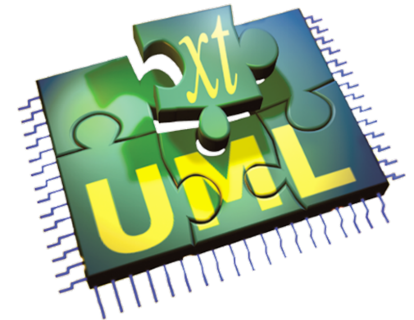
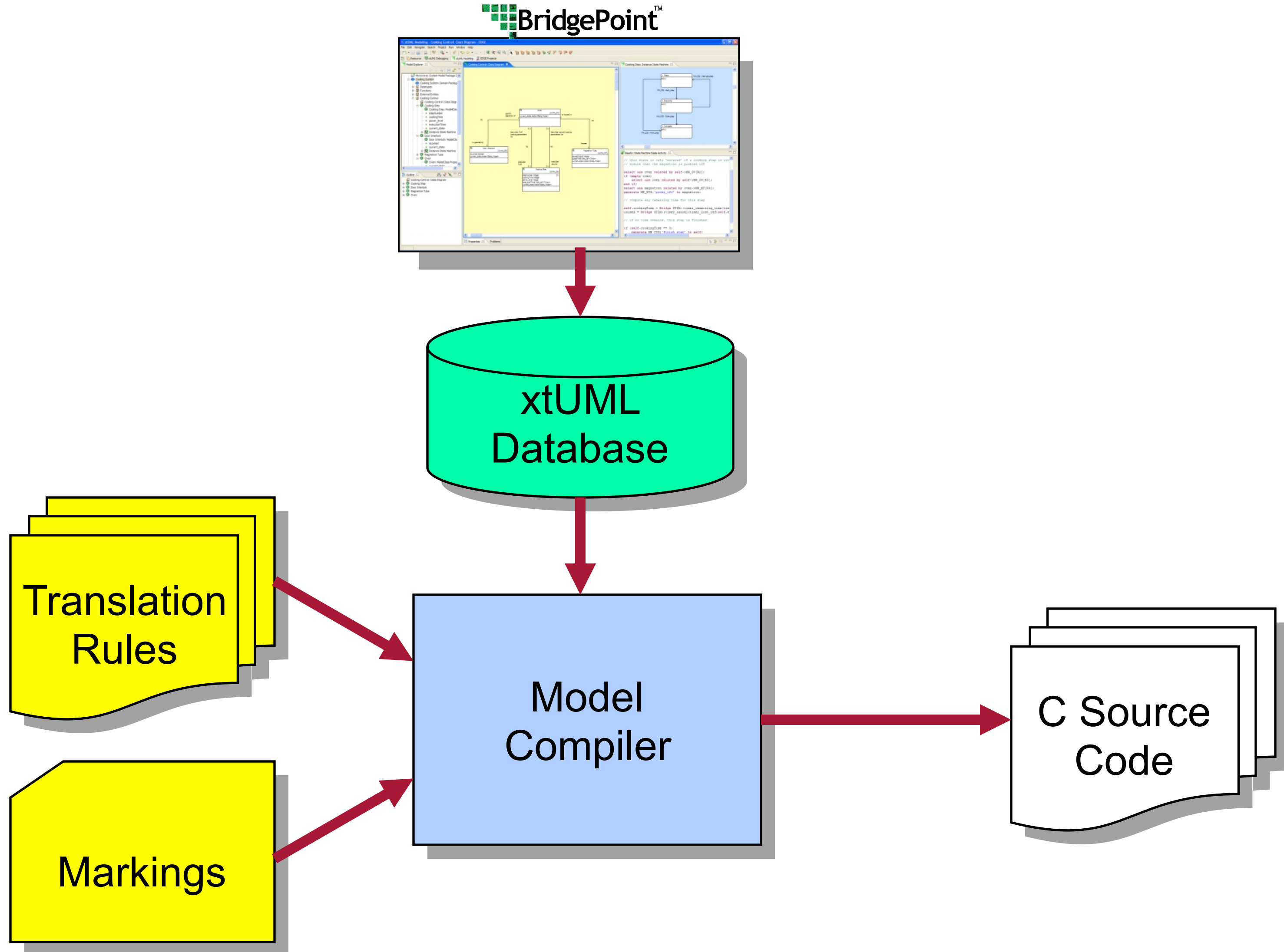
The xtUML method – Code Generation

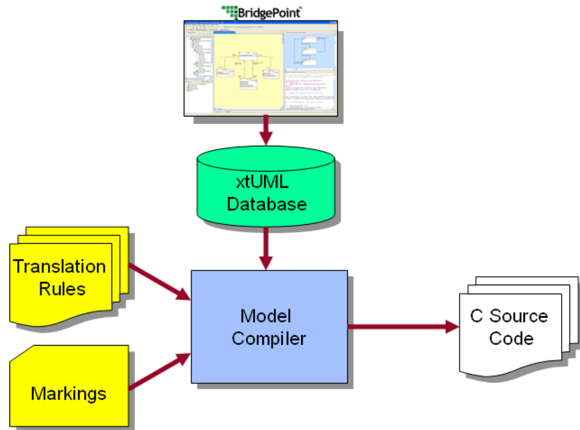
- **Analysis** – questioning, thinking, sketching...
  - Descriptive UML diagrams
    - use case, sequence, ...
- **Executable Modeling** – formalizing the analysis:
  - Component Diagrams (partitioning/interfaces)
  - Class Diagrams (data)
  - State Machines (control)
  - Activities (processing)
- **Verification**
  - Interpretive Model Execution
- **Code generation**
  - **Template and Rule-Based Translation**

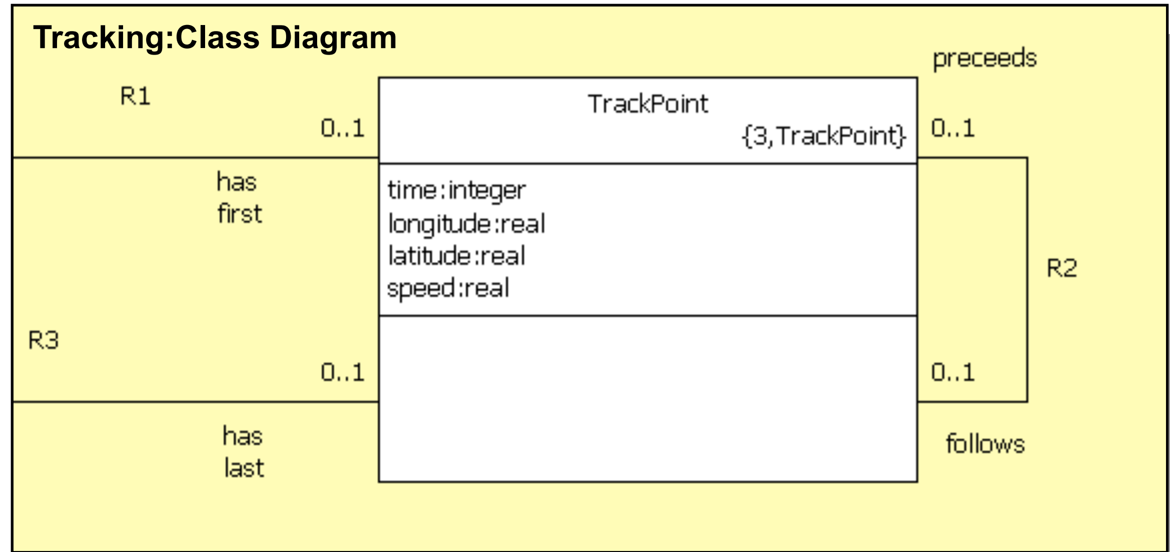


Code Generation



Generating Classes



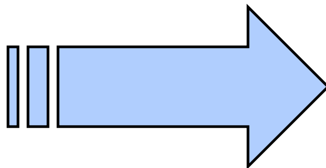


Tracking_Trackpoint_class.h

```
struct Tracking_TrackPoint {
  /* application analysis class attributes */
  i_t time;  /* - time */
  r4_t longitude;  /* - longitude */
  r4_t latitude;  /* - latitude */
  r4_t speed;  /* - speed */
  /* relationship storage */
  /* Note:  No storage needed for TrackPoint->TrackLog[R1] */
  Tracking_TrackPoint * mc_TrackPoint_R2_follows;
  Tracking_TrackPoint * mc_TrackPoint_R2_preceeds;
  /* Note:  No storage needed for TrackPoint->TrackLog[R3] */
};
```

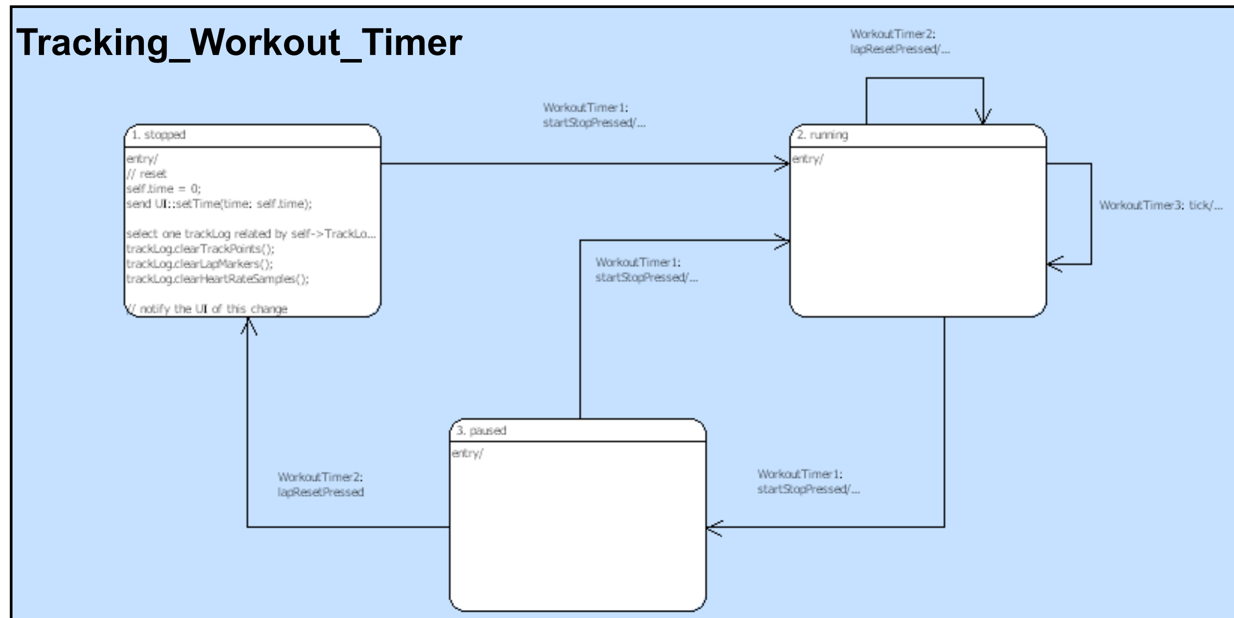
Translation Rules

```
struct <class name>_s {
  <attr1 type>    <attr1 name>; /*  <attr1 description> */
  <attr2 type>    <attr2 name>; /*  <attr2 description> */
  <attr3 type>    <attr3 name>; /*  <attr3 description> */
     . . .
/* Association storage */
  <ref1 class name>_s *   < ref1 class name >_<assoc1 number>;
  <ref2 class name>_s *   < ref2 class name >_<assoc2 number>;
     . . .
 /* State machine current state */
  StateNumber_t    current_state;
};
```



```
struct Tracking_TrackPoint {
  /* application analysis class attributes */
  i_t time;  /* - time */
  r4_t longitude;  /* - longitude */
  r4_t latitude;  /* - latitude */
  r4_t speed;  /* - speed */
  /* relationship storage */
  /* Note:  No storage needed for TrackPoint->TrackLog[R1] */
  Tracking_TrackPoint * mc_TrackPoint_R2_follows;
  Tracking_TrackPoint * mc_TrackPoint_R2_preceeds;
  /* Note:  No storage needed for TrackPoint->TrackLog[R3] */
};
```

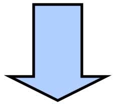
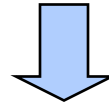
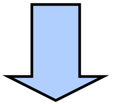
State Machine Generation




```
state = instance->current_state;
next_state = Tracking_Workout_Timer_StateEventMatrix[ state ]
[ event_number ];
/* Update state and execute the state action */
instance->current_state = next_state;
( *Tracking_Workout_Timer_Actions[ next_state ] )( instance, eventData );
 . . .
```

Translation Rules: Event Dispatch

```
. . .
state = instance->current_state;
next_state = <class name>_StateEventMatrix[ state ][ event_number ];
/* Update state and execute the state action */
instance->current_state = next_state;
( *<class name>_Actions[ next_state ] )( instance, eventData );
    . . .
```

```
. . .
state = instance->current_state;
next_state = Tracking_Workout_Timer_StateEventMatrix[ state ]
[ event_number ];
/* Update state and execute the state action */
instance->current_state = next_state;
( *Tracking_Workout_Timer_Actions[ next_state ] )( instance, eventData );
 . . .
```

Generated Code structure

- StateEventMatrix contains next state for each current state and input event.

```
Next_state = <class name>_StateEventMatrix[ state ][ event_number ];
```

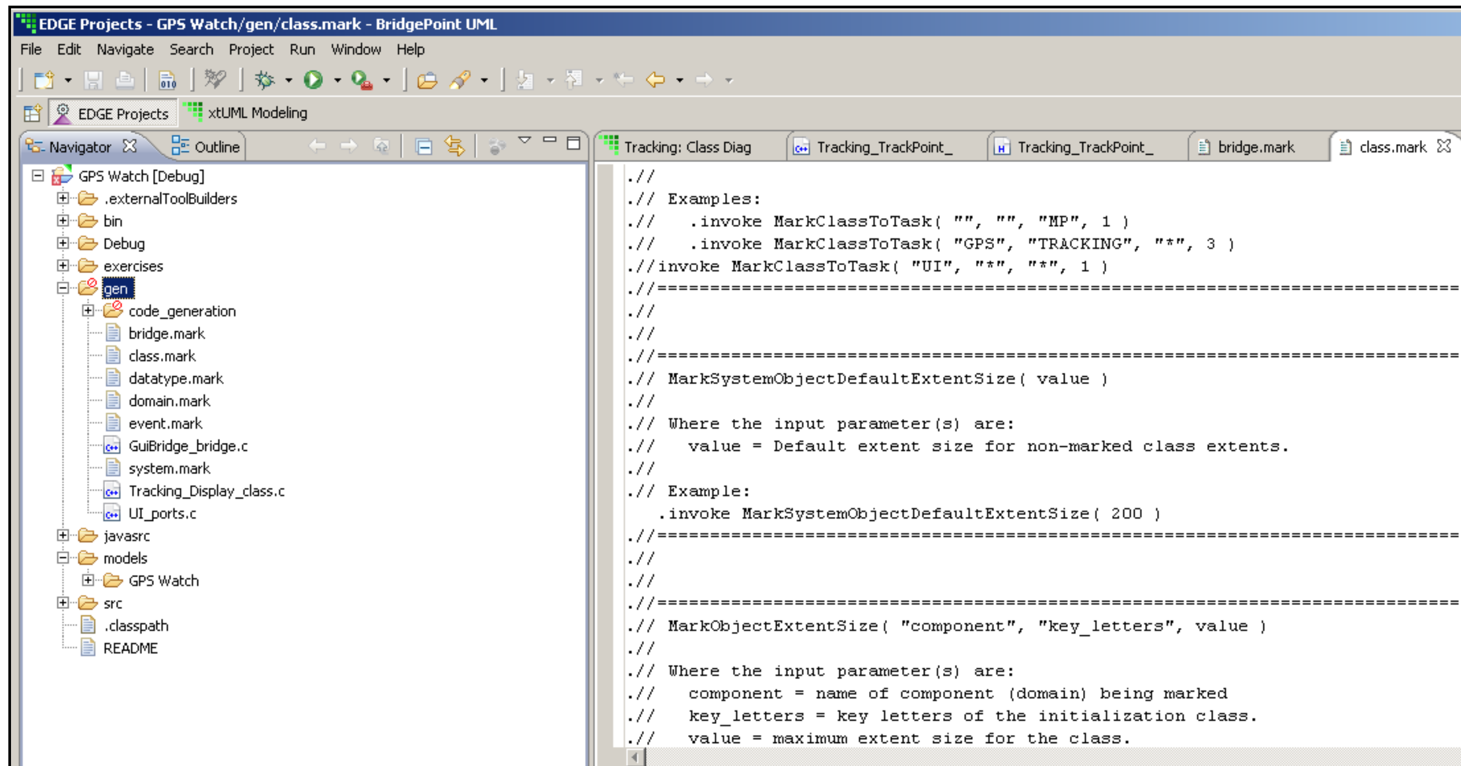
- Actions is an array of function pointers to the generated action code for the state itself.

```
( *<class name>_Actions[ next_state ] )( instance, eventData );
```

- One Procedure for each state machine, and they all are essentially the same – Only the class name is unique.

Markings

- Contained in the gen folder.
- .mark files control details of the code generation.
- 6 mark files available.
  System, domain, class, event, bridge, datatype



Useful Markings

- **MarkActionStatementTracingOn() in domain.mark**

  MarkActionStatementTracingOn is used to enable the generation of trace macros into the generated code that will output run-time trace statements of the Object Action Language statements executed during the run.

- **MarkInitializationFunction("comp", "fname") in domain.mark**

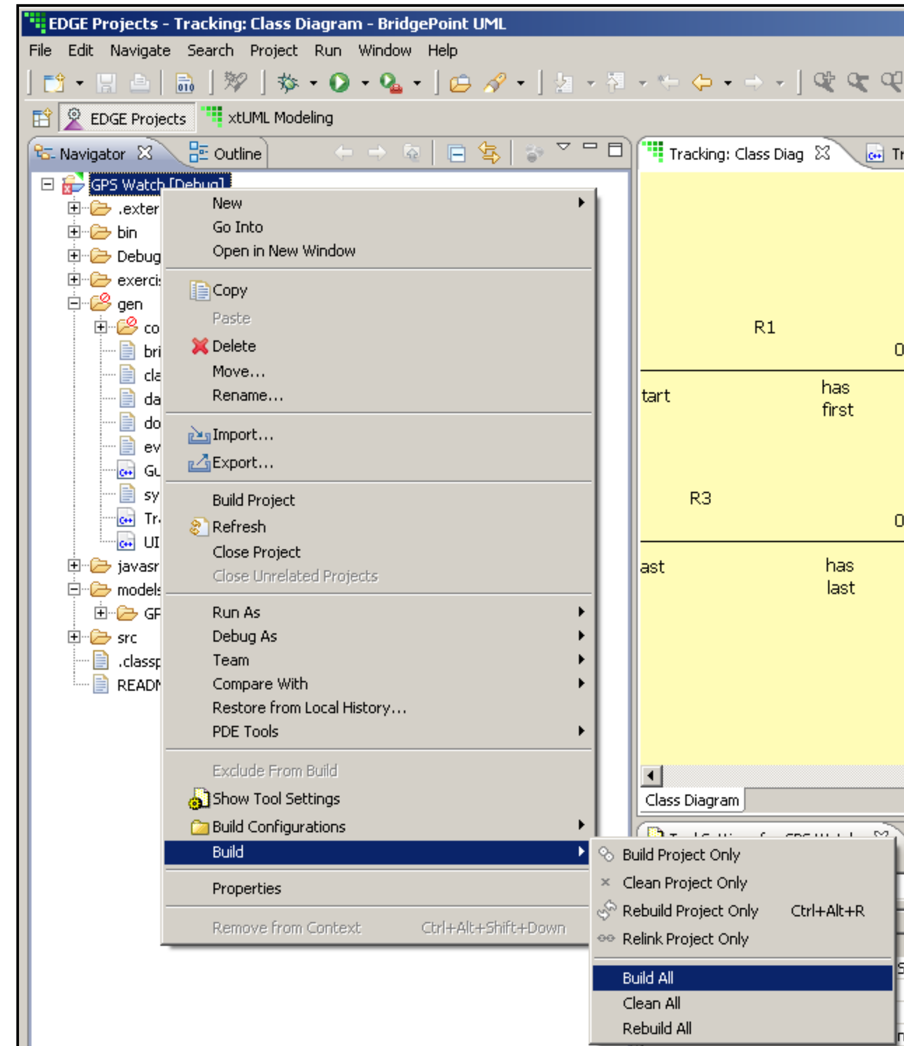
  Designate a function to serve as an initialization function in a domain.

- **TagDataTypePrecision("domain", "dt_name", "tagged_name", "initial_value") in datatype.mark**

  To indicate the 'precision' of a user defined data type. (e.g. double)

Performing Code Generation

- Use the C/C++ Perspective in Eclipse and Build Project.
- Batch generation: xtumlmc_gen_erate
  - -import xtUML_file
  - -nopersist
  - -v verbosity
  - -f output_filename



Document Generation

- **Generate HTML Documentation from your models.**