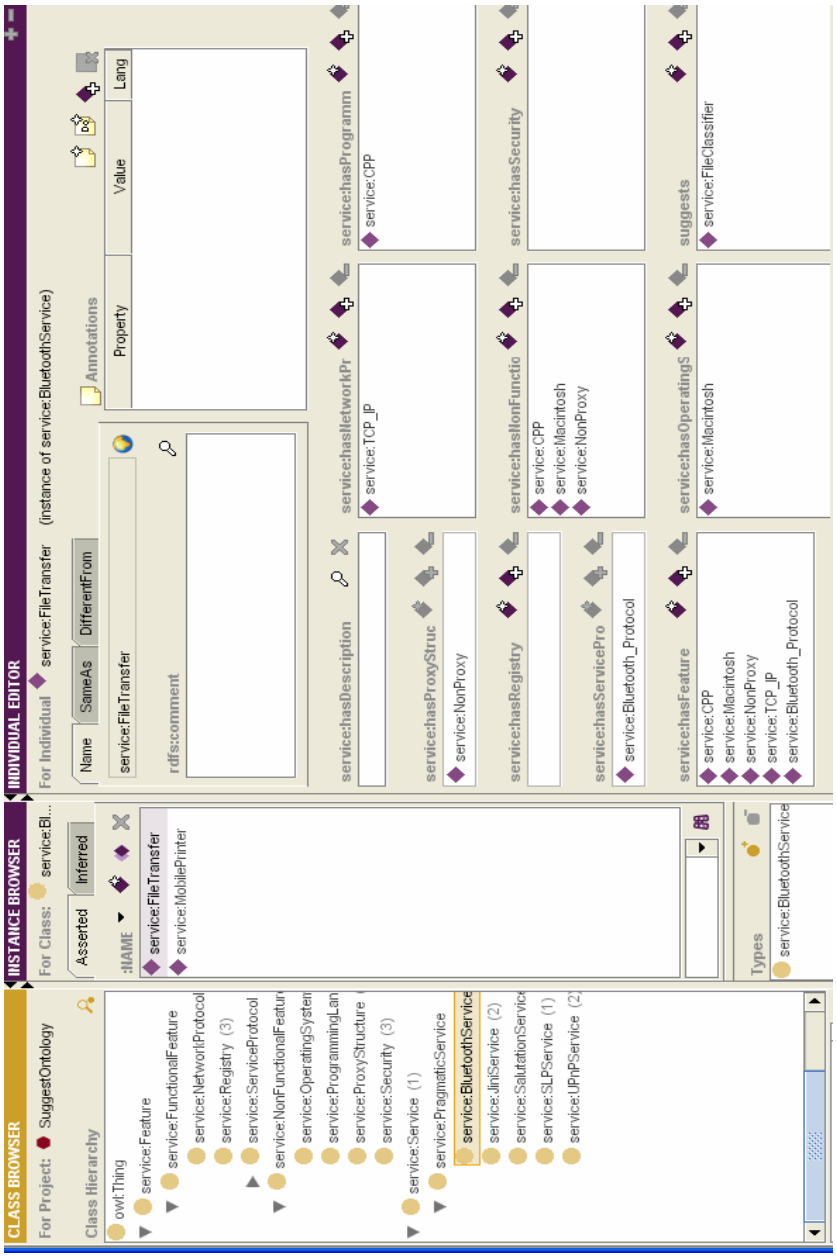
Εικόνα 69: Η οντολογία των υπηρεσιών με την αρχική της μορφή



Εικόνα 70: Η οντολογία των υπηρεσιών μετά τη χρήση μηχανισμού συμπερασμού

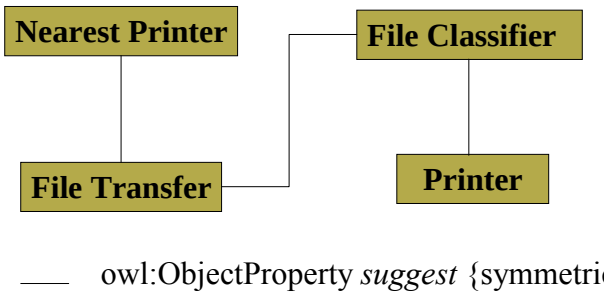
8.5 Περιγραφή Οντολογίας Προτεινόμενων Υπηρεσιών

Σαν επέκταση της διαδικασίας εύρεσης της πλησιέστερης υπηρεσίας που περιγράφηκε προηγουμένως, σ' αυτή την παράγραφο προτείνεται η δυνατότητα παροχής στο χρήστη μερικών συνεργάζομενων υπηρεσιών με αυτή που ζήτησε ή με αυτή που επιλέχθηκε ως η σημασιολογικά και λεξικογραφικά *κοντινότερη*. Γι' αυτό το σκοπό χρησιμοποιείται η οντολογία προτεινόμενων υπηρεσιών, που περιέχει τη βασική συσχέτιση *suggests*, η οποία συνδέει δύο υπηρεσίες (εικόνα 71). Ουσιαστικά, η ανάγκη για την εν λόγω επέκταση πηγάζει από το γεγονός ότι ο χρήστης που αναζητά μια συγκεκριμένη υπηρεσία, κάποιες φορές επιθυμεί ένα σύνολο από άλλες υπηρεσίες σχετιζόμενες με τη ζητηθείσα.



Εικόνα 71: Η οντολογία των προτεινόμενων υπηρεσιών

Στη συνέχεια παρουσιάζεται ένα παράδειγμα του μηχανισμού με τον οποίο προτείνονται στο χρήστη συνεργαζόμενες υπηρεσίες. Όπως φαίνεται στην εικόνα 72, σύμφωνα με τα μετα-δεδομένα της οντολογίας η υπηρεσία *Nearest Printer*, που εντοπίζει τον κοντινότερο εκτυπωτή προτείνει την υπηρεσία *File Transfer*, δηλαδή μια υπηρεσία μεταφοράς αρχείων. Η *File Transfer* με τη σειρά της συνιστά τη χρήση της *File Classifier*. Η ουσία έγκειται στο γεγονός ότι για να τυπωθεί ένα αρχείο ίσως απαιτείται η μεταφορά του καθώς και η γνώση για τον τύπο του, έτσι ώστε να σταλεί στον κατάλληλο εκτυπωτή. Τέλος, μια άλλη υπηρεσία, η *Printer*, προτείνει επίσης την *File Classifier*, μια που, όπως και προηγουμένως, χρήσιμη είναι η ταξινόμηση του αρχείου προκειμένου να επιλεγεί ο σωστός εκτυπωτής που αφορά το συγκεκριμένο τύπο αρχείων.



Εικόνα 72: Παράδειγμα επιλογής προτεινόμενων υπηρεσιών

9 ΑΛΓΟΡΙΘΜΟΣ ΕΥΡΕΣΗΣ ΣΥΓΓΕΝΙΚΩΝ ΕΝΝΟΙΩΝ

9.1 Περιγραφή

Οι έννοιες ερμηνεύονται σε στοιχεία συνόλου, σύμφωνα με την περιγραφική λογική, που αναφέρεται στο Κεφάλαιο 7. Ανάμεσα σε στοιχεία συνόλου υπάρχουν ή ενδέχεται να υπάρχουν αντικατοπτριζόμενες έννοιες που η ερμηνεία τους σχετίζεται τόσο σημασιολογικά όσο και λεξικογραφικά. Η συσχέτιση αυτή, δεν είναι ποτέ καθορισμένη ή καλά ορισμένη με έναν μαθηματικό λογισμό (Θεωρία Συνόλων) ή άλγεβρα (*Hedge Algebra*) [28]. Στην παρούσα εργασία εικάζεται μια μέθοδος υπολογισμού σημασιολογικής και κατ' επέκταση εννοιολογικής σχέσης μεταξύ στοιχείων συνόλου, που προδιαγράφονται με μια γλώσσα υποσύνολου της λογικής πρώτης τάξης ($\mathcal{ALC}$). Στο επόμενο σχήμα απεικονίζεται η εννοιολογική και σημασιολογική συσχέτιση εννοιών και στοιχείων συνόλων. Υποθέτουμε ότι η σημασιολογική μετρική συγγένειας δύο στοιχείων συνόλου, είναι ανάλογη (ισομορφική) με την εννοιολογική μετρική συγγένειας εννοιών. Δηλαδή τα στοιχεία α και β έχουν ίση μετρική συγγένειας με τη μετρική συγγένειας των εννοιών I(α) και I(β), όπου I είναι η αντίστροφη συνάρτηση ερμηνείας (interpretation). Στην βιβλιογραφία τα στοιχεία α και β, καλούνται ως μοντέλο ερμηνείας.
Δεδομένου του αλγορίθμου θεωρείται το σύνολο:

- Οντολογία περιγραφής *υπηρεσιών*, *O*
- Οντολογία προτεινόμενων *υπηρεσιών*, *S*
- Τρέχουσα πληροφορία *πλαισίου* του χρήστη, *C,* π.χ (λειτουργικό σύστημα του τερματικού, πρωτόκολλο δικτύου, προτιμήσεις)
- Λεξικογραφική ερώτηση *Q* εύρεσης *υπηρεσίας* με δεδομένα το *επιστημολογικό* επίπεδο του χρήστη (*Άπειρος*, *Έμπειρος*, *Εξειδικευμένος*), όσον αφορά την κατάλληλη γνώση πληροφορίας πλαισίου του. Ορίζεται, επίσης, το διάνυσμα doc_Q των *λέξεων* ή *όρων* που περιγράφουν την ζητούμενη υπηρεσία.

Αποτελέσματα του αλγορίθμου είναι μια λίστα *Σ* των συγγενικών υπηρεσιών που αντιστοιχούν στην τρέχουσα πληροφορία πλαισίου, καθώς και ένα σύνολο *Π* από προτεινόμενες ή λειτουργικά συνεργαζόμενες με την ζητηθείσα υπηρεσία.
Η παρακάτω λίστα ορίζει τις απαραίτητες συναρτήσεις που λαμβάνουν χώρα στον προτεινόμενο αλγόριθμο.

Συνάρτηση/ Τύπος	Επεξήγηση
$fts \equiv sim_T : \Xi \times 2^{\{docQ\}} \rightarrow [0,1]$	Τιμή Μονάδας Συγγένειας Λεξικογραφικής Ομοιότητας. Επιτυγχάνει την απεικόνιση ενός εγγράφου με όρους από το σύνολο {*doc*Q} για καθορισμένη λεξικογραφική κλάση $\xi \in \Xi$ στο μοναδιαίο διάστημα.
textcat: $2^O \rightarrow 2^{\Xi}$	Συνάρτηση κατάταξης του *πεδίου ορισμού* της DataType σχέσης *hasDescription* (Κεφ.8), ορισμένη στην οντολογία O, σύμφωνα με τις λεξικογραφικές κατηγορίες Ξ.
$sim_C(\sigma,\lambda) = \alpha\, RS(\sigma,\lambda) + (1-\alpha)TS(\sigma,\lambda)$, $\alpha \in (0,1]$, $\sigma,\lambda \in A$	Σημασιολογική ομοιότητα μιας οντότητας σ με μια οντότητα λ ως γραμμικός συνδυασμός των *RS* και *TS* με ισοζυγισμένο στην μονάδα συντελεστή ταξινομίας α.
$\vec{r} = <sim_C(r,q), sim_C(r,t_1),\ldots,sim_C(r,t_\kappa)>$, $t_i \in A$, $i=1..\kappa=\lvert A\rvert$	Το διάνυσμα ομοιότητας μιας οντότητας r με τις οντότητες t_i του συνόλου *A*.
$sim : 2^{\{r\}} \times A \rightarrow [0,1]$	Ομοιότητα μιας οντότητας r με μια οντότητα $\varepsilon \in A$: $r \in O$, $A \subseteq O$. Το r είναι στοιχείο υπερσυνόλου του A. Η συνάρτηση αυτή υπολογίζει την ομοιότητα του r, με οποιαδήποτε μετρική, με τα στοιχεία ενός υποσυνόλου A και απεικονίζεται στο διάστημα [0,1].

Πίνακας 6: Βασικές συναρτήσεις του αλγορίθμου

Ο προτεινόμενος αλγόριθμος αποτελείται από τα εξής βήματα:

- Φιλτράρισμα των λεξικογραφικών κλάσεων ξ_i , i=1..κ, της *O*, $|O|=m<\kappa$, που είναι λεξικογραφικά όμοιες με την υπηρεσία της *Q*. Η λεξικογραφική ομοιότητα επιτυγχάνεται με την εφαρμογή της Θεωρίας Ασάφειας, όπως αναφέρθηκε στο κεφάλαιο 4. Έπειτα παράγεται το ταξινομημένο σύνολο των λεξικογραφικά φιλτραρισμένων κλάσεων υπηρεσιών $O^Q_{lex} = \{<\xi_i, \mathrm{value}_{\mathrm{sorted}}> \mid fts(\xi_i, doc_Q)\}$, όπου $\mathrm{value}_{\mathrm{sorted}}$ είναι η τιμή της *fts* που υπολογίστηκε για την κλάση ξ_i με το διάνυσμα doc_Q. Αποτέλεσμα του

βήματος είναι το max$arg_{\xi m}(O^{Q}_{lex})$, δηλαδή η κλάση ξ_{MAX} που έχει την μέγιστη ομοιότητα με το διάνυσμα όρων doc_Q.

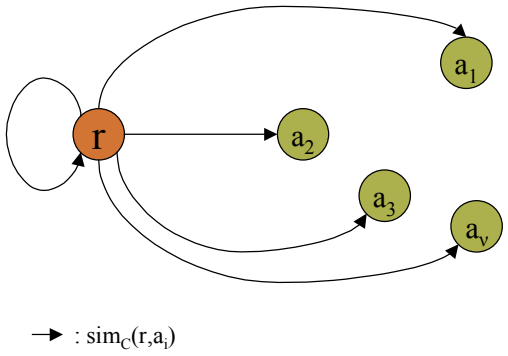
Λαμβάνουμε το σύνολο $A \subseteq O$ εκείνων των υπηρεσιών που είναι μέλη της κλάσης ξ_{MAX}. Σημειώνεται, ότι μια υπηρεσία μπορεί να είναι μέλος πολλών λεξικογραφικών κλάσεων ξ_i. Αυτή η πολλαπλή εμπλοκή (μια υπηρεσία να είναι πολλαπλό μέλος) επιφέρει την ασάφεια διαχωρισμού αυτής της υπηρεσίας ώστε να θεωρηθεί συγγενική με την υπηρεσία της ερώτησης.

- Η εισαγωγή της σημασιολογικής ταξινόμησης των υπηρεσιών ως συγγενικές με την υπηρεσία της *Q*, θεωρείται απαραίτητη, διότι λαμβάνει υπόψιν την πληροφορία πλαισίου, και όχι μόνο την, μερικές φορές εσφαλμένη ή κακώς περιγραφείσα λόγω του επιστομολογικού επιπέδου του χρήστη, λεξικογραφική περιγραφή. Από την άλλη πλευρά, πρέπει να σημειωθεί η αναγκαιότητα και των δύο φίλτρων. Η ομοιότητα που αφορά μόνον την πληροφορία πλαισίου δεν είναι ικανή και αναγκαία για την καταλληλότερη ταξινόμηση των συγγενικών υπηρεσιών. Στο στάδιο αυτό υπολογίζεται τόσο η ταξινομική ομοιότητα όσο και η ομοιότητα σχέσεων (Κεφ. 3) δύο *οντοτήτων*. Εφαρμόζουμε τις συναρτήσεις *RS* και *TS* όχι μεταξύ στοιχείων συνόλου, αλλά μεταξύ συνόλων. Ως προς τούτο, δημιουργούνται δύο διανύσματα που αντικατοπτρίζουν τα δύο σύνολα προς ομοιότητα. Τα δύο σύνολα είναι:
  i. Το σύνολο των υπηρεσιών που ανήκουν μόνο στην λεξικογραφική κλάση ξ_{MAX}, και,
  ii. το μονοσύνολο {*q*} που περιέχει την υπηρεσία της *Q*.

  Έστω το διάνυσμα της ζητηθείσας υπηρεσίας r, $\vec{r}$, και το διάνυσμα μιας υπηρεσίας g $\in A$, $\vec{g}$. Το $\vec{r}$ ορίζεται ως:

$$\vec{r} = <sim_C(q,q), sim_C(q,t_1),..., sim_C(q,t_k)>$$

  Σημειώνεται ότι $d(\vec{r}) = d(\vec{g}) = \kappa+1$, $\kappa \in \aleph$. Στο παρακάτω σχήμα φαίνεται η δημιουργία του διανύσματος ομοιότητας μιας υπηρεσίας r σε σχέση με ένα σύνολο υπηρεσιών $a_i \in A$.



Εικόνα 73: Διαδικασία δημιουργίας διανύσματος ομοιότητας

Η συνάρτηση sim_C δηλώνει την πλαισιακή ομοιότητα δύο στιγμιοτύπων σ, λ της οντολογίας O. Η σημασιολογική ομοιότητα $\overrightarrow{sim_C}$ των δύο διανυσμάτων $\vec{r}$ και $\vec{g}$ ορίζεται σύμφωνα με την κλασική ομοιότητα διανυσμάτων ως το συνημίτονο των διανυσμάτων αυτών.

$$sim^{\delta}{}_C(\vec{r},\vec{g}) = sim^{\delta}_C = \frac{\sum_{\rho\in r}\vec{r}.\sum_{\gamma\in g}\vec{g}}{\left|\sum_{\rho\in r}\vec{r}\right|.\left|\sum_{\gamma\in g}\vec{g}\right|}\text{. Εξ (10)}$$

Τέλος, η λεξικογραφική και σημασιολογική ομοιότητα της ζητηθείσας υπηρεσίας με τις υπηρεσίες a ∈ *A* εκφράζεται ως *sim*({r},a), όπου r είναι το υποτιθέμενο στυγμιότυπο της ζητηθείσας υπηρεσίας το οποίο εισάγεται *αυθαίρετα* στην οντολογία *O*. Η r εισάγεται ως στιγμιότυπο υπερ-κλάσης (Service) σε σχέση με τα υπάρχοντα στιγμιότυπα της *A*. Έτσι, ο αλγόριθμος *προσπαθεί* να ταξινομήσει την r στην *κατάλληλη* κλάση υπηρεσίας (JiniService, BluetoothService,…) με βάση την τιμή της ομοιότητάς της. Η συνάρτηση αυτή είναι το ισοζυγισμένο άθροισμα της εφαρμογής της *sigmod* συνάρτησης πάνω στις ομοιότητες $sim^{\delta}{}_C$ και sim_T.

$$sim(\{r\},\alpha)=\beta.\text{sigmod}(\overrightarrow{sim_C}\ (\vec{r},\vec{a})\text{-}0.5)+$$
$$(1\text{-}\beta).\text{sigmod}(sim_T(textcat(O),doc_Q(r))\text{-}0.5),$$

Συντελεστής λεξικογραφικής ομοιότητας β ∈ [0,1], Εξ.(11)

- Για την αποδοχή των πιο όμοιων υπηρεσιών του συνόλου *A* με την ζητηθείσα r υπηρεσία λαμβάνουμε την μέση (στατιστική) τιμή του *sim* για όλα τα a ∈ *A*. Έτσι, η μέση ομοιότητα της r με τις υπηρεσίες του *A*, είναι *μ(sim)*. Χρησιμοποιούμε την *μ(sim)* για να λάβουμε τις *πιο όμοιες*

υπηρεσίες με τη χρήση της Θεωρίας Ασάφειας στο επόμενο στάδιο του αλγορίθμου. Ορίζεται το ασαφές σύνολο *ομοιότητα* που συσχετίζει τις μέσες τιμές ομοιότητας (*sim*) στιγμιοτύπων οντολογίας *O* με την συνάρτηση συγγένειας $\mu_{\mu(SIM)}$. Ορίζονται επίσης τρεις ασαφείς λεξικές μεταβλητές του ασαφούς συνόλου *ομοιότητα* ως: *μικρή-ομοιότητα*, *μέτρια-ομοιότητα* και *μεγάλη-ομοιότητα*. Οι τιμές του *ομοιότητα* είναι οι τιμές της υπολογισμένης ομοιότητας, δηλαδή [0,1]. Όπου Φ είναι ο χρυσός αριθμός του Φειδία και φ = Φ-1=(sqrt(5)-1)/2 = 0.618…Ο αριθμός φ είναι το σημείο χρυσής τομής της απόστασης 0 μέχρι 1 (Η δεύτερη ρίζα της εξίσωσης του Φειδία).

- Η μ_{SIM} της *μικρή-ομοιότητα* ορίζεται ως εξής:

$$\mu_{SIM}(x) = (1/\varphi^2)\, x,\ x \in [0, \varphi^2],$$
$$= 1/(\varphi(\varphi-1))x - 1/(\varphi-1),\ x \in [\varphi^2, \varphi],$$

- Η μ_{SIM} της *μέτρια-ομοιότητα* ορίζεται ως εξής:

$$\mu_{SIM}(x) = (2/\varphi(1-\varphi))\, x - 2\varphi/(1-\varphi),\ x \in [\varphi^2, (\varphi(\varphi+1)/2)],$$
$$= 2/(\varphi(\varphi-1))x - 2/(1-\varphi),\ x \in [(\varphi(\varphi+1)/2), \varphi],$$

- Η μ_{SIM} της *μεγάλη-ομοιότητα* ορίζεται ως εξής:

$$\mu_{SIM}(x) = (2/(2-\varphi^2-\varphi))x + (1-((2/(2-\varphi^2-\varphi)))),\ x \in [(\varphi(\varphi+1)/2), 1],$$

Ορίζεται επίσης το ασαφές σύνολο *απόρριψη* που συσχετίζει το ποσοστό απόρριψης των υπηρεσιών που δεν θεωρούνται όμοιες, με τη συνάρτηση συγγένειας $\mu_{REJECTION}$. Ορίζονται ακόμη τρεις ασαφείς μεταβλητές του ασαφούς συνόλου απόρριψη ως: *μικρή-απόρριψη*, *μέτρια-απόρριψη* και *μεγάλη-απόρριψη*.

- Η $\mu_{REJECTION}$ της *μικρή-απόρριψη* ορίζεται ως εξής:

$$\mu_{REJECTION}(x) = 1,\ x \in [0, 0.25],$$
$$= -4x + 2,\ x \in [0.25, 0.5]$$

- Η $\mu_{REJECTION}$ της *μέτρια-απόρριψη* ορίζεται ως εξής:

$$\mu_{REJECTION}(x) = 4x-1,\ x \in [0.25, 0.5],$$
$$= -4x + 3,\ x \in [0.5, 0.75]$$

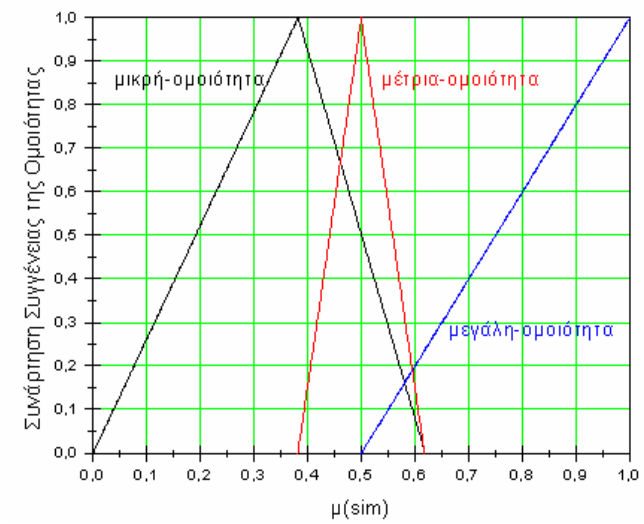
- Η $\mu_{REJECTION}$ της *μεγάλη-απόρριψη* ορίζεται ως εξής:

$$\mu_{REJECTION}(x) = 1,\ x \in [0.75, 1],$$
$$= 4x - 2,\ x \in [0.5, 0.75]$$

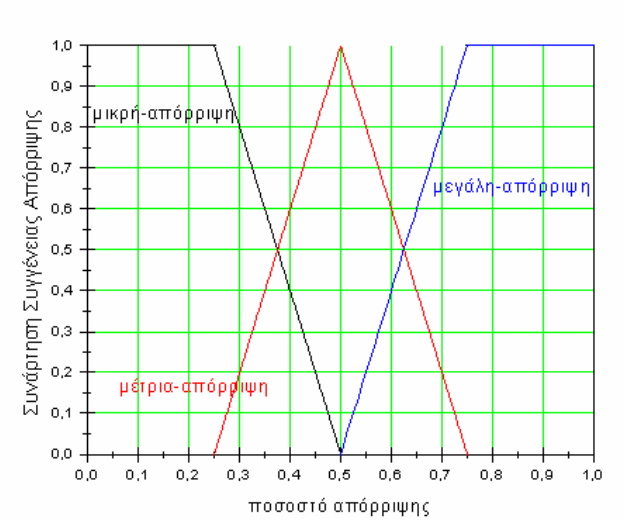
Με αυτά τα *ασαφή* σύνολα έχουμε τον *ασαφή συμπερασμό* των παρακάτω *ασαφών κανόνων* που θα δώσουν την κανονική τιμή *κτ* (crisp) του ποσοστού απόρριψης των υπηρεσιών. Οι κανόνες είναι της μορφής:

- Εάν *μικρή-ομοιότητα* τότε *μεγάλη-απόρριψη*
- Εάν *μέτρια-ομοιότητα* τότε *μέτρια-απόρριψη*
- Εάν *μεγάλη-ομοιότητα* τότε *μικρή-απόρριψη*

Για την αποσαφήνιση της *κτ* χρησιμοποιούμε τη μέθοδο της μέσης τιμής. Η γραφική αναπαράσταση για την αποδοχή ή απόρριψη μιας υπηρεσίας που συγκρίνεται με την ζητηθείσα υπηρεσία απεικονίζεται στο παρακάτω σχήμα.



Εικόνα 74: Η ασαφής μεταβλητή *ομοιότητα*



Εικόνα 75: Η ασαφής μεταβλητή *απόρριψη*

Το στάδιο αυτό του αλγορίθμου δίνει ως αποτέλεσμα το σύνολο *Σ* των περισότερο «συγγενικών» υπηρεσιών, τόσο σημασιολογικά όσο και λεξικογραφικά.

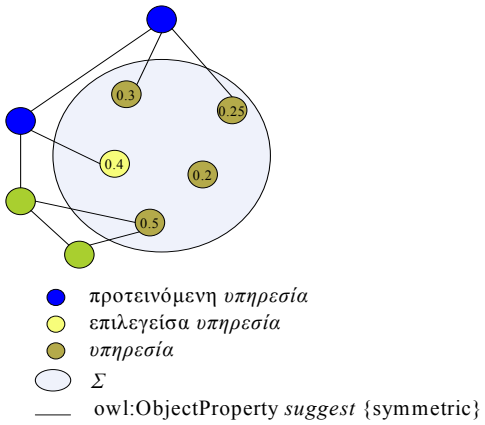
- Το στάδιο αυτό προσπαθεί να συσχετίσει τις $\Sigma \subseteq A$ αποδεκτές υπηρεσίες με την *S* οντολογία. Δηλαδή λαμβάνονται οι |*Σ*|. Εναλλακτικά μπορούν να ληφθούν με ταξινόμηση οι κ-πρώτες υπηρεσίες. Η δυσκολία, όμως, εντοπίζεται στην σωστή επιλογή του κ. Το σύνολο *Σ* (οντολογία των *πιο όμοιων* υπηρεσιών) περιέχει τις σχέσεις των υπηρεσιών που συνιστά η μία την άλλη μέσω της συμμετρικής και όχι μεταβατικής σχέσης *suggest*. Από το σύνολο των υπηρεσιών το σύστημα προτείνει εκείνες τις υπηρεσίες που είναι όχι μόνο *suggested* από την *S* οντολογία αλλά και σύμφωνα με τους παρακάτω κανόνες. Αυτοί οι κανόνες συμπεραίνουν τις προτεινόμενες υπηρεσίες. Όταν ο χρήστης επιλέξει μια υπηρεσία από το σύνολο *Σ*, τότε εφαρμόζονται οι παρακάτω κανόνες που παράγουν το σύνολο κανόνων βάθους-3.

9.2 Κανόνες βάθους-3

Με τον όρο R-σύνολο στιγμιοτύπων βάθους-κ, $\kappa \in \aleph$, ορίζεται εκείνο το σύνολο των στιγμιοτύπων που συσχετίζονται με μια συμμετρική και όχι μεταβατική σχέση R από ένα δοθέν και μόνο στιγμιότυπο, μέσα στον υπογράφο της οντολογίας *O*. Έτσι το σύνολο αυτό αποτελείται από στιγμιότυπα που έχουν έμμεσα συσχετισθεί (Κεφ. 7) μέσω της σχέσης R με μήκος μονοπατιού κ μέσα στον υπογράφο της οντολογίας.

- ***suggests***(X,V)∧***suggests***(V,G)∧***suggests***(G,Y)∧***sortedlow***(Y,X)→ S|=G
- ***suggests***(X,V) → S|=G

Το κατηγόρημα *sortedlow* (X,Y) ερμηνεύει πότε ένα στιγμιότυπο X έχει μικρότερη τιμή ομοιότητας από ότι το στιγμιότυπο Y με το δοθέν στιγμιότυπο r. Η εφαρμογή των κανόνων προϋποθέτει ότι τα στιγμιότυπα του συνόλου είναι ταξινομημένα κατά την τιμή της ομοιότητας με το δοθέν στιγμιότυπο r. Στην εικόνα 76 απεικονίζεται η εφαρμογή των κανόνων βάθους-3 και το σύνολο που παράγει την οντολογία *S*.



Εικόνα 76: Εφαρμογή κανόνων επιλογής προτεινόμενων υπηρεσιών

10 ΠΟΡΙΣΜΑΤΑ ΤΟΥ ΠΡΟΤΕΙΝΟΜΕΝΟΥ ΑΛΓΟΡΙΘΜΟΥ

Σ' αυτό το κεφάλαιο συνοψίζονται τα συμπεράσματα της εργασίας, λαμβάνοντας υπόψιν τα αποτελέσματα του αλγορίθμου που περιγράφηκε στο προηγούμενο κεφάλαιο. Καταρχάς για την καλύτερη εκτίμηση και κατανόηση αυτών των αποτελεσμάτων, θεωρούνται τρία επίπεδα της *ServiceQuery*, που παρουσιάστηκε στο κεφάλαιο 8, ανάλογα με το βαθμό γνώσης του χρήστη και την επαρκή γνώση του πλαισίου πληροφορίας του χρήστη. Η επιλογή του **α** ως παράμετρος στις γραφικές απεικονίσεις της τιμής της ομοιότητας δικαιολογεί την εκδήλωση του ενδιαφέροντος το σύστημα να *ταξινομεί* τις ερωτήσεις του χρήστη μέσα στην οντολογία. Με το όρο ταξινόμηση ερμηνεύεται η ικανότητα το σύστημα να αποφανθεί για το έαν μια ερώτηση ταξινομείται σε μιά κλάση με βαθμό ταξινόμησης όσο η τιμή της ομοιότητας. Στην περίπτωση όμως αυτή, μια ερώτηση μπορεί να ταξινομηθεί σε πολλές κλάσεις. Για τον λόγο αυτό το σύστημα αποκρίνεται με ένα σύνολο όμοιων στυγμιοτύπων από διαφορετικές κλάσεις. Έτσι η ερώτηση τίθεται ως στιγμιότυπο υπερ-κλάσης και το σύστημα προσπαθεί να προσεγγίσει την πιο κοντινή υπο-κλάση.

Συνεπώς, η πρώτη εκδοχή της ερώτησης αφορά έναν **άπειρο** χρήστη, ο οποίος όπως είναι φανερό και από την εικόνα 77, υποβάλλει την ερώτηση για την επιθυμητή υπηρεσία, παρέχοντας *μηδαμινή* πληροφορία σχετικά με τα διαθέσιμα τεχνικά χαρακτηριστικά. Σ' αυτήν την περίπτωση ο αλγόριθμος υπολογίζει μόνο τη λεξικογραφική ομοιότητα και το αποτέλεσμα είναι ότι η συνολική ομοιότητα δεν ξεπερνά ένα συγκεκριμένο όριο (65.14%). Επομένως, το συμπέρασμα για το παράδειγμα του αδαούς χρήστη είναι ότι με δεδομένη τη λεξική περιγραφή της υπηρεσίας, ο αλγόριθμος επιστρέφει ένα σύνολο από υπηρεσίες, οι οποίες «μοιάζουν» με τη ζητηθείσα το πολύ κατά 65.14%. Καθώς ο συντελεστής ταξινομίας α αυξάνει προς την μέγιστή του τιμή, που σημαίνει ότι το σύστημα προσπαθεί να ταξινομήσει την ερώτηση του άπειρου χρήστη, τόσο η μετρική της ομοιότητας τείνει προς την μέγιστή της τιμή (εικόνα 81).

Η δεύτερη εκδοχή της ερώτησης αφορά κάποιον **έμπειρο** χρήστη, ο οποίος εκτός από τη λεξική περιγραφή της υπηρεσίας εισάγει και ορισμένα από τα τεχνικά χαρακτηριστικά που θα προτιμούσε, δηλαδή η πληροφορία πλαισίου του χρήστη (εικόνα 78).

Σύμφωνα με το διάγραμμα της εικόνας 80, αυξάνοντας το α και μειώνοντας το β η καμπύλη ομοιότητας μετακινείται σε μεγαλύτερες τιμές, που συνεπάγεται ότι επιτυγχάνεται η μέγιστη σημασιολογική ταξινόμηση. Το πλεονέκτημα έγκειται στο γεγονός ότι μια σημασιολογικά «φτωχή» οντολογία (π.χ. χωρίς πολλούς υπαρξιακές και καθολικές συσχετίσεις, μικρές ταξινομίες) σε συνδυασμό με μια πλούσια πληροφορία

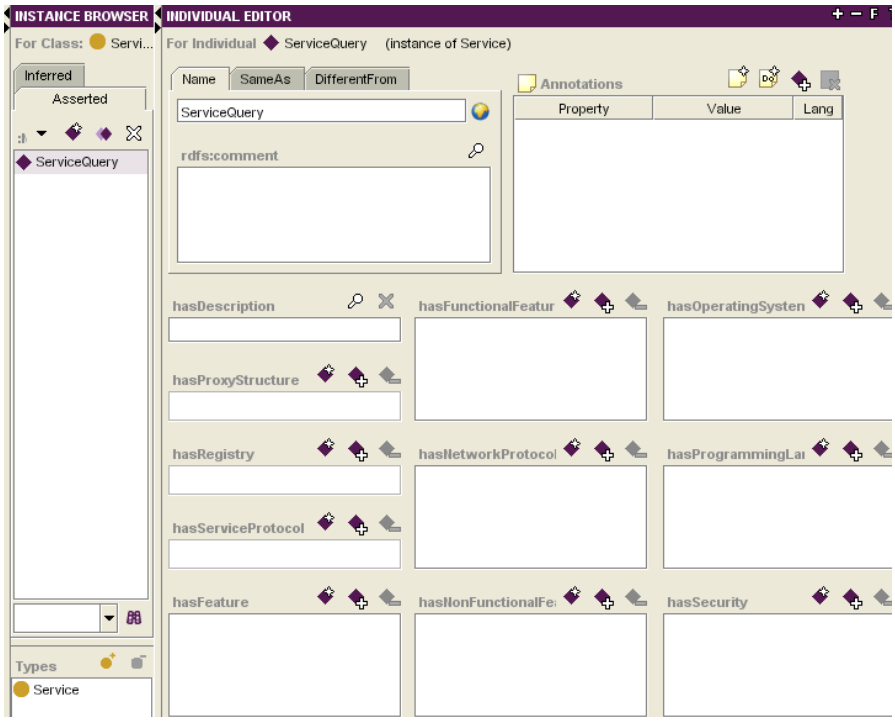
πλαισίου οδηγεί σε «συγκρατημένα» αποτελέσματα τιμής ομοιότητας. Η παρατήρηση αυτή διαπιστώνεται εντονότερα στην τρίτη εκδοχή.

Η τρίτη εκδοχή αφορά έναν **εξειδικευμένο** χρήστη, που όπως αποδεικνύεται και από την εικόνα 79, περιγράφει πλήρως την υπηρεσία που επιθυμεί, όσον αφορά τα τεχνικά χαρακτηριστικά.

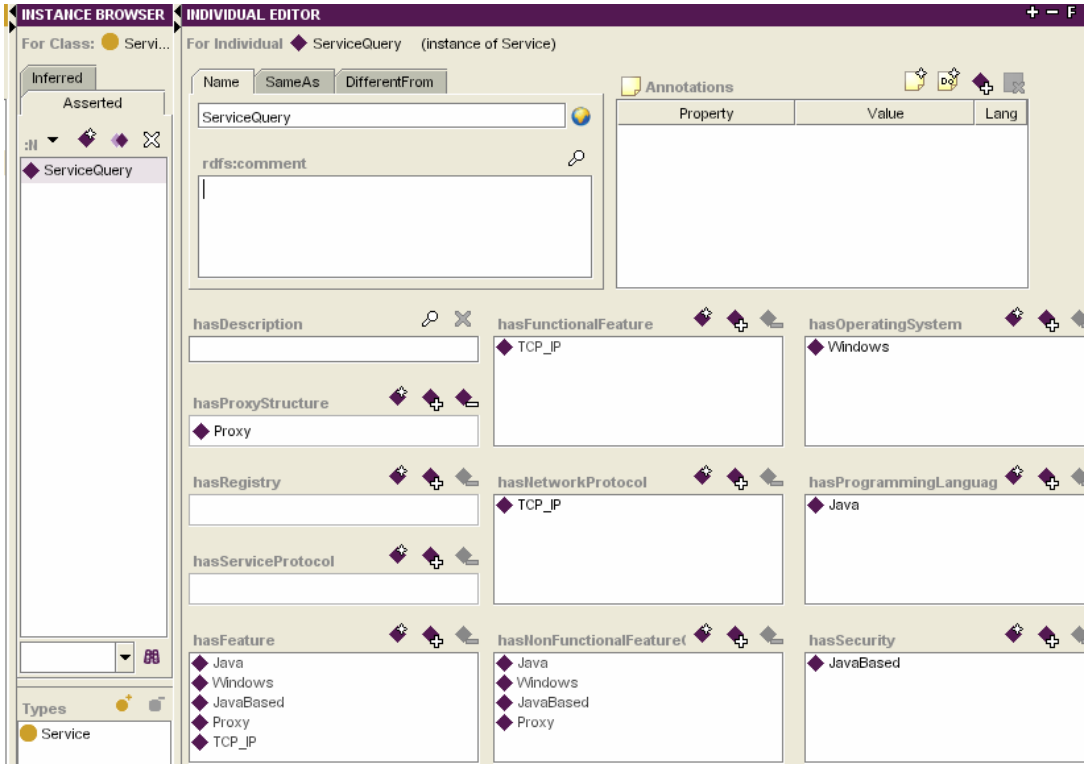
Σύμφωνα με το διάγραμμα της εικόνας 82, εξάγονται τα εξής πορίσματα:

- Ο ρυθμός με τον οποίο οι καμπυλές ομοιότητας του **άπειρου** χρήστη, που τείνουν στην τιμή της λεξικογραφικής ομοιότητας (β=1), είναι μεγαλύτερος από το ρυθμό των καμπύλων ομοιότητας του **εξειδικευμένου** και **έμπειρου** χρήστη,
- Υπάρχουν σημεία τομής των καμπύλων ομοιότητας του **εξειδικευμένου** και **έμπειρου** χρήστη, γεγονός που σημαίνει ότι μπορεί να επιτευχθεί η ίδια τιμή ομοιότητας *μειώνοντας* την πληροφορία πλαισίου και ταυτόχρονα *αυξάνοντας* το α,
- Λαμβάνοντας υπόψιν έστω και μια συσχέτιση R μεταξύ δύο υπο-κλάσεων A,B της ServiceQuery, αυτόματα η τιμή ομοιότητας ξεπερνά το όριο της λεξικογραφικής ομοιότητας (β=1). Τούτο σημαίνει ότι εισάγοντας την *ελάχιστη* (δηλαδή τουλάχιστον μία υπαρξιακή συσχέτιση ανάμεσα σε δύο πρωταρχικές υπο-κλάσεις της ServiceQuery) σημασιολογική γνώση ανεξαρτήτως από την τιμή του α, επηρεάζεται η τιμή της ομοιότητας, αφού το προτεινόμενο σύστημα τείνει *ελάχιστα* να ταξινομήσει την ερώτηση με βάση τη γνώση αυτή.

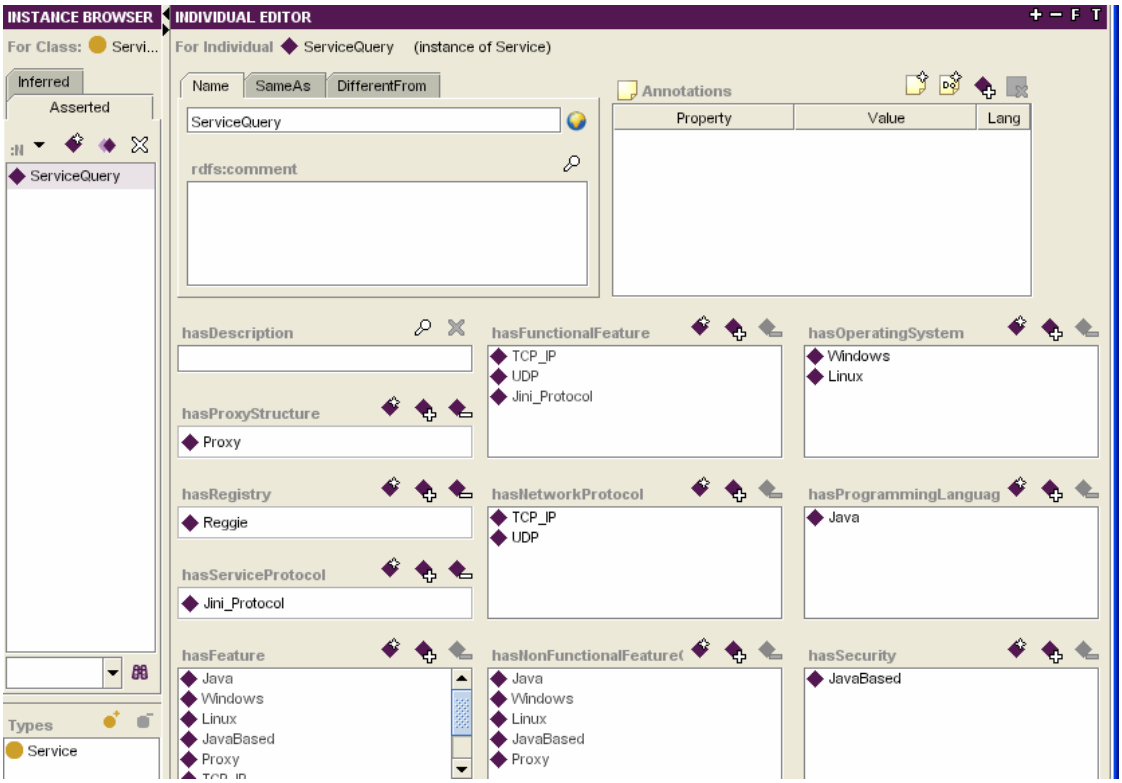
  Δηλαδή: $B \equiv \exists R.A \wedge A \subseteq \text{ServiceQuery} \wedge B \subseteq \text{ServiceQuery}$



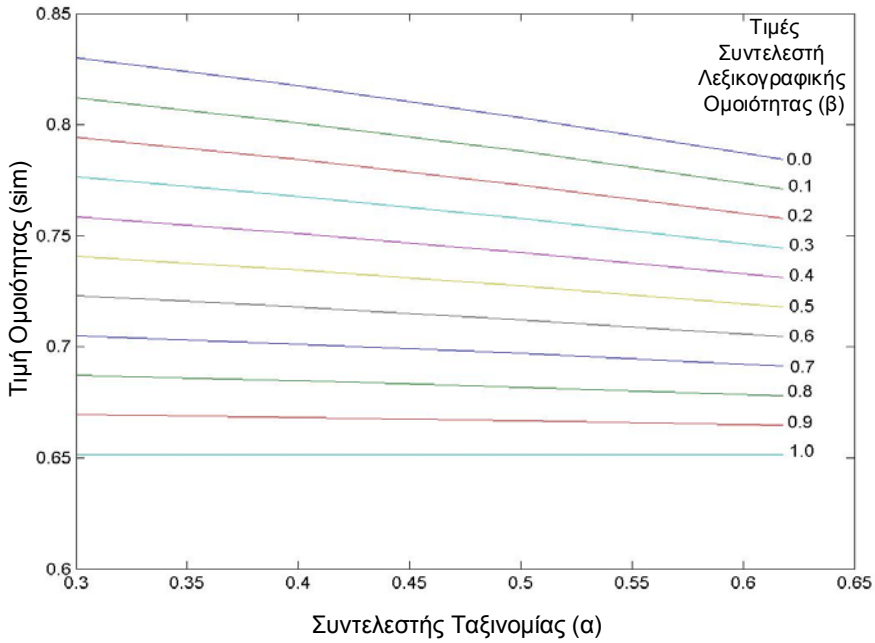
Εικόνα 77:Αναζήτηση υπηρεσίας από άπειρο χρήστη



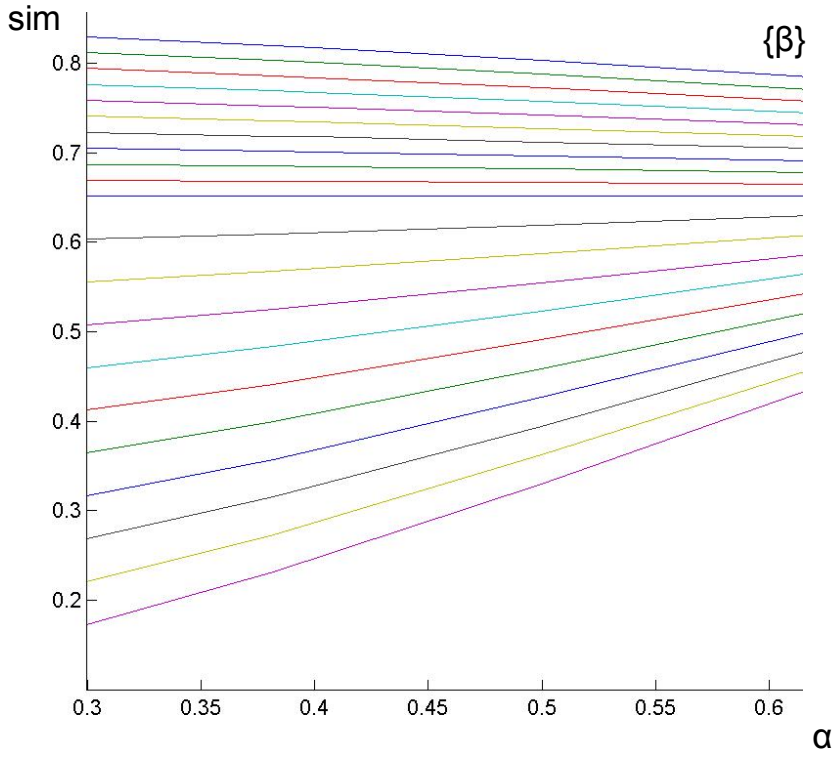
Εικόνα 78: Αναζήτηση υπηρεσίας από έμπειρο χρήστη



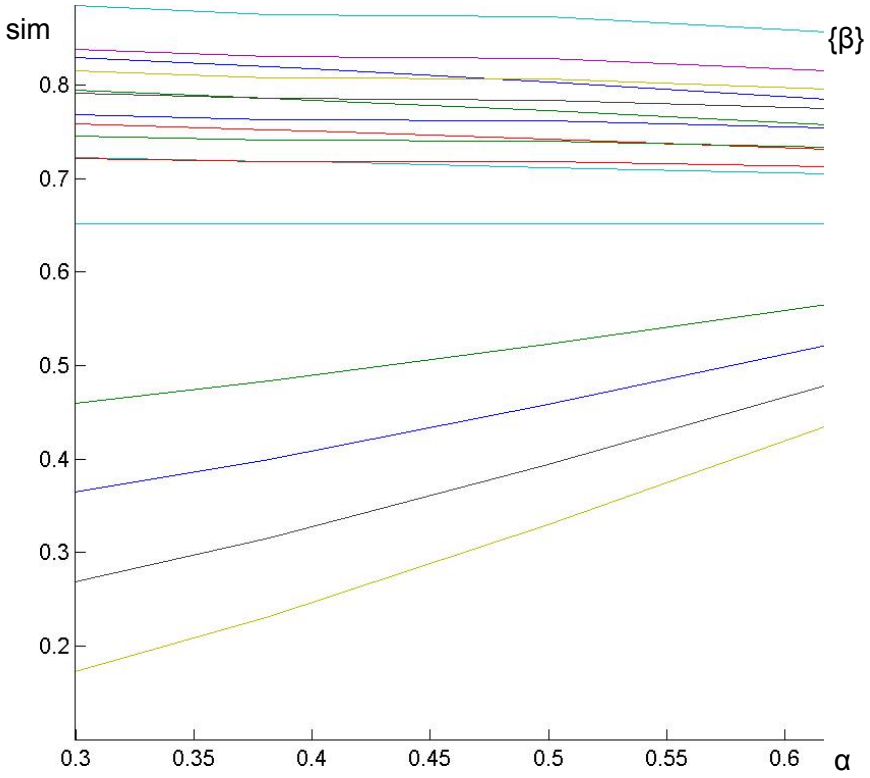
Εικόνα 79: Αναζήτηση υπηρεσίας από εξειδικευμένο χρήστη



Εικόνα 80: Καμπύλες ομοιότητας (sim) υπηρεσιών του εξειδικευμένου χρήστη



Εικόνα 81: Καμπύλες ομοιότητας (sim) υπηρεσιών εξειδικευμένου και άπειρου χρήστη



Εικόνα 82: Καμπύλες ομοιότητας (sim) υπηρεσιών εξειδικευμένου, έμπειρου και άπειρου χρήστη

11 ΠΑΡΑΡΤΗΜΑ

11.1 Κώδικας

11.1.1 Ασαφής Λεξικογραφική Ομοιότητα

class: fuzzytextsimilarity.DocumentSet

```
package fuzzytextsimilarity;
import java.util.*;
public class DocumentSet {
  public Vector documents = new Vector();
  public Vector terms = new Vector();
  public Vector cats = new Vector();
  public static Vector TCW = new Vector();
  public static Hashtable table = new Hashtable();

  public DocumentSet(Vector documents,Vector terms, Vector cats) {
    this.terms = terms;
    this.documents = documents;
    this.cats = cats;
  }
  public void getTable(){
    Enumeration doc = this.documents.elements();
    while(doc.hasMoreElements()){
      Document d = (Document)doc.nextElement();
      Enumeration ter = this.terms.elements();
      while(ter.hasMoreElements()){
        Term t = (Term)ter.nextElement();
         Context context = new Context();
        context.document = d;
        context.category = d.category;
        context.term = t;
        context.weight = d.getWeightOfTerm(t);
        DocumentSet.table.put(context.id,context);
      }
    }
  }

  private boolean inInVector(String cat,Vector cats){
    Enumeration e = cats.elements();
    while(e.hasMoreElements()){
      String index = (String)e.nextElement();
      if(index.equalsIgnoreCase(cat)) return true;
    }
    return false;
  }

  //Υπολογισμός μονάδα συγγένειας όρου-λέξης με λεξικογραφική κατηγορία ξi.
  public double computeTermCategoryMembership(Term term, String category){
    int upper = 0;
    int lower = 0;
    Enumeration e = DocumentSet.TCW.elements();
    while(e.hasMoreElements()){
      TermCategoryWeight tcw = (TermCategoryWeight)e.nextElement();
      if((tcw.category.equalsIgnoreCase(category))&&(tcw.term.name.equalsIgnoreCase(term.name)))
        upper += tcw.weight;
    }
    Enumeration u = DocumentSet.TCW.elements();
    while(u.hasMoreElements()){
      TermCategoryWeight tcw = (TermCategoryWeight)u.nextElement();
      if((tcw.term.name.equalsIgnoreCase(term.name)))
        lower += tcw.weight;
    }
    return (double)upper/(double)lower;
  }
```

//Υπολογισμός μονάδα συγγένειας όρου-λέξης με το σύνολο των λεξικογραφικών κατηγοριών Ξ.

```
 public void computeTermCategories(Term term, Vector cats){
  int weights[] = new int[cats.size()];
  String catname[] = new String[cats.size()];
  for(int j=0;j<cats.size();j++){
   catname[j] = (String)cats.elementAt(j);
   weights[j] = 0;
  }
  int i = 0;
  Enumeration catenum = cats.elements();
  while(catenum.hasMoreElements()){
   String category = (String)catenum.nextElement();
   Enumeration e = DocumentSet.table.elements();
   while(e.hasMoreElements()){
    Context context = (Context)e.nextElement();
    String catcontext = context.category;
    if((catcontext.equalsIgnoreCase(category))&&(term.name.equalsIgnoreCase(context.term.name))){
     weights[i]+=context.document.getWeightOfTerm(term);
    }
   }
   i++;
  }
  for(int j=0;j<cats.size();j++){
   TermCategoryWeight tcw = new TermCategoryWeight();
   tcw.category = catname[j];
   tcw.term = term;
   tcw.weight = weights[j];
   DocumentSet.TCW.addElement(tcw);
   //System.out.println("TERM "+tcw.term.name+" OF CAT NAME "+tcw.category+" WEIGHTS "+tcw.weight);
  }
 }

 private double max(double a,double b){
  if(a>b) return a;
  return b;
 }

 private double min(double a,double b){
  if(a<b) return a;
  return b;
 }

//Υπολογισμός μονάδα συγγένειας εγγράφου περιγραφής υπηρεσίας με λεξικογραφική κατηγορία.
 public double compupteSimilarityDocumentInCategory(Document doc, String category){
  double upper = 0.0;
  double lower = 0.0;
  for(int i=0;i<doc.terms.size();i++){
   double is = this.min(this.computeTermCategoryMembership((Term)doc.terms.elementAt(i),category),
                        doc.computeMembershipDegreeOfTermInDocument((Term)doc.terms.elementAt(i)));
   upper += is;
   double os = this.max(this.computeTermCategoryMembership((Term)doc.terms.elementAt(i),category),
                        doc.computeMembershipDegreeOfTermInDocument((Term)doc.terms.elementAt(i)));
   lower += os;
  }
  return upper/lower;
 }
}
```

11.1.2 Σημασιολογική Ομοιότητα

class: hola. SemanticSimilarity

```
package hola;
import java.io.*;
import java.util.*;
import edu.stanford.smi.protegex.owl.*;
import edu.stanford.smi.protegex.owl.model.*;

public class SemanticSimilarity {

//Υπολογισμός ομοιότητας σχέσεων μεταξύ δύο στιγμιοτύπων σε καθηορισμένη οντολογία.
```

```
protected static double relationalSimilarity(OWLModel model,
                                    OWLIndividual first,
                                    OWLIndividual second) {
  if (first.getLocalName().equalsIgnoreCase(second.getLocalName())) {
    return 1.0;
  }
  try {
    Vector firstRangeRelations = SemanticSimilarity. createCommonRangeProperty(model, first);
    Vector secondRangeRelations = SemanticSimilarity. createCommonRangeProperty(model, second);
    Vector commonPropertiesOfRange = SemanticSimilarity.
       owlObjectPropertyConjunctionOf(firstRangeRelations, secondRangeRelations);

    Vector firstDomainRelations = SemanticSimilarity.
       createCommonDomainProperty(model, first);
    Vector secondDomainRelations = SemanticSimilarity.
       createCommonDomainProperty(model, second);
    Vector commonPropertiesOfDomain = SemanticSimilarity.
       owlObjectPropertyConjunctionOf(firstDomainRelations,
                                secondDomainRelations);

    //get the rangeSumma of the relational similarity
    double rangeSumma = 0.0;
    Enumeration e = commonPropertiesOfRange.elements();
    while (e.hasMoreElements()) {
      OWLObjectProperty property = (OWLObjectProperty) e.nextElement();
      double localSimilarity = SemanticSimilarity.
         similarityForOneRelationAtRange(first, second, property);
      rangeSumma += localSimilarity;
    }

    //get the domainSumma of the relational similarity
    double domainSumma = 0.0;
    Enumeration x = commonPropertiesOfDomain.elements();
    while (x.hasMoreElements()) {
      OWLObjectProperty property = (OWLObjectProperty) x.nextElement();
      double localSimilarity = SemanticSimilarity.
         similarityForOneRelationAtDomain(first, second, property);
      domainSumma += localSimilarity;
    }

    double similarity =
       ( (double) rangeSumma + (double) domainSumma) /
       ( (double) (commonPropertiesOfRange.size()) +
        (double) (commonPropertiesOfDomain.size()));
    return similarity;
  }
  catch (Exception ex) {
    ex.printStackTrace(System.out);
  }
  return 0.0;
}

private static double similarityForOneRelationAtRange(OWLIndividual first,
    OWLIndividual second, OWLObjectProperty property) {
  try {
    Vector first_set = SemanticSimilarity.getAssociatedRangeInstances(
       property, first);
    Vector second_set = SemanticSimilarity.getAssociatedRangeInstances(
       property, second);
    Collection c_max, c_min;
    if (first_set.size() == 0 || second_set.size() == 0) {
      return 0.0;
    }
    if (first_set.size() > second_set.size()) {
      c_max = first_set;
      c_min = second_set;
    }
    else {
      c_max = second_set;
      c_min = first_set;
```

```
    }
    double max_similarity = 0.0;
    double summa_similarity = 0.0;
    for (Iterator it = c_max.iterator(); it.hasNext(); ) {
      OWLIndividual a = (OWLIndividual) it.next();
      for (Iterator esco = c_min.iterator(); esco.hasNext(); ) {
        OWLIndividual b = (OWLIndividual) esco.next();
        double similarity = SemanticSimilarity.taxonomySimilarity(a, b);
        if (similarity > max_similarity) {
          max_similarity = similarity;
        }
      }
      summa_similarity += max_similarity;
    }
    return ( (double) summa_similarity / c_max.size());
  }
  catch (Exception ex) {
    ex.printStackTrace(System.out);
    return -1.0;
  }
}

 private static double similarityForOneRelationAtDomain(
    OWLIndividual first,
    OWLIndividual second,
    OWLObjectProperty property) {

  try {
    Vector first_set = SemanticSimilarity.
        getAssociatedDomainInstances(property, first);
    Vector second_set = SemanticSimilarity.
        getAssociatedDomainInstances(property, second);

    Collection c_max, c_min;
    if (first_set.size() == 0 || second_set.size() == 0) {
      return 0.0;
    }
    if (first_set.size() > second_set.size()) {
      c_max = first_set;
      c_min = second_set;
    }
    else {
      c_max = second_set;
      c_min = first_set;
    }
    double max_similarity = 0.0;
    double summa_similarity = 0.0;
    for (Iterator it = c_max.iterator(); it.hasNext(); ) {
      OWLIndividual a = (OWLIndividual) it.next();
      for (Iterator esco = c_min.iterator(); esco.hasNext(); ) {
        OWLIndividual b = (OWLIndividual) esco.next();
        double similarity = SemanticSimilarity.taxonomySimilarity(a, b);
        if (similarity > max_similarity) {
          max_similarity = similarity;
        }
      }
      summa_similarity += max_similarity;
    }
    return ( (double) summa_similarity / c_max.size());
  }
  catch (Exception ex) {
    ex.printStackTrace(System.out);
    return -1.0;
  }
}

 private static Vector getAssociatedRangeInstances(
    OWLObjectProperty property,
    OWLIndividual individual) throws Exception {

   Vector associatedInstances = new Vector();
```

```
  OWLNamedClass domain = (OWLNamedClass) ( (RDFSClass) property.getDomain(false).
                              as(OWLNamedClass.class));
  for (Iterator it = domain.getInstances().iterator(); it.hasNext(); ) {
   OWLIndividual candidate = (OWLIndividual) it.next();
   if (candidate != null) {
    Collection candidateRelates = candidate.getPropertyValues(property);
    for (Iterator back = candidateRelates.iterator(); back.hasNext(); ) {
     OWLIndividual maybe = (OWLIndividual) back.next();
     if (maybe != null) {
      if (maybe.getLocalName().equalsIgnoreCase(individual.getLocalName())) {
       associatedInstances.addElement(candidate);
      }
     }
    }
   }
  }
  return associatedInstances;
 }

 private static Vector getAssociatedDomainInstances(
    OWLObjectProperty property,
    OWLIndividual individual) throws Exception {

  Vector associatedInstances = new Vector();
  OWLNamedClass range = (OWLNamedClass) ( (RDFSClass) property.getRange(false).
                               as(OWLNamedClass.class));

  for (Iterator it = range.getInstances().iterator(); it.hasNext(); ) {
   OWLIndividual candidate = (OWLIndividual) it.next();
   if (candidate != null) {
    Collection candidateRelates = individual.getPropertyValues(property);
    for (Iterator back = candidateRelates.iterator(); back.hasNext(); ) {
     OWLIndividual maybe = (OWLIndividual) back.next();
     if (maybe != null) {
      if (maybe.getLocalName().equalsIgnoreCase(candidate.getLocalName())) {
       associatedInstances.addElement(candidate);
      }
     }
    }
   }
  }
  return associatedInstances;
 }

 private static Vector createCommonRangeProperty(OWLModel model,
                                  OWLIndividual individual) throws
    Exception {

  Vector commonRangeProperties = new Vector();
  //retrieve the concept of the instance
  OWLNamedClass concept = (OWLNamedClass) individual.getRDFType().as(
     OWLNamedClass.class);
  //retrieve all the OWLObjectProperties defined by user
  Collection allProperties = model.getUserDefinedOWLObjectProperties();
  for (Iterator prop = allProperties.iterator(); prop.hasNext(); ) {
   OWLObjectProperty objectproperty = (OWLObjectProperty) prop.next();
   //examine for any null ranges of the property
   if (objectproperty.getRange() == null) {
    prop.next();
   }
   else {
    //examine whether such property is range of any (super)concept of the individual
    if (SemanticSimilarity.isInRangeHierarchy(objectproperty, concept)) {
     commonRangeProperties.addElement(objectproperty);
    }
   }
  }
  return commonRangeProperties;
 }

 private static Vector createCommonDomainProperty(
```

```
    OWLModel model,
    OWLIndividual individual) throws Exception {
  Vector commonDomainProperties = new Vector();
  //retrieve the concept of the instance
  OWLNamedClass concept = (OWLNamedClass) individual.
     getRDFType().as(OWLNamedClass.class);
  //retrieve all the OWLObjectProperties defined by user
  Collection allProperties = model.getUserDefinedOWLObjectProperties();
  for (Iterator prop = allProperties.iterator(); prop.hasNext(); ) {
   OWLObjectProperty objectproperty = (OWLObjectProperty) prop.next();
   //examine for any null ranges of the property
   if (objectproperty.getDomain() == null) {
     prop.next();
   }
   else {
     //examine whether such property is domain of any (super)concept of the individual
     //System.out.println(objectproperty.getLocalName());
     if (SemanticSimilarity.isInDomainHierarchy(objectproperty, concept)) {
       commonDomainProperties.addElement(objectproperty);
     }
   }
  }
  return commonDomainProperties;
 }

 private static boolean isInRangeHierarchy(OWLObjectProperty property,
                                 OWLNamedClass concept) {
  OWLNamedClass range = (OWLNamedClass) property.getRange();
  Collection hierarchy = SemanticSimilarity.upwardsCotopy(concept);
  for (Iterator it = hierarchy.iterator(); it.hasNext(); ) {
   OWLNamedClass candidate = (OWLNamedClass) it.next();
   if (candidate.getLocalName().equalsIgnoreCase(range.getLocalName())) {
     return true;
   }
  }
  return false;
 }

 private static boolean isInDomainHierarchy(OWLObjectProperty property,
                                 OWLNamedClass concept) {
  RDFSClass rdfsClass = (RDFSClass) property.getDomain(false);
  if (rdfsClass != null) {
   OWLNamedClass domain = (OWLNamedClass) ( (RDFSClass) property.getDomain(false)).
      as(OWLNamedClass.class);
   Collection hierarchy = SemanticSimilarity.upwardsCotopy(concept);
   for (Iterator it = hierarchy.iterator(); it.hasNext(); ) {
     OWLNamedClass candidate = (OWLNamedClass) it.next();
     if (candidate.getLocalName().equalsIgnoreCase(domain.getLocalName())) {
       return true;
     }
   }
  }
  return false;
 }

//Υπολογισμός upwardsCotopy κλάσης σε καθορισμένη ταξινομία
 protected static Vector upwardsCotopy(OWLNamedClass concept) {
  Vector cotopy = new Vector();
  Collection collection = concept.getSuperclasses(true);
  for (Iterator it = collection.iterator(); it.hasNext(); ) {
   RDFResource rdfr = (RDFResource) it.next();
   if (!rdfr.canAs(OWLRestriction.class)) {
     OWLNamedClass owlconcept = (OWLNamedClass) rdfr.as(OWLNamedClass.class);
     cotopy.addElement(owlconcept);
   }
  }
  cotopy.addElement(concept);
  return cotopy;
 }

//Υπολογισμός conceptMatching δύο κλάσεων σε καθορισμένη ταξινομία
```

```
 protected static double conceptMatching(OWLNamedClass first,
                                OWLNamedClass second) {
  int con = SemanticSimilarity.conjunctionOf(SemanticSimilarity.upwardsCotopy(
     first), SemanticSimilarity.upwardsCotopy(second)).size();
  int dis = SemanticSimilarity.disjunctionOf(SemanticSimilarity.upwardsCotopy(
     first), SemanticSimilarity.upwardsCotopy(second)).size();
  return ( (double) con / (double) dis);
 }

 private static Vector disjunctionOf(Collection C1, Collection C2) {
  Vector disjunction = new Vector();
  boolean deletion[] = new boolean[C1.size() + C2.size()];

  for (int i = 0; i < C1.size() + C2.size(); i++) {
   deletion[i] = false;

  }
  for (Iterator it = C1.iterator(); it.hasNext(); ) {
   disjunction.addElement( (OWLNamedClass) it.next());

  }
  for (Iterator it = C2.iterator(); it.hasNext(); ) {
   disjunction.addElement( (OWLNamedClass) it.next());

  }

  for (int j = 0; j < disjunction.size(); j++) {
   OWLNamedClass concept = (OWLNamedClass) disjunction.elementAt(j);
   for (int i = 0; i < disjunction.size(); i++) {
    if (i != j && deletion[j] == deletion[i]) {
     OWLNamedClass candidate = (OWLNamedClass) disjunction.elementAt(i);
     if (concept.getLocalName().equalsIgnoreCase(candidate.getLocalName())) {
      deletion[i] = true;
     }
    }
   }
  }
  Vector retour = new Vector();
  for (int i = 0; i < disjunction.size(); i++) {
   if (deletion[i] != true) {
    retour.addElement(disjunction.elementAt(i));
   }
  }

  return retour;
 }

//Υπολογισμός taxonomySimilarity δύο στιγμιοτύπων σε καθορισμένη οντολογία

 protected static double taxonomySimilarity(OWLIndividual first,
                                OWLIndividual second) {
  OWLNamedClass first_concept = (OWLNamedClass) first.getRDFType().as(
     OWLNamedClass.class);

  OWLNamedClass second_concept = (OWLNamedClass) second.getRDFType().as(
     OWLNamedClass.class);

  if (first.getLocalName().equalsIgnoreCase(second.getLocalName())) {
   return 1.0;
  }
  else {
   return
     SemanticSimilarity.conceptMatching(first_concept, second_concept) /
     (double) 2.0;
  }
 }

 private static Vector owlObjectPropertyConjunctionOf(
    Collection C1,
    Collection C2) {
```

```
  Collection c_min, c_max;
  Vector conjunction = new Vector();
  if (C1.size() >= C2.size()) {
    c_max = C1;
    c_min = C2;
  }
  else {
    c_max = C2;
    c_min = C1;
  }

  for (Iterator it = c_min.iterator(); it.hasNext(); ) {
    OWLObjectProperty concept = (OWLObjectProperty) it.next();
    for (Iterator tt = c_max.iterator(); tt.hasNext(); ) {
      OWLObjectProperty candidate = (OWLObjectProperty) tt.next();
      if (concept.getLocalName().equalsIgnoreCase(candidate.getLocalName())) {
        conjunction.addElement(concept);
      }
    }
  }
  return conjunction;
}

private static Vector conjunctionOf(Collection C1, Collection C2) {
  Collection c_min, c_max;
  Vector conjunction = new Vector();
  if (C1.size() >= C2.size()) {
    c_max = C1;
    c_min = C2;
  }
  else {
    c_max = C2;
    c_min = C1;
  }

  for (Iterator it = c_min.iterator(); it.hasNext(); ) {
    OWLNamedClass concept = (OWLNamedClass) it.next();
    for (Iterator tt = c_max.iterator(); tt.hasNext(); ) {
      OWLNamedClass candidate = (OWLNamedClass) tt.next();
      if (concept.getLocalName().equalsIgnoreCase(candidate.getLocalName())) {
        conjunction.addElement(concept);
      }
    }
  }
  return conjunction;
}
```

//Υπολογισμός similarityOf ομοιότητας στιγμιοτύπου σε καθορισμένη οντολογία, συντελεστή ομοιότητας ταξινομίας και επιστρέφει σύνολο με τις πιο όμοιες ταξινομημένα υπηρεσίες.

```
public static Vector similarityOf(int selection, double coefficient,
                          OWLIndividual individual, OWLModel model) {
  Collection individuals[] = new Collection[6];
  individuals[0] = model.getOWLNamedClass("UPnPService").getInstances(false);
  individuals[1] = model.getOWLNamedClass("SLPService").getInstances(false);
  individuals[2] = model.getOWLNamedClass("SalutationService").getInstances(false);
  individuals[3] = model.getOWLNamedClass("JiniService").getInstances(false);
  individuals[4] = model.getOWLNamedClass("BluetoothService").getInstances(false);
  individuals[5] = model.getOWLNamedClass("Service").getInstances(false);
  //System.out.println("SERVICES ARE "+individuals[5].size());
  //System.exit(0);
  for (int i = 0; i < 6; i++) {
    for (Iterator it = individuals[i].iterator(); it.hasNext(); ) {
      OWLIndividual candidate = (OWLIndividual) it.next();
      double taxonomy = SemanticSimilarity.taxonomySimilarity(individual,
          candidate);
      double relational = SemanticSimilarity.relationalSimilarity(model,
          individual, candidate);
      double similarity = coefficient * taxonomy +
          (1 - coefficient) * relational;
```

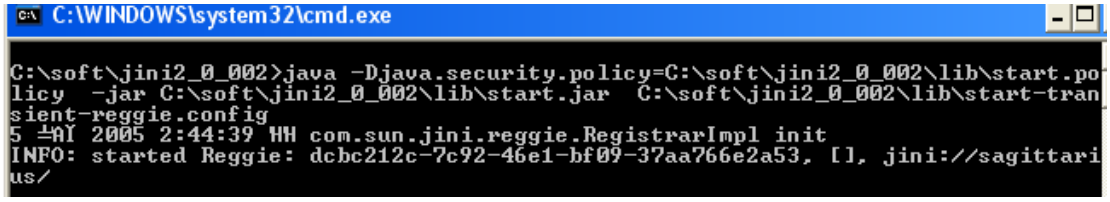
```
      SimilarObject.setSimilarObject(new SimilarObject(candidate, similarity));
    }
   }
   return SimilarObject.sortSimilarObjects();
 }

 private static SimVector calculateSimVectorOf(double coefficient,
                                               OWLIndividual individual,
                                               OWLModel model,
                                               Vector homeSet,
                                               Vector guestSet) {

   double home[] = new double[homeSet.size()];
   int home_index = 0;

   double guest[] = new double[guestSet.size()];
   int guest_index = 0;

   SimilarObject.clearSimilarObjects();
   Vector sorted = SemanticSimilarity.
      similarityOf(0, coefficient, individual, model);
   //select the individuals of homeSet and guestSet
   Enumeration e = sorted.elements();
   while (e.hasMoreElements()) {
    SimilarObject simObj = (SimilarObject) e.nextElement();
    if (SemanticSimilarity.isInSameSet(homeSet, simObj.service)) {
      //in the homeSet
      home[home_index] = simObj.similarity;
      //System.out.println("Individual "+ simObj.service.getLocalName()+" is in the same set");
      home_index++;
    }
    if (SemanticSimilarity.isInSameSet(guestSet, simObj.service)) {
      //in the guestSet
      guest[guest_index] = simObj.similarity;
      //System.out.println("Individual "+ simObj.service.getLocalName()+" is out of the set");
      guest_index++;
    }
   }
   return new SimVector(home, guest, individual);
 }

 private static boolean isInSameSet(Vector set, OWLIndividual individual) {
   Enumeration e = set.elements();
   while (e.hasMoreElements()) {
    OWLIndividual candidate = (OWLIndividual) e.nextElement();
    if (individual.getLocalName().
        equalsIgnoreCase(candidate.getLocalName())) {
      return true;
    }
   }
   return false;
 }

 private static double[] calculateSimVectorSumOf(Vector members) {
   int vectorLength = ( (SimVector) members.elementAt(0)).simVector.length;
   double dimensionSum[] = new double[vectorLength];
   double vector[] = new double[vectorLength];

   for (int current_dim = 0; current_dim < vectorLength; current_dim++) {
    for (int i = 0; i < members.size(); i++) {
      SimVector v = (SimVector) members.elementAt(i);
      dimensionSum[current_dim] += v.simVector[current_dim];
    }
    vector[current_dim] = dimensionSum[current_dim];
   }
   return vector;
 }

 private static double calculateABSValueOf(double vector[]) {
   double abs = 0;
   for (int i = 0; i < vector.length; i++) {
```

```
    double value = vector[i];
    abs += value * value;
   }
   return Math.sqrt(abs);
  }

  private static double calculateProductVectorOf(double a[], double b[]) {
   double sum = 0.0;
   for (int i = 0; i < a.length; i++) {
    sum += a[i] * b[i];
   }
   return sum;
  }

  public static void main(String[] args) {
   try{
   FileOutputStream p = new FileOutputStream(new File("output.html"));
   PrintWriter pw = new PrintWriter(p);

   for (double t = 0.3; t < 0.9; t += 1.0) {
    pw.write("<B>COEFFICIENT TAXONOMY = "+t+ "</B>");

      String fileonto = "C:\\Program Files\\ProtegeKleri\\Hola\\ServiceOntology.owl";
      FileInputStream fir = null;
      fir = new FileInputStream(new File(fileonto));
      OWLModel model = ProtegeOWL.createJenaOWLModelFromInputStream(fir);

      OWLIndividual JiniService_01 = model.getOWLIndividual("Printer");
      OWLIndividual JiniService_02 = model.getOWLIndividual("FileClassifier");
      OWLIndividual BluetoothService_01 = model.getOWLIndividual("FileTransfer");
      OWLIndividual BluetoothService_02 = model.getOWLIndividual("MobilePrinter");
      OWLIndividual SalutationService_01 = model.getOWLIndividual("NearestFriend");
      OWLIndividual SLPService_01 = model.getOWLIndividual("NearestPrinter");
      OWLIndividual UPnPService_01 = model.getOWLIndividual("PrinterFile");
      OWLIndividual UPnPService_02 = model.getOWLIndividual("DigitalMouse");

      OWLIndividual Service_01 = model.getOWLIndividual("ServiceQuery");

      if (Service_01 == null) {
       System.exit(0);
      }
      //request set element
      Vector setE = new Vector();
      setE.addElement(Service_01);

      //ontology instances
      Vector setF = new Vector();
      setF.addElement(JiniService_01);
      setF.addElement(JiniService_02);
      setF.addElement(BluetoothService_01);
      setF.addElement(BluetoothService_02);
      setF.addElement(SalutationService_01);
      setF.addElement(SLPService_01);
      setF.addElement(UPnPService_01);
      setF.addElement(UPnPService_02);

      SimVector Service_01_SimVector = SemanticSimilarity.calculateSimVectorOf(t, Service_01, model, setE, setF);

      SimVector BluetoothService_02_SimVector = SemanticSimilarity.calculateSimVectorOf(t, BluetoothService_02, model, setE,
setF);
      SimVector BluetoothService_01_SimVector = SemanticSimilarity.calculateSimVectorOf(t, BluetoothService_01, model, setE,
setF);
      SimVector SalutationService_01_SimVector = SemanticSimilarity.calculateSimVectorOf(t, SalutationService_01, model, setE,
setF);
      SimVector SLPService_01_SimVector = SemanticSimilarity.calculateSimVectorOf(t, SLPService_01, model, setE, setF);
      SimVector UPnPService_01_SimVector = SemanticSimilarity.calculateSimVectorOf(t, UPnPService_01, model, setE, setF);
      SimVector UPnPService_02_SimVector = SemanticSimilarity.calculateSimVectorOf(t, UPnPService_02, model, setE, setF);
      SimVector JiniService_01_SimVector = SemanticSimilarity.calculateSimVectorOf(t, JiniService_01, model, setE, setF);
      SimVector JiniService_02_SimVector = SemanticSimilarity.calculateSimVectorOf(t, JiniService_02, model, setE, setF);

      //home similarity
```

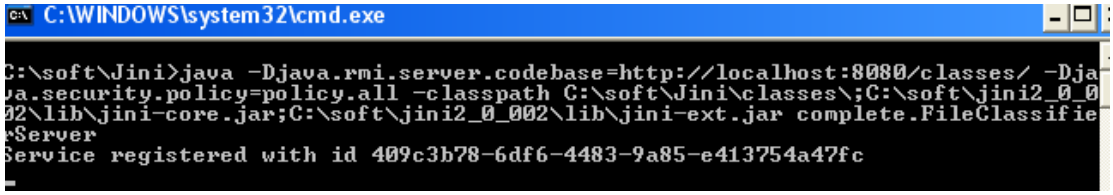
```
pw.print("<P>FIRST SET VECTORS</P>");
for (int i = 0; i < Service_01_SimVector.simVector.length; i++) {
  pw.print("<BR><I>"+Service_01_SimVector.simVector[i]+"</I></BR>");
}
pw.print("<HR>");

//guest similarity
pw.print("<P>SECOND SET VECTORS</P>");
for (int i = 0; i < BluetoothService_02_SimVector.simVector.length; i++) {
  pw.print("<BR><I>"+BluetoothService_02_SimVector.simVector[i]+"</I></BR>");
}
pw.print("<HR>");
for (int i = 0; i < BluetoothService_01_SimVector.simVector.length; i++) {
   pw.print("<BR><I>"+BluetoothService_01_SimVector.simVector[i]+"</I></BR>");
 }
pw.print("<HR>");
for (int i = 0; i < SalutationService_01_SimVector.simVector.length; i++) {
  pw.print("<BR><I>"+SalutationService_01_SimVector.simVector[i]+"</I></BR>");
}
pw.print("<HR>");
for (int i = 0; i < SLPService_01_SimVector.simVector.length; i++) {
  pw.print("<BR><I>"+SLPService_01_SimVector.simVector[i]+"</I></BR>");
}
pw.print("<HR>");
for (int i = 0; i < UPnPService_01_SimVector.simVector.length; i++) {
  pw.print("<BR><I>"+UPnPService_01_SimVector.simVector[i]+"</I></BR>");
}
pw.print("<HR>");
for (int i = 0; i < UPnPService_02_SimVector.simVector.length; i++) {
  pw.print("<BR><I>"+UPnPService_02_SimVector.simVector[i]+"</I></BR>");
}
pw.print("<HR>");
for (int i = 0; i < JiniService_01_SimVector.simVector.length; i++) {
  pw.print("<BR><I>"+JiniService_01_SimVector.simVector[i]+"</I></BR>");
}
pw.print("<HR>");
for (int i = 0; i < JiniService_02_SimVector.simVector.length; i++) {
  pw.print("<BR><I>"+JiniService_02_SimVector.simVector[i]+"</I></BR>");
}

//sum of home set
Vector members = new Vector();
members.addElement(Service_01_SimVector);
pw.print("<HR><B>THE SUMVECTOR OF THE FIRST SET</B>");
double a[] = SemanticSimilarity.calculateSimVectorSumOf(members);
for (int i = 0; i < a.length; i++) {
  pw.print("<BR><I>" + a[i] + "</I>");

}

//sum of guest set
Vector members1 = new Vector();
members1.addElement(BluetoothService_02_SimVector);
members1.addElement(BluetoothService_01_SimVector);
members1.addElement(SalutationService_01_SimVector);
members1.addElement(SLPService_01_SimVector);
members1.addElement(JiniService_02_SimVector);
members1.addElement(JiniService_01_SimVector);
members1.addElement(UPnPService_02_SimVector);
members1.addElement(UPnPService_01_SimVector);
pw.print("<HR><B>THE SUMVECTOR OF THE SECOND SET</B>");
double b[] = SemanticSimilarity.calculateSimVectorSumOf(members1);
for (int i = 0; i < b.length; i++) {
  pw.print("<BR><I>" + b[i] + "</I>");
}
double s = SemanticSimilarity.calculateProductVectorOf(a, b);
double abs_a = SemanticSimilarity.calculateABSValueOf(a);
double abs_b = SemanticSimilarity.calculateABSValueOf(b);
 pw.print("<HR>");
 pw.print("<BR><B>ABS OF FIRST SET " + abs_a+"</B>");
 pw.print("<BR><B>ABS OF SECOND SET " + abs_b+ "</B>");
```

```
        pw.print("<BR><B>SUM OF THE TWO SETS <I> " + s +"</I>");
       double similarityEF = (double) s / (double) (abs_a * abs_b);
        pw.print("<HR>");
        pw.print("<FONT COLOR = RED>SIMILARITY OF THE TWO SETS IS </B>" + similarityEF+"</FONT>");
       double radius = Math.acos(similarityEF);
       double angle = (double) radius * 180 / (double) Math.PI;
        pw.print("<BR><B>COEFICIENT "+t+" ANGLE " + angle);
       SimilarObject.clearSimilarObjects();
       pw.print("<HR>");
       pw.print("Similarity sorted with any in the Ontology");
       pw.print("<BR>");
       Enumeration e = SemanticSimilarity.similarityOf(0,t,Service_01,model).elements();
       while(e.hasMoreElements()){
         SimilarObject so = (SimilarObject)e.nextElement();
         pw.print("<BR>"+so.service.getLocalName()+"["+so.similarity+"]");
       }
       pw.flush();
  }
    pw.close();
    } catch (Exception e) {
       e.printStackTrace(System.out);
     }
    }
  }
```

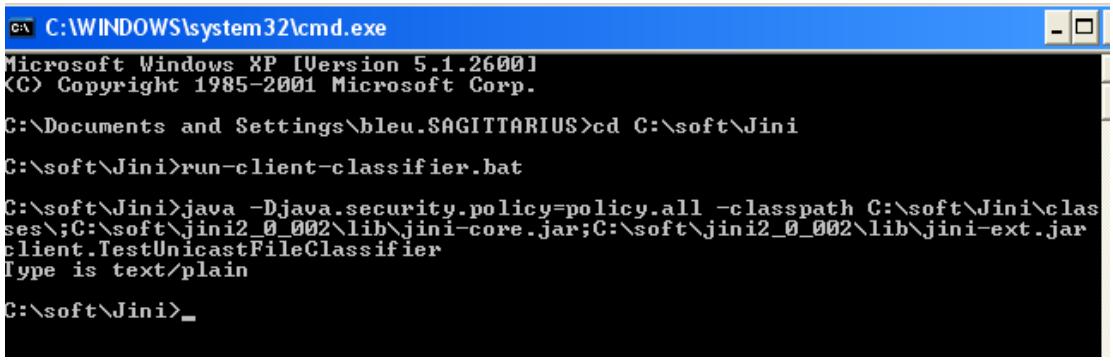
11.1.3 Jini Print Screens



Αρχικοποίηση Reggie



Καταχώρηση της υπηρεσίας FileClassifier στο Reggie



Επιτυχής εύρεση και χρήση της υπηρεσίας από client

11.2 Οντολογίες

11.2.1 Οντολογία Περιγραφής Υπηρεσιών

Αρχείο: ServiceOntology.owl

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns="http://www.owl-ontologies.com/service.owl#"
  xml:base="http://www.owl-ontologies.com/service.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Jini">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="ServiceProtocol"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:ID="Bluetooth"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="SLP"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="UPnP"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Salutation"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="SalutationService">
    <owl:disjointWith>
      <owl:Class rdf:ID="BluetoothService"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="JiniService"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="PragmaticService"/>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >
</rdfs:comment>
    <owl:disjointWith>
      <owl:Class rdf:ID="SLPService"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="UPnPService"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#UPnPService">
    <owl:disjointWith>
      <owl:Class rdf:about="#BluetoothService"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#SLPService"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#PragmaticService"/>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#SalutationService"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#JiniService"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="Service">
    <owl:disjointWith>
      <owl:Class rdf:ID="Feature"/>
```

```
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#Feature">
    <owl:disjointWith rdf:resource="#Service"/>
  </owl:Class>
  <owl:Class rdf:about="#UPnP">
    <owl:disjointWith rdf:resource="#Jini"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Salutation"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Bluetooth"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#ServiceProtocol"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:about="#SLP"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#SLPService">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#PragmaticService"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:about="#BluetoothService"/>
    </owl:disjointWith>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >
</rdfs:comment>
    <owl:disjointWith rdf:resource="#UPnPService"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#JiniService"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#SalutationService"/>
  </owl:Class>
  <owl:Class rdf:about="#Salutation">
    <owl:disjointWith>
      <owl:Class rdf:about="#SLP"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#Bluetooth"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#UPnP"/>
    <owl:disjointWith rdf:resource="#Jini"/>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#ServiceProtocol"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="ProgrammingLanguage">
    <owl:disjointWith>
      <owl:Class rdf:ID="ProxyStructure"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="Security"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:ID="OperatingSystem"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="NonFunctionalFeature"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="FunctionalFeature">
    <rdfs:subClassOf rdf:resource="#Feature"/>
  </owl:Class>
  <owl:Class rdf:about="#BluetoothService">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#PragmaticService"/>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#SalutationService"/>
```

```
  <owl:disjointWith>
    <owl:Class rdf:about="#JiniService"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#SLPService"/>
  <owl:disjointWith rdf:resource="#UPnPService"/>
 </owl:Class>
 <owl:Class rdf:about="#ProxyStructure">
  <owl:disjointWith>
    <owl:Class rdf:about="#OperatingSystem"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#Security"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#ProgrammingLanguage"/>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#NonFunctionalFeature"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:about="#OperatingSystem">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#NonFunctionalFeature"/>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#ProgrammingLanguage"/>
  <owl:disjointWith rdf:resource="#ProxyStructure"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#Security"/>
  </owl:disjointWith>
 </owl:Class>
 <owl:Class rdf:ID="NetworkProtocol">
  <rdfs:subClassOf rdf:resource="#FunctionalFeature"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#ServiceProtocol"/>
  </owl:disjointWith>
 </owl:Class>
 <owl:Class rdf:about="#Bluetooth">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ServiceProtocol"/>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Salutation"/>
  <owl:disjointWith rdf:resource="#UPnP"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#SLP"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Jini"/>
 </owl:Class>
 <owl:Class rdf:about="#Security">
  <owl:disjointWith rdf:resource="#ProxyStructure"/>
  <owl:disjointWith rdf:resource="#OperatingSystem"/>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#NonFunctionalFeature"/>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#ProgrammingLanguage"/>
 </owl:Class>
 <owl:Class rdf:about="#PragmaticService">
  <rdfs:subClassOf rdf:resource="#Service"/>
 </owl:Class>
 <owl:Class rdf:about="#ServiceProtocol">
  <owl:disjointWith rdf:resource="#NetworkProtocol"/>
  <rdfs:subClassOf rdf:resource="#FunctionalFeature"/>
 </owl:Class>
 <owl:Class rdf:about="#NonFunctionalFeature">
  <rdfs:subClassOf rdf:resource="#Feature"/>
 </owl:Class>
 <owl:Class rdf:ID="Registry">
  <rdfs:subClassOf rdf:resource="#FunctionalFeature"/>
 </owl:Class>
 <owl:Class rdf:about="#SLP">
  <rdfs:subClassOf rdf:resource="#ServiceProtocol"/>
  <owl:disjointWith rdf:resource="#Bluetooth"/>
  <owl:disjointWith rdf:resource="#Jini"/>
  <owl:disjointWith rdf:resource="#Salutation"/>
```

```
  <owl:disjointWith rdf:resource="#UPnP"/>
 </owl:Class>
 <owl:Class rdf:about="#JiniService">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >A service that has Jini Service Protocol
  </rdfs:comment>
  <owl:disjointWith rdf:resource="#SLPService"/>
  <owl:disjointWith rdf:resource="#BluetoothService"/>
  <owl:disjointWith rdf:resource="#UPnPService"/>
  <rdfs:subClassOf rdf:resource="#PragmaticService"/>
  <owl:disjointWith rdf:resource="#SalutationService"/>
 </owl:Class>
 <owl:ObjectProperty rdf:ID="isNonFunctionalFeatureOf">
  <rdfs:domain rdf:resource="#NonFunctionalFeature"/>
  <rdfs:range rdf:resource="#Service"/>
  <owl:inverseOf>
   <owl:ObjectProperty rdf:ID="hasNonFunctionalFeatureOf"/>
  </owl:inverseOf>
  <rdfs:subPropertyOf>
   <owl:ObjectProperty rdf:ID="isFeatureOf"/>
  </rdfs:subPropertyOf>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#hasNonFunctionalFeatureOf">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="#NonFunctionalFeature"/>
  <owl:inverseOf rdf:resource="#isNonFunctionalFeatureOf"/>
  <rdfs:subPropertyOf>
   <owl:ObjectProperty rdf:ID="hasFeature"/>
  </rdfs:subPropertyOf>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="isSecurityOf">
  <rdfs:domain rdf:resource="#Security"/>
  <rdfs:range rdf:resource="#Service"/>
  <rdfs:subPropertyOf rdf:resource="#isNonFunctionalFeatureOf"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <owl:inverseOf>
   <owl:ObjectProperty rdf:ID="hasSecurity"/>
  </owl:inverseOf>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="hasNetworkProtocol">
  <owl:inverseOf>
   <owl:ObjectProperty rdf:ID="isNetworkProtocolOf"/>
  </owl:inverseOf>
  <rdfs:subPropertyOf>
   <owl:ObjectProperty rdf:ID="hasFunctionalFeature"/>
  </rdfs:subPropertyOf>
  <rdfs:range rdf:resource="#NetworkProtocol"/>
  <rdfs:domain rdf:resource="#Service"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#hasFunctionalFeature">
  <owl:inverseOf>
   <owl:ObjectProperty rdf:ID="isFunctionalFeatureOf"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="#FunctionalFeature"/>
  <rdfs:subPropertyOf>
   <owl:ObjectProperty rdf:about="#hasFeature"/>
  </rdfs:subPropertyOf>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="isProxyStructureOf">
  <rdfs:subPropertyOf rdf:resource="#isNonFunctionalFeatureOf"/>
  <rdfs:range rdf:resource="#Service"/>
  <rdfs:domain rdf:resource="#ProxyStructure"/>
  <owl:inverseOf>
   <owl:FunctionalProperty rdf:ID="hasProxyStructure"/>
  </owl:inverseOf>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="hasRegistry">
  <rdfs:range rdf:resource="#Registry"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Service"/>
```

```
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#hasFeature">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="#Feature"/>
  <owl:inverseOf>
   <owl:ObjectProperty rdf:about="#isFeatureOf"/>
  </owl:inverseOf>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="isOperatingSystemOf">
  <rdfs:subPropertyOf rdf:resource="#isNonFunctionalFeatureOf"/>
  <rdfs:domain rdf:resource="#OperatingSystem"/>
  <rdfs:range rdf:resource="#Service"/>
  <owl:inverseOf>
   <owl:ObjectProperty rdf:ID="hasOperatingSystem"/>
  </owl:inverseOf>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="isProgrammingLanguageOf">
  <rdfs:range rdf:resource="#Service"/>
  <rdfs:subPropertyOf rdf:resource="#isNonFunctionalFeatureOf"/>
  <rdfs:domain rdf:resource="#ProgrammingLanguage"/>
  <owl:inverseOf>
   <owl:ObjectProperty rdf:ID="hasProgrammingLanguage"/>
  </owl:inverseOf>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#hasProgrammingLanguage">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:subPropertyOf rdf:resource="#hasNonFunctionalFeatureOf"/>
  <rdfs:range rdf:resource="#ProgrammingLanguage"/>
  <owl:inverseOf rdf:resource="#isProgrammingLanguageOf"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="isServiceProtocolOf">
  <owl:inverseOf>
   <owl:FunctionalProperty rdf:ID="hasServiceProtocol"/>
  </owl:inverseOf>
  <rdfs:range rdf:resource="#Service"/>
  <rdfs:subPropertyOf>
   <owl:ObjectProperty rdf:about="#isFunctionalFeatureOf"/>
  </rdfs:subPropertyOf>
  <rdfs:domain rdf:resource="#ServiceProtocol"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#isFunctionalFeatureOf">
  <rdfs:range rdf:resource="#Service"/>
  <owl:inverseOf rdf:resource="#hasFunctionalFeature"/>
  <rdfs:subPropertyOf>
   <owl:ObjectProperty rdf:about="#isFeatureOf"/>
  </rdfs:subPropertyOf>
  <rdfs:domain rdf:resource="#FunctionalFeature"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#isFeatureOf">
  <owl:inverseOf rdf:resource="#hasFeature"/>
  <rdfs:domain rdf:resource="#Feature"/>
  <rdfs:range rdf:resource="#Service"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#hasOperatingSystem">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:subPropertyOf rdf:resource="#hasNonFunctionalFeatureOf"/>
  <rdfs:range rdf:resource="#OperatingSystem"/>
  <owl:inverseOf rdf:resource="#isOperatingSystemOf"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#hasSecurity">
  <owl:inverseOf rdf:resource="#isSecurityOf"/>
  <rdfs:subPropertyOf rdf:resource="#hasNonFunctionalFeatureOf"/>
  <rdfs:range rdf:resource="#Security"/>
  <rdfs:domain rdf:resource="#Service"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#isNetworkProtocolOf">
  <rdfs:subPropertyOf rdf:resource="#isFunctionalFeatureOf"/>
  <rdfs:range rdf:resource="#Service"/>
  <rdfs:domain rdf:resource="#NetworkProtocol"/>
  <owl:inverseOf rdf:resource="#hasNetworkProtocol"/>
 </owl:ObjectProperty>
```

```
<owl:FunctionalProperty rdf:about="#hasProxyStructure">
  <rdfs:range rdf:resource="#ProxyStructure"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:subPropertyOf rdf:resource="#hasNonFunctionalFeatureOf"/>
  <owl:inverseOf rdf:resource="#isProxyStructureOf"/>
  <rdfs:domain rdf:resource="#Service"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="hasDescription">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#hasServiceProtocol">
  <rdfs:range rdf:resource="#ServiceProtocol"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#Service"/>
  <owl:inverseOf rdf:resource="#isServiceProtocolOf"/>
  <rdfs:subPropertyOf rdf:resource="#hasFunctionalFeature"/>
</owl:FunctionalProperty>
<Registry rdf:ID="DiscoveryAgent">
  <owl:differentFrom>
    <Registry rdf:ID="SLM">
      <owl:differentFrom>
        <Registry rdf:ID="Reggie">
          <owl:differentFrom rdf:resource="#DiscoveryAgent"/>
          <owl:differentFrom rdf:resource="#SLM"/>
        </Registry>
      </owl:differentFrom>
      <owl:differentFrom rdf:resource="#DiscoveryAgent"/>
    </Registry>
  </owl:differentFrom>
  <owl:differentFrom rdf:resource="#Reggie"/>
</Registry>
<OperatingSystem rdf:ID="Macintosh">
  <isOperatingSystemOf>
    <UPnPService rdf:ID="PrinterFile">
      <hasNetworkProtocol>
        <NetworkProtocol rdf:ID="TCP_IP">
          <isNetworkProtocolOf>
            <JiniService rdf:ID="FileClassifier">
              <hasProxyStructure>
                <ProxyStructure rdf:ID="Proxy">
                  <isProxyStructureOf>
                    <Service rdf:ID="ServiceQuery">
                      <hasProxyStructure rdf:resource="#Proxy"/>
                      <hasProgrammingLanguage>
                        <ProgrammingLanguage rdf:ID="Java">
                          <isProgrammingLanguageOf>
                            <BluetoothService rdf:ID="MobilePrinter">
                              <hasOperatingSystem>
                                <OperatingSystem rdf:ID="Unix">
                                  <isOperatingSystemOf rdf:resource="#MobilePrinter"/>
                                  <isOperatingSystemOf>
                                    <SLPService rdf:ID="NearestPrinter">
                                      <hasRegistry rdf:resource="#SLM"/>
                                      <hasOperatingSystem rdf:resource="#Macintosh"/>
                                      <hasNetworkProtocol rdf:resource="#TCP_IP"/>
                                      <hasOperatingSystem>
                                        <OperatingSystem rdf:ID="Windows">
                                          <isOperatingSystemOf rdf:resource="#MobilePrinter"/>
                                          <isOperatingSystemOf rdf:resource="#FileClassifier"/>
                                          <isOperatingSystemOf rdf:resource="#PrinterFile"/>
                                          <isOperatingSystemOf rdf:resource="#ServiceQuery"/>
                                          <isOperatingSystemOf>
                                            <UPnPService rdf:ID="DigitalMouse">
                                              <hasSecurity>
                                                <Security rdf:ID="IPSec">
                                                  <isSecurityOf rdf:resource="#DigitalMouse"/>
                                                </Security>
                                              </hasSecurity>
                                              <hasServiceProtocol>
```

```
                                <UPnP rdf:ID="UPnP_Protocol">
                                 <isServiceProtocolOf rdf:resource="#PrinterFile"/>
                                 <isServiceProtocolOf rdf:resource="#DigitalMouse"/>
                                </UPnP>
                               </hasServiceProtocol>
                               <hasProxyStructure rdf:resource="#Proxy"/>
                               <hasOperatingSystem rdf:resource="#Windows"/>
                               <hasNetworkProtocol rdf:resource="#TCP_IP"/>
                               <hasProgrammingLanguage>
                                <ProgrammingLanguage rdf:ID="CPP">
                                 <isProgrammingLanguageOf rdf:resource="#DigitalMouse"/>
                                 <isProgrammingLanguageOf>
                                  <BluetoothService rdf:ID="FileTransfer">
                                   <hasNetworkProtocol rdf:resource="#TCP_IP"/>
                                   <hasProxyStructure>
                                    <ProxyStructure rdf:ID="NonProxy">
                                     <isProxyStructureOf rdf:resource="#PrinterFile"/>
                                     <isProxyStructureOf rdf:resource="#FileTransfer"/>
                                     <isProxyStructureOf>
                                      <SalutationService rdf:ID="NearestFriend">
<hasServiceProtocol>
 <Salutation rdf:ID="Salutation_Protocol">
  <isServiceProtocolOf rdf:resource="#NearestFriend"/>
 </Salutation>
</hasServiceProtocol>
<hasProgrammingLanguage rdf:resource="#CPP"/>
<hasRegistry rdf:resource="#DiscoveryAgent"/>
<hasSecurity>
 <Security rdf:ID="Authentication">
  <isSecurityOf rdf:resource="#NearestFriend"/>
 </Security>
</hasSecurity>
<hasProxyStructure rdf:resource="#NonProxy"/>
<hasNetworkProtocol rdf:resource="#TCP_IP"/>
<hasOperatingSystem rdf:resource="#Windows"/>          </SalutationService>
                                     </isProxyStructureOf>
                                    </ProxyStructure>
                                   </hasProxyStructure>
                                   <hasOperatingSystem rdf:resource="#Macintosh"/>
                                   <hasProgrammingLanguage rdf:resource="#CPP"/>
                                   <hasServiceProtocol>
                                    <Bluetooth rdf:ID="Bluetooth_Protocol">
                                     <isServiceProtocolOf rdf:resource="#MobilePrinter"/>
                                     <isServiceProtocolOf rdf:resource="#FileTransfer"/>
                                    </Bluetooth>
                                   </hasServiceProtocol>
                                  </BluetoothService>
                                 </isProgrammingLanguageOf>
                                 <isProgrammingLanguageOf rdf:resource="#NearestFriend"/>
                                 <isProgrammingLanguageOf rdf:resource="#NearestPrinter"/>
                                </ProgrammingLanguage>
                               </hasProgrammingLanguage>
                               <hasOperatingSystem rdf:resource="#Unix"/>
                              </UPnPService>
                             </isOperatingSystemOf>
                             <isOperatingSystemOf rdf:resource="#NearestPrinter"/>
                             <isOperatingSystemOf rdf:resource="#NearestFriend"/>
                            </OperatingSystem>
                           </hasOperatingSystem>
                           <hasProxyStructure rdf:resource="#Proxy"/>
                           <hasOperatingSystem rdf:resource="#Unix"/>
                           <hasProgrammingLanguage rdf:resource="#CPP"/>
                           <hasServiceProtocol>
                            <SLP rdf:ID="SLP_Protocol">
                             <isServiceProtocolOf rdf:resource="#NearestPrinter"/>
                            </SLP>
                           </hasServiceProtocol>
                          </SLPService>
                         </isOperatingSystemOf>
                         <isOperatingSystemOf rdf:resource="#DigitalMouse"/>
                        </OperatingSystem>
```

```
                    </hasOperatingSystem>
                    <hasServiceProtocol rdf:resource="#Bluetooth_Protocol"/>
                    <hasProgrammingLanguage rdf:resource="#Java"/>
                    <hasNetworkProtocol>
                     <NetworkProtocol rdf:ID="UDP">
                      <isNetworkProtocolOf rdf:resource="#MobilePrinter"/>
                     </NetworkProtocol>
                    </hasNetworkProtocol>
                    <hasProxyStructure rdf:resource="#Proxy"/>
                    <hasOperatingSystem>
                     <OperatingSystem rdf:ID="Linux">
                      <isOperatingSystemOf rdf:resource="#MobilePrinter"/>
                      <isOperatingSystemOf>
                       <JiniService rdf:ID="Printer">
                        <hasProxyStructure rdf:resource="#Proxy"/>
                        <hasRegistry rdf:resource="#Reggie"/>
                        <hasOperatingSystem rdf:resource="#Macintosh"/>
                        <hasProgrammingLanguage rdf:resource="#Java"/>
                        <hasNetworkProtocol rdf:resource="#TCP_IP"/>
                        <hasServiceProtocol>
                         <Jini rdf:ID="Jini_Protocol">
                          <isServiceProtocolOf rdf:resource="#FileClassifier"/>
                          <isServiceProtocolOf rdf:resource="#Printer"/>
                         </Jini>
                        </hasServiceProtocol>
                        <hasOperatingSystem rdf:resource="#Linux"/>
                       </JiniService>
                      </isOperatingSystemOf>
                     </OperatingSystem>
                    </hasOperatingSystem>
                    <hasOperatingSystem rdf:resource="#Windows"/>
                   </BluetoothService>
                  </isProgrammingLanguageOf>
                  <isProgrammingLanguageOf rdf:resource="#Printer"/>
                  <isProgrammingLanguageOf rdf:resource="#FileClassifier"/>
                  <isProgrammingLanguageOf rdf:resource="#ServiceQuery"/>
                 </ProgrammingLanguage>
                </hasProgrammingLanguage>
                <hasOperatingSystem rdf:resource="#Windows"/>
                <hasSecurity>
                 <Security rdf:ID="JavaBased">
                  <isSecurityOf rdf:resource="#ServiceQuery"/>
                 </Security>
                </hasSecurity>
                <hasNetworkProtocol rdf:resource="#TCP_IP"/>
               </Service>
              </isProxyStructureOf>
              <isProxyStructureOf rdf:resource="#DigitalMouse"/>
              <isProxyStructureOf rdf:resource="#NearestPrinter"/>
              <isProxyStructureOf rdf:resource="#MobilePrinter"/>
              <isProxyStructureOf rdf:resource="#FileClassifier"/>
              <isProxyStructureOf rdf:resource="#Printer"/>
             </ProxyStructure>
            </hasProxyStructure>
            <hasOperatingSystem rdf:resource="#Macintosh"/>
            <hasOperatingSystem rdf:resource="#Windows"/>
            <hasServiceProtocol rdf:resource="#Jini_Protocol"/>
            <hasNetworkProtocol rdf:resource="#TCP_IP"/>
            <hasProgrammingLanguage rdf:resource="#Java"/>
            <hasRegistry rdf:resource="#Reggie"/>
           </JiniService>
          </isNetworkProtocolOf>
          <isNetworkProtocolOf rdf:resource="#DigitalMouse"/>
          <isNetworkProtocolOf rdf:resource="#NearestFriend"/>
          <isNetworkProtocolOf rdf:resource="#ServiceQuery"/>
          <isNetworkProtocolOf rdf:resource="#FileTransfer"/>
          <isNetworkProtocolOf rdf:resource="#PrinterFile"/>
          <isNetworkProtocolOf rdf:resource="#NearestPrinter"/>
          <isNetworkProtocolOf rdf:resource="#Printer"/>
         </NetworkProtocol>
        </hasNetworkProtocol>
```

```
      <hasOperatingSystem rdf:resource="#Windows"/>
      <hasProgrammingLanguage>
        <ProgrammingLanguage rdf:ID="C">
          <isProgrammingLanguageOf rdf:resource="#PrinterFile"/>
        </ProgrammingLanguage>
      </hasProgrammingLanguage>
      <hasOperatingSystem rdf:resource="#Macintosh"/>
      <hasServiceProtocol rdf:resource="#UPnP_Protocol"/>
      <hasProxyStructure rdf:resource="#NonProxy"/>
    </UPnPService>
   </isOperatingSystemOf>
   <isOperatingSystemOf rdf:resource="#Printer"/>
   <isOperatingSystemOf rdf:resource="#FileTransfer"/>
   <isOperatingSystemOf rdf:resource="#NearestPrinter"/>
   <isOperatingSystemOf rdf:resource="#FileClassifier"/>
  </OperatingSystem>
</rdf:RDF>
```

11.2.2 Οντολογία Προτεινόμενων Υπηρεσιών

Αρχείο: SuggestedOntology.owl

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns="http://www.owl-ontologies.com/suggestion.owl#"
   xmlns:service="http://www.owl-ontologies.com/service.owl#"
  xml:base="http://www.owl-ontologies.com/suggestion.owl">
  <owl:Ontology rdf:about="">
   <owl:imports rdf:resource="http://www.owl-ontologies.com/service.owl"/>
  </owl:Ontology>
  <owl:TransitiveProperty rdf:ID="suggests">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
   <rdfs:domain rdf:resource="http://www.owl-ontologies.com/service.owl#Service"/>
   <rdfs:range rdf:resource="http://www.owl-ontologies.com/service.owl#Service"/>
   <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
   >Temporal logic</rdfs:comment>
  </owl:TransitiveProperty>
  <rdf:Description rdf:about="http://www.owl-ontologies.com/service.owl#NearestPrinter">
   <suggests>
    <rdf:Description rdf:about="http://www.owl-ontologies.com/service.owl#FileTransfer">
     <suggests rdf:resource="http://www.owl-ontologies.com/service.owl#FileClassifier"/>
    </rdf:Description>
   </suggests>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.owl-ontologies.com/service.owl#Printer">
   <suggests rdf:resource="http://www.owl-ontologies.com/service.owl#FileClassifier"/>
  </rdf:Description>
</rdf:RDF>
```

12 ΟΡΟΛΟΓΙΑ

Ξενόγλωσσος Όρος	Ελληνικός Όρος
Agent	Πράκτορας
Axiom	Αξίωμα
Concept	Κλάση
Concept Hierarchy	Ιεραρχία κλάσεων
Context	Πληροφορία πλαισίου
Context OWL	OWL Πλαισίο
Data Repository	Αποθήκη Δεδομένων
Description Logics	Περιγραφική Λογική
Disjoint sets	Ξένα σύνολα
Domain	Πεδίο Ορισμού
Entity	Οντότητα
First Order Logic	Λογική Πρώτης Τάξης
Frame logic	Λογική Πλαισίου
Fuzzy Theory	Θεωρία Ασάφειας
Hyper-link	Υπερ-σύνδεσμος
Hypertext	Υπερκείμενο
Inference	Συμπερασμός
Inference Rules	Κανόνες Συμπερασμού
Instance	Στιγμιότυπο
Interoperability	Διαλειτουργικότητα
Key term	Όρος-κλειδί
Knowledge Representation	Αναπαράσταση Γνώσης
Link	Σύνδεσμος
Local Area Network (LAN)	Τοπικό Δίκτυο
Location based Services	Υπηρεσίες Βασισμένες στη Γεωγραφική Θέση
Object	Αντικείμενο
Ontology	Οντολογία
Pervasive Computing	Διάχυτος Υπολογισμός
Portal	Πύλη
Property	Συσχέτιση

Property Types	Τύποι Συσχετίσεων
Proxy	Αντιπρόσωπος
Range	Πεδίο Τιμών
Reasoner	Μηχανισμός Συμπερασμού
Reasoning	Συμπερασμός
Semantic	Σημασιολογία
Semantic Web	Σημασιολογικός Ιστός
Service Discovery	Εύρεση Υπηρεσιών
Service Profile	Προφίλ Υπηρεσίας
Service Provider	Πάροχος Υπηρεσίας
Similarity	Ομοιότητα
Sub-class	Υπο-κλάση
Sub-property	Υπο-συσχέτιση
Taxonomy	Ταξινομία
Textual	Λεξικογραφικός
Web Browser	Φυλλομετρητής Ιστού
Web Browsing	Φυλλομέτρηση Ιστού
Web-based Services	Βασισμένες στον Ιστό υπηρεσίες
Wireless LAN	Ασύρματο Τοπικό Δίκτυο
World Wide Web (WWW)	Παγκόσμιος Ιστός

13 ΑΡΚΤΙΚΟΛΕΞΑ

API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
DA	Directory Agent
DAML	DARPA Agent Markup Language
DARPA	Defence Advanced Research Project Agency
DCOM	Distributed Component Object Model
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System (or Service or Server)
EJB	Enterprise Java Bean
GENA	General Event Notification Architecture
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IETF	Internet Engineering Task Force
IP	Internet Protocol
Jini	Jini is not initial!
JNI	Java Native Interface
JVM	Java Virtual Machine
OCML	Operational Conceptual Modelling Language
OIL	Ontology Interchange Language

OML	Ontology Markup Language
OWL	Web Ontology Language
RDFS	Resource Description Framework Schema
RMI	Remote Method Invocation
SA	Service Agent
SHOE	Simple HTML Ontology Extensions
SLM	Salutation Manager
SLP	Service Location Protocol
SDP	Service Discovery Protocol
SOAP	Simple Object Access Protocol
SSDP	Simple Service Discovery Protocol
TCP	Transmission Control Protocol
UA	User Agent
UDP	User Datagram Protocol
UML	Unified Modeling Language
UPnP	Universal Plug and Play
URL	Uniform Resource Locator
XML	Extensible Markup Language
XOL	Ontology Exchange Language
W3C	World Wide Web Consortium
WWW	World Wide Web

14 ΒΙΒΛΙΟΓΡΑΦΙΚΕΣ ΑΝΑΦΟΡΕΣ

1. Tim Berners-Lee, James Hendler, Ora Lassila, “The Semantic Web”, Scientific American, May 2001
2. http://www.publictransport.co.uk/trains
3. *Αριστοτέλης: οντολογία, γνωσιοθεωρία, ηθική, πολιτική φιλοσοφία. Τριάντα δύο ομόκεντρες μελέτες,* επιμέλεια: Δ.Ζ. Ανδριόπουλος, Ν.Σ. Μελισσίδης, Παπαδήμας, Αθήνα, 1997
4. T. R. Gruber, “A translation approach to portable ontologies. *Knowledge Acquisition*”, 5(2):199-220, 1993
5. Pim Borst, Hans Akkermans, “An Ontology Approach to Product Disassembly”, EKAW 1997: 33-48
6. D. L. McGuinness, Frank van Harmelen, "OWL Web Ontology Language Overview", W3C Recommendation February 10, 2004
7. KAON, AIFB/FZI, University of Karlsruhe, http://kaon.semanticweb.org/
8. OilEd, University of Manchester, http://oiled.man.ac.uk/
9. Ontolingua, KSL (Stanford University), http://www-ksl.stanford.edu
10. OntoSaurus, ISI (USA), http://www.isi.edu/isd/ontosaurus.html
11. OntoEdit, University of Karlsruhe, http://ontoserver.aifb.unikarlsruhe.de/ontoedit/
12. Protégé 2000, SMI (Stanford University), http://protege.stanford.edu/
13. WebOnto, KMI (Open University), http://kmi.open.ac.uk/projects/webonto/
14. WebODE, UPM, http://webode.dia.fi.upm.es/webODE/
15. Alexander Maedche and Valentin Zacharias, “Clustering Ontology-based Metadata in the Semantic Web”, PKDD 2002: 348-360
16. Dwi H. Widyantoro and John Yen, “A Fuzzy Similarity Approach in Text Classification Task”, in Proceedings of Ninth IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2000.
17. A. Zadeh Fuzzy sets, Inf. Control 8, 338-353, 1965.

18. Rocchio, J. (1971), “Relevance Feedback Information Retrieval”, In Gerald Salton(Ed.), The Smart Retrieval System – Experiments in Automated Document Processing, 313-323. Prentice-Hall, Englewood Cliffs, NJ
19. Golden G. Richard III, “Service and Device Discovery, Protocols and Programming”, McGraw-Hill, 2002
20. C. Bettstetter and C. Renner, "A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol", Proceedings of the Sixth EUNICE Open European Summer School: Innovative Internet Applications, 2000
21. http://www.jetsend.hp.com
22. http://www.salutation.org
23. Universal Plug and Play Forum. Universal Plug and Play Device Architecture. Version 0.91, March 2000
24. Bluetooth Specification Part E. Service Discovery Protocol (SDP). http://www.bluetooth.com, November 1999
25. http://www.jini.org
26. Jim Waldo, “The Jini Architecture for Network-Centric Computing”, Communications of the ACM, 1999
27. Franz Baader, Ciego Calvanese, Deborah McGuiness, Daniele Nardi, Peter Patel-Schneider, “The Description Logic Handbook, Theory, Implementation and Application”, Cambridge University Press, 2002
28. Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe, “A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0”, The University of Manchester, 2004
29. Patrick Doherty, Witold Lukaszewicz, Andrzej Szalas, “Efficient Reasoning Using the Local Closed-World Assumption”, 1904, pp. 49–58, 2000. Springer-Verlag Berlin Heidelberg 2000
30. Steffen Hölldobler, Hans-Peter Störr, and Tran Dinh Khang, “The Fuzzy Description Logic ALCFH with Hedge Algebras as Concept Modifiers”, Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol.7, No.3 pp. 294-305, 2003

31. Alexander Maedche and Steffen Staab, “Measuring Similarity between Ontologies”, In Proc. Of the European Conference on Knowledge Acquisition and Management - EKAW-2002. Madrid, Spain, 2002

32. Xiaomeng Su and Atle Gulla, “Semantic Enrichment for Ontology Mapping”, Natural Language Text Understanding, 217-228, 2004

33. Dipanjan Chakraborty, Filip Perich, Sasikanth Avancha, and Anupam Joshi, "DReggie: Semantic Service Discovery for M-Commerce Applications", Workshop on Reliable and Secure Applications in Mobile Environment, In Conjunction with 20th Symposium on Reliable Distributed Systems (SRDS), October 2001

34. Samuel Kasper and Lorentz Buhrer, “Jini Discovers Bluetooth”, ETH, Semester Thesis SA-2002.30, Summer 2002.