boostcon
BOOST LIBRARIES CONFERENCE 2008

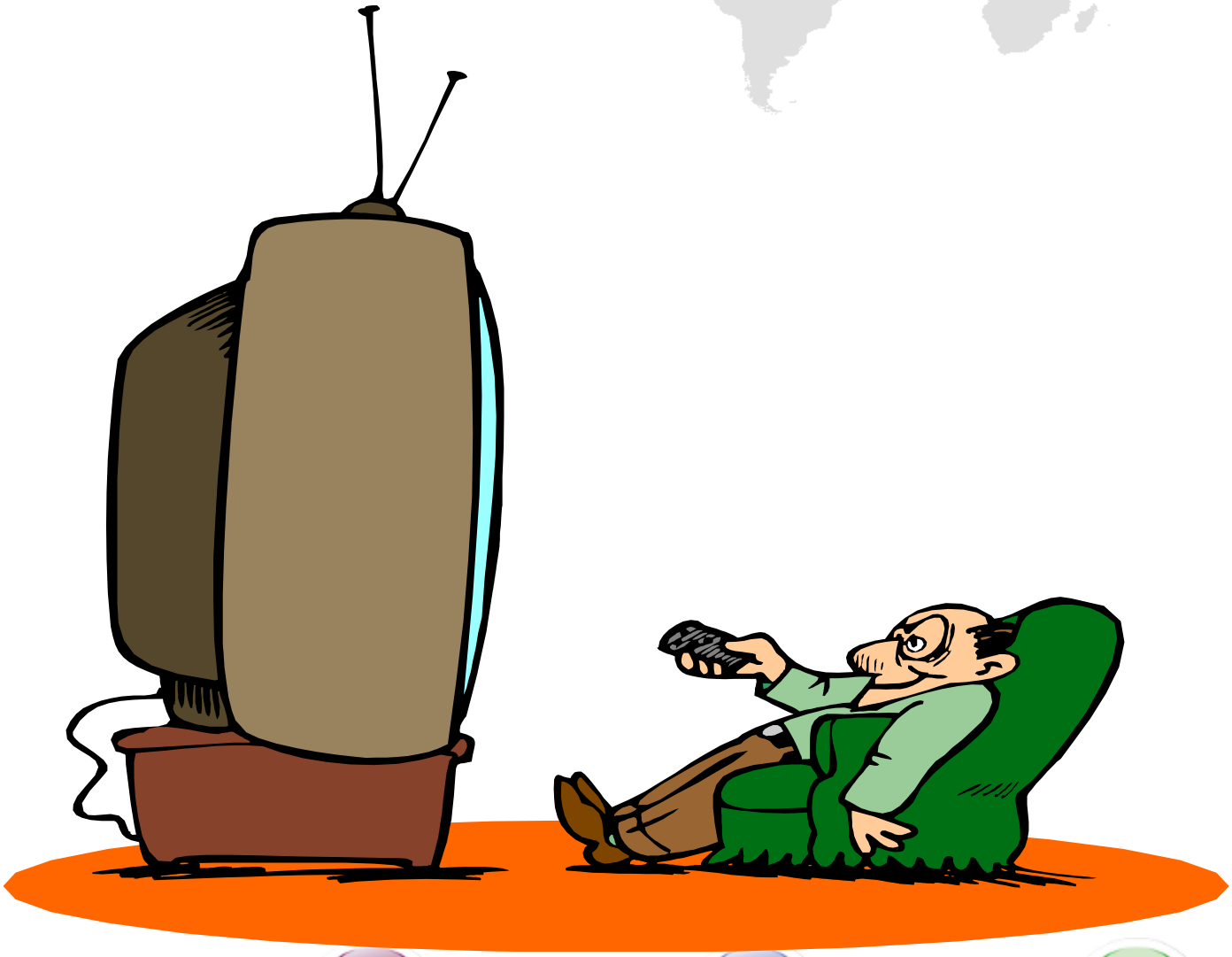
NDS

State-Oriented Programming with Boost Statecharts Library

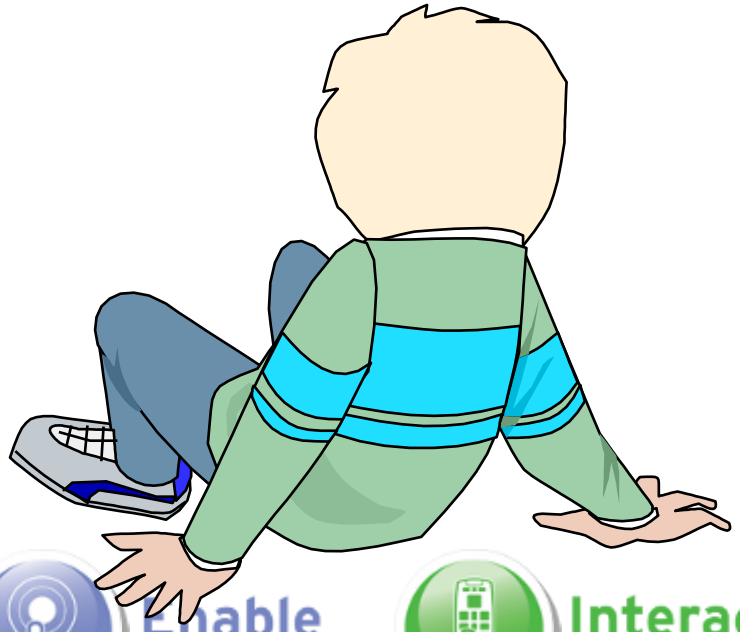
Asher Sterkin, NDS Technologies, Israel

Secure Enable Interact

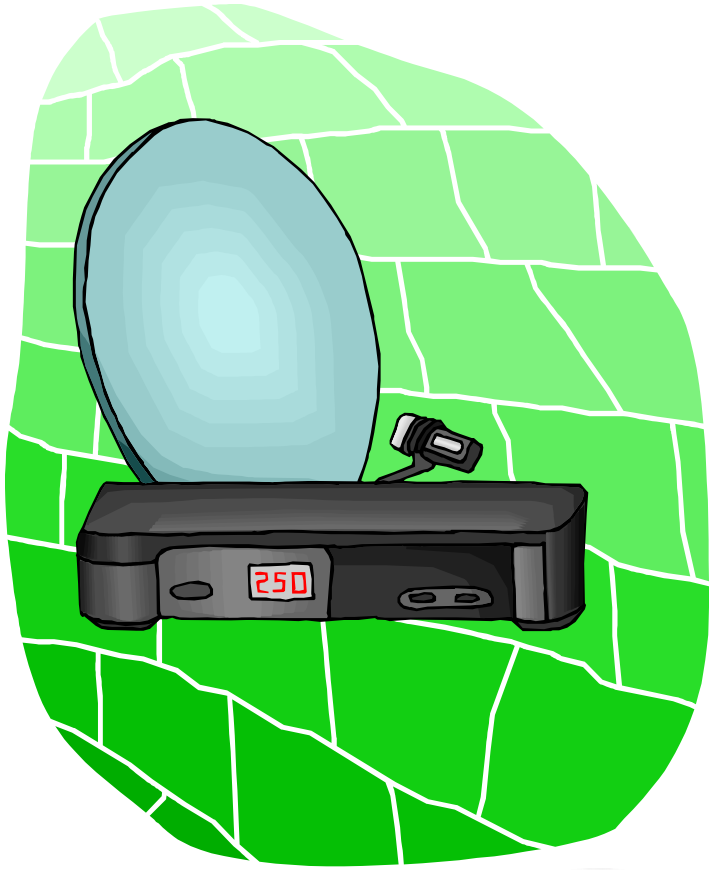
Let's Watch TV



For Some People TV is Like This



In Reality It's More Like This



Secure Enable Interact

ALL CHANNEL GUIDE	6:30pm	7:00pm	7:30pm	8:00pm	8:30pm
1001 WMAQ NBC5	Soccer Skills TV	Deal or No Deal		Dateline	
1002 WGBO Univision	National Geograp...	La Fea Mas Bella		Barrera de Amor	
1003 WLS ABC7	X-Men III: The ...	Serenity			Boston Legal
1004 WPWR UPN	Lost Without a Kiss	America's Next Top Model		Girlfriends	Fear Factor
1005 WGN	Movie 06	X-Men III: The Making Of		Bedford Diaries	Home Improv...
1006 WFLD FOX	Arrested Develo...	Bones		The OC	
1007 PTRS FILMS	The Life and Death of Peter Sellers			The Pink Panther: A Shot in the Dark	

Favourites Channel Type TV On Demand Search Up Down

Friday 8th September 06:35

10% / 22%

X-MEN III: THE MAKING OF

PG-13 6.30pm - 7.00pm DURATION 060 mins

The X-Men, mutant heroes sworn to defend a world that hates and fears them, are back! This time, with the help of new recruits The Beast and Angel, they must face evolution itself in the form of their former teammate, Jean Grey. Possessed with the cosmic power of the Dark Phoenix...

The Question Is …

How can we manage 700 screens under severe resource constraints?

Agenda

- Reactive Systems
- Why Statecharts?
- Statecharts Mechanics
- State-oriented Programming

Reactive Systems

Reminders

Metadata update

All these events happen assynchronously

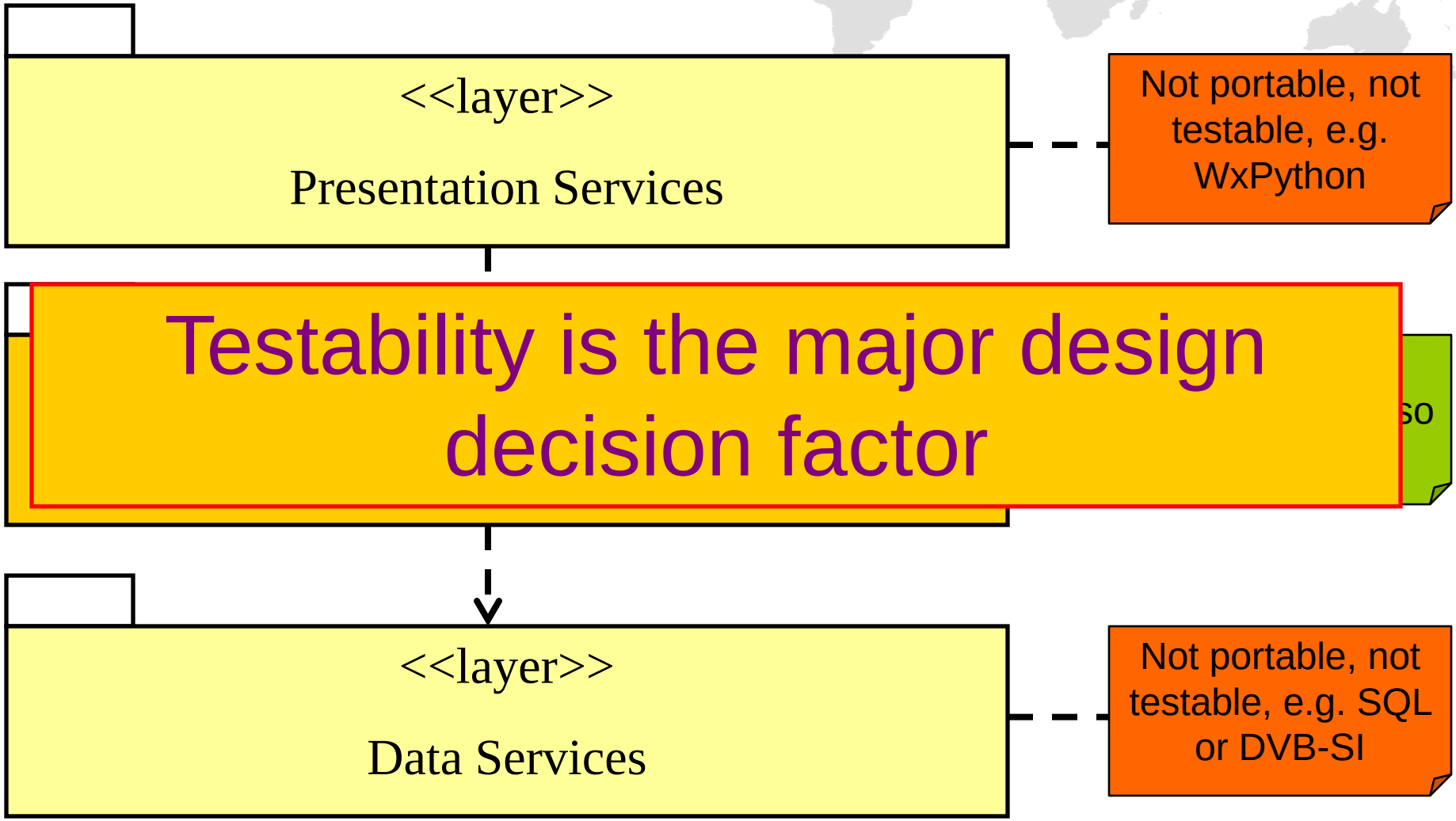
Auto booking (series link)

CA & Technical OSDs

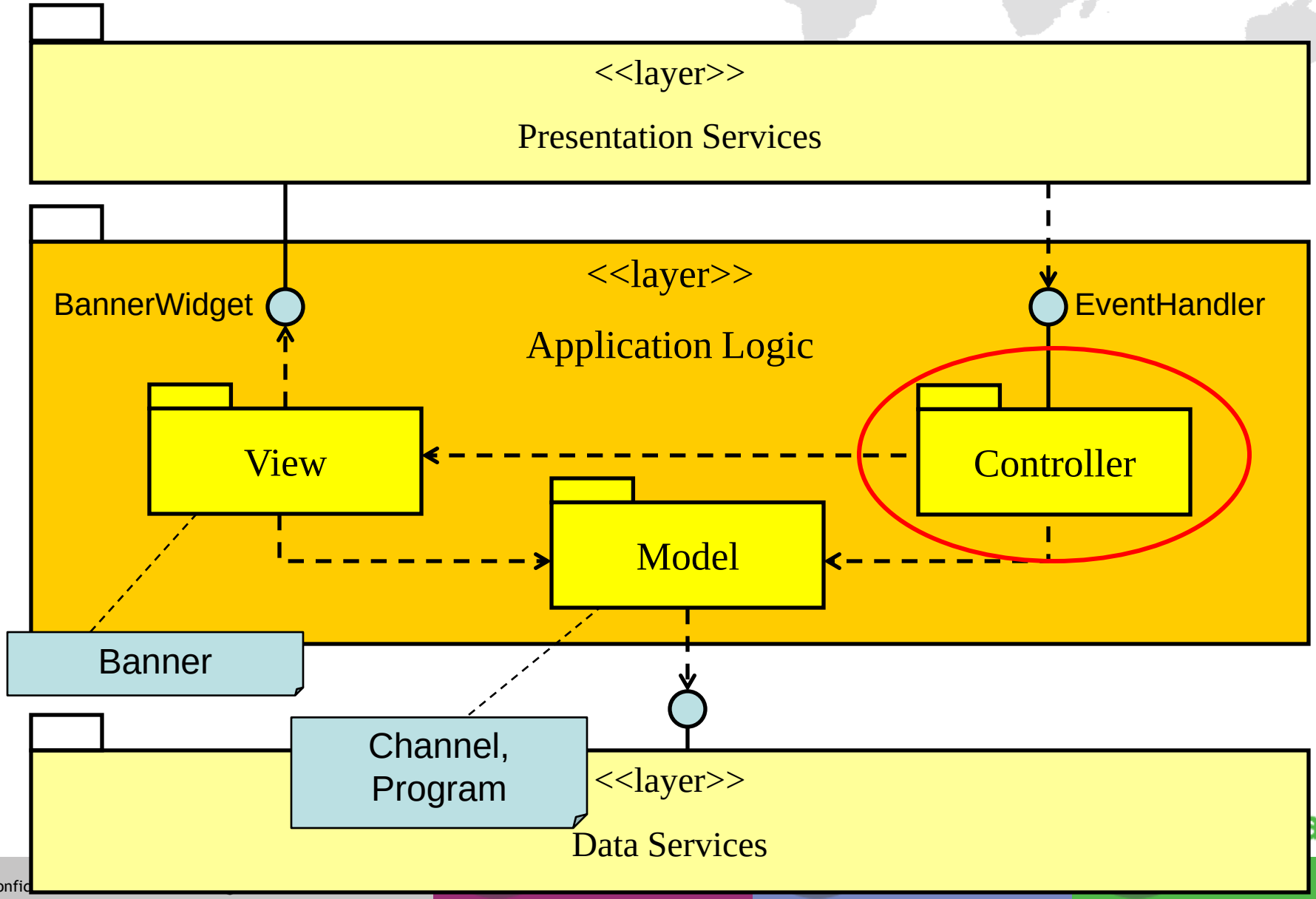
B-Mail icon

Some Design Patterns Would Help

Layers



Model-View-Controller

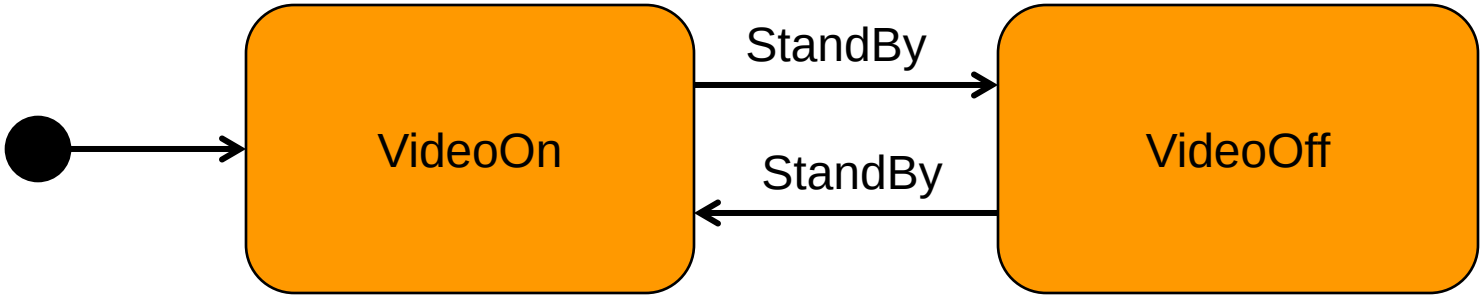


State-Driven Behavior

The number of buttons is limited

Interpretation depends on state

Example: StandBy Button



Intellectual Control

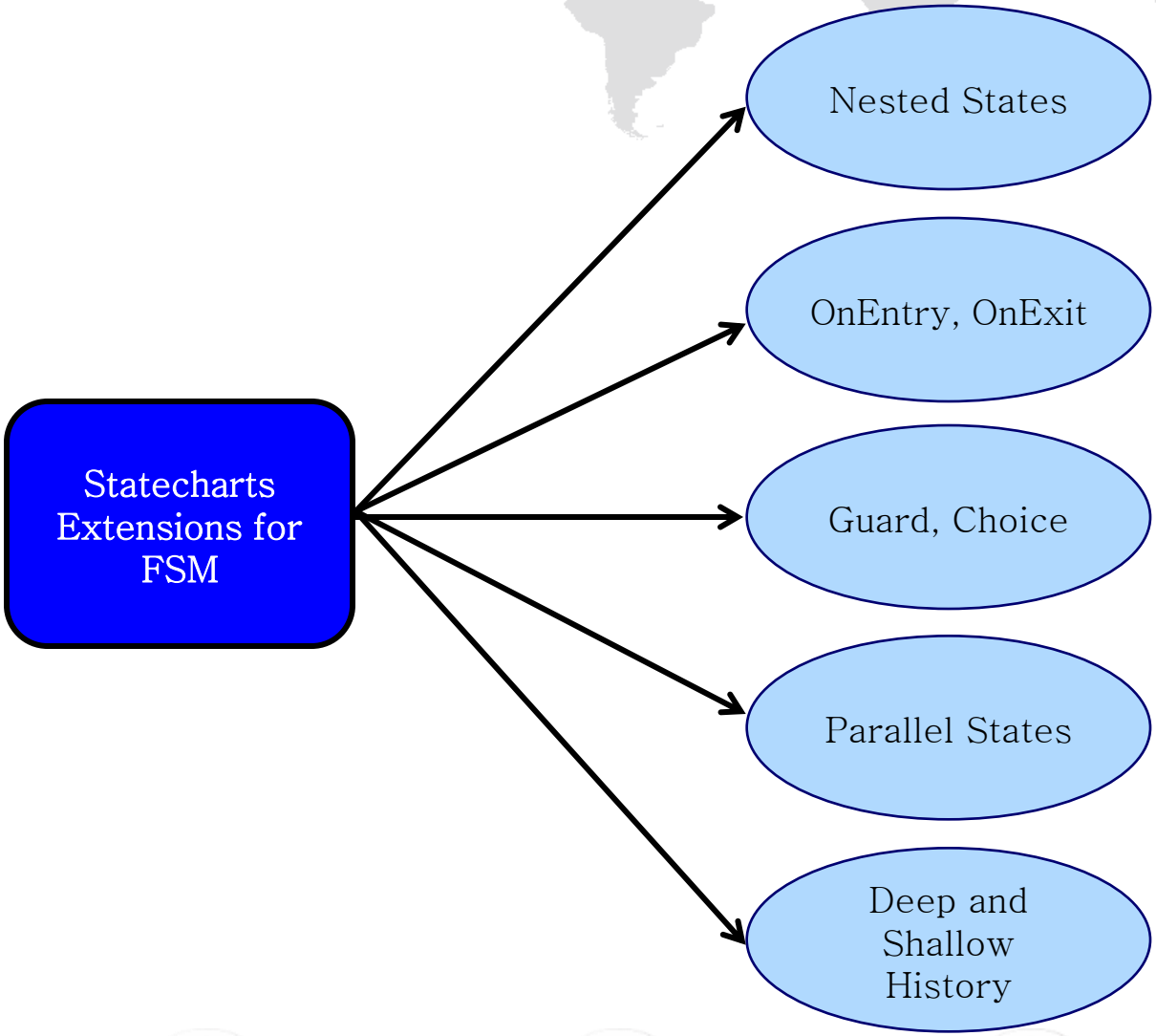
Number of Possible Transitions = #of Events * #of States^2

Lack of Intellectual Control

Large Number of States (~500 in modern EPG)

Poor Code Quality & Time To Market

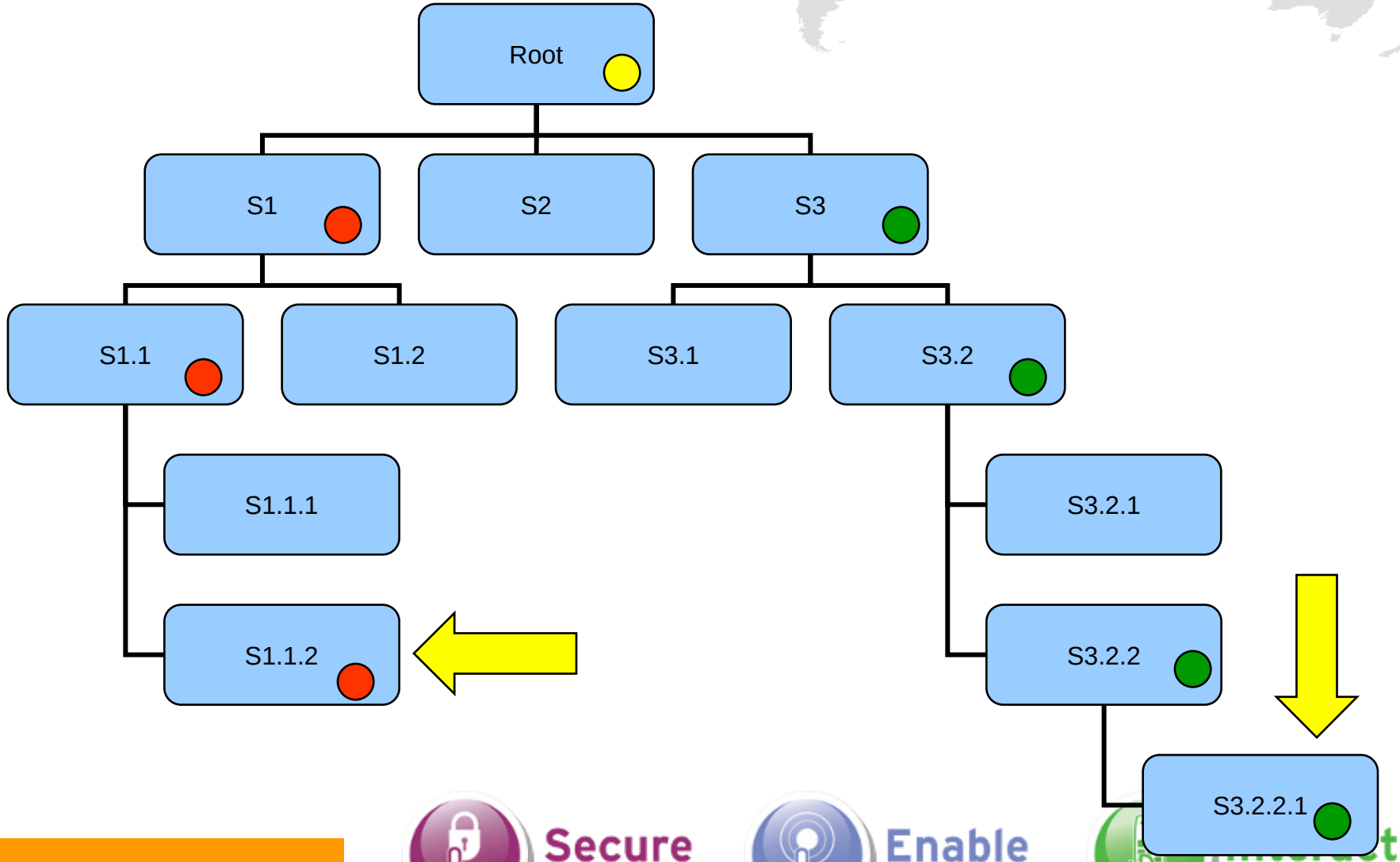
Statecharts



Nested States

To implement nested states correctly the Least Common Ancestor algorithm should be applied

LCA Basics



Statecharts Mechanism

UML 2.0 Graphical Tools

- Expensive
- Introduce Large Diagram<>Code Gap
- Dr

Want to use UML 2.0 statecharts formalism *directly* in code

EPG Tutorial

Mute

Program description comes here

Program Title

16:35 – 18:06

Pg

Synopsis

17:00

How Can We Test Statecharts?

Use PyFIt

Stand By

fitLib.DoFixture	
start	ProgramGuideTester

power up

video and audio are on

PlayerStatus	
Audio	Video
ON	ON

stand by

video and audio are off

PlayerStatus	
Audio	Video
OFF	OFF

stand by

video and audio are on

PlayerStatus	
Audio	Video
ON	ON

StandBy Statechart

- Use sample scenario to formulate requirements.
- Use statecharts to specify a cost-effective solution

PyFIT Fixture

```
import ProgramGuide as pg
from ViewFactoryStub import ViewFactoryStub as View
from DataFactoryStub import DataFactoryStub as DataAccess
from LocalSettingsFixture import LocalSettingsFixture

class NoWidget(object):
        def GetType(self):
                return "None"

class ProgramGuideTester(object):
    _typeDict = {}
    def __init__(self):
        self.view = View()
        self.data = DataAccess()

    _typeDict["powerUp.types"] = [None]
    def powerUp(self):
        self.currentWidget = NoWidget()
        self.guide         = pg.EventHandler(self.view, self.data)
        self.guide.StartEvt()

    _typeDict["widgetType.types"] = ["String"]
    def widgetType(self):
        return self.currentWidget.GetType()
```

Feeding Events to Controller

PyFIT Fixture

```
_typeDict["PlayerStatus.types"] = ["$Row"]
    def PlayerStatus(self):
        return (
            [self.view.player],
            {
                "Video" : "String",
                "Audio" : "String"
            }
        )

    _typeDict["LocalSettings.types"] = [None]
    def LocalSettings(self):
        return self.data.settings

    _typeDict["currentChannel.types"] = ["Int"]
    def currentChannel(self):
        return self.data.tuner.currentChannel

    _typeDict["standBy.types"] = [None]
    def standBy(self):
        self.guide.StandByEvt()

    def __del__(self):
        self.guide.ShutDown()
```

Feeding Events to Controller

Special treatment of shut down

StateMachine.h

```
#pragma once

namespace sc = boost::statechart;
namespace ProgramGuide
{
	namespace View { struct Factory; }
	namespace DataAccess { struct Factory; }
	namespace Controller
	{
		struct VideoOn;
		struct VideoOff;
		struct EvStandBy  : sc::event< EvStandBy > {};

		struct StateMachine : sc::state_machine< StateMachine, VideoOn >
		{
								StateMachine(View::Factory *v, DataAccess::Factory *d)
								:pView_(v)
								,pData_(d)
								{}
								void Start();
								void TuneToDefault();
								void VideoOn();
								void VideoOff();
			View::Factory		*pView_;
			DataAccess::Factory	*pData_;
		};
```

StateMachine.h

```
struct VideoOn : sc::state< VideoOn, StateMachine >
{
    VideoOn(my_context ctx)
        :sc::state<VideoOn, StateMachine>(ctx)
    {
        context< StateMachine >().VideoOn();
    }
    ~VideoOn()
    {
        context< StateMachine >().VideoOff();
    }

    typedef sc::transition< EvStandBy, VideoOff > reactions;
};

struct VideoOff : sc::simple_state< VideoOff, StateMachine >
{
    typedef sc::transition< EvStandBy, VideoOn > reactions;
};
}
}
```

OnEntry

OnExit

Simple transition

StateMachine.cpp

```
#include <stdafx.h>
#include "StateMachine.h"
namespace ProgramGuide
{
	BMOCK_VOID_METHOD(Controller::StateMachine, Start,0,())
	{
		TuneToDefault();
		initiate();
	}
	BMOCK_END
	BMOCK_VOID_METHOD(Controller::StateMachine, TuneToDefault,0,())
	{
		const int ch = pData_->GetLocalSettings()->GetDefaultChannel();
		pData_->GetTuner()->TuneTo(ch);
	}
	BMOCK_END
	BMOCK_VOID_METHOD(Controller::StateMachine, VideoOn, 0, ())
	{
		pView_->GetPlayer()->SwitchOn();
	}
	BMOCK_END
	BMOCK_VOID_METHOD(Controller::StateMachine, VideoOff, 0, ())
	{
		pView_->GetPlayer()->SwitchOff();
	}
	BMOCK_END
}
```

EventHandler.cpp

```
#include <stdafx.h>
#include "EventHandler.h"

namespace ProgramGuide
{
	using namespace Controller;

	EventHandler::EventHandler(View::Factory *v, DataAccess::Factory *d)
		:pStateMachine_(new StateMachine(v, d))
	{}

	void EventHandler::StartEvt()
	{
		pStateMachine_->Start();
	}

	void EventHandler::StandByEvt()
	{
		pStateMachine_->process_event( EvStandBy() );
	}

	void EventHandler::ShutDown()
	{
		pStateMachine_->terminate();
	}
}
```

Banner

How can we test it?

Banner: State Machine

fitLib.DoFixture	
start	ProgramGuideTester

LocalSettings
IdleTimeout
15

power up		
check	widget type	None

The "?" (help) button brings Banner widget up

help		
check	widget type	Banner

The next "?" brings Banner widget down

help		
check	widget type	None

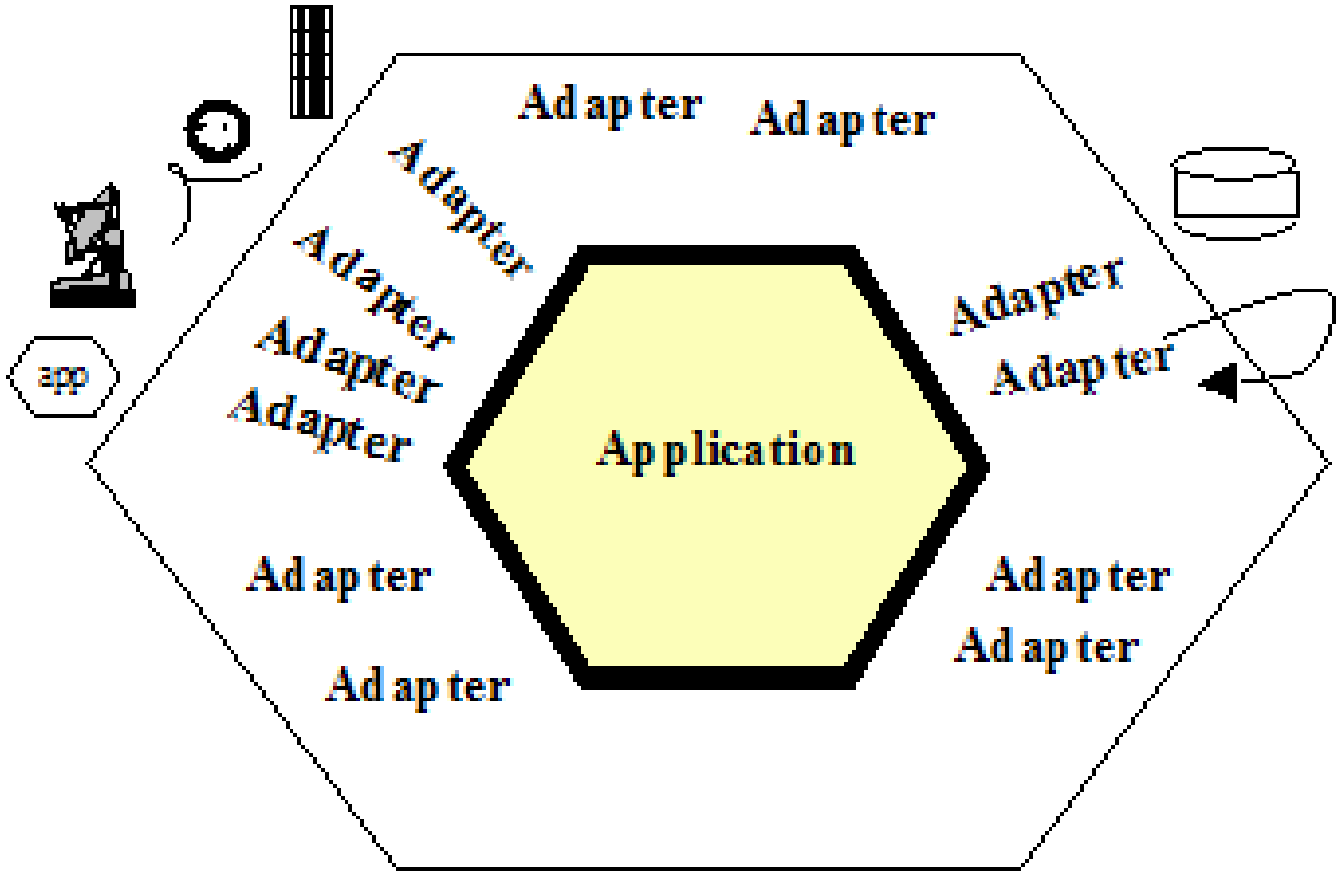
If no button is pressed the banner disapears after pre-defined timeout (in seconds)

help		
tick	5	
check	widget type	Banner
tick	16	
check	widget type	None

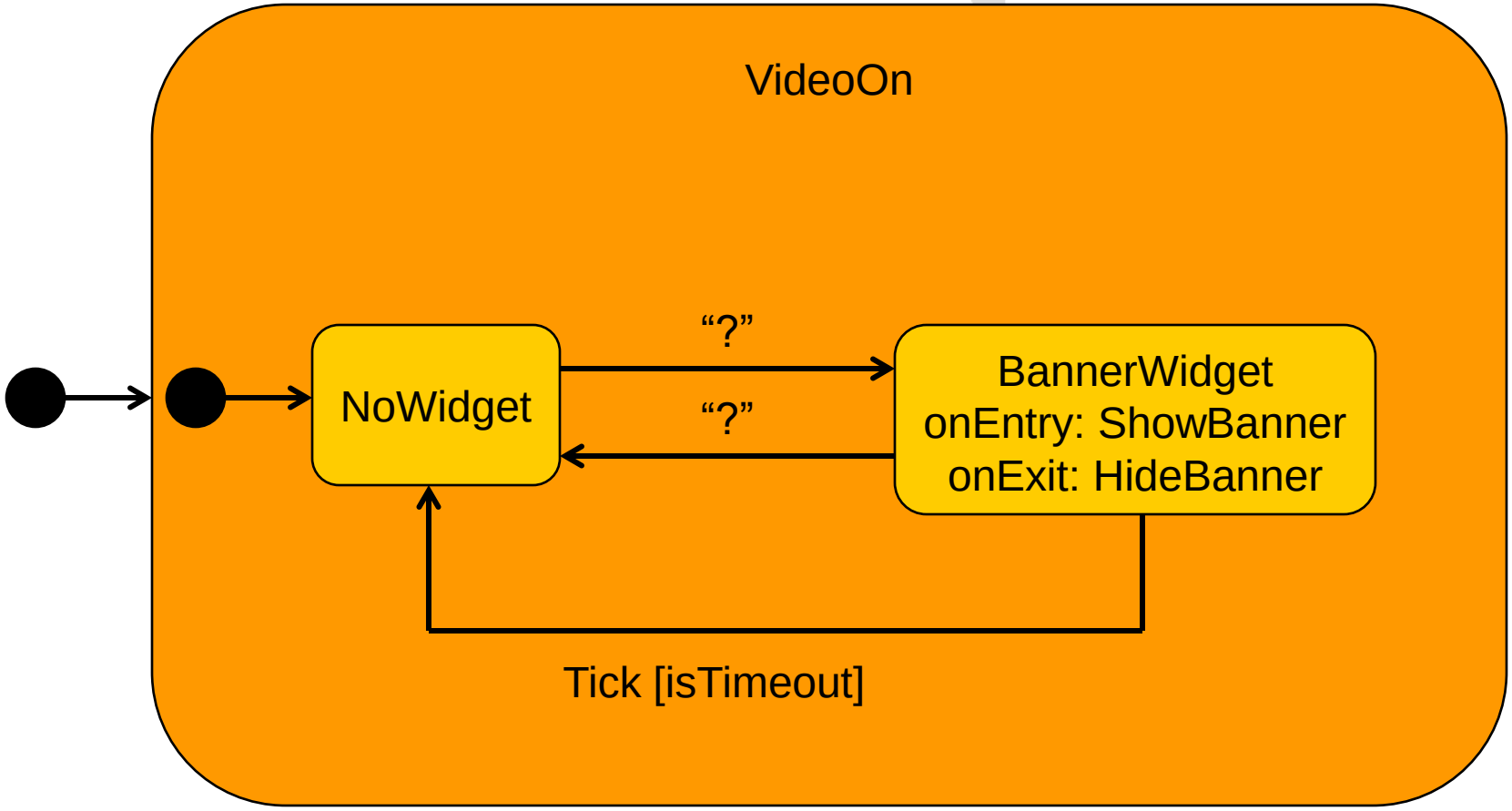
How to deal with timer?

Any ideas?

Hexagonal Architecture



Banner State Machine



StateMachine.h

```
struct VideoOn : sc::state< VideoOn, StateMachine, NoWidget >
{
        VideoOn(my_context ctx);
        ~VideoOn();

        typedef sc::transition< EvStandBy, VideoOff >          reactions;
        View::MediaPlayer                                      *pPlayer_;
};
struct NoWidget : sc::simple_state< NoWidget, VideoOn >
{
        typedef sc::transition< EvHelp, BannerWidget > reactions;
};
struct BannerWidget : sc::state< BannerWidget, VideoOn >
{
                        BannerWidget(my_context ctx);
        sc::result      react( const EvTick & );
                        ~BannerWidget();

        typedef mpl::list<
                sc::transition< EvHelp, NoWidget >,
                sc::custom_reaction< EvTick >
        > reactions;

        View::Banner *pBanner_;
};
```

Custom transition

Multiple transitions

BannerWidget

```
#include <stdafx.h>
#include "StateMachine.h"

namespace ProgramGuide
{
	namespace Controller
	{
		BannerWidget::BannerWidget(my_context ctx)
			:sc::state<BannerWidget, VideoOn>(ctx)
			,pBanner_(context< StateMachine >().pView_->GetBanner())
		{
			pBanner_ -> Show();
		}

		sc::result BannerWidget::react( const EvTick & )
		{
			if ( pBanner_->IsTimeout() )
				return transit < NoWidget >();
			return discard_event();
		}

		BannerWidget::~BannerWidget()
		{
			context< StateMachine >().pView_->DisposeBanner();
		}
	}
}
```

Implementing Guard

Synopsis

How can we test it?

Synopsis Acceptance Test

power up

check	widget type	None

press button	+

check	widget type	Synopsis

press button	+

check	widget type	None

press button	?

check	widget type	Banner

press button	+

check	widget type	Synopsis

press button	+

check	widget type	Banner

press button	+

check	widget type	Synopsis

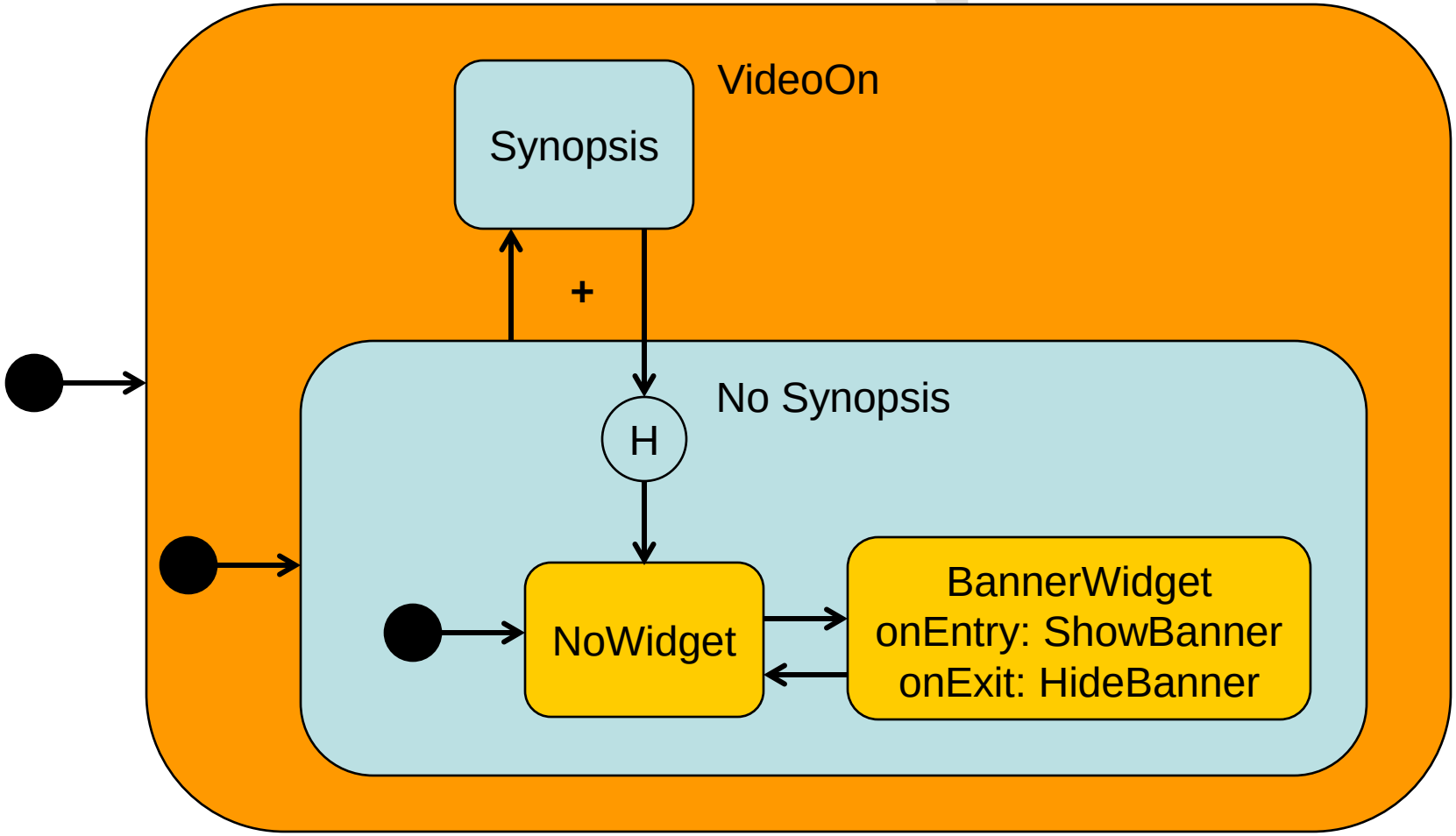
press button	?

check	widget type	Banner

press button	?

check	widget type	None

Synopsis State Machine



StateMachine.h

```
#pragma once

namespace sc = boost::statechart;
namespace mpl = boost::mpl;

namespace ProgramGuide
{
    namespace View          { struct Factory; }
    namespace DataAccess { struct Factory; }

    namespace Controller
    {
        struct VideoOn;
            struct NoSynopsis;
                struct NoWidget;
                struct BannerWidget;
            struct SynopsisWidget;
        struct VideoOff;

        struct EvStandBy : sc::event< EvStandBy > {};
        struct EvHelp    : sc::event< EvHelp > {};
        struct EvDetails : sc::event< EvDetails > {};
        struct EvTick    : sc::event< EvTick > {};
```

Reflect state nesting

StateMachine.h

```
struct StateMachine : sc::state_machine< StateMachine, VideoOn >
{
	StateMachine(View::Factory *v, Model::Factory *m)
		:pView_(v)
		,pModel_(m)
	{}

	void		Start();
	void		TuneToDefault();
	View::Banner	*GetBanner();
	void		RemoveBanner();
	View::Synopsis	*GetSynopsis();
	void		RemoveSynopsis();

	View::Factory		*pView_;
	Model::Factory		`*pModel_;
	Model::ChannelIterator`pChannel_;
};
```

StateMachine.h

```
struct VideoOn : sc::state< VideoOn, StateMachine, NoSynopsis >
{
	VideoOn(my_context ctx);
	~VideoOn();

	typedef mpl::list<
		sc::transition< EvStandBy, VideoOff >,
		sc::transition< EvHelp, BannerWidget >
	> reactions;

	View::MediaPlayer *pPlayer_;
};
struct VideoOff : sc::simple_state< VideoOff, StateMachine >
{
	typedef sc::transition< EvStandBy, VideoOn > reactions;
};
struct NoSynopsis : sc::simple_state< NoSynopsis, VideoOn, NoWidget
						, sc::has_deep_history >
{
	typedef sc::transition< EvDetails, SynopsisWidget > reactions;
};
```

StateMachine.h

```
        struct NoWidget : sc::simple_state< NoWidget, NoSynopsis > { };
        struct BannerWidget : sc::state< BannerWidget, NoSynopsis >
        {
                                BannerWidget(my_context ctx);
            sc::result          react( const EvTick & );
                                ~BannerWidget();
            typedef mpl::list<
                sc::transition< EvHelp, NoWidget >,
                sc::custom_reaction< EvTick >
            > reactions;
            View::Banner *pBanner_;
        };
        struct SynopsisWidget : sc::state< SynopsisWidget, VideoOn >
        {
            SynopsisWidget(my_context ctx);
            ~SynopsisWidget();

            typedef sc::transition< EvDetails, sc::deep_history< NoWidget > > reactions;
            View::Synopsis *pSynopsis_;
        };
    }
}
```

SynopsisWidget.cpp

```
#include <stdafx.h>
#include "StateMachine.h“
namespace ProgramGuide
{
	namespace Controller
	{
		SynopsisWidget::SynopsisWidget(my_context ctx)
			:sc::state<SynopsisWidget, VideoOn>(ctx)
		{
			context< StateMachine >().GetSynopsis()).Show();
		}

		SynopsisWidget::~SynopsisWidget()
		{
			context< StateMachine >().RemoveSynopsis();
		}
	}
}
```

Mute

Acceptance Test?

Mute Acceptance Test

power up

video and audio are on

PlayerStatus	
Audio	Video
ON	ON

check	OSD Types

press button	mute

video is on and audio is off

PlayerStatus	
Audio	Video
OFF	ON

check	OSD Types	Mute

press button	mute

video and audio are on

PlayerStatus	
Audio	Video
ON	ON

check	OSD Types

On pressing StandBy the Mute icon should disapear.

press button	mute
press button	standBy

check	OSD Types

StandBy preserves the Mute state, but blocks the mute button

press button	mute
press button	standBy

PlayerStatus	
Audio	Video
OFF	ON

check	OSD Types	Mute

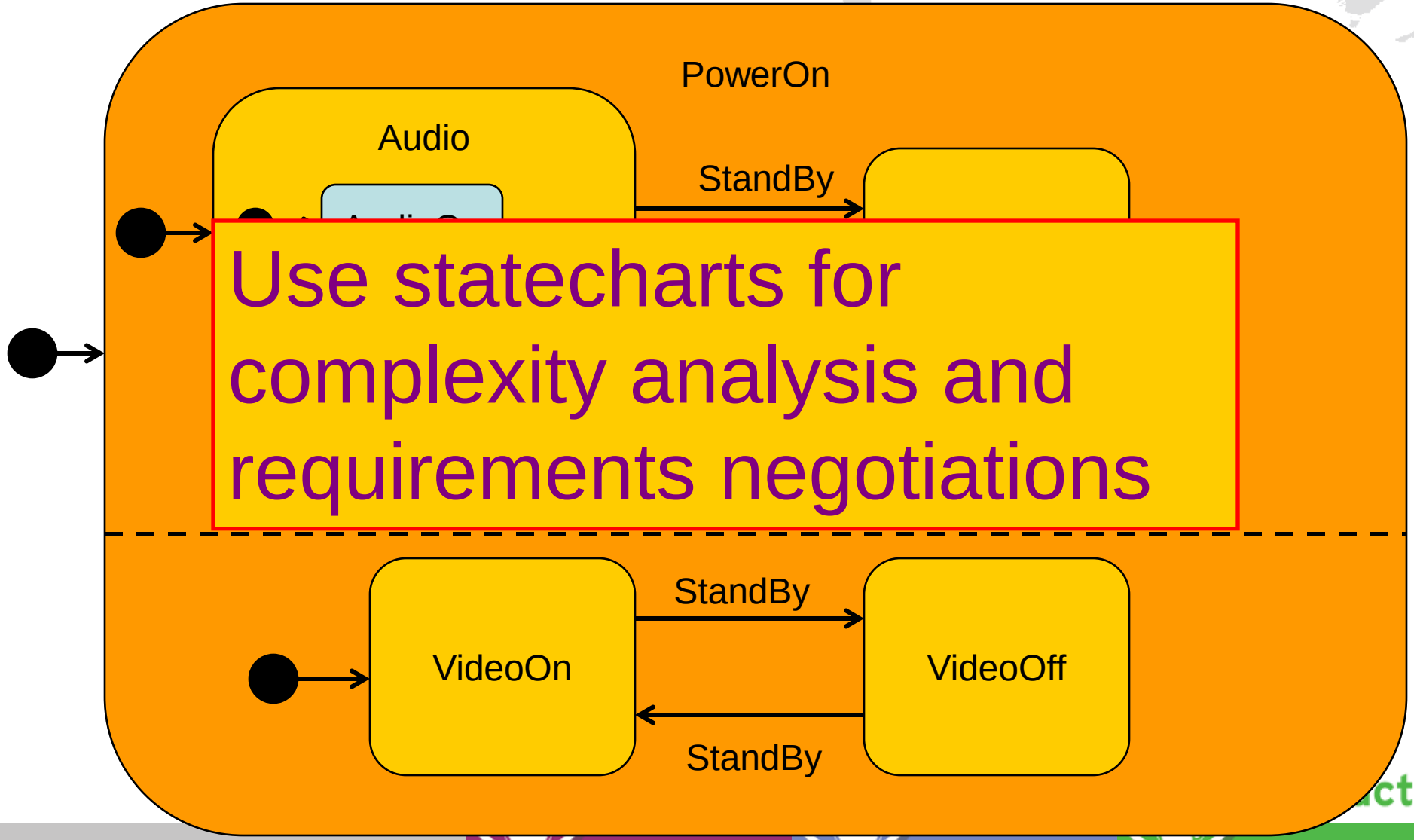
press button	?

check	OSD Types	Banner,Mute

press button	mute

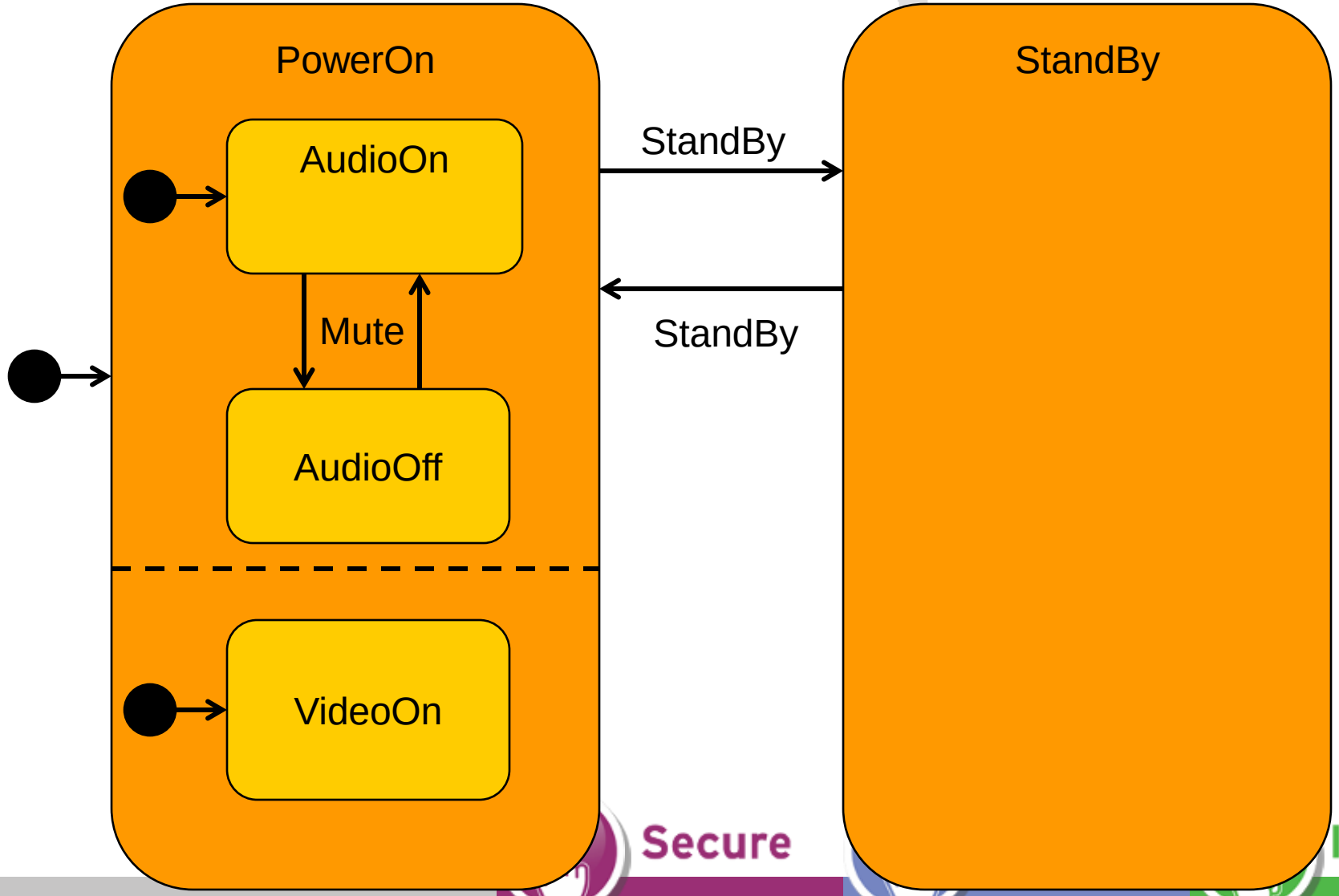
check	OSD Types	Banner

Mute Statechart



Use statecharts for complexity analysis and requirements negotiations

Mute Statechart (v.2)



StateMachine.h

```
#pragma once

namespace sc = boost::statechart;
namespace mpl = boost::mpl;

namespace ProgramGuide
{
    namespace View          { struct Factory; }
    namespace DataAccess { struct Factory; }

    namespace Controller
    {
        struct PowerOn;
            struct VideoOn;
                struct NoSynopsis;
                    struct NoWidget;
                    struct BannerWidget;
                struct SynopsisWidget;
            struct VideoOff;
            struct Audio;
                struct AudioOn;
                struct Muted;
            struct AudioOff;
```

Reflect state nesting

StateMachine.h

```
struct EvStandByVideo      : sc::event< EvStandByVideo > {};
struct EvStandByAudio      : sc::event< EvStandByAudio > {};
struct EvHelp              : sc::event< EvHelp > {};
struct EvDetails           : sc::event< EvDetails > {};
struct EvMute              : sc::event< EvMute > {};
struct EvTick              : sc::event< EvTick > {};
```

StateMachine.h

```
struct StateMachine : sc::state_machine< StateMachine, VideoOn >
{
	StateMachine(View::Factory *v, Model::Factory *m)
		:pView_(v)
		,pModel_(m)
	{}

	void		Start();
	void		TuneToDefault();
	View::Banner	*GetBanner();
	void		RemoveBanner();
	View::Synopsis	*GetSynopsis();
	void		RemoveSynopsis();

	View::Factory		*pView_;
	Model::Factory		`*pModel_;
	Model::ChannelIterator`pChannel_;
};
```

StateMachine.h

```
struct PowerOn : sc::simple_state<PowerOn, StateMachine
                                                , mpl::list< VideoOn,Audio > >
{
};

struct VideoOn : sc::state< VideoOn, PowerOn::orthogonal< 0 >
                                        , NoSynopsis >
{
    VideoOn(my_context ctx);
    ~VideoOn();

    typedef mpl::list<
        sc::transition< EvStandByVideo, VideoOff >,
        sc::transition< EvHelp, BannerWidget >
    > reactions;
};

struct VideoOff : sc::simple_state< VideoOff, PowerOn::orthogonal< 0 > >
{
    typedef sc::transition< EvStandByVideo, VideoOn > reactions;
};
```

StateMachine.h

```
struct NoSynopsis : sc::simple_state< NoSynopsis, VideoOn
                                      , NoWidget, sc::has_deep_history >
{
    typedef sc::transition< EvDetails, SynopsisWidget > reactions;
};
struct NoWidget : sc::simple_state< NoWidget, NoSynopsis >
{
};
struct BannerWidget : sc::state< BannerWidget, NoSynopsis >
{
                    BannerWidget(my_context ctx);
    sc::result      react( const EvTick & );
                    ~BannerWidget();

    typedef mpl::list<
        sc::transition< EvHelp, NoWidget >,
        sc::custom_reaction< EvTick >
    > reactions;

    View::Banner *pBanner_;
};
```

StateMachine.h

```
struct SynopsisWidget : sc::state< SynopsisWidget, VideoOn >
{
	SynopsisWidget(my_context ctx);
	~SynopsisWidget();

	typedef sc::transition<
		EvDetails,
		sc::deep_history< NoWidget >
	> reactions;
};

struct Audio : sc::simple_state<Audio, PowerOn::orthogonal< 1 >
                                       , AudioOn, sc::has_deep_history >
{
	typedef sc::transition< EvStandByAudio, AudioOff > reactions;
};
```

StateMachine.h

```
        struct AudioOn : sc::state<AudioOn, Audio >
        {
                        AudioOn(my_context ctx);
                        ~AudioOn();

            typedef sc::transition< EvMute, Muted > reactions;
        };
        struct Muted : sc::state<Muted, Audio >
        {
            Muted(my_context ctx);
            ~Muted();

            typedef sc::transition< EvMute, AudioOn > reactions;
        };
        struct AudioOff : sc::simple_state<AudioOff, PowerOn::orthogonal< 1 > >
        {
            typedef sc::transition<
                EvStandByAudio
                , sc::deep_history< AudioOn >
            > reactions;
        };
    }
}
```

AudioOn.cpp

```
#include <stdafx.h>
#include "StateMachine.h“

namespace ProgramGuide
{
	namespace Controller
	{
		AudioOn::AudioOn(my_context ctx)
			:sc::state<AudioOn, Audio >(ctx)
		{
			context< StateMachine >().pPlayer_->AudioOn();
		}
		AudioOn::~AudioOn()
		{
			context< StateMachine >().pPlayer_->AudioOff();
		}
	}
}
```

Muted.cpp

```
#include <stdafx.h>
#include "StateMachine.h"

namespace ProgramGuide
{
	namespace Controller
	{
		Muted::Muted( my_context ctx )
			:sc::state<Muted, Audion>(ctx)
		{
			context < StateMachine >().pView_->ShowMuteIcon();
		}

		Muted::~Muted ()
		{
			context < StateMachine >().pView_->HideMuteIcon();
		}
	}
}
```

EventHandler.cpp

```
#include <stdafx.h>
#include "EventHandler.h"

namespace ProgramGuide
{
	using namespace Controller;
	EventHandler::EventHandler(View::Factory *v, Model::Factory *m)
		:pStateMachine_(new StateMachine(v, m))
	{}
	void EventHandler::StartEvt()
	{
		pStateMachine_->Start();
	}
	void EventHandler::StandByEvt()
	{
		pStateMachine_->process_event( EvStandByAudio() );
		pStateMachine_->process_event( EvStandByVideo() );
	}
	void EventHandler::HelpEvt()
	{
		pStateMachine_->process_event( EvHelp() );
	}
```

EventHandler.cpp

```
    void EventHandler::DetailsEvt()
    {
        pStateMachine_->process_event( EvDetails() );
    }
    void EventHandler::MuteEvt()
    {
        pStateMachine_->process_event( EvMute() );
    }
    void EventHandler::TickEvt()
    {
        pStateMachine_->process_event( EvTick() );
    }
    void EventHandler::ShutDown()
    {
        pStateMachine_->terminate();
    }
}
```

Summary

- Testability is major design decision factor
- Use MVC for complex reactive systems
- Use scenario for requirements
- Use statecharts for cost-effective solutions
- Apply hexagonal architecture pattern
- Use statecharts for complexity analysis

Next Steps

- Read original D. Harel's paper
- Try your hands on BSL
- Attend my tutorial on TTD (Thursday)
- Drop me a line if you any have comments:

asher.sterkin@gmail.com

asterkin@nds.com