TELEN V

FRAMEWORK AND GUIDANCE FOR ARP HMI

OUTLINE

- Assumption
- Framework
  - Goal
  - Development View
  - Process View
- Underlying the Framework
- Guidance
- Coding Convention
- Tips

ASSUMPTION

- Knowledge of ARP HMI codebase
- Practice of contributing code
- Package and run app on board

- Basic knowledge of Linux
- Basic knowledge of Qt

FRAMEWORK

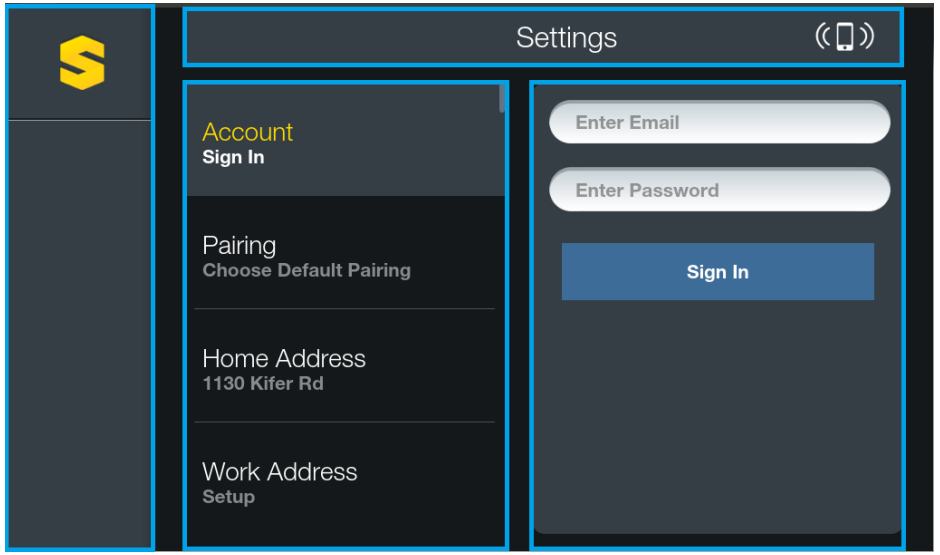
- GUI and utilities: Qt
- Render: OpenGL ES2
- Widgets and image resource: customized Qt widget, with 9-patch support
- Layout: XML
- Flow: MVC + command + state machine
- Core functionality: AutoSDK
- System service: alsa, bluetooth, etc.
- 3rd party: Neospeech VoiceText, etc.

FRAMEWORK – GOAL

- Flexible control flow
  - Flow based on state machine
  - Request / flow via command line style:
    “Service.searchPoi&query=kfc”
    “Controller.startPoi”
  - Runtime flow of control is decoupled from user interaction
    Means: the flow could be triggered from another source, i.e., connectivity from phone, or regression test framework

FRAMEWORK – GOAL

- UI fit reasonably well on all landscape screen size
  - Configure in XML, size in percentage
- Flexible UI component
  - Screen is composed of several Views, placed by Layout
  - UI elements (Widget) is specified in XML, and is constructed by WidgetBuilder



FRAMEWORK – GOAL

- Extensibility
- Reusable component
- Documentation
- Portability
- Code size and memory footprint
- I18n and l10n

FRAMEWORK – GOAL

- Never crashes (robustness)
  - Use smart pointer
  - Always check for null pointer
  - Eliminate the possibility of dangling pointer
  - No static_cast of pointer
  - Catch exception
  - Use higher level concurrency model than mutex+lock

Application Architecture Standards
- Multilayer design compliance (UI vs App Domain vs Infrastructure/Data)
- Data access performance
- Coupling Ratios
- Component (or pattern) reuse ratios

Coding Practices
- Error/exception handling (all layers UI/Logic/data)
- If applicable - compliance with OO and structured programming practices
- Secure controls (access to system functions, access controls to programs)

Complexity
- Transaction
- Algorithms
- Programming practices (eg use of polymorphism, dynamic instantation)
- Dirty programming (dead code, empty code…)

Documentation
- Code readability and structuredness
- Architecture -, program, - and code-level documentation ratios
- Source code file organization

Portability: Hardware, OS and Software component and DB dependency levels

Technical and Functional Volumes
- # LOC per technology, # of artifacts, files
- Function points - Adherence to specifications (IFPUG, Cosmic references..)

Reliability

Security

Efficiency

Maintainability

Size

SOFTWARE QUALITY

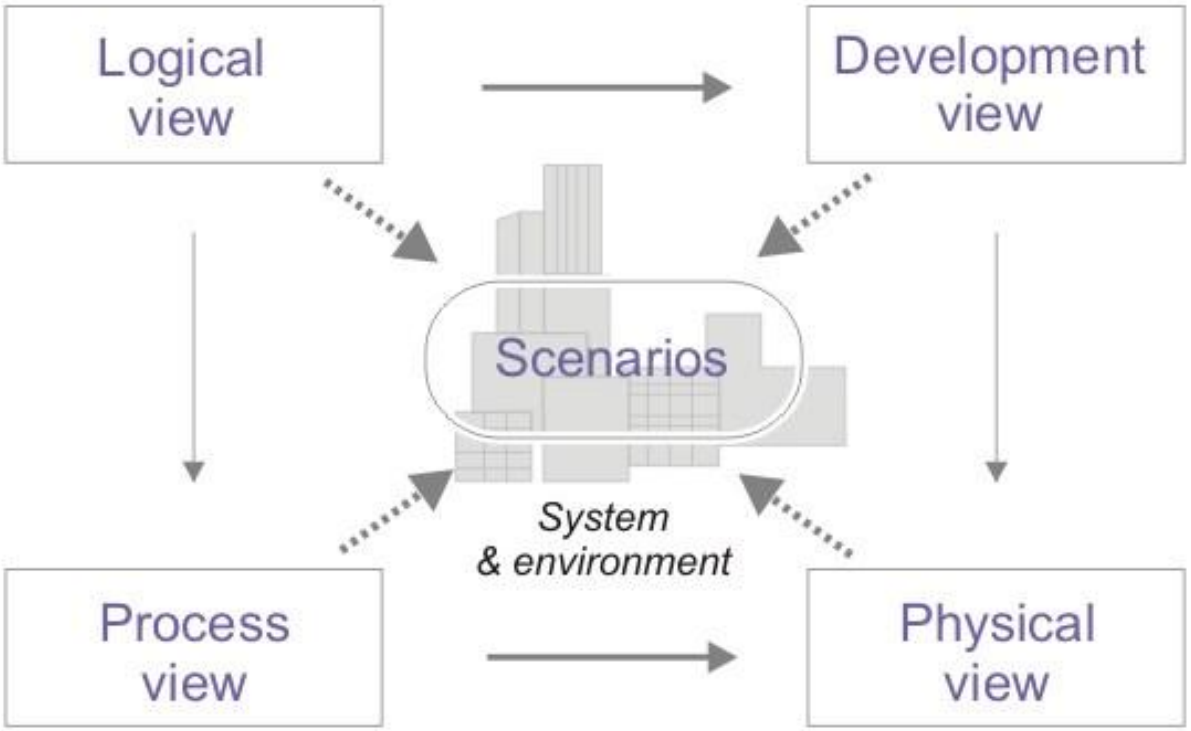
http://en.wikipedia.org/wiki/Software_quality

Software functional quality

Software structural quality

FRAMEWORK – DEVELOPMENT VIEW

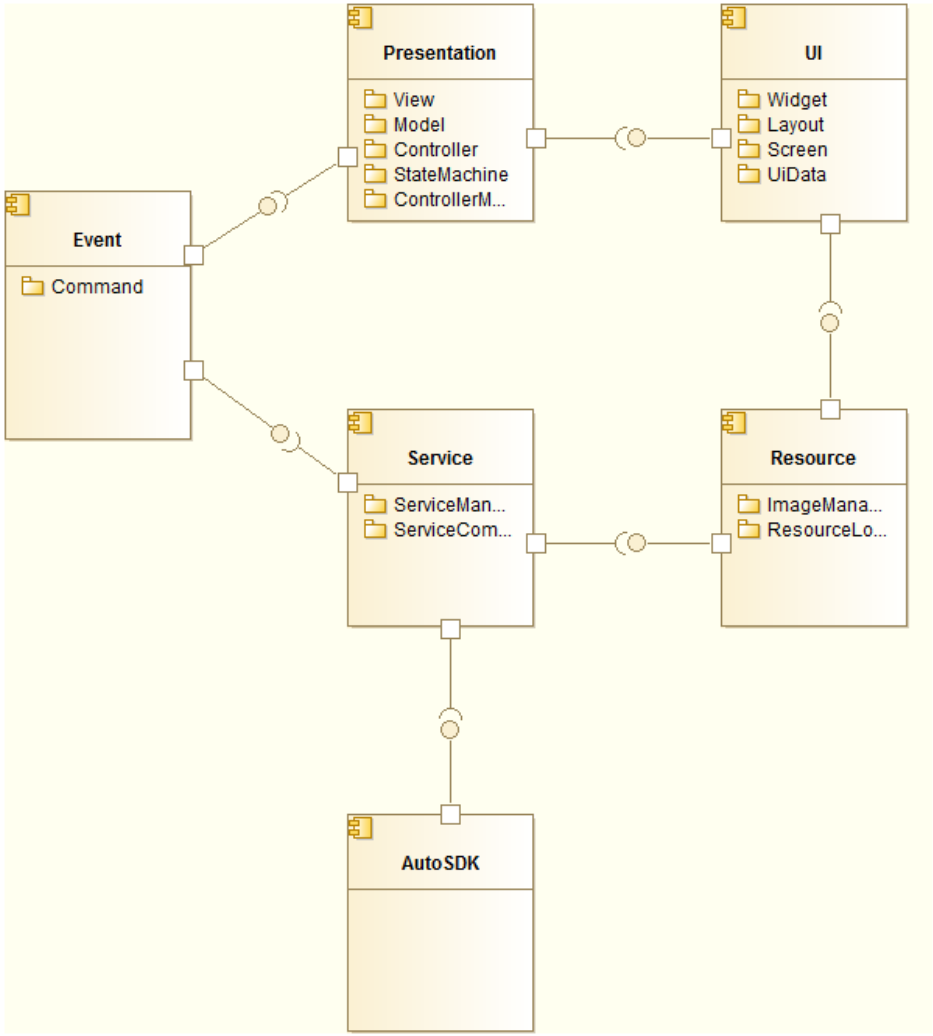
- Components
  - Common — Common and general utility classes
  - Event — Event subsystem (command pattern)
  - Presentation — Presentation subsystem, GUI (MVC pattern)
  - Resource — Resource subsystem
  - Service — Service subsystem. Integration of AutoSDK, adapter for other system services or 3rd party libraries
  - Widget — UI subsystem. Screen, widget, layout
  - InputMethod — IME subsystem



4+1 VIEW MODEL, FOR ARCHITECTURE DESCRIPTION

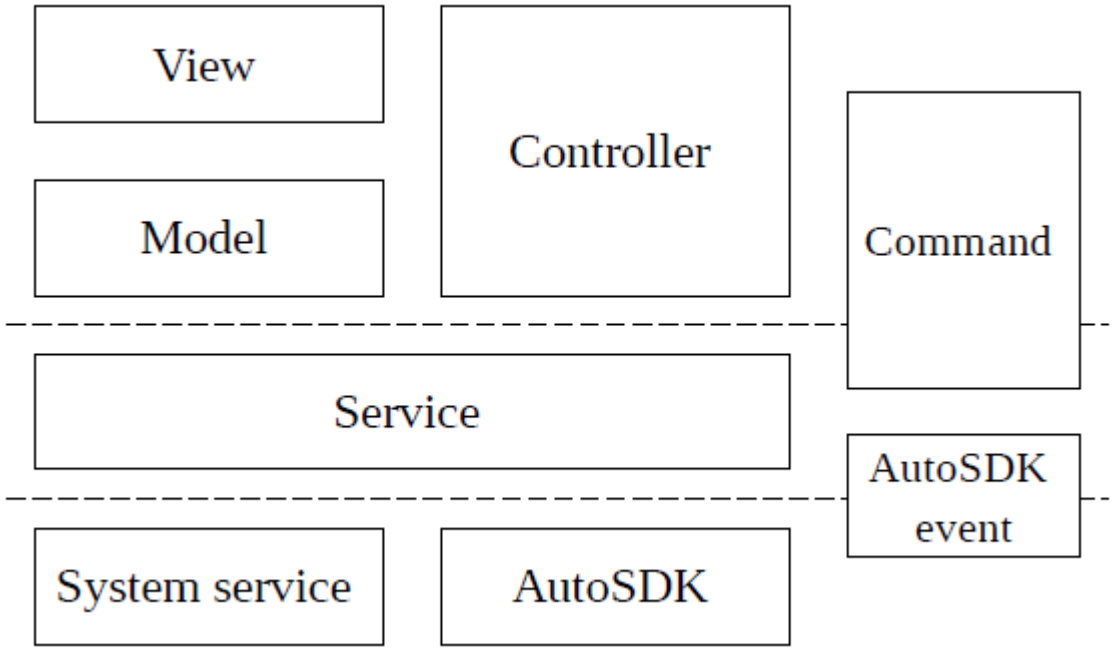
http://en.wikipedia.org/wiki/4%2B1

FRAMEWORK – DEVELOPMENT VIEW



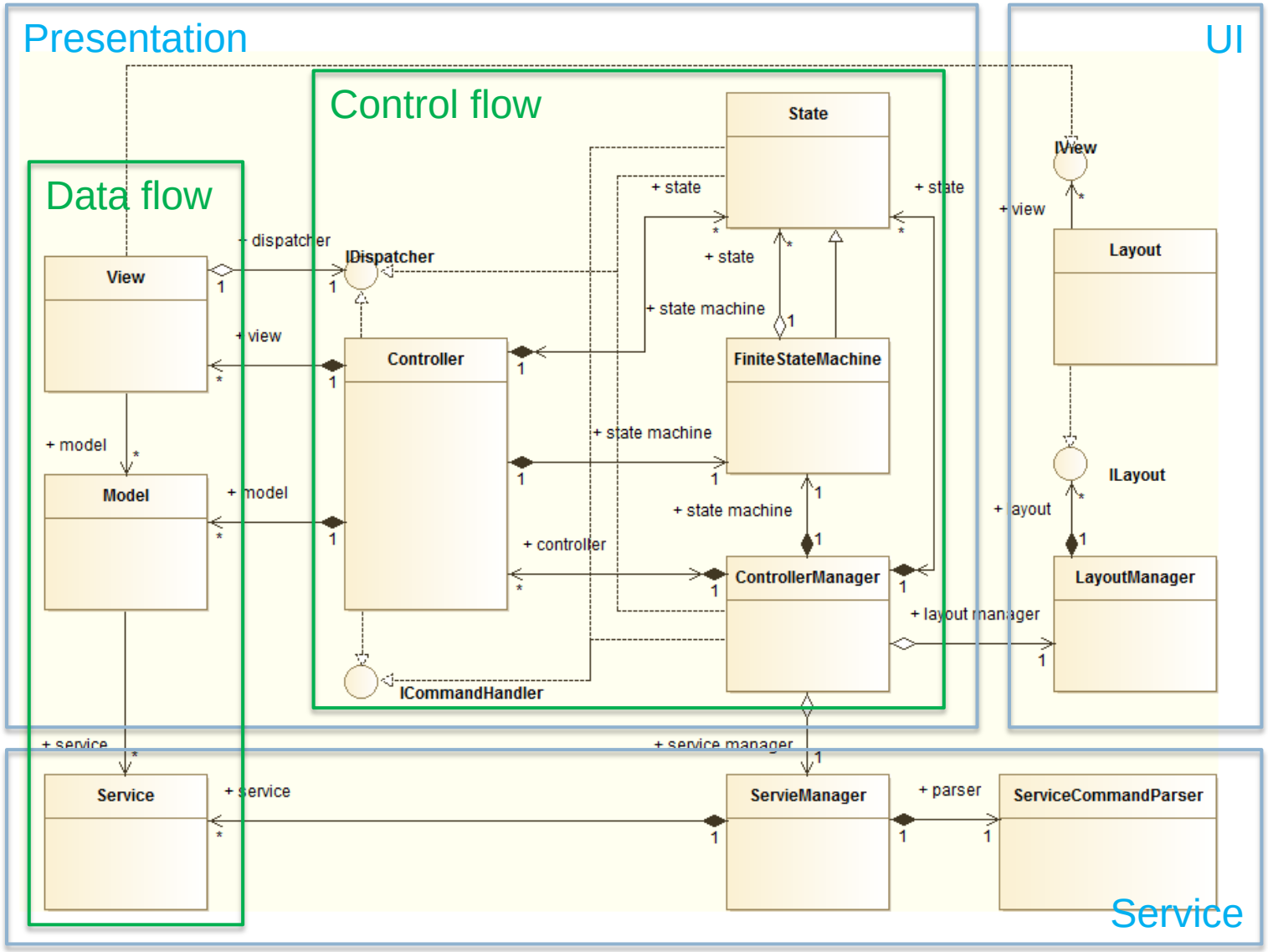
UML Component Diagram

FRAMEWORK – DEVELOPMENT VIEW



Layer Diagram

FRAMEWORK – DEVELOPMENT VIEW



UML Class Diagram – Controller

FRAMEWORK – DEVELOPMENT VIEW

- Control

  ControllerManager → Controller → View/Model
- Data

  Service → Model → View
- Command

  View → Controller → ControllerManager/ServiceManager

- StateMachine is used in Controller and ControllerManager to reduce complexity

FRAMEWORK – DEVELOPMENT VIEW

- 2 Stage StateMachine
  - 1st stage is major flow in ControllerManager, e.g.
    Init → Map → Poi → Navigation
  - 2nd stage is minor flow in every Controller, e.g.
    Within "Poi" state above:
    PoiStateCategory → PoiStateSearchResult → PoiStateDetail
    in PoiController

- Commonly, one State of 2nd stage handles one page (or some related pages) on VDD

FRAMEWORK – PROCESS VIEW

- Control flow
  - View → Controller
  - View → Controller → ControllerManager
  - View → Controller → Service

- Via Command and CommandImpl
  - Command is name + arguments, e.g. “Service.searchPoi&query=kfc”
  - CommandImpl interact with Service or Controller/Model to accomplish a task for Command

FRAMEWORK – PROCESS VIEW

- Data flow
  - Service → Model → View

- Via direct function call from View, or via listener pattern from Service to Model to View

  - For Model, Service is mostly read-only (MapService is an exception, since large portion of it in fact is related to UI)
  - For View, Service is not visible, including SDK data structure

UNDERLYING THE FRAMEWORK

- Render
  - OpenGL ES2
  - First-level UI elements (View) are rendered to texture
  - Decision made for portability (no code change for platform without window compositor, no need to code for Qt platform plugin), but may slightly affect performance (since all UI update is synchronized with OpenGL render)

UNDERLYING THE FRAMEWORK

- Render OpenGL in separate thread
  1. Run eglSwapBuffers in separate thread, keep rendering in main thread
  2. Run rendering and eglSwapBuffers in separate thread

  – Depends on platform/system
  – Hard to do in current framework (kind of trade-off for portability)

UNDERLYING THE FRAMEWORK

- Codec
  - SDK uses UTF-8
  - Qt uses Unicode
  - Other code page (VoiceText uses CP936 for Chinese)
- Set up by QTextCodec::setCodecForCStrings

- *Warning:* the function is deprecated in Qt 5.0
- Possible solution
  - Explicit encoding conversion in Model
  - Require a strict separation of Service (UTF-8 with std::string) and View (Unicode with QString)

GUIDANCE – VIEW

- Global layout is specified in “layout.xml”
- Most layout of the view could be done in “uiconfig.xml”

- Specification in “uiconfig.xml” is undocumented, so at present, please implement by example

- For support of multiple region, “uiconfig_[RegionCode].xml” is added

GUIDANCE – VIEW

1. Inherit from *BaseView*
2. Specify UI and layout in “uiconfig.xml”
3. Call *UiFactory::createWidget* to create QWidget from “uiconfig.xml”
4. Call *findChild* to retrieve necessary UI widgets, and add additional configure for them
5. Call *setUiWidget* to put QWidget to View
6. Make necessary connections

GUIDANCE – VIEW

- Example – PoiOneBoxView.cpp

```
QWidget *widget = UiFactory::createWidget(0, "PoiOneBox");
if (widget)
{
    mSearchEdit = widget->findChild<FixedSuggestionLineEdit *>("SearchEdit");
    if (mSearchEdit)
        mSearchEdit->setPlaceholderText(QObject::tr("Enter Bussiness Name"));

    this->setUiWidget(widget, true);

    mConnectivityDelay.setSingleShot(true);

    QObject::connect(mSearchEdit, SIGNAL(textEdited(QString)),
                     this, SLOT(onTextEdited(QString)));
    QObject::connect(mSearchEdit, SIGNAL(returnPressed()),
                     this, SLOT(onTextCompleted()));
    QObject::connect(mSearchEdit, SIGNAL(itemClicked(const QModelIndex &)),
                     this, SLOT(onSuggestionItemSelected(const QModelIndex &)));
    QObject::connect(&mConnectivityDelay, SIGNAL(timeout()),
                     this, SLOT(onConnectivityDelayTimeout()));
}
```

GUIDANCE – VIEW

- *onShow* is guaranteed to be called after View is set to visible
- *onHide* would be called after View is set to invisible
  - But not guaranteed. If View is destroyed before set to invisible, onHide might not be called.
- Call *sendCommand* to send a Command to Controller
  - Use a Qt slot to receive the response

```
class Transaction {                              // base class for all
public:                                          // transactions
    Transaction();
    virtual void logTransaction() const = 0;     // make type-dependent
                                                 // log entry
    ...
};

Transaction::Transaction()                       // implementation of
{                                                // base class ctor
    ...
    logTransaction();                            // as final action, log this
}                                                // transaction

class BuyTransaction: public Transaction {       // derived class
public:
    virtual void logTransaction() const;         // how to log trans-
                                                 // actions of this type
    ...
};

class SellTransaction: public Transaction {      // derived class
public:
    virtual void logTransaction() const;         // how to log trans-
                                                 // actions of this type
    ...
};
```

NEVER CALL VIRTUAL FUNCTIONS DURING CONSTRUCTION OR DESTRUCTION

Item 9 of *Effective C++*

http://www.artima.com/cppsource/nevercall.html

GUIDANCE – VIEW

- Example – PoiOneBoxView.cpp

```
void PoiOneBoxView::onShow()
{
    if (mSearchEdit) {
        mSearchEdit->setFocus();
        mSearchEdit->clearItems();
    }
}

void PoiOneBoxView::onTextCompleted()
{
    if (mSearchEdit) {
        QString text = mSearchEdit->getLineEdit()->text();
        if (!text.isEmpty()) {
            this->cancelLastSearch();
            this->sendCommand(makeServiceCommand(ActionName::searchPoiAndAddress),
                              ArgumentName::query, text.toStdString(),
                              ArgumentName::errorMessage, QObject::tr("No place found").toStdString(),
                              SLOT(onPoiSearchCompleted(bool)));
        }
    }
}
```

GUIDANCE – STATE

1. Add state name to *EntityName* namespace of the module
2. Inherit from *ControllerState<T>*
3. Insert State to *setupStateMachine* of appropriate Controller
4. In other related State, override *executeCommand*, call *FiniteStateMachine::transitionToState* with the state name, to complete the flow

GUIDANCE – STATE

- *onEnterState* is guaranteed to be called when FSM enter this State
- *onExitState* is guaranteed to be called after FSM exit this State (for either transit to another state, or StateMachine is paused or stopped)
- *excecuteCommand* is called when a command send to the State
  - Return “NoError” to indicate State handles the command, and no further processing is needed

- “back” is automatically handled by Controller

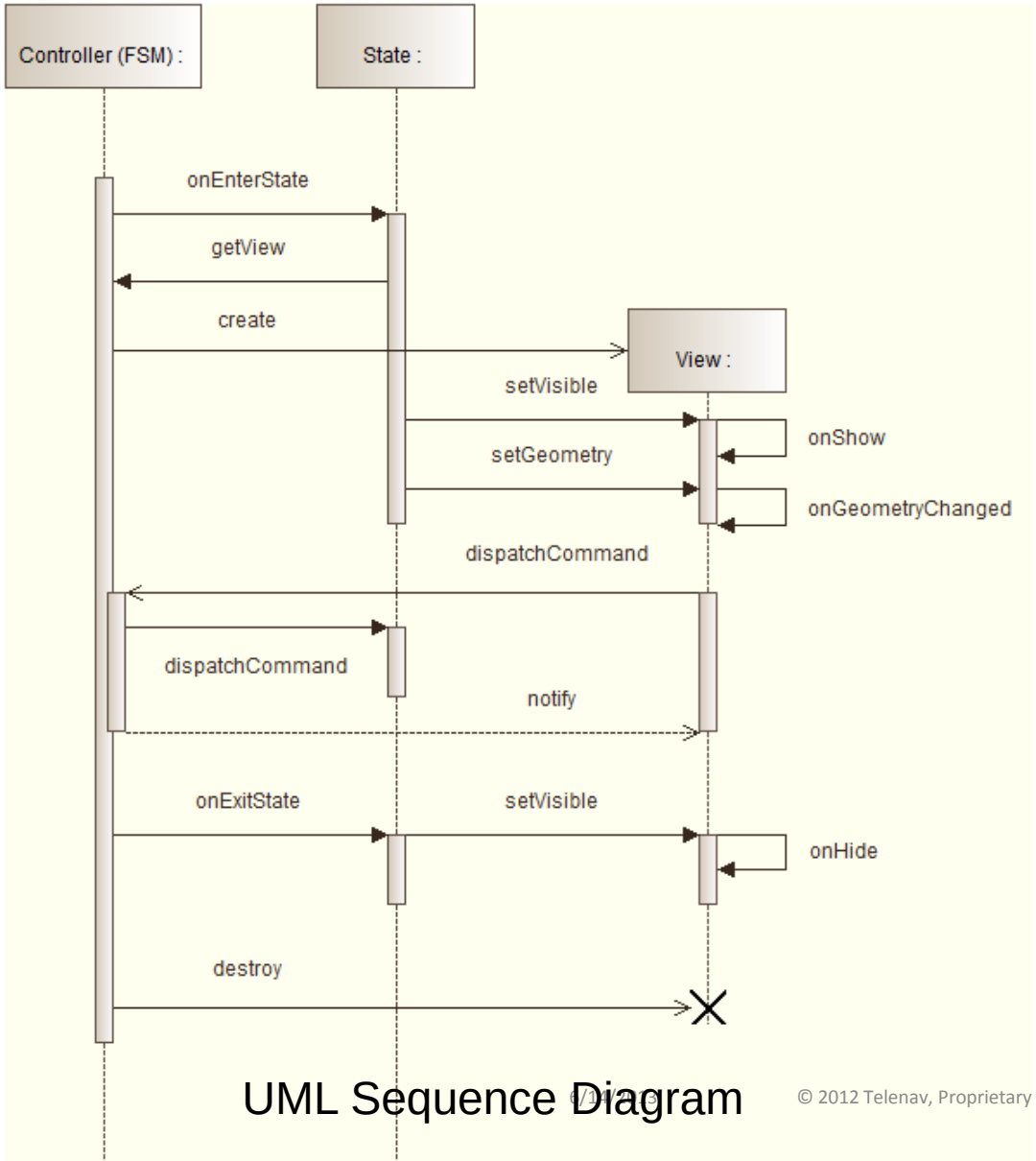
GUIDANCE – STATE

- Example – PoiStateMoreCategory.cpp

```
void PoiStateMoreCategory::onEnterState(const std::string &previousState, bool isBack, const Tn::Even
{
    boost::shared_ptr<TitleBarView> poiTitleView = mController.getPoiTitleView();
    boost::shared_ptr<PoiMoreCategoryView> poiMoreCategoryView = mController.getPoiMoreCategory
    if (poiTitleView && poiMoreCategoryView) {
        poiTitleView->setTitle(QObject::tr("More Categories"));
        poiTitleView->enableBack(true);
        Tn::Ui::LayoutManager::ptr layoutManager = mController.layoutManager();
        if (layoutManager && layoutManager->currentLayout()) {
            layoutManager->currentLayout()->setViewToLayout(poiTitleView, Tn::Ui::ILayout::HeaderView);
            layoutManager->currentLayout()->setViewToLayout(poiMoreCategoryView, Tn::Ui::ILayout::Cent
        }
    }
}

void PoiStateMoreCategory::onExitState(const std::string &nextState, bool isBack, const Tn::Event::Com
{
    mController.cancelCommandByView(mController.getPoiMoreCategoryView().get());
}
```

GUIDANCE – VIEW AND STATE



UML Sequence Diagram

GUIDANCE – CONTROLLER

- In simplest situation, Controller derived from BaseController works as a factory, creating instance of Views, Models, and States

- *onReset* is called when the Controller is deactivated and FSM is reset
  - Release all Views to free memory

- Override *parseCommand* and *executeCommand* for more control

GUIDANCE – COMMAND

- 3 types of Command
  - TypeInstant
    - Instant UI command, mainly for UI event and flow control
    - Usually handled in State
  - TypeService
    - Service command, mainly for asynchronous request and response
    - By default timeout after 30 sec
    - Need a CommandImpl in Service
  - TypeCompositeUi
    - Composite UI command, usually composed of UI manipulation and/or service request/response
    - No timeout
    - Need a CommandImpl in Controller

GUIDANCE – COMMAND

- Examples

  showSearchResult

  change minor state of PoiController to PoiStateSearchResult

  switchSetting&index=0

  update right-side View of SettingController without change state

  Controller.startPoi

  change major state to Poi

GUIDANCE – COMMAND

- Examples

  Service.searchPoi&query=KFC

  request SDK to search for “KFC” near current location

  showMessage&messageText=Success

  show a message box

GUIDANCE – COMMANDIMPL

- The implementation for a Command to actually accomplish a task

- Service uses them abundantly to do request and handle response

GUIDANCE – COMMANDIMPL

1. Inherit from ServiceCommandImpl
2. Override *parseCommand* method to parse the request from command arguments.
3. Override *execute* to access Service to process the request
4. Override *parseCommand* in ServiceCommandParser to attach the CommandImpl to Command with proper name

GUIDANCE – DEBUGGING ON BOARD

- gdb (GNU GDB)

- Useful gdb command
  - “r” run
  - “c” continue
  - “q” quit gdb
  - Ctrl+C break
  - “bt” backtrace
  - “info threads” show all threads

- Material: GNU GDB Debugger Command Cheat Sheet

CODING CONVENTION

- Mandatory
  - Put your name on the class you created. Add your name if you have done large modification on class created by others.
  - Check for NULL pointer for every input parameter or member variable.
  - No raw pointer pass across module, use shared pointer, or reference if object life cycle is clear.
  - No throw and exception. Catch all exception if it comes from other lib.
  - No warning from gcc.

CODING CONVENTION

- Mandatory
  - Use “QObject::tr” in code for literal text strings.
  - Follow Qt's public interface. Use camelCase.
  - Set tab = 4 spaces.

CODING CONVENTION

- Preferable
  - Use const and reference when applicable.
  - Avoid C style casts.
  - First alphabet for member variable should be “m”.
  - If not specified, follow the style of the class you are modifying.
  - Doxygen style comment on basic classes.
  - Unit test for basic classes and complicate logic.

CODING CONVENTION

- Advice
  - Don't repeat yourself.
    This is kind of mandatory. When you want to copy & paste, you probably need to refactor the code so you can move & paste.
  - Program to interface.
    This encourages low coupling.
    When you write unit test, you will want to do this.
  - When implementing, think whether the implement accomplish the definition of the interface.
    Don’t just write an implementation which only satisfy you current need. Developer later uses this interface does not want to read (and then fix) the implementation every time.

TIPS – INFORMATION

- Check “README.txt” in codebase
- Run doxygen in “doc”, and see the generated document

- Check wiki, though it is usually not very informative

TIPS – SUBVERSION

- GUI subversion tools
  - RapidSVN (svn)
  - Meld (diff + svn)

Both have cache of previous commit messages.

- For merge, it is easier to use TortoiseSVN on Windows

- Some subversion command line is still useful
  - “svn up” to update
  - “svn log -l10” to show recent 10 logs

TIPS – BUILD SCRIPT

- “./build.py -t [platform] --skip-update” to skip update, useful when you want to build and package without updating SDK
- “./build.py -t [platform] --skip-build” to skip build, useful when you want to update SDK, but prefer to build in you qtcreator

TIPS – TOOL SCRIPT

- “tools/generate_pro.sh” to re-generate the pro file after change/merge

- “tools/translate.py update” to update strings in code to tr file
  “tools/translate.py release” to release tr file to qm file

TIPS – TOOL SCRIPT

- “tools/updateTelenavConfig.py”
  - Write “TelenavConfig.cfg” in project folder, only with items you want to merge to SDK’s cfg (e.g. DATA_DIR, RuntimeDataPath, etc.)
  - Call the script after SDK update, it will replace the items in SDK with your updated configure

- Write your own script
  Python is easy