A Flexible and Testable Software Architecture

PCaPAC 2012, Kolkata, India
Andreas Beckmann
European XFEL GmbH
andreas.beckmann@xfel.eu
Outline

- Motivation
- Applying Presenter First
  - Technical Background
  - Presenter First
  - Example: Device Server
  - First Experience
- Summary
Motivation

Applying Presenter First
- Technical Background
- Presenter First
- Example: Device Server
- First Experience

Summary
Motivation

- **Testability**
  - System level tests are problematic
    - Require complete system environment
    - Difficult to test corner cases
    - Testing is a manual task
  - Unit level tests are better
    - Require simple test environment
    - Easy to generate all kind of stimuli
    - Testing is automated

- **Flexibility**
  - Allow integration into any machine control system
  - Allow exchange of message layer
Motivation

Applying Presenter First

- Technical Background
- Presenter First
- Example: Device Server
- First Experience

Summary
DOOCS Server
- Interfaces to machine control
- Runs on a Linux host

Device Server
- Interfaces to PLC
- Runs on a Windows host

Data Exchange with ZeroMQ messages
- More convenient to use than plain sockets
- Two schemes:
  ➔ Exchange on request (request/response scheme)
  ➔ Exchange on value change (publish/subscribe scheme)
Pattern

- Presented in 2006 by Atomic Objects
- Variant of MVP design pattern
  - MVP invented by Taligent as a generalized MVC
- Components:
  - **Model**: manages application data and logic
  - **View**: interacts with the environment
  - **Presenter**: represents the behaviour of the application
- Communication over defined interfaces
  - Essential for unit testing => Testability
- No connection between model and view (!)
  - Simplifies exchange of view => Flexibility
Process

- Start with implementing the Presenter
  - Remember: “Presenter represents the behaviour”
  - Analyse the functional requirements
  - Define model and view interfaces
  - Unit test the presenter

- Implement Model and View afterwards
  - Simply implement according to the just defined interfaces
  - Unit test the model
  - View is not unit testable, since it requires an environment
Example: Device Server

- Software Architecture
  - C# application run as a Windows Service
  - 2 main modules
    - Windows Service module
      - Integrates the Device Server into Windows Service Manager
      - Simply starts and stops the Control module
    - Control module
      - Instances MVP
      - Presenter waits on events from Model and View
        » Model sends events when PLC variables change their value
        » View sends events when ZeroMQ messages arrive
Applying Presenter First

- Select a requirement
  
  “The server shall respond to request messages received at the message interface”

- Analyse the requirement
  
  What is the impact on Model and View?
  
  - View indicates incoming message to Presenter
  - Presenter reads message from View
  - Presenter sends the message to the Model to process it
  - Model returns a response message
  - Presenter sends the response message to the View

- Define interface events and methods
Example: Device Server (contd.)

Testing

- Unit level tests
  - Some parts cannot be unit tested
    - Windows Service module, View, part of Model
    - Approx. 20% of the code due to complex View
  - Testing the Presenter
    - Replace Model and View with mocks
    - Mocks verify interface methods calls
  - Testing the Model
    - Wrap (untestable) PLC interface into an adapter
    - Replace adapter with mock
Seen all over Kolkata
Testing (contd.)

- Screenshots
Testing (contd.)

- **System Level tests**
  - Install Device Server
  - Start Device Server
  - Test the system interactively using a DOOCS client
    - `doocspu -t 2 -c XFEL/SASE1/UND01/GAP.SET -d 10.5`
    - `doocsget -c XFEL/SASE1/UND01/GAP`
    - `rpc_test`
First Experience

- Initial effort to set up unit test environment
  - Required for test driven development
  - Essential for refactoring

- Reduced effort for commissioning
  - No extensive debugging on system level necessary
  - In principle “just install and run”

- Stable operation
  - Server runs for long time without failures
  - Reliable communication with DOOCS Server
Motivation

Applying Presenter First
- Technical Background
- Presenter First
- Example: Device Server
- First Experience

Summary
By using Presenter First, we got:

- **Testable Software**
  - Comprehensive use of unit level tests
    - Find bugs early
  - Implementation follows behaviour requirements
    - Reduce code complexity

- **Flexible Software**
  - Message layer code decoupled from application logic
    - Simplify exchange of message layer
  - *Device Server separated from DOOCS Server*
    - Allow use of other machine control servers
Thank you for your attention