CS763 Project: Rules with MetaEdit+

By Willer Travassos

260232775

Quick Review

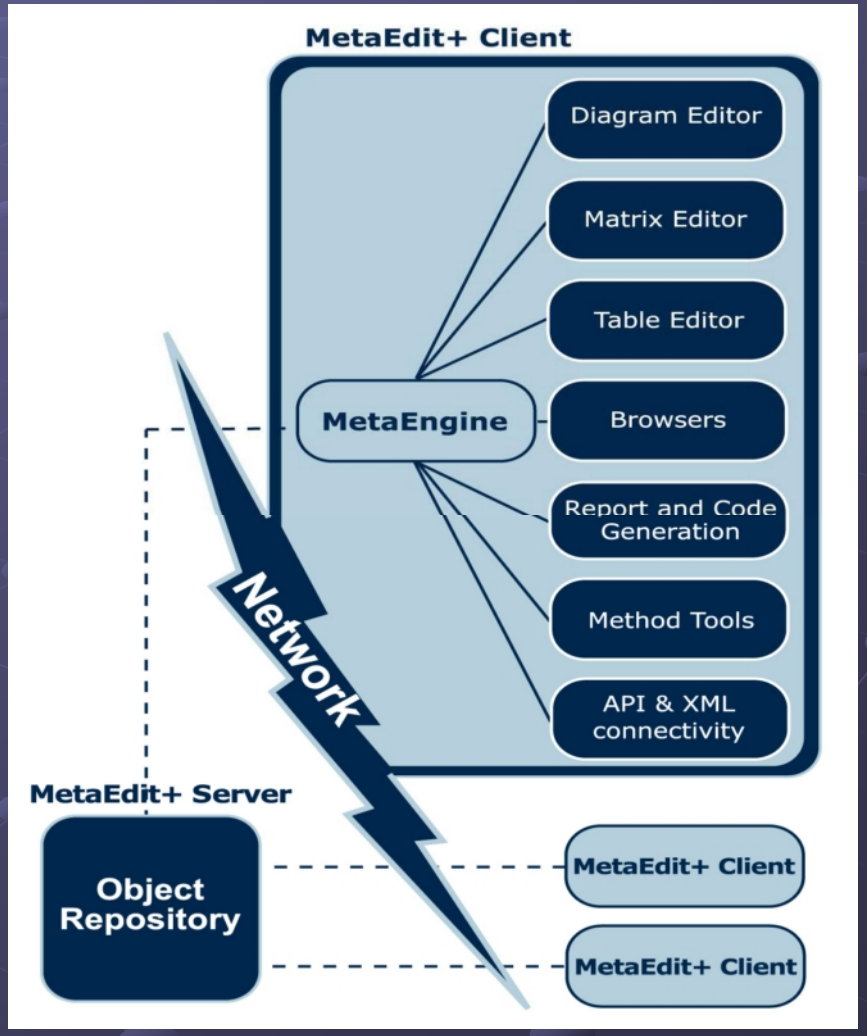
- Main Reasons to develop MetaEdit+:
  - To speed up software development, while minimizing problems common to it. Ex: mediocre production due to time constraints.
  - To solve problems that were common and apparent with the DSM solutions at the time.
  - To provide services that were inexistent in these software solutions. Ex: Inclusion of a multi-tool environment, and multi-visual representation.

Underlying Ideas

- MetaEdit+ vows to provide:
  - Model Repository functions for consistency check, and integration
  - Multi-user access to created models and meta-models
  - Varied modeling methods (graphical, textual, tabular)
  - A true meta modeled environment created with a real meta-meta-model language
  - Varied data reporting techniques, and code generation methods

The software's Architecture

- MetaEdit+ architecture consists of:
  - An Object Repository: holds data related to models and meta-models, user, and data access info
  - A MetaEngine: processes and handles all user requests, and data sent from the O.R.
  - Various independent client-side tools: provides an interface for model access
- Architecture is centered around the MetaEngine to update the software in a simple and fast manner



MetaEdit+'s Meta-meta-language

- It is called GOPPRR, and its name is an amalgamation of the types of objects that can be created with MetaEdit+
- G is for graph, and it is the model that created based on a project (meta-model). Here we also determine how objects, ports, roles and relationships are connected to one another
- O is for object, and it represents elements of a meta-model. Ex: places and transitions on a Petri Net
- P is for the properties that a graph, object, or relationship can have
- P also stands for ports, and it is how we determine types of connections
- R is for relationship between objects
- And the last R is for roles which determine the behavior of an object in a model

MetaEdit+'s Meta-meta-language

- Everything that is created with MetaEdit+ is stored on the O.R. using GOPPRR
- The GOPPRR is designed to allow creation and re-use of objects and their behaviors in a fast manner
- Also the GOPPRR provides means to maintain consistency of objects, roles, and etc among different graphs (models).

Rules with MetaEdit+

- MetaEdit+ does not have an engine that uses or generates Graph Grammar like ATOM3
- Therefore, there is not a rule execution scheme like the one we used for assignment 5
- MetaEdit+, though, does provide an API to access graphs contained in a model

Project

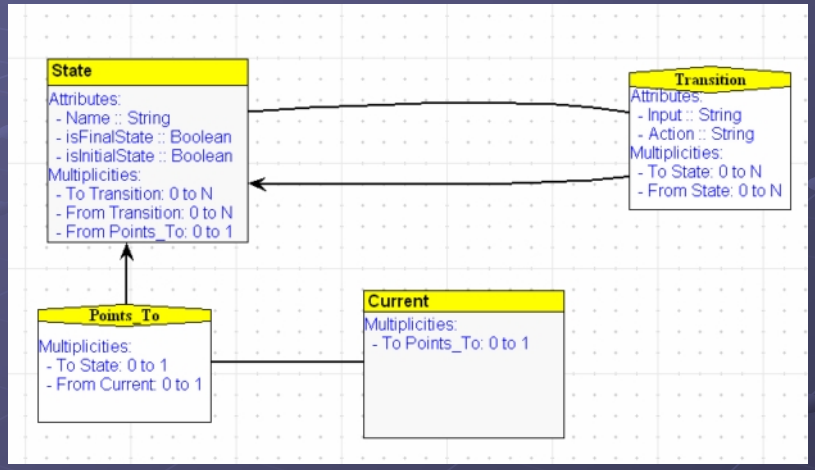
- This project is about adapting Graph Grammars to the API calls in MetaEdit+
- The first part of the project would be carried out using ATOM3, so that I could understand the underlying work of FSM simulation
- The second part concentrated on “translating” the work done with ATOM3 to MetaEdit+
- To maintain simplicity, the rules that were going to be adapted were based on the simulation of Finite State Automata

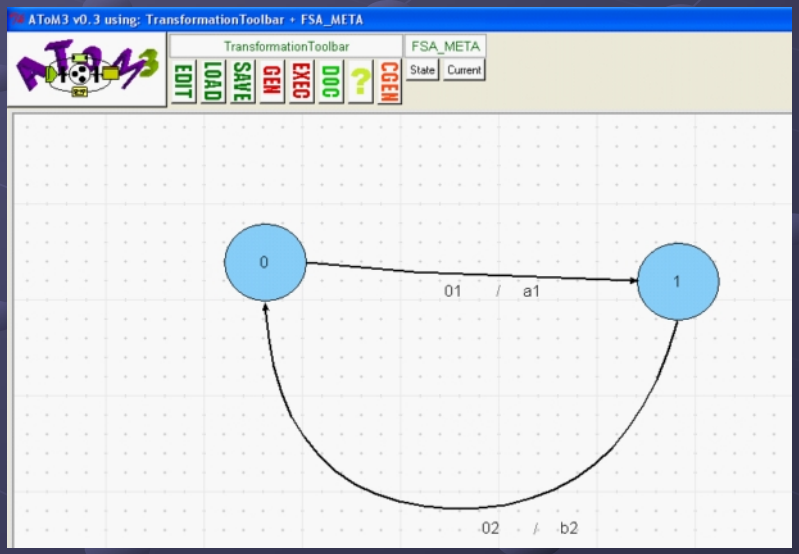
Project steps

- The projects was carried as follows:
  1. Meta-model Finite State Machines in ATOM3
  2. Model simulation rules for FSM simulation in ATOM3
  3. Simulate FSMs and inspect their how rules where executed
  4. Meta-Model FSMs in MetaEdit+
  5. Study the MetaEdit+'s API and learn how to access and manipulate graphs
  6. Apply simulation rules to the FSM in MetaEdit+

Working with ATOM3

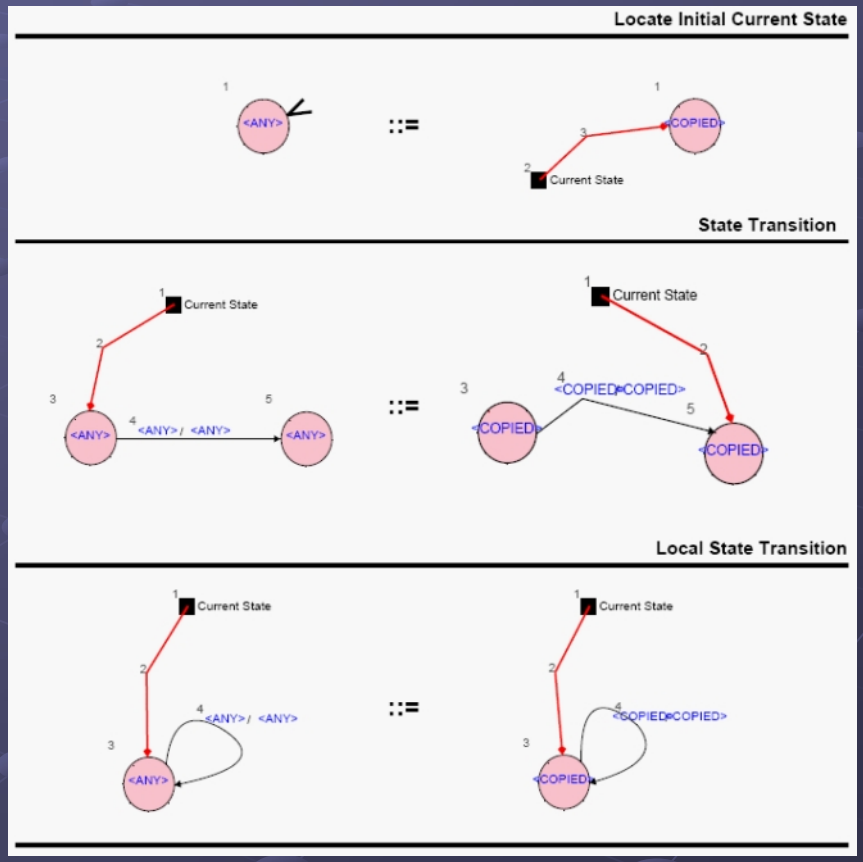
- The FSM meta-model was implemented close to FSA in ATOM3
- Differently from the FSA implemented in ATOM3, my FSM used Class Diagrams
- The Current class is just an empty class used device to help in the FSM simulation





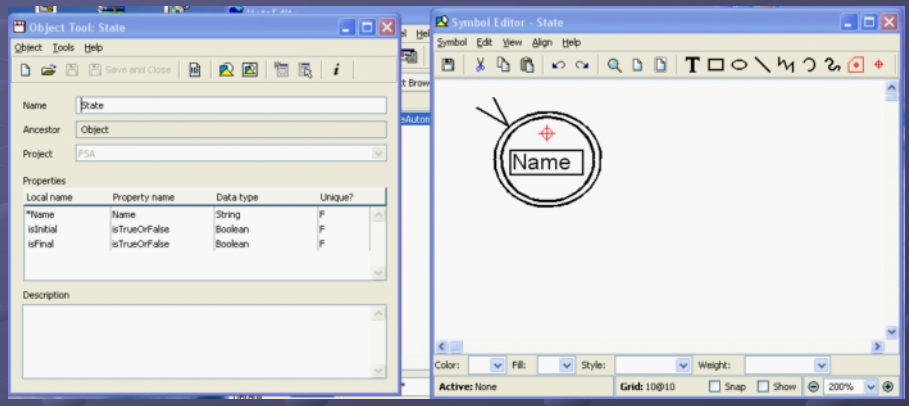
Working with ATOM3

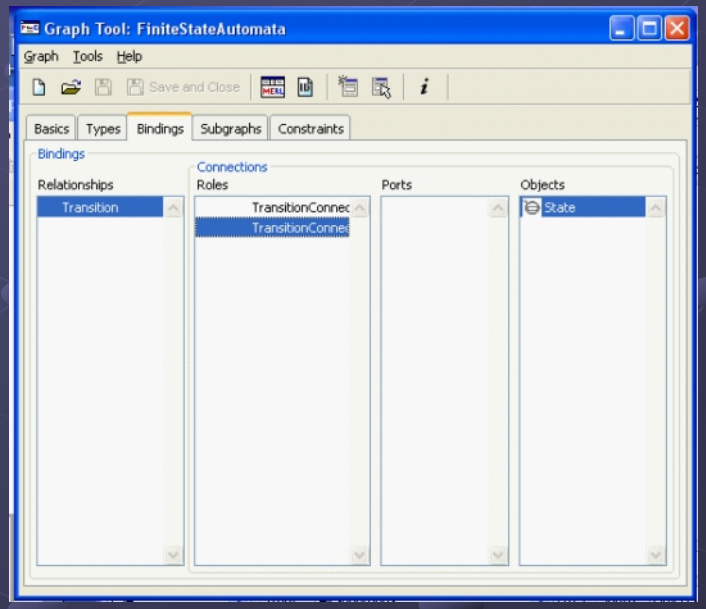
- With the meta-model implemented, the next step was to implement the rules
- Once more, the design of the rules were based on the paper by Prof. Hans, Meta-Models are models too
- Differently from the paper though, only two rules were implemented
- As it was seen later on my work, the rules for state transition and local state transition could be condensed into one



Working with MetaEdit+

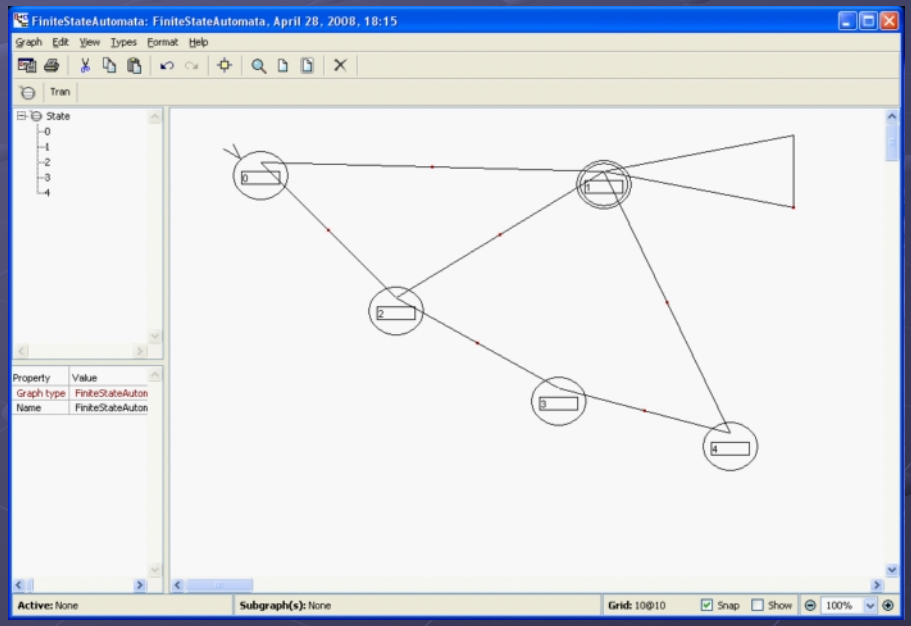
- Meta-modeling the FSM was done by using MetaEdit+'s forms
- State symbols also used MetaEdit+'s dynamic symbol display.
- Initial states had and arrow on its top, and final states had an extra circle





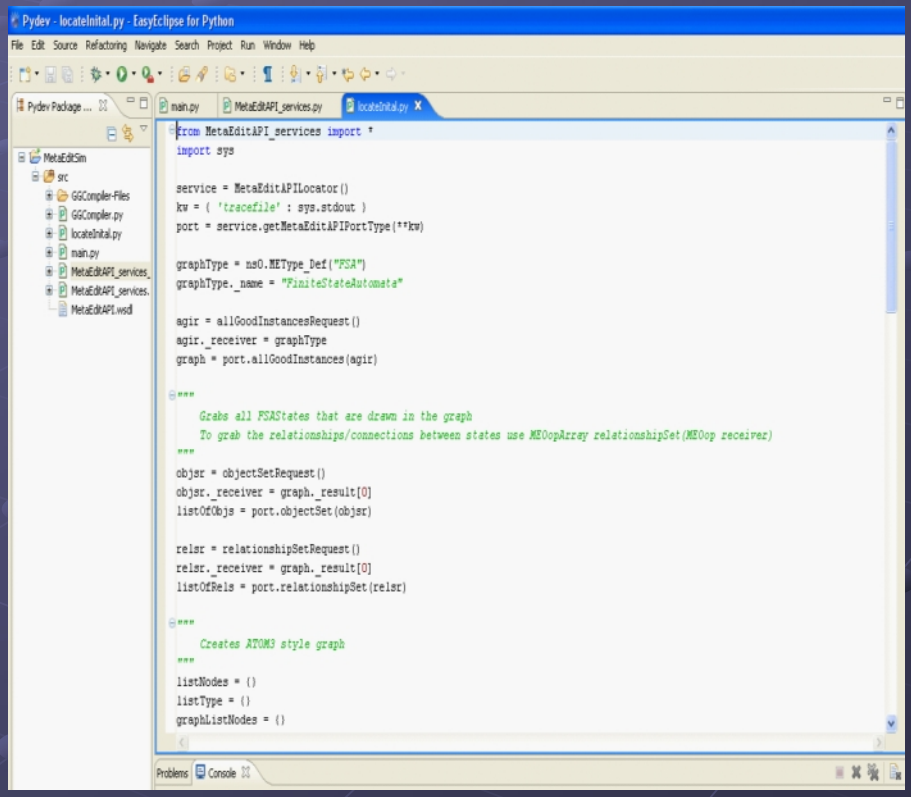
Working with MetaEdit+

- With the FSM meta-model implemented, the next move was to create a model
- This model would be used as a test ground for the use of the MetaEdit+'s API



Working with MetaEdit+

- The API testing that were performed can be summed to the use of its methods to:
  - Retrieve the states and transitions in a model
  - Use animation for the simulation
  - Retrieve a particular properties of objects in the model



Adapting Rules

- In order to generate rules that could be transposed from ATOM3 to MetaEdit+, I used Eugene Syriani's MOTIF (code generation tool)
- Using the FSM rules defined on the first stages of the project, python files that handled GG's were generated by Eugene's code
- These files made use of ATOM3 data to manipulate the graph structures (models) that were created using the FSM meta-model
- The generated python files would serve as the basis for the transformation rules in MetaEdit+

Adapting Rules

- As the rule generation was finished with ATOM3, next was to use them with MetaEdit+
- Due to the MetaEdit+‘s API’s structure, the rules could not be “translated literally”
- MetaEdit+‘s API consists of web service methods (defined in a WSDL file) that can only be called through its local API server (provided by MetaEdit+)
- Also the API does not work with use standard structures, like lists or dictionaries, and data types in their calls

Translation Problems

- The lack of manipulation of standard data structures and data types did cause problems since ATOM3 use them extensively
- MetaEdit+ manipulates everything with its own data types and array variants. Everything centers around them
- To make matters a bit more difficult the API never provides direct access to a model.
- For example, in the FSM case, you can only retrieve all the states, or all the transitions, but never retrieve them together as a whole graph. Thus having access to how they are connected to each other becomes non trivial.

Translation Problems

- To add to this problem the API, when used with the SOAP library I used, calls seem to break within loops, so the code flow for these rules had to be broken down, and were not optimally arranged
- Also you never have access to the objects in a graph and their, you can only get references to it, namely their object IDs.
- Never having direct access to objects and properties, a programmer can access only them through a provided web service.
- In large programs, this becomes a problem (the methods, most of the time, retrieve single properties and objects and not in bundles) since a large number of API calls can be made, which considerably slows down the program

Translation Problems

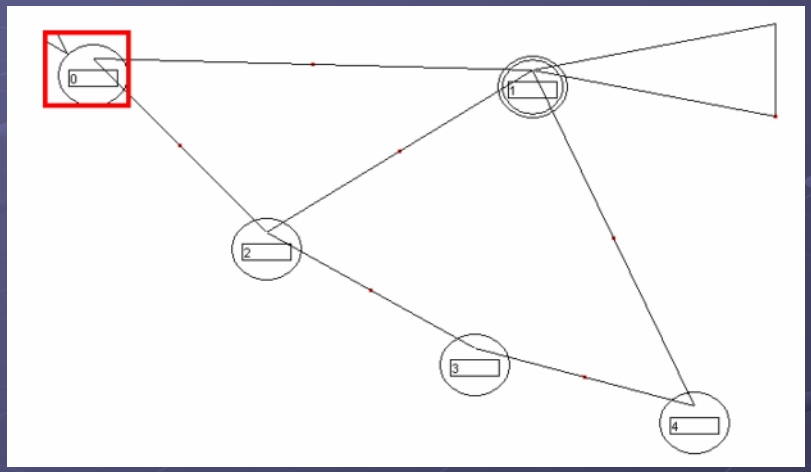
- These limitations make the translated rule code much bigger than it actually is on ATOM3. Ex: below to the left is the Locate Initial from ATOM3, and to the left is its MetaEdit+ counterpart

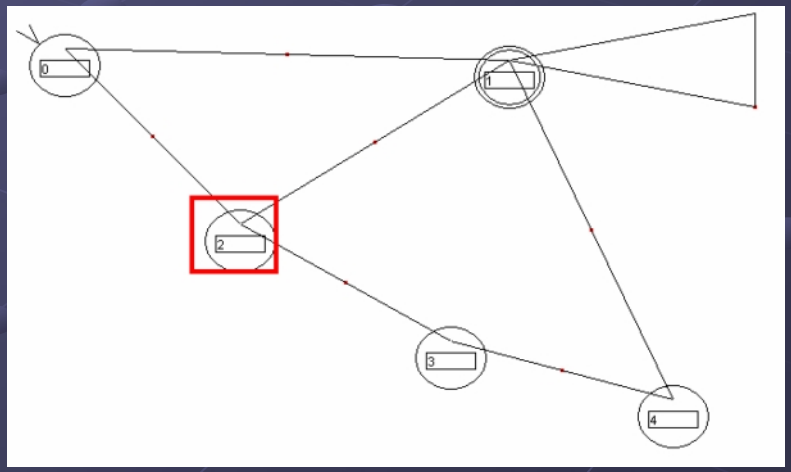
```
#######################
# Transform subgraph #
#######################

# Remove
# Modify attributes
# State
M[1].isInitial.setValue((None, 1))

# Create new nodes
# Current
M[3] = Current()
graph.listNodes['Current'].append(M[3])
# Points_to
M[4] = Points_to()
graph.listNodes['Points_to'].append(M[4])
# Current -> Points_to
M[4].in_connections_.append(M[3])
M[3].out_connections_.append(M[4])
# Current -> Points_to
M[3].out_connections_.append(M[4])
M[4].in_connections_.append(M[3])
# Points_to -> State
M[1].in_connections_.append(M[4])
M[4].out_connections_.append(M[1])
```

```
def initSim():

    service = MetaEditAPILocator()
    kw = { 'tracefile' : sys.stdout }
    port = service.getMetaEditAPIPortType(**kw)

    graphType = nsO.METype_Def("FSA")
    graphType._name = "FiniteStateAutomata"

    agir = allGoodInstancesRequest()
    agir._receiver = graphType
    graph = port.allGoodInstances(agir)

    """
        Grabs all FSAStates that are drawn in the graph
        To grab the relationships/connections between states use MEOopArray relationshipSet(MEOop receiver)
    """
    objsr = objectSetRequest()
    objsr._receiver = graph._result[0]
    listOfObjs = port.objectSet(objsr)

    relsr = relationshipSetRequest()
    relsr._receiver = graph._result[0]
    listOfRels = port.relationshipSet(relsr)

    """
        Creates ATOM3 style graph
    """
    listNodes = ()
    listNodes['State'] = []
    listNodes['Transition'] = []

    for i in range(0, len(listOfObjs._result)):
        listNodes['State'].append(listOfObjs._result[i])

    for i in range(0, len(listOfRels._result)):
        listNodes['Transition'].append(listOfRels._result[i])

    index = 0
    for i in range(0, len(listNodes['State'])):
        request = valueAtRequest()
        request._receiver = listNodes['State'][i]
        request._valueAt = 2
        rxc = port.valueAt(request)

        if(rxc._result._meValue == "true"):
            index = i
            break

    """
        Animates the current state
    """
    animate = animateRequest()
    animate._receiver = graph._result[0]
    animate._animate = listNodes['State'][index]
    listNodes['Current'] = listNodes['State'][index]
    animateResult = port.animate(animate)
    print "Animate State %s" % listNodes['Current']._objectID

    return listNodes, graph, port
```

Executing the FSM simulation

- Like the rules generated by Eugene's code, the MetaEdit+ rules are classes that must have the graph of objects in a model passed to it
- Once the graph is received, the MetaEdit+ rules emulate the ATOM3 three rules and by first locating the initial state, and then switching states according to strings passed through stdin
- One thing to note is that with problem with loops, the animation of the simulation cannot be completely done
- Only the first and second states can be animated before the program crashes. To avoid this problem the ID of the current state is printed out on the console





A note about API access

- Also it is important to say that since the API only provides access through a WSDL file, languages that do not have tools to parse and generate the data types and methods from this file, are rendered useless
- I mention this, because the first Python library that I used (SOAPpy) to parse the WSDL ignored the data types defined in it
- Thus I was forced to switch to a different library, the Zolera SOAP Infrastructure (ZSI) library , which did all the parsing necessary to provide complete access to the API

Conclusions

- The API provides some great access methods to models that even though different from the intern logic of ATOM3 can be very useful in transformation rules
- The use of web services for their API does provide great flexibility and access for any programming language, as long as there is a library that parses XML correctly

Conclusions

- The API documentation is simplistic and does not provide much information besides what methods do
- There is no support for error messages, so API error messages can be cryptic
- Not having direct access to objects, within models, helped slowing down the FSM simulation considerably
- There are some handling issues with method calls and their arguments. Since, MetaEdit+ is a closed source software you cannot investigate its internal parts to figure out these problems
- Also there is need for a SOAP library that execute these API calls properly, or most of the API work will be ravaged by crashes

References

- Kelly, S., Lyytinen, K., and Rossi, M., "MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment", Proceedings of CAiSE'96, 8th Intl. Conference on Advanced Information Systems Engineering, Lecture Notes in Computer Science 1080, Springer-Verlag, pp. 1-21, 1996
- Juha-Pekka Tolvanen, Risto Pohjonen, Steven Kelly, "Advanced Tooling for Domain-Specific Modeling: MetaEdit+", 7th OOPSLA workshop on Domain-Specific Modeling, pp. 1-8, 2007
- Hans Vangheluwe, Juan de Lara, "Meta-Models are Models too", Proceedings of the 2002 Winter Simulation Conference, pp. 1-9, 2002
- Mark Pilgrim, "Dive into Python", Chapter 12, pp 168-182, 2000, 2001, 2002, 2003, 2004. (http://www.diveintopython.org/toc/index.html)
- Rich Salz, Christopher Blunck, "ZSI: The Zolera SOAP Infrastructure", Chapters 2, 3, and 11, February 2005. (http://pywebsvcs.sourceforge.net/zsi.html)
- MetaEdit+ 4.5 User's Manuals, MetaCase, March 2008 (http://www.metacase.com/support/45/manuals/index.html)
- ATOM3 Documentation, McGill University (http://atom3.cs.mcgill.ca/#doc)

Questions?