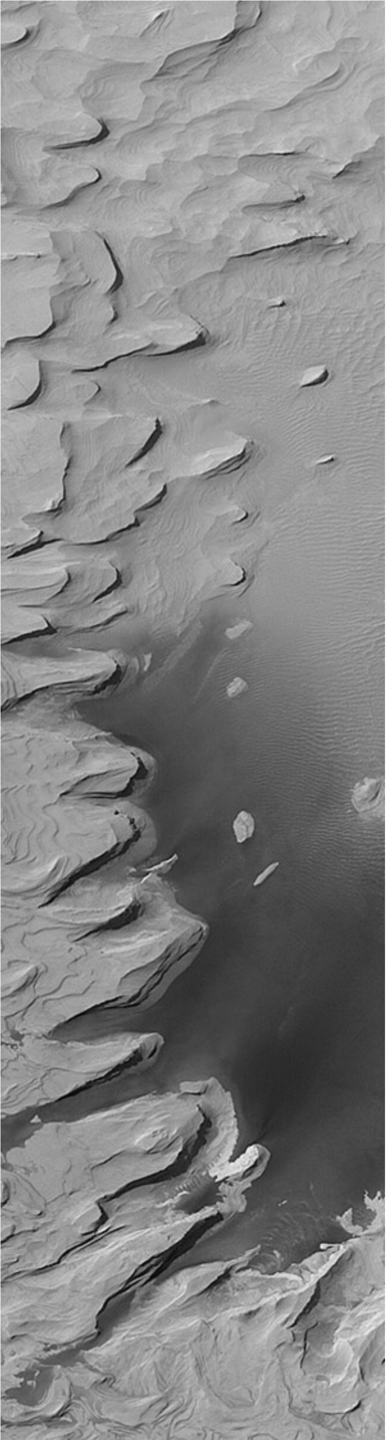
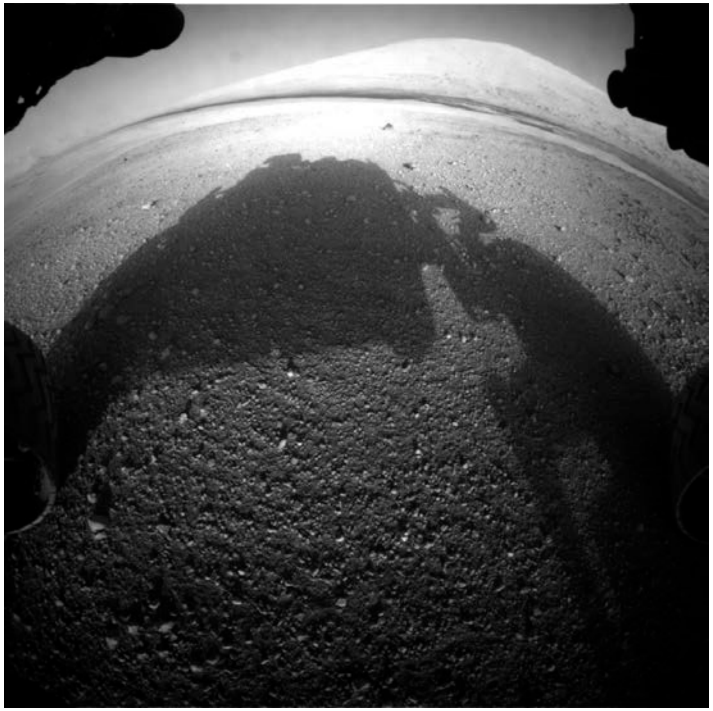


UML Statechart Autocoding for the Mars Science Lab (MSL) Mission

Ed Benowitz
NASA/Jet Propulsion Laboratory
Caltech

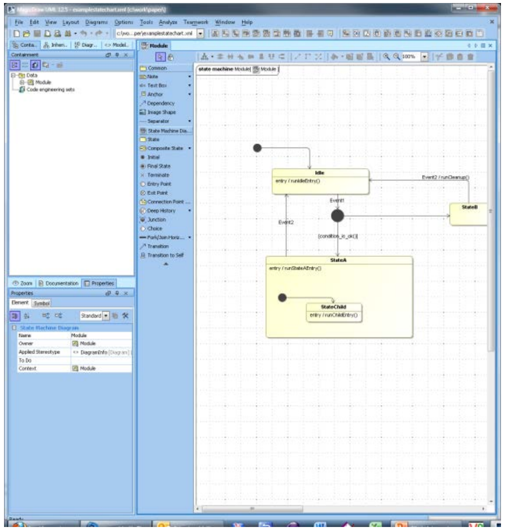
Curiosity is on Mars now



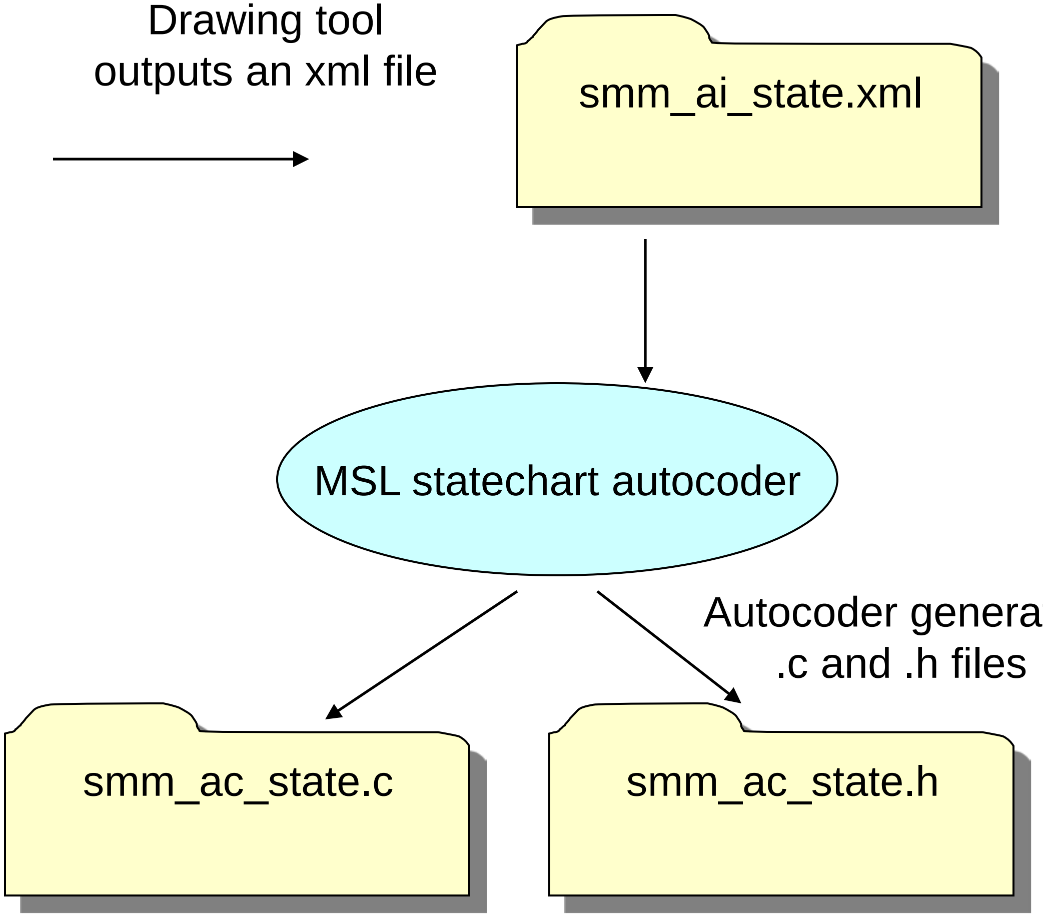
Statechart autocoding

- Generate flight code automatically from a state machine diagram.
- The generated code has been part of Curiosity’s flight software since launch, and continues to run onboard today.

Process



Developer draws
a statechart
in MagicDraw



Drawing tool
outputs an xml file

smm_ai_state.xml

MSL statechart autocoder

Autocoder generates
.c and .h files

smm_ac_state.c

smm_ac_state.h

Pros and Cons

- Advantages
  - Code and documentation are always in sync
  - More precise diagrams
  - Easier to accommodate changes late in the game
  - Encourages communication between systems, flight, test
  - Forces the developer to consider off-nominal scenarios
- Cons
  - Could be overkill for list-like state machines
  - Drawing diagrams takes time

Areas of Use

- Auto-maneuver (Cruise phase)
  - High level state machines sending messages to the attitude control system
  - Handles retries, high-level off nominal situations
  - Turns, acquire attitude knowledge, trajectory correction maneuvers
  - ~ 10 state charts intercommunicating
  - ~ 100 states

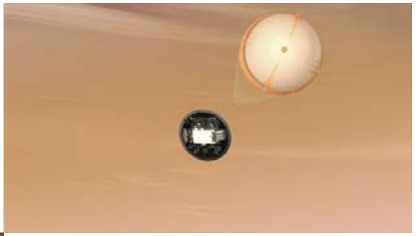
Areas of Use

- Spacecraft Modes
  - ~ 50 states
  - Configures the spacecraft when booting up
  - Re-configures the spacecraft when changing modes

Launch mode

Cruise mode

Entry, descent, and landing mode

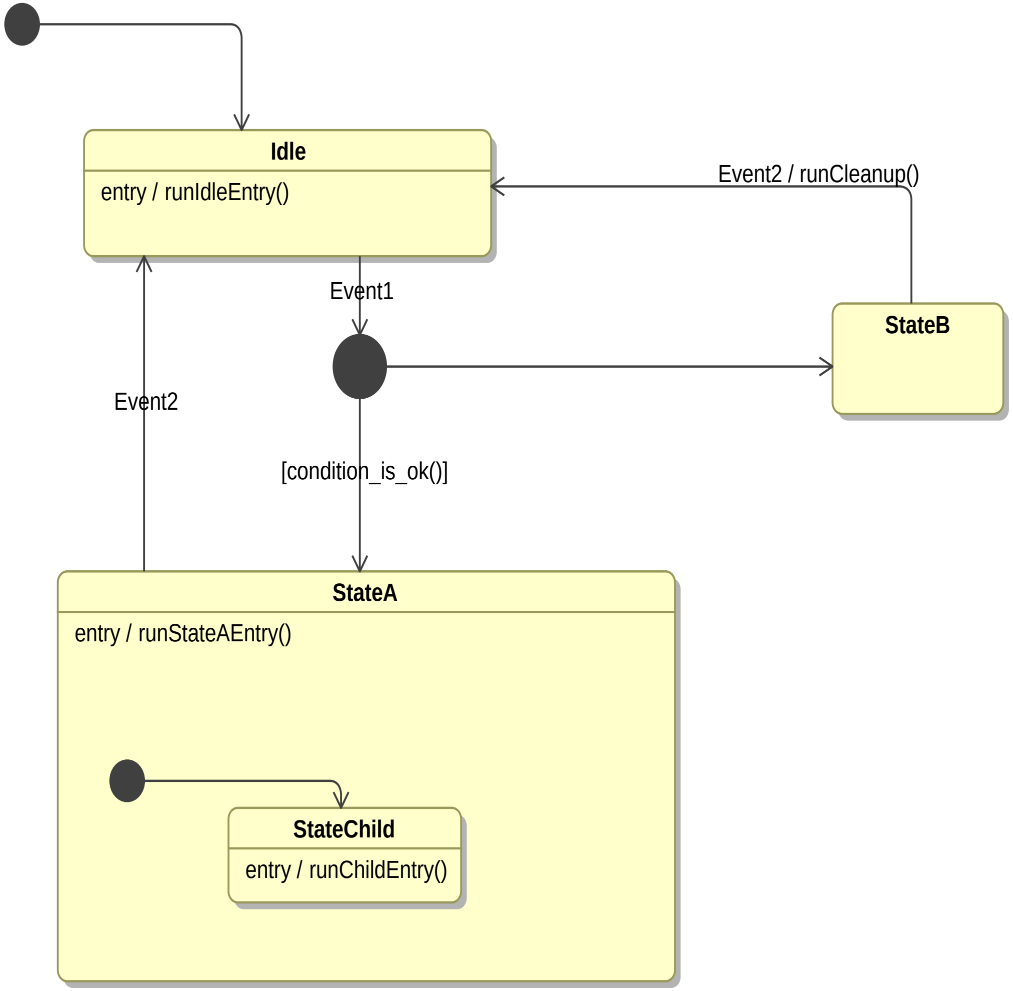


Rover mode

Key Ideas

- Events are function calls
  - mode_event_interruptA(StateMachine * machine);
  - Can have additional arguments
- States are enumerated types
- Event functions contain switch statements
  - Switch on the machine's state enum.
- Composite states are flattened.

Example: Statechart



- This is a generic example statechart; it is not a flight state chart. The generated code shown on the next slides comes from this example only, and is not flight code.

Example Generated Code: .h

```
#ifndef MODULE_AC_STATE_H
#define MODULE_AC_STATE_H

#include <module/module_state_types.h>

typedef enum module_states {
   Idle,
   StateChild,
   StateB
} ModuleStates;

typedef struct module_machine {
   ModuleStates state;
} ModuleMachine;

void module_init_state(ModuleMachine * m);
void module_event_Event2(ModuleMachine * m);
void module_event_Event1(ModuleMachine * m);
void module_report_unrecognizeable_state(ModuleMachine * m);

#endif
```

Example Generated Code: .c

```
#include <module/module_ac_state.h>
#include <module/module.h>

void module_init_state(ModuleMachine * m) {
  ModuleMachine temp = *m;

  temp.state = Idle;
  runIdleEntry();
  *m = temp;
}

void module_event_Event2(ModuleMachine * m) {
  ModuleMachine temp = *m;

  switch(m->state) {
    case Idle:
      break;
    case StateChild:
      temp.state = Idle;
      runIdleEntry();
      break;
    case StateB:
      temp.state = Idle;
      runCleanup();
      runIdleEntry();
      break;
    default:
      module_report_unrecognizeable_state(m);
  }
  *m = temp;
}
```

Example: Generated Code: .c

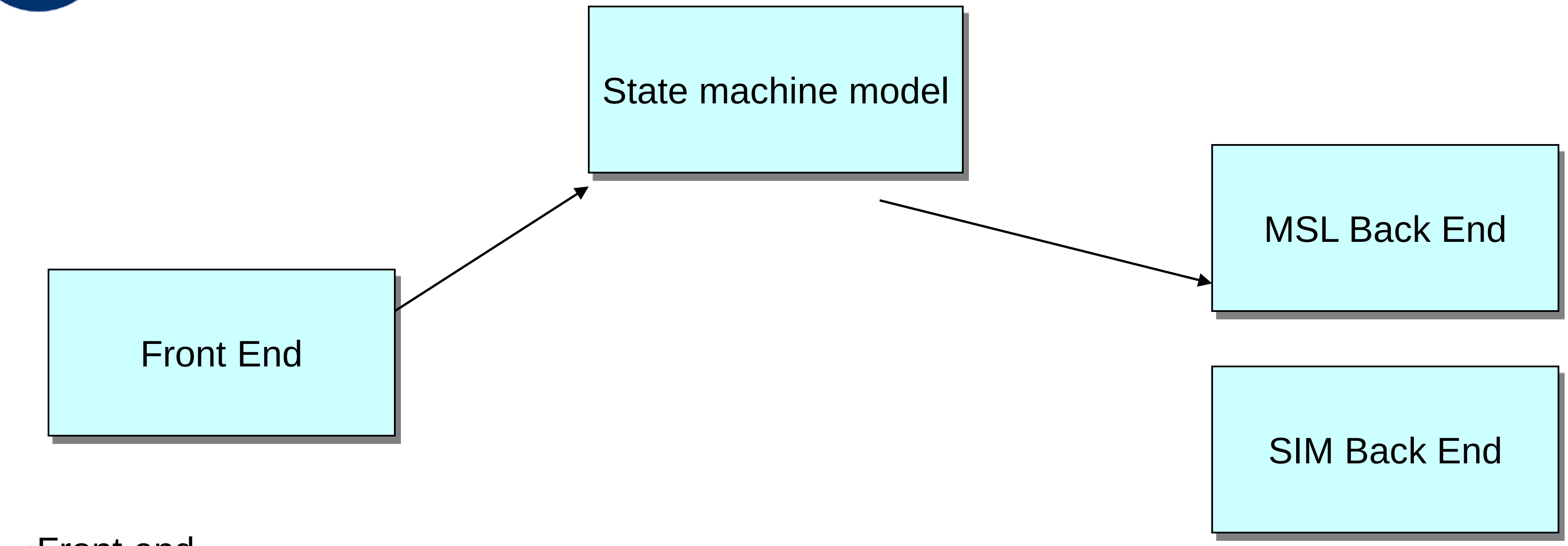
```
void module_event_Event1(ModuleMachine * m) {
  ModuleMachine temp = *m;

  switch(m->state) {
    case Idle:
      if(condition_is_ok() ) {
        temp.state = StateChild;
        runStateAEntry();
        runChildEntry();
      }
      else {
        temp.state = StateB;
      }
      break;
    case StateChild:
      break;
    case StateB:
      break;
    default:
      module_report_unrecognizeable_state(m);
  }
  *m = temp;
}
```

Inter-process communication

- State machines are independent of synchronization mechanism
- Each state machine can only be used within one thread
- If inter-process communication is used to communicate between threads
  - Upon receiving a message, send an event to a state machine

Autocoder Internal Architecture



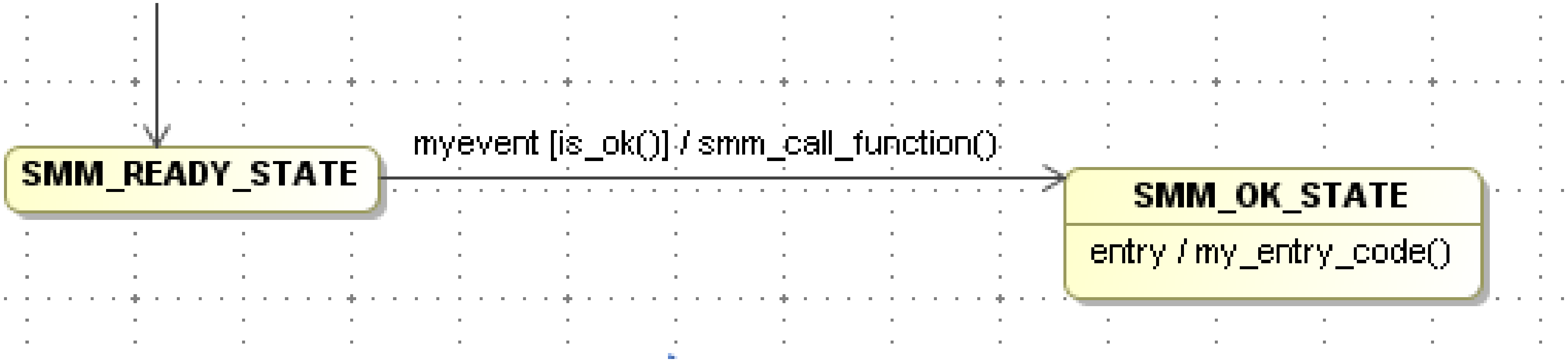
- Front end
  - Builds the state machine model from the XML file

Thanks to Ken Clark for his work on the front end and state machine model

- Back ends
  - Traverse the state machine model
  - Generate code

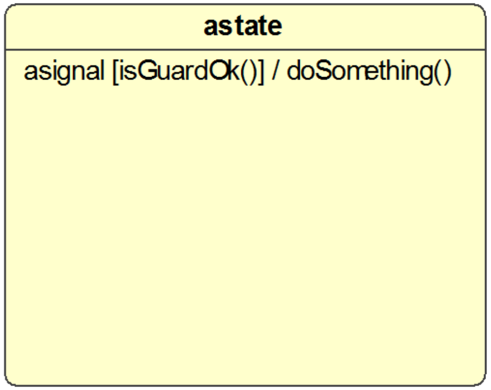
Supported features

- Simple states
- Transitions with
  - Events
  - Guards
  - Actions
- Entry/Exit actions



Supported features

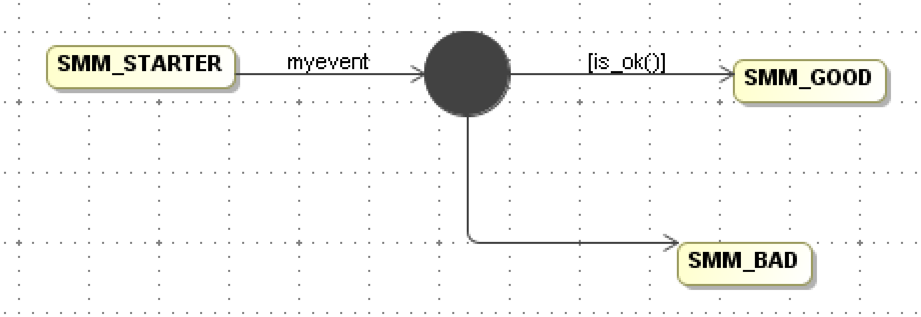
- Internal transitons



astate

asignal [isGuardOk()] / doSomething()

- Self loops
- Junctions



SMM_STARTER

myevent

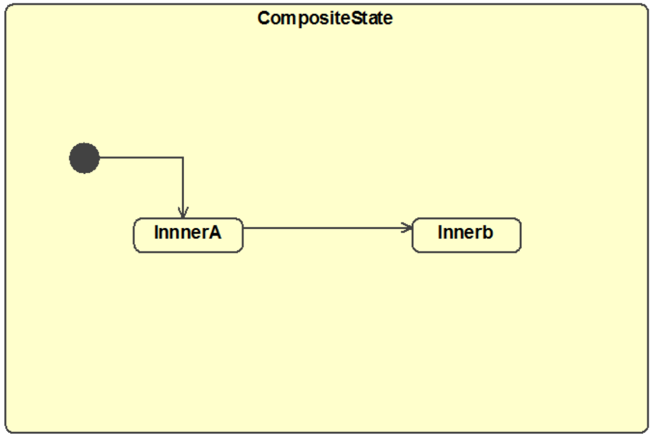
[is_ok()]

SMM_GOOD

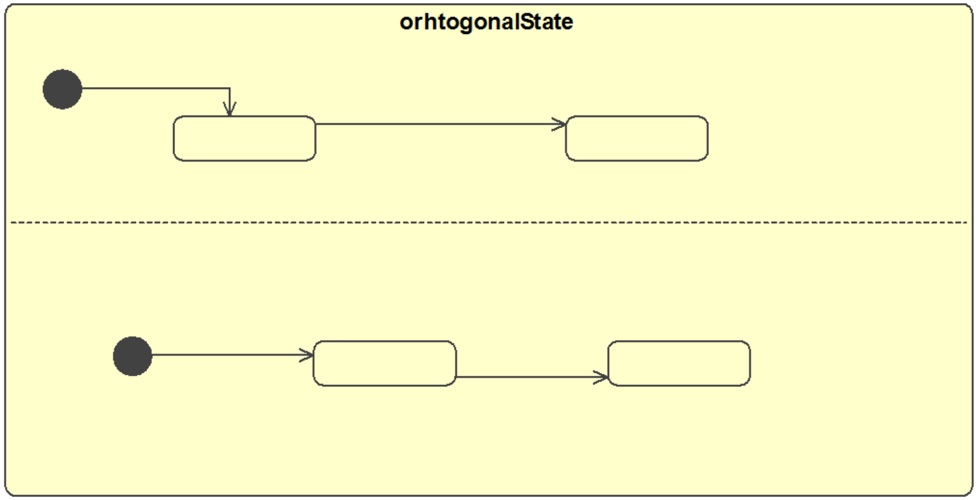
SMM_BAD

Supported features

- Composite states



- Orthogonal regions

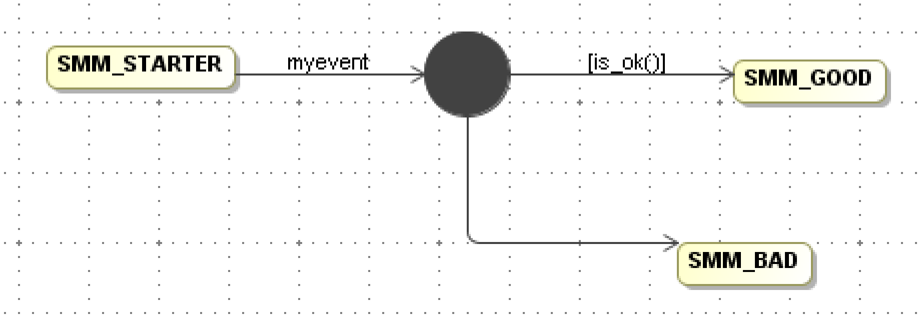


Key Restrictions

- Every transition must be started by an event
  - No simple transitions with only a guard
- Don’t call event functions from within event functions
  - May need to send a message to yourself via IPC instead
- Do not nest orthogonal regions

Key Restrictions

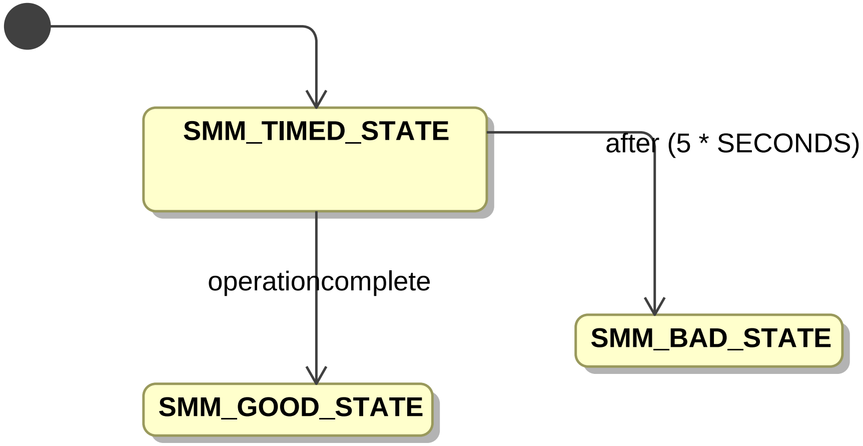
- Avoid ambiguity
  - Use junction in if/else configurations only to avoid ambiguity



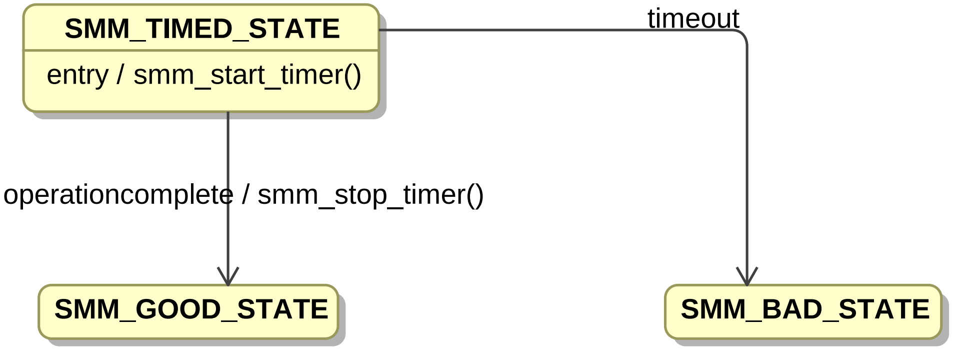
  - The autocoder does not guarantee which orthogonal region executes first
  - Don’t use the same event on multiple transitions from a single state.

Timers

Not supported



Workaround

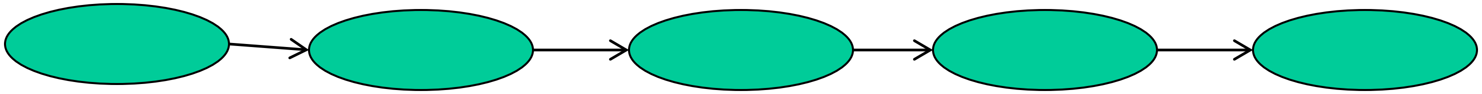

SMM_TIMED_STATE
entry / smm_start_timer()
timeout
operationcomplete / smm_stop_timer()
SMM_GOOD_STATE
SMM_BAD_STATE

Some Lessons Learned

- Accommodated late-breaking requirements changes
- Statecharts were used outside of flight software
  - Communicate with systems and ACS engineers
    - Establish what should be implemented
  - Test engineers
    - Cover every path through the state charts
- What looks like a simple state machine grows larger when off-nominal is added
- Style: Avoid orthogonal regions
  - State chart becomes visually too large to see
  - Determinism: Sending the same event to two regions
    - Who runs first?
- Drawing tool formats change frequently.

Some Lessons Learned

- When to use a statechart
  - Branching, nesting, and looping
- When NOT to use a statechart
  - When the state chart is a single chain



- Do not hand-edit auto-generated code
  - Keep hand-edited and auto-generated code in separate files
- Getting project buy-in
  - Get the project’s blessing on the generated code.
  - Auto-generated code must strictly follow project coding standards for acceptance.

BACKUPS

References

[I] N.F. Rouquette, T. Neilson, and 0. Chen, "The 13"' Technology of Deep Space One", *Proceedings of the 1999 IEEE Aerospace Conference,* Vol 1, March 1999, pp. 477-487.

[2] K. Barltrop, E. Kan, J**.** Levison, C. Schira, and K. Epstein, "Deep Impact: ACS Fault Tolerance in a Comet Critlcal Encounter", *Advances in the Astronautical Sciences,* Vol. **1 1** 1, 2002, pp. **1** 1 1-1 26.

[3] Samek, M.. *Practical Statecharts in C/C++,* CMP Books, San Francisco, 2002.

[4] E. Benowitz, K. Clark, Watney. Auto-Coding UML Statecharts for Flight Software, SMC-IT '06 Proceedings of the 2nd IEEE International Conference on Space Mission Challenges for Information Technology, Pages 413-417.

http://mars.jpl.nasa.gov/msl/

All photos in this presentation came from the public JPL MSL web site.

Not supported

- Forks/joins
- History states
- Entry point/exit point/final state/terminate