chem-bla-ics

Adding a new dictionary to Oscar

Egon Willighagen

Published November 29, 2010

Citation
Willighagen, E. (n.d.). In *chem-bla-ics*. chem-bla-ics. https://doi.org/10.59350/v90k2-5a907

Keywords
Oscar, Java

Copyright
Copyright © Egon Willighagen 2010. Distributed under the terms of the Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

chem-bla-ics

Say, you have your own dictionary of chemical compounds. For example, like your company's list of yet-unpublished internal research codes. Still, you want to index your local listserv to make it easier for your employees to search for particular chemistry you are working on and perhaps related to something done at other company sites. This is what Oscar is for.

But, it will need to understand things like UK-92,480. This is made possible with the Oscar4 refactorings we are currently working on. You only need to register a dedicated dictionary . Oscar4 has a default dictionary which corresponds to the dictionary used by Oscar3, and a dictionary based on ChEBI (an old version) (see this folder in the source code repository).

Adding a new dictionary is very straightforward: you just implement the IChemNameDict interface. This is, for example, what the OPSIN dictionary looks like:

```
public class OpsinDictionary
implements IChemNameDict, IInChIProvider {

  private URI uri;

  public OpsinDictionary() throws URISyntaxException {
    this.uri = new URI(
      "http://wwmm.cam.ac.uk/oscar/dictionariy/opsin/"
    );
  }

  // the URI is somewhat like a namespace
  public URI getURI() {
    return uri;
  }

  // there are no stop words defined in this
  // dictionary
  public boolean hasStopWord(String queryWord) {
    return false;
  }

  // see hasStopWord()
  public Set getStopWords() {
    return Collections.emptySet();
  }

  // it has the name in the dictionary if the name
  // can be converted into an InChI
  public boolean hasName(String queryName) {
    return getInChI(queryName).size() != 0;
```

```
  }

  // this dictionary can return InChIs for names
  // so, it implements the IInChIProvider interface
  public Set getInChI(String queryName) {
    try {
      NameToStructure nameToStructure =
        NameToStructure.getInstance();
      OpsinResult result = nameToStructure
        .parseChemicalName(
          queryName, false
        );
      if (result.getStatus()
          == OPSIN_RESULT_STATUS.SUCCESS) {
        Set inchis = new HashSet();
        String inchi = NameToInchi
          .convertResultToInChI(
            result, false
          );
        inchis.add(inchi);
        return inchis;
      }
    } catch (NameToStructureException e) {
      e.printStackTrace();
    }
    return Collections.emptySet();
  }

  public String getInChIforShortestSMILES(
    String queryName)
  {
    Set inchis = getInChI(queryName);
    if (inchis.size() == 0) return null;
    return inchis.iterator().next();
  }

  // since names are converted on the fly, we do
  // not enumerate them
  public Set getNames(String inchi) {
    return Collections.emptySet();
  }
  public Set getNames() {
    return Collections.emptySet();
  }
```

```
  public Set getOrphanNames() {
    return Collections.emptySet();
  }
  public Set getChemRecords() {
    return Collections.emptySet();
  }
  public boolean hasOntologyIdentifier(
    String identifier)
  {
    // this ontology does not use ontology
    // identifiers
    return false;
  }
}
```

Now, you can implement the interface in various ways. You can even have the implementation hook into a SQL database with JDBC, or use something else fancy. The dictionary will be used at various steps of the Oscar4 text analysis workflow.

Mind you, the refactoring is not over yet, and the details may change here and there.

Your comments are most welcome!