Soma-referentials

Under construction...

Aim

Provide a lightweight SQL database-based representation of a referentials and transformations system, with several requirements:

- Easy queries from C++ and Python languages.
- Independent from our other libraries so it can be used from several contexts (aims/anatomist, ptk, axon, ...)
- API able to provide paths from one referential to another, providing a chain of transforms.
- Transformations contents do not need to be interpreted at this lower level. Upper level layers should get access to the transformations data and interpret them at this higher level.
- Several referentials databases may be used at once and combined, at least in a read-only mode

Technical characteristics

Not to be taken as definitive implementation choices, but probably:

- C++ implementation.
- Python binding on top of the C++ implementation.
- Based on SQLite, but we may add network-enbled SQL databases for remote access.
  Do we allow using QtSql, to easily switch between several database systems ?

SQL tables structure

- Referentials: list of referentials identified by an UUID
  Additional attributes: (a) stored in the same table columns, or (b) as separate entries in another table ?
    - **(a)**
      <table><tr><td>*uuid*</td> <td>*name*</td> <td>*axes_orientation*</td> <td>*direct*</td> </tr><tr><td>a2a820ac-a686-461e-bcf8-856400740a6c</td><td>Talairach-AC/PC-Anatomist</td><td>right-left, anterior-posterior, top-bottom</td><td>0</td></tr> <tr><td>803552a6-ac4d-491d-99f5-b938392b674b</td><td>Talairach-MNI template-SPM</td><td>left-right, posterior-anterior, bottom-top</td> <td>1</td></tr></table>
    - **(b)**
      <table><tr><td>*uuid*</td> <td>*name*</td> <td>*value*</td></tr><tr><td>a2a820ac-a686-461e-bcf8-856400740a6c</td><td>name</td> <td>Talairach-AC/PC-Anatomist</td></tr><tr><td>a2a820ac-a686-461e-bcf8-856400740a6c</td><td>axes_orientation</td> <td>right-left, anterior-posterior, top-bottom</td></tr> <tr><td>a2a820ac-a686-461e-bcf8-856400740a6c</td> <td>direct</td> <td>0</td></tr> <tr><td>803552a6-ac4d-491d-99f5-b938392b674b</td> <td>name</td> <td>Talairach-MNI template-SPM</td></tr> <tr><td>803552a6-ac4d-491d-99f5-b938392b674b</td> <td>axes_orientation</td> <td>left-right, posterior-anterior, bottom-top</td> </tr><tr><td>803552a6-ac4d-491d-99f5-b938392b674b</td> <td>direct</td> <td>1</td></tr></table>

  Schema (b) is much more flexible, but forces to store every attribute as strings, makes bigger databases, bigger requests, and makes attributes filtering/queries on values less efficient.

- Transformations:
    - a main transformations table containing an UUID, two referentials UUIDs, and a transformation type
      <table><tr><td>*uuid*</td> <td>*source*</td> <td>*dest*</td> <td>*type*</td> </tr><tr><td>ffaa15d1-a1a7-ab7d-3f3e-56f6124add0d</td><td>a2a820ac-a686-461e-bcf8-856400740a6c</td><td>f3848046-b581-cae4-ecb9-d80ada0278d5</td><td>affine</td></tr></table>
    - For each transformation type, a table with transformations contents
      <table><tr><td>*uuid*</td><td>*matrix*</td></tr><tr><td>ffaa15d1-a1a7-ab7d-3f3e-56f6124add0d</td><td>0.97, -0.0055, -0.017, 78.1, 0.011, 0.96, 0.077, 75.8, 0.01, -0.071, 0.95, 87.9, 0, 0, 0, 1</td></tr></table>

  Here again, we could use something like schema (b) to store arbitrary attributes, but maybe don't need to since each transformation type will have its own specific table with specific columns.

At the beginning, only affine transformations will be used.

Main API

```
ReferentialsDatabase db;
db.open( "/home/gaston/refdb.sqlite" );
TransformationPathsListIterator itc
  = db.getTransformationPaths( "a2a820ac-a686-461e-bcf8-856400740a6c",
                               "803552a6-ac4d-491d-99f5-b938392b674b" );
while( itc )
{
  cout << "path:" << endl;
  TransformationPathIterator it;
  for( it=itc->begin(); it!=itc->end(); ++it )
  {
    cout << itc->uuid() << ", ";
  }
  ++itc;
}
```

- Iteration system to navigate through possible paths, along a path.
- Define a class for referentials: Referential.
- Do we define a transformation class, knowing that Aims and Ptk also define theirs ?
  If not, how do we provide information associated with transformations (UUID, source/destination, matrix or other) ?