ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Περιβάλλον πλαίσιο για τον υπομνηματισμό πόρων αρχιτεκτονικής REST

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Χαραλάμπους Τηλέμαχου

Επιβλέπων: Κώστας Κοντογιάννης
Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Περιβάλλον πλαίσιο για τον υπομνηματισμό πόρων αρχιτεκτονικής REST

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Χαραλάμπους Τηλέμαχου

Επιβλέπων: Κώστας Κοντογιάννης
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14η Ιουλίου 2017.

…….	…….	…….
Κώστας Κοντογιάννης	Ανδρέας-Γεώργιος Σταφυλοπάτης	Γιώργος Στάμου
Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Αν. Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Copyright @ Χαραλάμπους Τηλέμαχος, 2017. Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Ο σκοπός αυτής της διπλωματικής εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός συστήματος REST που θα επιτρέπει την προσαρμογή του ίδιου του συστήματος σε άλλα, εξωτερικά, συστήματα REST, καθώς και την προσθήκη affordances στους εκάστοτε REST πόρους. Η εκθετική αύξηση της βάσης των χρηστών και κατά συνέπεια των διαφορετικών προτιμήσεων έχει καταστήσει αναγκαία την εξατομίκευση των προσφερόμενων υπηρεσιών και την ευκολία πρόσβασης σε αυτές. Για αυτό το λόγο είναι σημαντική η ανάπτυξη συστημάτων τα οποία θα πρέπει να μπορούν να προσφέρουν εξατομικευμένες και εύκολες ως προς την πρόσβαση υπηρεσίες. Η συγκεκριμένη διπλωματική αποτελεί μία λύση στο πρόβλημα της εξατομίκευσης, παρουσιάζοντας ένα τέτοιο σύστημα το οποίο μπορεί να προσαρμόζεται πάνω σε άλλα συστήματα που προσφέρουν RESTful υπηρεσίες και να παρέχει τις ίδιες υπηρεσίες με εξατομικευμένο τρόπο. Η προσέγγιση αυτή είναι μία από όλες τις δυνατές προσεγγίσεις με κύριο πλεονέκτημα το ότι μπορεί να προσαρμόζεται ταυτόχρονα σε πολλές υπηρεσίες, χωρίς να χρειάζεται ο επανασχεδιασμός και η υλοποίηση εκ νέου του συστήματος στο οποίο γίνεται η προσαρμογή. Αυτό καθιστά την υλοποίηση πιο εύκολη, αφού υπάρχει μικρή σύζευξη ανάμεσα στο σύστημα που κάνει την εξατομίκευση (προσθήκη affordances) και σε αυτό που προσφέρει τις υπηρεσίες, καθώς επίσης επιτρέπει την επεκτασιμότητα σε επίπεδο λογισμικού. Για την πρακτική αντιμετώπιση του προβλήματος χρησιμοποιήσαμε δέντρα στόχων (goal models) για τη μοντελοποίηση των προϋποθέσεων που θα πρέπει να ισχύουν για κάθε διαφορετικό affordance. Πρέπει να αναφερθεί επίσης ότι για κάθε ερώτημα σε μια συγκεκριμένη υπηρεσία δεν είναι αναγκαία η επίλυση όλων των δέντρων, αλλά μόνο αυτών που αφορούν το συγκεκριμένο ερώτημα, καθιστώντας τη συγκεκριμένη λύση όχι μόνο επεκτάσιμη ως προς το λογισμικό, αλλά και ως προς το χρόνο. Η βάση της υλοποίησης έγινε χρησιμοποιώντας το περιβάλλον λογισμικού Spring [12] σε Java, δίνοντας ιδιαίτερη έμφαση στην αρχιτεκτονική και την ασφάλεια, δεδομένου του ότι το σύστημα πρέπει να προσαρμόζεται σε εξωτερικά συστήματα. Αρχικά ορίστηκε η αρχιτεκτονική του συστήματος, στη συνέχεια δημιουργήθηκε το framework το οποίο είναι η “καρδιά” της υλοποίησης και στη συνέχεια υλοποιήθηκε το REST σύστημα, το οποίο χρησιμοποιεί το προαναφερθέν framework για να επιτύχει τον τελικό στόχο. Το θετικό με αυτήν την προσέγγιση είναι ότι όχι μόνο μπορεί το πρωτότυπο σύστημα να επεκταθεί από τρίτους, αλλά και το ίδιο το framework να χρησιμοποιηθεί/επεκταθεί για να δημιουργηθεί κάτι διαφορετικό το οποίο έχει ως πυρήνα τα affordances.

Λέξεις κλειδιά: goal models, satisfiability, affordance, distributed affordances, software architecture, design patterns, REST, RESTful Web Services

Abstract

The main goal of this diploma thesis is to design and implement a RESTful Web Service that adapts to other, external, RESTful Web Services, as well as annotate the resources with affordances. The exponential growth of users on the Internet and in turn their different preferences has created the need of personalization of the various provided services as well as make easier for them to be accessed. For this particular reason it is important to develop systems that are able to offer personalized and easily accessible services. This diploma thesis poses a solution to the problem of personalization offering a sample implementation of a such system that can be adapted to other RESTful Web Services and offer the same service with extra affordances. This implementation is only one of the various possible ones with the main advantage of the adaptation to various services simultaneously without the need to redesign and reimplement the system. This makes the very implementation of the system easier, since there is little coupling between the system that annotates the resources with affordances and the system that actually provides the services, yielding benefits such as scalability and expandability of the software. To deal with personalization problem, goal models were used to model the requirements for each affordance. However it is critical to mention that for each request to a specific RESTful Web Service not all goal models must be evaluated, but only a subset of them that are relative to that specific request. The main tool that was used to implement this system is the Spring Framework [12] using the Java programming language, with emphasis in software architecture and security, provided that the system must be adaptable to other external systems. Initially the architecture of this system was designed and then a framework, the heart, of this implementation was created. In the end the RESTful Web Service was implemented which used the aforementioned framework. The reason behind this approach is that not only the system can be adapted/extended, but also the framework itself to create software that may be totally different than a RESTful Web Service, with the need to make decisions using goal models and affordances.

Keywords: goal models, satisfiability, affordance, distributed affordances, software architecture, design patterns, REST, RESTful Web Services

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντά μου, Αναπληρωτή Καθηγητή Ε.Μ.Π. κ. Κώστα Κοντογιάννη για το ενδιαφέρον και την υπομονή που έδειξε κατά τη διάρκεια της διπλωματικής μου εργασίας. Μου μετέδωσε πολλές και χρήσιμες γνώσεις γύρω από τον τομέα της Τεχνολογίας και Αρχιτεκτονικής Λογισμικού, έναν τομέα που έχει ιδιαίτερο ενδιαφέρον αν αναλογιστεί κανείς την αυξανόμενη ανάγκη για σωστή δόμηση του λογισμικού στις μέρες μας. Η συνεργασία μας ήταν μια μοναδική εμπειρία και τον ευχαριστώ θερμά που με εμπιστεύτηκε με αυτή τη διπλωματική.

Θα ήθελα επίσης, να ευχαριστήσω τους συμφοιτητές και τους φίλους μου, που μέσα από όλες τις στιγμές που περάσαμε μαζί, όμορφες και άσχημες, με βοήθησαν να εξελιχθώ όχι μόνο σαν μηχανικός, αλλά και σαν άνθρωπος γενικότερα.

Τέλος, το πιο μεγάλο ευχαριστώ θα ήθελα να το πω στην οικογένειά μου, τη μητέρα μου Μαρία και τον αδερφό μου Νίκο, που όλα αυτά τα χρόνια με στήριξαν και με βοήθησαν ώστε να πετύχω τους στόχους μου.

Τηλέμαχος

Πίνακας περιεχομένων

Πίνακας Σχημάτων

Κεφάλαιο 1: Εισαγωγή

Η εισαγωγή αυτή ξεκινά με το κίνητρο που μας οδήγησε να ασχοληθούμε με αυτήν την εργασία και συνεχίζει με την περιγραφή του προβλήματος και τα επιμέρους προβλήματα που συναντήσαμε. Αμέσως μετά γίνεται αναφορά στη συνεισφορά της εργασίας, ενώ στο τέλος παρουσιάζεται το πως έχει οργανωθεί το κείμενο που ακολουθεί.

1.1 Κίνητρο

Τα τελευταία χρόνια έχει παρατηρηθεί μια ραγδαία αύξηση στις ηλεκτρονικές υπηρεσίες. Πάρα πολλά από αυτά που κάνει κάποιος καθημερινά έχουν αρχίσει σταδιακά να παίρνουν ηλεκτρονική μορφή, άλλες φορές ως προαιρετικά και άλλες υποχρεωτικά. Παραδείγματα τέτοιων υπηρεσιών είναι το ηλεκτρονικό ταχυδρομείο, το ηλεκτρονικό εμπόριο και οι ηλεκτρονικές συναλλαγές. Ο πιο δημοφιλής τρόπος για να υλοποιηθεί μια ηλεκτρονική υπηρεσία είναι ακολουθώντας την αρχιτεκτονική REST. Ο κυριότερος λόγος πίσω από αυτό είναι ότι το REST είναι αποδοτικό, επεκτάσιμο, απλό και βασίζεται πάνω σε ήδη υπάρχουσες τεχνολογίες, όπως το HTTP.

Συνέπεια του παραπάνω είναι η ανάγκη στο να γίνουν οι RESTful υπηρεσίες πιο απλές και εξατομικευμένες για το χρήστη. Κάτι τέτοιο μπορεί να επιτευχθεί χρησιμοποιώντας έννοιες όπως distributed affordances και HATEOAS (Hypermedia As The Engine Of Application State), ώστε να βοηθήσει τον χρήστη όχι μόνο να πλοηγηθεί σε μια τέτοια υπηρεσία, αλλά και να έχει μια πιο ολοκληρωμένη και ουσιαστική εμπειρία.

Πιο συγκεκριμένα, σήμερα υπάρχουν χιλιάδες RESTful υπηρεσίες οι οποίες, παρά το γεγονός ότι ακολουθούν μια REST αρχιτεκτονική δεν ενστερνίζονται την ιδέα του affordance και του HATEOAS. Κάτι τέτοιο μπορεί να φαίνεται λογικό, μιας και αυτά αποτελούν επιπλέον στοιχεία του REST, αλλά δεν παύουν να είναι κύρια κομμάτια του. Για αυτό το λόγο υπάρχει η ανάγκη της δημιουργίας ενός περιβάλλοντος πλαισίου το οποίο να μπορεί να παρέχει εργαλεία για την δημιουργία και χρήση affordances, ακόμα και πάνω σε RESTful υπηρεσίες που δεν ακολουθούν αυτή τη λογική. Η συγκεκριμένη εργασία παρουσιάζει μια προσέγγιση για το πως θα μπορούσε να επιτευχθεί κάτι τέτοιο.

1.2 Περιγραφή προβλήματος

Ο στόχος αυτής της διπλωματικής εργασίας είναι η υλοποίηση ενός περιβάλλοντος πλαισίου, καθώς και ενός proxy διακομιστή που θα χρησιμοποιεί αυτό το πλαίσιο, για να προσαρμόζεται

πάνω σε REST συστήματα που παρέχουν υπηρεσίες προσθέτοντας affordances στα resources και αλλάζοντάς τα με σκοπό την εξατομίκευσή τους. Το περιβάλλον πλάισιο θα παρέχει τα κατάλληλα εργαλεία που χρειάζονται για την επιλογή των κατάλληλων affordances με βάση τα συμφραζόμενα που προκύπτουν από την κάθε περίπτωση χρήσης. Είναι σημαντικό, το συγκεκριμένο πλαίσιο να υλοποιηθεί με τέτοιον τρόπο ώστε να είναι χρησιμοποιήσιμο, όχι μόνο από τη συγκεκριμένη υλοποίηση που αφορά υπηρεσίες REST, αλλά και από άλλες υλοποιήσεις που χειρίζονται affordances. Πιο συγκεκριμένα, ο διακομιστής ο οποίος θα χρησιμοποιεί αυτό το περιβάλλον πλαίσιο θα προωθεί τα REST ερωτήματα στην εκάστοτε υπηρεσία και θα προσαρμόζει την απάντηση που πήρε προσθέτοντάς της affordances που αφορούν το χρήστη που έκανε το ερώτημα, χρησιμοποιώντας δέντρα στόχων.

Με βάση όλα αυτά προκύπτουν τα παρακάτω επιμέρους ερωτήματα και προβλήματα που πρέπει να επιλυθούν κατά τη σχεδίαση ενός τέτοιου συστήματος:

- Πως θα πρέπει να γίνει η σχεδίαση του περιβάλλοντος πλαισίου ώστε να είναι όσο το δυνατόν περισσότερο αφηρημένο και προσαρμόσιμο σε οποιαδήποτε υλοποίηση αφορά affordances;

- Με ποιο τρόπο επιλέγονται τα affordances που θα χρησιμοποιηθούν για να εμπλουτίσουν την απάντηση μιας REST υπηρεσίας.

- Με ποιο τρόπο επιλέγονται τα συμφραζόμενα τα οποία αφορούν ένα affordance;

- Αν υπάρχουν παραπάνω από ένα affordances που αφορούν μια συγκεκριμένη απάντηση με ποιο τρόπο επιλέγεται ποιο θα είναι το πιο κατάλληλο για χρήση;

- Αν υπάρχουν παραπάνω από ένα affordances που αφορούν μια συγκεκριμένη απάντηση με ποιο τρόπο επιλέγεται αν στην απάντηση θα προστεθούν ένα ή περισσότερα affordances;

- Πως μπορεί να δοθεί το σύνολο όλων των προϋποθέσεων για να ισχύει ένα affordance με τον πιο απλό, αποδοτικό και γενικό τρόπο;

- Πως πρέπει να σχεδιαστεί το σύστημα για να είναι ασφαλές, δεδομένου ότι προσαρμόζεται πάνω σε εξωτερικές υπηρεσίες;

- Πως θα γίνει η αντιστοίχηση μεταξύ των εσωτερικών συνδέσμων του διακομιστή και των εξωτερικών συνδέσμων των εκάστοτε υπηρεσιών, ώστε να μπορεί επιτυχώς ο διακομιστής να προσαρμόζεται ταυτόχρονα σε πολλές υπηρεσίες; Και κατά συνέπεια πως θα γίνει υπηρεσίες οι οποίες έχουν διαφορετικό τρόπο διεπαφής να ενοποιηθούν ομοιόμορφα κάτω από μια κεντρική υπηρεσία;

1.3 Συνεισφορά της Διπλωματικής Εργασίας

Η παρούσα διπλωματική παρουσιάζει το σχεδιασμό και την υλοποίηση ενός περιβάλλοντος πλαισίου λογισμικού που καλείται να επιλύσει το πρόβλημα της εξατομίκευσης υπηρεσιών REST προσθέτοντας affordances σε τυχόν απαντήσεις. Η συνεισφορά της συγκεκριμένης διπλωματικής έχει βάση τα εξής σημεία:

- Η υλοποίηση έχει χωριστεί στην υλοποίηση του περιβάλλοντος πλαισίου και στην υλοποίηση της REST υπηρεσίας που χρησιμοποιεί το πλαίσιο και προσαρμόζεται σε άλλες υπηρεσίες. Το περιβάλλον πλαίσιο είναι όσο το δυνατόν πιο γενικό και αφηρημένο και προσφέρει εργαλεία που αφορούν τη δημιουργία affordances και των προϋποθέσεων τους, την κατάλληλη επιλογή του καταλληλότερου και την επιλογή ενός ή περισσότερων από αυτά. Με τον τρόπο αυτό μπορεί να γίνει χρήση και επέκταση όχι μόνο της συγκεκριμένης υλοποίησης που αφορά το REST, αλλά και του περιβάλλοντος πλαισίου για να δημιουργηθεί κάτι τελείως διαφορετικό το οποίο έχει ανάγκη τη λήψη αποφάσεων και την εξατομίκευση μιας κατάστασης ή ενός συστήματος.

- Την επέκταση του Μοντέλου Δέντρων Στόχων (goal models) ώστε να μπορεί να χρησιμοποιηθεί για τη μοντελοποίηση των affordances και των απαιτήσεων που χρειάζονται για να ικανοποιηθούν. Πιο συγκεκριμένα σε κάθε δέντρο αντιστοιχεί ένα συγκεκριμένο affordance, ενώ τα φύλλα του δέντρου αντιπροσωπεύουν καταστάσεις και συνθήκες που πρέπει να ισχύουν ώστε να επαληθευθεί το δέντρο. Οι καταστάσεις αυτές στο περιβάλλον πλαίσιο είναι γενικές και δύνανται να είναι οτιδήποτε, παρ' ΄όλα αυτά στην υλοποίηση της REST υπηρεσίας μπορεί να είναι στοιχεία της κατάστασης του χρήστη, ή στοιχεία που αφορούν τους πόρους/απάντηση της εκάστοτε υπηρεσίας.

- Την αντιστοίχηση συνδέσμων της υλοποιημένης υπηρεσίας με εξωτερικές υπηρεσίες. Πιο συγκεκριμένα, η συγκεκριμένη υλοποίηση επιτρέπει την ταυτόχρονη προσαρμογή σε πολλές διαφορετικές RESTful υπηρεσίες. Αυτό έχει γίνει με τη χρήση λογισμικού αντιστοίχησης το οποίο μεταφράζει συνδέσμους οι οποίοι μπορεί να έχουν συγκεκριμένες παραμέτρους για οποιαδήποτε http μέθοδο (GET, POST, PUT, DELETE κλπ) στους αντίστοιχους συνδέσμους της εξωτερικής υπηρεσίας ώστε να μπορεί να γίνει σωστά η "κατανάλωση" αυτής της υπηρεσίας. Επίσης σημαντικό είναι να αναφερθεί πως οι εξωτερικές υπηρεσίες δεν είναι απαραίτητο να έχουν ομοιόμορφη διεπαφή, παρ' όλα 'υτά το υλοποιηθέν λογισμικό έχει φροντίσει για κάτι τέτοιο.

1.4 Οργάνωση Κειμένου

Η οργάνωση του κειμένου της συγκεκριμένης εργασίας είναι:

- Κεφάλαιο 1: Η συγκεκριμένη εισαγωγή

- Κεφάλαιο2: Παρουσιάζονται όλες οι σχετικές εργασίες που βοήθησαν στην ολοκλήρωση αυτής εδώ της εργασίας. Αναφέρονται έννοιες σχετικές με τη μοντελοποίηση λογισμικού, τα affordances και τα δέντρα στόχων, στα οποία βασίστηκε η μοντελοποίηση της ικανοποιησιμότητας των affordances. Επίσης γίνεται αναφορά σε διάφορες τεχνολογίες και εργαλεία που χρησιμοποιήθηκαν ή έχουν σχέση, όπως η προδιαγραφή του REST και το EMF, ένα εργαλείο μοντελοποίησης που βασίζεται στο πρόγραμμα ανάπτυξης λογισμικού eclipse.

- Κεφάλαιο 3: Παρουσιάζεται η αρχιτεκτονική της υλοποίησης συνοδευόμενη από τα απαραίτητα ψηφιδικά και ακολουθιακά διαγράμματα.

- Κεφάλαιο 4: Αναλύονται τα σχετικά εννοιολογικά μοντέλα των δέντρων στόχων και των affordances.

- Κεφάλαιο 5: Αναλύεται η συλλογιστική των δέντρων στόχων καθώς και ο τρόπος με τον οποίο έγινε ο υπολογισμός της ικανοποιησιμότητας των δέντρων αυτών στο συγκεκριμένο σύστημα.

- Κεφάλαιο 6: Παρουσιάζονται διάφορα παραδείγματα χρήσης που προκύπτουν από την εκτέλεση των βασικών λειτουργιών της παρούσας υλοποίησης, συνοδευόμενα από εικόνες, καθώς και διάφορες μετρικές του συστήματος.

- Κεφάλαιο 7: Παρουσιάζονται τα συμπεράσματα που προκύπτουν από την παρούσα εργασία, καθώς και μελλοντικές εργασίες που θα μπορούσαν να βασιστούν πάνω της.

Κεφάλαιο 2: Σχετικές εργασίες

2.1 Εισαγωγή

Το αντικείμενο αυτής της διπλωματικής εργασίας βρίσκεται σε στενή σχέση με άλλες εργασίες. Πρώτο και βασικότερο είναι η εργασία που αφορά την αρχιτεκτονική REST, αφού η υλοποίηση της εργασίας έχει γίνει με βάση αυτό. Επίσης γίνεται αναφορά στο Hydra, ένα περιβάλλον πλαίσιο που έχει στενή σχέση με το REST και κυρίως με τις αναπαραστάσεις των πόρων μια υπηρεσίας, όπως επίσης και στην έννοια των distributed affordances, κύρια ιδέα αυτής της διπλωματικής εργασίας. Ενδιαφέρον παρουσιάζουν, επίσης,ι οι αντίστοιχες εργασίες που αφορούν τη μοντελοποίηση λογισμικού, καθώς και αντίστοιχα προγράμματα που αφορούν σε αυτήν, όπως το EMF. Τέλος δεν πρέπει να γίνει παράληψη στην αναφορά στα δέντρα στόχων τα οποία χρησιμοποιήθηκαν για να μοντελοποιήσουν την ικανοποιησιμότητα και την σχετικότητα ενός affordance με έναν συγκεκριμένο πόρο μιας υπηρεσίας.

2.2 REST

Ίσως η πιο βασική εργασία που αφορά αυτή τη διπλωματική είναι αυτή που αφορά το REST [1][2]. Είναι δημιούργημα του Roy Thomas Fielding στα πλαίσια της διδακτορικής διατριβής του το 2000. Δημιουργήθηκε παράλληλα με την εργασία του πάνω στο HTTP 1.1 (1996-1999), βασισμένο στο ήδη, τότε, υπάρχων HTTP 1.0. Δικαίως αναφέρεται πρώτη στην σειρά των σχετικών εργασιών, αφού έχει παίξει καθοριστικό ρόλο στον τρόπο που δομούνται πλέον όλες οι ηλεκτρονικές υπηρεσίες.

2.2.1 Γενικά

Οι Representational State Transfer (REST) υπηρεσίες (RESTful Web Services) είναι ένα υποσύνολο των υπηρεσιών που αφορούν την επικοινωνία μεταξύ υπολογιστικών συστημάτων στο διαδίκτυο. Μία υπηρεσία που συμβαδίζει με τη λογική και τους κανόνες του REST επιτρέπει σε εξωτερικά συστήματα να αναζητήσουν ή να επεξεργαστούν το κείμενο διαδικτυακών πόρων χρησιμοποιώντας ένα ομοιόμορφο και σαφώς καθορισμένο σύνολο εντολών το οποίο έχει την ιδιότητα να μην αλλάζει την κατάσταση της υπηρεσίας (stateless).

Αρχικά οι διαδικτυακοί πόροι ήταν απλά αρχεία κειμένου τα οποία μπορούσε κάποιος να επισκεφτεί χρησιμοποιώντας μοναδικούς συνδέσμους (URL), παρά όλα αυτά με τους ρυθμούς που αναπτύσσονται τα πάντα γύρω από την τεχνολογία σήμερα, ένας διαδικτυακός πόρος μπορεί να είναι κάτι παραπάνω από ένα αρχείο. Μπορεί να περιέχει πληροφορία όχι μόνο για τον πόρο τον ίδιο, αλλά επίσης και για τις διάφορες ενέργειες που μπορεί κάποιος να εκτελέσει. Σε μία RESTful υπηρεσία όλα τα ερωτήματα που αφορούν έναν πόρο γίνονται χρησιμοποιώντας το μοναδικό αναγνωριστικό του (URI). Τα ερωτήματα που απαντώνται μπορούν να έχουν διάφορες μορφές, όπως για παράδειγμα κείμενο σε HTML, JSON, XML ή ακόμα και εικόνες. Μία απάντηση, ανεξαρτήτως μορφής μπορεί να περιέχει πληροφορία για την ενέργεια που έγινε (πχ αν ο πόρος που ζητήθηκε υπάρχει ή όχι, αν άλλαξε κλπ), πληροφορίες που αφορούν μεταδεδομένα του ερωτήματος, ή διάφορους συνδέσμους υπερκειμένου που αφορούν τον πόρο που αφορά το ερώτημα.

Όσον αφορά τα ερωτήματα σε μία RESTful υπηρεσία, ο πιο κοινός και διαδεδομένος τρόπος είναι χρησιμοποιώντας το πρωτόκολλο HTTP και το σύνολο των εντολών που αυτό προσφέρει. Οι βασικότερες εντολές είναι: GET, POST, PUT, DELETE, OPTION και PATCH. Χρησιμοποιώντας αυτές τις εντολές και με βάση την ίδια την αρχιτεκτονική του REST επιτυγχάνεται αποδοτικότητα, σταθερότητα και αξιοπιστία του συστήματος, όπως επίσης και επεκτασιμότητα χρησιμοποιώντας ήδη υλοποιημένα μέρη του για την δημιουργία κάποιου καινούργιου.

2.2.2 Περιορισμοί της αρχιτεκτονικής του REST

Στην ουσία μια αρχιτεκτονική REST δεν είναι τίποτα παραπάνω από ένα σύνολο κανόνων που θα πρέπει ένα σύστημα να ακολουθεί προκειμένου να να θεωρείται RESTful. Αν έστω και ένας από αυτούς τους κανόνες παραβιάζεται, τότε ένα σύστημα δεν μπορεί να θεωρείται ως τέτοιο. Ακολουθώντας αυτούς τους κανόνες ένα σύστημα μπορεί να επιτύχει αποδοτικότητα, επεκτασιμότητα, απλότητα, δυνατότητα τροποποίησης, εύκολη ορατότητα, φορητότητα και αξιοπιστία, όπως θα εξηγηθεί παρακάτω.

Οι αναφερόμενοι κανόνες είναι οι εξής:

2.2.2.1 Μοντέλο πελάτη-διακομιστή (client-server)

Ο πρώτος περιορισμός αφορά την υλοποίηση μιας RESTful υπηρεσίας ακολουθώντας τη λογική του πελάτη-διακομιστή. Αυτό πρακτικά σημαίνει ότι η διεπαφή του χρήστη (πελάτη) διαχωρίζεται από αυτήν του διακομιστή, ο οποίος είναι υπεύθυνος για την αποθήκευση των δεδομένων.

Αυτή η προσέγγιση έχει διάφορα πλεονεκτήματα. Από την πλευρά του διακομιστή αυτό βοηθάει στον διαχωρισμό μεταξύ των λειτουργικών μονάδων βοηθώντας τες να αναπτυχθούν αυτόνομα, χωρίς να εξαρτάται η μία από την άλλη. Ως συνέπεια αυτό βοηθάει στην επεκτασιμότητα του συστήματος, αφού πλέον οι λειτουργικές μονάδες είναι διαχωρίσιμες και κατά συνέπεια πιο απλές.
Επίσης συμβάλλει στην φορητότητα του συστήματος από την πλευρά του πελάτη, αφού πλέον για να μπορεί να λειτουργήσει σε πλατφόρμες χρειάζεται μόνο η αναπροσαρμογή της διεπαφής του χρήστη και όχι όλου του συστήματος εκ νέου.

2.2.2.2 Stateless

Η επικοινωνία πελάτη-διακομιστή περιορίζεται από το γεγονός ότι δεν επιτρέπεται η αποθήκευση πληροφορίας που αφορά ένα ερώτημα με σκοπό να χρησιμοποιηθεί σε κάποιο επόμενο ερώτημα.

Η ουσία πίσω από αυτή τη λογική είναι ότι κάθε ερώτημα θα πρέπει να είναι σαφώς καθορισμένο και να περιέχει όση πληροφορία είναι απαραίτητη για να απαντηθεί. Ο διακομιστής δεν είναι υποχρεωμένος και ούτε πρέπει να “θυμάται” τι ερωτήθηκε σε προηγούμενη στιγμή για να απαντήσει αναλόγως.

Ο πελάτης μπορεί, παρά όλα αυτά, να επιθυμεί να αλλάξει την κατάσταση του συστήματος. Αυτό το επιτυγχάνει με ερωτήματα προς το διακομιστή. Ο διακομιστής τότε είναι υποχρεωμένος να απαντήσει αναλόγως, δίνοντας, προαιρετικά, συνδέσμους που να αφορούν στο επόμενο βήμα που μπορεί να ακολουθήσει ο πελάτης σε κάποιο επόμενο ερώτημα. Κατά τη διαδικασία αλλαγής κατάστασης του συστήματος ο πελάτης θεωρείται ότι είναι “σε μετάβαση” (in transision).

Πλεονεκτήματα αυτού του περιορισμού είναι ότι καθιστά το σύστημα λιγότερο πολύπλοκο και βοηθάει στην επεκτασιμότητά του, αφού δεν χρειάζεται ο διακομιστής να “θυμάται” τι είχε περιείχαν προηγούμενα ερωτήματα του πελάτη.

2.2.2.3 Cacheable

Κάποιος πελάτης που βρίσκεται στον παγκόσμιο ιστό μπορεί κατά βούληση να αποθηκεύσει κάποια απάντηση που παίρνει από κάποιον διακομιστή με σκοπό την μετέπειτα χρήση του.

Αυτό έχει ως συνέπεια ότι θα πρέπει οι απαντήσεις ενός RESTful διακομιστή να ορίζουν αν επιτρέπεται να είναι αποθηκεύσιμες (cacheable) ή όχι ώστε να αποτρέπει τους πελάτες να την επαναχρησιμοποιούν με κίνδυνο την ανεπιθύμητη αλλαγή κατάστασης ή την λήψη λανθασμένων δεδομένων.

Πλεονεκτήματα αυτού του κανόνα είναι η αποδοτικότητα και η επεκτασιμότητα, αφού βοηθάει στην εξάλειψη ερωτημάτων που θα οδηγήσουν σε λανθασμένη κατάσταση που θα είχε ως αποτέλεσμα να πρέπει να υποβληθούν ξανά από την αρχή.

2.2.2.4 Διαβαθμισμένο σύστημα (layered system)

Ένας πελάτης δε θα πρέπει να μπορεί να αντιληφθεί αν επικοινωνεί με τον κεντρικό διακομιστή ή με κάποιον αντιπρόσωπό του (πχ proxy).

Αυτό έχει ως αποτέλεσμα να μπορεί το σύστημα να επιβάλει πολιτική εξομάλυνσης του φόρτου δεδομένων, πολιτική ασφάλειας κλπ ώστε να είναι πιο σταθερό και ανταποκρίσιμο σε μεγάλο φόρτο εργασίας.

2.2.2.5 Ομοιόμορφη διεπαφή

Η ομοιόμορφη διεπαφή έχει ιδιαίτερη σημασία στην REST αρχιτεκτονική και αφορά σε ένα σύνολο κανόνων με σκοπό την απλοποίηση της αρχιτεκτονικής και την διευκόλυνση στο να αναπτυχθούν οι επιμέρους λειτουργικές μονάδες ξεχωριστά.

Οι περιορισμοί αυτοί είναι οι εξής:

- Ταυτοποίηση των πόρων

  Σε ένα διακομιστή οι πόροι μπορεί να είναι αποθηκευμένοι σε μία ή περισσότερες βάσεις δεδομένων. Ένας πελάτης μπορεί να ζητήσει πληροφορίες που αφορούν σε έναν συγκεκριμένο πόρο και να πάρει μια απάντηση σε διάφορες μορφές (HTML, XML, JSON). Είναι προφανές ότι η αναπαράσταση της απάντησης που παίρνει ο πελάτης δεν έχει καμία σχέση με την αναπαράσταση του πόρου και το πως είναι αποθηκευμένος αυτός στον διακομιστή. Παρά ταύτα θα πρέπει ο κάθε πόρος να είναι μοναδικά ταυτοποιήσιμος μέσω ενός συγκεκριμένου αναγνωριστικού (URI).

- Χειρισμός των πόρων μέσω αναπαραστάσεων

  Όπως προαναφέρθηκε η αναπαράσταση ενός πόρου δεν έχει απαραίτητα σχέση με το πως είναι αποθηκευμένος αυτός στον διακομιστή. Ο πελάτης ωστόσο θα πρέπει να έχει τη δυνατότητα να αλλάξει τον πόρο αυτόν (διαγράφοντας ή αλλάζοντάς τον) χρησιμοποιώντας μόνο την αναπαράσταση που έχει για αυτόν.

- Αυτόεπεξηγηματικά μηνύματα

  Κάθε μήνυμα από τον διακομιστή θα πρέπει να έχει αρκετή πληροφορία ώστε να μπορεί ο πελάτης να καταλάβει πως να χειριστεί το ίδιο το μήνυμα. Για παράδειγμα, αν το μήνυμα περιέχει εικόνα θα πρέπει να έχει αρκετή πληροφορία ώστε να ξέρει ο πελάτης ποιο πρόγραμμα θα χρησιμοποιήσει για να την επεξεργαστεί.

- Hypermedia as the engine of application state (HATEOAS)

  Αυτό είναι κάτι στο οποίο θα δώσουμε περισσότερη έκταση σε επόμενο κεφάλαιο. Εν συντομία θα πρέπει ο πελάτης να μπορεί να “ανακαλύψει” μια υπηρεσία εξερευνώντας την όπως θα έκανε ένας άνθρωπος. Κατά συνέπεια ο διακομιστής είναι υποχρεωμένος να παρέχει συνδέσμους για το πως μπορεί ο πελάτης να πλοηγηθεί από ένα μέρος της υπηρεσίας σε κάποιο άλλο, χωρίς να είναι ο πελάτης υποχρεωμένος να ξέρει από πριν ποιους συνδέσμους πρέπει να ακολουθήσει, ή πως είναι δομημένο ένα σύστημα.

2.2.2.6 Code on demand (προαιρετικό)

Ο διακομιστής θα πρέπει να μπορεί να επεκτείνει τη λειτουργικότητα ενός πελάτης με τη μεταφορά εκτελέσιμου κώδικα. Παραδείγματα αυτού μπορούν να βρεθούν σε Java Applets, όπως επίσης και με τη δημοφιλή γλώσσα πελάτη Javascript.

2.2.3 Ιδιότητες της αρχιτεκτονικής του REST

Οι ιδιότητες και τα πλεονεκτήματα που προκύπτουν από τους παραπάνω περιορισμούς μιας REST αρχιτεκτονικής είναι οι εξής:

- Αποδοτικότητα
  Η αποδοτικότητα των επιμέρους λειτουργικών μονάδων ενός συστήματος είναι ο κύριος παράγοντας για την αποδοτικότητα του συστήματος.

- Επεκτασιμότητα
  Αφορά την δυνατότητα ύπαρξης πολλών επιμέρους λειτουργικών μονάδων, όπως επίσης και την δυνατότητα υποστήριξης μεγάλου όγκου αλληλεπιδράσεων μεταξύ τους.

- Απλότητα
  Αφορά την απλότητα στη διεπαφή του πελάτη, όπως περιγράφηκε παραπάνω.

- Δυνατότητα τροποποίησης
  Αφορά τη δυνατότητα τροποποίησης των λειτουργικών μονάδων του συστήματος όποτε χρειάζεται, ακόμα και στην περίπτωση που αυτό είναι σε λειτουργία.

- Ορατότητα
  Αφορά τη δυνατότητα του συστήματος να μπορεί να παρακολουθεί τι μηνύματα ανταλλάσσουν διαφορετικές λειτουργικές μονάδες μεταξύ τους.
- Φορητότητα
  Είναι η δυνατότητα του συστήματος να μπορεί να λειτουργήσει σε διαφορετικές πλατφόρμες.

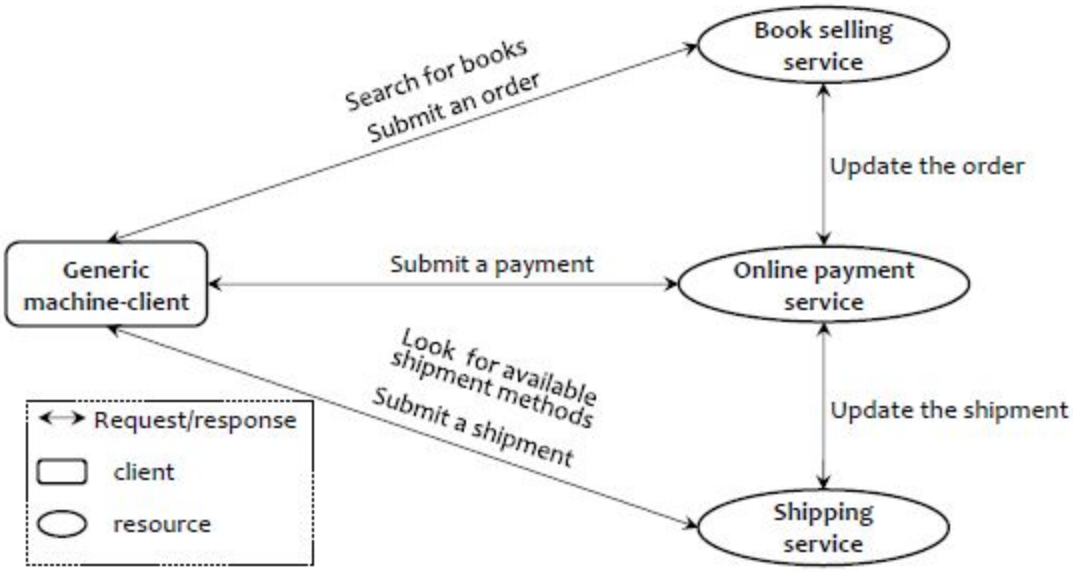
- Αξιοπιστία
  Αφορά τη δυνατότητα του συστήματος να μπορεί να λειτουργεί κανονικά ακόμα και σε περίπτωση σφάλματος μιας επιμέρους μονάδας του.

2.2.4 HATEOAS

Έγινε ήδη μια σύντομη αναφορά στο HATEOAS [1][2][5][] προηγουμένως στους περιορισμούς που θέτει το REST, ωστόσο θα θέλαμε να εμβαθύνουμε λίγο περισσότερο σε αυτό. Ο κύριος λόγος για αυτό είναι το γεγονός, ότι παρά το ότι το HATEOAS αποτελεί δομικό περιορισμό ώστε ένα σύστημα να μπορεί να χαρακτηριστεί ως RESTful δεν είναι πολλά τα συστήματα σήμερα τα οποία συμμορφώνονται με αυτόν.

Εδώ θα πρέπει να εξηγήσουμε τη σημαίνει πρακτικά HATEOAS. Η κύρια ιδέα πίσω από το HATEOAS είναι ότι θα πρέπει ο διακομιστής και το σύστημα να παρέχει στον πελάτη τη δυνατότητα, μέσω υπερσυνδέσμων, να πλοηγηθεί και να ανακαλύψει μια υπηρεσία και τα επιμέρους μέρη της χωρίς να χρειάζεται να γνωρίζει πολλές λεπτομέρειες για αυτήν από πριν. Θα πρέπει δηλαδή ο πελάτης μα λειτουργεί, όπως θα λειτουργούσε ένας άνθρωπος ο οποίος προσπαθεί να ανακαλύψει μία ιστοσελίδα κάνοντας περιήγηση μέσα από τις διαθέσιμες επιλογές της.

Για να γίνει πιο εύκολα αντιληπτό ας πάρουμε το παράδειγμα ενός ηλεκτρονικού καταστήματος το οποίο πουλάει βιβλία. Αυτό το κατάστημα θα έχει μία υπηρεσία που θα αφορά την επιλογή του βιβλίου προς αγορά, μία υπηρεσία για την ηλεκτρονική πληρωμή και μία υπηρεσία που θα αφορά την αποστολή του βιβλίου στο χώρο του χρήστη. Χωρίς τη χρήση HATEOAS, ένα πρόγραμμα-πελάτης θα πρέπει να ξέρει από πριν πως είναι δομημένη η υπηρεσία, τη λογική που ακολουθεί καθώς και τα URIs των εκάστοτε πόρων της υπηρεσίας για να μπορεί να έχει πρόσβαση σε αυτούς. Μία τέτοια αρχιτεκτονική παρουσιάζεται στο παρακάτω σχήμα.



Σχήμα 2.1: Υπηρεσία χωρίς HATEOAS

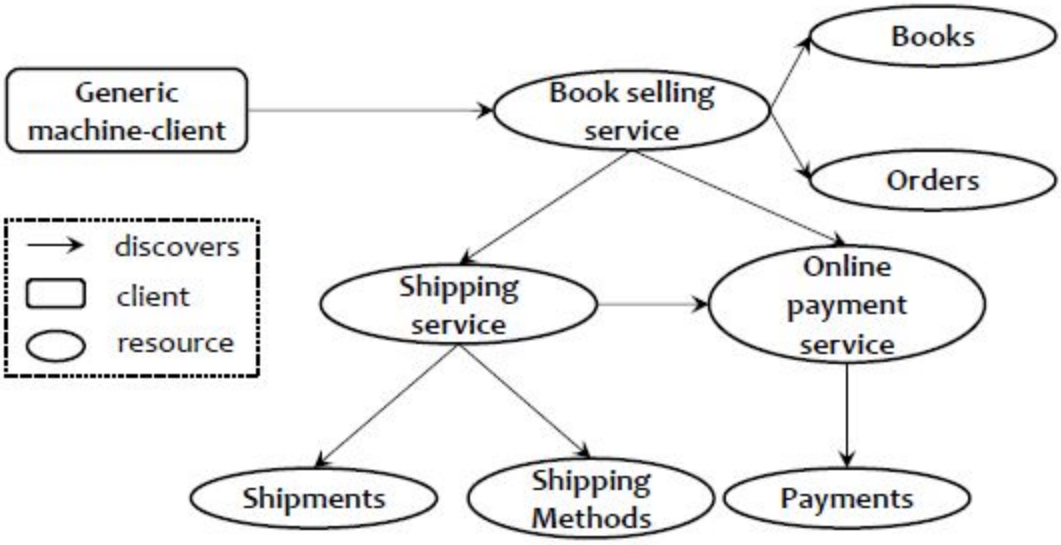
Μια τέτοια υπηρεσία αν θέλουμε να ικανοποιεί τον περιορισμό περί HATEOAS θα έπρεπε να διαμορφωθεί λίγο διαφορετικά. Αρχικά θα υπάρχει μια κεντρική υπηρεσία που λειτουργεί ως κόμβος εισαγωγής στο σύστημα και είναι ουσιαστικά το μόνο που χρειάζεται να ξέρει το πρόγραμμα-πελάτης. Όταν ο πελάτης κάνει ένα ερώτημα σε αυτόν τον κόμβο αυτός θα απαντάει με όλες τις δυνατές επιλογές του πελάτη. Αρχικά αυτές θα είναι μόνο η επιλογή βιβλίου προς αγορά, αφού δεν θα έχει προηγηθεί κάποια επιλογή.

Στη συνέχεια αφού ο πελάτης επιλέξει βιβλίο η υπηρεσία μπορεί να απαντήσει με μια αναπαράσταση που θα αφορά το βιβλίο/α που επιλέχτηκαν καθώς επίσης και με υπερσυνδέσμους που θα αφορούν την συγκεκριμένη παραγγελία, όπως επίσης και την ηλεκτρονική πληρωμή.

Έπειτα ο πελάτης μπορεί να επισκεφτεί τον υπερσύνδεσμο για την παραγγελία για να δει λεπτομέρειες για αυτήν ή τον σύνδεσμο που αφορά την ηλεκτρονική πληρωμή για να πληρώσει για την παραγγελία του. Σε περίπτωση που γίνει το δεύτερο, ο διακομιστής μπορεί να απαντήσει με μία αναπαράσταση που θα αφορά την απόδειξη της παραγγελίας με επιπλέον υπερσυνδέσμους για την πρόοδο της αποστολής της παραγγελίας που μόλις έγινε.

Είναι εύκολα αντιληπτό πως το πρόγραμμα πελάτης αρχικά δεν γνώριζε για τις επιμέρους υπηρεσίες ηλεκτρονικών πληρωμών και αποστολής των βιβλίων και ούτε χρειαζόταν. Το σύστημα από μόνο του έδωσε τη δυνατότητα στον πελάτη να περιηγηθεί σε αυτές τις υπηρεσίες χρησιμοποιώντας υπερσυνδέσμους σε αυτές, όταν μια ενέργεια του πελάτη είχε σαν επόμενο “λογικό” βήμα την εκάστοτε υπηρεσία. Αυτή είναι και η κεντρική ιδέα του HATEOAS.

Στη συνέχεια δίνεται σχήμα της δόμησης μιας υπηρεσίας πώλησης βιβλίων με βάση το HATEOAS



Σχήμα 2.2: Υπηρεσία πώλησης βιβλίων σε REST HATEOAS αρχιτεκτονική

Πλεονεκτήματα μιας τέτοιας προσέγγισης υπάρχουν πολλά. Το κύριο πλεονέκτημα είναι ότι αν μια υπηρεσία χρησιμοποιεί μια HATEOAS αρχιτεκτονική, τότε μπορούν να υλοποιηθούν “έξυπνοι” πελάτες που θα μπορούν να προσπελάσουν και να εξερευνήσουν μια τέτοια υπηρεσία αυτόματα. Σε περίπτωση δε που σε αυτήν την υπηρεσία προστεθούν επιπλέον λειτουργίες δε θα χρειάζεται η αναπροσαρμογή του προγράμματος-πελάτη.

Επίσης όσον αφορά μερικές λογικές σκέψεις που γίνονται σχετικά με το πόσο πιο αποδοτικό είναι κάτι τέτοιο, δεδομένου ότι για να προσπελάσει ένας πελάτης μια υπηρεσία που είναι “κρυμμένη” πίσω από μια κεντρική υπηρεσία, πρέπει πρώτα να κάνει επιπλέον ερωτήματα στην κεντρική υπηρεσία η απάντηση κρύβεται στο ίδιο το REST. Το REST με βάση τους δομικούς του περιορισμούς επιτρέπει την αποθήκευση αναπαραστάσεων και απαντήσεων (cacheability). Οπότε ένας πελάτης δεν είναι ανάγκη να κάνει κάθε φορά την ίδια σειρά ερωτημάτων για να φτάσει σε μια πιο “κρυμμένη” υπηρεσία. Αρκεί το σύστημα να είναι υλοποιημένο σωστά και να επιτρέπει στον πελάτη να αποθηκεύει απαντήσεις σε ερωτήματα που έχει κάνει ώστε να “θυμάται” πως είναι δομημένη η υπηρεσία. Εξάλλου η πραγματοποίηση μερικών παραπάνω ερωτημάτων είναι πολύ μικρό μειονέκτημα σε σχέση με το πλεονέκτημα των “έξυπνων” και αυτόματων προγραμμάτων-πελάτη που θα μπορούσαν να δημιουργηθούν.

2.3 Αναπαράσταση πόρων RESTful υπηρεσιών

Ένα ακόμα σημαντικό κομμάτι με το οποίο ασχοληθήκαμε σε αυτήν την διπλωματική εργασία είναι αυτό που αφορά τον τρόπο αναπαράστασης των πόρων μιας RESTful υπηρεσίας. Η αναπαράσταση ενός πόρου, τις περισσότερες φορές δεν έχει καμία σχέση με το πως είναι αποθηκευμένος ο πόρος στο σύστημα (διακομιστή). Η αναπαράσταση αυτή μπορεί να αποκρύπτει δεδομένα που υπάρχουν για έναν συγκεκριμένο πόρο που είναι αποθηκευμένος σε μία βάση δεδομένων, είτε για ασφάλεια, είτε γιατί κάποια δεδομένα δεν έχουν ιδιαίτερο ενδιαφέρον και σχέση με κάποιο ερώτημα.

Μπορούν να υπάρχουν πολλές αναπαραστάσεις για τον ίδιο πόρο, οι οποίες μάλιστα να έχουν ακριβή αντιστοιχία μεταξύ τους. Πολλές φορές βλέπουμε συστήματα να προσφέρουν διάφορους τρόπους αναπαράστασης του ίδιου πόρου με την επιλογή να επαφίεται στον χρήστη (πρόγραμμα-πελάτη) που κάνει το ερώτημα. Στην παρούσα εργασία και υλοποίηση δόθηκε ιδιαίτερο ενδιαφέρον στην υποστήριξη μεγάλου αριθμού διαθέσιμων αναπαραστάσεων, όσο αυτό είναι δυνατό. Ο λόγος για κάτι τέτοιο έγκειται στο γεγονός ότι η φύση της παρούσας υλοποίησης είναι να μπορεί να προσαρμόζεται σε εξωτερικές RESTful υπηρεσίες χωρίς μάλιστα οι ίδιες οι υπηρεσίες να γνωρίζουν απαραίτητα για αυτό. Δεδομένου ότι δεν μπορούμε να απαιτήσουμε από όλες τις υπηρεσίες να υποστηρίζουν μία συγκεκριμένη αναπαράσταση θα πρέπει να λάβουμε υπόψιν τις πιο διαδεδομένες από αυτές και να παρέχουμε υποστήριξη για αυτές.

Μερικές από τις πιο διαδεδομένες αναπαραστάσεις πόρων REST παρουσιάζονται στη συνέχεια.

2.3.1 XML

2.3.1.1 Γενικά

Το XML [16] ή αλλιώς eXtensible Markup Language αποτελεί έναν τρόπο αναπαράστασης πόρων. Είναι η πιο απλή μορφή αναπαράστασης με κείμενο και ουσιαστικά αποτελείται από ένα σύνολο κανόνων για την κωδικοποίηση κειμένου σε μορφή που να είναι αναγνώσιμη από ανθρώπους και υπολογιστές ταυτόχρονα.

Ο σχεδιασμός του XML στοχεύει στην παρουσίαση πόρων μέσω διαδικτύου με τρόπο γενικό και ταυτόχρονα απλό και χρήσιμο. Παρέχει υποστήριξη για μεγάλο αριθμό ανθρώπινων γλωσσών

(Unicode). Παρά το γεγονός ότι το XML αφορά κυρίως αναπαράσταση και δόμηση πόρων με κείμενο, χρησιμοποιείται επίσης για την αναπαράσταση αόριστων δομών δεδομένων (πχ εικόνες, κλάσεις αντικειμένων κλπ) όπως αυτές που χρησιμοποιούν οι σημερινές RESTful υπηρεσίες.

2.3.1.2 Δομή του XML

Με βάση το πως έχει προσδιοριστεί το XML, υπάρχει συγκεκριμένη δομή ενός κειμένου ώστε να μπορεί να θεωρηθεί ως κείμενο XML. Τα βασικότερα δομικά στοιχεία είναι τα εξής:

- Ετικέτα (Tag)

  Μία ετικέτα ουσιαστικά ορίζει την ύπαρξη ενός γνωστικού περιεχομένου (πχ βιβλίο, τίτλος, συγγραφέας) σε ένα κείμενο. Ξεκινάει με το χαρακτήρα < και τελειώνει με το χαρακτήρα >. Υπάρχουν τρία είδη ετικετών.
  - Ετικέτα έναρξης (start-tag), όπως για παράδειγμα `<title>`
  - Ετικέτα τερματισμού (end-tag), όπως για παράδειγμα `</title>`
  - Ετικέτα χωρίς στοιχείο (empty-element tag), οπως για παράδειγμα `<separator />`

- Στοιχείο (Element)

  Ένα στοιχείο αποτελεί τη βασική δομική μονάδα σε ένα κείμενο XML. Συνήθως ξεκινάει με μία ετικέτα έναρξης και τελειώνει με την αντίστοιχη ετικέτα τερματισμού, αλλά μπορεί να αποτελείται και από μια ετικέτα χωρίς στοιχείο. Το κείμενο που περιέχεται ανάμεσα στη ετικέτα έναρξης και τερματισμού ονομάζεται περιεχόμενο του στοιχείου και μπορεί να αποτελείται από άλλα στοιχεία (elements), τα οποία ονομάζονται παιδιά (children) του στοιχείου.
  Παράδειγμα: `<title>The Hitchhiker’s guide to galaxy</title>`

- Χαρακτηριστικό (Attribute)

  Αυτή η δομική μονάδα του XML χρησιμοποιείται για να προσδώσει χαρακτηριστικά σε κάποιο στοιχείο. Μπορεί να υπάρξει μόνο σε ετικέτες έναρξης και σε ετικέτες χωρίς στοιχείο και αποτελείται από ένα ζευγάρι κλειδιού-τιμής. Στο παράδειγμα: `<book id=”3” />` το χαρακτηριστικό είναι το id με τιμή 3 και περιγράφει ότι ένα συγκεκριμένο βιβλίο έχει χαρακτηριστικό αριθμό 3. Συνήθως μόνο μία τιμή υπάρχει για κάθε χαρακτηριστικό, αλλά μπορούν να δοθούν λίστες για τιμές χωριζόμενες είτε από κόμμα, είτε από την αγγλική άνω τελεία (semicolon), ή σε περίπτωση που δεν υπάρχουν τιμές με κενά, από κενά.
  Παράδειγμα: `<div class=”wrapper header outline”></class>`

- Δήλωση XML (XML Declaration)

Υπάρχει στην αρχή ενός XML κειμένου και χρησιμοποιείται για να περιγράψει και δώσει περισσότερες πληροφορίες για το ίδιο το κείμενο. Παράδειγμα: `<?xml version="1.0" encoding="UTF-8"?>`

2.3.1.3 Παράδειγμα XML

Ένα απλό αλλά ολοκληρωμένο παράδειγμα XML κειμένου ακολουθεί.

```
<?xml version="1.0" encoding="UTF-8"?>
<note expires=”false” categories=”home entertainment friends”>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

2.3.1.4 Παρατηρήσεις

Το XML σήμερα βρίσκει ιδιαίτερη χρήση στη μοντελοποίηση λογισμικού και γενικότερα σε μοντέλα και μεταμοντέλα. Ωστόσο έχει χάσει έδαφος στην αναπαράστασή πόρων από το JSON, το οποίο αναλύεται παρακάτω.

2.3.2 JSON

2.3.2.1 Γενικά

Το JSON [17] ή αλλιώς JavaScript Object Notation είναι, όπως και το XML, ένα πρότυπο αναπαράστασης πόρων και ανταλλαγής δεδομένων στο διαδίκτυο. Είναι ελαφρύ υπολογιστικά και εύκολο στη χρήση από ανθρώπους και υπολογιστές.

Βασίζεται σε ένα υποσύνολο της γλώσσας προγραμματισμού JavaScript και βρίσκει μεγάλη απήχηση στην ασύγχρονη επικοινωνία πελάτη-διακομιστή (AJAX). Ο λόγος που προτιμάται

από μεγάλη μερίδα προγραμματιστών είναι η οικειότητα που προσφέρει, αφού έχει πολλές ομοιότητες με διάφορες γλώσσες προγραμματισμού.

2.3.2.2 Δομή του JSON

Όπως και το XML έτσι και το JSON έχει ένα σύνολο κανόνων και συγκεκριμένη δομή την οποία πρέπει ένα κείμενο να ακολουθεί για να είναι τέτοιας μορφής.

Τα βασικά δομικά στοιχεία του JSON είναι:

- Αντικείμενο (Object)

  Το αντικείμενο είναι η κεντρική δομική μονάδα του JSON. Ένα JSON αρχείο μπορεί να περιέχει ένα η περισσότερα αντικείμενα σαν ρίζες. Ξεκινάει με το χαρακτήρα `{` και τερματίζει με το `}`. Μπορεί να περιέχει ένα ή περισσότερα μέλη (βλέπε παρακάτω)

- Μέλη (Members)

  Τα μέλη αποτελούνται από ένα ή περισσότερα ζευγάρια χωριζόμενα με κόμμα.

- Ζευγάρι (Pair)

  Το ζευγάρι είναι ουσιαστικά η δομική μονάδα που περιέχει την πληροφορία σε ένα αρχείο JSON και οργανώνεται με τη μορφή κλειδιού-τιμής χωρίζοντας το κλειδί από την τιμή με την άνω κάτω τελεία. Το κλειδί είναι τύπου συμβολοσειράς.
  Παράδειγμα: “`title”: The Hitchhiker’s guide to galaxy`

Οι τιμές σε ένα ζευγάρι μπορεί να είναι διαφόρων ειδών:

- Συμβολοσειρά (String)
  Η συμβολοσειρά είναι μια σειρά χαρακτήρων.
  Παράδειγμα: “`The Hitchhiker’s guide to galaxy`”

- Αριθμός (Number)

  Ο αριθμός μπορεί να είναι οποιοσδήποτε πραγματικός αριθμός
  Παράδειγμα: `42.14`

- Αντικείμενο (Object)

  Η τιμή ενός κλειδιού μπορεί να είναι ένα άλλο αντικείμενο.

- Λογική τιμή (Boolean)

  Μπορεί να είναι `true` ή `false`

- Κενό (null)

- Πίνακας (Array)

  Ο πίνακας μπορεί να περιέχει μια σειρά από τις παραπάνω διαθέσιμες τιμές χωριζόμενες με κόμμα. Ξεκινάει με `[` και τερματίζει με `]`.
  Παράδειγμα: `[“Title”, 42, {}, false, null]`

2.3.2.3 Παράδειγμα JSON

Παρακάτω δίνεται το αντίστοιχο παράδειγμα που δόθηκε για το XML σε JSON.

```
{
	“expires”: false,
	“categories”: [“home”, “entertainment”, “friends”],
	“to: “Tove”,
	“from”: “Jani”,
	“heading”: “Reminder”,
	“body”: “Don’t forget me this weekend!”
}
```

2.3.2.4 Παρατηρήσεις

Σήμερα το JSON είναι ο πιο διαδεδομένος τρόπος μεταφοράς δεδομένων. Σε αυτό έχει παίξει σημαντικό ρόλο η ραγδαία εξάπλωση της γλώσσας προγραμματισμού JavaScript. Όσον αφορά το REST και την αναπαράσταση πόρων είναι το πιο ευρέως χρησιμοποιούμενο πρότυπο.

2.3.3 HAL

2.3.3.1 Γενικά

Το HAL [18] ή αλλιώς Hypertext Application Language είναι ένα απλό πρότυπο που βοηθάει στην δημιουργία υπερσυνδέσμων μεταξύ πόρων σε μία διεπαφή εφαρμογής (API). Το HAL βοηθάει έτσι ώστε μια υπηρεσία να διαθέτει διεπαφές με τέτοιο τρόπο ώστε να είναι εύκολα προσβάσιμες και πλοηγήσιμες από προγράμματα-πελάτες ικανοποιώντας τον δομικό περιορισμό του REST περί HATEOAS, όπως αναφέρθηκε προηγουμένως.

Επί της ουσίας το HAL αποτελεί ένα σύνολο κανόνων και συμβιβασμών πάνω στα ήδη υπάρχοντα πρότυπα XML και JSON και η ύπαρξή του βοηθάει στο να αποφεύγεται ανούσιος σχεδιασμός και δημιουργία επιπλέον προτύπων για την περιγραφή των δομών και σχέσεων των πόρων σε ένα σύστημα.

2.3.3.2 Δομή HAL

Το HAL αφορά δύο βασικά στοιχεία. Τις αναπαραστάσεις πόρων (Resources) και τους συνδέσμους (Links).

Οι αναπαραστάσεις πόρων περιέχουν:

- Συνδέσμους σε άλλους πόρους (Links)

- Ενσωματωμένες αναπαραστάσεις
  Πχ άλλες αναπαραστάσεις που περιέχονται στην βασική αναπαράσταση.

- Καταστάσεις
  Διάφορα δεδομένα που αφορούν την τρέχουσα αναπαράσταση.

Οι σύνδεσμοι περιέχουν:

- Τον ίδιο τον σύνδεσμο (URI)

- Συσχετίσεις
  Χρησιμοποιείται η λέξη rel για να ορίσει μια συσχέτιση.

- Διάφορες προαιρετικές επιλογές σχετικές με το περιεχόμενο που αφορά ο σύνδεσμος.

Πχ η χρήση της λέξης curies που αφορά στην περιγραφή των προδιαγραφών ενός πόρου.

2.3.3.3 Παράδειγμα HAL

Παρακάτω δίνεται ένα παράδειγμα HAL με βάση το πρότυπο JSON.

```
{
    "_links": {
            "self": { "href": "/example_resource" },
            "next": { "href": "/page=2" },
      "items": [{
            "href": "/first_item"
            },{
            "href": "/second_item"
      }],
      "curies": [{
            "name": "doc",
            "href": "http://haltalk.herokuapp.com/docs/{rel}",
            "templated": true
      }],
      "doc:latest-posts": {
            "href": "/posts/latest"
      }
    }
}
```

2.3.3.4 Παρατηρήσεις

Παρά το γεγονός ότι το HAL δεν εξαρτάται από τα πρότυπα JSON και XML, σήμερα χρησιμοποιείται ευρέως το πρότυπο HAL JSON με μεγάλη υποστήριξη μέσω βιβλιοθηκών ανοιχτού κώδικα σε πληθώρα γλωσσών προγραμματισμού. Έχει μάλιστα καθιερωθεί και το αντίστοιχο MediaType: application/hal+json
Είναι ένα από τα βασικά πρότυπα που χρησιμοποιούνται από RESTful υπηρεσίες για να ικανοποιήσουν τον HATEOAS περιορισμό. Ωστόσο υπάρχουν και άλλα πρότυπα παρόμοια με αυτό, όπως το JSON-LD/Hydra, τα οποία αναλύονται παρακάτω.

2.3.4 JSON-LD

2.3.4.1 Γενικά

Το JSON-LD [19][22] ή αλλιώς JavaScript Object Notation for Linked Data είναι ένα διαδεδομένο πρότυπο για περιγραφή συσχετίσεων μεταξύ δεδομένων. Ο κύριος στόχος του είναι να χρειάζεται όσο το δυνατόν λιγότερη προσπάθεια από τους προγραμματιστές για να περιγραφούν οι ζητούμενες συσχετίσεις χωρίς να χρειάζεται ο σχεδιασμός ενός καινούργιου προτύπου περιγραφής δεδομένων.

Για αυτό το λόγο το JSON-LD αποτελεί ένα σύνολο κανόνων και ορολογιών με βάση το ήδη υπάρχον πρότυπο JSON.

2.3.4.1 Δομή JSON-LD

Η δομή του JSON-LD ακολουθεί την δομή που ακολουθεί και το JSON. Παρά ταύτα έχει σχεδιαστεί για να παρέχει αντιστοιχίσεις μεταξύ αντικειμένων JSON και μοντέλων περιγραφής οντοτήτων, όπως για παράδειγμα το Resource Description Framework (RDF).

Επί της ουσίας παρέχει πληροφορία για το περιεχόμενο των πόρων και των οντοτήτων που υπάρχουν σε μια JSON αναπαράσταση. Η πληροφορία αυτή αφορά κυρίως τον τύπο του αντικειμένου και είναι συνήθως ένας σύνδεσμος (Link) σε κάποιο εξωτερικό μοντέλο περιγραφής οντοτήτων.

2.3.4.1 Παράδειγμα JSON-LD

Παρακάτω δίνεται ένα παράδειγμα σε JSON-LD το οποίο αφορά την περιγραφή ενός αντικειμένου JSON το οποίο είναι ένα φυσικό πρόσωπο.

```
{
  "@context": "http://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
```

```
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
```

2.3.4.1 Παρατηρήσεις

Είναι φανερό από το παραπάνω παράδειγμα ότι με το JSON-LD γίνεται εύκολο να περιγράψουμε τι τύπου είναι το JSON αντικείμενο με τη χρήση της λέξης κλειδιού `@context`. Η τιμή της παραπέμπει σε κάποιο μοντέλο το οποίο περιγράφει τι περιεχόμενα μπορεί να έχει, καθώς και πως είναι δομημένο μία οντότητα τύπου person.

Επίσης παρέχεται σύνδεσμος στον πόρο που αφορά το συγκεκριμένο πρόσωπο (John Lennon) με χρήση του κλειδιού `@id`, ενώ μπορούμε να δούμε ότι υπάρχει και συσχετισμός του συγκεκριμένου προσώπου με ένα τρίτο (`spouse`) χρησιμοποιώντας κάποιον σύνδεσμο.

Όλα τα παραπάνω δεν παραβιάζουν τους δομικούς περιορισμούς του JSON, ωστόσο είναι φανερό ότι τα αντικείμενα περιέχουν επιπλέον πληροφορία με την οποία μπορεί κάποιο πρόγραμμα-πελάτης να την “κατανοήσει” και να την προσδιορίσει πιο εύκολα. Με αυτόν τον τρόπο δίνεται η δυνατότητα στους προγραμματιστές να δημιουργήσουν “έξυπνα” προγράμματα-πελάτες που θα μπορούν χωρίς επιπλέον εργασία να καταλάβουν αντικείμενα JSON και κατά συνέπεια αναπαραστάσεις REST πόρων οι οποίες μπορεί να προέρχονται ακόμα και από διαφορετικές διαδικτυακές υπηρεσίες, αρκεί να ακολουθούν το ίδιο μοντέλο οντοτήτων για να την περιγραφή τους.

2.3.5 Hydra

2.3.5.1 Γενικά

Το Hydra [20][23][25] αποτελεί μια προσπάθεια που στοχεύει στην απλοποίηση των διαδικτυακών διεπαφών εφαρμογών (Web APIs). Οι βασικές δομικές μονάδες του είναι το JSON-LD, όπως περιγράφηκε πριν και ένα συγκεκριμένο λεξιλόγιο το οποίο έχει οριστεί σαν σύμβαση για προσθήκη περισσότερον σημασιολογικών εννοιών που περιγράφουν μια αναπαράσταση ενός πόρου.

2.3.5.2 Δομή Hydra

Η δομή του Hydra είναι η ίδια με αυτή του JSON-LD. Παρακάτω παραθέτουμε το λεξιλόγιο που ορίζει το Hydra και προσθέτει επιπλέον σημασιολογική έννοια στις αναπαραστάσεις, χωρίς κάποια επεξήγηση, δεδομένου ότι αυτή μπορεί να βρεθεί στον σχετικό σύνδεσμο που αφορά την προδιαγραφή του.

```
hydra:ApiDocumentation, hydra:Class, hydra:Collection, hydra:Error,
hydra:IriTemplate, hydra:IriTemplateMapping, hydra:Link, hydra:Operation,
hydra:PartialCollectionView, hydra:Resource, hydra:Status, hydra:SupportedProperty,
hydra:TemplatedLink, hydra:VariableRepresentation, hydra:apiDocumentation,
hydra:description, hydra:entrypoint, hydra:expects, hydra:first, hydra:freetextQuery,
hydra:last, hydra:mapping, hydra:member, hydra:method, hydra:next, hydra:operation,
hydra:possibleStatus, hydra:previous, hydra:property, hydra:readable, hydra:required,
hydra:returns, hydra:search, hydra:statusCode, hydra:supportedClass,
hydra:supportedOperation, hydra:supportedProperty, hydra:template, hydra:title,
hydra:totalItems, hydra:variable, hydra:variableRepresentation, hydra:view,
hydra:writeable
```

2.3.5.3 Παράδειγμα Hydra

Παρακάτω παραθέτουμε ένα παράδειγμα σε Hydra το οποίο δείχνει τις λειτουργίες που μπορούμε να εφαρμόσουμε πάνω σε μία αναπαράσταση ενός REST πόρου. Το παράδειγμα ακολουθεί τη δομή του JSON-LD.

```
{
  "@context": "http://www.w3.org/ns/hydra/context.jsonld",
  "@id": "/an-issue",
  "title": "An exemplary issue representation",
  "description": "This issue can be deleted with an HTTP DELETE request",
  "operation": [
    {
      "@type": "Operation",
      "method": "DELETE"
    }
  ]
}
```

2.3.5.4 Παρατηρήσεις

Είναι εμφανές στο συγκεκριμένο παράδειγμα ότι η δομή του κειμένου είναι JSON-LD, ωστόσο πρέπει να παρατηρηθεί η προσθήκη της λέξης κλειδί `operation` η οποία αφορά τις ενέργειες τις οποίες μπορούμε να εφαρμόσουμε στον συγκεκριμένο πόρο (το issue). Πιο συγκεκριμένα η συγκεκριμένη αναπαράσταση του issue μας λέει ότι αν εφαρμόσουμε τη μέθοδο `DELETE` στον σύνδεσμο `/an-issue` μπορούμε να το διαγράψουμε.

Κάτι τέτοιο είναι πολύ σημαντικό διότι με την απλή χρήση του JSON-LD δεν μπορούσαμε να το κάνουμε, ωστόσο με τη χρήση του Hydra μας επιτρέπεται μια σειρά από νέες σημασιολογικές έννοιες γύρω από μία αναπαράσταση η οποία βοηθάει όχι μόνο τη δόμηση του ίδιου το REST, αλλά επίσης και τα προγράμματα-πελάτες να ανακαλύψουν μία υπηρεσία. (HATEOAS)

2.4 Distributed Affordances

2.4.1 Γενικά

Για να καταλάβουμε τι σημαίνει affordance στην παρούσα διπλωματική εργασία θα πρέπει πρώτα να καταλάβουμε πρώτα τι σημαίνει πρακτικά αυτή η λέξη. Ο κόσμος γύρω μας είναι γεμάτος από affordances, ιδιότητες, δηλαδή, αντικειμένων που μας επιτρέπουν να εκτελέσουμε κάποιες ενέργειες. Για παράδειγμα το στυλό είναι το affordance το οποίο μας επιτρέπει να γράψουμε ένα κείμενο, ενώ το χερούλι μιας πόρτας είναι αυτό που μας επιτρέπει να ανοίξουμε μία πόρτα.

Ενδιαφέρον παρουσιάζει το γεγονός ότι στο σημερινό διαδίκτυο η επιλογή και διαθεσιμότητα των affordances, δηλαδή των ενεργειών που μπορεί ο χρήστης να εκτελέσει πάνω σε κάποια αναπαράσταση ενός πόρου, εξαρτάται από αυτόν που παρέχει μια υπηρεσία και όχι από τις ανάγκες του χρήστη. Για παράδειγμα όταν ένας χρήστης ο οποίος κάνει αναζήτηση για ένα βιβλίο μέσω μίας υπηρεσίας, η υπηρεσία μπορεί να επιστρέψει μια αναπαράσταση του βιβλίου και να δώσει επιλογή στον χρήστη να διαβάσει την ηλεκτρονική έκδοση του βιβλίου, ή να προσθέσει τον σύνδεσμο του βιβλίου της τοπικής βιβλιοθήκης που αφορά τον χρήστη. Και οι δύο αυτές επιλογές αποτελούν affordances, ωστόσο η δεύτερη επιλογή, αυτή δηλαδή της βιβλιοθήκης, εξαρτάται εξ' ολοκλήρου από το περιβάλλον του χρήστη (που βρίσκεται, αν υπάρχει τοπική βιβλιοθήκη κλπ). Οι επιλογές του χρήστη επίσης μπορούν να αυξηθούν δραματικά αν αναλογιστεί κανείς ότι ο χρήστης μπορεί να θέλει, για παράδειγμα, να διαβάσει μια περίληψη αυτού του βιβλίου, ένα κεφάλαιο, ή ακόμα να ζητήσει κριτικές από φίλους του στα κοινωνικά δίκτυα..
Το παραπάνω παράδειγμα δείχνει με απλό τρόπο το βασικό πρόβλημα. Οι επιλογές του χρήστη στην πραγματικότητα είναι άπειρες και η υπηρεσία που παρέχει τα affordances δεν μπορεί να δώσει στο χρήστη όλες τις δυνατές επιλογές, αλλά θα πρέπει να μπορεί να κάνει εξατομίκευση του περιεχομένου που προσφέρει στον χρήστη ώστε η εμπειρία του να είναι όσο το δυνατόν πιο πλούσια και ομαλή.

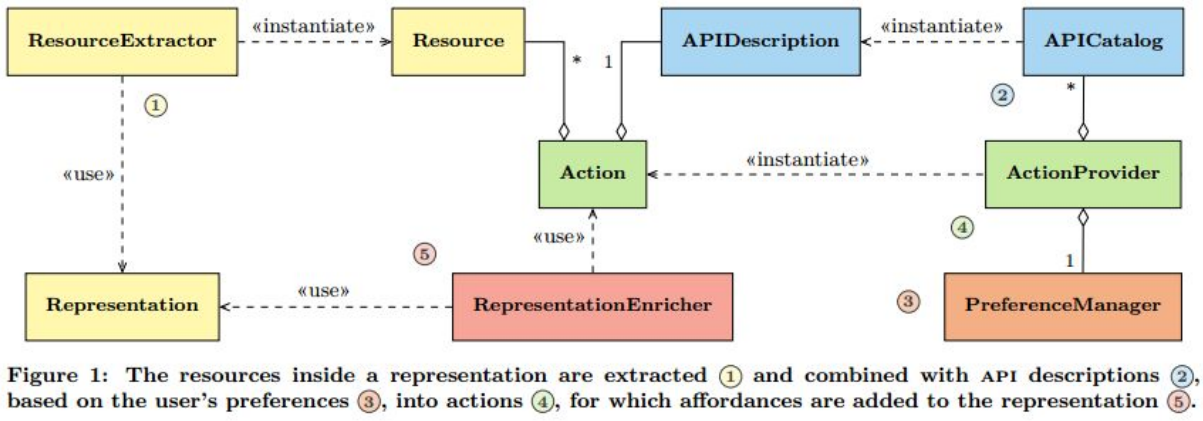
Με βάση το παραπάνω καθίσταται σαφές ότι είναι σημαντική η έννοια των κατανεμημένων affordances (distributed affordances) [9][26][27]. Η ιδέα είναι να μπορούν να δημιουργηθούν affordances δυναμικά με βάση την αναπαράσταση ενός πόρου, καθώς επίσης και από την πληροφορία που υπάρχει για κάποιον χρήστη/πελάτη με βάση διάφορες κατανεμημένες πηγές.

Επίσης δε χρειάζεται να υποθέσουμε ότι η υπηρεσία γνωρίζει τις ενέργειες που επιθυμεί να εκτελέσει ο χρήστης. Το μόνο που χρειάζεται να γνωρίζει είναι το περιεχόμενο της αναπαράστασης ενός πόρου, καθώς και πληροφορίες που αφορούν το χρήστη, όπως το προφίλ του, τις επιλογές του και το περιβάλλον περιήγησής του. Στη συνέχεια κατασκευάζεται το affordance που είναι πιο πιθανό να ενδιαφέρει το χρήστη.

2.4.2 Αρχιτεκτονική συστήματος distributed affordances

Αφού περιγράψαμε όσο πιο απλά γινόταν την έννοια των distributed affordances, θα πρέπει να παρουσιάσουμε την αρχιτεκτονική ενός συστήματος distributed affordances. Υπάρχουν διάφορες τεχνικές δυσκολίες σχετικά με μια τέτοια υλοποίηση. Αρχικά πρέπει να εξαχθεί όλη η πληροφορία που αφορά μια αναπαράσταση, ενώ είναι σημαντικό να γνωρίζουμε όλες τις δυνατές ενέργειες που επιτρέπει μια υπηρεσία. Τέλος θα πρέπει να υπάρχει τρόπος για να συλλέγονται όλες οι προσωπικές επιλογές ενός χρήστη ώστε να καταστεί δυνατή η εξατομίκευση των affordances.

Στο παρακάτω σχήμα παρουσιάζεται η αρχιτεκτονική ενός συστήματος που χρησιμοποιεί τη λογική των distributed affordances



Σχήμα 2.3: Αρχιτεκτονική συστήματος με distributed affordances

Παρακάτω αναλύουμε τα πέντε βασικά δομικά χαρακτηριστικά της παραπάνω αρχιτεκτονικής

1. **Εξαγωγή πληροφορίας**. Δεδομένης μιας αναπαράστασης που δεν περιέχει ενέργειες που μπορεί να κάνει ο χρήστης, ο ResourceExtractor εξάγει πληροφορίες που αφορούν την αναπαράσταση ενός πόρου. Οι πληροφορίες αυτές μπορούν να αποθηκεύονται σε ζευγάρια, όπως για παράδειγμα γίνεται στο JSON, ή σε μορφή RDF. Η τελευταία μορφή

είναι προτιμότερη δεδομένου του ότι το RDF επιτρέπει καλύτερα την κατανόηση οντοτήτων ακόμα και μεταξύ υπηρεσιών που δεν έχουν σχέση η μία με την άλλη.

2. **Πάροχος γνώσης**. Αφορά την γνώση γύρω από τις ενέργειες που επιτρέπει μια υπηρεσία. Η γνώση αυτή περιέχει πληροφορίες για την κάθε αναπαράσταση ξεχωριστά και τις ενέργειες που επιτρέπονται πάνω σε αυτήν. Η οργάνωση αυτής της γνώσης θα πρέπει να γίνει με τέτοιο τρόπο ώστε να είναι εύκολο δεδομένων των ιδιοτήτων μιας αναπαράστασης να μπορεί η υπηρεσία να αποφασίσει εύκολα ποιες διεπαφές (APIs) επιτρέπουν ενέργειες πάνω σε μία αναπαράσταση. Το δομικό στοιχείο υπεύθυνο για αυτό είναι το APICatalog το οποίο είναι απλά μια διεπαφή (interface) και επαφίεται στην ευχέρεια του προγραμματιστή για την υλοποίηση.

3. **Επιλογές χρήστη**. Ο PreferenceManager συλλέγει όλη την πληροφορία που αφορά το χρήστη και τις επιλογές του και δρα ως φίλτρο στο προαναφερθέν APICatalog, διαλέγοντας μόνο τις κατάλληλες διεπαφές (APIs) και τις ταξινομεί κατάλληλα με βάση το πόσο σχετικές είναι για το συγκεκριμένο χρήστη..

4. **Γεννήτρια ενεργειών**. Ο ActionProvider παράγει όλες τις ενέργειες με βάση τις επιλογές του χρήστη. Κάθε ενέργεια έχει άμεση σχέση με κάποιον πόρο της υπηρεσίας μέσα σε μία αναπαράσταση.

5. **Ενσωμάτωση affordances**. Τέλος ο RepresentationEnricher είναι το στοιχείο που μετατρέπει τις ενέργειες σε affordances και τα προσθέτει στην αναπαράσταση.

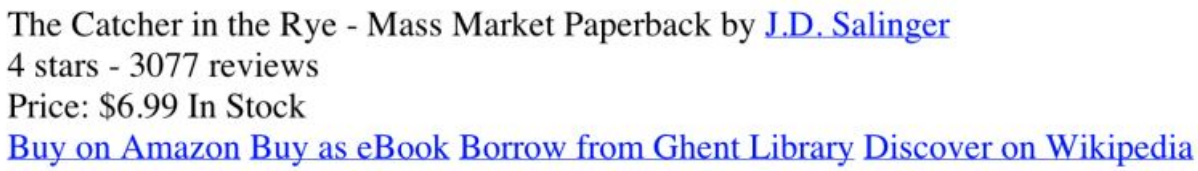
Όσον αφορά την ανάπτυξη ενός τέτοιου συστήματος, δύο είναι οι πιθανές λύσεις:

1. Βασισμένο στον πελάτη (Client-based). Η λύση αυτή αφορά την ανάπτυξη κώδικα πελάτη ο οποίος θα λειτουργεί ως πρόσθετο (πχ πρόσθετο περιηγητή) και κάθε φορά που ο πελάτης παίρνει απάντηση με τη μορφή αναπαράστασης πόρου σε προηγούμενο ερώτημά του θα ενεργοποιείται το πρόσθετο και θα προσθέτει τα αντίστοιχα affordances.

2. Ως υπηρεσία παροχής affordances (Affordance as a service). Το πλεονέκτημα αυτής της μεθόδου έναντι της προηγούμενης είναι ότι ο χρήστης δεν είναι απαραίτητο να χρησιμοποιεί συμβατά προγράμματα πελάτη για να μπορεί να έχει τα επιπλέον affordances σε μία αναπαράσταση. Τα affordances θα παρέχονται από την ίδια την υπηρεσία.

2.4.3 Παράδειγμα distributed affordances

Παρακάτω παρατίθεται ένα παράδειγμα που εφαρμόζει τα παραπάνω και παρέχει distributed affordances.



Σχήμα 2.4: Μια αναπαράσταση με distributed affordances

Στο παραπάνω παράδειγμα οι σύνδεσμοι (μπλε) αποτελούν affordances που μπορούν να ενεργοποιηθούν και έχουν σχέση με το περιεχόμενο της αναπαράστασης που είναι ένα μυθιστόρημα του J.D. Salinger.

2.4.4 Παρατηρήσεις

Η χρησιμότητα των affordances είναι σημαντική αν αναλογιστεί κανείς ότι δεν επιβάλλουν κάποιο συγκεκριμένο πρότυπο. Μπορούν να υπάρχουν σε οποιοδήποτε μέσο και πρότυπο αναπαράστασης (HYDRA, JSON, XML). Σημαντικό επίσης είναι το γεγονός ότι η λογική πίσω από τα distributed affordances, όχι μόνο συμβαδίζει με τη λογική του REST και πιο συγκεκριμένα του HATEOAS, αλλά την ενισχύει και την επεκτείνει, προσφέροντας ενέργειες που έχουν να κάνουν με οτιδήποτε αφορά κάποιον χρήστη, πέρα από τη δυνατότητα να εξερευνήσει μια υπηρεσία μέσω συνδέσμων.

2.5 Μοντελοποίηση Συστημάτων Λογισμικού

2.5.1 MOF

Το MOF [13] ή αλλιώς Meta Object Facility είναι ένα πρότυπο που δημιουργήθηκε από το Object Management Group (OMG) [14] για την ανάπτυξη λογισμικού με βάση τα μοντέλα. Η μοντελοποίηση των δεδομένων έχει πρακτικά οφέλη αφού μπορούμε να αποθηκεύσουμε δεδομένα χωρίς να χάσουν την βασική ιδιότητά τους που αφορά την περιγραφή ενός πόρου/αντικειμένου ανεξάρτητα από το μέσο περιγραφής. Επίσης ιδιαίτερη χρησιμότητα έχει και στο REST, αφού σημαντικό κομμάτι είναι η δυνατότητα επικοινωνίας διαφορετικών υπηρεσιών μεταξύ τους με βάση κάποιο κοινό μοντέλο, όπως για παράδειγμα αναφέραμε πριν με το RDF.

Το MOF χωρίζει τα μοντέλα σε τέσσερα επίπεδα:

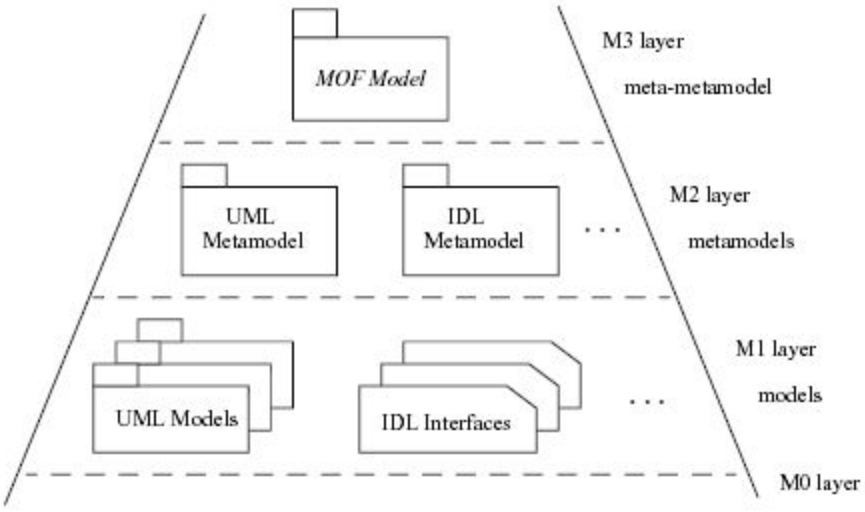
1. M3. Αυτό το επίπεδο αφορά τα μετά-μεταμοντέλα και ουσιαστικά χρησιμοποιείται για να περιγράψει τα μεταμοντέλα (επίπεδο M2). Είναι το ανώτερο επίπεδο που υπάρχει και μπορεί να περιγράψει γλώσσες μοντέλων, όπως για παράδειγμα η UML. Σε αυτό το επίπεδο βρίσκονται MOF μοντέλα και για αυτό μπορούμε να πούμε ότι το ίδιο το MOF μοντέλο μπορεί να χρησιμοποιηθεί για να περιγράψει τον εαυτό του, δηλαδή το ίδιο το MOF.

2. M2. Αυτό το επίπεδο αφορά όλα τα μεταμοντέλα και τις γλώσσες περιγραφής μοντέλων (πχ UML)

3. M1. Αυτό είναι το επίπεδο των μοντέλων το οποίο περιγράφει ουσιαστικά ό,τι υπάρχει στο επίπεδο M0.

4. M0. Το επίπεδο αυτό είναι το τελευταίο επίπεδο και περιέχει όλη την πληροφορία, τα αντικείμενα και γενικά οτιδήποτε χρειάζεται μοντελοποίηση.

Παρακάτω παραθέτουμε το αντίστοιχο σχήμα των τεσσάρων επιπέδων:



Σχήμα 2.5: Τα επίπεδα του MOF.

Ένα βασικό πλεονέκτημα του MOF είναι ότι προσφέρει τη δυνατότητα στον προγραμματιστή να χρησιμοποιήσει αφαιρετική λογική δημιουργώντας μοντέλα και στη συνέχεια να τα μεταφράσει σε κάποια γλώσσα, όπως η XML. Ιδιαίτερα χρήσιμα εργαλεία για κάτι τέτοιο είναι τα XMI και EMF, που προσφέρουν στον προγραμματιστή τη δυνατότητα να παράξει εκτελέσιμο κώδικα από ένα μοντέλο.

2.5.2 XMI

Το XMI [15] ή αλλιώς XML Metadata Interchange είναι ένα πρότυπο της OMG που έχει πιστοποιηθεί για ανταλλαγή μεταδεδομένων μέσω της Extensible Markup Language (XML). Μπορεί να χρησιμοποιηθεί για κάθε μεταδεδομένο του οποίου το μεταμοντέλο μπορεί να αναπαρασταθεί στο Meta Object Facility (MOF). Η πιο κοινότυπη χρήση του XMI είναι σαν σχήμα ανταλλαγής για UML μοντέλα.

Επίσης μπορεί να χρησιμοποιηθεί για την σειριοποίηση (serialization) μοντέλων άλλων γλωσσών (μεταμοντέλα). Σύμφωνα με το OMG, τα δεδομένα χωρίζονται σε αφηρημένα μοντέλα και σε συγκεκριμένα μοντέλα (concrete models). Τα αφηρημένα μοντέλα χρησιμοποιούνται για να περιγράψουν την σημασιολογία της πληροφορίας, ενώ τα συγκεκριμένα μοντέλα αναπαριστούν εποπτικά διαγράμματα. Αφηρημένα μοντέλα είναι στιγμιότυπα αυθαίρετων γλωσσών μοντελοποίησης που βασίζονται στο MOF όπως για παράδειγμα η UML.

Μέχρι σήμερα υπάρχουν πολλές ασυμβατότητες μεταξύ διάφορων εργαλείων μοντελοποίησης που επιτρέπουν την εξαγωγή διαγραμμάτων σε XMI, ακόμη και μεταξύ ανταλλαγών αφηρημένων δεδομένων. Δυστυχώς, αυτό σημαίνει ότι η ανταλλαγή αρχείων μεταξύ εργαλείων χρησιμοποιώντας το XMI, σπάνια είναι δυνατή. Ο οργανισμός OMG επιδιώκει, με τον ορισμό του XMI, να υλοποιήσει την εύκολη ανταλλαγή μεταδεδομένων μεταξύ εργαλείων

μοντελοποίησης που βασίζονται στην γλώσσα μοντελοποίησης UML και αποθηκών μεταδεδομένων που βασίζονται στο MOF σε κατανεμημένα ετερογενή περιβάλλοντα. Η XMI χρησιμοποιείται επίσης ως ο μεσολαβητής με τον οποίο μοντέλα διέρχονται από εργαλεία μοντελοποίησης σε εργαλεία παραγωγής λογισμικού ως μέρος της μοντελοκεντρικής μηχανικής.

2.5.3 EMF

Το EMF [29] ή αλλιώς Eclipse Modeling Framework είναι ένα έργο ανοικτού κώδικα που χρηματοδοτείται και βρίσκεται υπό την επίβλεψη της IBM. Αποτελεί μία προσέγγιση στον χώρο της τεχνολογίας λογισμικού με την οποία δημιουργούνται συστήματα λογισμικού που υπακούν στον ορισμό της μοντελοκεντρικής αρχιτεκτονικής.

Το EMF δίνει στον σχεδιαστή την δυνατότητα να σχεδιάσει μοντέλα που περιγράφουν ένα σύστημα λογισμικού. Δεδομένου ότι ο πηγαίος κώδικας είναι ένα μοντέλο περιγραφής του συστήματος, τα UML διαγράμματα είναι επίσης μοντέλα περιγραφής του συστήματος όπως και τα XML σχήματα, το EMF προσπαθεί να ενοποιήσει τις υπάρχουσες τεχνικές μοντελοποίησης και να δώσει στον μηχανικό λογισμικού την δυνατότητα να ορίσει το σύστημα στο μοντέλο με το οποίο αυτός είναι πιο εξοικειωμένος αλλά ταυτόχρονα να έχει και την περιγραφή των άλλων μοντέλων.

Το EMF στηρίζεται σε δύο ευρέως διαδεδομένα πρότυπα του OMG, το MOF και το XMI τα οποία περιγράψαμε παραπάνω. Μέσω του μοντέλου MOF, με την βοήθεια του EMF, μπορεί να περιγράψει κανείς ένα δομημένο και ιεραρχικό μοντέλο δεδομένων που σκοπό έχει να αποτελέσει την προδιαγραφή των δεδομένων που χρησιμοποιεί το σύστημα λογισμικού ή ακόμα και την προδιαγραφή του ίδιου του συστήματος λογισμικού. Έπειτα το EMF μπορεί να δημιουργήσει αυτόματα κώδικα, ο οποίος μπορεί να δημιουργήσει στιγμιότυπα των μοντέλων που ορίστηκαν σε MOF είτε με την βοήθεια του ίδιου του EMF είτε με κάποιο άλλο εργαλείο και εισήχθησαν στο EMF με την μορφή XMI.

Το EMF χρησιμοποιεί το πρότυπο XMI ως μέσο ανταλλαγής MOF μοντέλων μεταξύ εφαρμογών και ως μέσο αρχικοποίησης και αποθήκευσης MOF μοντέλων στον κώδικα της εφαρμογής. Το EMF προσφέρει κλάσεις εκτενών λειτουργιών στο EMF API, την διεπαφή προγραμματισμού που προσφέρει το EMF στους χρήστες, για καλύτερη χρήση των προτύπων MOF και XMI.

Στα πλαίσια αυτής της διπλωματικής το EMF μπορεί να χρησιμοποιηθεί για να περιγράψει μοντέλα πόρων REST αρχιτεκτονικής. Δεδομένου ότι υλοποιήσαμε μια RESTful υπηρεσία η οποία προσαρμόζεται σε άλλες παρόμοιες υπηρεσίες και προσθέτει affordances σε αναπαραστάσεις πόρων με δυναμικό τρόπο, το EMF προσφέρει δυνατότητα μοντελοποίησης στους πόρους των εκάστοτε υπηρεσιών βοηθώντας τες να “καταλαβαίνουν” η μία την άλλη χωρίς να χρειάζεται να μοιράζονται κοινή δομή στη βάση δεδομένων τους ή ακόμα και στις ίδιες τις αναπαραστάσεις των πόρων που ανταλλάσσουν.

Το EMF είναι το πιο διαδεδομένο framework για τους σκοπούς μας, ωστόσο θα μπορούσαν να χρησιμοποιηθούν εναλλακτικά εργαλεία όπως το Kevoree Modeling Framework (KMF) [30][31].

2.6 Δέντρα στόχων (Goal Models)

Ένας σημαντικός κλάδος της Τεχνολογίας Λογισμικού είναι η Μηχανική Απαιτήσεων. Ο κλάδος αυτός ασχολείται με την συλλογή και την ανάλυση απαιτήσεων. Η συλλογή και ανάλυση απαιτήσεων αποτελεί ένα σημαντικό στάδιο στην διαδικασία ανάπτυξης λογισμικού. Οι μηχανικοί λογισμικού πρέπει να προσδιορίσουν τις απαιτήσεις του συστήματος που φτιάχνουν έτσι ώστε να εξασφαλίσουν την καλύτερη δυνατή ποιότητα του. Ανεπαρκείς, ατελείς, διφορούμενες ή ασυνεπείς απαιτήσεις λογισμικού έχουν πολύ μεγάλη επίδραση στην ποιότητα του λογισμικού που παράγεται.

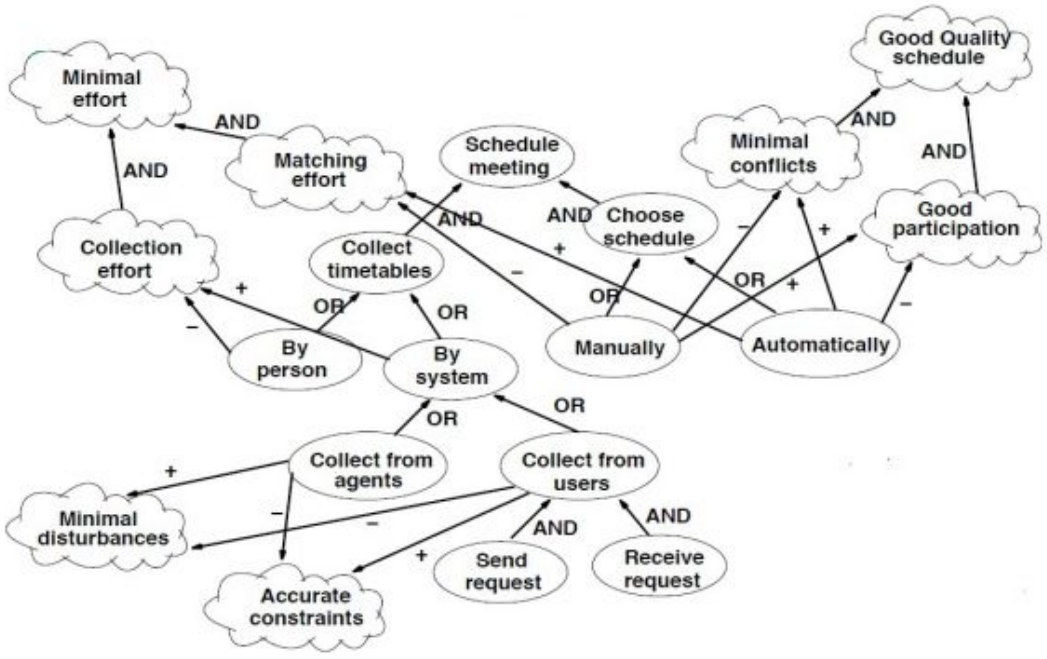
Μια προσέγγιση στην Μηχανική Απαιτήσεων είναι η προσανατολισμένη σε στόχους Μηχανική Απαιτήσεων. Στα [32] και [33] εξηγούνται τα πλεονεκτήματα αυτής της προσέγγισης. Ενδεικτικό του πόσο δημοφιλής είναι η Μηχανική Απαιτήσεων αποτελεί η πληθώρα μεθοδολογιών που έχει προταθεί για την προσανατολισμένη σε στόχους Μηχανική Απαιτήσεων πχ τα KAOS [34], GRL [35], GBRAM [36], TROPOS [37], CREWS [38], i* [39] κτλ.

Ένα σημαντικό μέρος της Μηχανικής Απαιτήσεων αποτελεί η μοντελοποίηση των απαιτήσεων. Για την μοντελοποίηση των λειτουργικών και μη απαιτήσεων ενός συστήματος λογισμικού έχει προταθεί πληθώρα μεθόδων, ενώ έχουν αναπτυχθεί και αρκετά εργαλεία που στηρίζονται στις μεθόδους αυτές. Πρωτοεισήχθησαν στον χώρο της μηχανικής και μοντελοποίησης λογισμικού στο [40] και χρησιμοποιούνται για την μοντελοποίηση των στόχων στο TROPOS που αναφέρεται παραπάνω.

Τα Δέντρα Στόχων μοντελοποιούν τους στόχους σε μια δενδρική δομή αποδομώντας κάθε στόχο σε υποστόχους η επαλήθευση των οποίων οδηγεί στην επαλήθευση του αρχικού. Σύμφωνα με αυτό το μοντέλο ένας στόχος αναλύεται σε υποστόχους οι οποίοι αναπαρίστανται σαν παιδιά του πρώτου στην δενδρική δομή. Το ίδιο ισχύει και για κάθε έναν από τους στόχους-παιδιά. Η αποσύνθεση ενός στόχου μπορεί να είναι διαφόρων τύπων και ο κάθε τύπος υποδηλώνει τον τρόπο επαλήθευσης του στόχου-πατέρα από τους στόχους-παιδιά. Για τις ανάγκες της διπλωματικής περιοριζόμαστε μόνο σε αποσυνθέσεις τύπου ΚΑΙ ή Ή. Ο τύπος ΚΑΙ όπως υποδηλώνει και το όνομά του, συνδέει τους στόχους παιδιά με την λογική σύζευξη. Ως εκ τούτου ο στόχος-πατέρας επαληθεύεται αν και μόνο αν επαληθεύονται όλοι οι στόχοι- παιδιά. Αντίστοιχα η αποσύνθεση Ή ενός στόχου συνδέει τους στόχους-παιδιά με την λογική διάζευξη.

Δηλαδή επαληθεύει τον στόχο-πατέρα αν τουλάχιστον ένας από τους στόχους-παιδιά επαληθεύεται.

Τέτοιου είδους στόχοι αναφέρονται στην βιβλιογραφία ως hard goals. Οι στόχοι αυτοί ορίζουν ένα σύνολο υποστόχων, με βάση το οποίο υπάρχει μονοσήμαντη σημασιολογία επαλήθευσης του στόχου-πατέρα και του συνόλου στόχων-παιδιών. Υπάρχουν όμως και περιπτώσεις απαιτήσεων, στην μοντελοποίηση απαιτήσεων λογισμικού, όπου ένας στόχος μπορεί να μην επαληθεύεται μονοσήμαντα από ένα σύνολο υποστόχων ένα είδος θετικής ή αρνητικής συνεισφοράς στον στόχο αυτό από άλλους στόχους. Ανάλογα με το ποσοστό της θετική έναντι της αρνητικής συνεισφοράς, μπορεί να επαληθεύεται ή όχι ο στόχος. Τέτοιου είδους στόχοι αναφέρονται ως soft goals. Παρακάτω φαίνεται ένα παράδειγμα ενός δέντρου στόχων όπου συνυπάρχουν τόσο hard goals και soft goals.



Σχήμα 2.6: Παράδειγμα δέντρου στόχων

Στο παραπάνω διάγραμμα τα hard goals παρουσιάζονται με τα οβάλ ενώ τα soft goals με τα σύννεφα. Φαίνεται ξεκάθαρα η δενδρική δομή και οι αποσυνθέσεις των κόμβων καθώς επίσης και οι συνεισφορές των στόχων σε κάθε soft goal. Οι θετικές συνεισφορές παρουσιάζονται με το σύμβολο + (συν) ενώ οι αρνητικές με το σύμβολο - (μείον).

Τα ΚΑΙ/Ή Δέντρα Στόχων προσφέρουν έναν εύκολο τρόπο αναπαράστασης των στόχων που ήταν πολύ χρήσιμος για την μοντελοποίηση δεδομένων που χρησιμοποιεί το περιβάλλον πλαίσιο που προτείνουμε σε αυτήν την διπλωματική. Μπορούν εύκολα να κατασκευαστούν και να προσπελαστούν προγραμματιστικά. Επιπροσθέτως η άμεση σύνδεση τους με την άλγεβρα

Boole και την λογική πρώτης τάξης, καθιστά τα δέντρα στόχων μια ελκυστική λύση καθώς μετατρέπονται εύκολα σε ένα σύνολο κανόνων λογικής πρώτης τάξης. Αυτοί οι κανόνες στην συνέχεια μπορούν να χρησιμοποιηθούν από έτοιμα εργαλεία που υλοποιούν αλγορίθμους λογικής πρώτης τάξης. Έτσι τα δέντρα στόχων έρχονται με μια πληθώρα εργαλείων και αλγορίθμων που εφαρμόζονται στην άλγεβρα Boole και μας είναι άμεσα διαθέσιμα. Επιπλέον τα δέντρα στόχων είναι εύκολα επεκτάσιμα καθώς επιτρέπουν την επισύναψη οποιασδήποτε πληροφορίας πάνω στους κόμβους-στόχους με την μορφή επισημειώσεων ή ακόμα και την επέκταση του μοντέλου δέντρων στόχων με επιπλέον σχέσεις μεταξύ των κόμβων στόχων.

Στα πλαίσια αυτής της διπλωματικής χρησιμοποιήσαμε τα δέντρα στόχων για να επιτύχουμε εξατομίκευση όσον αφορά τα affordances. Χρησιμοποιήθηκαν δηλαδή δέντρα στα οποία δώσαμε μοναδικά affordances, ενώ οι κόμβοι των δέντρων χρησιμοποιήθηκαν για να περιγράψουν τις προϋποθέσεις που πρέπει να ισχύουν για να χρησιμοποιηθεί ένα affordance και κατά πόσο είναι σχετικό με το ερώτημα ενός χρήστη για ένα συγκεκριμένο πόρο.

Κεφάλαιο 3: Αρχιτεκτονική του Συστήματος

3.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζεται η δομή της αρχιτεκτονικής του συστήματος το οποίο καλείται να προσθέσει επιπλέον περιεχόμενο (affordances) σε REST απαντήσεις.

Ένα από τα βασικότερα ζητήματα όταν σχεδιάζουμε ένα σύστημα και κατ' επέκταση την αρχιτεκτονική του είναι να λάβουμε υπ' όψιν ότι αυτό το σύστημα πρέπει να είναι επεκτάσιμο και εύκολα προσαρμόσιμο σε νέες ανάγκες. Αυτό σημαίνει ότι το σύστημα θα πρέπει να περιέχει απλές και αυτόνομες λειτουργικές μονάδες (components). Ένας τρόπος επίσης για να επιτευχθεί κάτι τέτοιο είναι να δημιουργήσουμε ένα περιβάλλον πλαίσιο (framework) και στη συνέχεια να χρησιμοποιήσουμε αυτό το πλαίσιο στην REST υπηρεσία που θέλουμε να κατασκευάσουμε.

Στο πλαίσιο αυτής της διπλωματικής δημιουργήθηκε ένα framework για αυτό το σκοπό. Δόθηκε έμφαση ώστε αυτό να είναι όσο το δυνατόν πιο γενικό για να μπορεί να χρησιμοποιηθεί και για άλλες χρήσεις, πέραν αυτής της REST υπηρεσίας. Στην ουσία οποιαδήποτε εφαρμογή περιλαμβάνει affordances και goal models θα μπορούσε να χρησιμοποιήσει το framework αυτό ή ακόμα και να το επεκτείνει. Στη συνέχεια δημιουργήθηκε η REST υπηρεσία, που είναι και το ζητούμενο αυτής της διπλωματικής, χρησιμοποιώντας το προαναφερθέν framework.

Σε αυτό το σημείο θα πρέπει να αναλύσουμε κάποιες βασικές έννοιες που χρησιμοποιήθηκαν για την διευκόλυνση της υλοποίησης. Πρώτη έννοια είναι το ίδιο το affordance. Το affordance στη συγκεκριμένη υλοποίηση είναι πρακτικά ένας σύνδεσμος που εμπλουτίζει μια REST απάντηση και γίνεται ανάθεσή του σε ένα goal model. Όταν διαλέξουμε το πιο κατάλληλο goal model με βάση το πόσο ικανοποιήσιμο είναι, τότε στην ουσία διαλέγουμε το affordance που έχει δηλωθεί για το συγκεκριμένο goal model. Τα φύλλα του κάθε δέντρου μπορούν να περιέχουν αριθμούς, συμβολοσειρές, ημερομηνίες, προτιμήσεις του χρήστη κλπ. Κατά την υλοποίηση του framework τα goal models μπορούν να περιέχουν οτιδήποτε, όχι απαραίτητα affordances. Αυτό βοηθάει στο να είναι εύκολα προσαρμόσιμο στης ανάγκες οποιουδήποτε θέλει να το χρησιμοποιήσει.

Μία άλλη σημαντική έννοια είναι αυτή του context. Το context, ή AffordanceContext όπως παρουσιάζεται σε αυτή τη διπλωματική, είναι ουσιαστικά το περιεχόμενο και οι μετα-πληροφορίες που αφορούν το affordance. Όταν η υπηρεσία μας δεχθεί την προς εμπλούτιση απάντηση μιας εξωτερικής, τρίτης, υπηρεσίας, θα πρέπει να αποφασίσει ποια μοντέλα θα πρέπει να χρησιμοποιηθούν και να υπολογισθούν. Ο λόγος που γίνεται αυτό είναι γιατί όλα τα μοντέλα δεν έχουν "νόημα" σε όλα τα ερωτήματα. Κάποια μοντέλα αφορούν

συγκεκριμένα ερωτήματα. Έτσι χρησιμοποιήσαμε την έννοια του AffordanceContext σαν ένα περίβλημα για την πληροφορία που αφορά τα Affordances, καθώς και την ίδια την απάντηση. Έτσι όταν η υπηρεσία δέχεται την απάντηση μπορεί να γνωρίζει ποια είναι τα κατάλληλα goal models και affordances που θα πρέπει να λάβει υπ' όψιν.

Η λειτουργία αυτή της υπηρεσίας μπορεί να περιγραφεί σε μερικά απλά βήματα:

1. Λήψη REST ερωτήματος.
2. Έλεγχος ταυτότητας του χρήστη για το συγκεκριμένο ερώτημα
3. Αν ο χρήστης είναι διαχειριστής, τότε πρέπει να γίνει χρήση της διεπαφής του διαχειριστή
4. Αν είναι απλός χρήστης τότε προώθησε το ερώτημα στην εξωτερική υπηρεσία.
5. Όταν γίνει λήψη της απάντησης από την εξωτερική υπηρεσία χρησιμοποίησε το AffordanceContext για να βρεθούν όλα τα κατάλληλα μοντέλα.
6. "Τρέξε" τα μοντέλα και βρες την ικανοποιησιμοτητά τους.
7. Πρόσθεσε τα αντίστοιχα affordances στην REST απάντηση και στείλε την στον χρήστη.

Με βάση τα παραπάνω είναι δυνατός ο προσδιορισμός μερικών απαραίτητων components για την υλοποίηση. Παρακάτω παραθέτουμε μερικά απαραίτητα components:

- Ένα component για την υποδοχή ερωτημάτων και αποστολή απαντήσεων προς το χρήστη.
- Ένα component υπεύθυνο για την ασφάλεια και την ταυτοποίηση των χρηστών.
- Ένα component για την μοντελοποίηση του AffordanceContext.
- Ένα component υπεύθυνο για την προσθήκη affordances σε ένα resource.
- Ένα component υπεύθυνο για την επίλυση της ικανοποιησιμότητας ενός goal model.

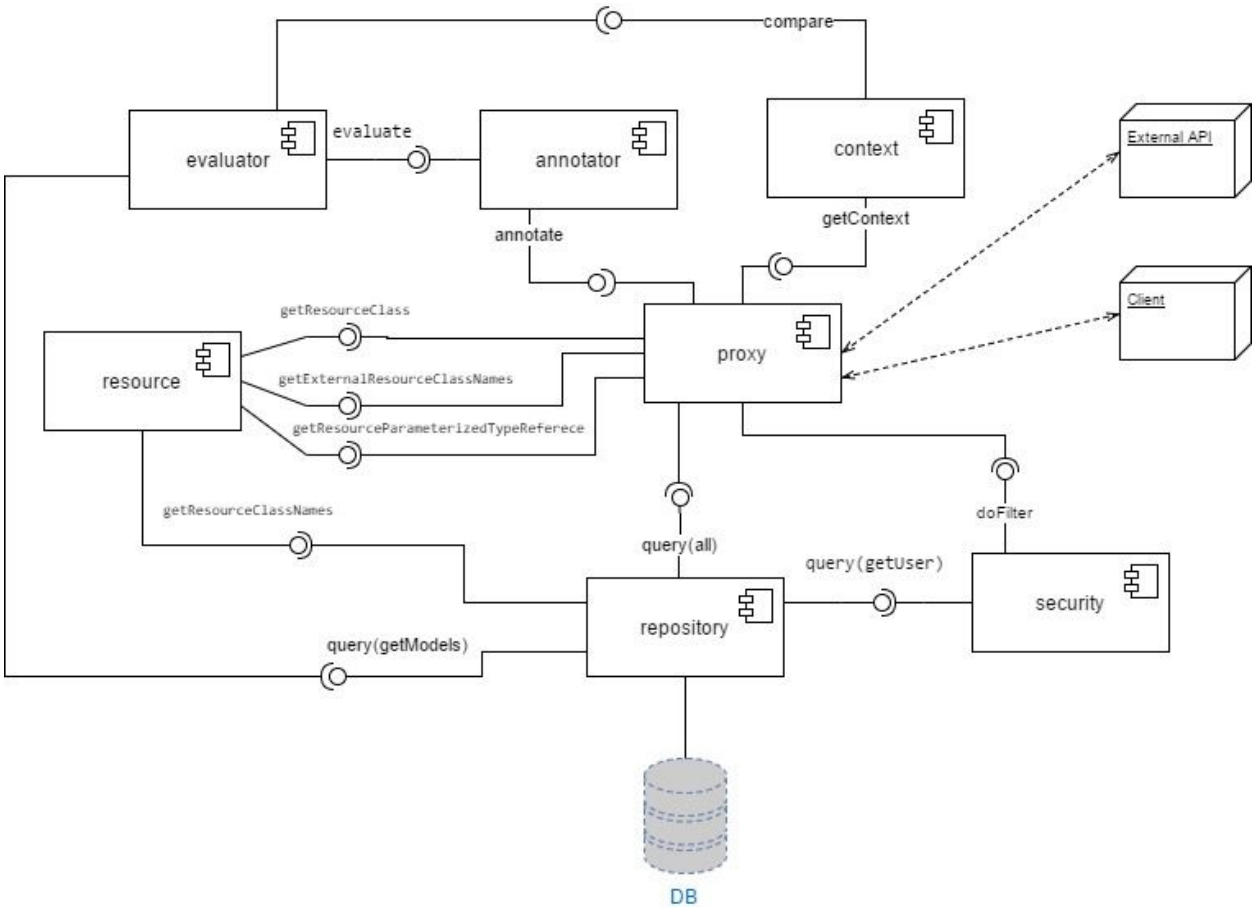
Επιπλέον χρειαζόμαστε και κάποια components για κάποιες βασικές λειτουργίες του συστήματος:

- Για την επικοινωνία με τη βάση δεδομένων.
- Για τη μοντελοποίηση των REST resources.

Στις επόμενες ενότητες που ακολουθούν γίνεται η περιγραφή των components του συστήματος με διαγράμματα, καθώς και αναλύεται το κάθε component ξεχωριστά. Στο τέλος παρατίθενται ακολουθιακά διαγράμματα.

3.2 Επισκόπηση της αρχιτεκτονικής

Με βάση τα παραπάνω μπορούμε τώρα να ορίσουμε το ψηφιδικό διάγραμμα (component diagram) της αρχιτεκτονικής του συστήματος και να αναλύσουμε το κάθε component ξεχωριστά



Σχήμα 3.1: Component diagram του συστήματος

Με βάση το παραπάνω διάγραμμα διακρίνονται τα εξής components:

- **proxy**: Αυτό είναι το βασικό component που αφορά την επικοινωνία με το χρήστη και τις εξωτερικές RESTful υπηρεσίες. Είναι ουσιαστικά η “καρδιά” του συστήματος και τα πάντα ξεκινάνε και καταλήγουν εκεί.

- **security**: Είναι το component υπεύθυνο για την ασφάλεια του συστήματος και την ταυτοποίηση του χρήστη. Κάθε ερώτημα πριν προχωρήσει πρώτα περνάει από αυτό το component για ταυτοποίηση.

- **repository**: Είναι το component υπεύθυνο για την επικοινωνία με τη βάση δεδομένων και τη μετατροπή των στοιχείων της βάσης σε δεδομένα κατάλληλα για επεξεργασία.

- **context**: Είναι το component υπεύθυνο για την μοντελοποίηση της λογικής περί AffordanceContext. Περιέχει τις κατάλληλες διεπαφές και τα κατάλληλα αντικείμενα για να περιγράψει την μετα-πληροφορία των affordances και κατ’ επέκταση των REST απαντήσεων/ερωτημάτων.

- **annotator**: Προσθέτει τα κατάλληλα affordances στις REST απαντήσεις. Επίσης περιέχει την κατάλληλη λογική για το ποια πολιτική θα ακολουθηθεί. Δηλαδή από τα “επιλυμένα” goal models διαλέγει ποια από αυτά (δηλαδή ποια affordances) θα προστεθούν στις απαντήσεις.

- **evaluator**: Είναι υπεύθυνο για την “επίλυση” (evaluation) των κατάλληλων goal models με βάση το AffordanceContext.

Στη συνέχεια αναλύονται περαιτέρω τα παραπάνω components.

3.3 Αναλυτική περιγραφή των components

3.3.1 proxy

Ο proxy είναι υπεύθυνος για την κεντρική λειτουργία του συστήματος. Η δουλειά του είναι να παραλαμβάνει ερωτήματα από τον πελάτη (client) και να τα επεξεργάζεται κατάλληλα. Υπάρχουν τρία σημεία επικοινωνίας με τον client τα οποία διαχειρίζονται τρεις controllers:

- **EntryController**: Διαχειρίζεται ερωτήματα στο path `/api` και ουσιαστικά παρέχει πληροφορίες για τις υποστηριζόμενες υπηρεσίες με HATEOAS τρόπο όπως περιγράφηκε στο κεφάλαιο 2.

- **WebController**: Διαχειρίζεται και παρέχει διεπαφές για τη διαχείρηση του συστήματος από τον διαχειριστή μέσω του path `/admin`. Οι διεπαφές αφορούν τη δημιουργία και επεξεργασία χαρτογραφήσεων των διάφορων paths (mappings), τη δημιουργία χρήστη και την δημιουργία και επεξεργασία goal models.

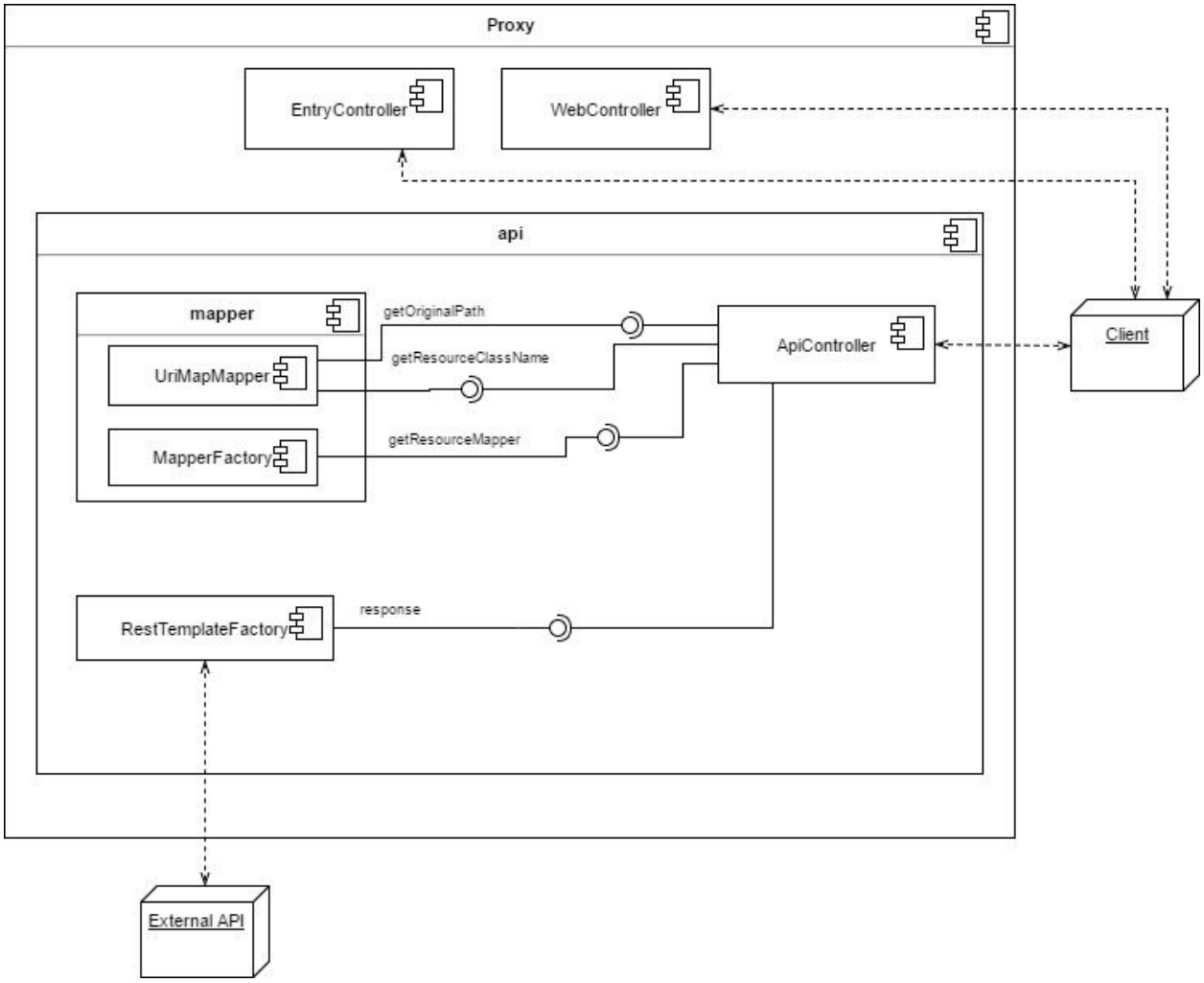
- **ApiController**: Ο βασικότερος controller που ουσιαστικά αναλαμβάνει την προώθηση των ερωτημάτων σε εξωτερικές RESTful υπηρεσίες, την παραλαβή της αντίστοιχης απάντησης και τον εμπλουτισμό αυτής με τα κατάλληλα affordances. Αφορά το path `/api/*`.

Υπάρχουν επίσης δύο κεντρικά components που αφορούν το api.

- **mapper**: Περιέχει το **MapperFactory** το οποίο μέσω της διεπαφής `getResourceMap` παρέχει τον κατάλληλο mapper για ένα συγκεκριμένο api ερώτημα. Ο mapper είναι υπεύθυνος για τη μετάφραση της εσωτερικής διεύθυνσης ενός ερωτήματος σε κατάλληλη εξωτερική διεύθυνση της τρίτης υπηρεσίας που θα προωθηθεί το ερώτημα. Επίσης περιέχει και το **UriMapMapper** που είναι στην ουσία η υλοποίηση του mapper παρέχοντας τις διεπαφές `getOriginalPath` και `getResourceClassName` για την εύρεση της πληροφορίας που χρειάζεται για να γίνει η μετάφραση και η προώθηση του μηνύματος στην εξωτερική υπηρεσία.

- **RestTemplateFactory**: Είναι το component το οποίο αναλαμβάνει την επικοινωνία με την εξωτερική υπηρεσία. Προσαρμόζει το ερώτημα κατάλληλα και αναλαμβάνει να παραλάβει την απάντηση να τη μετατρέψει σε δεδομένα κατάλληλα προς επεξεργασία και να τα επιστρέψει στον **ApiController** μέσω της διεπαφής `response`. Επίσης αναλαμβάνει όλη τη λειτουργία που αφορά σε διάφορες εξαιρέσεις, όπως για παράδειγμα την περίπτωση που η εξωτερική υπηρεσία επιστρέψει κάποιον κωδικό σφάλματος, ή δεν επιστρέψει τίποτα.

Παρακάτω παρατίθεται το σχήμα του proxy component.



Σχήμα 3.2: Component diagram του proxy

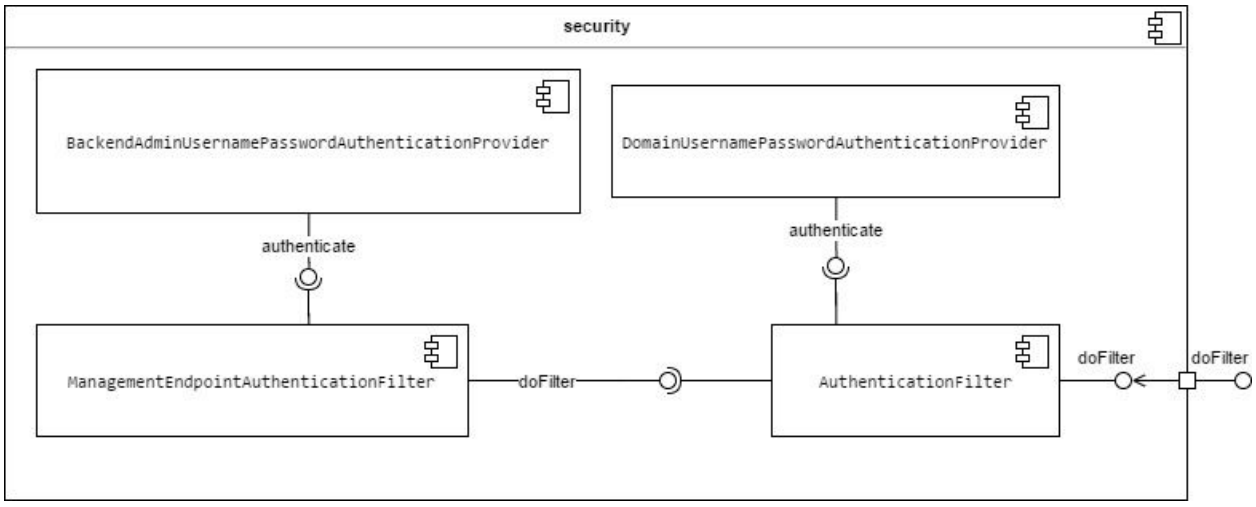
3.3.2 security

Το security component αφορά την ασφάλεια του συστήματος και την ταυτοποίηση του χρήστη όταν αυτός προσπαθεί να επικοινωνήσει με έναν από τους τρεις προαναφερθέντες controllers. Παρέχει μια σειρά από φίλτρα τα οποία διαβάζουν την πληροφορία του ερωτήματος και με τη χρήση των αντίστοιχων providers προσπαθούν να επαληθεύσουν τα διαπιστευτήρια (`authenticate`) του χρήστη. Σε περίπτωση που το ερώτημα δεν παρέχει τα κατάλληλα διαπιστευτήρια χρήστη το `doFilter` δεν επιστρέφει το ερώτημα, αλλά προκαλεί μια εξαίρεση συστήματος και επιστρέφεται στο χρήστη η κατάλληλη απάντηση. Διακρίνονται δύο φίλτρα και providers για δύο διαφορετικούς τύπους ερωτημάτων:

- **AuthenticationFilter** και **DomainUsernamePasswordAuthenticationProvider**: Αυτά τα δύο είναι υπεύθυνα για την επαλήθευση ενός απλού χρήστη που θέλει να χρησιμοποιήσει το path `/api` για να κάνει ερωτήματα μέσω της υπηρεσίας σε μια εξωτερική υπηρεσία και να του επιστραφεί απάντηση που περιέχει affordances. Αν το ερώτημα δεν αφορά χρήση αυτού του path, ή το ερώτημα δεν περιέχει τις κατάλληλες επικεφαλίδες (headers), `X-AUTH-USERNAME` και `X-AUTH-PASSWORD`, τότε το component προωθεί το ερώτημα προς φιλτράρισμα στα επόμενα δύο.

- **ManagementEndpointAuthenticationFilter** και **BackendAdminUsernamePasswordAuthenticationProvider**: Αυτά τα δύο είναι υπεύθυνα για τον διαχειριστή (admin) που θέλει να αλλάξει ρυθμίσεις στο σύστημα, όπως περιγράφηκε και παραπάνω στον **WebController**. Η επαλήθευση του διαχειριστή γίνεται χρησιμοποιώντας τον header για **Basic Authentication** και ο λόγος είναι γιατί η διεπαφή του διαχειριστή γίνεται μέσω περιηγητή ιστοσελίδων, οι οποίοι παρέχουν ενσωματωμένες λειτουργίες για διαχείρηση ερωτημάτων που χρησιμοποιούν αυτό το authentication. Σε περίπτωση που δεν περιέχεται Basic Authentication και το συγκεκριμένο ερώτημα αφορά διαχείρηση συστήματος, τότε επιστρέφεται κατάλληλη απάντηση με αντίστοιχο header WWW-Authenticate. Οπότε τα συγκεκριμένα δύο components αναλαμβάνουν την ταυτοποίηση ότι ένα ερώτημα προορίζεται για το path `/admin` και ότι έγινε από διαχειριστή.

Παρακάτω παρατίθεται το αντίστοιχο component diagram για το security.



Σχήμα 3.3: Component diagram του security

3.3.3 context

Το component context περιέχει όλη τη λειτουργικότητα για τη λογική πίσω από το AffordanceContext, όπως περιγράφηκε προηγουμένως.

Διακρίνονται τέσσερα σημαντικά υπό-components:

- **ContextFactory**: Χρησιμοποιείται για να επιστρέψει το κατάλληλο **AffordanceContext** που αφορά ένα συγκεκριμένο ερώτημα. Κάθε ξεχωριστό ερώτημα δεν είναι απαραίτητο ότι θα έχει την ίδια μετα-πληροφορία. Οπότε είναι λογικό ότι δεν θα έχει και το ίδιο AffordanceContext και για αυτό το λόγο χρησιμοποιούμε ένα **factory** για να παράγουμε πολλά διαφορετικά AffordanceContext, αναλόγως την περίσταση. Παρέχει την διεπαφή `getContext`.

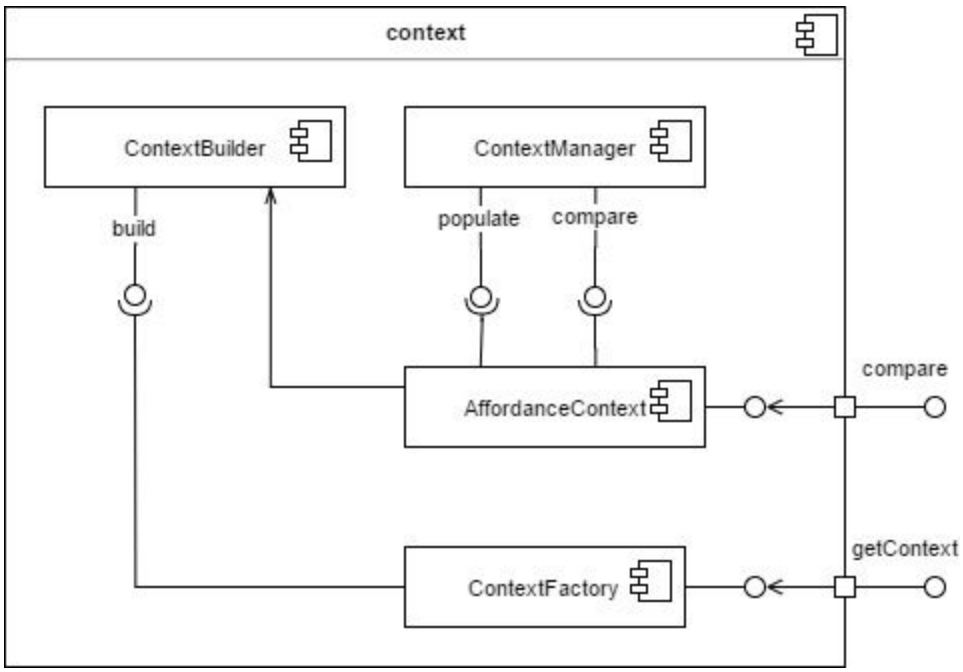
- **ContextBuilder**: Είναι διαπροσωπία και χρησιμοποιείται από το **ContextFactory** για να "χτίσει" (build) ένα **AffordanceContext**, αλλά στην διπλωματική παρέχονται διάφορες υλοποιήσεις της διαπροσωπίας. Περιέχει διάφορες διεπαφές, με βασικότερη το `build` και βοηθάει στην δημιουργία του AffordanceContext. Ο λόγος που δημιουργήσαμε ένα ξεχωριστό component για αυτή τη λειτουργία είνα διότι σε περίπτωση που κάποιος τρίτος θέλει να χρησιμοποιήσει το framework που αναπτύχθηκε, θα μπορεί να χρησιμοποιήσει και να επεκτείνει τον ContextBuilder χωρίς να είναι απαραίτητο να χρησιμοποιήσει το ContextFactory.

- **AffordanceContext**: Είναι το κεντρικό component που περιλαμβάνει να περιτυλίξει (wrap) όλη την μετα-πληροφορία που αφορά τα affordances, και τα ερωτήματα ώστε να μπορεί να γίνει εύκολα διαχωρισμός για το ποια είναι τα κατάλληλα goal models για συγκεκριμένα ερωτήματα. Περιέχει την πληροφορία μέσω ενός ζεύγους κλειδιού-τιμής (key-value). Το κλειδί αφορά ένα μοναδικό προσδιοριστικό μιας πληροφορίας, ενώ η τιμή αφορά την τιμή αυτής της πληροφορίας. Επειδή αυτή η τιμή θα πρέπει να είναι εύκολα συγκρίσιμη με τις αντίστοιχες τιμές/πληροφορίες των φύλλων ενός goal model το AffordanceContext περιέχει τους αντίστοιχους **ContextManagers**, όπως περιγράφονται παρακάτω. Παρέχει μια πληθώρα από διεπαφές, με τη βασικότερη όλων το `compare`.

- **ContextManager**: Το **AffordanceContext**, όπως αναφέρθηκε περιλαμβάνει όλες της μετα-πληροφορίες που αφορούν το Affordance. Κατά συνέπεια θα πρέπει να υπάρχει ένα component που θα μπορεί να διαχειρίζεται αυτήν την πληροφορία, να δίνει αρχικές τιμές σε αυτήν όταν πρωτοδημιουργείται το AffordanceContext, όπως επίσης και να παρέχει υποστήριξη για σύγκριση αυτής της πληροφορίας με την αντίστοιχη πληροφορία από τα φύλλα ενός goal model (δέντρου). Αυτή η λειτουργία καλύπτεται από τον ContextManager μέσω των διεπαφών `populate` και `compare`. Επίσης δημιουργήθηκε με

τέτοιο τρόπο, ώστε αν κάποιος τρίτος θέλει να επεκτείνει τη λειτουργία του για να καλύπτει τις δικές του ανάγκες να μπορεί να το κάνει μέσω του παρεχόμενου framework.

Αφού έγινε περιγραφή όλων των υπό-components του context, παρακάτω δίνεται το αντίστοιχο διάγραμμα.



Σχήμα 3.4: Component diagram του context

3.3.4 annotator

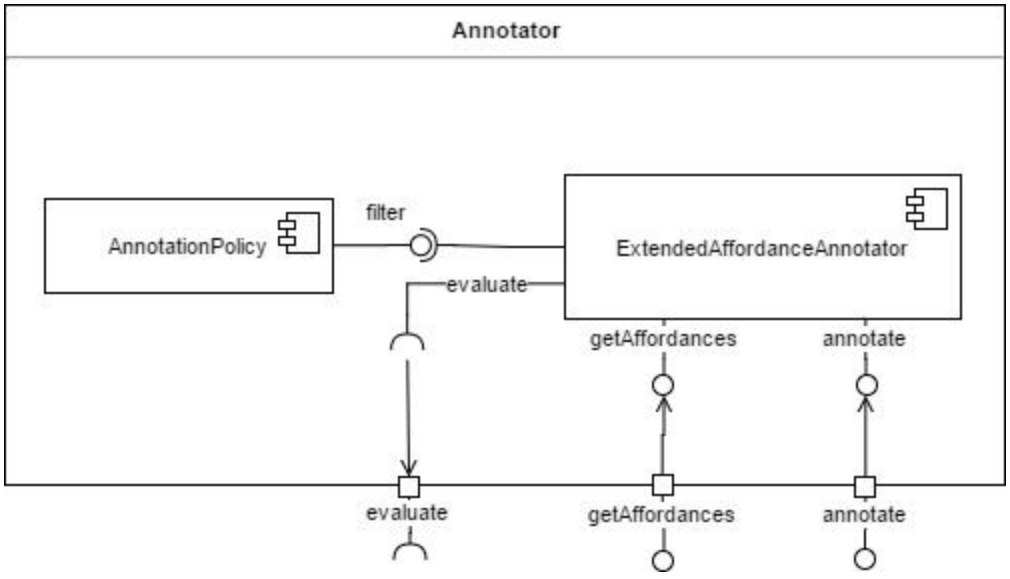
Είναι το βασικό component για την προσθήκη affordances (annotate) σε ένα REST resource. Όταν ο proxy παραλάβει την απάντηση από το ερώτημα που αρχικά είχε προωθηθεί, τότε αφού πάρει το κατάλληλο AffordanceContext χρησιμοποιεί τον annotator για να κάνει annotate την αντίστοιχη απάντηση. Διακρίνονται δύο βασικά υπό-components.

- **ExtendedAffordanceAnnotator**: Αποτελεί μια υλοποίηση/επέκταση της διαπροσωπίας **Annotator** (δεν περιγράφεται). Η δομή είναι τέτοια ώστε να μπορεί κάποιος τρίτος να επεκτείνει είτε αυτό το component, είτε την ίδια την διαπροσωπία για να δημιουργήσει τον δικό του Annotator. Παρέχει δύο διεπαφές. Την διεπαφή `annotate` που εμπλουτίζει ένα REST Resource και την διαπροσωπία `getAffordances` που επιστρέφει όλα τα κατάλληλα affordances. Επίσης κάνει χρήση της διεπαφής `evaluate` του **evaluator** που

ουσιαστικά επιστρέφει όλα τα κατάλληλα, με το **AffordanceContext**, δέντρα των οποίων η ικανοποιησιμότητα έχει υπολογιστεί.

- **AnnotationPolicy**: Αποτελεί μια διαπροσωπία, ωστόσο στη διπλωματική παρέχονται και αντίστοιχες υλοποιήσεις. Αυτό ουσιαστικά το component φιλτράρει όλα τα evaluations τα οποία επιστρέφονται μέσω της διεπαφής `evaluate` στον **ExtendedAffordanceAnnotator** και μέσω της διεπαφής `filter`, φιλτράρει και επιστρέφει ένα υποσύνολο των αρχικών evaluations από τα οποία στη συνέχεια εξάγονται τα affordances και επιστρέφονται μέσω του ExtendedAffordanceAnnotator. Ο χρήστης είναι ελεύθερος να υλοποιήσει δικά του AnnotationPolicies με βάση για παράδειγμα την υψηλότερη βαθμολογία ικανοποιησιμότητας, ή των δέκα καλύτερων ή ακόμα και όλων με βάση κάποιο κατώτατο όριο.

Παρακάτω δίνουμε το αντίστοιχο component diagram.



Σχήμα 3.5: Component diagram του annotator.

3.3.5 evaluator

Αποτελεί ένα από τα βασικότερα components του συστήματος και είναι υπεύθυνο για τον υπολογισμό της ικανοποιησιμότητας των goal models. Περιέχει τρία διαφορετικά components:

- **AffordanceEvaluator**: Είναι μια υλοποίηση της διαπροσωπίας **Evaluator** και παρέχει τη διεπαφή `evaluate` που εμφανίζεται σαν διεπαφή του **evaluator**. Στην ουσία χρησιμοποιεί τις διεπαφές `getModels` και `applyReasoning` του component **models** (αναλύεται παρακάτω) για να πάρει όλα τα κατάλληλα μοντέλα που συμβαδίζουν με το **AffordanceContext** και να βρει την ικανοποιησιμότητά τους αντίστοιχα.

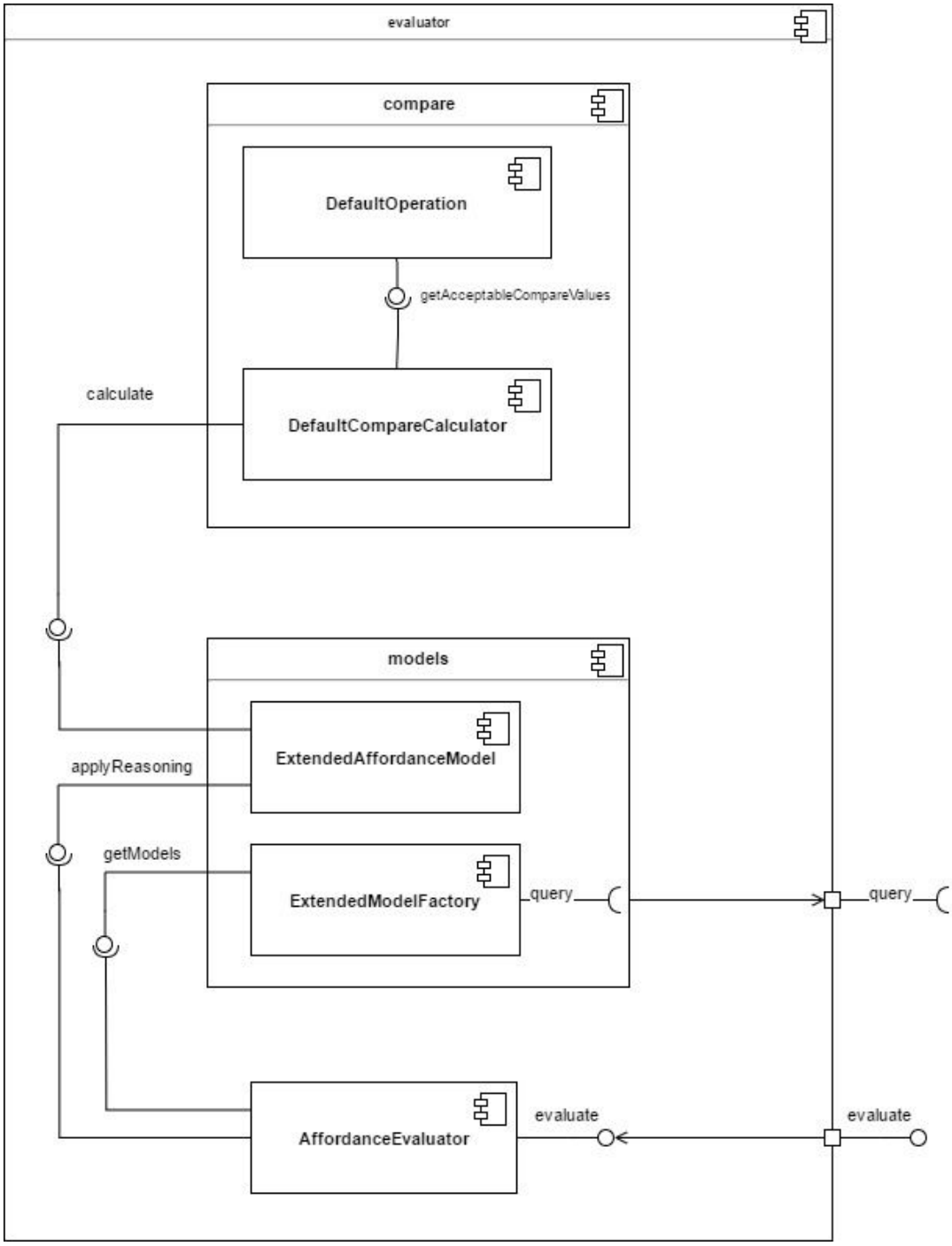
- **models**: Αφορά όλη τη λειτουργικότητα των goal models και περιέχει δύο διαφορετικά υπό-components:
    - **ExtendedAffordanceModel**: Επέκταση του **AffordanceModel** και της αφηρημένης κλάσης **Model** που περιέχει όλες τις πληροφορίες που αφορούν το δέντρο, όπως για παράδειγμα το affordance και τις τιμές των φύλλων καθώς και τη διεπαφή `applyReasoning` με την οποία μπορεί να υπολογιστεί η ικανοποιησιμότητά του.

    - **ExtendedModelFactory**: Υλοποίηση της διαπροσωπίας **ModelFactory** παρέχει τη διεπαφή `getModels` για την εύρεση όλων των σχετικών με το **AffordanceContext** goal models χρησιμοποιώντας τη διεπαφή `query` για να βρει όλα τα goal models από τη βάση δεδομένων.

- **compare**: Αφορά τη λειτουργία περί σύγκρισης, υπολογισμού και ανάθεσης τιμών στα φύλλα των goal models. Περιέχει δύο υπό-components.
    - **DefaultOperation**: Υλοποίηση της διεπαφής **Operation** που παρέχει τη διεπαφή `getAcceptableCompareValues` με την οποία επιστρέφει τις αποδεχόμενες τιμές των συγκρίσεων. Μπορεί να επεκταθεί για να υποστηρίζει μεγαλύτερο υποσύνολο συγκρίσεων.

    - **DefaultCompareCalculator**: Υλοποίηση της διαπροσωπίας **CompareCalculator** που παρέχει τη διεπαφή `calculate` με την οποία υπολογίζει τις τιμές στα φύλλα των goal models χρησιμοποιώντας το **DefaultOperation** όπως περιγράφηκε πριν.

Αναλυτικό διάγραμμα του evaluator διάγραμμα του evaluator δίνεται παρακάτω.



Σχήμα 3.6: Component diagram του evaluator

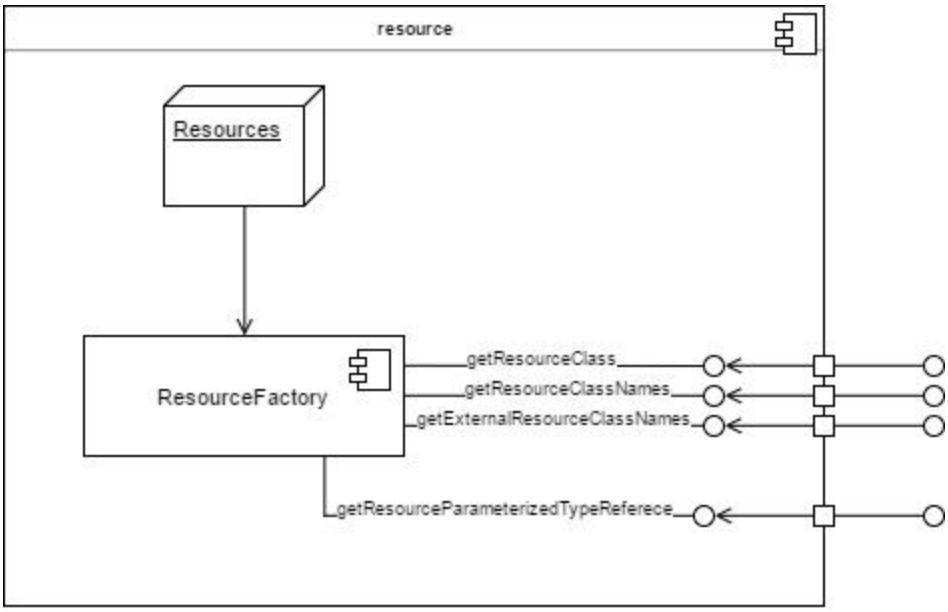
3.3.6 resource

Αυτό το component περικλείει όλη τη λειτουργικότητα γύρω από τα REST resources. Τα resources ουσιαστικά είναι αντικείμενα με βάση τον αντικειμενοστραφή προγραμματισμό τα οποία στέλνονται και λαμβάνονται από το REST ως απαντήσεις. Μοντελοποιούν δηλαδή τις REST απαντήσεις. Περιέχει ένα component και ένα πακέτο:

- **ResourceFactory**: Παρέχει τις τέσσερις διεπαφές (φαίνονται παρακάτω) που αφορούν τις κλάσεις και τα ονόματα των υποστηριζόμενων resources.

- **Resources**: Περιέχει όλα τα resources/απαντήσεις REST μοντελοποιημένα σε κλάσεις.

Παρακάτω δίνεται το αντίστοιχο διάγραμμα του resource component.



Σχήμα 3.7: Component diagram του resource

3.3.7 repository

Αυτό το component περικλείει ουσιαστικά όλη τη λειτουργικότητα που αφορά τη βάση δεδομένων. Αντί να κάνουμε τα συνήθη ερωτήματα για μια σχεσιακή βάση δεδομένων χρησιμοποιούμε αυτό το component για να παρέχει τα δεδομένα από τη βάση σε πιο κατάλληλη προς επεξεργασία μορφή. Περιέχει τέσσερα διαφορετικά components και ένα πακέτο:

- **GoalModelRepository**: Περιέχει διεπαφή για την εύρεση goal models από τη βάση δεδομένων.

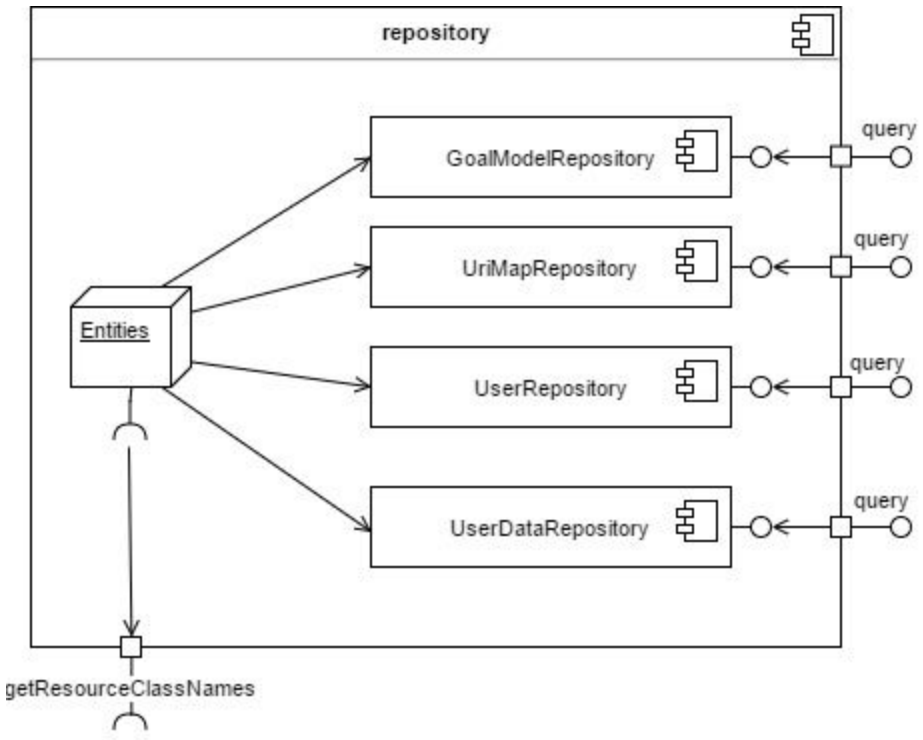
- **UriMapRepository**: Περιέχει διεπαφή για την εύρεση των mappings από τη βάση δεδομένων.

- **UserRepository**: Περιέχει τη διεπαφή για την εύρεση των χρηστών από τη βάση δεδομένων.

- **UserDataRepository**: Περιέχει τη διεπαφή για την εύρεση των δεδομένων των χρηστών από τη βάση δεδομένων.

- **Entities**: Πακέτο που περιέχει όλες τις αντίστοιχες οντότητες των παραπάνω components που αποθηκεύονται στη βάση με τη μορφή αντικειμένων του αντικειμενοστραφούς προγραμματισμού.

Παρακάτω δίνεται το αντίστοιχο διάγραμμα.



Σχήμα 3.8: Component diagram του repository.

3.4 Επικοινωνία μεταξύ των components του συστήματος

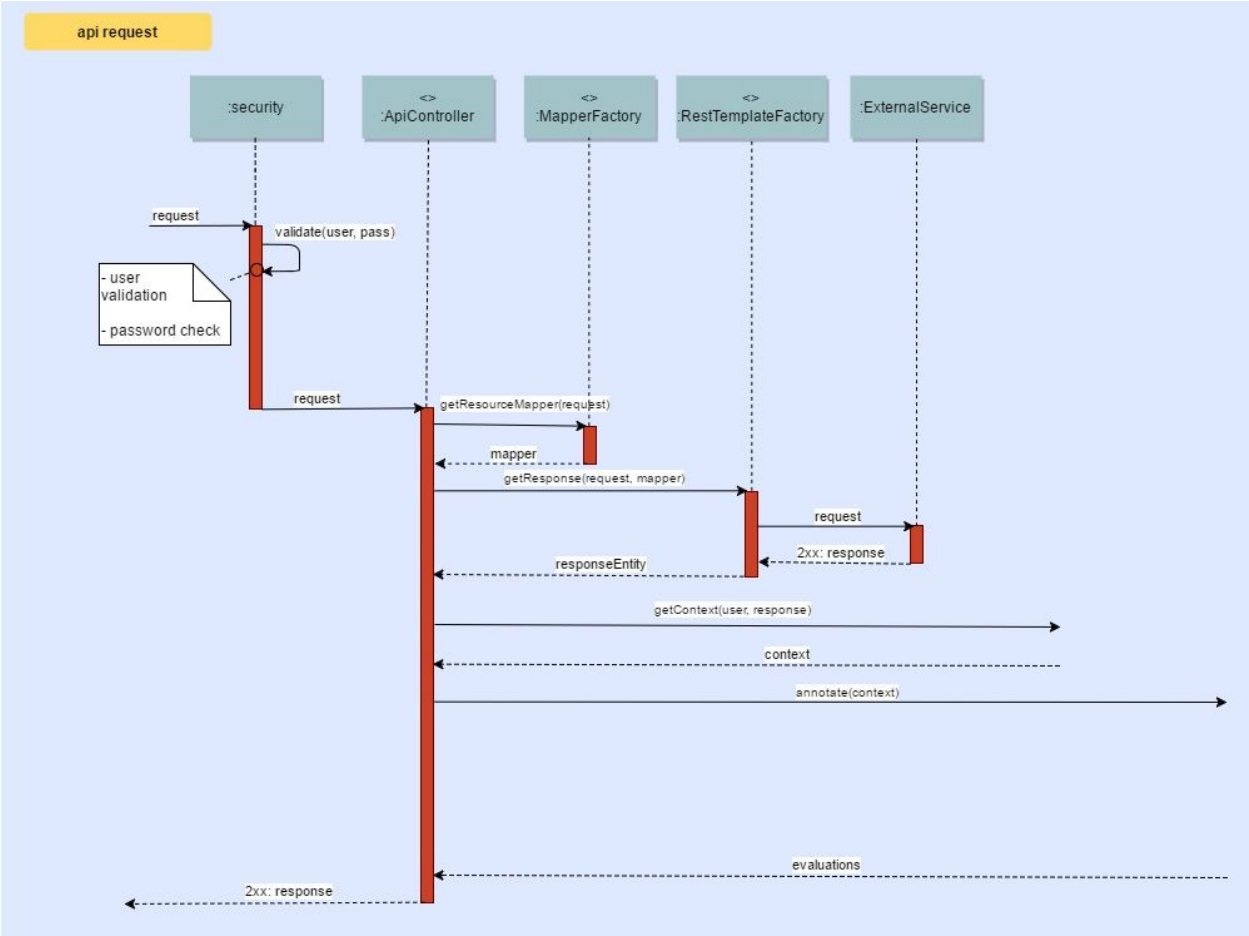
Αυτό το κεφάλαιο παρουσιάζει μερικά ακολουθιακά διαγράμματα τα οποία δείχνουν τη σειρά και τον τρόπο αλληλεπίδρασης μεταξύ των components τα οποία περιγράφηκαν παραπάνω.

3.4.1 Ακολουθιακό διάγραμμα ενός REST api ερωτήματος χωρίς προβλήματα

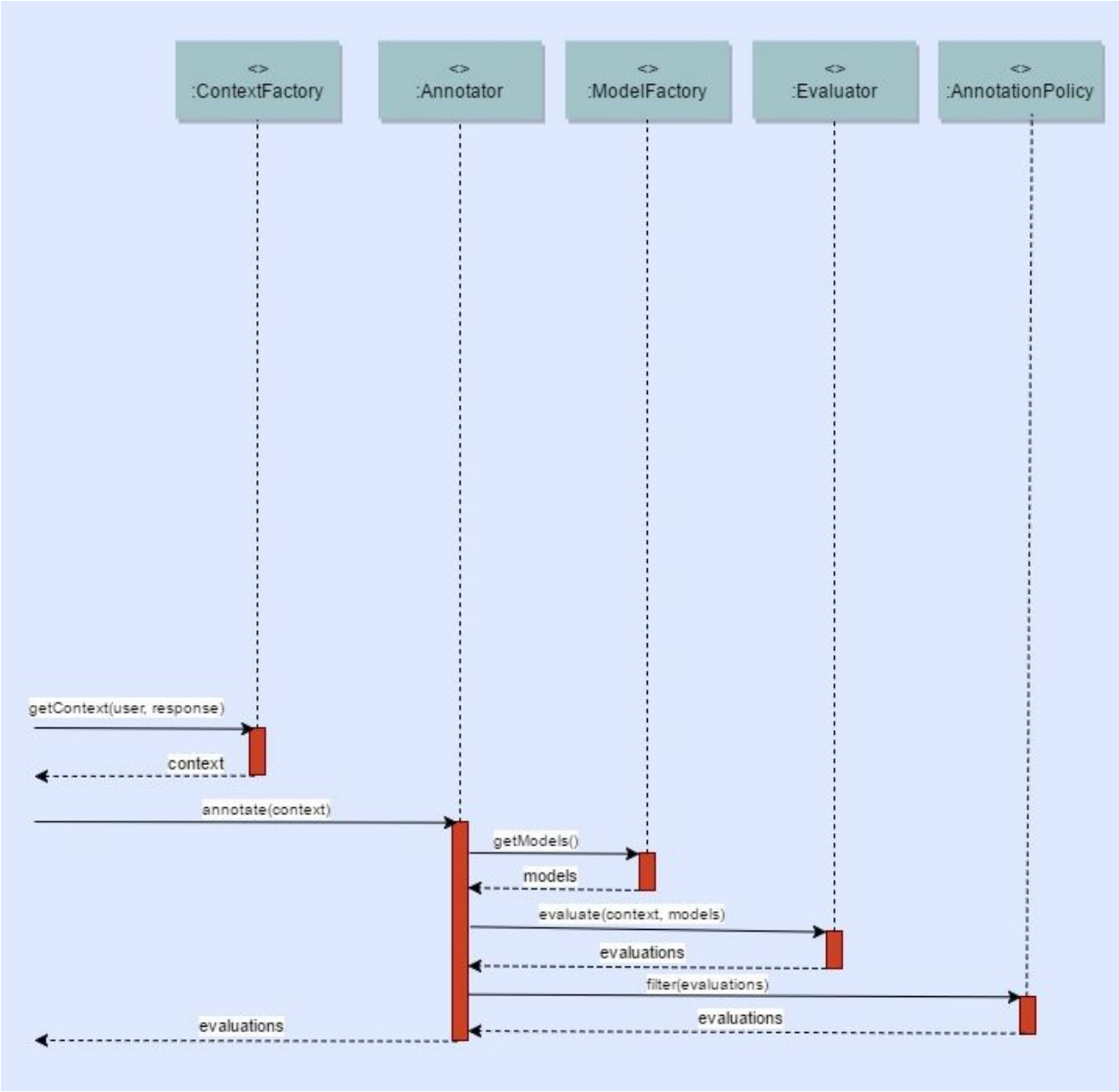
Όταν το σύστημα λαμβάνει ένα ερώτημα στο path `/api/*` ακολουθεί μία σειρά από βήματα:

- Το request περνάει από το security το οποίο επιβεβαιώνει τα διαπιστευτήρια του χρήστη (authenticate)
- Στη συνέχεια ο **ApiController** μέσω του `getResourceMapper` που παρέχει το **MapperFactory** παίρνει τον αντίστοιχο mapper για να βρει την εξωτερική διεύθυνση που πρέπει να προωθηθεί το request.
- Το **RestTemplateFactory** στη συνέχεια μέσω του `getResponse` προωθεί το request στην εξωτερική υπηρεσία και όταν λάβει απάντηση επιστρέφει στον **ApiController** το αντίστοιχο ResponseEntity (Spring αντικείμενο).
- Ο **ApiController** καλεί το `getContext` του **ContextFactory** για να λάβει το **AffordanceContext**.
- Στη συνέχεια καλείται ο **Annotator** μέσω του annotate για να προσθέσει τα affordances.
- Ο **Annotator** καλεί τον **Evaluator** μέσω του evaluate.
- Ο **Evaluator** καλεί το `getModels` του **ModelFactory** για να πάρει όλα τα σχετικά με το **AffordanceContext** goal models.
- Στη συνέχεια αφού κάνει evaluate τα goal models, επιστρέφει τα evaluations στον **Annotator**.
- Ο **Annotator** μέσω του `filter` του **AnnotationPolicy** φιλτράρει αυτά τα evaluations και τα επιστρέφει στον **ApiController**.
- Ο **ApiController** επιστρέφει την απάντηση στον πελάτη.

Παρακάτω δίνεται και το αντίστοιχο διάγραμμα σε δύο μέρη λόγω ευκολίας παρουσίασης.



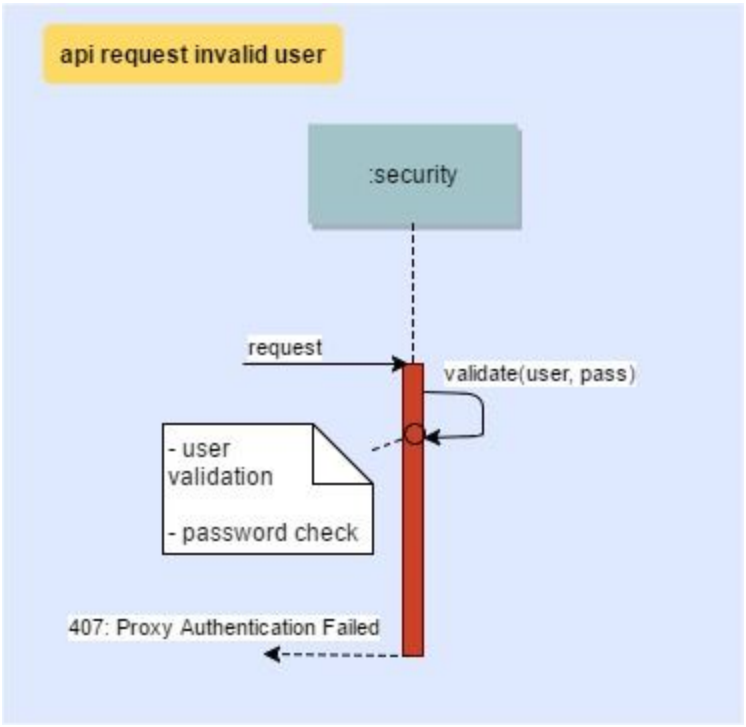
Σχήμα 3.9: Ακολουθιακό διάγραμμα 1 REST api request



Σχήμα 3.10: Ακολουθιακό διάγραμμα 2 REST api request

3.4.2 Ακολουθιακό διάγραμμα ερωτήματος με λανθασμένα διαπιστευτήρια χρήστη

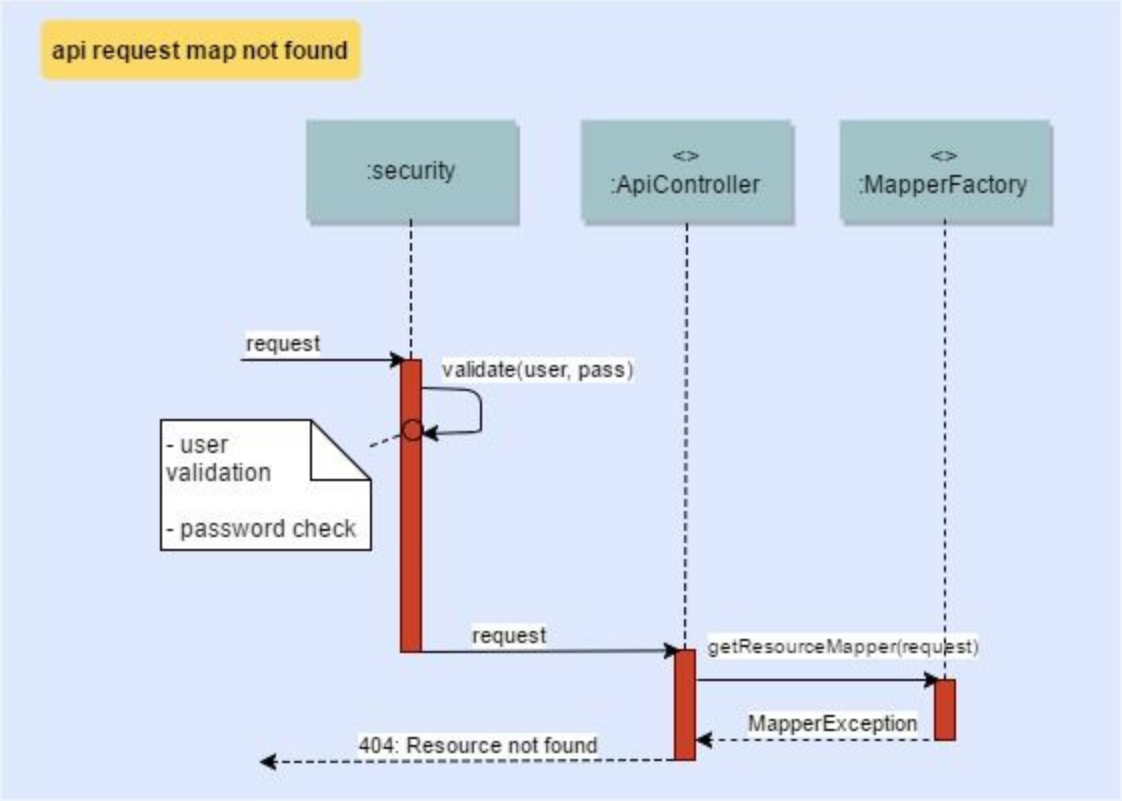
Σε περίπτωση που ο πελάτης ενός api request δώσει λανθασμένα διαπιστευτήρια (μη έγκυρες ή καθόλου επικεφαλίδες X-AUTH-USERNAME, X-AUTH-PASSWORD), τότε του επιστρέφεται απάντηση με κωδικό σφάλματος 407: Proxy Authentication Failed.



Σχήμα 3.11: Ακολουθιακό διάγραμμα REST api request με λανθασμένα διαπιστευτήρια χρήστη

3.4.3 Ακολουθιακό διάγραμμα σε path που δεν υπάρχει

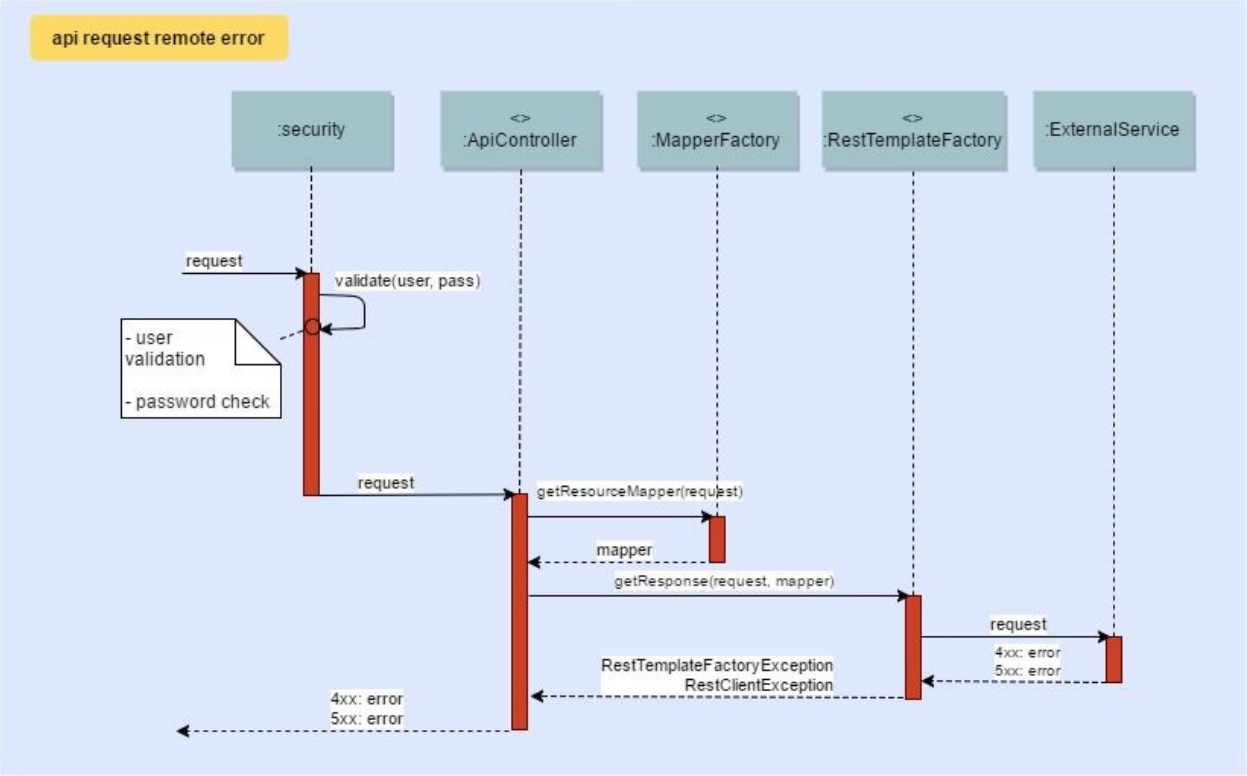
Σε περίπτωση που ο πελάτης κάνει ερώτημα σε path το οποίο δεν υποστηρίζει η υπηρεσία, δηλαδή δεν υπάρχει αντίστοιχο mapping για αυτήν την υπηρεσία, τότε η αντίστοιχη κλήση `getResourceMapper` του **MapperFactory** θα προκαλέσει εξαίρεση (exception) την οποία θα διαχειριστεί ο **ApiController** για να επιστρέψει στον πελάτη απάντηση με κωδικό σφάλματος 404: Resource not found.



Σχήμα 3.12: Ακολουθιακό διάγραμμα REST api ερωτήματος σε path που δεν υπάρχει

3.4.4 Ακολουθιακό διάγραμμα σφάλματος της εξωτερικής υπηρεσίας

Στην περίπτωση που η εξωτερική υπηρεσία επιστρέψει κάποιο σφάλμα (4xx ή 5xx), τότε τα βήματα που ακολουθούνται αρχικά είναι όπως και στο 3.4.1, με τη διαφορά ότι το RestTemplateFactory θα προκαλέσει εξαίρεση (exception), την οποία θα διαχειριστεί ο ApiController για να επιστρέψει το αντίστοιχο σφάλμα που του επέστρεψε η εξωτερική υπηρεσία.

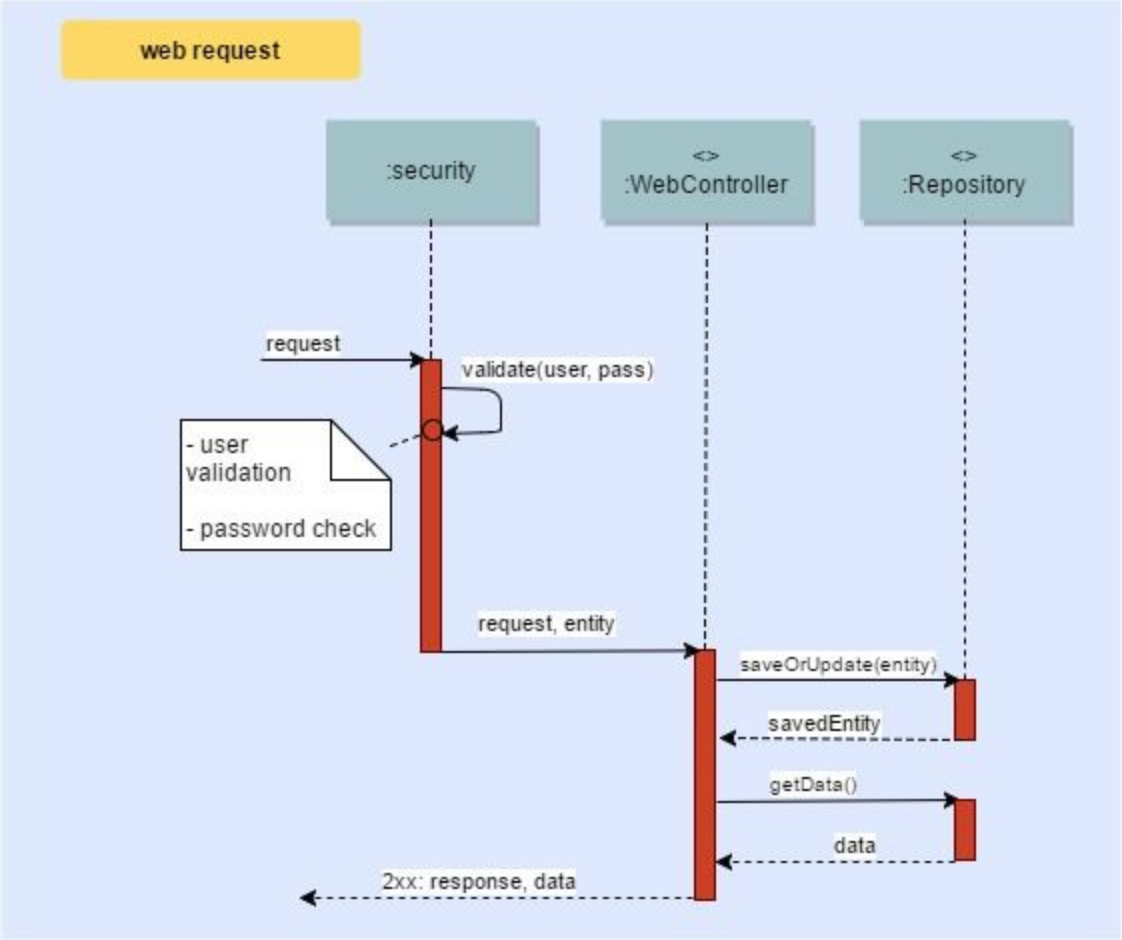


Σχήμα 3.13: Ακολουθιακό διάγραμμα σε REST api ερώτημα με σφάλμα της εξωτερικής υπηρεσίας

3.4.5 Ακολουθιακό διάγραμμα ερωτήματος από διαχειριστή

Στην περίπτωση που θέλει ο διαχειριστής να συνδεθεί τότε μέσω της διεπαφής που παρέχεται από τον περιηγητή θα κάνει ένα ερώτημα σε κάποιο path στο /admin. Τότε τα βήματα που ακολουθούνται είναι:

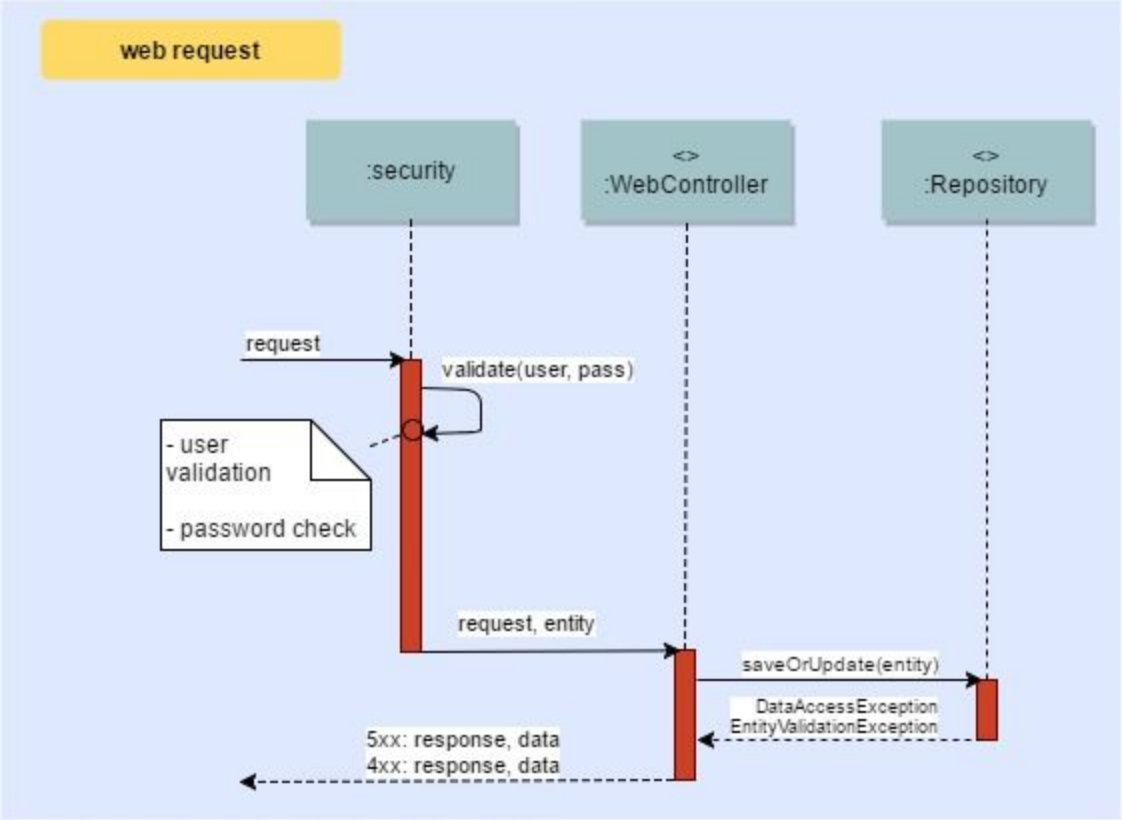
- Επιβεβαίωση στοιχείων μέσω του **security**. Η επιβεβαίωση γίνεται με την HTTP επικεφαλίδα Basic Authentication.
- Προώθηση ερωτήματος στον **WebController**
- Ο **WebController** χρησιμοποιεί το `query` interface του **Repository** και πιο συγκεκριμένα το `saveOrUpdate` για να αποθηκεύσει τυχόν αλλαγές που έκανε ο διαχειριστής και στη συνέχεια καλεί το `getData` για να λάβει τα αντίστοιχα αποθηκευμένα δεδομένα.
- Επιστρέφει τα δεδομένα στο χρήστη σε μορφή απάντησης REST.



Σχήμα 3.14: Ακολουθιακό διάγραμμα REST ερωτήματος διαχειριστή

3.4.6 Ακολουθιακό διάγραμμα ερωτήματος από διαχειριστή με σφάλμα στο σύστημα

Στην παραπάνω περίπτωση κατά τη διάρκεια του saveOrUpdate ή του getData μπορεί να υπάρξει σφάλμα. Σε αυτήν την περίπτωση το Repository προκαλεί εξαίρεση την οποία διαχειρίζεται ο WebController για να επιστρέψει το αντίστοιχο σφάλμα στον πελάτη.



Σχήμα 3.15: Ακολουθιακό διάγραμμα REST ερωτήματος διαχειριστή με σφάλμα στο σύστημα

Κεφάλαιο 4: Domain Model

4.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφουμε το εννοιολογικό μοντέλο (domain model) των δέντρων στόχων και των affordances του περιβάλλοντος πλαισίου που υλοποιήθηκε στα πλαίσια αυτής της διπλωματικής εργασίας. Το μοντέλο, ουσιαστικά, περιγράφει τις οντότητες που χρησιμοποιήθηκαν για να προσαρμόσουμε ήδη υπάρχουσες υλοποιήσεις που αφορούν τα δέντρα στόχων και affordances στο συγκεκριμένο πρόβλημα, καθώς και τις απαραίτητες επεκτάσεις ώστε η υλοποίηση να είναι όσο το δυνατόν περισσότερο αφηρημένη (abstract).

Στη συνέχεια παρουσιάζονται τα εννοιολογικά μοντέλα δέντρων στόχων και affordances.

4.2 Εννοιολογικό μοντέλο δέντρων στόχων

Το εννοιολογικό μοντέλο των δέντρων στόχων αφορά ουσιαστικά όλες τις οντότητες που χρησιμοποιήθηκαν στην υλοποίηση που αφορούν τα δέντρα στόχων και την ικανοποιησιμότητα τους. Έχουν χρησιμοποιηθεί πολλά στρώματα (layers) και επεκτάσεις για να επιτευχθεί αυτό. Ο λόγος είναι ότι στη συγκεκριμένη διπλωματική το κύριο αντικείμενο που σχετίζεται με τα δέντρα στόχων είναι τα affordances, αλλά σε μια γενικότερη εικόνα τα δέντρα στόχων θα μπορούσαν να χρησιμοποιηθούν για οτιδήποτε βρίσκει εφαρμογή στα πλαίσια της ικανοποιησιμότητας, της εξατομίκευσης και γενικότερα της λήψης αποφάσεων. Συνεπώς κάποιος που θα ήθελε να χρησιμοποιήσει τα δέντρα στόχων για κάτι τέτοιο θα μπορούσε να επεκτείνει συγκεκριμένες οντότητες ή ακόμα και όλο το προσφερόμενο περιβάλλον πλαίσιο χωρίς να χρειαστεί να το υλοποιήσει όλο εκ νέου.

Στη συνέχεια περιγράφουμε τις κλάσεις που σχετίζονται με τα δέντρα στόχων μία προς μία και στο τέλος παρουσιάζουμε το αντίστοιχο διάγραμμα κλάσεων.

4.2.1 Η κλάση GoalModelNode

Η κλάση αυτή αποτελεί μια διεπαφή (interface) με μοναδική μέθοδο την `getNodeName()` που είναι τύπου `String` και αφορά ουσιαστικά τον κόμβο ενός δέντρου στόχων. Αυτό σημαίνει πως κάθε τι που θα θέλει να υλοποιεί τη συγκεκριμένη διεπαφή θα πρέπει να υλοποιεί και τη συγκεκριμένη μέθοδο. Ο λόγος που γίνεται αυτό είναι διότι κατά τι διάρκεια του reasoning ο κάθε κόμβος θα πρέπει να μπορεί να προσδιοριστεί μοναδικά και θα πρέπει να έχει μοναδικό όνομα.

Για αυτό το λόγο θα πρέπει να φροντίζουμε ότι το όνομα που δίνουμε στον κάθε κόμβο του δέντρου είναι μοναδικό.

4.2.2 Η κλάση ContributionType

Η κλάση αυτή αποτελεί μία απαρίθμηση (enumeration) με τέσσερα πεδία

- `PPS`
- `MMS`
- `PPD`
- `MMD`

4.2.3 Η κλάση DecompositionType

Η κλάση αυτή αποτελεί μία απαρίθμηση (enumeration) με δύο πεδία

- `AND`
- `OR`

και ουσιαστικά συσχετίζει ένα “πατέρα” κόμβο με τα παιδιά του είτε με σχέση `OR` είτε με σχέση `AND`.

4.2.4 Η κλάση GoalModelRule

Αποτελεί επίσης μια διεπαφή και αφορά τους “κανόνες” ενός δέντρου στόχων. Παρέχει δύο μεθόδους.

- `String getConsequent()`: Επιστρέφει τον κανόνα που αφορά τον στόχο.
- `Collection<String> getAntecedents()`: Επιστρέφει τους κανόνες που αφορούν τις προϋποθέσεις που χρειάζονται για να ικανοποιηθεί ο στόχος.

4.2.5 Η κλάση GenericGoalModel

Η κλάση αυτή είναι επίσης μια διεπαφή και αποτελεί τη ραχοκοκαλιά των δέντρων στόχων. Οποιαδήποτε κλάση θέλει να “είναι” δέντρο στόχων θα πρέπει να επεκτείνει ή να υλοποιεί αυτήν την κλάση. Παρέχει τις εξής μεθόδους:

- `void addDecomposition(T parentNode, List<T> childNodes, DecompositionType type)`: Μέθοδος που χρησιμοποιείται για να προσθέσει μια συσχέτιση μεταξύ ενός κόμβου “πατέρα” και κάποιων κόμβων παιδιών με `AND` ή `OR`
- `void addContributionLink(T sourceNode, T targetNode, ContributionType type)`: Μέθοδος που χρησιμοποιείται για να προσθέσει μια συσχέτιση μεταξύ δύο κόμβων.
- `void addWeightedContributionLink(T sourceNode, T targetNode, ContributionType type, double weight)`: Μέθοδος που χρησιμοποιείται για να προσθέσει μια συσχέτιση μεταξύ δύο κόμβων με συγκεκριμένο βάρος.
- `List<T> getRootNodes()`: Μέθοδος που χρησιμοποιείται για να πάρει όλους τους κόμβους που θεωρούνται ρίζες.
- `List<T> getLeafNodes()`: Μέθοδος που χρησιμοποιείται για να πάρει όλα τα φύλλα κόμβους.
- `List<T> getAllNodes()`: Επιστρέφει όλους τους κόμβους.
- `T getNodeByName(String nodeName)`: Επιστρέφει τον κόμβο με το παρεχόμενο όνομα.
- `Set<GoalModelRule> getGoalModelRules()`: Επιστρέφει όλους τους κανόνες.

Εδώ είναι χρήσιμο να σημειώσουμε πως η παράμετρος `T` μπορεί να είναι οποιαδήποτε υποκλάση τύπου `GoalModelNode` και ο λόγος που χρησιμοποιείται κατ’ αυτόν τον τρόπο είναι ώστε να γίνει πιο αφαιρετική η κλάση `GenericGoalModel` και κατά συνέπεια περισσότερο παραμετροποιήσιμη.

4.2.6 Η κλάση DefaultGoalModelImpl

Η κλάση αυτή αποτελεί μια υλοποίηση της διεπαφής GenericGoalModel και χρησιμοποιεί επίσης την παράμετρο T για τους λόγους που αναφέρθηκαν προηγουμένως.

4.2.7 Η κλάση Model

Αυτή η κλάση είναι η πρώτη κλάση προς την κατεύθυνση των μοντέλων με affordances. Είναι μια αφηρημένη κλάση η οποία επίσης χρησιμοποιεί την παράμετρο `T`, αλλά ταυτοχρόνως εισάγει και μία νέα παράμετρο `R`, η οποία αφορά το affordance. Ο λόγος που το affordance είναι τύπου `R` είναι για να μπορεί εύκολα να προσαρμοστεί σε άλλες εργασίες/υλοποιήσεις που δεν αφορούν αμιγώς affordances με την έννοια της προσθήκης περιεχομένου σε πόρους REST. Εσωτερικά κρατάει πληροφορία για το όνομα του μοντέλου καθώς και τα ContextKeys, που περιγράφηκαν στο κεφάλαιο 3, που χρειάζονται για να περιγραφεί το μοντέλο. Εκτός από προκαθορισμένες μεθόδους `getters` και `setters` παρέχει τη μέθοδο:

```
double applyReasoning(AffordanceContext context)
```

Η μέθοδος αυτή είναι η “καρδιά” του reasoning και οποιαδήποτε κλάση επεκτείνει ή χρησιμοποιεί την κλάση `Model` χρησιμοποιεί το `applyReasoning` για να πάρει το αποτέλεσμα του reasoning για το συγκεκριμένο μοντέλο/δέντρο στόχων.

Επίσης σημαντικό είναι να σημειώσουμε και την ύπαρξη της αφηρημένης μεθόδου (abstract method) `Map<String, LowHighNodeValue> getInitialValues(@Nonnull AffordanceContext context)` η οποία δεν υλοποιείται, αλλά αφήνεται στις κλάσεις που θα επεκτείνουν την κλάση `Model` να το κάνουν. Για αυτό το λόγο η κλάση `Model` δεν μπορεί να χρησιμοποιηθεί απευθείας, παρά μόνο να επεκταθεί και υπάρχει για λόγους καλύτερης δόμησης του περιβάλλοντος πλαισίου.

4.2.8 Η κλάση LeafData

Βοηθητική κλάση που αφορά τα φύλλα κόμβους και “κρατάει” πληροφορία που αφορά τον τύπο του REST πόρου, τη μέθοδο σύγκρισης, την τιμή και το βάρος μέσω των ακόλουθων πεδίων (τα πεδία παρέχονται μέσω `getters` και `setters`):

- `String resourceType`
- `String resourceField`
- `Operation op`
- `String value`
- `Double weight`

4.2.9 Η κλάση AffordanceModel

Κλάση που επεκτείνει την κλάση Model και χρησιμοποιεί κόμβους που μπορούν να κρατήσουν την πληροφορία που παρέχει η κλάση `LeafData`, μέσω της αφηρημένης παραμέτρου `T`, όπως περιγράφηκε προηγουμένως. Αντιθέτως η παράμετρος R παραμένει αφηρημένη. Εσωτερικά περιέχει ένα αντικείμενο τύπου `CompareCalculator`, όπως περιγράφηκε στο κεφάλαιο 3, για να μπορεί να "υπολογίσει" την τιμή κάθε φύλλου με βάση τα δεδομένα που παρέχει το LeafData. Παρέχει τη μέθοδο `Map<String, LowHighNodeValue> getInitialValues(@Nonnull AffordanceContext context)` όπως περιγράφηκε και παραπάνω στην κλάση `Model`. Η μέθοδος αυτή ορίζει τις τιμές στα φύλλα κόμβους σαν "προεργασία" πριν την εφαρμογή της μεθόδου `applyReasoning()`. Ο λόγος που γίνεται αυτό είναι γιατί σε κάθε REST ερώτημα η παράμετρος του κάθε φύλλου μπορεί να διαφέρει και μπορεί να είναι από αριθμός που χρειάζεται σύγκριση με κάποιον άλλο, μέχρι συμβολοσειρά, ημερομηνία κλπ. Παρολαυτά η τιμή του φύλλου δεν μπορεί να είναι τίποτα άλλο από ένας αριθμός ώστε να γίνει επιτυχώς το reasoning, γιαυτό χρησιμοποιούμε την κλάση `CompareCalculator` για να υπολογίσουμε αυτήν την τιμή πριν το reasoning.

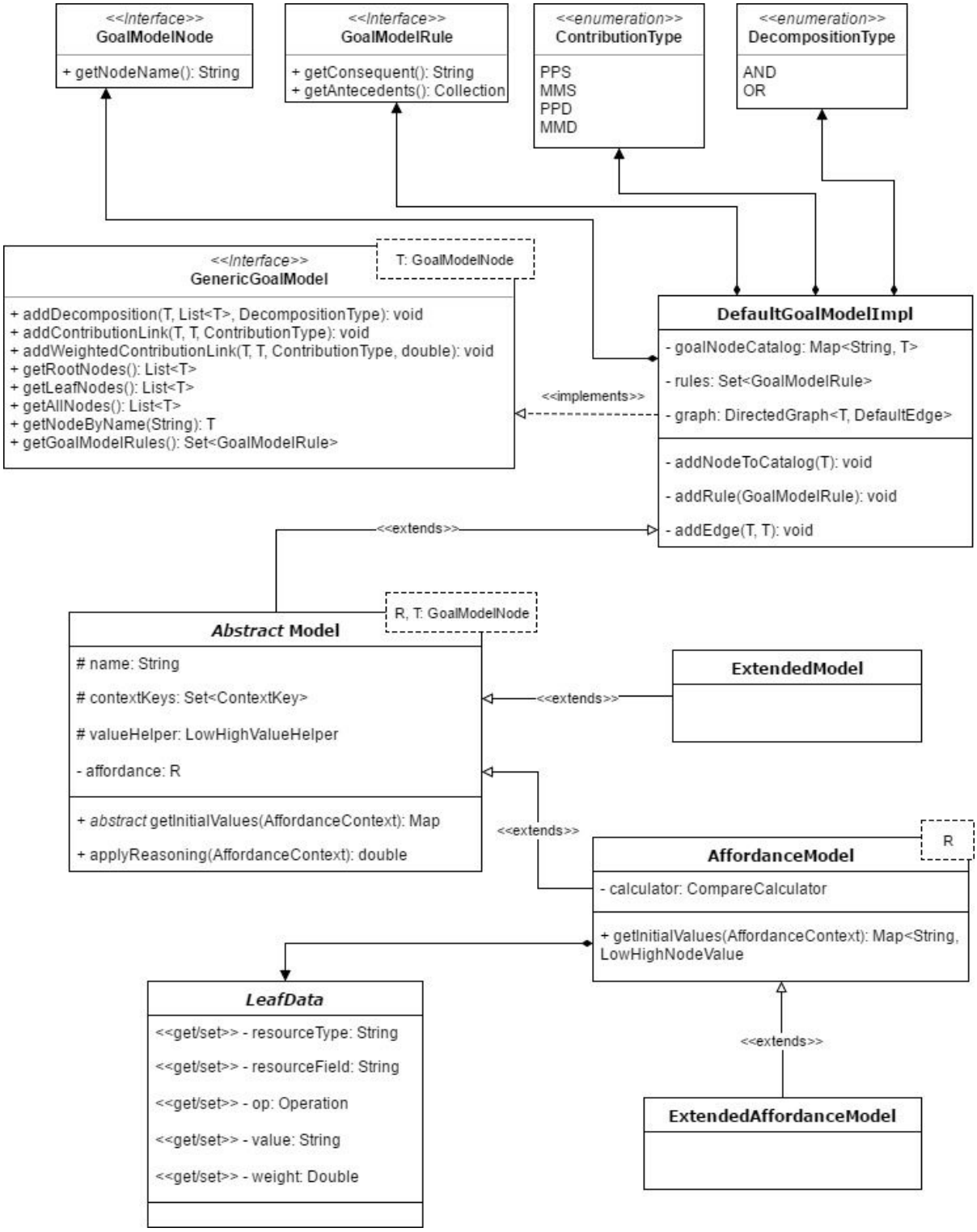
4.2.10 Η κλάση ExtendedModel

Αποτελεί μια απλή επέκταση της κλάσης `Model` και θέτει στην παράμετρο `R` το αντικείμενο `Affordance`, ώστε να γίνεται ευκολότερη η χρήση στον proxy χωρίς να δηλώνουμε κάθε φορά την παράμετρο ως `Affordance`.

4.2.11 Η κλάση ExtendedAffordanceModel

Αποτελεί μια απλή επέκταση της κλάσης `AffordanceModel` και θέτει στην παράμετρο `R` το αντικείμενο `Affordance`, όπως και με την περίπτωση της κλάσης `ExtendedModel`.

Στη συνέχεια παραθέτουμε το διάγραμμα κλάσεων με βάση τις παραπάνω κλάσεις.



Σχήμα 4.1: Διάγραμμα Κλάσεων Δέντρων Στόχων

4.3 Εννοιολογικό μοντέλο Affordances

Το εννοιολογικό μοντέλο των `Affordances`, αφορά όλες τις οντότητες που αφορούν τα Affordances, όπως χρησιμοποιήθηκαν στα πλαίσια αυτής της διπλωματικής. Παρουσιάζεται αναλυτικά η έννοια του `AffordanceContext`, που είναι η "καρδιά" των Affordances με όλα τα πεδία και τις μεθόδους που περιέχει. Επίσης παρουσιάζεται και το αντίστοιχο διάγραμμα κλάσεων όπως και στο υποκεφάλαιο 4.2.

4.3.1 Link

Η κλάση αυτή αποτελεί κομμάτι του Spring Framework και αφορά την προσθήκη συνδέσμων σε REST πόρους. Περιέχει μία πληθώρα από εσωτερικά πεδία και σταθερές καθώς και τις εξής μεθόδους:

- `String getHref()`: Επιστρέφει τον σύνδεσμο που περιέχεται στο `Link`
- `String getRel()`: Επιστρέφει το `rel` (όνομα που συσχετίζεται) του `Link`
- `Link withRel(String rel)`: Προσθέτει ένα `rel` στο `Link` και το επιστρέφει
- `Link withSelfRel()`: Προσθέτει ένα `self rel` στο `Link` και το επιστρέφει
- `List<String> getVariableNames()`: Επιστρέφει τα ονόματα των μεταβλητών του `Link`
- `List<TemplateVariables> getVariables()`: Επιστρέφει τις μεταβλητές του `Link`
- `boolean isTemplated()`: Ελέγχει αν το `Link` έχει μορφή Template
- `Link expand(Object... arguments)`: Επεκτείνει το `Link` με τις παρεχομενες παραμέτρους και το επιστρέφει
- `Link expand(Map<String, ? extends Object...> arguments)`: Επεκτείνει το `Link` με τις παρεχομενες παραμέτρους και το επιστρέφει.

4.3.2 Affordance

Η κλάση αυτή παρέχεται από το Spring Hydra Framework [21] και αποτελεί επέκταση του Link. Περιέχει ένα τεράστιο αριθμό πεδίων και μεθόδων και για αυτό το λόγο δεν θα αναλυθούν στη διπλωματική αυτή. Παρολαυτά οποιοσδήποτε ενδιαφέρεται μπορεί να ανατρέξει στον πηγαίο κώδικα του Affordance για να δει όλες τις παρεχόμενες μεθόδους. Το Affordance αποτελεί στην ουσία ένα Link, αφού δεν είναι τίποτα άλλο από επιπλέον περιεχόμενο με τη μορφή συνδέσμου σε κάποιον REST πόρο. Ωστόσο πρέπει να επισημάνουμε ότι παρέχει επιπλέον λειτουργίες,

όπως για παράδειγμα το να ορίσει κάποιος τι τύπου είναι το Affordance (MediaType type), ή την προτεραιότητα του σε σχέση με άλλα Affordances (cardinality).

4.3.3 ContextKey

Η κλάση αυτή αποτελεί το “κλειδί” στην σχέση κλειδιού-τιμής που υπάρχει στο `AffordanceContext`, όπως περιγράφηκε και σε προηγούμενα κεφάλαια. Περιέχει δύο πεδία που προσδιορίζουν ουσιαστικά κάποιον πόρο. Ουσιαστικά το ένα πεδίο προσδιορίζει το αντικείμενο που αφορά τον πόρο και το άλλο πεδίο αφορά το συγκεκριμένο πεδίο του πόρου που αναφέρεται το συγκεκριμένο `ContextKey`

4.3.4 ContextValue

Η κλάση αυτή αποτελεί την “τιμή” στη σχέση κλειδιού-τιμής και περιέχει ουσιαστικά ένα πεδίο που είναι η πραγματική τιμή του `ContextValue`. Το πεδίο είναι τύπου `Object` και αυτό διότι θέλουμε να είναι όσο πιο γενικό γίνεται. Το κάθε αντικείμενο μετά μπορεί να μετατραπεί και να συγκριθεί στο εκάστοτε δέντρο με τη χρήση των `ContextManagers` όπως περιγράφηκε στο Κεφάλαιο 3.

4.3.5 AffordanceContext

Η κλάση αυτή αποτελεί την “καρδιά” της λειτουργίας που αφορά τα Affordances. Περιέχει τρία βασικά εσωτερικά πεδία:

- `Map<ContextKey, ContextValue> isDefinedBy`: Αποτελεί ουσιαστικά το πεδίο που κρατάει όλες τις σχέσεις κλειδιού-τιμής, όπως αναφέρθηκε. Η ουσιαστική σημασία για αυτό το πεδίο είναι για να μπορούμε να γνωρίζουμε αν ένας πόρος αφορά κάποιο συγκεκριμένο affordance. Δηλαδή αν υπάρχει κάποιο συγκεκριμένο `ContextKey` στο `Map`, τότε γνωρίζουμε πως για το συγκεκριμένο REST πόρο έχει νόημα να λάβουμε υπόψιν κάποιο συγκεκριμένο Affordance (`AffordanceContext`) και κατά συνέπεια το σχετιζόμενο δέντρο/μοντέλο.
- `Collection<Set<ContextKey>> cnfList`: Αυτό το πεδίο αφορά μια έκφραση CNF. Η έκφραση αυτή περιέχει σαν όρους στοιχεία `ContextKey` και ο λόγος ύπαρξής της είναι για να μπορούμε να επιλέξουμε σε ποια συγκεκριμένα Affordances και κατά συνέπεια δέντρα θα πρέπει να κάνουμε reasoning. Αν δεν υπήρχε αυτή η έκφραση τότε όλα τα δέντρα που περιέχουν όλα τα αντίστοιχα `ContextKeys` με κάποιο REST πόρο θα υπολογίζονταν με κίνδυνο κάποιο δέντρο με πολύ λίγα ContextKeys και κατά συνέπεια,

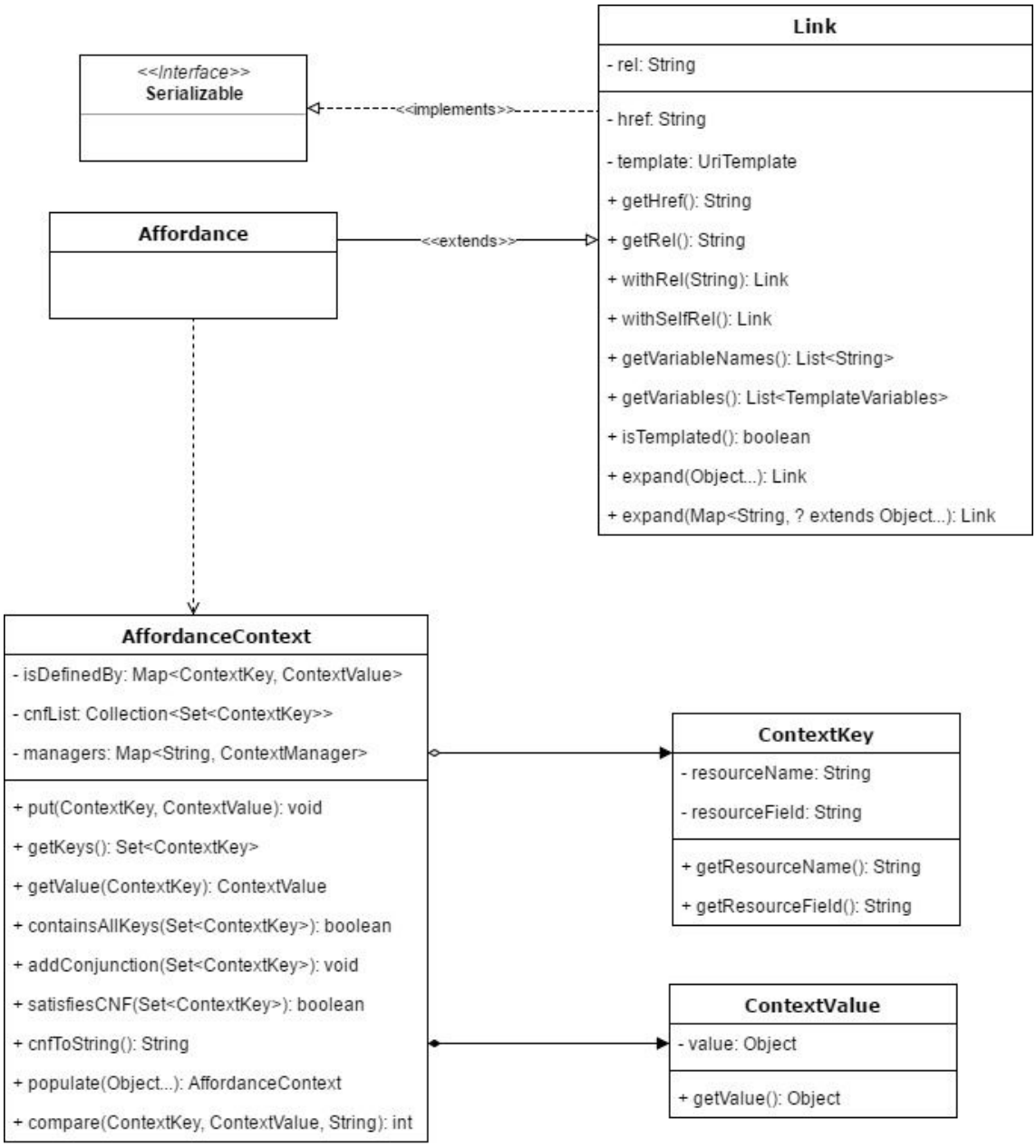
ίσως, φύλλα, να επισκιάζει (shadowing) κάποιο άλλο με πολύ περισσότερα. Η κλάση `ContextBuilder` (κεφάλαιο 3) μπορεί να αναλάβει να εμπλουτίσει κατάλληλα αυτό το πεδίο με χρήση των παρεχόμενων μεθόδων (βλ. παρακάτω)

- `Map<String, ContextManager> managers`: Αυτό το πεδίο αποθηκεύει όλους τους `ContextManagers` που χρειάζονται για να εμπλουτίσουν και να υπολογίσουν τις τιμές στα φύλλα των δέντρων στόχων.

Παράλληλα παρέχεται και μια πληθώρα από χρήσιμες μεθόδους:

- `void put(@Nonnull ContextKey key, @Nullable ContextValue value)`: Προσθέτει μια σχέση κλειδιού-τιμής.
- `Set<ContextKey> getKeys()`: Επιστρέφει όλα τα `ContextKeys` που αφορούν αυτό το `AffordanceContext`.
- `ContextValue getValue(@Nonnull ContextKey key)`: Επιστρέφει την τιμή ενός συγκεκριμένου κλειδιού (`ContextKey`).
- `boolean containsAllKeys(@Nonnull Set<ContextKey> keys)`: Ελέγχει αν CNF πρόταση που είναι αποθηκευμένη περιέχει όλα τα κλειδιά (`ContextKeys`) που παρέχονται στη μέθοδο. Εδώ πρέπει να σημειωθεί ότι δεν ελέγχει αν η CNF πρόταση ικανοποιείται, αλλά αν απλά τα κλειδιά περιέχονται σαν όροι.
- `void addConjunction(@Nonnull Set<ContextKey> keys)`: Προσθέτει μια σύζευξη με τα παρεχόμενα κλειδιά.
- `boolean satisfiesCnf(@Nonnull Set<ContextKey> contextKeys)`: Ελέγχει αν ικανοποιείται η CNF πρόταση.
- `String cnfToString()`: Μετατρέπει την CNF πρόταση σε συμβολοσειρά.
- `AffordanceContext populate(@Nonnull Object... data)`: Εμπλουτίζει τις τιμές (`ContextValue`) με τα παρεχόμενα δεδομένα.
- `int compare(@Nonnull ContextKey key, @Nullable ContextValue value1, @Nullable String value2)`: Συγκρίνει τις δύο τιμές για κάποιο κλειδί και επιστρέφει έναν ακέραιο σαν αποτέλεσμα σύγκρισης σύμφωνα με τη σύμβαση που υπάρχει στις συγκρίσεις. Δηλαδή -1 για μικρότερο, 0 για ίσον και 1 για μεγαλύτερο.

Στη συνέχεια παρουσιάζεται το αντίστοιχο διάγραμμα κλάσεων για τα Affordances.



Σχήμα 4.2: Διάγραμμα Κλάσεων Affordances

Κεφάλαιο 5: Συλλογιστική δέντρων στόχων

5.1 Δέντρα στόχων

5.1.1 Περιγραφή προβλήματος

Η επαλήθευση στο χρόνο εκτέλεσης αφορά την αξιολόγηση ενός συστήματος, όταν αυτό βρίσκεται σε λειτουργία, με βάση αν αυτό ικανοποιεί κάποια προβλεπόμενη συμπεριφορά ή στόχους. Στο πλαίσιο αυτής της διπλωματικής θα αναφερόμαστε σε αυτό το πρόβλημα ως ένα ReqRV πρόβλημα [28]. Έχοντας σαν δεδομένα ένα σύνολο από υποθέσεις (D) και προαπαιτούμενα (R), το πρόβλημα ικανοποιησιμότητας των προαπαιτούμενων R στοχεύει στο να προσδιοριστούν οι συνθήκες (S) για τις οποίες ικανοποιούνται οι R . Το παραπάνω μπορεί να εκφραστή και με την παρακάτω φόρμουλα:

$$D \cup S \vdash R \ (1)$$

Η σχέση (1) μπορεί να προσαρμοστεί στο ReqRV πρόβλημα ως εξής. Δεδομένου ενός συνόλου υποθέσεων D και της περιγραφής του συστήματος ως ένα σύνολο S από τα χαρακτηριστικά του (πχ logs), υπολόγισε τις τιμές των προαπαιτούμενων R , ή αλλιώς στόχους. Ωστόσο είναι λογικό πως σε ένα σύστημα λογισμικού οι στόχοι και οι εξαρτήσεις μεταξύ του μπορεί να διαφέρουν διότι αφορούν στόχους κατά τη διάρκεια εκτέλεσης, καθώς και στόχους που εξαρτώνται από τις διάφορες παραμέτρους του συστήματος (C). Η σχέση (1) τώρα μπορεί να γραφτεί και ως:

$$D(S) \cup S \vDash R(S) \ (2)$$

Όπου $D(S)$ είναι οι υποθέσεις με βάση το C , S είναι διάφορα χαρακτηριστικά της κατάστασης του συστήματος και $R(S)$ οι τιμές των αντίστοιχων στόχων (αληθής ή ψευδής). Εδώ πρέπει να σημειωθεί ότι το S δεν εξαρτάται από τις διάφορες παραμέτρους του συστήματος (C) διότι θεωρούμε ότι αυτά τα χαρακτηριστικά συλλέγονται ανεξάρτητα από αυτές.

Σε αυτή τη διπλωματική, ωστόσο, και κατά συνέπεια στην ανάλυσή μας θεωρούμε μία ασαφή (fuzzy) προσέγγιση στην ικανοποιησιμότητα των στόχων. Δηλαδή ένας στόχος δεν είναι αληθής ή ψευδής, αλλά μπορεί να πάρει τιμές στο διάστημα [0, 1] εκφρασμένο σε ποσοστό επί τοις εκατό. Δηλαδή μπορούμε να πούμε, ότι ένας στόχος ικανοποιείται κατά 60%. Επίσης σε κάθε υπόθεση στο $D(S)$ ανατίθεται μία τιμή (weight) που υποδηλώνει το βάρος της (δηλαδή πόσο σημαντική είναι ή όχι). Εφαρμόζοντας την εξίσωση (2) παίρνουμε ότι κάθε στόχος $R(S)$ έχει

τιμή στο διάστημα [0, 1]. Μπορούμε να ελέγξουμε την ικανοποιησιμότητα του προβλήματος είτε ελέγχοντας αν οι διαδοχικές τιμές που υπολογίζονται είναι κάτω από κάποιο όριο, ή αν οι τιμές των στόχων $R(S)$ βρίσκονται στις αποδεκτές τιμές με βάση τις υποθέσεις $D(S)$.

5.1.2 Υπόβαθρο

5.1.2.1 ΚΑΙ/Ή Δέντρα Στόχων

Τα ΚΑΙ/Ή (AND/OR) δέντρα στόχων αποτελούν επί της ουσίας μοντέλα τα οποία χρησιμοποιούνται από μηχανικούς στην επαλήθευση και εύρεση απαιτήσεων. Η βασική ιδέα αφορά την από πάνω προς τα κάτω (top down) αποσύνθεση των στόχων σε μικρότερους υπό-στόχους. Ένας ΚΑΙ στόχος μπορεί να επαληθευτεί αν όλοι οι υπό-στόχοι του είναι αληθείς, ενώ ένας Ή, αν τουλάχιστον ένας από αυτούς επαληθεύεται. Εκτός από τους ΚΑΙ/Ή στόχους, ένας στόχος (g_s) μπορεί να συνδέεται με κάποιον άλλο (g_t) μέσω ενός συνδέσμου συνεισφοράς (contribution link) και κάποιο συγκεκριμένο βάρος (w). Υπάρχουν τέσσερις τέτοιοι σύνδεσμοι:

- S^P: Η ικανοποιησιμότητα του g_t αυξάνεται κατά w αν ικανοποιείται ο g_s
- S^N: Η μη ικανοποιησιμότητα του g_t αυξάνεται κατά w αν ικανοποιείται ο g_s
- D^P: Η μη ικανοποιησιμότητα του g_t αυξάνεται κατά w αν δεν ικανοποιείται ο g_s
- D^N: Η ικανοποιησιμότητα του g_t αυξάνεται κατά w αν δεν ικανοποιείται ο g_s

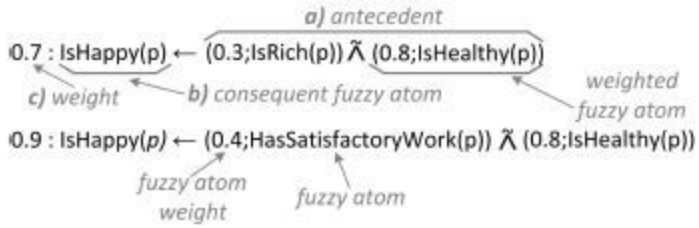
Τέλος οι στόχοι διακρίνονται σε δύο κατηγορίες τους σταθερούς (crisp) και τους ασαφείς (fuzzy). Οι crisp στόχοι είναι αυτοί οι οποίοι μπορούν με βεβαιότητα να πάρουν αληθοτιμή, ενώ οι fuzzy είναι αυτοί για τους οποίους υπάρχουν αρκετά θετικά και λίγα αρνητικά στοιχεία προς την ικανοποιησιμότητά τους.

5.1.2.2 Ασαφής Κανόνες με Βάρος

Η ασαφής λογική σε αντίθεση με την boolean λογική αφορά τιμές στο διάστημα [0, 1]. Η γνώση που αφορά την ασαφή λογική μπορεί να προσδιοριστεί από ένα σύνολο ασαφών κανόνων με βάρος (weighted fuzzy rules) ή εν συντομία wf-rules. Ένας τέτοιος κανόνας αποτελείται από:

- antecedent: Είναι ουσιαστικά το μέρος του κανόνα που αφορά τις προϋποθέσεις που χρειάζονται για να επαληθευτεί ο στόχος.
- consequent: Αφορά το κομμάτι του ίδιου του στόχου.
- weight: Το βάρος στο διάστημα [0, 1] που δηλώνει πόσο “σημαντικός” είναι ένας όρος.

Στη συνέχεια δίνεται ένα παράδειγμα με δύο τέτοιους κανόνες.



Σχήμα 5.1: Παράδειγμα wf-rules

5.1.3 Διαδικασία Reasoning σε στόχους

Για να βρεθεί η ικανοποιησιμότητα στους στόχους σε ένα σύστημα υπάρχουν συγκεκριμένα βήματα που πρέπει να ακολουθηθούν:

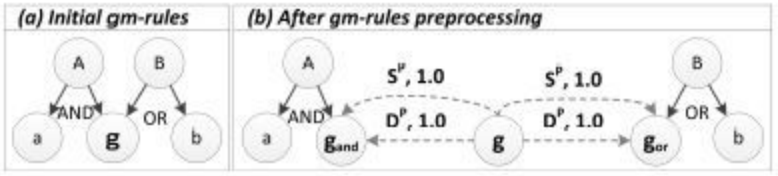
1. Παραγωγή wf-rules.
2. Μετατροπή των χαρακτηριστικών του συστήματος σε ασαφή μορφή (fuzzification).
3. Παραγωγή των συμπερασμάτων με βάση τα μεταφρασμένα χαρακτηριστικά του βήματος 2 (inference).
4. Μετατροπή των ασαφών δεδομένων σε σαφή και εξαγωγή των δεικτών ικανοποιησιμότητας (defuzzification).

5.1.3.1 Παραγωγή των wf-rules

Για να γίνει επιτυχώς η παραγωγή των wf-rules πρέπει πρώτα να προσδιοριστεί το σύνολο των ασαφών όρων που θα χρησιμοποιηθούν στους κανόνες. Μπορούν να οριστούν δύο τέτοιοι όροι. Το $HighSat(g)$ και το $LowSat(g)$. Το πρώτο σημαίνει ότι ένας κόμβος g είναι επαρκώς ικανοποιήσιμος, ενώ ο δεύτερος το αντίθετο, παίρνοντας τιμές στο διάστημα [0. 1].

Σε πρώτη φάση πρέπει να αλλάξουμε όλους τους κόμβους του δέντρου που συμμετέχουν σε ΚΑΙ και Ή κανόνες ως παιδιά κόμβοι με δύο ψευδοκόμβους, έναν που αφορά το ΚΑΙ και έναν που αφορά το Ή, έστω g_{and} και g_{or} αντίστοιχα. Αφού γίνει αυτό πρέπει να βεβαιωθούμε ότι οι ίδιοι κανόνες που θα χρησιμοποιούνταν για να παράξουν την τιμή του αρχικού κόμβου g θα χρησιμοποιηθούν για να παράξουν τις αντίστοιχες τιμές στους κόμβους g_{and} και g_{or}. Για να

γίνει αυτό προσθέτουμε τέσσερις συνδέσμους συνεισφοράς που ουσιαστικά αναθέτουν την τιμή στον g. Ένα παράδειγμα φαίνεται στο ακόλουθο σχήμα.



Σχήμα 5.2: Μετατροπή του δέντρου.

Εφαρμόζοντας αυτή τη διαδικασία παράγουμε ένα δεύτερο μοντέλο από το αρχικό το οποίο σε δεύτερη φάση χρησιμοποιώντας έναν αλγόριθμο που παρατίθεται στη συνέχεια παίρνουμε τους wf-rules.

```
Input : G0: goals, D0: gm-rules
Output : CondRB : cond. wf-rules
1: for all gm r=< R_t, g_t, G_s, w > ε D'  do
2:     f ←CNF(Cond(gm_r), Cond( g_t ), Cond( G_S ))
3:     WF ← produceWFRules(gm_r)
4:     for all wf_r ε WF do
5:           add < wf_r > in CondRB
6:     end for
7: end for
8: for all g ε G' do
9:     if Cond(g) − {⊤} ≠ 0  then
10:          if isAndChild(g) then
11:                f ← ¬CNF(Cond(g))
12:                add <f, 1.0 : HighSat(g)←(1.0; t)> in CondRB
13:                add <f, 0.0 : LowSat(g)←(1.0; t)> in CondRB
14:          end if
15:          if isOrChild(g) then
16:                f ← ¬CNF(Cond(g))
17:                add <f, 0.0 : HighSat(g)←(1.0; t)> in CondRB
18:                add <f, 1.0 : LowSat(g)←(1.0; t)> in CondRB
19:          end if
20:    end if
21: end for
22: return CondRB
```

5.1.3.2 Fuzzification

Το fuzzification συντελείται στη μετατροπή των χαρακτηριστικών του συστήματος σε ασαφή δεδομένα με τη χρήση των HighSat και LowSat, όπως αναφέρθηκε προηγουμένως. Για να μπορέσουμε να αποφασίσουμε αν ένας κόμβος είναι επαρκώς ή όχι ικανοποιήσιμος πρέπει να λάβουμε υπόψιν τον τύπο του κόμβου. Αν ο κόμβος είναι crisp, δηλαδή παίρνει τιμές μόνο αληθής ή ψευδής και έχει δείκτη ικανοποιησιμότητας s, τότε ορίζουμε τις τιμές:

$$w_L(s) = \begin{cases} 1 & , s = 0 \\ 0 & , s \neq 0 \end{cases} \qquad w_H(s) = \begin{cases} 0 & , s \neq 100 \\ 1 & , s = 100 \end{cases}$$

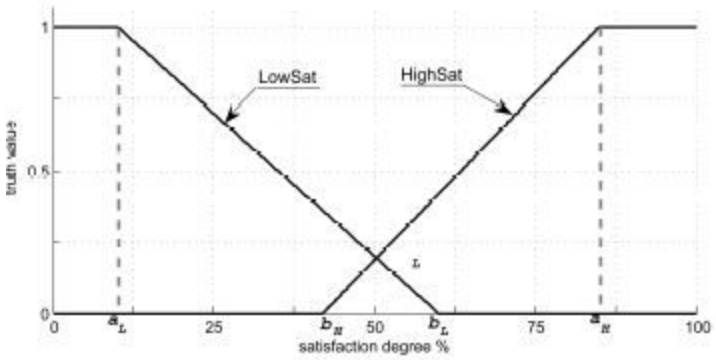
Όπου 0 σημαίνει ψευδής ενώ 1 αληθής. Στην περίπτωση όμως που ο κόμβος είναι fuzzy με δείκτη ικανοποιησιμότητας s, δηλαδή παίρνει τιμές στο διάστημα [0, 1] οι αντίστοιχες τιμές είναι:

$$w_L(s; a_L, b_L) = \begin{cases} 1 & , s \leq a_L \\ \frac{b_L - s}{b_L - a_L} & , a_L \leq s \leq b_L \\ 0 & , b_L \leq s \end{cases}$$

$$w_H(s; a_H, b_H) = \begin{cases} 0 & , s \leq b_H \\ \frac{s - b_H}{a_H - b_H} & , b_H \leq s \leq a_H \\ 1 & , a_H \leq s \end{cases}$$

Όπου a_H, a_L, b_H, b_L είναι παράμετροι που πρέπει να δοθούν από πριν. Συνεπώς ανεξάρτητα από τον τύπο του κόμβου, για κάθε χαρακτηριστικό παράγονται δύο fuzzy όροι, οι w_L και w_H.

Παρακάτω παρατίθεται σχήμα που δείχνει τη σχέση μεταξύ του LowSat και του HighSat.



Σχήμα 5.3: Σχέση LowSat και HighSat

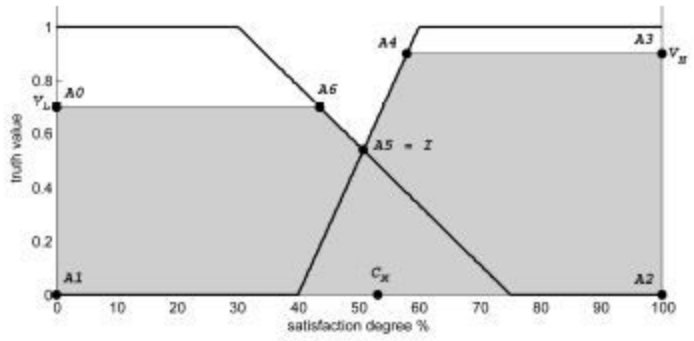
5.1.3.3 Παραγωγή συμπερασμάτων

Αφού γίνει το fuzzification ο reasoner εκτελείται και δίνει τιμές στα HighSat και LowSat όλων των ριζικών κόμβων για τους οποίους υπάρχει μετρήσιμο αποτέλεσμα. Στη συνέχεια ακολουθεί η διαδικασία του defuzzification.

5.1.3.4 Defuzzification

Αυτή η διαδικασία εξαρτάται από τον τύπο του κόμβου. Πιο συγκεκριμένα για crisp κόμβους, αν HighSat = 1.0 και LowSat = 0.0, τότε η τιμή είναι αληθής, ενώ για HighSat = 0.0 και LowSat = 1.0 η τιμή είναι ψευδής. Σε κάθε άλλη περίπτωση δεν μπορούμε να βγάλουμε συμπέρασμα για τον συγκεκριμένο κόμβο.

Στην περίπτωση που ο κόμβος είναι fuzzy ακολουθείται μια πιο εξειδικευμένη διαδικασία, όπου υπολογίζεται το κέντρο βαρύτητας κάτω από τις συναρτήσεις w_L και w_H, όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 5.4: Defuzzification, κέντρο βαρύτητας

Οπότε στο συγκεκριμένο παράδειγμα του σχήματος, όπου $a_L = 30,\ b_L = 75,\ b_H = 40,\ a_H = 60$ και δεδομένου ότι οι τιμές το LowSat και HighSat είναι $V_L = 70$ και $V_H = 90$ αντίστοιχα, βλέπουμε ότι σχηματίζεται το πολύγωνο A0,..., A6. Μπορούμε να βρούμε τη συνιστώσα x του κέντρου βάρους χρησιμοποιώντας την παρακάτω εξίσωση και έτσι βρίσκουμε τη ζητούμενη defuzzified τιμή.

$C_x(P) = \frac{1}{6E(P)} \sum_{k=0}^{m-1} (x_k + x_{k+1})(x_k y_{k+1} - x_{k+1} y_k)$, όπου $E(P) = \frac{1}{2} \sum_{k=0}^{m-1} (x_k y_{k+1} - x_{k+1} y_k)$

και $x_k,\ y_k$ τα αντίστοιχα σημεία της εκάστοτε κορυφής του πολυγώνου. Οπότε στο παραπάνω παράδειγμα βρίσκουμε $C_x = 53.17\%$.

5.2 Εφαρμογή του Reasoning στα affordances.

Αφού λοιπόν περιγράψαμε σε γενικές γραμμές πως γίνεται το reasoning στα δέντρα στόχων πρέπει να περιγράψουμε με ποιον τρόπο χρησιμοποιείται στην παρούσα διπλωματική σε σχέση με τα affordances.

Όπως είναι φανερό και από το κεφάλαιο 4 όπου περιγράφονται όλες οι σχετικές κλάσεις με τα δέντρα στόχων/μοντέλα και τα affordances, κάθε δέντρο που δημιουργείται περιέχει ένα affordance. Οπότε δημιουργείται το ερώτημα για το πως αρχικοποιούνται οι τιμές που αφορούν ένα ερώτημα, πως επιλέγονται τα μοντέλα στα οποία πρέπει να εφαρμοστεί reasoning, πως γίνεται η σύγκριση ώστε να δοθούν τιμές στα φύλλα που να έχουν νόημα και τέλος αφού εφαρμοστεί το reasoning ποια μοντέλα και κατά συνέπεια affordances επιλέγονται για να προστεθούν στον REST πόρο.

5.2.1 Αρχικοποίηση τιμών

Αρχικά η αρχικοποίηση των τιμών (`ContextValue`) γίνεται χρησιμοποιώντας τη μέθοδο `populate` του `AffordanceContext`, η οποία χρησιμοποιεί τους αντίστοιχους `ContextManagers` που αναθέτουν τις κατάλληλες τιμές.

5.2.2 Επιλογή μοντέλων

Για να βρούμε ποια από όλα τα μοντέλα πρέπει να επιλεχθούν προς reasoning χρησιμοποιούμε το `AffordanceContext`, όπως ειπώθηκε και σε προηγούμενα κεφάλαια. Στο κεφάλαιο 4 περιγράφηκε αναλυτικά η κλάση αυτή και το πεδίο `isDefinedBy`. Αυτό το πεδίο είναι το σύνολο όλων των πόρων που σχετίζονται με το affordance. Επίσης κάθε μοντέλο έχει το δικό του σύνολο πόρων το οποίο μπορεί εύκολα να εξαχθεί από τα φύλλα του ως η ένωση όλων των πόρων και χαρακτηριστικών που αφορά το κάθε φύλλο.

Οπότε είναι εύκολα κατανοητό πως από το σύνολο όλων των μοντέλων τα υποψήφια μοντέλα προς reasoning είναι εκείνα τα οποία το σύνολο των πόρων τους είναι υποσύνολο του συνόλου που περιγράφει το πεδίο `isDefinedBy`, διαφορετικά δεν θα είχε νόημα να τεθεί προς reasoning ένα μοντέλο το οποίο περιέχει πόρους ή χαρακτηριστικά τα οποία δεν υπάρχουν σε μια συγκεκριμένη κατάσταση που περιγράφεται από το `AffordanceContext`.

Ένα ακόμα πεδίο του `AffordanceContext` είναι το `cnfList`, το οποίο αφορά μια CNF μορφή στην οποία αντί για λογικούς όρους περιέχονται πόροι REST. Αν για παράδειγμα έχουμε την CNF πρόταση: $C = (a \vee b) \wedge (c \vee d)$, όπου τα a, b, c, d είναι πόροι REST, τότε κάποιο μοντέλο που περιέχει στα φύλλα του, τους πόρους a, d θα ικανοποιούσε την C και έτσι θα ήταν υποψήφιο. Οπότε από τα προηγουμένως φιλτραρισμένα μοντέλα με βάση το `isDefinedBy`, χρησιμοποιούμε την CNF πρόταση (μέθοδος `satisfiesCNF` του `AffordanceContext`) για να πάρουμε μόνο όλα τα μοντέλα που ικανοποιούν αυτήν την πρόταση. Όπως αναφέρθηκε και στο κεφάλαιο 4, ο λόγος που γίνεται αυτό είναι διότι αν ένα μοντέλο έχει για παράδειγμα μόνο ένα φύλλο και κατά συνέπεια μόνο ένα χαρακτηριστικό/πόρο, τότε είναι πιο εύκολα ικανοποιήσιμο από κάποιο άλλο μοντέλο με περισσότερα φύλλα.

5.2.3 Σύγκριση και ανάθεση τιμών στα φύλλα πριν το Reasoning

Η κλάση AffordanceModel και όλες οι υποκλάσεις της παρέχουν το μέθοδο `getInitialValues`. Αυτή η μέθοδος χρησιμοποιεί την κλάση CompareCalculator για να υπολογίσει μια τιμή για τη σύγκριση ανάμεσα στην τιμή έχει αναθέσει ο διαχειριστής όταν δημιούργησε το μοντέλο και στην τιμή προς σύγκριση που υπολογίστηκε από το προηγούμενο βήμα. Αρχικά χρησιμοποιείται η

μέθοδος compare του `AffordanceContext`, όπου μέσω του κατάλληλου ContextManager συγκρίνει αυτές τις δύο τιμές και επιστρέφει ένα αποτέλεσμα το οποίο είναι -1, 0 ή 1. Στη συνέχεια ο `CompareCalculator` αναλαμβάνει να μετατρέψει αυτήν την τιμή με βάση τον τύπο της σύγκρισης και το βάρος του φύλλου σε μία τιμή από το 0 έως το 100 και αναθέτει αυτήν την τιμή στο φύλλο.
Κατ' αυτόν τον τρόπο γίνεται η αρχικοποίηση των τιμών στα φύλλα του μοντέλου.

5.2.4 Επιλογή κατάλληλων affordances

Αφού γίνει ανάθεση τιμών στα φύλλα εκτελείται η μέθοδος `applyReasoning` της αφηρημένης κλάσης Model η οποία εκτελεί το reasoning όπως περιγράφηκε παραπάνω σε αυτό το κεφάλαιο.
Αφού λοιπόν εκτελεστεί το reasoning και δοθούν τιμές στις ρίζες του κάθε μοντέλου, έχουμε μία λίστα η οποία περιέχει όλα τα μοντέλα με τα affordances και τις τιμές στις ρίζες, από 0 έως 100. Χρησιμοποιώντας κάποια κλάση που υλοποιεί τη διεπαφή `AffordancePolicy` μέσω της μεθόδου `filter`, επιλέγονται κάποια από τα μοντέλα και στη συνέχεια προστίθενται τα affordances τους στην RESTαπάντηση.

Ένα παράδειγμα ενός `AffordancePolicy` θα μπορούσε να είναι για παράδειγμα να επιστρέφεται το μοντέλο με την υψηλότερη τιμή, ή τα 10 καλύτερα εξ' αυτόν, ή ακόμα και όλα ταξινομημένα σε αύξουσα σειρά, ή όλα με βάση κάποιο κατώτατο όριο.

5.2.5 Αλγόριθμος

Στη συνέχεια παρατίθεται ψευδοκώδικας με όσα περιγράφηκαν παραπάνω.

```
Input : M: models, A: AffordanceContext, P: AffordancePolicy
Output : Aff : affordances
1: for all context_value in context_values(A) do
2:     populate(A, context_value)
3: end for
4: for all m in M do
5:     if not context_keys(m) ⊆ isDefinedBy(A)
6:           remove m from M
7:     end if
8:     if not satisfiesCNF(A, context_keys(m))
9:           remove m from M
10:    end if
```

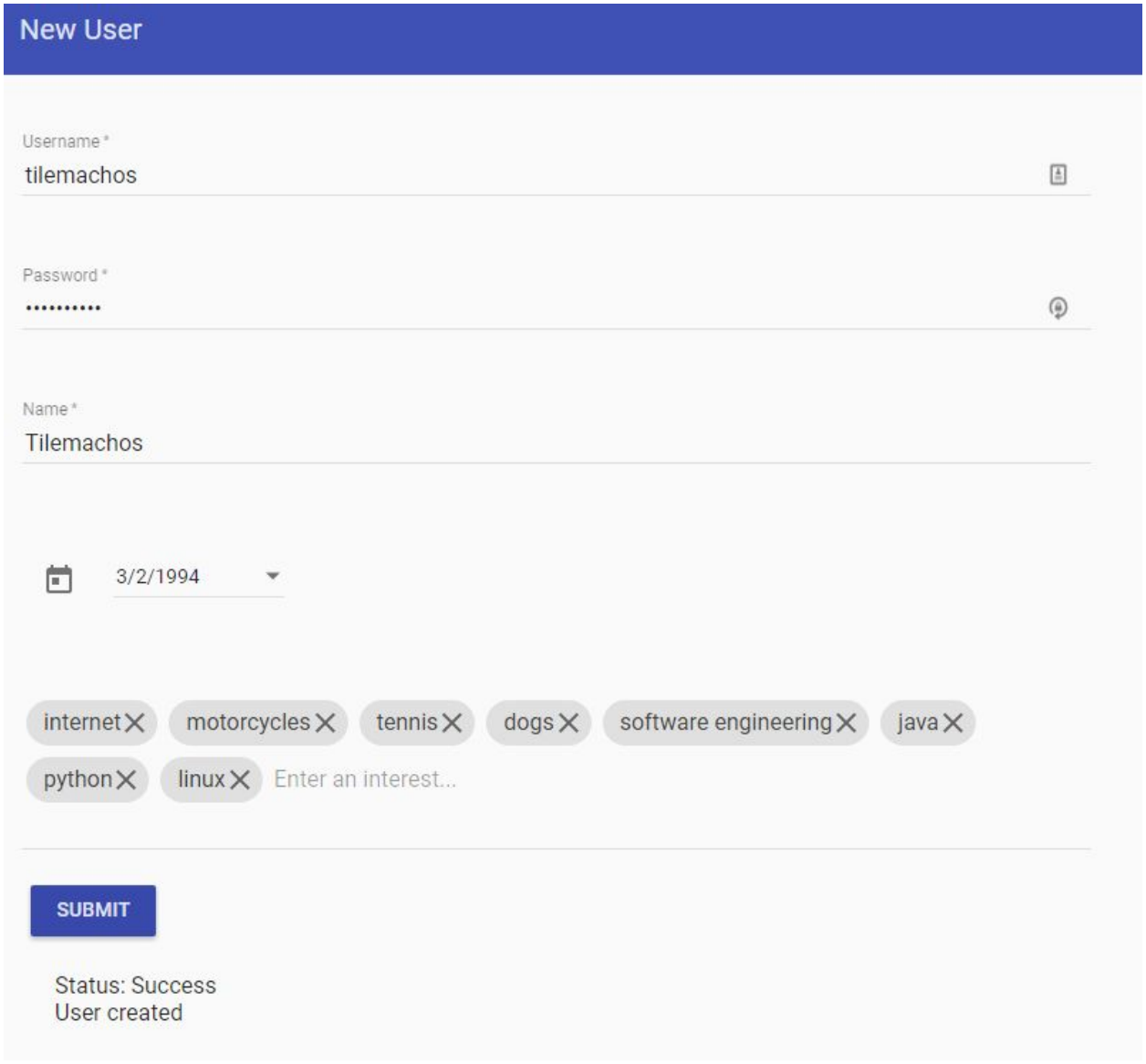
```
11: end for
12: for all m in M do
13:    applyReasoning(A, getInitialValues(A, m))
14: end for
15: M← filter(P, M)
16: Aff← []
17: for all m in M
18:    add affordance(m) in Aff
19: end for
20: return Aff
```

Κεφάλαιο 6: Παραδείγματα χρήσης και μετρικές

6.1 Παραδείγματα χρήσης

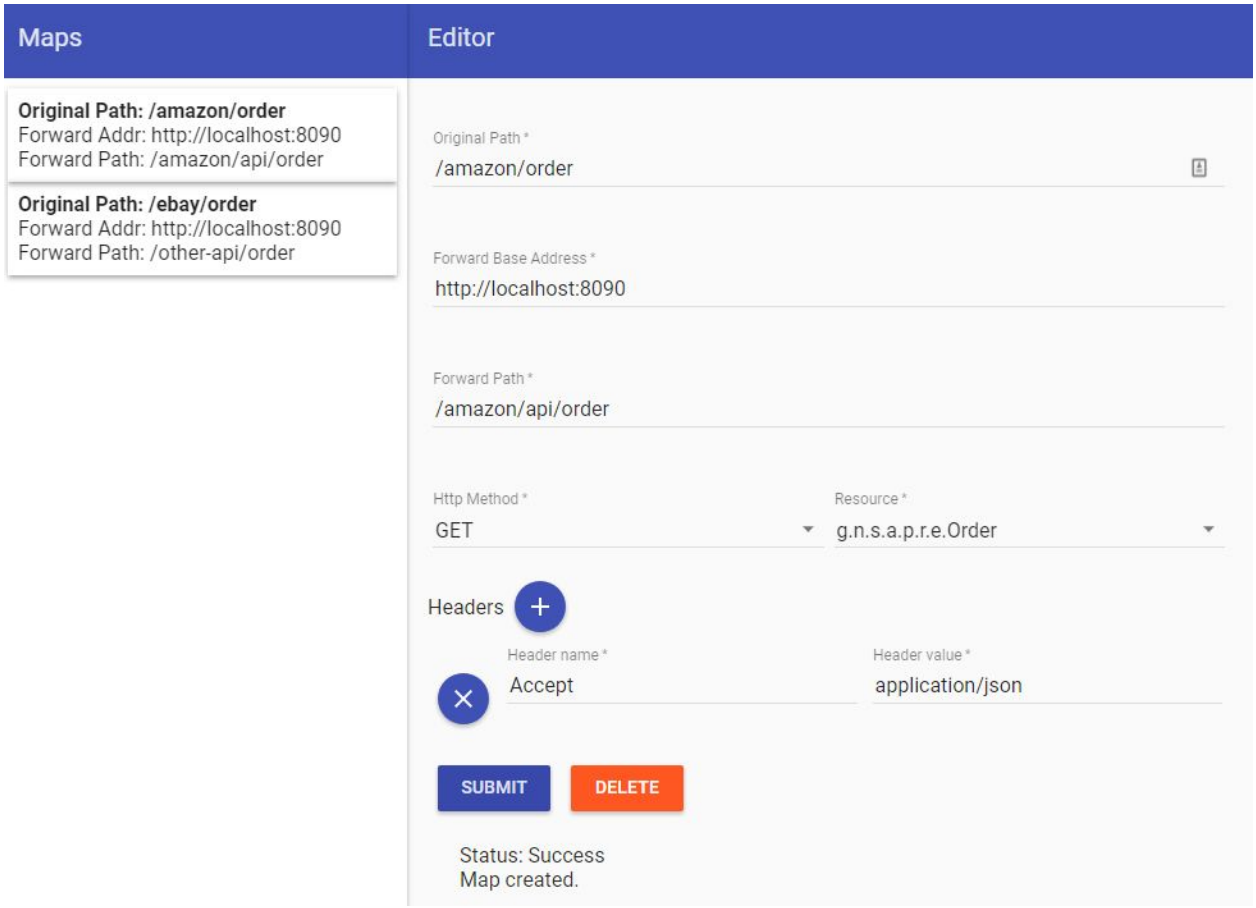
Σε αυτό το κεφάλαιο παρουσιάζονται μερικά παραδείγματα χρήσης του συστήματος.

Αρχικά ξεκινάμε να δημιουργήσουμε έναν νέο χρήστη (το interface αυτό δεν αποτελούσε μέρος της διπλωματικής, απλά δημιουργήθηκε για το σκοπό αυτού του κεφαλαίου). Στο path `/user` του περιηγητή εμφανίζεται η ακόλουθη σελίδα όπου μπορούμε να δημιουργήσουμε έναν νέο χρήστη με username, password, πραγματικό όνομα, ημερομηνία γέννησης και τα ενδιαφέροντα του χρήστη.



Σχήμα 6.1: Δημιουργία νέου χρήστη

Αφού δημιουργήσουμε το χρήστη, ο διαχειριστής μπορεί να επισκεφθεί το path `/admin/mapper` και να δημιουργήσει ένα νέο mapping, αφού πρώτα δώσει τα δικά του username και password.



Σχήμα 6.2: Δημιουργία νέου mapping

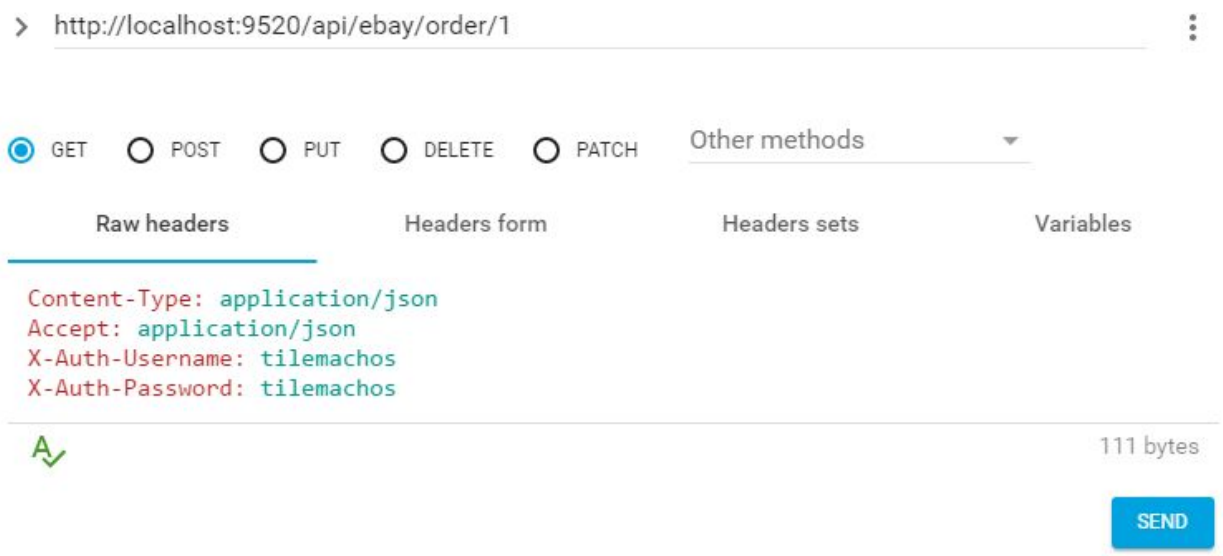
Στη συνέχεια μπορεί μέσω του path `/admin/goalmodel` να δημιουργήσει ένα goal model μέσω της παρεχόμενης διεπαφής χρήστη, όπως φαίνεται στο παρακάτω σχήμα.

Σχήμα 6.3: Δημιουργία goal model

Αφού δημιουργηθεί ο νέος χρήστης, το mapping και το καινούργιο goal model, μπορούμε να δοκιμάσουμε να κάνουμε api requests στο σύστημα.

Για παράδειγμα στο path `/api/ebay/order/1` το οποίο αντιστοιχεί το σύστημα εσωτερικά σε μία άλλη υπηρεσία που δημιουργήθηκε για τους σκοπούς αυτής της διπλωματικής εργασίας, μπορούμε να ζητήσουμε την παραγγελία με αναγνωριστικό `1`. Όπως παρατηρείται και στη συνέχεια επιβάλλεται να δώσουμε τις επικεφαλίδες X-Auth-Username και X-Auth-Password ως διαπιστευτήρια, ενώ μπορούμε να δώσουμε και άλλες επικεφαλίδες που αφορούν τη μορφή της απάντησης και του περιεχομένου που στέλνουμε και λαμβάνουμε.

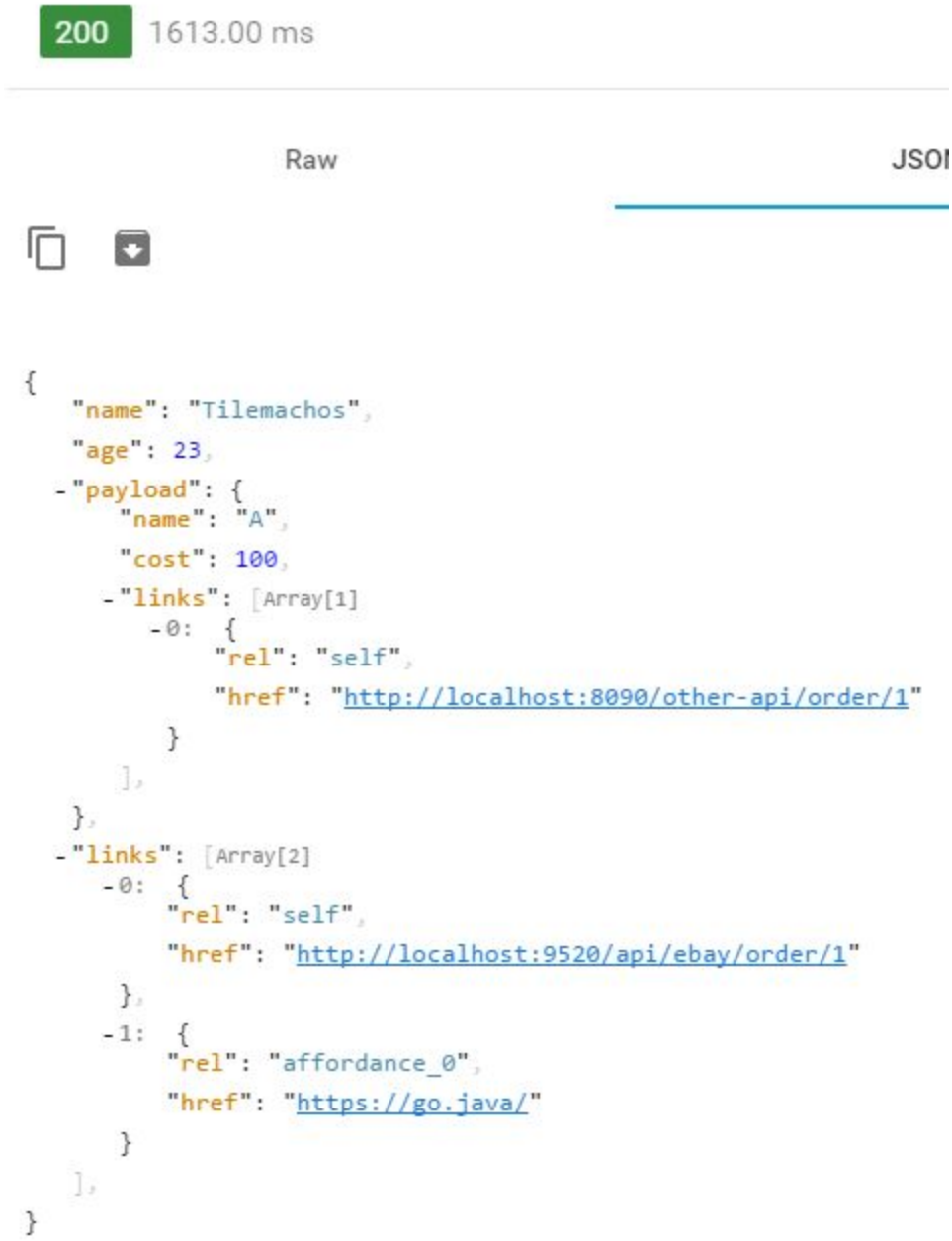
Παρακάτω φαίνεται το αντίστοιχο ερώτημα



Σχήμα 6.4: Παράδειγμα REST api ερωτήματος

Σε περίπτωση που ο συγκεκριμένος πόρος, δηλαδή το order με αναγνωριστικό 1 υπάρχει, τότε θα πάρουμε μια απάντηση η οποία θα έχει μέσα διάφορα στοιχεία για το χρήστη, ένα πεδίο `payload` που είναι το περιεχόμενο της απάντησης της εξωτερικής υπηρεσίας, τον/τους συνδέσμους που αφορούν αυτό το ερώτημα, καθώς και ένα ή περισσότερα affordances. Στην παρακάτω εικόνα παρατηρούμε ότι το affordance που λάβαμε σχετίζεται με τη java. Αυτό συνέβη διότι ο χρήστης `tilemachos` που χρησιμοποιήθηκε στο παράδειγμα έχει ενδιαφέροντα το `software engineering` και τη `java` και το μεγαλύτερο βαθμό ικανοποιησιμότητας είχε το goal model που δημιουργήθηκε στο παραπάνω παράδειγμα.

Ακολουθεί η σχετική εικόνα.



Σχήμα 6.5: Παράδειγμα απάντησης σε REST api ερώτημα χωρίς προβλήματα

Ωστόσο σε περίπτωση που στο ερώτημα λείπει τουλάχιστον μία από τις επικεφαλίδες X-Auth-Username ή X-Auth-Password, ή ακόμα και αν το περιεχόμενό τους δεν αντιστοιχεί σε κάποιον χρήστη, τότε θα λάβουμε απάντηση με κωδικό σφάλματος `407: Proxy Authentication Required` όπως φαίνεται παρακάτω.

Σχήμα 6.6: Παράδειγμα απάντησης σε REST api ερώτημα με λανθασμένα διαπιστευτήρια χρήστη

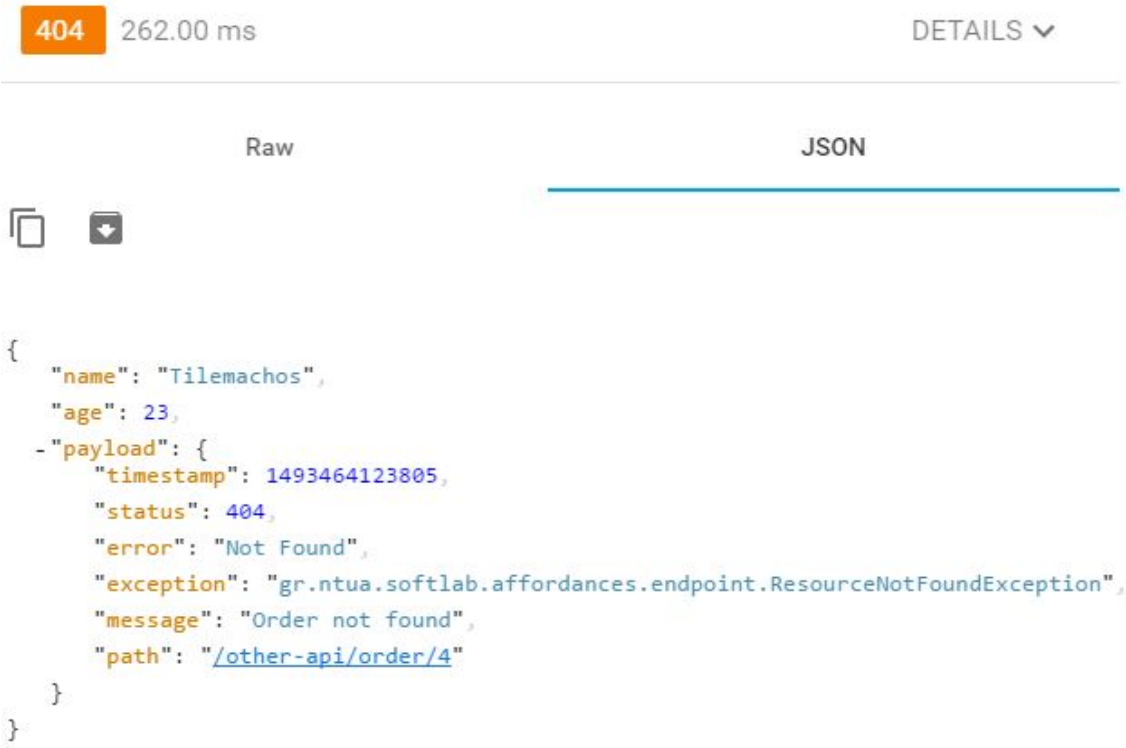
Επίσης αν στο αρχικό ερώτημα δώσουμε ένα path για τον οποίο το σύστημα δεν έχει αντίστοιχο mapping, όπως το `/api/doesnotexist/1`, τότε θα λάβουμε απάντηση με κωδικό σφάλματος `404: Resource not found`.

Αν ζητήσουμε από το σύστημα να μας επιστρέψει την αναπαράσταση ενός πόρου (resource) που η εξωτερική υπηρεσία δεν παρέχει, θα πάρουμε απάντηση με τον ίδιο κωδικό σφάλματος με τη διαφορά ότι το μήνυμα σφάλματος θα βρίσκεται μέσα στο πεδίο payload που αφορά τα δεδομένα της εξωτερικής υπηρεσίας.

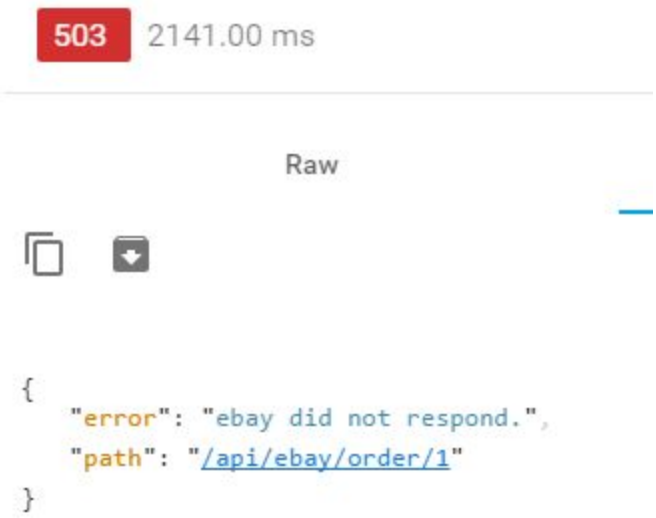
Τέλος στην περίπτωση που το mapping είναι σωστό, αλλά η εξωτερική υπηρεσία δεν ανταποκρίνεται, είτε γιατί υπάρχουν προβλήματα δικτύου, είτε γιατί αυτή παρουσίασε κάποιο σφάλμα τότε θα πάρουμε κάποιον κωδικό σφάλματος που ξεκινάει από 5 (5xx).

Οι παραπάνω περιπτώσεις φαίνονται σε εικόνες στη συνέχεια.

Σχήμα 6.7: Το αντίστοιχο mapping δεν υπάρχει



Σχήμα 6.8: Το resource στην εξωτερική υπηρεσία δεν υπάρχει



Σχήμα 6.9: Παρουσιάστηκε κάποιο σφάλμα στην εξωτερική υπηρεσία

6.2 Μετρικές

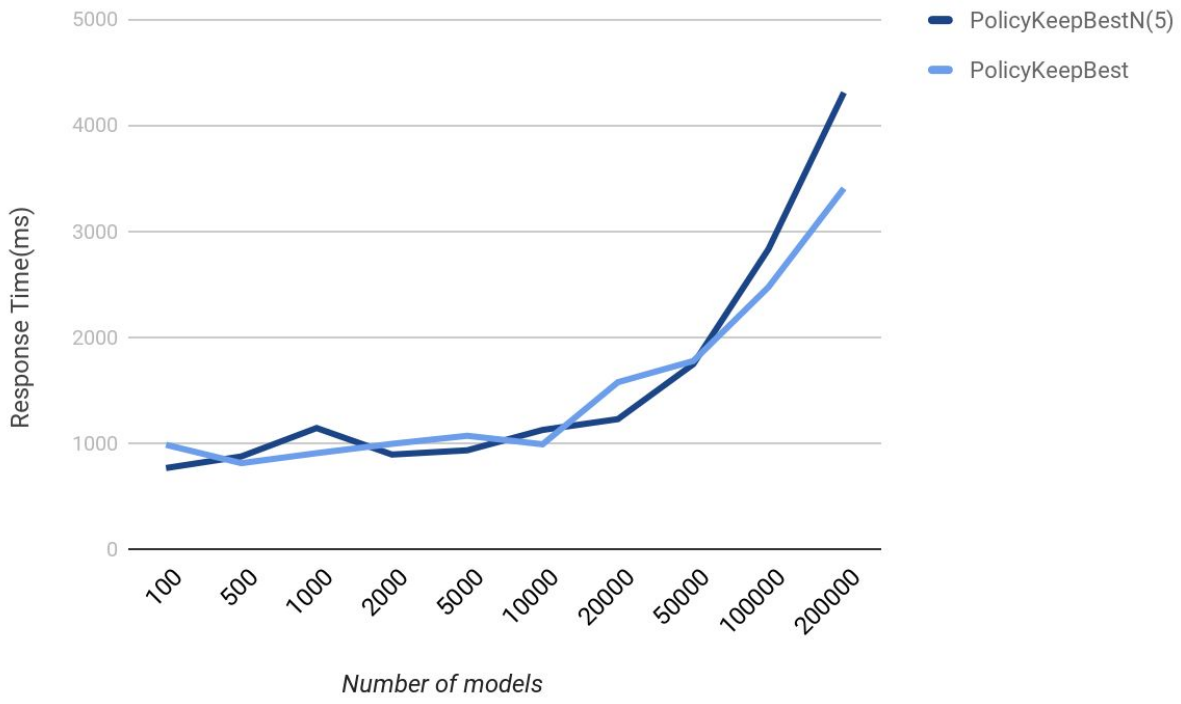
Στη συνέχεια παραθέτουμε μερικές μετρικές που πάρθηκαν για να καταλήξουμε σε συμπεράσματα σχετικά με το πόσο ικανοποιητικά δουλεύει το σύστημα. Οι μετρικές αυτές είναι τρεις και αφορούν χρόνους απόκρισης σε σχέση με αριθμό μοντέλων και αριθμό πελατών κυρίως.

Οι μετρικές πραγματοποιήθηκαν σε σύστημα με διπύρηνο επεξεργαστή ταχύτητας 3.16 Ghz και μνήμη 8.00 GB DDR3 χρονισμένη στα 2000 Mhz. Τα ανώτερα όρια τιμών που χρησιμοποιήθηκαν έγιναν με βάση το πότε το σύστημα άρχισε να παρουσιάζει κορεσμό όσον αφορά την επεξεργαστική ισχύ και την εναπομένουσα χωρητικότητα σε μνήμη.

6.2.1 Σχέση χρόνου απόκρισης προς αριθμό δέντρων στόχων που είναι αποθηκευμένα στο σύστημα

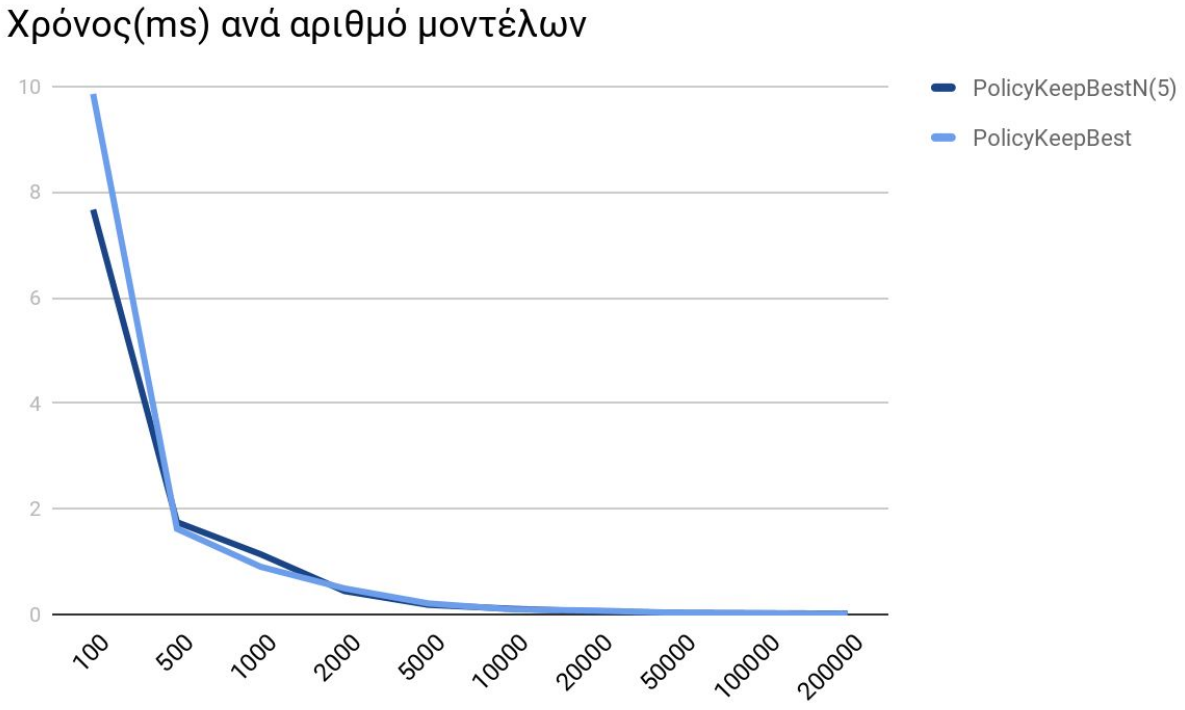
Σε αυτή τη μετρική μετρήσαμε το χρόνο απόκρισης ενός ερωτήματος σε σχέση με τον αριθμό των μοντέλων που είναι αποθηκευμένα στο σύστημα. Εδώ πρέπει να παρατηρηθεί ότι ο αριθμός των μοντέλων που είναι αποθηκευμένα στο σύστημα δεν έχει άμεση σχέση με τον αριθμό των μοντέλων που τελικά θα επιλεχθούν για να γίνουν evaluate σύμφωνα με το `AffordanceContext`.

Παρακάτω δίνεται το σχετικό διάγραμμα με τον κάθετο άξονα να παριστάνει το χρόνο απόκρισης και τον οριζόντιο, τον αριθμό των μοντέλων. Έγιναν δύο μετρήσεις: Η πρώτη μέτρηση έγινε με χρήση της πολιτικής PolicyKeepBestN(5), δηλαδή κρατώντας τα πέντε affordances με βάση το αντίστοιχο υψηλότερο evaluation. Η πολιτική αυτή χρησιμοποιεί ταξινόμηση και κρατάει τα καλύτερα N affordances. Η δεύτερη πολιτική που χρησιμοποιήθηκε είναι η PolicyKeepBest η οποία επιλέγει το affordance με το υψηλότερο evaluation.



Σχήμα 6.10: Χρόνος απόκρισης προς αριθμό μοντέλων

Παρά το γεγονός ότι στο παραπάνω διάγραμμα οι δύο γραμμές φαίνονται να είναι εκθετικές στην πραγματικότητα δεν είναι διότι ο οριζόντιος άξονας δεν έχει τιμές γραμμικά κατανεμημένες. Αυτό μπορεί να φανεί καλύτερα και από το παρακάτω διάγραμμα που απεικονίζει χρόνο απόκρισης ανά αριθμό μοντέλων.



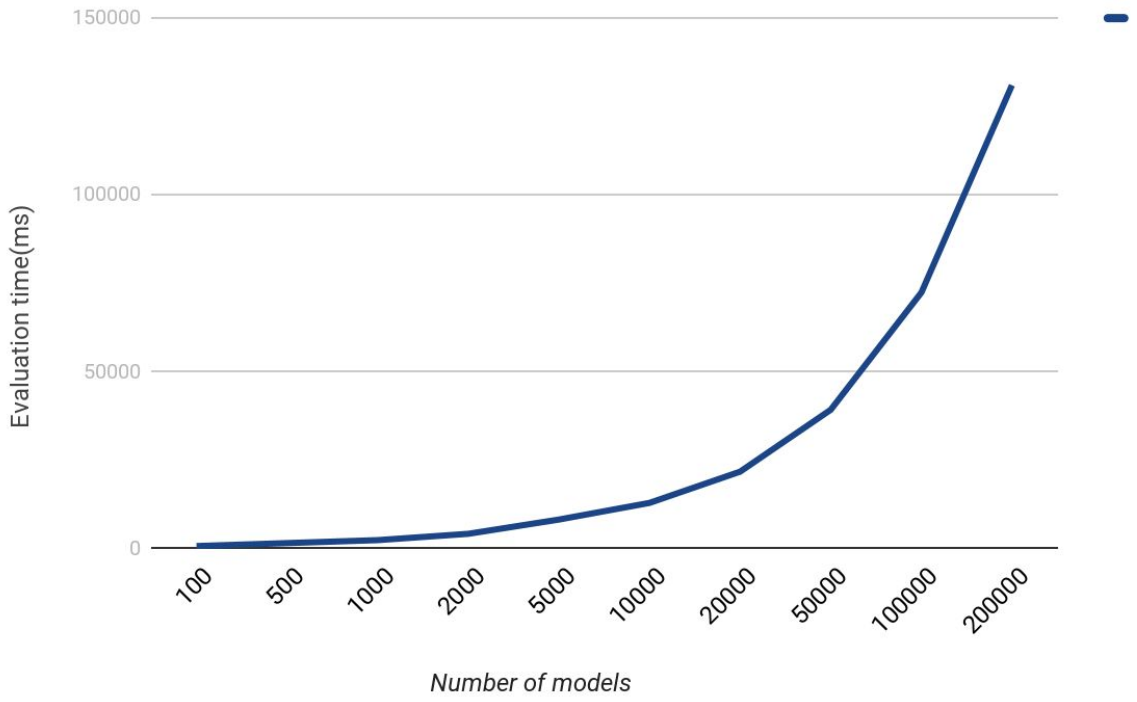
Σχήμα 6.11: Χρόνος απόκρισης ανά αριθμό μοντέλων προς αριθμό μοντέλων

Από αυτή τη γραφική παράσταση φαίνεται ξεκάθαρα ότι όχι μόνο ο χρόνος απόκρισης δεν είναι εκθετικός, αλλά είναι και καλύτερος από γραμμικός.

6.2.2 Σχέση χρόνου evaluation προς αριθμό δέντρων στόχων προς evaluation

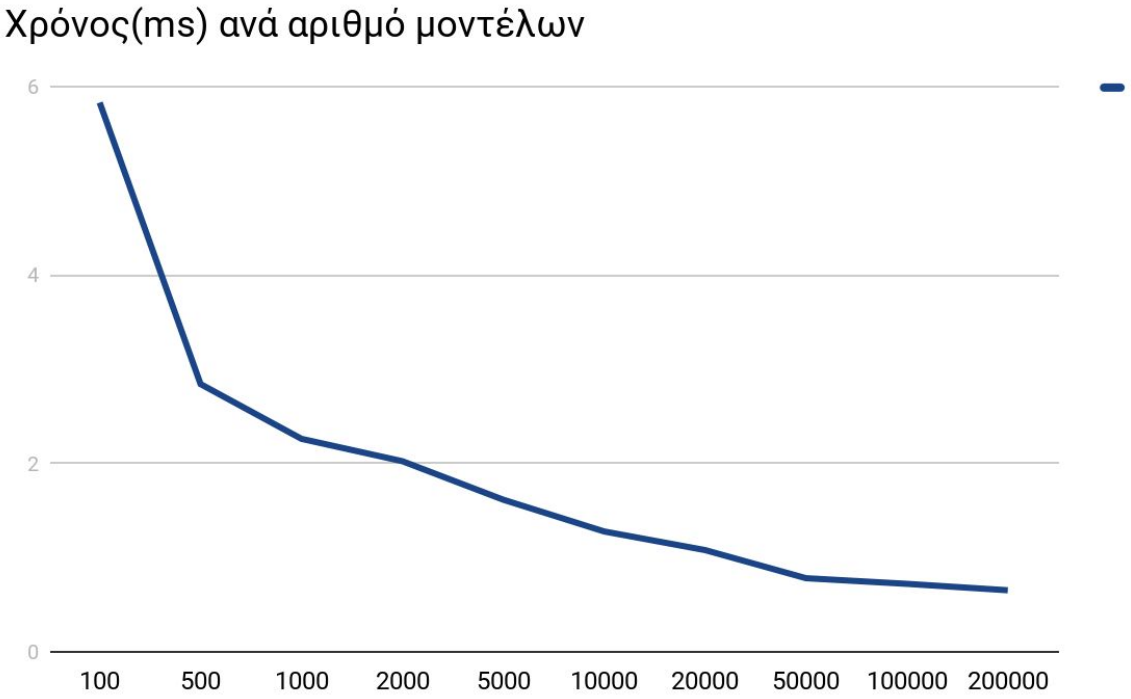
Σε αυτή τη μετρική μετρήσαμε πόσο χρόνο χρειάζεται για να γίνει το evaluate όλων των μοντέλων που έχουν επιλεχθεί, σε σχέση με τον αριθμό των μοντέλων που έχουν επιλεχθεί προς evaluation. Δηλαδή τον αριθμό τον μοντέλων αφού γίνει το αντίστοιχο φιλτράρισμα με βάση το `AffordanceContext`.

Παρακάτω δίνεται το σχετικό διάγραμμα με τον κάθετο άξονα να παριστάνει τον χρόνο του evaluation και τον οριζόντιο τον αριθμό των μοντέλων προς evaluation.



Σχήμα 6.12: Χρόνος evaluation προς αριθμό μοντέλων

Εδώ παρατηρούμε πάλι μια γραφική παράσταση που θυμίζει εκθετική συνάρτηση, αλλά επειδή ο αριθμός των μοντέλων στον οριζόντιο άξονα δεν είναι γραμμικός, στην πραγματικότητα δεν είναι. Αυτό φαίνεται καλύτερα από το παρακάτω σχήμα όπου απεικονίζεται ο χρόνος απόκρισης ανά αριθμό μοντέλων προς αριθμό μοντέλων.



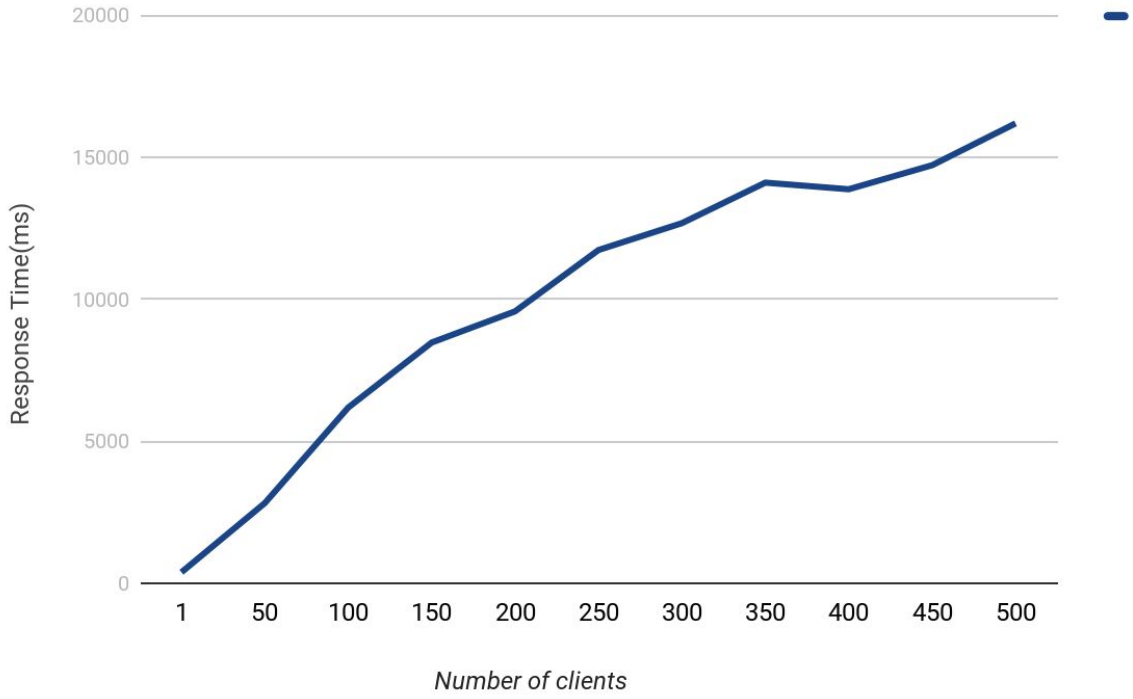
Σχήμα 6.13: Χρόνος evaluation ανά αριθμό μοντέλων προς αριθμό μοντέλων.

Από αυτή τη γραφική παράσταση φαίνεται ξεκάθαρα ότι όχι μόνο ο χρόνος απόκρισης δεν είναι εκθετικός, αλλά είναι και καλύτερος από γραμμικός.

6.2.3 Σχέση χρόνου απόκρισης προς αριθμό πελατών

Σε αυτή τη μετρική μετρήσαμε το χρόνο απόκρισης σε ένα ερώτημα σε σχέση με τον αριθμό των πελατών που κάνουν ταυτόχρονα ερωτήματα.

Παρακάτω δίνεται το σχετικό διάγραμμα με τον κάθετο άξονα να παριστάνει το χρόνο απόκρισης ερωτήματος και τον οριζόντιο τον αριθμό των πελατών που κάνουν ταυτόχρονα ερωτήματα.



Σχήμα 6.14: Χρόνος απόκρισης προς αριθμό πελατών

Σε αυτήν την περίπτωση ακολουθήσαμε γραμμική κατανομή τιμών στον οριζόντιο άξονα, και με βάση αυτή τη γραφική παράσταση μπορούμε να πούμε ότι ο χρόνος απόκρισης προς τον αριθμό πελατών είναι περίπου γραμμικός.

Κεφάλαιο 7: Μελλοντικές εργασίες

7.1 EMF - Modeling Frameworks

Μία μελλοντική επέκταση του περιβάλλοντος πλαισίου που δημιουργήθηκε στα πλαίσια αυτής της διπλωματικής είναι η χρήση κάποιου modeling framework. Το πιο δημοφιλές είναι το EMF, όπως περιγράφηκε στο κεφάλαιο 2. Αυτή τη στιγμή οι REST πόροι έχουν τη μορφή κλάσεων. Έτσι αν για παράδειγμα υπάρχει ένας REST πόρος σε μια εξωτερική υπηρεσία με τη μορφή μίας κλάσης, τότε θα πρέπει να υπάρχει μία αντίστοιχη κλάση στον proxy server με τα ίδια τουλάχιστον πεδία. Κατά συνέπεια, αν ο διαχειριστής θέλει να προσθέσει κάποιον καινούργιο πόρο, θα πρέπει να προσθέσει την κατάλληλη κλάση στο σύστημα (και τον αντίστοιχο ContextManager φυσικά) ώστε αυτό να μπορεί να επικοινωνήσει με την εξωτερική υπηρεσία χρησιμοποιώντας αυτόν τον πόρο.
Με τη χρήση του EMF και ίσως κάποιου διαδεδομένου τρόπου μοντελοποίησης πόρων όπως είναι το rdf, θα μπορούσε αυτή η διαδικασία να αυτοματοποιηθεί. Για παράδειγμα η εξωτερική υπηρεσία μπορεί να αναφέρει τη μορφή σε rdf έχει κάποιος πόρος της σε κάποιον σύνδεσμο και το σύστημα, χρησιμοποιώντας το EMF να παράγει την αντίστοιχη κλάση, αντί για το διαχειριστή.

Άλλο γνωστό modeling framework που θα μπορούσε να πραγματοποιηθεί είναι το KMF [14], το οποίο παρά του ότι δεν προσφέρει κάποιο εύχρηστο περιβάλλον όπως το EMF, παρέχει περισσότερες λειτουργίες, ειδικότερα όσον αφορά την επικοινωνία μεταξύ υπηρεσιών σε περίπτωση που γίνει κάποια αλλαγή σε κάποιο μοντέλο, όπως επίσης και σύγχρονες μεθόδους όπως το persistence και το lazy loading.

7.2 Βελτίωση του mapper

Μία ακόμα μελλοντική βελτίωση του υπάρχοντος περιβάλλοντος πλαισίου θα μπορούσε να είναι αυτή στον mapper. Αυτή τη στιγμή ο mapper μπορεί να χαρτογραφήσει υπηρεσίες οι οποίες ακολουθούν απλή λογική στους συνδέσμους της, της μορφής service/resource?params, δηλαδή απλούς άμεσους συνδέσμους στους πόρους της συνοδευόμενους από κάποιες παραμέτρους όπου αυτό χρειάζεται. Μερικές υπηρεσίες, ωστόσο, έχουν πολύ πολύπλοκη δομή στους πόρους τους και ενώ αυτό σχεδιαστικά μπορεί να μην είναι απολύτως σύμφωνο με την απλότητα του REST, δεν παύει να αποτελεί ένα εμπόδιο για το υπάρχων υλοποιημένο σύστημα.

7.3 Database Migration

Μία μελλοντική εργασία που θα μπορούσε να πραγματοποιηθεί πάνω σε αυτήν τη διπλωματική είναι αυτή του database migration. Επειδή τα affordances προκύπτουν ως εξατομικευμένοι σύνδεσμοι για κάποιο ερώτημα και κάποιον χρήστη, είναι φυσικό να υπάρχει μία βάση δεδομένων που αποθηκεύει όλες τις επιλογές και τα στοιχεία του χρήστη. Αυτή τη στιγμή αυτή η βάση έχει μία απλή μορφή, ωστόσο μελλοντικά θα μπορούσε να επεκταθεί. Για να γίνει εύκολα κατανοητό θα δώσουμε ένα παράδειγμα. Έστω ότι ένας χρήστης κάνει μία αγορά από μία υπηρεσία Α και στη συνέχεια από μία άλλη υπηρεσία Β χρησιμοποιώντας τον proxy server που υλοποιήθηκε. Η υπηρεσία Α, όμως και η υπηρεσία Β μπορεί να μην έχουν τον ίδιο ακριβώς πόρο για την παραγγελία, ωστόσο είναι λογικό αυτοί οι δύο πόροι να έχουν πολλά κοινά, όπως το ποσό αγοράς, τη μέθοδο πληρωμής του χρήστη, την κατηγορία του προϊόντος κλπ. Μία εργασία που θα αφορούσε το database migration θα μπορούσε να προσφέρει λύση σε αυτό το πρόβλημα ώστε το ίδιο το σύστημα να μπορεί να “γνωρίζει” τις παραγγελίες του χρήστη ακόμα και αν αυτές έχουν γίνει μέσω διαφορετικών εξωτερικών υπηρεσιών. Έτσι αυτόματα η ποιότητα των affordances που θα εμπλουτίζουν τη REST απάντηση θα είναι πολύ καλύτερη δεδομένου ότι θα υπάρχει μεγαλύτερη και πιο εξειδικευμένη γνώση για το χρήστη.

7.4 Εργασίες βασισμένες στο Reasoning δέντρων στόχων

Τέλος εξαιτίας του τρόπου που δομήθηκε η αρχιτεκτονική του συστήματος, δηλαδή η υλοποίηση ενός περιβάλλοντος πλαισίου και στη συνέχεια η χρήση αυτού από τον proxy server, οποιαδήποτε άλλη εργασία αφορά χρήση δέντρων στόχων όπου σε κάθε δέντρο ανατίθεται κάποια ενέργεια ή οντότητα (στην περίπτωσή μας είναι το affordance) θα μπορούσε να χρησιμοποιήσει το περιβάλλον πλαίσιο ή ακόμα και να το επεκτείνει χωρίς να χρειάζεται να δημιουργήσει εκ νέου ένα δικό της περιβάλλον πλαίσιο.

Βιβλιογραφία

[1] Roy Thomas Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[2] REST, https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[3] Jim Webber, Savas Parastatidis, Ian Robinson, “REST in practice”, http://restinpractice.com/book/

[4] Silvia Schreier, “Modeling RESTful applications”

[5] Mahdi Bennara, Michael Mrissa, Youssef Amghar, “An Approach for Composing RESTful Linked Services on the Web”

[6] Mike Amundsen, “From APIs to Affordances: A New Paradigm for Web Services”

[7] Mike Amundsen, “Hypermedia-Oriented Design: An Approach for Supporting Evolvable Distributed Network Applications”

[8] Markus Lanthaler, Christian Gütl, “A Semantic Description Language for RESTful Data Services to Combat Semaphobia”

[9] Ruben Verborgh, Michael Hausenblas, Thomas Steiner, “Distributed Affordance: An Open-World Assumption for Hypermedia”

[10] Michael Petychakis, Fenareti Lampathaki, Dimitrios Askounis, “Adding Rules on Existing Hypermedia APIs”

[11] Dr. Dobb's, “RESTful Web Services: A Tutorial”, http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069

[12] Spring framework, https://projects.spring.io/spring-framework/

[13] Meta Object Facility, http://www.omg.org/mof/

[14] Object Management Group, “OMG Meta Object Facility (MOF) Core Specification”

[15] Object Management Group, “XML Metadata Interchange (XMI) Specification”

[16] W3, “Extensible Markup Language (XML)”, https://www.w3.org/XML/

[17] ECMA International, “The JSON Data Interchange Format”, http://www.json.org/

[18] Mike Kelly, “HAL - Hypertext Application Language“, http://stateless.co/hal_specification.html

[19] JSON-LD, http://json-ld.org/

[20] HYDRA, http://www.hydra-cg.com/, http://www.markus-lanthaler.com/hydra/

[21] Hydra Library for Spring framework, https://github.com/dschulten/hydra-java/tree/master/hydra-spring

[22] Markus Lanthaler, Christian Gütl, “On Using JSON-LD to Create Evolvable RESTful Services”, http://www.markus-lanthaler.com/research/on-using-json-ld-to-create-evolvable-restful-services.pdf

[23] Markus Lanthaler, Christian Gütl, “Hydra: A Vocabulary for Hypermedia-Driven Web APIs”, http://www.markus-lanthaler.com/research/hydra-a-vocabulary-for-hypermedia-driven-web-apis.pdf

[24] Markus Lanthaler, Christian Gütl, “Model Your Application Domain, Not Your JSON Structures”, http://www.markus-lanthaler.com/research/model-your-application-domain-not-your-json-structures.pdf

[25] Markus Lanthaler, “Creating 3rd Generation Web APIs with Hydra”, http://www.markus-lanthaler.com/research/creating-3rd-generation-web-apis-with-hydra.pdf

[26] Distributed Affordances, http://distributedaffordance.org/

[27] http://distributedaffordance.org/publications/ws-rest2013.pdf

[28] George Chatzikonstantinou , Kostas Kontogiannis, “Run-Time Requirements Verification for Reconfigurable Systems”

[29] Eclipse Modeling Framework (EMF), http://www.eclipse.org/modeling/emf/

[30] Kevoree Modeling Framework, http://modeling.kevoree.org/

[31] François Fouquet, Grégory Nain, Brice Morin, Jean-Marc Jézéquel Kevoree Modeling Framework (KMF): Efficient modeling techniques for runtime use

[32] A. Lapouchnian. Goal-Oriented Requirements Engineering: An Overview of the Current Research. Depth Report, University of Toronto, 2005.

[33] Yu E, Mylopoulos J. Why goal-oriented requirements engineering. In: Dubois E, Opdahl AL, Pohl K (eds). Proceedings of the 4th international workshop on requirements engineering: foundations of software quality, Pisa, Italy, 8–9 June 1998, Pisa, Italy. Presses Universitaires de Namur, 1998, pp 15–22

[34] Dardenne, A., van Lamsweerde A. and Fickas, S., “Goal Directed Requirements Acquisition”, Science of Computer Programming, Vol. 20, No. 1-2, pp. 3–50, 1993

[35] ITU - Telecommunication Standardization Sector Draft Specification of the Goaloriented Requirement Language (Z.151), September 2013.

[36] Anton, A.I., Goal Identification and Refinement in the Specification of Software-Based Information Systems, Ph.D. Dissertation, Georgia Institute of Technology, Atlanta GA, 1997.

[37] Mylopoulos, J., Kolp, M., and Castro, J. “UML for Agent-Oriented Software Development: The Tropos Proposal,” in Proceedings of UML 2001.

[38] Rolland, C., Souveyet, C., and Ben Achour, C. “Guiding goal modeling using scenarios,” IEEE Trans. Software Eng., vol. 24, pp. 1055–1071, Dec. 1998.

[39] Yu, E.S.K. “Towards modelling and reasoning support for early-phase requirements engineering,” in Proceedings of the Third IEEE International Symposium on Requirements Engineering, Annapolis, Maryland, January 1997.

[40] J. Mylopoulos, L. Chung, and E. Yu, "From object-oriented to goal-oriented requirements analysis," Commun. ACM, vol. 42, no. 1, pp. 31--37, 1999.

[41] Advanced Rest Client, https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo

[42] Source code, https://bitbucket.org/tchar/affordancesframework, https://bitbucket.org/tchar/distributedaffordances, https://bitbucket.org/tchar/distributedaffordancesserver