# Mode Model Generator Design

After receiving the mode request from TA3, Mode Model Generator (MMG) is invoked for generating hardware specific mode descriptions. Figure 1 shows the data flow diagram of MMG. We can see from the diagram that the generator first invokes BaseVISor by taking hardware agnostic rules for the requested mode that include the knowledge base(ConcertOlogy) as input and generates generic mode descriptions. The generic mode descriptions are then forwarded to Solver. Solver returns a CSV file that shows realization types of components in the mode as well as their scheduling. After that, MMG then extracts RDF facts on component realizations by parsing the file. These facts are then inserted into the generic mode descriptions. Lastly, MMG invokes BaseVISor for the second time by taking the same set of rules and flushes out all the facts as hardware specific mode descriptions into an external file.

# Technical Discussions:

1. How do you like me to insert facts that are parsed from the CSV file? As far as I know, I can use OWL API to load the ontology, insert them into the ontology, and then save the ontology as another file. This is the same as what I showed in the data flow diagram.

# Problem Discussions:

1. We are not able to invoke BaseVISor twice with the same set of rules. In order to invoke BaseVISor with rules, we need to specify the path to the ontology in the rules. In the first time, the rules should include the path to ConcertOlogy while in the second time, the rules should include the path to the generic mode descriptions, which does not exist in the first time.

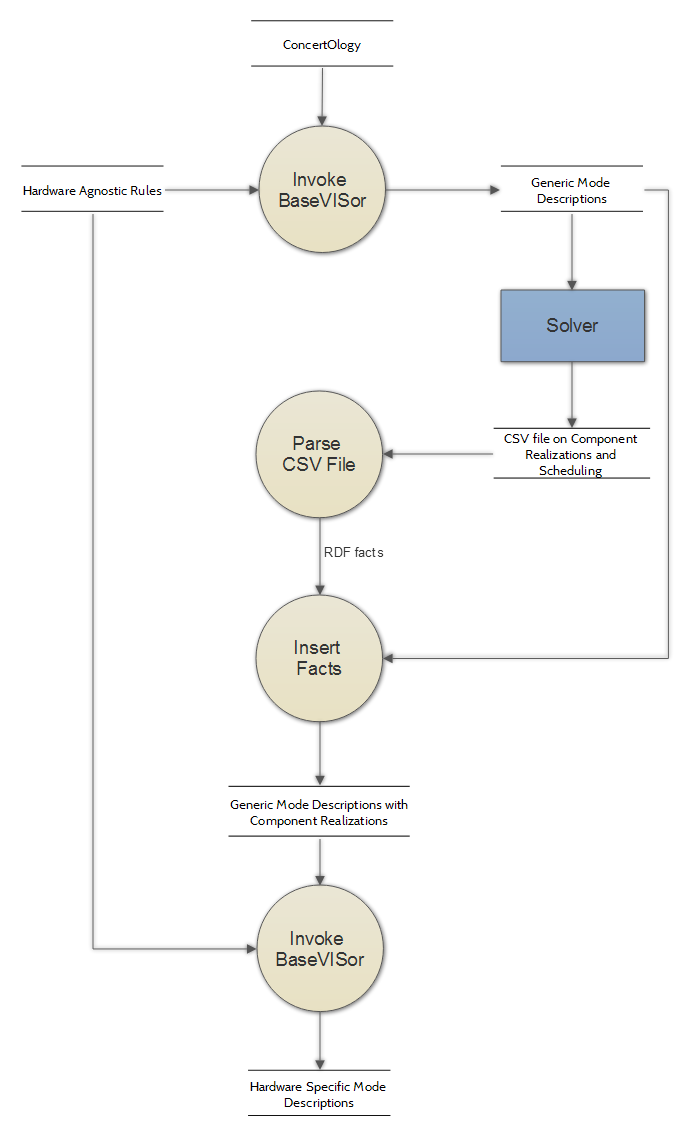


Figure 1 Mode Model Generator Data Flow Diagram

# Mode Code Generator (Software version)

The code generator is implemented in Java and performs the following steps.

* It reads the given RDF mode model from the file using Jena and SPARQL.
* It creates the mode data structure
* It generates C++ code for the state machine
* It generates the mode component C++ code

The mode data structure includes the following.

* Library component list
* State machine data structure
  + State machine data structure includes
    - List of states
    - List of properties
    - List of events
    - List of conduits
  + Each state includes
    - List of transitions
    - List of value to port assignment actions
    - List of property to port assignment actions
  + Each transition includes
    - Source state name
    - Target state name
    - Guard expression
    - List of property increment actions

# Conduit data structure (Software version)

The conduit C++ data structure includes the following.

* Value of C++ template type
* Semaphore file name
* Shared memory file name
* setValue/getValue methods

# Mode component process (Software version)

* This process initializes conduits between the state machine process and library component processes
* It launches (using fork() system call) state machine process and library component processes
* It waits for the STOP command from the user
* Once user enters the STOP command the mode component process sends the STOP signal to all the other processes
* It waits till all the other processes terminate
* It destroys all the conduits
* The mode component process terminates

# Ontology to C++ code Mapping

The following describes mapping between the mode model elements and the generated state machine C++ constructs.

Software State Machine

|  |  |
| --- | --- |
| State Machine Element | C++ Code Construct |
| Specific state  State OnEnter/OnExit actions  Specific transition  Transition guard  Transition actions  Event  Port  Property | Specific state class + instance + state ID constant  OnEnter/OnExit methods  Instance of Transition class  If statement + conditional expression in the Event handler method  Statements in the transition guard if statment block  Event handler method  Conduit class + Linux shared memory + Linux semaphore  Global variable |

Hardware State Machine

|  |  |
| --- | --- |
| State Machine Element | C++ Code Construct |
| Specific state  State OnEnter/OnExit actions  Specific transition  Transition guard  Transition actions  Event  Input port  Output port    Property | Specific Switch/Case block + state ID constant  None  Assignment of a state ID constant to the state machine interface function parameter of pointer type, assignment of a state ID constant to the state machine interface function  If statement + conditional expression in the specific  Statements in the transition guard if statement block  Specific Switch/Case block  State machine interface function parameter  State machine interface function parameter of pointer type  Local variable |