Semantic Web

from DR-Device to DR-Prolog

Konsolaki Konstantina
konsolak@csd.uoc.gr

Fafalios Pavlos
fafalios@csd.uoc.gr

Mathioudakis Georgios
gmathiou@csd.uoc.gr

Bouloukakis Georgios
boulouk@csd.uoc.gr

Mavridou Eva
mavridou@csd.uoc.gr

Papantriantafyllou Aggelos
angelpap@csd.uoc.gr

Symeonidoy Danah
simeon@csd.uoc.gr

September 2010

Περιεχόμενα

1. Εισαγωγή

Η ανάπτυξη του Παγκόσμιου Ιστού (World Wide Web) κατέστησε το διαδίκτυο προσβάσιμο σε εκατομμύρια χρήστες, επιτρέποντας την απρόσκοπτη δημοσιοποίηση και πρόσβαση σε έγγραφα στο διαδίκτυο. Η εκρηκτική ανάπτυξη του Παγκόσμιου Ιστού δημιούργησε προβλήματα "πληροφοριακής υπερφόρτισης". Η παγκόσμια ερευνητική κοινότητα έχει στραφεί εδώ και λίγα χρόνια σε μία νέα κατεύθυνση εξέλιξης του ιστού, η οποία ονομάζεται "Σημασιολογικός Ιστός" (Semantic Web) και περιλαμβάνει τη σαφή αναπαράσταση του νοήματος των πληροφοριών και των εγγράφων, επιτρέποντας την αυτόματη επεξεργασία και ενοποίηση διαδικτυακών πόρων από "έξυπνα" προγράμματα-πράκτορες. Ο Σημασιολογικός Ιστός θα επιτρέψει τον γρήγορο και ακριβή εντοπισμό πληροφοριών στον παγκόσμιο ιστό καθώς και την ανάπτυξη ευφυών διαδικτυακών πρακτόρων οι οποίοι θα διευκολύνουν την επικοινωνία μεταξύ πληθώρας ετερογενών ηλεκτρονικών συσκευών με πρόσβαση στο διαδίκτυο.

Στόχοι της ομάδας μας ήταν:

- Κατασκευή προτύπου για τα *ruleML* αρχεία.
- Δημιουργία βάση του παραπάνω προτύπου των αρχείων όπου εκφράζονται οι προτιμήσεις και οι απαιτήσεις του «πελάτη», στο παράδειγμα broker – customer.
- Κατασκευή αναλυτή (parser) με σκοπό την μετατροπή των *ruleML* αρχείων σε *DR-Prolog* σύνταξη.
- Επέκταση του *metaprogram* ώστε να υποστηρίζει αριθμητικές παραστάσεις, συγκρίσεις και τον κανόνα *naf.*
- Δημιουργία μηχανής για την εκτέλεση ερωτημάτων και την επιστροφή αποτελεσμάτων με βάση τις προτιμήσεις του «πελάτη».

Στα επόμενα κεφάλαια θα δούμε αναλυτικά πως υλοποιήθηκαν οι παραπάνω στόχοι.

2. Υλοποίηση

2.1 Πρότυπο RuleML

Οι κανόνες που εκφράζουν τις απαιτήσεις *(carlo_1.ruleml)* και προτιμήσεις *(carlo_2.ruleml)* του πελάτη εκφράστηκαν με ένα νέο πρότυπο που δημιουργήσαμε το οποίο είναι επέκταση της RuleML σύνταξης. Για τη δημιουργία του προτύπου βασιστήκαμε στο υπάρχον πρότυπο RuleML που χρησιμοποιεί η DR-Prolog και εν μέρη στα RuleML αρχεία από το Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης. Η σύνταξη που επιλέξαμε για το πρότυπο είναι τέτοια ώστε να είναι εφαρμόσιμη στην DR-Prolog, εύκολα αναγνώσιμη και επεκτάσιμη. Το σχήμα του προτύπου (DTD) επισυνάπτεται στο φάκελο της εργασίας *(dr-prolog.dtd)*.

2.2 Αναλυτής

Για την εξαγωγή των κανόνων DR-Prolog, δημιουργήθηκε ο μετατροπέας *RuleMLparser*. Σκοπός του είναι από τα *RuleML* αρχεία να εξάγει τους *DR-Prolog* κανόνες. Ο μετατροπέας έχει γραφτεί εξ' ολοκλήρου στη γλώσσα Java και αποτελείται από δύο κλάσεις, την κεντρική κλάση *RuleMLparser.java* και την βοηθητική κλάση *Tagger.java*. Δέχεται σαν είσοδο ένα αρχείο *RuleML* και δημιουργεί ένα νέο αρχείο *.P* με τους κανόνες εκφρασμένους σε *DR-Prolog* σύνταξη.

Η κλάση *RuleMLparser.java* περιέχει τέσσερις στατικές μεθόδους. Η πρώτη μέθοδος είναι η "*void createRules(File ruleml)*". Αυτή η μέθοδος διαβάσει το *RuleML* αρχείο και δημιουργεί τους κανόνες. Πιο αναλυτικά, χωρίζει το αρχείο σε διακριτά μέρη έτσι ώστε να ξεχωρίσει τους κανόνες. Στη συνέχεια, χωρίζει κάθε κανόνα σε επιμέρους τμήματα και καλεί τις κατάλληλες συναρτήσεις για να τα επεξεργαστούν. Κάθε επιμέρους συνάρτηση δέχεται ένα από τα παραπάνω τμήματα και παράγει το αντίστοιχο μέρος του κάθε κανόνα. Στο τέλος, όλα τα τμήματα ενώνονται, σχηματίζοντας έτσι τους τελικούς κανόνες. Για παράδειγμα, από το παρακάτω κομμάτι *RuleML* σύνταξης (σχήμα 1), παράγεται ο κανόνας του σχήματος 2.

```
<Implies ruletype="defeasiblerule">
   <oid>r1</oid>
   <head>
      <Atom neg="no">
         <Rel>acceptable</Rel>
         <Slot type="var">X</Slot>
         <Slot type="var">Y</Slot>
         <Slot type="var">Z</Slot>
         <Slot type="var">W</Slot>
      </Atom>
   </head>
   <body>
      <part type="atom">
         <Rel>name</Rel>
         <Slot type="var">X</Slot>
         <Slot type="var">X</Slot>
      </part>
          <part type="atom">
         <Rel>price</Rel>
         <Slot type="var">X</Slot>
         <Slot type="var">Y</Slot>
      </part>
          <part type="atom">
         <Rel>size</Rel>
         <Slot type="var">X</Slot>
         <Slot type="var">Z</Slot>
      </part>
          <part type="atom">
          <Rel>gardenSize</Rel>
          <Slot type="var">X</Slot>
          <Slot type="var">W</Slot>
      </part>
        </body>
</Implies>
```

Σχήμα 1 – Τμήμα ruleML αρχείου

```
defeasible(r1,acceptable(X,Y,Z,W),[name(X,X),price(X,Y),size(X,Z),gardenSize(X,W)]).
```

Σχήμα 2 – Κανόνας σε DR-Prolog σύνταξη

Η δεύτερη μέθοδος είναι η *"String getQuery(File ruleml)"*. Αυτή η μέθοδος βρίσκει το ερώτημα (Query) στο RuleML, το μετατρέπει σε DR-Prolog σύνταξη και επιστρέφει ένα αλφαριθμητικό που το αναπαριστά (σχήμα 3).

```
defeasibly(acceptable(X,Y,Z,W))
%(apartment, price, size, gardenSize)
```

Σχήμα 3 – Ερώτημα σε DR-Prolog σύνταξη

Η τρίτη μέθοδος είναι η *"String getQueryMode(File ruleml)"*. Αυτή η μέθοδος επιστρέφει τον τύπο του ερωτήματος. Ο τύπος μπορεί να είναι είτε *answer* είτε *proof*.

Η τελευταία μέθοδος είναι η *"String getQueryVars(File ruleml)"*. Η μέθοδος αυτή επιστρέφει τις μεταβλητές του ερωτήματος. Σκοπός της είναι η εύρεση των μεταβλητών του ερωτήματος για την κατασκευή της δομής της επιστρεφόμενης λίστας.

Η κλάση "Tagger.java" είναι μία βοηθητική κλάση για την ανάγνωση XML κώδικα. Περιέχει ένα σύνολο από συναρτήσεις που βρίσκουν ετικέτες και εξάγουν το περιεχόμενο τους.

Για παράδειγμα, η συνάρτηση *"String getFirstTagData(String theTag)"*, δέχεται ένα όνομα ετικέτας, πχ "Head" και επιστρέφει ένα αλφαριθμητικό με το περιεχόμενο αυτής της ετικέτας.

2.3 Metaprogram

Η υπάρχουσα έκδοση του *metaprogram* δεν υποστήριζε κάποιες λειτουργίες που ήταν απαραίτητες για την εκτέλεση των ερωτημάτων. Γι' αυτό το λόγο χρειάστηκε να αναβαθμιστεί ώστε να υποστηρίζει τις παρακάτω λειτουργίες:

- Σύγκριση αριθμών και υπολογισμός αριθμητικών παραστάσεων
- Υποστήριξη του κανόνα *naf*

2.3.1 Σύγκριση αριθμών και υπολογισμός αριθμητικών παραστάσεων

Για την σύγκριση αριθμών εισάγαμε τους παρακάτω 5 κανόνες:

```
definitely(X<Y):- compute([X,+,0],K), compute([Y,+,0],M), K @< M.
definitely(X>Y):- compute([X,+,0],K), compute([Y,+,0],M), K @> M.
definitely(X>=Y):- compute([X,+,0],K), compute([Y,+,0],M), K @>= M.
definitely(X=<Y):- compute([X,+,0],K), compute([Y,+,0],M), K @=< M.
definitely(X\==Y):- X \== Y.
```

Ο πρώτος κανόνας ελέγχει αν το Χ είναι μικρότερο του Υ. Ο δεύτερος αν ο Χ είναι μεγαλύτερος του Υ. Ο τρίτος αν ο Χ είναι μεγαλύτερος ή ίσος του Υ. Ο τέταρτος αν ο Χ είναι μικρότερος ή ίσος του Υ και ο τελευταίος αν ο Χ είναι διαφορετικός από το Υ. Καθώς μπαίνουμε στο κάθε κανόνα καλείται ο κανόνας compute. Η παρουσία του κανόνα *compute* είναι απαραίτητη και ο λόγος είναι ότι θέλουμε ταυτόχρονα να συγκρίνουμε και αριθμούς αλλά και αριθμητικές παραστάσεις. Ο λόγος για τον οποίο καλούμε τη *compute* με λίστα είναι ότι οι αριθμητικές παραστάσεις πρέπει να έχουν συγκεκριμένη μορφή. Για παράδειγμα μια απλή αριθμητική παράσταση είναι η [100,+,200] η οποία εκφράζει την πρόσθεση δυο αριθμών, του 100 με το 200.

Γενικώς, οι αριθμητικές παραστάσεις θα πρέπει να εκφράζονται σε μορφή λίστας και η κάθε λίστα εμφωλευμένη ή μη θα πρέπει να έχει τρία μέλη: αριστερό μέλος(αριθμός ή λίστα), δεξί μέλος(αριθμός ή λίστα) και μεσαίο μέλος(τελεστής πράξης). Για παράδειγμα αν θέλαμε να κάνουμε τη πράξη του πολλαπλασιασμού δυο αριθμών όπου ο δεύτερος αριθμός προκύπτει από τη πρόσθεση άλλων δυο (θέλουμε να κρατήσουμε τη προτεραιότητα των πράξεων), τότε προκύπτει η εξής έκφραση: [[10,+,20], *, 5]. Στη περίπτωση που έχουμε μόνο αριθμούς να συγκρίνουμε τότε καλείται πάλι η compute. Επειδή όμως θα αντιμετωπίζαμε πρόβλημα από τη στιγμή που η compute δέχεται μόνο λίστες, έτσι αναγκαστήκαμε να μετατρέψουμε τον αριθμό σε λίστα με έναν πολύ απλό τρόπο: [Χ,+,0], το οποίο θα επιστρέφει πάντα Χ. Με αυτό το τρόπο παρακάμπτουμε την compute.

Η υλοποίηση της *compute* είναι απλή και βασίζεται στη λογική της επίσκεψης κάθε κόμβου ενός δέντρου. Από τη στιγμή που μια αριθμητική παράσταση μπορεί να αποτελείται από πολλά επίπεδα εμφωλευμένων πράξεων – λιστών είναι σαν να δουλεύουμε πάνω σε κάποιο δέντρο. Ο υπολογισμός των πράξεων ξεκινάει από χαμηλά, από τα φύλλα του δέντρου και καταλήγει στη ρίζα όπου και επιστρέφεται το τελικό αποτέλεσμα. Παρακάτω παρατίθεται η υλοποίηση της:

```
compute([H1, H2, H3|_], K2) :- compute(H1, K), compute([K, H2, H3], K2).
compute([X, Y, Z|_], K) :- number(X), number(Z), Y == *, K is X * Z.
compute([X, Y, Z|_], K) :- number(X), number(Z), Y == /, K is X / Z.
compute([X, Y, Z|_], K) :- number(X), number(Z), Y == +, K is X + Z.
```

```
compute([X, Y, Z|_], K) :- number(X), number(Z), Y == -, K is X - Z.
compute([H1, H2, H3|_], K2) :- compute(H3, K), compute([H1, H2, K], K2).
```

2.3.2 Κανόνας naf

Η τελευταία αναβάθμιση που έγινε ήταν η εισαγωγή του κανόνα *naf*:

```
definitely(naf(X,Z,M)):- not(naf(X,Z,M)).
```

Ο παραπάνω κανόνας χρειάστηκε στο παράδειγμά μας για τις προτιμήσεις του πελάτη.

2.4 Μηχανή

Για τον έλεγχο των λειτουργιών που υλοποιήσαμε κατασκευάσαμε μία μηχανή η οποία αρχικά φορτώνει τις απαιτήσεις και προτιμήσεις του πελάτη (ruleML αρχεία), τη βάση γνώσης (facts.p) και τέλος το metaprogram.

Στη συνέχεια καλείται ο αναλυτής ο οποίος παράγει τους κανόνες, τους αποθηκεύει σε ξεχωριστό αρχείο και ταυτόχρονα τους φορτώνει στη μηχανή. Με συναρτήσεις του αναλυτή, εξάγουμε το ερώτημα, τον τύπο και τις μεταβλητές του καθώς και το τι αντιπροσωπεύει η κάθε μεταβλητή. Για παράδειγμα, η μεταβλητή X του σχήματος 3 αντιπροσωπεύει το όνομα του διαμερίσματος.

Τέλος, σαν αποτέλεσμα επιστρέφεται μία λίστα η οποία έχει μορφή ανάλογη των μεταβλητών του ερωτήματος. Για παράδειγμα, για το ερώτημα του σχήματος 3, η απάντηση είναι:

```
result=[a3,350,65,0,a5,350,55,15,a7,375,65,12]
```

Το παραπάνω αποτέλεσμα αποθηκεύεται σε ένα αρχείο με όνομα *result.txt*.

3. Παραδοτέα

1. carlo_1.ruleml
2. carlo_2.ruleml
3. dr-prolog.dtd
4. amb_metaprogram_add.P
5. Netbeans Project
    a. Engine.java
    b. RuleMLparser.java
    c. Tagger.java