C++Now 2016

C++14 META STATE MACHINE LIBRARY

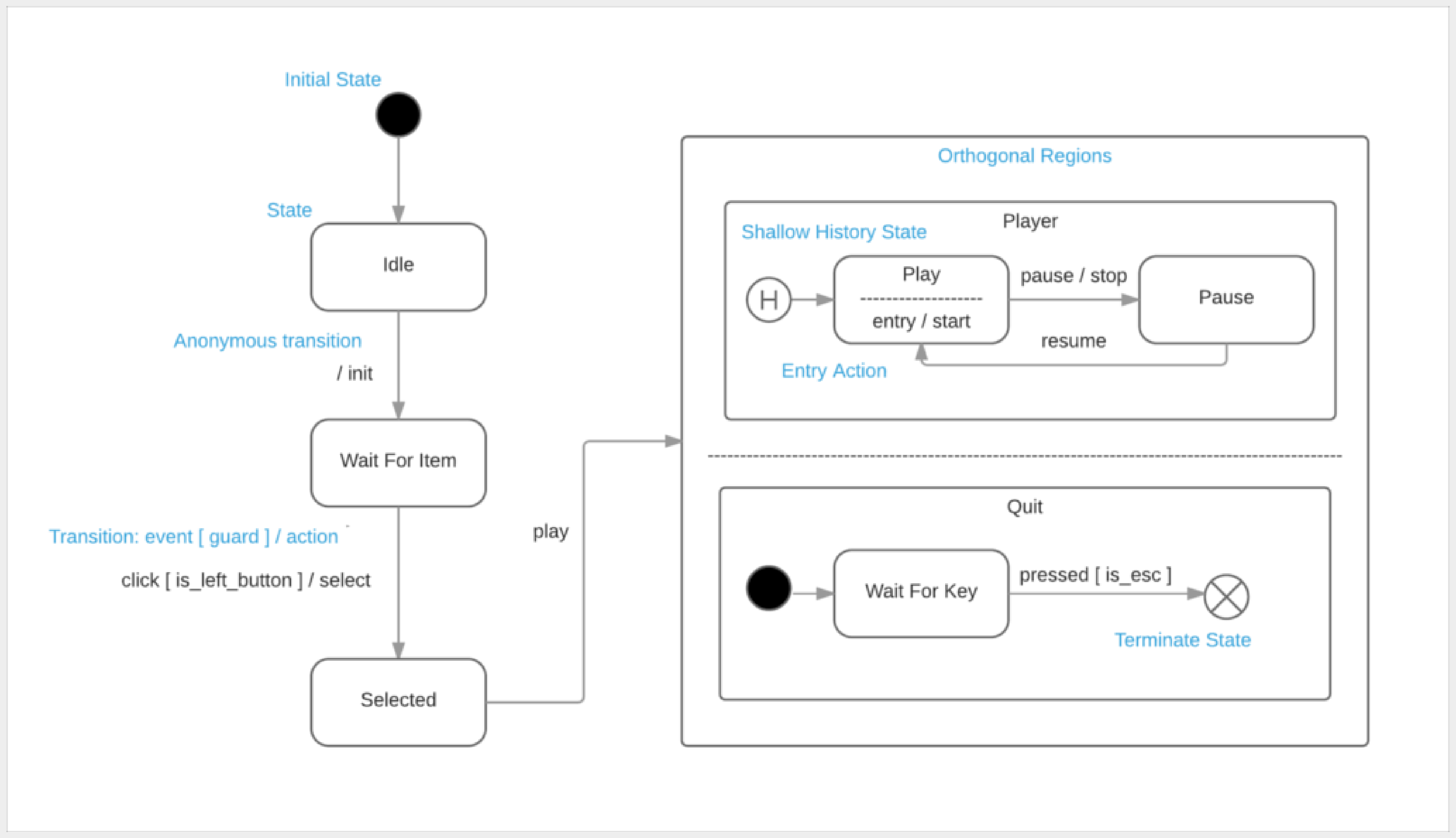
Kris Jusiak

UML STATE MACHINE

UML State Machine (SM) depict the dynamic behavior of an entity based on its response to events, showing how the entity reacts to various events depending on the current state that it is in

*Formerly called **UML state chart** in UML 1*

STATE MACHINE CONCEPTS



UML Concept	Description
Event	Trigger which affects the system, ex. `button click`
State	Captures aspect of relavant system history, ex. `Playing`
Transition	Conditionally update current state, ex. `src_state + event [ guard ] / action -> dst_state`
Orthogonal Region	Sudo parallel state machines, ex. `Play and Quit`

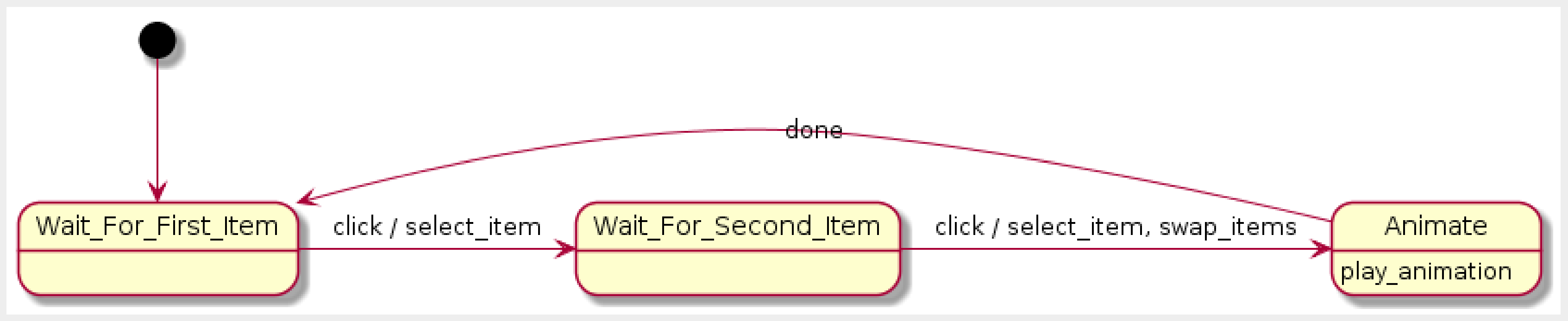
DO I NEED A STATE MACHINE?

NO, BUT...

- SM promotes better design
    - Compliant with UML documentation
- SM creates easier to maintain code
    - Declarative approach / Expresses `WHAT`, not `HOW`!
- SM creates easy to test code
    - Can be tested in isolation without trying to reach all conditional branches

EXAMPLE

SWAP ITEMS


done
Wait_For_First_Item
click / select_item
Wait_For_Second_Item
click / select_item, swap_items
Animate
play_animation

NO STATE MACHINE

```
void handle(auto event) {
  if (event == click && !isAnimating) {
    select_item();
    ++itemsSelected;

    if (itemsSelected == 2) {
      swap_items();
      animate();
      isAnimating = true;
    }
  } else if(event == done) {
    isAnimating = false;
    itemsSelected = 0;
  }
}
```

STATE MACHINE (PSEUDO-CODE)

```
transition_table {
  *Wait_For_First_Item + click / select_item -> Wait_For_Second_Item,
   Wait_For_Second_Item + click / select_item, swap_items -> Animate,
   Animate + on_entry / play_animation(),
   Animate + done -> Wait_For_First_Item
};
```

MSM-LITE VS BOOST.MSM-EUML VS BOOST.STATECHART

OVERVIEW

Library	MSM-lite	Boost.MSM-eUML	Boost.Statechart
Standard	C++14	C++98/03	C++98/03
Version	1.0.1	1.61	1.61
License	Boost 1.0	Boost 1.0	Boost 1.0
Linkage	header only	header only	header only

IMPLEMENTATION DETAILS

Library	MSM-lite	Boost.MSM-eUML	Boost.Statechart
UML	2.0	2.0	1.5
RTTI	-	-	✓
Exceptions	-	-	✓
Memory Allocations	-	-	✓

FEATURES

UML FEATURES

Library	MSM-lite	Boost.MSM-eUML	Boost.Statechart
Transition	✓	✓	✓
Anonymous transition	✓	✓	✓
Internal transition	✓	✓	✓
Local transitions	-	-	-

Library	MSM-lite	Boost.MSM-eUML	Boost.Statechart
State entry/exit	✓	✓	✓
Guard	✓	✓	✓
Action	✓	✓	✓
Event defering	~	✓	✓
Error handling	✓	✓	✓

Library	MSM-lite	Boost.MSM-eUML	Boost.Statechart
Initial state	✓	✓	✓
Terminate State	✓	✓	✓
Explicit entry	✓	✓	✓
Explicit exit	-	✓	✓
Fork	-	✓	-

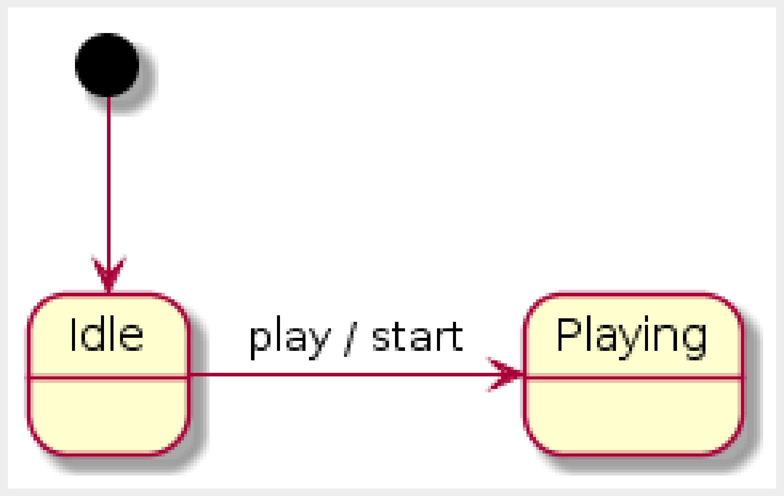
Library	MSM-lite	Boost.MSM-eUML	Boost.Statechart
Orthogonal regions	✓	✓	✓
Sub / Composite	✓	✓	✓
Shallow History	✓	✓	✓
Deep History	~	~	✓

NON-UML FEATURES

Library	MSM-lite	Boost.MSM-eUML	Boost.Statechart
Any event	-	✓	-
Flags	-	✓	-
Interrupt state	-	✓	-
State Visitor	✓	✓	✓
Serialization	-	✓	-
Dispatcher	✓	-	-
Asynchronous SM	-	-	✓

EXAMPLE

SIMPLE STATE MACHINE



BOOST.MSM-EUML

EVENTS

```
BOOST_MSM_EUML_EVENT(play)
```

STATES

```
BOOST_MSM_EUML_STATE((), Idle)
BOOST_MSM_EUML_STATE((), Playing)
```

ACTIONS

```
BOOST_MSM_EUML_ACTION(start) {
template <class FSM, class EVT, class SourceState, class TargetState>
  void operator()(EVT const &, FSM &, SourceState &, TargetState &) {
    // ...
  }
};
```

TRANSITION TABLE

```
BOOST_MSM_EUML_TRANSITION_TABLE((
  Playing == Idle + play / start
), transition_table)
```

STATE MACHINE

```
BOOST_MSM_EUML_ACTION(Log_No_Transition){
  template <class FSM, class Event>
  void operator()(Event const &, FSM &, int state) {
      // ...
  }
};
```

```
BOOST_MSM_EUML_DECLARE_STATE_MACHINE((
  transition_table,
  init_ << Idle,
  no_action, // entry
  no_action, // exit
  attributes_ << no_attributes_,
  configure_ << no_exception << no_msg_queue,
  Log_No_Transition
), player_);

using player = msm::back::state_machine<player_>;
```

PROCESS EVENT

```
player sm;
sm.process_event(play);
```

BOOST.STATECHART

EVENTS

```
struct play : sc::event<play> { };
```

ACTIONS

```
struct player : sc::state_machine<player, Idle> {
  void start(play const &) {
    // ...
  }
}
```

STATES

```cpp
struct Idle : sc::simple_state<Idle, player> {
  using reactions = mpl::list<
    sc::transition<play, Playing, player, &player::start>
  >;
};

struct Playing : sc::simple_state<Playing, player> {
  using reactions = mpl::list<>;
};
```

PROCESS EVENT

```
player sm;
sm.process_event(play());
```

EXPERIMENTAL BOOST.MSM-LITE

EVENTS

```
struct play { };
```

STATE MACHINE

```
struct player {
  auto configure() {
    return make_transition_table(
      "Idle"_s + event<play> / []{ ... } = "Playing"_s
    )
  )
);
```

PROCESS EVENT

```
msm::lite::sm<player> sm;
sm.process_event(play{});
```

BENCHMARKS

ENVIRONMENT

2.3 GHZ INTEL CORE I7 / 16 GB 1600 MHZ DDR3

```
CXXFLAGS: -O2 -s
```

SIMPLE TEST

Events	States	Transitions	Process Events
6	5	12	1'000'000

Clang-3.7	**MSM-lite**	**Boost.MSM-eUML**	**Boost.Statechart**
Compilation time	0.144s	3.855s	1.028s
Execution time	15ms	17ms	1232ms
Memory usage	14b	32b	200b
Executable size	11K	91K	59K

GCC-5.2	**MSM-lite**	**Boost.MSM-eUML**	**Boost.Statechart**
Compilation time	0.175s	7.879s	1.790s
Execution time	15ms	19ms	929ms
Memory usage	14b	32b	224b
Executable size	11K	67K	63K

COMPOSITE TEST

Events	States	Transitions	Process Events
8	5 + 3	12 + 4	1'000 * 1'000

Clang-3.7	MSM-lite	Boost.MSM-eUML	Boost.Statechart
Compilation time	0.184s	4.526s	1.293s
Execution time	10ms	14ms	491ms
Memory usage	20b	60b	200b
Executable size	15K	111K	83K

GCC-5.2	MSM-lite	Boost.MSM-eUML	Boost.Statechart
Compilation time	0.248s	9.363s	2.037s
Execution time	9ms	13ms	404ms
Memory usage	20b	60b	224b
Executable size	12K	91K	83K

COMPLEX TEST

Events	States	Transitions	Process Events
50	50	50	1'000'000

Clang-3.7	MSM-lite	Boost.MSM-eUML	Boost.Statechart
Compilation time	0.582s	1m15.935s	3.661s
Execution time	69ms	81ms	6221ms
Memory usage	102b	120b	200b
Executable size	35K	611K	343K

GCC-5.2	MSM-lite	Boost.MSM-eUML	Boost.Statechart
Compilation time	0.816s	52.238s	4.997s
Execution time	72ms	77ms	5520ms
Memory usage	102b	120b	224b
Executable size	35K	271K	215K

MORE BENCHMARKS

https://github.com/boost-experimental/msm-lite/benchmarks

EXPERIMENTAL BOOST.MSM-LITE

MOTIVATION

BOOST.MSM-EUML IS AWESOME, BUT...

PROBLEMS WITH BOOST.MSM-EUML

LONG COMPILATION TIMES

HUGE BINARIES

LONG ERROR MESSAGES

BASED ON MACROS

FUNCTIONAL PROGRAMMING EMULATION (C++03)

OVERVIEW

A BIT OF HISTORY

2013

Version based on Boost.MSM-eUML

---

https://github.com/krzysztof-jusiak/msm

- Compiled slowly
- Long error messages
- Based on macros (inherited from Boost.MSM-eUML)

2016

Version C++14 - v1.0.1

https://github.com/boost-experimental/msm-lite

- One header (boost/msm-lite.hpp)
- 1.5k lines
- Neither Boost nor STL is required
- No 'virtual' methods
- No 'exceptions' (-fno-exceptions)

TESTED COMPILERS

- Clang-3.4+
- GCC-5.2+

DESIGN

GOALS

KEEP THE BOOST.MSM-EUML 'GOODIES'

DESIGN (FRONT/BACK-END)

MAX PERFORMANCE

LOW MEMORY USAGE

EUML DSL

```
src_state + event [ guard ] / action -> dst_state
```

UML STANDARD COMPLIANT (AS MUCH AS POSSIBLE)

ELIMINATE BOOST.MSM-EUML PROBLEMS

SPEED UP COMPILATION TIMES

- UP TO 60X FASTER

REDUCE BINARY SIZE

- 3X SMALLER

BETTER ERROR MESSAGES

- CONCEPTS EMULATION / NO `MPL`

LESS BOILERPLATE

- NO MACROS

TAKE ADVANTAGE OF MODERN C++ FUNCTIONAL CAPABILITIES

- LAMBDAS AS GUARDS AND ACTIONS

ARCHITECTURE

COMPILE-TIME + RUN-TIME

Generate jump table at compile-time and operates on it at run-time

FRONT-END

RESPONSIBLE FOR PROVIDING AN UNIFIED LIST OF TRANSITIONS

DSL

Domain Specific language

```
"src_state"_s + event [ guard ] / action = "dst_state"_s
```

BACK-END

RESPONSIBLE FOR OPERATING ON TRANSITIONS

USER GUIDE

EVENTS

EVENT DECLARATION

```
class my_event { ... };
```

EVENT DECLARATION ON THE FLY

```
using namespace msm;
auto event  = "event"_t;
```

It's not standard!

STATES

STATE DECLARATION

```
auto idle = msm::state<class idle>{ };
```

STATE DECLARATION ON THE FLY

```
using namespace msm;
auto state  = "idle"_s;
```

It's not standard!

STATE TYPES

State	Description	Notation
Initial state	Tells SM where to start	`*"state"_s`
Sub-SM	State which is a SM itself	`sm<...>`
History state	Remember Sub-SM state when reentered	`"state"_s(H)`
Terminate state	Indicates SM termination	`X`

GUARDS

GUARD IS FUNCTOR AND MUST RETURN BOOLEAN VALUE

```
auto guard_always_success = [] {
  return true;
};
```

```
auto guard_with_dependencies = [](int, double) {
  return true;
};
```

```
auto guard_with_dependencies_and_event = [](int, auto event, double) {
  return true;
};
```

ACTIONS

ACTION IS A FUNCTOR AND MUST NOT RETURN

```
auto action_empty = [] { };
```

```
auto action_with_dependencies = [](int, double) { };
```

```
auto action_with_dependencies_and_events = [](int, auto event, double) {
```

TRANSITION TABLE

TRANSITION TABLE DSL (DOMAIN SPECIFIC LANGUAGE)

Expression	Description
state + event [guard]	internal transition on event e when guard
src_state / [] {} = dst_state	anonymous transition with action
src_state + event = dst_state	transition on event e without guard or action

Expression	Description
src_state + event [guard] / action = dst_state	transition from src_state to dst_state on event e with guard and action
src_state + event [guard && (![]{return true;} && guard2)] / (action, action2, []{}) = dst_state	transition from src_state to dst_state on event e with guard and action

MAKE TRANSITION TABLE

```
using namespace msm;

make_transition_table(
  "src_state"_s + event<my_event> [guard] / action = "dst_state"_s
, "dst_state"_s + event<game_over> = X // 'X' - terminate state
);
```

ORTHOGONAL REGIONS

DEFINE ORTHOGONAL REGIONS BY MARKING MULTIPLE STATES AS INITIAL

```
using namespace msm;

make_transition_table(
 *"region_1"_s   + event<my_event1> [guard]/action = "dst_state1"_s
, "dst_state1"_s + event<game_over> = X,
//---------------------------------------------------------------//
 *"region_2"_s   + event<my_event2> [guard]/action = "dst_state2"_s
, "dst_state2"_s + event<game_over> = X
);
```

STATE MACHINE

DEFINE STATE MACHINE

```
class example {
public:
  auto configure() noexcept {
    using namespace msm;

    return make_transition_table(
     *"src"_s + event<my_event> [ guard ] / action = "dst"_s,
      "dst"_s + event<game_over> = X
    );
  }
};
```

DECLARE STATE MACHINE

```
msm::sm<example> sm;
```

STATE MACHINE DEPENDENCIES

Required for guards and actions

SM DEPENDENCIES

```
                            /-- event (injected from process_event)
                            |
auto guard = [](double d, auto event) { return true; }
                   |
                   \-------\
                           |
auto action = [](int i){}  |
                 |         |
                 |         |
                 \-\   /---/
                   |   |
msm::sm<exmple> s{42, 87.0};

// order in which parameters have to passed is not specificied
msm::sm<exmple> s{87.0, 42};
```

ALTERNATIVELY YOU CAN USE EXPERIMENTAL BOOST.DI

https://raw.githubusercontent.com/boost-experimental/di

CREATE SM USING DI FRAMEWORK

```
auto injector = di::make_injector<example>();
```

PROCESS EVENTS

PROCESS EVENT BY STATE MACHINE

```
sm.process_event(my_event{});
```

DISPATCH EVENTS

Useful for run-time events

DISPATCH SDL2 EVENTS

```
struct game_over {
  static constexpr auto id = SDL_QUIT;
  // explicit game_over(const SDL_Event&) noexcept;
};

auto dispatch_event =
  msm::make_dispatch_table<SDL_Event
                         , SDL_FIRSTEVENT
                         , SDL_LASTEVENT>(sm);

SDL_Event event{SDL_QUIT};

// will call sm.process(game_over{});
dispatch_event(event, event.type);
```

HANDLE ERRORS

UNEXPECTED EVENT

HANDLE UNEXPECTED EVENT

```
make_transition_table(
 *"src"_s + event<my_event> [ guard ] / action = "dst"_s
, "src"_s + unexpected_event<some_event> = X
);
```

```
sm.process_event(some_event{}); // SM will enter terminate (X) state
```

EXCEPTIONS

Only if NOT compiled with `-fno-exceptions`

EXCEPTION HANDLING

```
make_transition_table(
 *"state"_s + event<event> / []{throw std::runtime_error{"error"};}
,*"state"_s + exception<std::runtime_error> = X
, "state"_s + exception<std::logic_error> = X
, "state"_s + exception<> / [] { cleanup...; } = X // catch(...)
);
```

TESTING UTILITIES

SHOW CURRENT STATES

```
sm.visit_current_states([](auto state) {
  std::cout << state.c_str() << std::endl;
});
```

VERIFY CURRENT STATE

```
assert(sm.is("idle"_s));
```

VERIFY CURRENT STATES (ORTHOGONAL REGIONS)

```
assert(sm.is("region1"_s, "region2"_s, ...));
```

ADDITIONAL READINGS

http://boost-experimental.github.io/msm-lite/user_guide

http://boost-experimental.github.io/msm-lite/tutorial

IMPLEMENTATION

TRANSITION

Glue between Front-end and Back-end

TRANSITION

```
template <class TSrcState
        , class TDstState
        , class TEvent = anonymous
        , class TGuard = always
        , class TAction = none>
struct transition {
  template <class SM>
  auto execute(SM &, TEvent &&);

  TGuard guard;
  TAction action;
};
```

FRONT-END

DSL (DOMAIN SPECIFIC LANGUAGE)

```
src_state + event [ guard ] / action -> dst_state
```

STATE

STATE

```
template <class TState>
struct state {
  template <class T>
  auto operator=(const T &t) const { return transition<T, state>{ }; }

  template <class T>
  auto operator<=(const T &t) const { return transition<TState, T>{ }; }

  template <class T>
  auto operator+(const T &t) const { return transition<TState, T>{ }; }

  template <class T> requires callable<bool, T>()
  auto operator[](const T &t) const { return transition<TState, T>{ }; }

  template <class T> requires callable<void, T>()
  auto operator/(const T &t) const { return transition<TState, T>{ }; }
};
```

EVENT

EVENT

```
template <class>
struct event {
  template <class T> requires callable<bool, T>()
  auto operator[](const T &t) const { return transition<event, T>{ }; }

  template <class T> requires callable<void, T>()
  auto operator/(const T &t) const { return transition<event, T>{ }; }
};
```

GUARD

OPERATORS

```
and_, or_, not_
```

BASE

```
struct operator_base { };
```

AND

```
template <class... Ts>
struct and_ : operator_base {
  template <class TEvent, class TDeps>
  auto operator()(const TEvent &event, TDeps &deps) {
    return (call(std::get<Ns-1>(args), event, deps) && ...);
  }

  tuple<Ts...> args;
};
```

WHERE

CALL / IS OPERATOR BASE

```
template<class T, class TEvent, class TDeps>
  requires std::is_base_of_v<operator_base, T>
auto call(T op, const TEvent&, TDeps& deps) {
  return op(event, deps);
}
```

CALL / FUNCTOR TYPE

```
template<class T, class TEvent, class TDeps>
  requires !std::is_base_of_v<operator_base, T>
auto call(T op, const TEvent& event, TDeps& deps) {
  return call_impl<decltype(&T::operator())>::execute(op, event, deps);
}
```

```
template <class R, class... TArgs>
struct call_impl<R(TArgs...)> {
  template<class T, class TEvent, class TDeps>
  static auto execute(T op, const TEvent& event, TDeps& deps) {
    return op(std::get<TArgs>(tuple_cat(event, deps))...);
  }
};
```

SIMILARY FOR `OR_` AND `NOT_`

USER-DEFINED OPERATORS

```
namespace msm
```

NOT

```
template <class T> requires callable<bool, T>()
auto operator!(T&& t) {
  return not_<T>(std::forward<T>(t));
}
```

AND

```
template <class T1, class T2>
  requires callable<bool, T1>() && callable<bool, T2>()
auto operator&&(T1&& t1, T2&& t2)  {
  return and_<T1, T2>(std::forward<T1>(t1), std::forward<T2>(t2));
}
```

OR

```
template <class T1, class T2>
  requires callable<bool, T1>() && callable<bool, T2>()
auto operator||(T1&& t1, T2&& t2) {
  return or_<T1, T2>(std::forward<T1>(t1), std::forward<T2>(t2));
}
```

EXAMPLE

GUARDS

```
auto true_ = [] { return true; };
auto false_ = [] { return false_; };

auto guards = (!true_ || (true_ && false_ || ![]{ return false; }));
```

TYPE

```
static_assert(std::is_same<decltype(guards)
            , not_<or_<and_<false_, not_<[]{return false;}>>>>>{});
```

ACTION

USER-DEFINED OPERATORS

```
namespace msm
```

SEQUENCE

```
template <class T1, class T2>
  requires callable<void, T1>() && callable<void, T2>()
auto operator,(T1&& t1, T2&& t2) {
  return seq_<T1, T2>(std::forward<T1>(t1), std::forward<T2>(t2));
}
```

EXAMPLE

ACTIONS

```
auto action = [] { };

auto actions = (action, action, []{}, action);
```

TYPE

```
static_assert(std::is_same<decltype(actions),
              seq_<action, action, []{}, action>>{});
```

BRINING IT ALL TOGETHER

MAKE TRANSITION TABLE

```
template <class... Ts> requires transitional<Ts>()...
auto make_transition_table(Ts&&... ts) {
  return tuple<Ts...>{std::forward<Ts>(ts)...};
}
```

WHERE

TRANSITIONAL CONCEPT

```
template <class T>
concept bool transitional() {
  return requires(T transition) {
    typename T::src_state;
    typename T::dst_state;
    typename T::event;
    typename T::deps;
    T::initial;
    T::history;
    { transition.execute(...) } -> bool;
  }
}
```

EXAMPLE

FRONT-END

```
auto play_song = [] { ... };
auto is_playing =  [] { return true; }

make_transition_table(
  state<class Idle> + play [is_playing] = state<class Wait>
, state<class Idle> + play [!is_playing] / play_song = state<class Play>
, state<class Play> + stop / (stop_song, reset_song) = state<class Idle>
);
```

TYPE

```
tuple<
  transition<Idle, Wait, Play, is_playing>
, transition<Idle, Wait, Play, not_<is_playing>, play_song>
, transition<Play, Idle, Stop, always, seq_<stop_song
                                          , reset_song>, play_song>
>
```

DSL LIMITATIONS

DSL

```
"idle"_s + play [ [] { return true; } ] / action
                ^ ^
                \ /
                 |
/----------------/
\
 Compilation error: `C++11 generalized attributes syntax: [[`
```

SOLUTION - PARENS

```
"idle"_s + play [ ([] { return true; }) ] / action
                  ^                   ^
                  \                   /
                   -------------------
```

BACK-END

OPERATES ON TRANSITIONS

GOALS

- MAX RUN-TIME PERFORMANCE
- QUICK COMPILATION TIMES

HOW?

STATE MACHINE

FRONT-END

```
tuple<
  transition<Idle, Wait, Play, is_playing>
, transition<Idle, Wait, Play, not_<is_playing>, play_song>
, transition<Play, Idle, Stop, always, seq_<stop_song
                                           , reset_song>, play_song>
>
```

CONVERSION TO BACK-END (STATE MACHINE)

```
template<class>
struct sm;

template<class... Transitions>
struct sm<tuple<Transitions...>> : core::sm<Transitions...>
{ };
```

WHERE

STATE MACHINE

```cpp
template <class... Ts> requires transitional<Ts>()...
class sm { // namespace core
  using states = unique_t<get_state<Ts...>>; // unique list of states
  using events = unique_t<get_event<Ts...>>; // unique list of events
  using deps = unique_t<get_deps<Ts...>>; // unique list of dependencies
  using mappings = mappings_t<Ts...>; // preprocessed event mapping

public:
  template <class... TDeps> requires std::is_base_of_v<TDeps, deps>...
  explicit sm(TDeps &&... deps); // init deps

  template <class TEvent> void process_event(TEvent&&);

private:
  int current_state_ = 0;
  tuple<deps> deps_;
};
```

DETAILS

UNIQUE_T

UTILITIES

```
template <class...> struct type { };
template <class...> struct type_list { using type = type_list; };
template <class... Ts> struct inherit : Ts... { using type = inherit; };
```

UNIQUE_T

```
template <class...> struct unique;

template <class... Rs, class T, class... Ts>
struct unique<type<Rs...>, T, Ts...> : std::conditional_t<
  std::is_base_of<type<T>, inherit<type<Rs>...>>{},
  unique<type<Rs...>, Ts...>,
  unique<type<Rs..., T>, Ts...>
> { };

template <class... Rs> struct unique<type<Rs...>>
  : type_list<Rs...>
{ };

template <class... Ts> using unique_t =
  typename unique<type<>, Ts...>::type;
```

BOOST.HANA VS BOOST.MPL VS UNIQUE_T

Boost-1.61

BENCHMARK

```
using u = decltype(
  hana::unique(hana::sort(hana::make_tuple(event1, event2, event1)))
);
static_assert(make_tuple(event1, event2) == u);
```

```
using u = typename mpl::unique<
  typename mpl::sort<
    mpl::vector<event1, event2, event1>
  , boost::is_same<mpl::_1, mpl::_2>
  >::type;
>;
static_assert(std::is_same<u, mpl::vector<event1, event2>>{});
```

```
using u = unique_t<event1, event2, event1>;
static_assert(std::is_same<u, type_list<event1, event2>>{});
```

Number of elements	unique_t	Boost.Hana	Boost.MPL
16	0.067s	0.620s	1.194s
64	0.078s	1.781s	2.482s
128	0.085s	7.821s	10.409s

EXAMPLE

STATE MACHINE

```
auto is_playing = [](const player& p, auto event) {
  return p.is_playing(event.song);
};
auto play_song = [](recorder& r) { r.play(); };

sm<
  transition<Idle, Wait, Play, is_playing>
, transition<Idle, Wait, Play, not_<is_playing>, play_song>
, transition<Play, Idle, Stop, always, seq_<stop_song
                                          , reset_song>, play_song>
>
```

INTERNAL TYPES

```
states = type_list<Idle, Wait, Play>;
events = type_list<Play, Stop>;
deps = tuple_list<const player&, recorder&>;
```

MAPPINGS_T

APPROACH

PRE-PROCESSED TRANSITIONS

AVOID TEMPLATE 'MAGIC' PER EVENT!

GOAL

MAPPINGS

```
mappings = {
  Play = {
    Idle = {
      transition<Idle, Wait, Play, is_playing>
    , transition<Idle, Wait, Play, not_<is_playing>, play_song>
    }
  },
  Stop = {
    Play = {
      transition<Play, Idle, Stop, always, seq_<stop_song
                                              , reset_song>, play_song>
    };
  }
};
```

HOW?

ALGORITHM

[Mappings]

- For each unique event
  - Find transitions with that event
  - For each found transition create a pair of an event and [StateTransitions]

[StateTransitions]

- For each unique state
  - Create a pair of a state and transitions from that state and given event

DETAILS

MAP

MAP

```
template <class, class> struct pair { };

template <class... Ts> struct map : Ts... { };

template <class T> struct no_decay { using type = T; };
template <class TDefault, class> static no_decay<TDefault> lookup(...);

template <class, class TKey, class TValue>
static no_decay<TValue> lookup(pair<TKey, TValue> *);

template <class TDefault, class TKey, class T>
using at_key = decltype(lookup<TDefault, TKey>((T *)0));

template <class T, class TKey, class TDefault = void>
using at_key_t = typename at_key<TDefault, TKey, T>::type;
```

Used in some modern C++ meta-programming libraries

BOOST.HANA VS BOOST.MPL VS MAP

Boost-1.61

BENCHMARK

```
constexpr auto m =
  hana::make_map(hana::make_pair(event, transitions), ...);
static_assert(m[event] == transitions);
```

```
using m =
  mpl::map<mpl::pair<event, transitions>...>;
static_assert(boost::is_same<mpl::at<m, event>::type, transitions>{});
```

```
using m =
  map<pair<event, transitions...>>;
static_assert(boost::is_same<at_key_t<m, event>, transitions>{});
```

Number of elements	map	Boost.Hana	Boost.MPL
16	0.050s	0.435s	1.110s
64	0.069s	0.923s	1.121s
128	0.076s	1.186s	1.453s

EXAMPLE

STATE MACHINE

```
sm<
  transition<Idle, Wait, Play, is_playing>
, transition<Idle, Wait, Play, not_<is_playing>, play_song>
, transition<Play, Idle, Stop, always, seq_<stop_song
                                          , reset_song>, play_song>
>
```

MAPPINGS

```
mappings = map<
  pair<Play, map<pair<Idle, type_list<
    transition<Idle, Wait, Play, is_playing>,
    transition<Idle, Wait, Play, not_<is_playing>, play_song>
  >>>>,
  pair<Stop, map<pair<Play, type_list<
    transition<Play, Idle, Stop, always,
               seq_<stop_song, reset_song>, play_song>
>>>>>;
```

USAGE

TRANSITIONS FOR EVENT 'PLAY' AND STATE 'IDLE'

```
struct empty { };

static_assert(
  std::is_same<
    type_list<
      transition<Idle, Wait, Play, is_playing>,
      transition<Idle, Wait, Play, not_<is_playing>, play_song>
    >,
    at_key_t<at_key_t<mappings, Play, empty>, Idle, empty>
  >{}
);
```

BENCHMARK

Map size	Find transitions for event/state
16	0.002s
64	0.004s
128	0.005s

PROCESS EVENT

GENERATED DISPATCH/JUMP TABLE

JUMP TABLE

```cpp
template<int N> int f() { return N; }

int main(int argc, char**) {
  int (*jump_table[])() = { f<0>, f<1> }; // f<2>, ... f<N>
  return jump_table[argc - 1]();
}
```

Command	Exit code
`./jump_table`	0
`./jump_table 1`	1
`./jump_table 1 2`	Segmentation fault

JUMP TABLE - ASM X86-64

```
main:
  movslq  %edi, %rax
  jmpq    *.L_ZZ4mainE10jump_table-8(,%rax,8)

int f<0>():
  xorl    %eax, %eax
  retq

int f<1>():
  movl    $1, %eax
  retq

.L_ZZ4mainE10jump_table:
  .quad   int f<0>()
  .quad   int f<1>()
```

IDEA

GENERATE TABLE FOR EACH STATE

```
void (*dispatch_table[])(TEvent&&) = {
  HandleTEventInState1
, HandleTEventInState2
, ...
, HandleTEventInStateN
};
```

JUMP USING CURRENT STATE

```
dispatch_table[current_state_](event);
```

DETAILS

PROCESS EVENT

```cpp
template<class TEvent>
void process_event(TEvent&& event) {
  process_event_impl<at_key_t<mappings, TEvent, do_transition<>>>(
    std::forward<TEvent>(event), states{}
  );
}
```

```cpp
template<class TStateMappings, class TEvent, class... TStates>
void process_event_impl(type_list<TStates...>) {
  void (*dispatch_table[])(TEvent&&) = {
    &at_key_t<TStateMappings, TStates, do_transition<>>::template
      execute<TEvent>...
  };
  dispatch_table[current_state_](std::forward<TEvent>(event));
}
```

WHERE

DO TRANSITION

```cpp
template<class... Ts>
struct do_transition {
  template<class TEvent>
  static auto execute(TEvent&& event) {
    return (Ts{}.execute(std::forward<TEvent>(event)) && ...);
  }
};
```

```cpp
template<>
struct do_transition<> {
  template<class TEvent>
  static auto execute(TEvent&&) {
    assert(false && "no transition!");
    return { };
  }
};
```

UPDATE CURRENT STATE

STATE TYPE TO ID (INT)

TYPE TO ID

```cpp
template <std::size_t, class>
struct type_id_type { };

template <class, class...>
struct type_id_impl;

template <std::size_t... Ns, class... Ts>
struct type_id_impl<std::index_sequence<Ns...>, Ts...>
  : type_id_type<Ns, Ts>...
{ };

template <class... Ts>
struct type_id
  : type_id_impl<std::make_index_sequence<sizeof...(Ts)>, Ts...>
{ };
```

GET ID

```
template <class T, int, int N>
constexpr auto get_id_impl(type_id_type<N, T> *) {
  return N;
}
template <class T, int D>
constexpr auto get_id_impl(...) {
  return D;
}
template <class TIds, int D, class T>
constexpr auto get_id()  {
  return get_id_impl<T, D>((TIds *)0);
}
```

TEST

```
using type_ids = type_id<int, double>;
static_assert(-1 == get_id<type_ids, -1, float>());
static_assert(0 == get_id<type_ids, -1, int>());
static_assert(1 == get_id<type_ids, -1, double>());
```

WHERE/WHEN TO UPDATE?

TRANSITION::EXECUTE

TRANSITIONS.EXECUTE

```
template <class TSrcState, class TDstState
        , class TEvent, class TGuard, class TAction>
struct transition {
  template <class TDeps>
  constexpr auto execute(SM & self, const TEvent & event) {
    if (call(guard, event, self.deps_)) { // guards
      self.current_state_ = // set dst state id as the current one
        get_id<type_id<typename SM::states...>, -1, TDstState>();

      call(action, event, self.deps_); // actions
      return true;
    }
    return false;
  }

  TGuard guard; TAction action;
};
```

ALL IN ALL

PROCESS EVENT (PSEUDO-CODE)

```
+ process_event(e)
|
\->  process_event_impl<mappings<event>>(event)
  |
  \-> dispatch_table = {
    |   mappings<event, states>::transition.execute(event)... }
    |
    \-> dispatch_table[current_state](event);
      |
      \-> transition.execute(event)
```

EXAMPLES

GENERATE UML STATE MACHINE DIAGRAM

DUMP TRANSITIONS

```cpp
template <class SM>
void dump(const SM&) noexcept {
  std::cout << "@startuml" << std::endl << std::endl;
  dump_transitions(typename SM::transitions{});
  std::cout << std::endl << "@enduml" << std::endl;
}

template <template <class...> class T, class... Ts>
void dump_transitions(const T<Ts...>&) noexcept {
  (dump_transition<Ts>(), ...);
}
```

DUMP TRANSITION

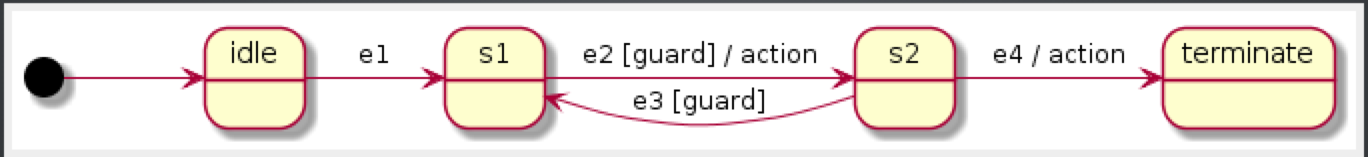
```
template <class T>
void dump_transition() noexcept {
  const auto src_state = msm::state<typename T::src_state>::c_str();
  const auto dst_state = msm::state<typename T::dst_state>::c_str();

  if (T::initial) {
    std::cout << "[*] -> " << src_state << std::endl;
  }

  std::cout << src_state << " -> " << dst_state << " : "
            << typeid(typename T::event) << "["
            << typeid(typename T::guard) << "] / "
            << typeid(typename T::action);
}
```

EXAMPLE - TRANSITION TABLE

```
struct plant_uml {
  auto configure() const noexcept {
    using namespace msm;
    return make_transition_table(
       *"idle"_s + event<e1> = "s1"_s
      , "s1"_s + event<e2> [ guard ] / action = "s2"_s
      , "s2"_s + event<e3> [ guard ] = "s1"_s
      , "s2"_s + event<e4> / action = X
    );
  }
};
```

OUTPUT

```
@startuml
  [*] -> idle
  idle -> s1 : e1
  s1 -> s2 : e2 [guard] / action
  s2 -> s1 : e3 [guard]
  s2 -> terminate : e4 / action
@enduml
```



MORE EXAMPLES

Hello World

Events | States | Actions Guards | Transitions

Orthogonal Regions | Composite | History

Error handling | Logging | Testing

Runtime Dispatcher | eUML Emulation | Dependency Injection

SDL2 Integration | Plant UML Integration

QUESTIONS?

- Documentation
  - http://boost-experimental.github.io/msm-lite
- Source Code
  - https://github.com/boost-experimental/msm-lite
- Try it online
  - http://boost-experimental.github.io/msm-lite/examples

THANK YOU