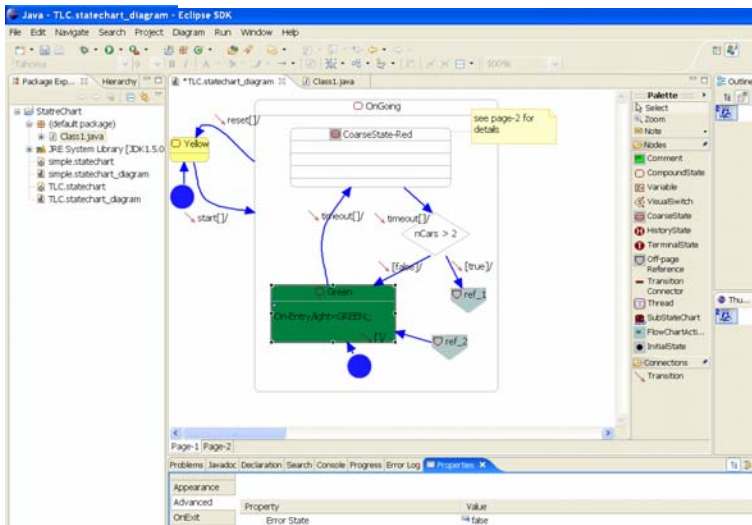


First in

# Temporal Solutions



## Software module specification

The StateRover provides a broad range of features that enable users to specify, generate, and verify the desired behavior of their software.

### Features

- ❑ Eclipse graphical user interface for mixed UML statechart and flowchart models and specifications.
- ❑ Automatic C, C++, and Java code generation.
- ❑ Simultaneous diagram and source level debugging.
- ❑ Run-time verification of behavioral requirements.

*The StateRover graphical programming environment supports statecharts, flowcharts, and advanced software design constructs for representing, building, and verifying complex software modules and classes.*

### Benefits

- ❑ Intuitive graphical programming tool with automatic modular and portable code generation
- ❑ Combined modeling, formal specification, run-time verification, and automatic test generation.
- ❑ Easier, faster, and more correct than hand written code.

Time Rover's StateRover™ is a diagrammatic programming, specification, validation, and verification tool based on graphical constructs such as statecharts, state transition diagrams and flowcharts.

The State Rover is optimized for each phase of the embedded systems and software development process. Using the StateRover requires just three steps to convert software to production-ready software classes or modules:

1. Specification of software modules and classes using advanced diagrammatic constructs.
2. Implementation of software modules and classes via automatic code generation in C, C++, and Java. Generated code is efficient, readable, and ready for integration and deployment.
3. Graphical debugging of software modules and objects that execute on a host or a target.

In addition, an optional fourth step supports validation, automatic test generation, and run-time verification of behavioral requirements specified as statechart-assertions.

### Supported hosts

- ❑ Eclipse development environment.



## Statechart constructs

States and transitions are the basic elements of a statecharts.

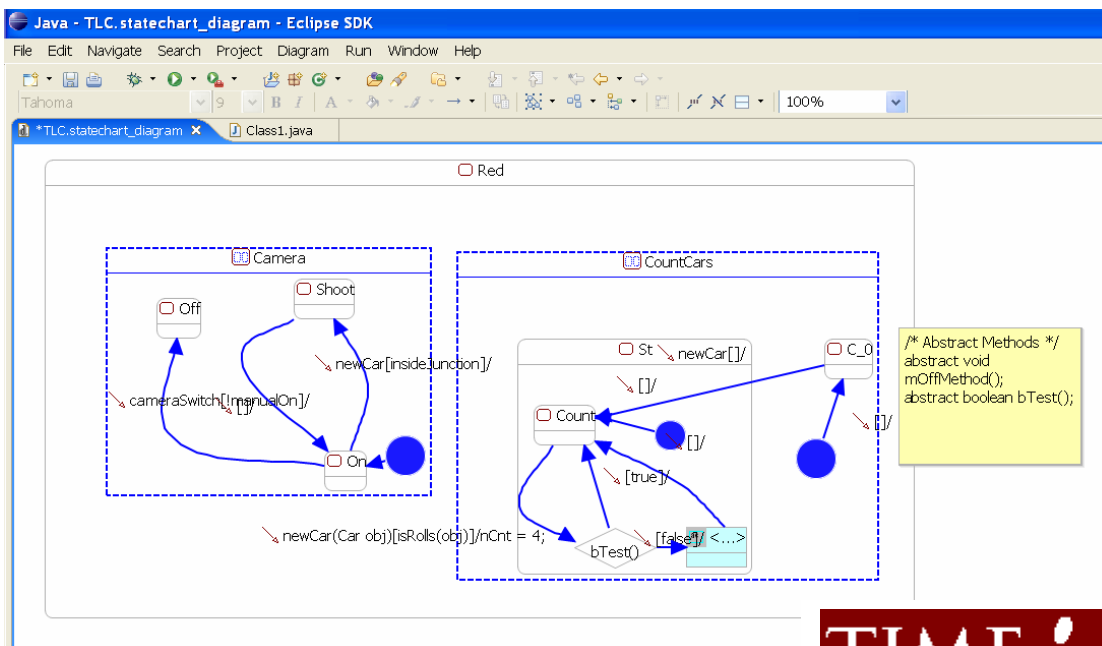
Each state has one or more of the following characteristics:

- ❑ Nested state – a state nested in a higher state; enables top down state refinement.
- ❑ Default state – specifies the default activated state within a nested state or thread.
- ❑ Terminal state – a means to suspend a chart's execution until the chart is reactivated.
- ❑ Actions executed “On-Entry”, “During”, and/or “On-Exit”.
- ❑ Statechart assertions for formal requirement specification and run-time monitoring.
- ❑ Concurrent/independent threads – a graphical way to represent two or more “independent” threads of execution within a given state.
- ❑ Flowchart decision and action boxes – visually integrate flowcharts within statecharts, anywhere within the state hierarchy.
- ❑ Transitions are condition-based, event-based, or both and contain an optional action.
- ❑ Code generation for multi-threaded applications.

## Advanced software design constructs

In addition to statechart and flowchart constructs, the StateRover supports graphical specification of advanced software concepts including:

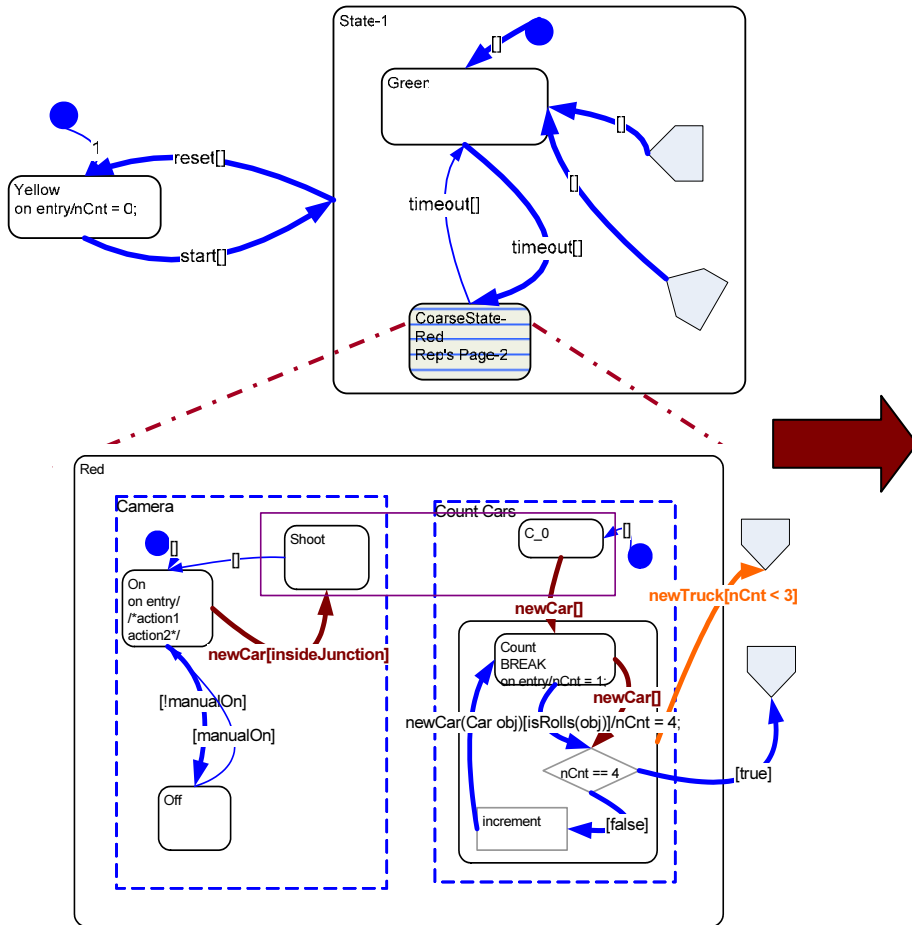
- ❑ Visual synchronization – a graphical means to “synchronize” threads of execution based on active states in one or multiple threads of execution.
- ❑ Visual priority – a means to resolve “race” conditions that occur when two or more transition conditions are active at the same time.
- ❑ Critical Region – a graphical representation of an interthread exclusive region, wherein only one state can be active at a time.
- ❑ Sub-statechart – a means to reuse a chart within other charts.
- ❑ Auto-generated try-catch-finally protection and auto-generation of interface with JML annotation.
- ❑ Local attributes, variables, and methods can be defined using the *Locals* box.



**TIME ROVER**

## Automatic code generation

The StateRover's sophisticated code generation algorithms automatically generate readable, production-ready, optimized code. Users maintain control over the code generation process through several user-selectable options.



```

class TLC {

    TLC() {
        TRreset();
    } // constructor

    void TRreset() {
        ...
    } // TRreset()

    /* Event handler for event:timeout()*/
    int timeout(){
        ...
    } //timeout()

    /* Event handler for event:newCar()*/
    int newCar(){
        ...
    } //newCar()

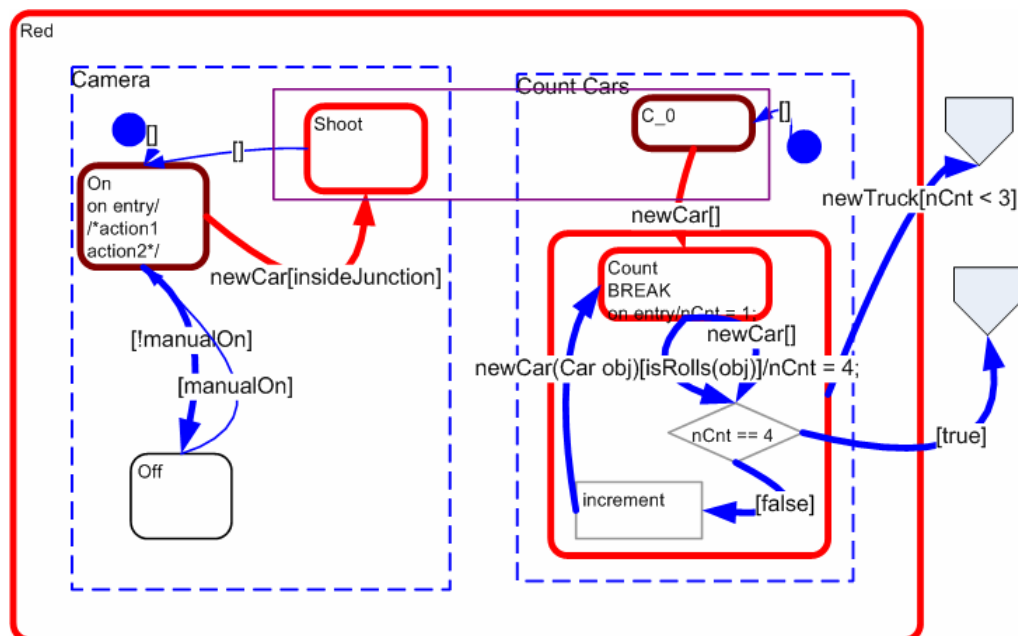
    /* Event handler for event:newTruck()*/
    int newTruck(){
        ...
    } //newTruck()

} // class TLC
    
```

## Visual debugging (animation)

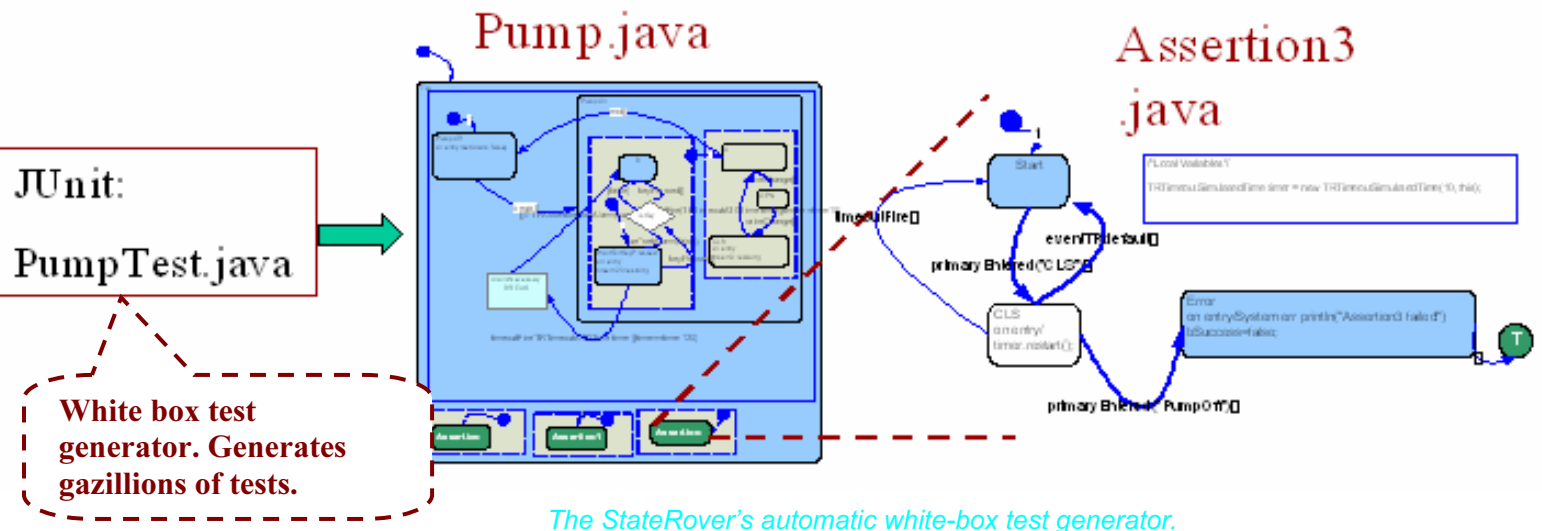
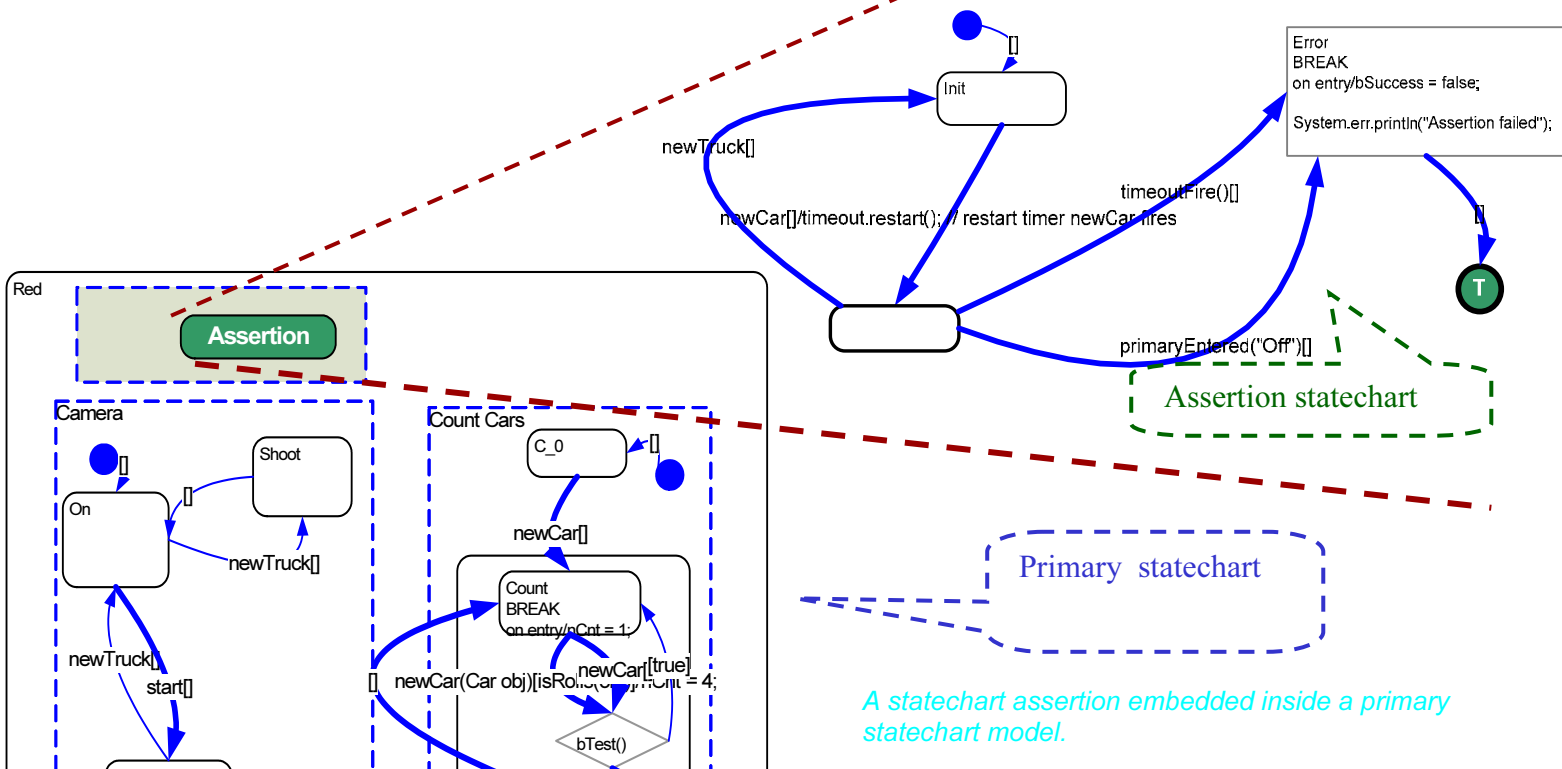
Once a software class or module has been implemented and integrated, the StateRover can be used as a graphical debugging tool, including:

- Visual debug animation. While executing StateRover generated code on a host or target platform, the diagram is automatically animated to reflect program execution. The active states, previously visited states, and path from previous states to active states are all animated.



## Run-time monitoring and model-checking of behavioral requirements using assertion statecharts

- ❑ Assertion statecharts are visual behavioral specifications. The StateRover code generator and visual debug animator supports code generation for *primary* and *assertion* statecharts. Please read the book “*Modeling and Verification Using UML Statecharts*” by D. Drusinsky.
- ❑ The StateRover’s non-deterministic statechart code generation supports non-deterministic assertion-statechart behavioral specifications.
- ❑ The MCRover is an automatic white box test generator based on JUnit. The MCRover used the primary model and all associated assertion statecharts as the white-box.
- ❑ The MCRover combined with automatic assertion checking constitutes automatic model checking.



Contact, pricing and ordering information:

**Time Rover, Inc.**

Tel: 408-252-2808

Fax: 408-777-8615

URL: [www.time-rover.com](http://www.time-rover.com)

e-mail: available on <http://www.time-rover.com/contacting.html>