NAME

state − CHSM state class

SYNOPSIS

```
#define CHSM_STATE_ARGS /* ... */
#define CHSM_STATE_INIT /* ... */

namespace Concurrent_Hierarchical_State_Machine {

    class state {
    public:
        state( CHSM_STATE_ARGS );

        bool            active() const;
        virtual void    deep_clear();
        char const*     name() const;
        parent*         parent_of() const;
    protected:
        virtual bool    enter( event const &trigger, state* = 0 );
        virtual bool    exit ( event const &trigger, state* = 0 );
    };

}
```

DESCRIPTION

A `state` is the class used to represent an atomic state in a CHSM. It is also the base class for **CHSM::parent**(3).

Member Functions

`bool active() const`
: Returns `true` only if the state is currently active.

`virtual void deep_clear()`
: Recursively clears the deep history of all child clusters.

`virtual bool enter( event const &trigger, state* = 0 )`

`virtual bool exit ( event const &trigger, state* = 0 )`
: Returns `true` only if the state was actually entered or exited, respectively. The `trigger` is a reference to the event that is causing the state to be entered or exited. The remaining argument is for use of the implementation only and subject to change. These functions can be overridden in a derived class to alter the behavior of states.

`char const* name() const`
: Returns the fully-qualified name of the state.

`parent* parent_of() const`
: Returns a pointer to the parent state, if any; null otherwise.

Derived Classes

The `state` class can be derived from either to add additional data members and members functions to a state or to alter its behavior upon being entered or exited. The macros `CHSM_STATE_ARGS` and `CHSM_STATE_INIT` are used in the derived class's constructor and shield the user from the ugly arguments lists used in the CHSM implementation.

EXAMPLE

```
#include <chsm.h>

class place : public CHSM::state {
    //
```

```
    // This class is meant to simulate the behavior of a "place"
    // in Petri Net terminology.  Places have zero or more tokens
    // at them.
    //
public:
    place( CHSM_STATE_ARGS ) :
        CHSM::state( CHSM_STATE_INIT ), tokens_( -1 ) { }
    //
    // Conversion to int: Allows state name to be used in
    // arithmetic expressions.
    //
    operator int() const { return tokens_; }

    //
    // Operator to allow the number of tokens to be decrememented.
    //
    int operator-=( int used ) { return tokens_ -= used + 1; }

protected:
    bool enter( event const &trigger, state *s = 0 ) {
        //
        // Call base-class enter() function as required and
        // proceed only if it returns true.
        //
        if ( !CHSM::state::enter( trigger, s ) )
            return false;
        //
        // Increment the number of tokens for every entrance.
        //
        ++tokens_;

        return true;
    }
private:
    int tokens_;
};

int water_molecules = 0;

%%
chsm make_water is {
    set H2O( H2, O2 ) {
        //
        // The conditions might seem to be off by one for
        // hydrogen on the transition of hydrogen and similarly
        // for oxygen.  You have to remember though that the
        // transition being triggered itself is either a hydrogen
        // or oxygen so we have to take that into account.
        //
        hydrogen[ ${H2} >= 1 && ${O2} >= 1 ] -> H2O %{
            ${H2} -= 1, ${O2} -= 1;
            water_molecules += 2;
        %}
        oxygen[ ${H2} >= 2 ] -> H2O %{
```

```
                ${H2} -= 2;
                water_molecules += 2;
            %}
        } is {
            state<place> H2 { hydrogen -> H2; }
            state<place> O2 { oxygen -> O2; }
        }
    }
```

SEE ALSO

CHSM::cluster(3), **CHSM::parent**(3), **CHSM::set**(3), **chsm-c++**(4)

Tadao Murata. ‘‘Petri Nets: Properties, Analysis and Applications.’’ *Proceedings of the IEEE*, 99(4), April 1989. pp. 541-580.

AUTHORS

Paul J. Lucas <*paul@lucasmail.org*>
Fabio Riccardi <*fabio.riccardi@mac.com*>