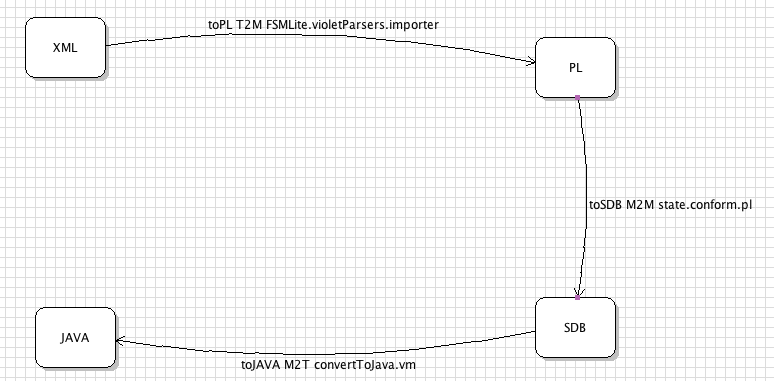
Programming Assignment-7

Akanksha Bansal ([akankshabansal90@gmail.com](mailto:akankshabansal90@gmail.com?subject=Assignment%20%20Submission))

Gaurav Nanda ([gaurav324@gmail.com](mailto:gaurav324@gmail.com?subject=Assignment%20%20Submission))

Prateek Agarwal ([prat0318@gmail.com](mailto:prat0318@gmail.com?subject=Assignment%20%20Submission))

This assignment asks us to write a tool Catlite that is able to bootstrap FSMList created in previous assignment.



This Category diagram drawn in violet to represents the various states of FSMLite in form of an xml with the details of the nodes and the transitions is parsed and generates the code base for FSMLite. We parse the xml file generated for categories and extract the nodes and the transitions from it. Then write a .pl files out of the data that has been extracted to create tables of the Category Nodes and transitions possible between the various domains. With the nodes being defined in the first table and transitions being defined in the next.

The prolog file generated after reading the File represents the FSMLite in the given example. Then CATList runs various constraints checks on the prolog tables created to make sure that the xml represents a valid MDELite tool. It has various checks to ensure that the various transitions from nodes are consistent with its type. That is from the same Node, a transition of type MTM and T2M are not possible.

The presence of these constraints ensures that CATlite is able to parse the category diagram for any MDELite type tool.

After this Model-to-Model transition CATLite generates classes for the Tool using Model-to-Text transition.

Classes are created by the tool:

*Convert.java*

*Violet.java*

All the transitions that are possible based on the xml provided are generated in Convert.java, and the user can make call to these functions to be able to generate whatever code he wants.

*Jar created for this execution is named: CATLite2.3.jar*

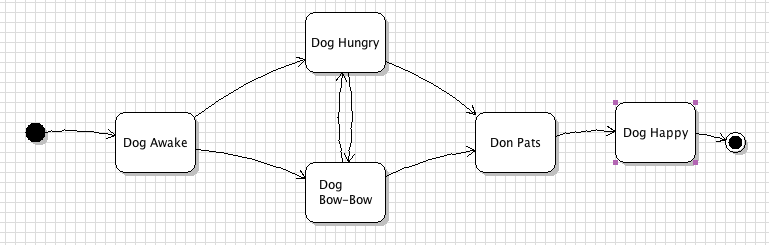
**Implementation details**

We show the effectiveness of CATLite by generating  FSMLite code using the diagram shown above. Once the files have been generated rom CatLite, we plug this code piece with the code written for programming assignment 5 and 6 we get the prolog files, Importer class, and Model-to-text transition (vm file) needed for FSMLiet to be able to perform function completely.

In the final step we generate the main class for FSMLiet from the run.sh

We have then compiled the code generated for FSMLite to show that the java files being generated work correctly.

Running FSMLite for the xml’s defined for below state machine:



Running Instructions:

*$ sh run.script*

This script will generate the code for the above sections and then make minor changes to the code generated and generate a class through which FSMLite various calls are made. This class is given the path of the xml as an argument. The output indicates if the tests have run successfully or not on the classes generated from the bootstrapped FSMLite.