Metabolite annotation by Liquid-Chromatography-Mass Spectrometry (LC-MS) Practice

1. Overview

This practice emulates the core functionality of the metabolite annotation tools, which integrates liquid chomatography and mass spectrometry data with metabolite databases to identify potential compounds and adducts based on the data acquired by analytical instrumentation. Concretely it will be based on the Retention Time (RT) acquired by Liquid Chromatography (LC) and the experimental mass-to-charge ratio (m/z) obtained by Mass Spectrometry (MS) means. The template code will be a J2EE project using:

- **Drools**: For rule-based reasoning to match m/z values to metabolite candidates and subsequently use the elution order based on the RT to confirm or reject annotations.

- **J2EE**: For application structure.

- **Maven**: For dependency management and build automation.

- **JUnit + Mockito**: For unit testing.

2. Architecture

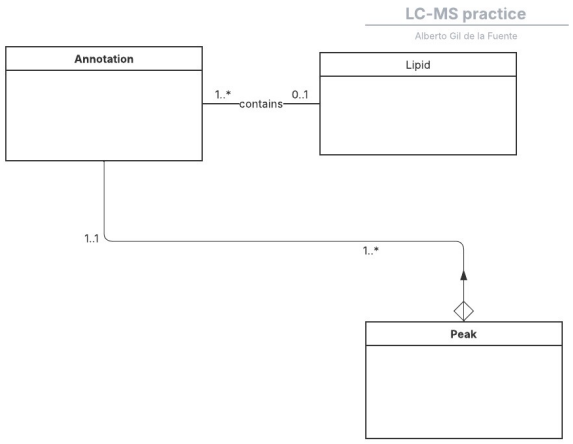
mass-spec-ceu-practice/

├── src/

```
│   ├── main/
│   │   ├── java/
│   │   │
│   │   └── resources/
│   │       ├── rules/
│   │       │   └── metabolite-matching.drl
│   └── test/
│       └── java
│           └── lipid/AdductDetectionTest.java
│           └── lipid/ElutionOrderTest.java
└── pom.xml
```

3. Sample Model

The proposed UML is simplified to focus



Where the class Annotation is the data type which will be used as a fact. It contains the next attributes

+---------------------------+

| Annotation |

+---------------------------+

| - lipid: Lipid |

| - mz: double |

| - intensity: double |

| - rtMin: double |

| - adduct: String |

| - groupedSignals: Set<Peak>|

| - score: int |

+---------------------------+

| + Annotation(...) |

| + getLipid(): Lipid |

| + getMz(): double |

| + getRtMin(): double |

| + getAdduct(): String |

| + setAdduct(String): void |

| + getIntensity(): double |

| + getGroupedSignals(): Set<Peak> |

| + getScore(): int |

| + setScore(int): void |

| + addScore(int): void |

| + equals(Object): boolean |

| + hashCode(): int |

| + toString(): String |

+---------------------------+

Check the project template and go to the Junit test classes. There are 2 main goals:

1- Adduct detection. Based on the masses difference between the grouped mzs obtained in a corresponding RT. The mass differences will be used to detect the adduct based on the mass difference within a certain tolerance specified in mDa or in ppm. The ppm

```
@Test
public void shouldDetectAdductBasedOnMzDifference() {

    // Given two peaks with ~21.98 Da difference (e.g., [M+H]+ and [M+Na]+)
```

```
    Peak mH = new Peak(700.500, 100000.0); // [M+H]+
    Peak mNa = new Peak(722.482, 80000.0);  // [M+Na]+
    Lipid lipid = new Lipid(1, "PC 34:1", "C42H82NO8P", "PC", 34, 1);

    double annotationMZ = 700.49999d;
    double annotationIntensity = 80000.0;
    double annotationRT = 6.5d;
    Annotation annotation = new Annotation(lipid, annotationMZ, annotationIntensity,
annotationRT, Set.of(mH, mNa));


    // Then we should call the algorithmic/knowledge system rules fired to detect the
adduct and Set it!
    //
    assertNotNull("[M+H]+ should be detected", annotation.getAdduct());
    assertEquals( "Adduct inferred from lowest mz in group","[M+H]+",
annotation.getAdduct());
}


@Test
public void shouldDetectLossOfWaterAdduct() {
    Peak mh = new Peak(700.500, 90000.0);        // [M+H]+
    Peak mhH2O = new Peak(682.4894, 70000.0);     // [M+H–H₂O]+, ~18.0106 Da
less

    Lipid lipid = new Lipid(1, "PE 36:2", "C41H78NO8P", "PE", 36, 2);
    Annotation annotation = new Annotation(lipid, mh.getMz(), mh.getIntensity(),
7.5d, Set.of(mh, mhH2O));



    assertNotNull("[M+H]+ should be detected", annotation.getAdduct());

    assertEquals( "Adduct inferred from lowest mz in group","[M+H]+",
annotation.getAdduct());
}

@Test
public void shouldDetectDoublyChargedAdduct() {
    // Assume real M = (700.500 - 1.0073) = 699.4927
    // So [M+2H]2+ = (M + 2.0146) / 2 = 350.7536
    Peak singlyCharged = new Peak(700.500, 100000.0);  // [M+H]+
    Peak doublyCharged = new Peak(350.754, 85000.0);   // [M+2H]2+

    Lipid lipid = new Lipid(3, "TG 54:3", "C57H104O6", "TG", 54, 3);
    Annotation annotation = new Annotation(lipid, singlyCharged.getMz(),
singlyCharged.getIntensity(), 10d, Set.of(singlyCharged, doublyCharged));
```

```
    assertNotNull("[M+H]+ should be detected", annotation.getAdduct());

    assertEquals( "Adduct inferred from lowest mz in group","[M+H]+",
annotation.getAdduct());
}
```

Elution order of compounds:

```
@Test
public void score1BasedOnRT() {
    // Assume lipids already annotated
    LOG.info("Creating RuleUnit");
    LipidScoreUnit lipidScoreUnit = new LipidScoreUnit();

    RuleUnitInstance<LipidScoreUnit> instance =
RuleUnitProvider.get().createRuleUnitInstance(lipidScoreUnit);


    // TODO CHECK THE Monoisotopic MASSES OF THE COMPOUNDS IN
https://chemcalc.org/
    Lipid lipid1 = new Lipid(1, "TG 54:3", "C57H104O6", "TG", 54, 3); // MZ of [M+H]+
= 885.79057
    Lipid lipid2 = new Lipid(2, "TG 52:3", "C55H100O6", "TG", 52, 3); // MZ of [M+H]+
= 857.75927
    Lipid lipid3 = new Lipid(3, "TG 56:3", "C59H108O6", "TG", 56, 3); // MZ of [M+H]+
= 913.82187
    Annotation annotation1 = new Annotation(lipid1, 885.79056, 10E6, 10d);
    Annotation annotation2 = new Annotation(lipid2, 857.7593, 10E7, 9d);
    Annotation annotation3 = new Annotation(lipid3, 913.822, 10E5, 11d);


    LOG.info("Insert data");

    try {
        lipidScoreUnit.getAnnotations().add(annotation1);
        lipidScoreUnit.getAnnotations().add(annotation2);
        lipidScoreUnit.getAnnotations().add(annotation3);

        LOG.info("Run query. Rules are also fired");
        instance.fire();

        // Here the logic that we expect. In this case we expect the full 3 annotations to
have a positive score of 1

        assertEquals(1.0, annotation1.getNormalizedScore(), 0.01);
```

```
        assertEquals(1.0, annotation2.getNormalizedScore(), 0.01);
        assertEquals(1.0, annotation3.getNormalizedScore(), 0.01);

    }
    finally {
        instance.close();
    }
}
```

// TODO NEW RULES WILL BE ADDED HERE FOR THE RT-ELUTION ORDER

4. Drools Rule (metabolite-matching.drl)

This is a dummy example of a rule. This rule is the basis of the RT rules, but the rules will be defined after so you can translate them into your project.

```
// TODO Include here rules and queries to fulfill the practice requirements

// This is one example of rules that only prints the factorial combination of all pairs eliminating the A-A combination.
rule "Score 1 for lipid pair with increasing RT and carbon count"
when
    $a1 : /annotations [$rt1 : rtMin, $carbonCount1 : lipid.getCarbonCount(), $doubleBondCount : lipid.getDoubleBondsCount()]
    $a2 : /annotations [this!= $a1, lipid.getDoubleBondsCount() == $doubleBondCount, rtMin > $rt1, lipid.getCarbonCount() > $carbonCount1]
then
    // in this case, the only change is the addition of the score, but the fact does not change so we do not break the principle of refractoriness
    $a1.addScore(1);
    $a2.addScore(1);
    // !! TODO ONLY FOR DEBUGGING
    System.out.println("Scored 1 for annotations: " + $a1 + " and " + $a2);
end
```

5. Rule Engine Service

The student can choose among the classical drools KieSessions or the ruleUnitData. The example of the practice currently is performed with RuleUnitData to keep the consistency with the previous examples.

```
package droolsService;


import Lipid;

import org.kie.api.KieServices;

import org.kie.api.runtime.KieContainer;

import org.kie.api.runtime.KieSession;

import java.util.List;


public class RuleEngineService {

  private final KieContainer kieContainer;


  public RuleEngineService() {

    this.kieContainer = KieServices.Factory.get().getKieClasspathContainer();

  }


  public void applyRules(List<Lipid> lipids) {

    KieSession kieSession = kieContainer.newKieSession();

    for (Lipid l : lipids) {

      kieSession.insert(l);

    }

    kieSession.fireAllRules();

    kieSession.dispose();
```

```
    }

}
```

RuleUnitData example:

```
LipidScoreUnit lipidScoreUnit = new LipidScoreUnit();

RuleUnitInstance<LipidScoreUnit> instance =
RuleUnitProvider.get().createRuleUnitInstance(lipidScoreUnit);




Lipid lipid1 = new Lipid(1, "TG 54:3", "C57H104O6", "TG", 54, 3); // MZ of [M+H]+ =
885.79057
Lipid lipid2 = new Lipid(2, "TG 52:3", "C55H100O6", "TG", 52, 3); // MZ of [M+H]+ =
857.75927
Lipid lipid3 = new Lipid(3, "TG 56:3", "C59H108O6", "TG", 56, 3); // MZ of [M+H]+ =
913.82187
Annotation annotation1 = new Annotation(lipid1, 885.79056, 10E6, 10d);
Annotation annotation2 = new Annotation(lipid2, 857.7593, 10E7, 9d);
Annotation annotation3 = new Annotation(lipid3, 913.822, 10E5, 11d);


LOG.info("Insert data");

try {
    lipidScoreUnit.getAnnotations().add(annotation1);
    lipidScoreUnit.getAnnotations().add(annotation2);
    lipidScoreUnit.getAnnotations().add(annotation3);

    LOG.info("Run query. Rules are also fired");
    instance.fire();



}
finally {
```

```
    instance.close();
}
```

7. Maven Dependencies (pom.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>dss</groupId>
  <artifactId>template</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>kjar</packaging>

  <name>template</name>

  <properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.release>21</maven.compiler.release>
    <maven-compiler-version>3.14.0</maven-compiler-version>
    <drools-version>10.0.0</drools-version>
    <junit-version>4.13.2</junit-version>
    <logback-version>1.5.17</logback-version>
    <slf4j-version>2.0.17</slf4j-version>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.drools</groupId>
        <artifactId>drools-bom</artifactId>
        <type>pom</type>
        <version>${drools-version}</version>
      </dependency>
    </dependencies>
  </dependencyManagement>

  <dependencies>

    <dependency>
```

```
    <groupId>org.drools</groupId>
    <artifactId>drools-ruleunits-engine</artifactId>
    <version>${drools-version}</version>
  </dependency>

  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit-version}</version>
    <scope>test</scope>
  </dependency>


  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${slf4j-version}</version>
  </dependency>

  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${logback-version}</version>
  </dependency>



  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-wiring-dynamic</artifactId>
    <version>${drools-version}</version>
  </dependency>

</dependencies>

<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${maven-compiler-version}</version>
      <configuration>
        <release>${maven.compiler.release}</release>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.kie</groupId>
      <artifactId>kie-maven-plugin</artifactId>
      <version>${drools-version}</version>
```

```
      <extensions>true</extensions>
    </plugin>
   </plugins>

  </build>
</project>
```

8. Unit Tests

Find them in the template project. These unit tests contain the requirements of the practice. There are two type of knowledge here:

1- Adduct detection based on the mass differences of the grouped peaks

9. Running the Application

Use the following command to package and run the application if maven is in your PATH. Otherwise, use your reference framework. If IntelliJ IDEA, use the Lifecycle maven plugin to run it.

mvn clean test

10. Conclusion

This practice illustrates a simplified lipid annotation simulation using Drools and J2EE, demonstrating how rule engines can be integrated into scientific workflows. The project supports extensibility via new rules and metabolites and prepares the foundation for more complex reasoning and API integration.