OO Frameworks for On-Board Satellite Software

Wolfgang Pree
Dept. of Computer Science
University of Constance (Germany)
pree@acm.org

Contents

- Overview
- Design of selected subsystems
  - Telemetry Manager
  - Closed-Loop Controller Manager
- Reuse model, resource requirements
- Framework development methodology
  - Framelets
  - Implementation cases
- Project status & future activities
Overview

Background

R&D contract with the European Space Agency (ESA) to study SW reuse in Attitude and Orbit Control System (AOCS)
Selected Software Technology

Software Frameworks: collection of components with pre-defined interactions defining a reusable architecture optimized for a certain domain.

FWs define a reusable architectural skeleton (unshaded in figure) that is customized for a specific application by plug-in components (gray components in the figure).

Our Aim: design and prototype a sw framework for the AOCS

Reuse Approach: The RTOS Model

RTOS’s are examples of reuse in real-time field → inspiration for AOCS f/w

Task management separated from task implementation through an abstract i/f.
The RTOS Example and the AOCS

- The RTOS example shows that the management of some functionalities – like task scheduling – can be packaged in reusable components.

- For typical AOCS functionalities such as:
  - telecommand management, telemetry management,
  - closed-loop controller management, failure detection management, failure recovery management, sensor\actuator management, etc

  Can application-independent (and hence reusable) functionality managers be constructed?

Reuse Approach for the AOCS

- Divide the AOCS into functionalities: TM management, TC management, unit management, FD management, FR management, etc.

- For each functionality:
  - Define an abstract interface separating the functionality management from its implementation
  - Build a reusable functionality manager component (core component)
  - Build reusable component providing default implementations of recurring functionality implementations (default components)
What the AOCS FW Offers

- AOCS-Specific Design Patterns
  - Solutions to commonly occurring architectural problems in AOCS systems

- Core Components
  - Binary modules encapsulating application-independent functionalities (e.g., closed-loop controller manager)

- Default Components
  - Binary modules encapsulating common AOCS functionalities (e.g., PID controller)
  - Abstract Classes/Interfaces
  - Definition of external interfaces to be implemented by default components

Some AOCS Design Patterns - 1

- The AOCS is made up of independent framelet components that cooperate by exchanging data...
  - Shared memory design pattern modelled on JavaSpaces
JavaSpaces architecture overview

Some AOCS Design Patterns - 2

- AOCS components need to monitor each other to detect failures and to synchronize their behavior …
  - monitoring mechanisms modelled on JavaBeans property monitoring: direct monitoring, conditional monitoring, monitoring with notification
- AOCS components exhibit mode-dependent behavior …
  - Modified version of the “Strategy Pattern” from Gamma et al
AOCS Strategy Pattern

Some AOCS Design Patterns - 3

- AOCS need to implement data processing chains ...
  - Block/Superblock mechanism mimicking the similarly named concepts in MatrixX/Xmath

- Data from sensors and to actuators need to go several processing stages ...
  - Definition of UnitFunctional abstract interface to be implemented by each unit processing component
Some AOCS framelets

Telemetry Management

Example

Identify abstract operations required to handle telemetry:

- `writeToTm()`: object writes its own state to the TM stream
- `setTmFormat(newFmt)`: set the TM format to newFmt
- `getTmImageLength()`: return the length (in bytes) of the object's TM image

Define an abstract interface for telemetry operations:

```text
Telemeterable
```

- `writeToTm()`
- `setTmFormat(newFmt)`
- `getTmImageLength()`
Telemetry Manager

TelemetryManager

XAxisController
- WriteToTm
- SetTmFormat
- GetTmImageLength

FineSunSensor
- WriteToTm
- SetTmFormat
- GetTmImageLength

TM Manager maintains a list of pointers to objects of type Telemeterable and periodically processes them using the Telemeterable operations:

```c
Telemeterable* list[N];
for ( i=0; i<N; i++ )
{
    list[i]->setTmFormat(fmt);
    size = list[i]->getTmImageLength();
    if (image fits into TM buffer)
        list[i]->writeToTelemetry();
    else
        . . . // error!
}
```

Closed-Loop Controller Manager

ControllerManager

XAxisController
- AcquireReference
- ApplyControl
- SetOpenLoop
- SetClosedLoop
- IsStable

YAxisController
- AcquireReference
- ApplyControl
- SetOpenLoop
- SetClosedLoop
- IsStable

Controllable* list[N];
for ( i=0; i<N; i++ ) do
    if ( list[i]->isStable() )
        list[i]->acquireReference();
        list[i]->applyControl();
    else
        . . . // error situation!
Conceptual AOCS Architecture

- Red blocks are application-independent and reusable
- Blue blocks tailor generic architecture to needs of a specific application
- Each FM defines an abstract interface
- A component may contribute to tailoring several FM's

Multiple Interface Implementation

Java-model of multiple inheritance is used (safe!)
Summary of Reuse Model

- Abstract interfaces decouple functionality management from functionality implementation
- Core components encapsulate reusable functionality managers
  Functionality managers do not perform actions upon objects, rather they ask objects to perform actions upon themselves
- OO language is required, in particular to support multiple abstract interfaces
- AOCS framework captures many aspects of any embedded control systems

Prototype Implementation

- Prototype implementation language: C++ (GNU compiler)
  - Any OO language can be used
  - Ada95 was considered but discarded due to poor support for MI
- No dynamic memory allocation
  - Non-trivial objects are created at initialization and never destroyed (no dangling pointers)
- No exceptions, no run-time type identification, …
  - Error situations are handled through creation of event objects in shared memory areas
- Target processor: ERC32 + RTEMS operating system
  - SPARC processor qualified for use in space by ESA (megabytes memory, ~ 10 MIPS)
Resource Requirements

- Timing requirements for "empty" functionality managers: 0.2 ms @ 14 MHz per AOCS cycle
- This is the overhead introduced by the framework infrastructure
- Typical AOCS cycle durations are 50-500 ms
- Memory requirements for functionality managers: 43 kB (code) + 19 kB (data)
- AOCS Prototype requirements (inclusive of RTEMS and C++ run time systems but with some modules missing):
  - 1 AOCS cycle in 3.9 ms
  - 245 kB (code) + 92 kB (data)

AOCS prototype not really representative of "real" AOCS

Framelets & implementation cases
Product Line Complexity

- Two forms of complexity
  - *Quantitative complexity*: large number of design constructs
  - *Qualitative complexity*: high level of abstraction

- Two forms of complexity → Two conceptual tools
  - *Framelets* address quantitative complexity
  - *Implementation Cases* address qualitative complexity

The framelet concept
Framelet Features

- Framelets are units of design that address, as self-standing units, specific problems within the framework, in particular the modularization of large frameworks.
- The overall framework is obtained by combining the framelets.
- Framelet features:
  - Small size
  - No execution control assumptions
  - Self-standing

Framelet constructs

F/Ls export either horizontally (towards other F/Ls) or vertically (towards applications) three types of constructs:
- Abstract interfaces
- Design patterns
- Interface implementations (as components)
Framelets and subsystems

Space of Objects
- Object
- S/S Boundary

Space of Interfaces
- Interface
- F/L Boundary

Space of Des. Patt.
- Design Pattern
- F/L Boundary

Subsystems partition the space of objects in an application
Framelets partition the space of interfaces and design patterns in a product line

Heuristics for F/L identification

The following heuristics were found useful in the AOCS project
- Map clusters of related application requirements to a F/L
- Build a F/L around one (or a related set of) hot-spot(s)
- Map a task to a F/L (for embedded applications)
- Map an abstract use case to a F/L
Framelet design description

Implementation cases
Use of Implementation Cases

- Continuous check of FL design adequacy
  ICs are **initially defined when FL design commences** and are worked out in parallel to the FL design. They are thus used to verify the adequacy of the abstractions proposed by the FL.

- Product line specification
  ICs describe how the FL is used (in the same sense in which use cases describe how an application is used)

- Product line user manual = cookbook
  At the end of FL development, ICs are available as commented pseudo-code illustrating how the FL is used (cookbook-type recipes).

IC example

**Objective**
Build a telecommand to perform an attitude slew manoeuvre

**Description**
Attitude slews are normally started by a ground command (telecommand). This implementation case shows how to build a telecommand to perform an attitude slew manoeuvre.

**Framelets**
- Telecommand Framelet
- Manoeuvre Management Framelet

**Framelet Constructs**
- Telecommand Interface
  (exported from Telecommand Framelet)
- AocsManoeuvre Interface
- ManoeuvreManager Component
  (exported from Manoeuvre Management Framelet)

**Framelet Hot Spots**
- Telecommand Definition
  (exposed by Telecommand Framelet)

**Related Implementation Cases**
- Attitude Slew Manoeuvre Implementation Case
  (this implementation case uses the component built in the attitude slew manoeuvre implementation case)

As design progresses, pseudo-code is added to the IC
IC & design process

Product Line Specification

System Concept Def.

Define IC for each F/L

Implementation Case Def.

Define IC for each construct/hot-spot exported by F/L

Framelet 1 Concept Def.

Framelet N Concept Def.

Progressive Work Out of Implementation Cases

Framelet 1 Arch. Design

Update Framelet design

Framelet N Arch. Design

Integrate Framelets

Concluding remarks

- F/Ls and ICs were successfully tested in the AOCS project
  - Use of F/Ls made design more manageable and will make it easier to extend the FL to other application domains (some F/Ls can be carried over unchanged to other domains)
  - ICs were the main source of design changes in the AOCS project
- F/Ls and ICs are being used as the core of a complete methodology for FL development
Project status & future activities

Future Activities

- Contract with ESA set up to use the AOCS framework to develop and test AOCS for the Proba satellite
  - Proba is mini-satellite to be launched by ESA in 2001 as a technology demonstration mission
- Evolution of the AOCS F/W to form a generic embedded control system framework
- Contract with ESA under negotiation to port the AOCS framework to a real-time version of Java
  - Java is a “natural” implementation medium for sw frameworks
AOCS Framework Status

- AOCS Framework project to be completed in December 2000
- Framework code and design documentation to be made publicly available on our web site