Contacts Application Demo (13 mar 2015)

Alice and Bob

Computer laboratory, University of Cambridge, carlos.molina@cl.cam.ac.uk

Abstract. The moana layer was implemented by Yan.

1 Contact data is modelled as tuples

1.1 Data about Jon

```
let jon =
  Helper.to_tuple_lst
    "{(a,fn,Jon,contacts)
          (a,last,Crowcroft,contacts)
   (a,email,jon.crowcroft@cl.cam.ac.uk,contacts)
         (a,twitter,@tforcworc,contacts)
         (a,mobile, 617-000-0001,contacts)
   (a,title,Professor,contacts)
         (a,image,jon.jpg, contacts)
         (a,knows,c,contacts)
         (a,knows,d,contacts)
         (a,knows,e,contacts)
        }"
```

1.2 Every line is a tuple

```
let tup_jo1=    (* tuple of the fisrt line *)
 {
  subj=  Constant "a";          (* a is a unique id *)
  pred=  Constant "fn";         (* fn= first name   *)
  obj=   Constant "Jon";
  ctxt=  Constant "contacts";
  time_stp= None;
  sign= None;
 }
 (* In Eng: "a has_fn Jon" *)



let tup_jo2=  (* tuple of the second line *)
```

```
 {
  subj=  Constant "a";
  pred=  Constant "last";        (* last: last name *)
  obj=   Constant "Crowcroft";
  ctxt=  Constant "contacts";
  time_stp= None;
  sign= None;
 }
 (* In Eng: "a has_last Crowcroft" *)


let tup_jo3=
 {
 subj= Constant "a";
 pred= Constant "email";
 obj=  Constant "jon.crowcroft@cl.cam.ac.uk";
 ctxt= Constant "contacts";
 time_stp= None;
 sign= None;
 }
 (* In Eng: "a has_email Crowcroft" *)


let tup_jo5=
 {
  subj= Constant "a";
  pred= Constant  "mobile";
  obj=  Constant  "617-000-0001";
  ctxt= Constant  "contacts";
  time_stp= None;
  sign= None;
 }

...
let tup_jo10=  (* tuple of the last line *)
 {
  subj= Constant "a";
  pred= Constant "knows";
  obj=  Constant "e";
  ctxt= Constant "contacts";
  time_stp= None;
  sign= None;
 }
```

1.3 More examples of contacs data

```
let amir =
  Helper.to_tuple_lst
    "{(b,fn,Amir, contacts)
                (b,last,Chaudhry,contacts)
   (b,email,amir.chaudhry@cl.cam.ac.uk,contacts)
         (b,twitter,@amirmc,contacts)
         (b,image,amir.jpg, contacts)
   (b,title,Postdoc,contacts)
   (b,knows,a,contacts)
   (b,knows,c,contacts)
         (b,knows,d,contacts)
        }"

let anil =
  Helper.to_tuple_lst
    "{(c,fn,Anil, contacts)
          (c,last,Madhavapeddy, contacts)
   (c,email,anil@recoil.org,contacts)
         (c,image,anil.jpg, contacts)
         (c,twitter,@avsm,contacts)
         (c,email,anil.madhavapeddy@recoil.org,contacts)
   (c,title,Lecturer,contacts)}"

let carlos =
  Helper.to_tuple_lst
    "{(d,fn,Carlos, contacts)
         (d,last,Molina-jimenez, contacts)
         (d,image,carlos.jpg, contacts)
   (d,email,cm770@cam.ac.uk,contacts)
   (d,title,Postdoc,contacts)
         (d,twitter,@carlos,contacts)
         (d,knows,a,contacts)
         (d,knows,c,contacts)}"

let richard =
  Helper.to_tuple_lst
    "{(e,fn,Richard,contacts)
                (e,last,Mortier,contacts)
   (e,email,richard.mortier@nottingham.ac.uk,contacts)
         (e,image,mort.png, contacts)
         (e,twitter,@mort__,contacts)
   (e,title,Lecturer,contacts)
         (e,knows,c,contacts)
         (e,knows,d,contacts)
        }"
```

1.4 SPARQL queries can be placed against moana

SPARQL queries can be placed against moana to retrive information from irmim. The response is the set (possibly empty) of tuples that satisfy the query.

```
let qt1jon =
 {
  subj= Variable "a";        (* a: unique id for Jon *)
  pred= Constant "knows";
  obj=  Constant "?y";
  ctxt= Constant "contacts";
  time_stp= None;
  sign= None;
 }
(* In Eng: retrieve all the people that Jon knows *)

let qt1jon =
 {
  subj= Variable "a";        (* a: unique id used for Jon *)
  pred= Constant "knows";
  obj=  Constant "?y";
  ctxt= Constant "contacts";
  time_stp= None;
  sign= None;
 }
(* In Eng: retrieve all the people that Jon knows *)


let qt1amir =
 {
  subj= Variable "b";       (* b: unique id used for amir *)
  pred= Constant "knows";
  obj=  Constant "?y";
  ctxt= Constant "contacts";
  time_stp= None;
  sign= None;
 }
(* In Eng: retrieve all the people that amir knows *)
```

2 Policies can be used to constrain visibility of contacts data

```
let q1_policy_email_and_fn =
  "MAP {
         b,  canView, ?x,     policies
         ?x, knows,   ?o,     contacts
         ?o, email,   ?email, contacts
         ?o, fn,      ?n,     contacts
        }"
(* In Eng: Bring all tuples that "b" can view.
           e,g., if "b" can view "a" then, bring
           everyone whom "a" knows, their "email" and
           "fn" name--but nothing else.
           A query to retrive the "last" name of a person will
           produce no tuples.

let q1_policy_email_fn_and_last =
  "MAP {
         b,  canView, ?x,     policies
         ?x, knows,   ?o,     contacts
         ?o, email,   ?email, contacts
         ?o, fn,      ?n,     contacts
         ?o, last,    ?last,  contacts
        }"
(* In Eng: Bring all tuples that "b" can view.
           e,g., if "b" can view "a" then, bring
           everyone whom "a" knows, their "email",
           "fn" and "last" name--but nothing else.
           A query to retrive the "last" name of a person will
           produce some tuple.
```