NAME

event − CHSM event class

SYNOPSIS

```
namespace Concurrent_Hierarchical_State_Machine {

    class event {
    public:
        void operator()();
        template<typename EventClass> bool is_type() const;
        char const* name();
    protected:
        struct param_block {
            param_block( event const& );
            virtual ~param_block();
        };
    };

    class user_event : public event {
    public:
        struct param_block : event::param_block {
            parameter declarations
            param_block( event const& );
            virtual ~param_block();
        };
        void operator()( parameter declarations );
        param_block* operator->();

        // inherited
        char const* name();
    };

    bool operator==( event const&, event const& );
    bool operator!=( event const&, event const& );
}
```

DESCRIPTION

Instances of the `event` class, or classes derived therefrom, cause transitions to occur in a CHSM.

event

The base event class. It is the base class for all user-specified events; it is also used directly for `enter()` and `exit()` events.

Member Functions

`void operator()()`
: Broadcasts the event. (See *user*`_event::operator()()` below for more information.)

`template<typename EventClass> bool is_type()`
: Returns `true` only if the event is of the given event class.

`char const* name()`
: Returns the name of the event.

*user*_event

A ***user_event*** is a class generated by the **chsmc**(1) compiler for user-specified events. For example, a user-specified event `alpha` has a class named `alpha_event` generated for it.

User events may have parameters. The `param_block` data members are the parameters of the event plus all the parameters of all base events, if any, via inheritance, taken from event declarations in a CHSM

description by the **chsmc**(1) compiler. For derived events, parameters are inherited in base-to-derived order.

`void operator()(` *parameter declarations* `)`
: Broadcasts the event with the specified parameters, if any.

`param_block* operator->()`
: Returns a reference to the parameter named on the right-hand-side.

GLOBAL FUNCTIONS

`bool operator==( event const&, event const& )`

`bool operator!=( event const&, event const& )`
: Returns `true` only if the two given events are equal, or not equal, respectively. Because there is a single instance of each event per CHSM, these functions test *identity* rather than equality; hence two events of the same name from different instances of a CHSM will not compare equal.

SEMANTICS

Broadcasting

Broadcasting an event that is already ‘‘in progress’’ does nothing; the event is not rebroadcast. Broadcasting a base event of an event already ‘‘in progress’’ does nothing.

Preconditions

A precondition for an event is the logical-and of all base event preconditions, if any, evaluated in base-to-derived order. Evaluation is ‘‘short-circuited’’ via the traditional semantics of the C++ && operator. If a precondition is not satisfied, the event is not broadcast.

Finding Transitions

After an event has satisfied its precondition, all transitions on the given event out of currently-active states have their conditions, if any, evaluated. For derived events, transitions are inherited in derived-to-base order so that transitions on derived events will dominate those on base events.

EXAMPLE

```
#include <iostream>
using namespace std;

%%
chsm my_machine is {
    event alpha;
    event<alpha> beta( int n );
    event<beta> gamma( char const *message );

    state x {
        alpha -> y %{
            alpha();                    // in progress -- does nothing
        %};
    }
    state y {
        beta -> z %{
            alpha();                    // does nothing since beta is-an alpha
            cout << beta->n << endl;    // access parameter
        %};
    }
    state z {
        gamma -> x %{
            cout << gamma->n << endl;   // inherited parameter
            cout << gamma->message << endl;
        %};
    }
```

```
}

%%
int main() {
    my_machine m;
    m.alpha();                          // broadcast alpha

    my_machine::alpha_event *e;         // example of generated event class
    e = &m.beta;                        // legal since beta is-an alpha

    if ( my_machine::beta_event *b = dynamic_cast<machine::beta_event*>(e) )
        (*b)( 42 );                     // broadcast beta with parameter

    m.gamma( 42, "hello, world" );      // inherited parameters
}
```

SEE ALSO

chsmc(1), **CHSM::state**(3), **chsm-c++**(4)

AUTHORS

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