Design Patterns and Frameworks for Real-Time Embedded Control Software

Tuomas Ihme
VTT Electronics
Outline

• Logical product-line architecture
  – The Measurement Control architectural pattern for spectrometer control software.
  – The high-level architecture of the Measurement domain
  – The product family of spectrometer controllers

• Product-line architecture designed using an ADL (Architecture Description Language)
  – A ROOM (Real-Time Object-Oriented Modeling) framework for measurement subsystems in the spectrometer controller product line.
  – Design patterns in the development and use of the ROOM framework

• Experience and conclusions
Measurement Control architectural pattern

• Intent
  – The Measurement Control architectural pattern introduces a centralised control architecture for the Measurement subsystem of X-ray spectrometer controllers.

• Motivation
  – Centralised control architecture is very common in the embedded control software of various products, as well as in industrial equipment and scientific instruments. The architecture is also known as master-slave architecture.
  – The complexity of control is centralised on the master. This makes it easy to modify and maintain the software, provided that the system does not get too complex when the distributed control architecture becomes simpler.
  – The master-slave architecture is well suited to hard real-time systems requiring complete timing predictability.
The structure of the Measurement Control architecture

- **MeasurementControl**
- **DataAcquisitionControl**
- **DataManagement**
- **ControlsDataAcquisition**
- **ControlsScienceDataSending**
- **OutputsScienceData**
A message scenario typical of the Measurement subsystem

Tuomas Ihme
The state chart fragment of Data Acquisition Control

- **initialize**
- **DA_Wait_Start**
  - StartMeas
  - **DA_Wait_Stop**
    - StopMeas/MeasurementControl.MeasOK
Semantic properties of Data Acquisition Control

- If the assumptions stated below hold, Measurement Control will eventually receive the MeasOK signal from Data Acquisition Control after sending the StartMeas or StopMeas signal. The assumptions are as follows:
  
  - The StartMeas, StopMeas and MeasOK signals are not implicitly consumed by the superclasses.
  - The transmission of the signals between Measurement Control and Data Acquisition Control is reliable.
  - The MeasOK signal will always be sent before the state DA_WAIT_START.
  - The state DA_WAIT_START will eventually always be reached.
Redefinition of Data Acquisition Control

- The embedded state chart fragment will be supplemented by additional statements for acquiring spectrum data from an array detector and sending spectrum signals to Data Management. The above property still holds, if the pattern is redefined by the introduction of additional statements, which do not disrupt or bypass the thread of control from predefined input to predefined output statements. However, the thread of control from the StopMeas input signal will be bypassed by threads of control that end in the MeasOK output statement and the state DA_WAIT_START.

- During observation in the DA_WAIT_STOP state, the thread of control stays in the polling loop of the hardware/software interface for several hours. The polling will have to be continuous to meet the requirements set on the data collection speed. For the simulation purposes of the system, continuous polling will be replaced with periodic polling using a timer-triggered transition from the DA_WAIT_STOP state.
The state chart fragment of Data Management

**Initialize**
- **SendNo**/
  - Environment.SendNoBlocks(blocksUsed)
- **ClearData**/
  - sD = Clear(sD); blockNo = 1; blocksUsed = 1;

**DM_Wait**
- [blockNo <= blocksUsed]
  - **SendBlock**/
    - Environment.SendNextBlock(sD(blocksNo)); blockNo = blockNo + 1;
  - [blockNo > blocksUsed] /
    - MeasurementControl.SendOK; blockNo = 1;
Semantic properties of Data Management

- The SendNoBlocks signal has to be sent to the environment with the number of blocks after the SendNo signal from Measurement Control. The SendNextBlock signal has to be sent to the environment with the next data block after the SendBlock signal from Measurement Control.

- Property 1: If the assumptions stated below hold, then Measurement Control will eventually receive the SendOK signal from Data Management after sending enough SendBlock signals. The assumptions are as follows:
  
  - The ClearData, SendNo and SendBlock signals are not implicitly consumed by the superclasses.
  - The indexes and counters in Data Management are properly initialised and modified.
  - The SendOK signal will always be sent after receiving the SendBlock signal when the last science data block has been sent.
  - The state DM_WAIT will eventually always be reached.
Redefinition of Data Management

• The embedded SDL fragment will be supplemented by additional transitions and statements for saving the science data sent by Data Acquisition Control. Property 2 determines the allowed redefinitions:

  − Property 2: Property 1 still holds, if the pattern is redefined by the introduction of additional transitions from the DM_WAIT state for saving the science data associated with spectrum signals from Data Acquisition Control. The indexes and counters in Data Management must be properly initialised and modified in the additional transitions. The state DM_WAIT must eventually always be reached at the end of the additional transitions.
The high-level architecture of the Measurement domain

- **Measurement Control**
  - BeginGroundContact()
  - FinishObservationTime()
  - ReadScienceData()
  - StartObservationTime()

- **Data Acquisition Control**
  - StartMeasurement()
  - StopMeasurement()

- **Data Management**
  - ClearData()
  - SaveData()
  - SendData()

- **Energy Data Acquisition Control**

- **SEC Data Acquisition Control**

- **Controls Data Acquisition**

- **Controls Science Data Sending**

- **Stand Alone Control**

- **Coordinated Control**

- **Data Management**

- **Energy Data Management**
The product family of spectrometer controllers

- **SpectrometerController**
  - **Product Type**: EnergyController
    - **Product**: EGYController
  - **Product Type**: SECController
    - **Product**: Autonomous_SECController
    - **Product**: SECwithEGY_Controller
A snapshot of the spectrometer controller ROOM framework
The design architecture of the measurement subsystem
Protocol classes and the consistency of connections

- ObjecTime uses protocol classes to check automatically the consistency of architectural connections
- The Data Management Commands protocol class is used to check the consistency of the connection between the measurement Control and data Management actor objects.
The Spectrometer Controller product line architecture has been built integrating several architectural styles and patterns hierarchically and simultaneously:

- The Layers pattern has been applied to the control structure of spectrometer controllers.
- Measurement controller pattern: the main architectural pattern of the measurement subsystem
- The recursive control pattern separates functional interfaces and control interfaces
- Strategy Pattern for the documentation of reuse strategies
Strategies for reusing the ROOM framework

- The selection of concrete spectrometer controller models that have the same type of interface to the Ground Station. The General Spectrometer Controller actor hides the differences between concrete controllers.
- The selection of an existing controller class for designing a new version of the measurement subsystem.
- The selection of the Abstract Spectrometer framework model for designing a new framework model. For example, the EGY and SEC framework definition models are designed by specialising definitions in the Abstract Spectrometer Features package.
The selection of different Spectrometer Controller system models

Strategies for the selection of controllers

- GroundStation
- Spectrometer_Controller
- GeneralSpectrometer_Controller
- EGY_Controller
- SEC_Controller
- SECwithEGY_Controller
The selection of concrete control components

Strategies for the selection of control

GroundStation

Measurement_Subsystem

Context

AbstractStrategy

ConcreteStrategy

Measurement_Control

Client

StandAlone_Control

Coordinated_Control
The selection of concrete data acquisition strategies

Strategies for the selection of data acquisition

- Client
- Context
- AbstractStrategy
- ConcreteStrategy

ConcreteStrategy

EGY_DataAcquisition

SEC_DataAcquisition

GroundStation

1

Measurement_Subsystem

1

DataAcquisition

Tuomas Ihme
Component System Dependencies

- Dependence is at its highest at the top of the diagram and at its lowest at the bottom.
Conclusions

- The ROOM language and the ObjecTime Developer tool proved suitable for the development of the product line architecture and core design components.
- This experience advocates that a production system should have a systematic approach to identify and use patterns for product line architectures:
  - The Spectrometer Controller product line architecture was built integrating several architectural styles and patterns hierarchically and simultaneously.
  - The strategy pattern appeared to be useful in the documentation of the reuse strategies of the ROOM framework.
- Special means are needed for constraining the free modifiability of the framework:
  - the application-specific configuration rules of components
  - constraining the specialisation of components
  - the documentation of designed variability and dependencies between components
- The benefits offered by simulation and validation of the framework are convincing:
  - The development of ROOM frameworks is worth considering, even if the final coding or module testing has to be done manually.