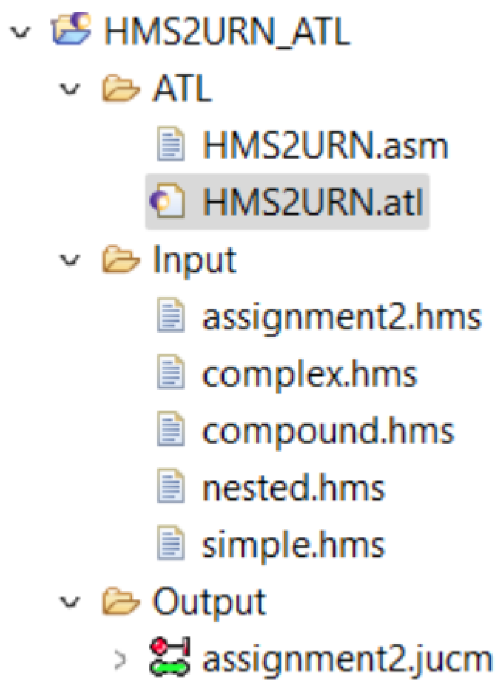
ECSE 439 - Assignment 2
March 9, 2023

Group 10
Annabelle Dion 260928845
Amani Jammoul 260381641
Paul Teng 260862906

Assignment 2 – Model Transformation to URN

File Locations

Within the "HMS2URN_ATL" folder, you will find 3 subfolders: ALT, Input, and Output. Each contains the files corresponding to their names. The following is an image of the folder structure:



```
HMS2URN_ATL
  ATL
    HMS2URN.asm
    HMS2URN.atl
  Input
    assignment2.hms
    complex.hms
    compound.hms
    nested.hms
    simple.hms
  Output
    assignment2.jucm
```

ECSE 439 - Assignment 2
March 9, 2023

Group 10
Annabelle Dion 260928845
Amani Jammoul 260381641
Paul Teng 260862906

Discussion

When implementing the ATL transformation between the Hospital Management Language in the Hospital Management System (HMS) in UML to User Requirements Notation (URN), there was a combination of aspects that went well and others that proved to be a struggle.

First, understanding the ATL syntax and rule patterns which were presented in class was crucial. We also need to understand the three main transformation execution phases (module initialization, matching, and target initialization). After this, it was easy to lay out the framework for the rules we needed to apply to map the top-level entity and set up the URN diagram.

It was useful to have the tutorial code, as it acted as a reference whenever we were unsure about something and a template to make sure we were performing the implementation properly. For example, when we were specifying the URNdefinition and wanted to make a separate diagram and component for each hospital management rule, we wrote “specDiagrams <- h.rules'' and “components <- h.rules”. However, we were getting odd errors which we didn’t understand. In the tutorial, we saw the use of the select() method, which prompted us to find the similar collect() method — which we were able to find — that was able to resolve our issue.

This brings us to a separate aspect that did not go so well, which was trying to decipher the perplexing and ambiguous error messages. Many problems that we ran into had to be solved the hard way, involving a lot of head scratching, debugging, and digging through documentation because the error messages we encountered were not helpful in actually indicating what the problem was. The good part was that the official ATL documentation was easy to read and navigate through, which made the implementation and debugging easier.

A very challenging component of this task was figuring out how to properly position the nodes and components in the diagram. The coding of the positioning methods/rules was quite time consuming. It was also a challenge to understand that we needed to perform multiple tree traversals in order to create, link, and position all the nodes.

More coverage of OCL in class would have made the implementation easier. In addition, although the tutorial was nice to have, it was also very simple compared to the level of complexity of this assignment, and since this is our first time working with OCL, having more exposure to the more complex topics either in class or in the tutorials would have made this easier. As mentioned before, if the error messages were clearer, the implementation would surely have gone smoother as well. We also found that there were some scenarios when error messages at editing-time were *not given at all* even though they could (and maybe should) be. For example, given a called rule with some parameters, it is actually possible to call that method without specifying all the parameters and no error message is shown in Eclipse. However, the behavior is not as expected and the method does not always execute properly. So in this case, it would be more useful to indicate that one or more parameters are missing.