How-To Guide for MCL2.0a
Matt Schmill
8/23/2011

The following tasks are common to integration efforts.

How do I create a response fringe for my host?

This is accomplished by adding concrete responses to the response ontology.

1. Create a new ontology file to hold your fringe nodes.
2. Use the 'include' directive to include CORE ontologies.
3. Add fringe nodes for the concrete responses to your host, linking them in to the existing response ontology. You can use the extended CRC code to avoid having to create new symbols.

```
node concResponse(name=rebuildDecisionTrees,
 doc="rebuild decision trees used to model X.",
 code=crc_extended_code)
link specification(src=rebuildPredictiveModels,dst=rebuildDecisionTrees)
```

You will need to re-run the mclconfig program to generate default CPTs that incorporate your new ontology structure.

1. `make config`
2. `bin/mclconfig cpt myOntology default`

How do I add a sensor class to MCL?

1. add a symbol entry to the “SC” group in utils/symbols.def
   `psym MYSCLASS SC`
2. `make clean`
3. `make install`
4. add a node to your ontology fringe
   `node concInd(name=sc:mysclass)`
5. link the sensor class node into the ontology core
   `link abstraction(src=sc:mysclass,dst=observable)`
   (actually there should probably be a hierarchy of sclasses here)
6. `make config`
7. tweak your CPTs

How do I add an expectation class to MCL?

Replace all instances of XXX with your expectation name.

1. add symbol to symbols.def
   `psym XXX_EXPECTATION EC       //!< expectation: always remain legal`
2. add class def to expectations.h (use examples, make sure you subclass!)
3. implement mandatory methods in expectations.cc
   1. `XXX::violation()`
   2. `XXX::describe()`
   3. `XXX::baseClassName()`
4. implement XXX::addLinkTags() in dynamicILinks.cc
   see examples in expectations.cc – the expectation class can induce initial activation in the indications ontology here
5. add a factory method to expectationFactory (expectations.cc)
   these are basically wrapper functions for the expectation constructor
   `mclExp* expectationFactory::makeXXXExp(mcl *m, … )`
6. add cases for EC_XXX_EXPECTATION in applicable methods
   I apologize for the chain you will need to connect, starting from the API calls:
   mclMA::declareSelfExpectation(…) // pre-processes obs names
   mclMA::declareObjExpectation(…) // pre-processes obs names
   declare_expectation_pp(…) // checks for variable and EG's
   _ma_makeExp(…)

7. `make clean`
8. `make install`

Note: look at the code for the other expectation classes! *USE PP METHODS TO CHECK SENSOR VALUES*! PP means “PreProcessed” and your sensor names will be preprocessed in the API. In other words, self sensors will get to your violation method as “self.x”. If you use sensor_value() rather than sensor_value_pp() your code will not work properly!