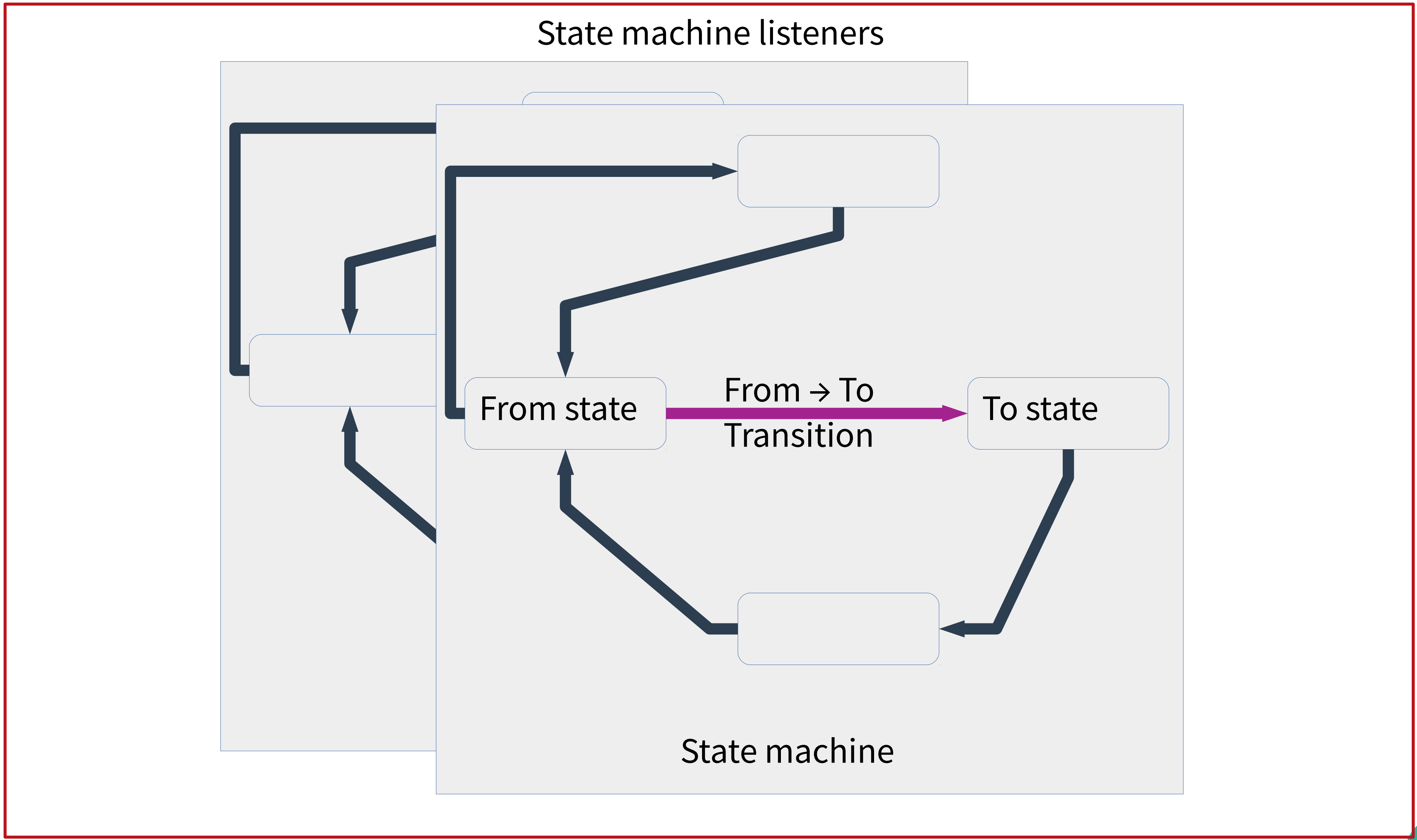
C++ callbacks and their many incarnations

A case study: state machines with transition action delegation.

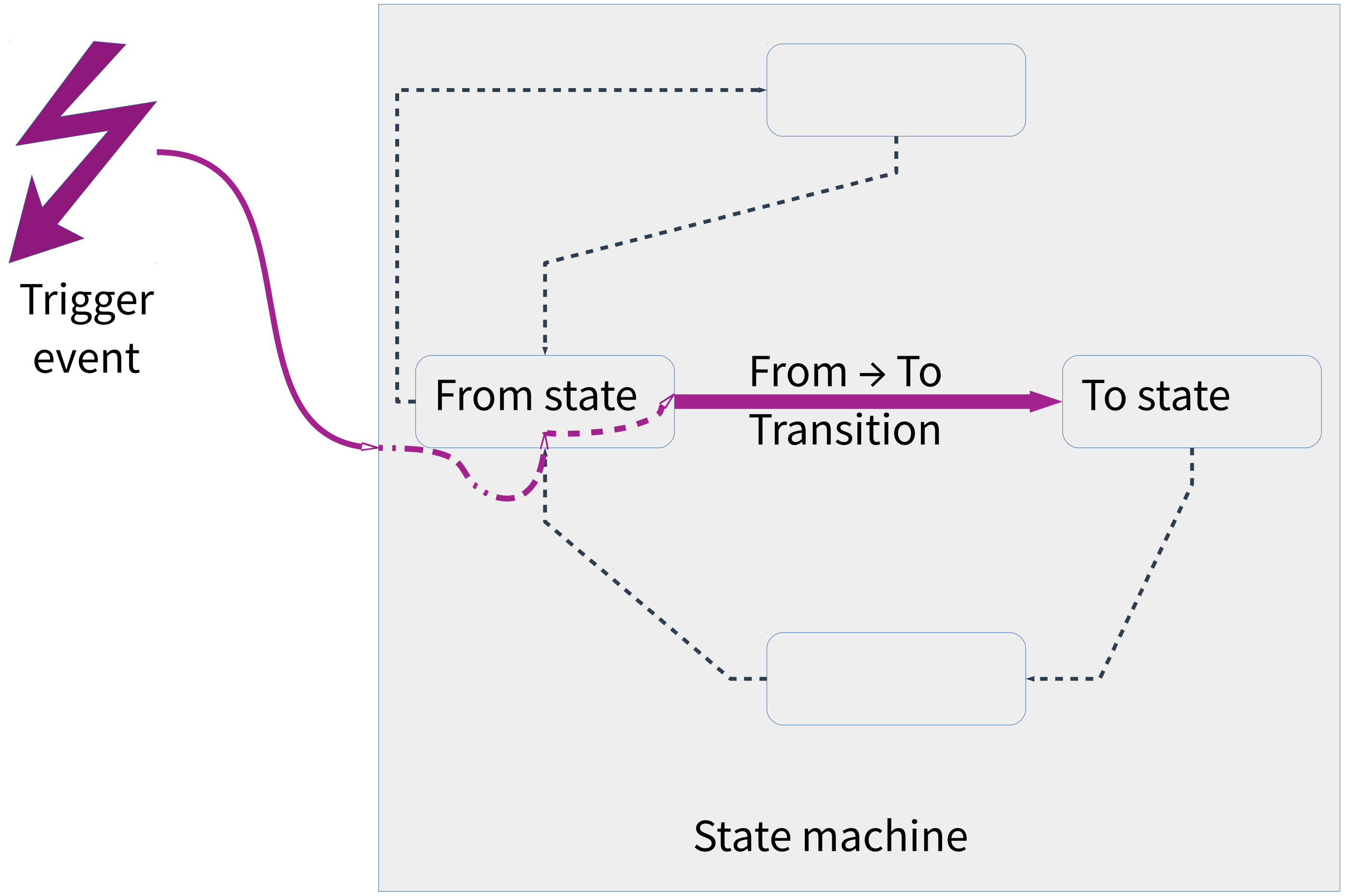
Contents

Case study: a state machine implementation



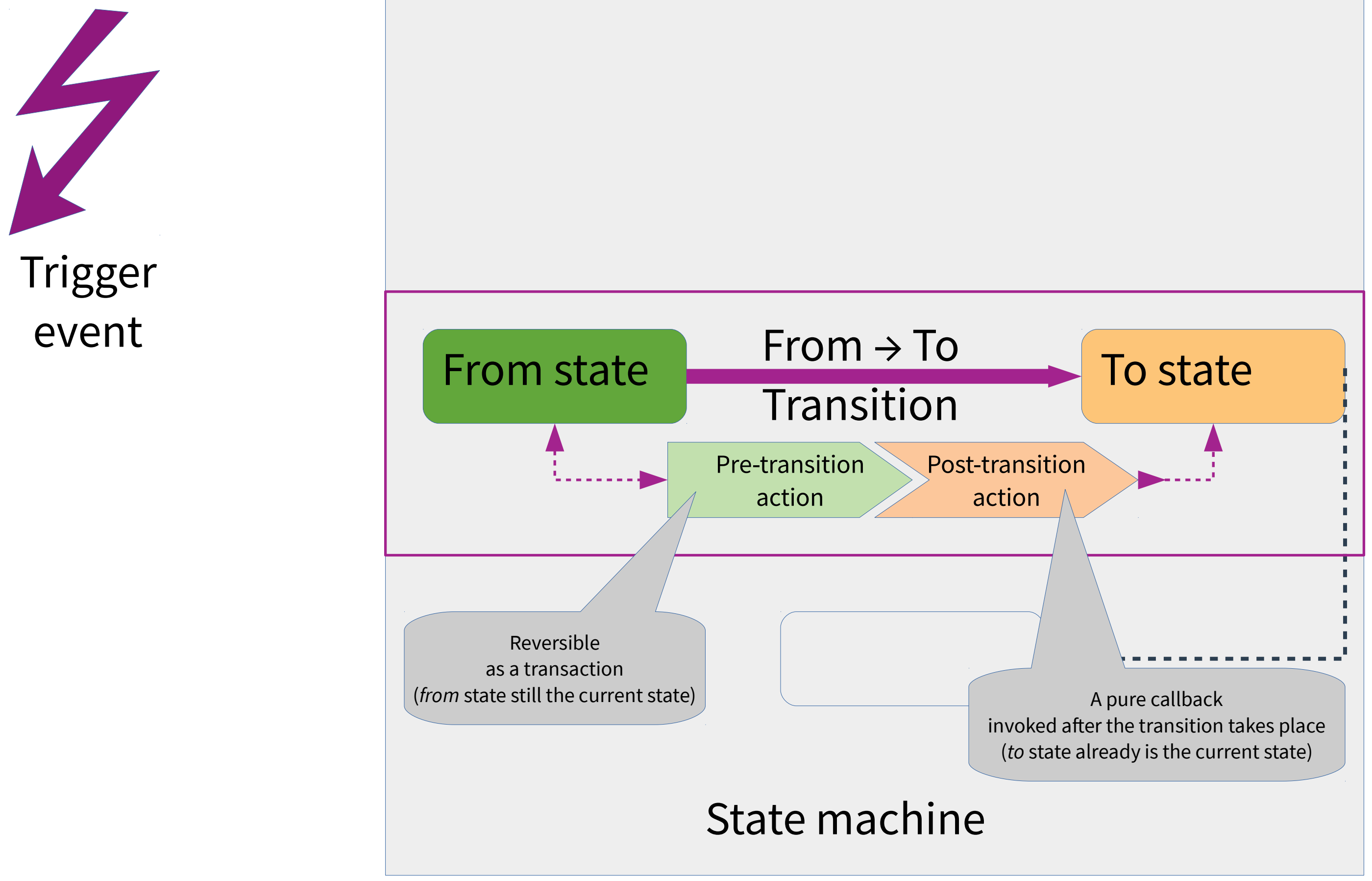
Case study: a state machine implementation

Triggering a transition



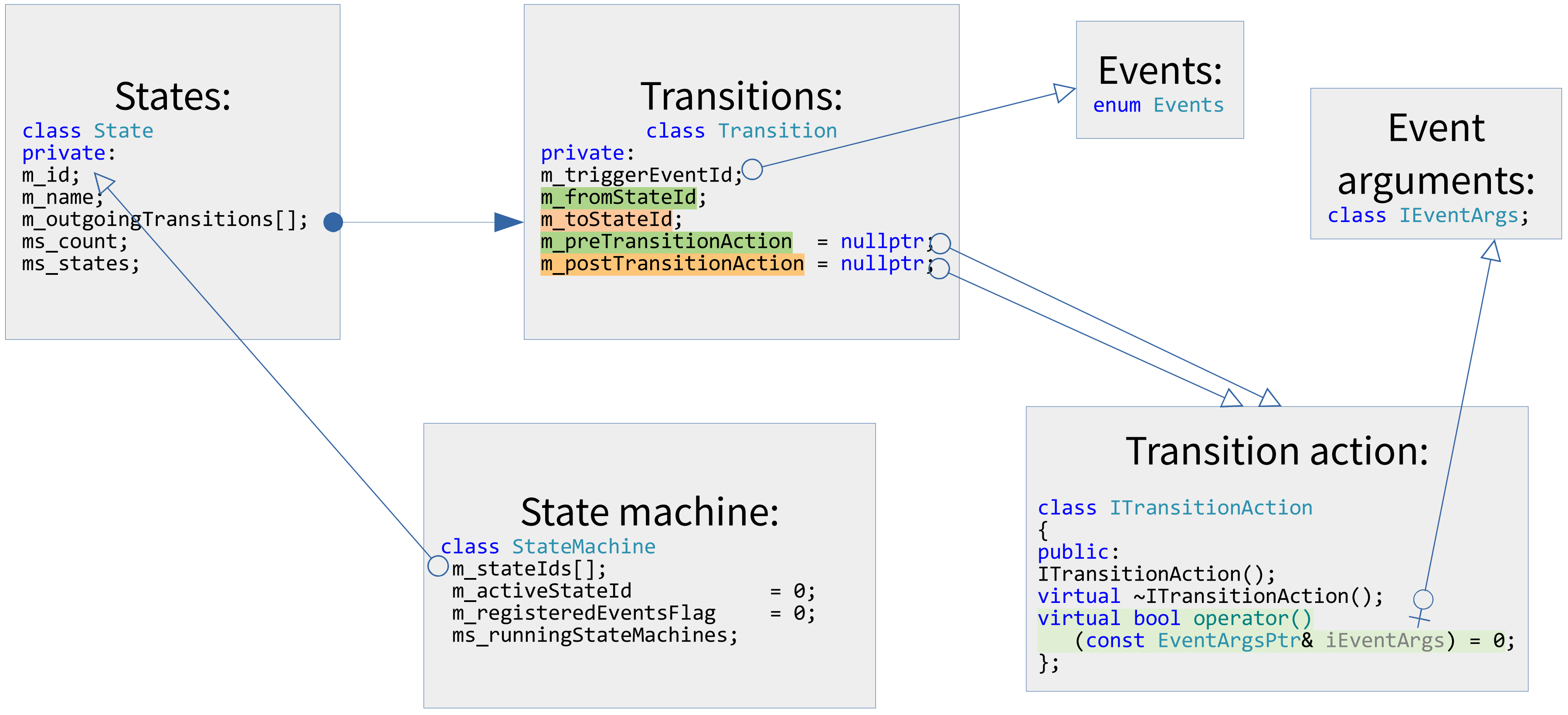
Case study: a state machine implementation

Stages of a transition



Case study: a state machine implementation

State machine API elements



```
class State
private:
m_id;
m_name;
m_outgoingTransitions[];
ms_count;
ms_states;
```

```
class Transition
private:
m_triggerEventId;
m_fromStateId;
m_toStateId;
m_preTransitionAction   = nullptr;
m_postTransitionAction  = nullptr;
```

```
enum Events
```

```
class IEventArgs;
```

```
class StateMachine
 m_stateIds[];
 m_activeStateId             = 0;
 m_registeredEventsFlag      = 0;
 ms_runningStateMachines;
```

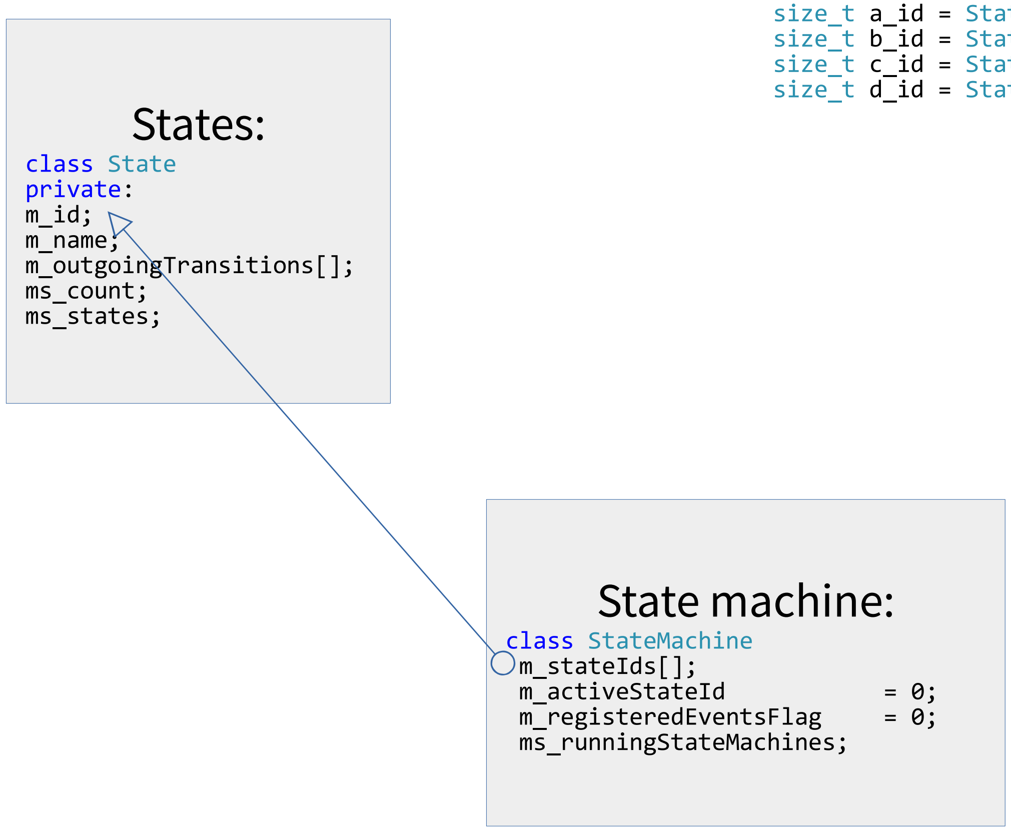
```
class ITransitionAction
{
public:
ITransitionAction();
virtual ~ITransitionAction();
virtual bool operator()
   (const EventArgsPtr& iEventArgs) = 0;
};
```

Creating the state machine

Defining the states

```
StateMachine stateMachine({ "A", "B", "C", "D" });

size_t a_id = State::Factory::GetByName("A")->GetId();
size_t b_id = State::Factory::GetByName("B")->GetId();
size_t c_id = State::Factory::GetByName("C")->GetId();
size_t d_id = State::Factory::GetByName("D")->GetId();
```



States:

```
class State
private:
m_id;
m_name;
m_outgoingTransitions[];
ms_count;
ms_states;
```

State machine:

```
class StateMachine
m_stateIds[];
m_activeStateId         = 0;
m_registeredEventsFlag  = 0;
ms_runningStateMachines;
```

```
/*  Layout of a simple test state machine
        --->B---\
       /         \
   *A -           ->D-----\
     ^ \         /         \
      \ --->C---/           \
       \_______________________/

*/
```

Creating the state machine

Defining the transitions

States:

```
class State
private:
m_id;
m_name;
m_outgoingTransitions[];
ms_count;
ms_states;
```

```
enum Events
{
        eAtoB,
        eAtoC,
        eBtoD,
        eCtoD,
        eDtoA
};
```

```
stateMachine.CreateTransition(eAtoB, a_id, b_id,
CreateTransitionActionPtr(delegate_a_b_pre),
CreateTransitionActionPtr(delegate_a_b_post));
```

Transitions:

```
         class Transition
private:
m_triggerEventId;
m_fromStateId;
m_toStateId;
m_preTransitionAction   = nullptr;
m_postTransitionAction  = nullptr;
```

State machine:

```
class StateMachine
 m_stateIds[];
 m_activeStateId            = 0;
 m_registeredEventsFlag     = 0;
 ms_runningStateMachines;
```

```
/*  Layout of a simple test state machine
        --->B---\
       /         \
   *A -             ->D-----\
     ^ \         /           \
      \ --->C---/             \
       \______________________/

*/
```

Using the state machine

Triggering the transitions

States:

```
class State
private:
m_id;
m_name;
m_outgoingTransitions[];
ms_count;
ms_states;
```

Transitions:

```
         class Transition
private:
m_triggerEventId;
m_fromStateId;
m_toStateId;
m_preTransitionAction  = nullptr;
m_postTransitionAction = nullptr;
```

```
stateMachine.OnEvent(eAtoB, a_b_str, a_b_str);  // go to b
stateMachine.OnEvent(eAtoC, a_b_str, a_b_str);  // go to c
stateMachine.OnEvent(eBtoD, b_d_str, b_d_str);  // go to d
stateMachine.OnEvent(eDtoA, d_a_str, d_a_str);  // go to a
stateMachine.OnEvent(eAtoC, a_c_str, a_c_str);  // go to c
//stateMachine.OnEvent(eCtoD, c_d_str, c_d_str);  // go to d
StateMachine::NotifyEvent(eCtoD, c_d_str, c_d_str);
```

State machine:

```
class StateMachine
 m_stateIds[];
 m_activeStateId          = 0;
 m_registeredEventsFlag   = 0;
 ms_runningStateMachines;
```

```
/*  Layout of a simple test state machine
        --->B---\
       /         \
   *A -           ->D-----\
     ^ \         /         \
      \ --->C---/           \
       \______________________/

*/
```

The C++ callback universe - flexibility

Transition action derived class

Transition action:

```
class ITransitionAction
{
public:
ITransitionAction();
virtual ~ITransitionAction();
virtual bool operator()
   (const EventArgsPtr& iEventArgs) = 0;
};
```

Transitions:

```
        class Transition
private:
m_triggerEventId;
m_fromStateId;
m_toStateId;
m_preTransitionAction   = nullptr;
m_postTransitionAction  = nullptr;
```

Classic polymorphism:

```
class SuccessfulTransitionAction :
public ITransitionAction
{
...
       SuccessfulTransitionAction(const std::string& iTransitionString) :
              m_transitionString(iTransitionString)
       {
       }

       bool operator()(const EventArgsPtr& iEventArgs) override
       {
              IGTKLOG(m_transitionString);
              if (iEventArgs != nullptr)
              {
                     ...
              }
              ...
              return true;
       }

private:
       std::string m_transitionString;
};
```

```
/*  Layout of a simple test state machine
        --->B---\
       /         \
   *A -             ->D-----\
     ^ \         /           \
      \ --->C---/             \
       \_______________________/

*/
```

The C++ callback universe - flexibility

Transition actions from (static or global) functions

Transition action:

```cpp
class ITransitionAction
{
public:
ITransitionAction();
virtual ~ITransitionAction();
virtual bool operator()
   (const EventArgsPtr& iEventArgs) = 0;
};
```

Transitions:

```cpp
        class Transition
private:
m_triggerEventId;
m_fromStateId;
m_toStateId;
m_preTransitionAction   = nullptr;
m_postTransitionAction  = nullptr;
```

Wrap the function in an *action functor* class:

```cpp
bool ActionFunction(const EventArgsPtr& iEventArgs)
{
       IGTKLOG(" Attempting transition function. ");
       if (iEventArgs != nullptr)
       {
              ...
       }
       return true;
}
struct StaticStruct
{
       static bool Action(const EventArgsPtr& iEventArgs)
       {
              IGTKLOG(" Attempting transition function. ");
              ActionFunction(iEventArgs);
       }
};

stateMachine.CreateTransition(eBtoD, b_id, d_id, nullptr,
       CreateTransitionActionPtr(&ActionFunction));
stateMachine.CreateTransition(eBtoD, b_id, d_id,
       CreateTransitionActionPtr(&StaticStruct::Action),
       CreateTransitionActionPtr(&ActionFunction));
```

```cpp
/*  Layout of a simple test state machine
        --->B---\
       /         \
   *A -             ->D-----\
     ^ \         /           \
      \ --->C---/             \
       \______________________/


*/
```

The C++ callback universe - flexibility

Transition action:

```
class ITransitionAction
{
public:
ITransitionAction();
virtual ~ITransitionAction();
virtual bool operator()
   (const EventArgsPtr& iEventArgs) = 0;
};
```

Transition actions from member functions (methods)

Transitions:

```
class Transition
private:
m_triggerEventId;
m_fromStateId;
m_toStateId;
m_preTransitionAction  = nullptr;
m_postTransitionAction = nullptr;
```

Wrap the object instance and method pointer in an *delegate* class:

```
struct OutgoingAction
{
       bool Pre(const EventArgsPtr& iEventArgs)
       {
              IGTKLOG(m_bundleName);...

              return true;
       }

       bool Post(const EventArgsPtr& iEventArgs)
       ...
       std::string m_bundleName;
};
OutgoingAction action_a_c;
ActionDelegate<OutgoingAction> delegate_a_c_post(&action_a_c, &OutgoingAction::Post);
stateMachine.CreateTransition(eAtoC, a_id, c_id,
CreateTransitionActionPtr(&action_a_c, &OutgoingAction::Pre),
CreateTransitionActionPtr(delegate_a_c_post));
```

```
/*  Layout of a simple test state machine
        --->B---\
       /         \
   *A -            ->D-----\
     ^ \         /          \
      \ --->C---/            \
       \______________________/

*/
```

The C++ callback universe - flexibility

Transition action:

```cpp
class ITransitionAction
{
public:
ITransitionAction();
virtual ~ITransitionAction();
virtual bool operator()
   (const EventArgsPtr& iEventArgs) = 0;
};
```

Transition actions from std::function or std::bind

Transitions:

```cpp
          class Transition
private:
m_triggerEventId;
m_fromStateId;
m_toStateId;
m_preTransitionAction   = nullptr;
m_postTransitionAction  = nullptr;
```

Std::function and std::bind:

```cpp
using ActionFunctor = std::function<bool(const EventArgsPtr&)>;

bool BindableActionFunction(const std::string& iTransitionName,
const EventArgsPtr& iEventArgs)
{
       IGTKLOG(iTransitionName);
       return ActionFunction(iEventArgs);
}

auto action_b_d = ActionFunctor(&ActionFunction);


stateMachine.CreateTransition(eCtoD, c_id, d_id,
nullptr,
CreateTransitionActionPtr(action_b_d));

auto boundFunction = std::bind(BindableActionFunction, d_a, std::placeholders::_1);
auto action_d_a_pre = CreateTransitionActionPtr(boundFunction);
stateMachine.CreateTransition(eDtoA, d_id, a_id, action_d_a_pre, nullptr);
```

```
/*  Layout of a simple test state machine
        --->B---\
       /         \
   *A -             ->D-----\
     ^ \         /           \
      \ --->C---/             \
       \______________________/

*/
```

The C++ callback universe - flexibility

Transition actions from lambda expressions

Transition action:

```
class ITransitionAction
{
public:
ITransitionAction();
virtual ~ITransitionAction();
virtual bool operator()
   (const EventArgsPtr& iEventArgs) = 0;
};
```

Transitions:

```
        class Transition
private:
m_triggerEventId;
m_fromStateId;
m_toStateId;
m_preTransitionAction   = nullptr;
m_postTransitionAction  = nullptr;
```

Lambda expressions:

```
auto action_c_d_pre = ActionFunctor(&ActionFunction);
auto action_c_d_post = [](const EventArgsPtr& iArgs)->bool
{
   IGTKLOG("c->d_post"); return true;
};
stateMachine.CreateTransition(eCtoD, c_id, d_id,
       CreateTransitionActionPtr(action_c_d_pre),
       CreateTransitionActionPtr(action_c_d_post));


auto boundFunction = std::bind(BindableActionFunction, d_a, std::placeholders::_1);
auto action_d_a_pre = CreateTransitionActionPtr(boundFunction);

stateMachine.CreateTransition(eDtoA, d_id, a_id, action_d_a_pre,
CreateTransitionActionPtr(
              [&d_a_str](const EventArgsPtr& iArgs)
       {
              IGTKLOG(*(StringEventArgs*)(&*iArgs)); return false;
       }
       )
);
```

```
/*  Layout of a simple test state machine
        --->B---\
       /         \
   *A -            ->D-----\
     ^ \         /          \
      \ --->C---/            \
       \______________________/

*/
```

The C++ callback universe – flexibility – the GenericTransactionAction core

```cpp
#pragma once
#include <functional>
#include "ITransitionAction.h"

using ActionFunctor = std::function<bool(const EventArgsPtr&)>;

template <class TWrappedClass>
class ActionDelegate
{
	using MethodPointer = bool (TWrappedClass::*)(const EventArgsPtr&);

public:
	ActionDelegate() = delete;
	ActionDelegate(TWrappedClass* iWrappableInstance, MethodPointer iMethodPointer) :
		m_wrappedInstance(iWrappableInstance),
		m_methodPointer(iMethodPointer)
	{
	}

	bool operator()(const EventArgsPtr& iEventArgs)
	{
		if (m_wrappedInstance == nullptr || m_methodPointer == nullptr)
		{
			return false;
		}
		return (m_wrappedInstance->*m_methodPointer)(iEventArgs);
	}
private:
	TWrappedClass*		m_wrappedInstance;
	MethodPointer m_methodPointer;

};
```

The C++ callback universe – flexibility – the GenericTransactionAction core

```cpp
/// This class is able to wrap function pointers, std::functions and lambda expressions
template <class TFunction>
class GenericTransitionAction :
	public ITransitionAction
{
public:
	GenericTransitionAction(TFunction iFunctor) :
		m_functor(iFunctor)
	{
	}


	~GenericTransitionAction() = default;

	bool operator()(const EventArgsPtr& iEventArgs)
	{
		return m_functor(iEventArgs);
	}

private:
	TFunction m_functor;
};

template <class TFunction>
GenericTransitionAction<TFunction> CreateTransitionAction(TFunction iFunctor)
{
	return GenericTransitionAction<TFunction>(iFunctor);
}

template <class TFunction>
TransitionActionPtr CreateTransitionActionPtr(TFunction iFunctor)
{
	return TransitionActionPtr(new GenericTransitionAction<TFunction>(iFunctor));
}

template <class TWrappedClass, typename TMethodPointer>
TransitionActionPtr CreateTransitionActionPtr(TWrappedClass* iWrappedObject, TMethodPointer iMethodPointer)
{
	return CreateTransitionActionPtr(ActionDelegate<TWrappedClass>(iWrappedObject, iMethodPointer));
}
```

https://github.com/teodron/IGTK/wiki/The-state-machine-design