Real-Time UML Workshop

Bruce Powel Douglass
About the Author

• Chief Evangelist for Telelogic
• Author of
  • *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns* (Addison-Wesley, 1999)
• Teaches and consults on OO, project management, real-time and related topics
• Advisory Board
  • Embedded Systems Conference
  • UML World Conference
• Contributor to
  • UML v 1 and 2
  • SPT, SysML, UPDM Profiles
• (Former) co-chair of OMG RTAD Work Group
What you will learn

• How to apply UML to real-time and embedded development projects, emphasis on
  – Object Analysis
  – Object Design

• Basic concepts of Model-Driven Architecture (MDA)

• Basic concepts of design patterns

This is a hands-on class! Prepare to get your hands dirty!

Prerequisites

• Basic knowledge of and experience with UML

• Familiarity with Real-Time UML helpful but not required

• Note: this class is appropriate for either software or systems engineers as well as technical leads
Hey! Didn’t CASE tools fail in the 80s?

• YES!
  – The application of CASE tools failed to realize their hype

• WHY???
  – The models were unverifiable
  – The tools were divorced from the code
    • “Dual maintenance” is not a viable option
  – Testing came at the end and only the code was tested
  – The processes that encouraged them didn’t emphasize the right things, such as
    • Design for testability
    • Validatable approaches
    • Active early risk reduction
Take Home Lessons

• Pretty pictures are not enough
  – Semantic depth is necessary
  – Mapping in obvious ways to the code is required

• Dual Maintenance is not a viable option
  – It is error prone and expensive
  – Eventually models will fall out of sync with the code

• Testing is too hard and too expensive
  – Models that can’t be tested must be tested by “inspection” or by testing the code
  – You can’t test things that don’t execute

• Testing can’t happen at the end
  – Need the ability to test all the way through
  – This is the basic premise of agile methods

UML and MDA

- **CIM (Computationally Independent Model)**: Use cases clustering requirements and detailed with sequence, state and activity diagrams
- **PIM (Platform Independent Model)**: Formal specifications (in UML) of the structure and function of the system that abstracts away technological details.
- **PSM (Platform Specific Model)**: Refines PIM by adding technical details, such as:
  - Middleware
  - Operating system
  - Network
  - CPU characteristics
- **PSI (Platform Specific Implementation)**: Source (and compiled) code produced from the model, running on the desired target platforms

---

© Telelogic AB

v 1.1 (c) Telelogic 2007 Do not copy or reproduce without permission
UML and MDA

Using the Harmony Process for MDA
Key Concepts

- The Harmony Model lifecycle overview
- Systems Engineering Phases
  - Requirements Capture
  - Systems and Architecture
  - Systems engineering hand-off
- Subsystem Iterative Development Cycle (IDC)
  - Analysis
  - Design
  - Implementation
  - Test
  - Increment Review

Harmony Timescales

- **macrocyle**
  - Stakeholder focus
  - Months

- **microcyle**
  - Team focus
  - Weeks

- **nanocyle**
  - Personal focus
  - Days

- Deployment
  - Optimization
  - Secondary Concept
  - Key Concept
  - Project Plan

- Iteration Plan

- Demo-able or Shippable Build

- Revision of Work Items

- Project Work Item
Incremental Software Development

- Prespiral Planning defines the properties of all acceptable solutions (PIM)
- Analysis specifies a particular “optimal” Solution (PSM)
- Design translates model into executable code and constructs working architectural pieces of prototype
- Implementation integrates architectural pieces into prototype and validates against prototype mission followed by assessment and replanning
- Testing verifies the model by testing and validating prototypes
- Control Project

Incremental Development

- A project is developed as a series of small subprojects
  - Each subproject realizes and validates a set of requirements
  - Over time, the system becomes increasingly complete
  - Primary advantage is that strategic defects are identified and corrected much earlier and at much lower cost
  - Each microcycle is normally done in 4-6 weeks (your mileage may vary)
Prototypes are the Primary Scheduling Points

Every Microcycle Mission Has a Mission

- A microcycle mission is a combination of
  - Requirements architected, designed, implemented, and tested
  - Risks reduced
  - Target platforms supported
  - Existing defects repaired
  - That mission is *realized* in a prototype

- Prototype is typically organized around
  - A small set of use cases
  - A small set of risks to be mitigated

- Every microcycle produces at least two artifacts
  - The prototype
  - List of defects to be repaired in the next microcycle
A note about Prototypes

• An incremental prototype is a working version of the system that may incomplete.
  – Incremental prototypes contain code that will be shipped to the customer
  – One of the prototypes produced is the “completed system”
  – We do not mean “throw-away code”

• Prototypes have a mission
  – Some set of features (e.g. use cases)
  – Reduction of some risks
  – Prototypes are tested against this mission

• Idea is to remove strategic defects early when it is inexpensive to do so

Incremental Construction

Prototype 1
Name: ‘Hello World’
Mission:
- Subsystem Architecture
- Data Acquisition
- Basic UI for monitoring

Prototype 2
Name: ‘Revision 1’
Mission:
- Basic Distribution Architecture
- Data Waveform Display
- User setting of control values
- Data Logging

Prototype 3
Name: ‘Customer Review Prototype’
Mission:
- Reliable Distribution Architecture
- Reliable transport protocol
- Sockets
- Closed Loop Control
- Built-in Test
Key Concepts

- **Analysis**
  - Prototype specification
  - Object analysis for Platform-Independent Model (PIM)
- **Design**
  - Architectural design for Platform-Specific Model (PSM)
  - Mechanistic design for PSM
  - Detailed design for PSM
- **Implementation**
  - Translation
  - Unit Test
  - Model review
- **Test**
  - Integration
  - Validation
Workflow for a single incremental cycle

Analysis in Harmony
Analysis Activities

• Prototype Definition
  – Selects (General Form)
    • the use cases to be designed/implemented/validated in the prototype
    • the risks to be reduced
  – Specifies (SW-Only)
    • details the use case requirements with requirements elements (SysML), sequence diagrams, statecharts, activity diagrams

• Object analysis
  – Identifies and characterizes object and classes essential to the system
    • Identify objects and their structural relations
    • Define essential behavior of objects with statecharts
    • Shows how the collaboration meets the requirements via elaboration of the use case scenarios
  – This object analysis model is known in MDA as the Platform-Independent Model (PIM)

PIM Collaborations Realize Use Cases
Platform-Independent Models (PIMs)

Key Concepts

- What constitutes a PIM
- Constructing PIMs
- Reusing PIMs
What is a PIM?

- Platform-Independent Model (AKA “Essential Model” or “Analysis Model”)
  - Contains characteristics and properties that are essential for correctness
- Does not contain
  - Realizing technology
  - Supporting infrastructure
  - Middleware
- Example
  - A microwave oven must contain
    - Microwave emitter (or it’s not a microwave oven!)
    - Power control mechanisms
    - Front panel display
    - User interface
    - Timing mechanisms

Constructing PIMs

- A PIM contains a set of collaborations, each of which realizes a system use case
- Each collaboration contains a set of object roles connected via relations that realizes a use case
- This is done in the object analysis phase of the incremental development cycle
- Avoid adding in technology-specific details
  - Middleware
  - Communications protocols
  - OS
  - CPU/Hardware Platform
  - Design patterns
Reusing PIMs

• Targeting a platform by elaboration
  – Process Overview
    • Make copy of PIM
    • Elaborate in the design phase by adding technology and infrastructure
    • Elaborate by incorporating architectural and mechanistic design patterns
  – Pros
    • Most common and natural approach
    • Simple to manage and add technology
    • May be validated by execution or simulation on the host or target
    • Highly flexible
    • Relatively easy to support “model-code associativity”
  – Cons
    • Defects discovered in the PIM result in “dual maintenance”
    • Difficult to automate

Reusing PIMs

• Targeting a platform by translation
  – Process Overview
    • Construct a translator
      – Translator is platform and technology-specific
      – Translator incorporates architectural and mechanistic design patterns
    • Translate PIM to a PSM in the design phase by applying the translator
  – Pros
    • If the PIM has a defect, then only the PIM need be changed
      – No “dual maintenance”
    • May be validated by execution or simulation on the host or target
    • Simple to automate
  – Cons
    • Translator may be difficult to create and manage
      – Especially if code readability is important
    • Very difficult to get a bi-directional translator
      – Difficult or impossible to “back in” changes made to the code directly
    • May be less flexible
Reusing PIMs

- We recommend both approaches simultaneously
- Model Translator
  - Model compiler encapsulates technologies
    - CORBA, COM, DCOM
  - Code generation supports infrastructure in a general way
    - State machine execution
    - Timer management
    - OS Services
  - Rule-based code generation and properties allow customization and highly readable code
- UML elaboration
  - Simple to apply design patterns in a manual way while using the model compiler
  - Supports dynamic model-code associativity
    - "Code is a view of the model"

What is a PSM?

- Platform-Specific Model (AKA “Design Model”)
- PSM must support the semantics of its parent PIM
- PSM incorporates design technology, infrastructure, and patterns about
  - Subsystem organization
  - Distribution architecture
  - Safety and reliability architecture
  - Concurrency and resource management
  - Deployment architecture
Constructing PIMS

- As we shall see *real soon now*, PIMs are constructed iteratively:
- PIMs are constructed a use case at a time
  - For each use case in the specific spiral
    - Construct the collaboration
- Constructing the Collaboration
  - Start with a single class
  - Specify some relevant class properties, e.g.
    - Attribute
    - Operation
    - State-behavior
  - Get that to work by itself via execution
  - Does this meet all the needs? (Can I replicate all the requirements scenarios?)
  - If not, then Identify the next class and repeat

It is important to construct a PIM that is *high quality and correct*. This is done by applying object identification strategies to find the elements of the collaboration and adding them one at a time, validating the work via execution as you proceed.

Example: Elevator Scenario (1 of 3)
Creating the PIM

• Creating a PIM is a step-by-step process
  – It is important to validate after each small incremental step!
• In the slides following, the steps shown are captured as class diagrams as the collaboration becomes increasingly complete
• In this example, we will walk through the scenario, adding objects, classes, relations and features in order to construct the PIM
  – There are many more scenarios (not shown for space reasons) that will result in additional model elements
• In actual use, the Harmony process provides several different object identification strategies, as shown on the next slide, but only one will be used in this example
Step 1: Requesting the elevator

ElevatorStateType elevator:
int floor;

for i=0;i<NELEVATORS;i++:
    if (ELEVATOR[i].queryStatus(elevatorState, pos) == Idle):
        if (elevatorState == Idle):
            select(floor);
        else:
            // end if
    // end if

OMBoolean success = dispatch(e10, floorID);
if (success):
    theHallButton(floorID) = backlight();

for i=0;i<NELEVATORS;i++:
    iteElevator[i].gotoFloor(floorName);
return TRUE;

Button
- floorBtn
- buttonUp
- buttonDir
- pressQ void
- backlightQ void
- discardLightQ void

1: NFLOORBUTTONS
themeBtn

1: NFLOOR

ElevatorController
- buttonUp
- buttonDir
- select(floorID)
- acceptElevatorRequest(floorID, upDown, DirectionType)
- dispatch

ElevatorStateType
- currentState
- direction
- position

queryState(elevatorState, position, void)

gotoFloor(floorID); OMBoolean
Step 2: Make classes reactive

- Button state machine
- Elevator state machine

Step 2: Class Diagram

- Step 2
  - Upon reflection, the decision is made to go to reactive objects.
  - The Button state machine is added and the operations are made into event receptions.
  - The Elevator state machine is added (but only enough to support the steps so far)
Executing Step 2

Step 3: Closing the Doors
Step 3: Closing the Doors

Doors need to close. We need to support timing to close the doors, ability to control a motor with a movement profile, and to engage and disengage with the doors on the floor.

In other scenarios, we’d add the “safety chain” that ensures that the doors on all halls must be closed before the elevator can move, what happens when there is an obstruction, etc.

Elevator state machine

MovingUpState

ClosingDoors

Stopped_Open

Stopped_Closed

OpeningDoors

Moving

MovingUpState

MovingDownState

IdleState

moveUP

moveDOWN

[newDirection (params->floorID) == direction] success = TRUE;
if (direction == UP) GEN(MoveUP);
else GEN(MoveDOWN);

Elevator state machine
Step 3: Closing the Doors

ElevatorDoor state machine

Interlock state machine
Step 3: Closing the Doors

Object Analysis Workflow

Strategies:
- Nouns in problem statement
- Actors or Causal Agents
- Services to be performed
- Transactions
- Physical devices
- Key concepts
- Persistent Data
- Players in scenarios
Identify the Nouns

- Find nouns and noun phrases in the problem statement
- Provides a first-cut list of potential classes and objects
- Can be problematic to apply to large scale problem statements with an incremental (per use case) approach
- Discovers
  - objects of interest
  - uninteresting objects
  - attributes

What do I know?
What do I do?
What are my responsibilities?

Example: Identify the Nouns

A software system must control a set of 8 elevators for a building with 20 floors. In each elevator, there are a set of buttons corresponding to each desired floor and a current floor indicator panel above the door. On each floor there are 2 request buttons, “Up” and “Down.” Additionally, each elevator shaft on each floor has an indicator showing the floor position of that elevator and a direction indicator. The system shall respond to an elevator request by sending the nearest elevator either idle or going in the proper direction.

- Which are interesting objects?
- Are any attributes identified?
- Are any uninteresting objects identified?
Find the Causal Agents

- Look for objects which
  - produce or control actions
  - produce or analyze data
- Such causal agents may be named or merely implied

Example: Find the Causal Agents

A software system must control a set of 8 elevators for a building with 20 floors. In each elevator, there are a set of buttons corresponding to each desired floor and a current floor indicator panel above the door. On each floor there are 2 request buttons, “Up” and “Down.” Additionally, each elevator shaft on each floor has an indicator showing the floor position of that elevator and a direction indicator. The system shall respond to an elevator request by sending the nearest elevator either idle or going in the proper direction.

- Are any causal agents named?
- Are any causal agents implied?
Services to Be Performed

- Service providers are usually passive servers
- May provide
  - control e.g., light switch
  - data e.g., measurement server
  - storage e.g., database

Example: Services to Be Performed

A software system must control a set of 8 elevators for a building with 20 floors. In each elevator, there are a set of buttons corresponding to each desired floor and a current floor indicator panel above the door. On each floor there are 2 request buttons “Up” and “Down.” Additionally, each elevator shaft on each floor has an indicator showing the floor position of that elevator and a direction indicator. The system shall respond to an elevator request by sending the nearest elevator either idle or going in the proper direction.

- Can you find any servers?
- Who are their clients?
Real-world Items

- Model might represent the interface to the item
  - Create cause and effects through interface
- Model might represent a resource to be managed or represented
  - keep operations within appropriate constraints
- Model might be a simulation

Physical Devices

- Very common approach in embedded systems
- Devices are often modeled as classes and objects
- ECG Example
  - Heart
  - A/D converter
  - switch
  - Button

Physical devices are modeled either as interfaces to the actual hardware (actual behavior is in the hardware, the software class provides a means for objects to talk to it) or, less commonly, simulations (simulates actual hardware behavior)
Example: Physical Devices

A software system must control a set of 8 elevators for a building with 20 floors. In each elevator, there are a set of buttons corresponding to each desired floor and a current floor indicator panel above the door. On each floor there are 2 request buttons, "Up" and "Down." Additionally, each elevator shaft on each floor has an indicator showing the floor position of that elevator and a direction indicator. The system shall respond to an elevator request by sending the nearest elevator either idle or going in the proper direction.

- Identify all the physical devices named or implied
- Should all these be modeled in your system?

Key Concepts

- These are the fundamental abstractions in the domain
- These concepts often have no physical manifestation
- Examples
  - UI Domain Window
  - Robotics Task plan
  - Banking Account
  - OS Thread
  - Navigation Waypoint
Example: Key Concepts

• Each elevator shall be capable of carrying 10 average-weight people (total cargo weight 3200Kg, floor space > 4 m², height > 2.5m.

• The top five floors are reachable only with a special key.

• In each elevator cabin, there is a sensor for the alignment marks indicating the location of each floor.

• Depending upon the day of the week, doors open at each floor passed, or only at selected floors.

• Depending upon the position of a key-controlled emergency switch, expedited travel can be demanded.

• Identify all the key concepts named or implied

• Should all these be modeled in your system?

Transactions

• Transactions are objects arising from the need to manage or track the interaction of other objects

• Transaction objects may be
  – Persistent: require long term storage
  – Volatile: disappear when the transaction completes
Example: Transactions

A software system must control a set of 8 elevators for a building with 20 floors. In each elevator, there are a set of buttons corresponding to each desired floor and a current floor indicator panel above the door. On each floor there are 2 request buttons, “Up” and “Down.” Additionally, each elevator shaft on each floor has an indicator showing the floor position of that elevator and a direction indicator. The system shall respond to an elevator request by sending the nearest elevator either idle or going in the proper direction.

• Are there any transactions here?
• If so, are they persistent or volatile?

Persistent Information

• Information that must be stored for later retrieval is either objects or attributes
• Example
  – A medical device must maintain an alarm history for later reporting
  – Waveform data must be maintained for 20s buffer to be routed to a chart recorder
  – Access passwords and user IDs
  – Calibration constants
Example: Elevator Persistent Information

A software system must control a set of 8 elevators for a building with 20 floors. Documentation, licensing and routine maintenance is required by local law. The system must maintain a history of maintenance calls, including the date, time, and any defects identified or repaired.

Each elevator contains a telephone in each elevator cabin for the purpose of making emergency calls. Statistics on use versus time of day and day of week are kept.

The motors for controlling elevator elevation and door movement shall be performed smoothly and within acceleration limits defined by the user during the configuration and setup process.

- Is there any persistent information here?

Visual Elements

- Parts of the system viewed by people
- Similar to real-world device or physical device, but
  - could be a software window
- Example
  - tool bar
  - icon
  - cursor
**Example: Elevator Visual Elements**

Each elevator contains a low power LCD display that show current floor (as a number), direction (as a large arrow), next targeted floor (as “Next floor: xx” text), and the manufacturer’s name (as graphics); also prior to the door opening, the text will flash “Door Opening”. When complete, the text will display “Door Open”. When the doors are closing, the text shall display “Door Closing” and when the doors are closed, the text shall read “Door Closed.” An example is shown below:

![Elevator Display](Image)

```
Floor: 15  Next Floor: 8
```

- What are the visual elements here?

---

**Control Elements**

- Entities that control other objects
  - a specific type of causal agent
- Example
  - transaction controller in database
  - audio and video synchronizer
  - autopilot
  - clothes washer cycle controller
Example: Elevator Control Elements

A software system must control a set of 8 elevators for a building with 20 floors. In each elevator, there are a set of buttons corresponding to each desired floor and a current floor indicator panel above the door. On each floor there are 2 request buttons, “Up” and “Down.” Additionally, each elevator shaft on each floor has an indicator showing the floor position of that elevator and a direction indicator. The system shall respond to an elevator request by sending the nearest elevator either idle or going in the proper direction.

• Is there any control activity here?

Players in Scenarios

• Scenarios consist of objects interacting in specific ways over time
• Allow “testing” of class and object models
  – Are all necessary facilities and services available?
  – Do the object’s attributes and behaviors support its responsibilities?
Exercise Problem Specification

For a complete problem description and set of worked exercises with this problem see *Real-Time UML Workshop for Embedded Systems* by Bruce Powel Douglass, Elsevier Press, September 2006

RoadRunner™ Traffic Light Control System

Overview
The Roadrunner Intersection Controller (RIC) is an automobile intersection that controls individual traffic lights, subsurface and above-surface vehicle detectors and pedestrian button signal sources for a single intersection. Each intersection is limited to two intersecting roads, and supports both pedestrian and vehicular traffic in both directions along the road; options include one-way roads and turn lanes. The RIC can be programmed from a Front Panel Display (FPD) or a remote traffic manager via a wired Ethernet Interface, such as the Coyote Traffic Management System, available separately.

The Intersection Controller (IC)
Each intersection is controlled by a separate intersection controller (IC) that may be tuned manually from a secured front panel or through a remote network connection. The intersection controller supports up to 6 vehicular lanes (3 in each direction, including a turn lane), which may be subsurface passive induction loop, or above surface infra red or radar detectors. Pedestrian lights and buttons are supported in each direction. Initial set up of the intersection controller shall configure the number of sensor sources for the intersection.

Figure A-1: Intersection
RoadRunner™ Traffic Light Control System

Each intersection controller shall have a panel control that allows direct local configuration and mode setting.

In addition to all normal operational modes, the RIC shall have a parameter to respond to both priority and emergency vehicle transponders. Priority vehicle transponders are used primarily for mass transit vehicles (e.g. buses) and allow the optimization of their schedules. Emergency vehicle transponders indicate an approaching emergency vehicle (typically fire and police agencies). Such transmitters shall be highly directional so that it is possible to identify which road (primary or secondary) and which lane the vehicle is in. Both priority and emergency modes are only available with the above surface infrared and radar vehicle sensor configurations.

For priority vehicle transmitters, when activated and when the Priority Active parameter is set to TRUE, then when the transponder is within range of the intersection, the intersection shall either extend the through traffic GREEN by 10 seconds or, if the through traffic light is RED, shall shorten the cross-traffic light green by 10 seconds. This is to expedite priority traffic through the intersection.

For emergency transponder, when activated and when the intersection has its Enable Emergency Traffic parameter set to TRUE, then when the transponder is within range of the intersection, the intersection shall immediately cycle the lights to RED for cross traffic and GREEN to same-direction traffic; with all turn lanes set to RED. If the same-direction traffic is already GREEN, then the GREEN time shall be extended. The light shall stay in the emergency state (GREEN for same-direction, RED for cross direction) until 5 seconds after the transponder has passed the intersection or been disabled. Any intersection traffic priority processing shall be aborted in the presence of an active emergency transmitter.

Configuration Parameters

A number of configuration parameters may be set that apply to all or many modes. Parameters specific to a particular mode are described within that mode. All parameters may be set on the front panel or set via the RIC, if present.
## RoadRunner™ Traffic Light Control System

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commanded Mode (CMp)</td>
<td>0..5</td>
<td>Sets the currently active mode (default 0)</td>
</tr>
<tr>
<td>Primary Road (PRp)</td>
<td>0 .. 1</td>
<td>Identifies which road is primary (default, first inputs)</td>
</tr>
<tr>
<td>Primary road directions (PRD)</td>
<td>SINGLE, DUAL</td>
<td>Identifies if the primary road is one-way (SINGLE) or two-way (DUAL) (default DUAL)</td>
</tr>
<tr>
<td>Secondary road directions (SRD)</td>
<td>SINGLE, DUAL</td>
<td>Identifies if the secondary road is one-way (SINGLE) or two-way (DUAL) (default DUAL)</td>
</tr>
<tr>
<td>Vehicle detector type (VDTp)</td>
<td>NONE, SPLI, ASI, ASR</td>
<td>Identifies which kind of vehicle detectors are used. Note: all active lanes within an intersection must use the same type of vehicle detector (default NONE)</td>
</tr>
<tr>
<td>Wireless Frequency (WFp)</td>
<td>0..10</td>
<td>Selectable from 10 wireless frequencies (default 0 – NONE)</td>
</tr>
<tr>
<td>Primary Turn Lanes (PTLp)</td>
<td>FALSE, TRUE</td>
<td>Indicates whether primary road has separate turn lane detectors (default FALSE)</td>
</tr>
<tr>
<td>Secondary Turn Lanes (STLp)</td>
<td>FALSE, TRUE</td>
<td>Indicates whether secondary road has separate turn lane detectors (default FALSE)</td>
</tr>
<tr>
<td>Turn Lane Mode (TMp)</td>
<td>SIM, SEQ</td>
<td>If the turn mode is SIM, then turn lane lights for both directions activate simultaneously; if the mode is SEQ, then the turn lights for both directions are done sequentially, and the turn light GREEN occurs at the same time as the through-light GREEN.</td>
</tr>
<tr>
<td>Primary Pedestrian (PPp)</td>
<td>FALSE, TRUE</td>
<td>Indicates whether primary road has pedestrian buttons and walk indicators (default TRUE)</td>
</tr>
<tr>
<td>Secondary Pedestrian (SPp)</td>
<td>FALSE, TRUE</td>
<td>Indicates whether secondary road has pedestrian buttons and walk indicators (default TRUE)</td>
</tr>
<tr>
<td>Priority Active (PAp)</td>
<td>FALSE, TRUE</td>
<td>When TRUE, RIC receives and responds to the presence of signals from priority transponders (note: only valid with above-surface infrared and radar vehicle detectors) (default FALSE)</td>
</tr>
</tbody>
</table>

## RoadRunner™ Traffic Light Control System

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emergency Active (Eap)</td>
<td>FALSE, TRUE</td>
<td>When TRUE, RIC receives and responds to the presence of signals from emergency priority transponders (note: only valid with above-surface infrared and radar vehicle detectors). (default FALSE)</td>
</tr>
<tr>
<td>Current Time</td>
<td>Hh:mm:ss</td>
<td>Current time of day in 24-hr format (no default)</td>
</tr>
<tr>
<td>Current Date</td>
<td>M:D:Y</td>
<td>Current date month:date:year (no default)</td>
</tr>
<tr>
<td>Morning Start</td>
<td>Hh:mm</td>
<td>Start of morning rush mode (default 06:00)</td>
</tr>
<tr>
<td>Morning Mode</td>
<td>Mode</td>
<td>Mode of the intersection for morning rush (default 0)</td>
</tr>
<tr>
<td>Midday Start</td>
<td>Hh:mm</td>
<td>Start of midday traffic mode (default 10:00)</td>
</tr>
<tr>
<td>Midday Mode</td>
<td>Mode</td>
<td>Mode of the intersection for midday traffic (default 0)</td>
</tr>
<tr>
<td>Evening Start</td>
<td>Hh:mm</td>
<td>Start of evening rush mode (default 16:00)</td>
</tr>
<tr>
<td>Evening Mode</td>
<td>Mode</td>
<td>Mode of the intersection for evening traffic (default 0)</td>
</tr>
<tr>
<td>Night Start</td>
<td>Hh:mm</td>
<td>Start of night mode (default 21:00)</td>
</tr>
<tr>
<td>Night Mode</td>
<td>Mode</td>
<td>Mode of the intersection for night traffic (default 0)</td>
</tr>
</tbody>
</table>
**RoadRunner™ Traffic Light Control System**

The intersection controller shall be able to perform vehicle counting and produce the following performance statistics:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary vehicle count</td>
<td>Number of vehicles that have passed through the intersection since manual reset (primary road)</td>
</tr>
<tr>
<td>Primary traffic count morning</td>
<td>Count of vehicles through the previous morning period, or the current morning period if active</td>
</tr>
<tr>
<td>Primary traffic count midday</td>
<td>Count of vehicles through the previous midday period, or the current morning period if active</td>
</tr>
<tr>
<td>Primary traffic count evening</td>
<td>Count of vehicles through the previous evening period, or the current morning period if active</td>
</tr>
<tr>
<td>Primary traffic count night</td>
<td>Count of vehicles through the previous night period, or the current morning period if active</td>
</tr>
<tr>
<td>Secondary vehicle count</td>
<td>Number of vehicles that have passed through the intersection since manual reset (secondary road)</td>
</tr>
<tr>
<td>Secondary traffic count morning</td>
<td>Count of vehicles through the previous morning period, or the current morning period if active</td>
</tr>
<tr>
<td>Secondary traffic count midday</td>
<td>Count of vehicles through the previous midday period, or the current morning period if active</td>
</tr>
<tr>
<td>Secondary traffic count evening</td>
<td>Count of vehicles through the previous evening period, or the current morning period if active</td>
</tr>
<tr>
<td>Secondary traffic count night</td>
<td>Count of vehicles through the previous night period, or the current morning period if active</td>
</tr>
</tbody>
</table>

Intersection Modes

The RIC supports a number of different operational modes. Modes may be set on the secured front panel for the intersection, or they may be set by the RIC.

**Mode 0: Safe Mode**

This mode is the default when the system has not been configured. All vehicle lights outputs are set to FLASHING RED and pedestrian lights are disabled. This mode persists until another mode is selected. Flash cycle time is 75% "on" duty cycle on at a rate of 0.5 Hz.

**Mode 1: Evening Low Volume Mode**

In this mode, the designated primary road is set to FLASHING YELLOW, secondary road is set to FLASHING RED, and pedestrian lights are set to off. Same cycle times as in Mode 0 shall be used.

**Mode 2: Fixed Cycle Time**

Mode 2 is the most common operational mode. In this mode, the lanes cycle GREEN-YELLOW-RED in opposite sequences with fixed intervals. The system shall ensure that if any traffic light is non-RED, then all the lights for cross traffic shall be RED and pedestrian lights (if any) shall be set to DON’T WALK. Note that the turn lane times and/or pedestrian times are only valid in this mode if (1) the turn lane and/or pedestrian parameter is set TRUE in the RIC system parameters and (2) if a signal from the appropriate detector determines the existence of waiting traffic for the turn or pedestrian light.

The durations of the light times shall be independently adjustable by setting the appropriate parameters (see below). Note that in the table, the values in parentheses are defaults.
Table A-1: Mode 2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset Parameters</td>
<td>FALSE, TRUE</td>
<td>(FALSE) Sets all the parameters for Mode 2 to defaults</td>
</tr>
<tr>
<td>Primary Green Time (PG2)</td>
<td>10 to 180</td>
<td>(30) Length of time the primary green light is on</td>
</tr>
<tr>
<td>Primary Yellow Time (PY2)</td>
<td>2 to 10</td>
<td>(5) Length of time the primary yellow light is on</td>
</tr>
<tr>
<td>Primary Red Delay Time (PR2)</td>
<td>0 to 5</td>
<td>(0) Length of time between when primary red light is turned on and the secondary green light is activated</td>
</tr>
<tr>
<td>Primary Walk Time (PW2)</td>
<td>0 to 60</td>
<td>(20) Length of time the primary WALK light is on when the primary GREEN light is activated</td>
</tr>
<tr>
<td>Primary Warn Time (PA2)</td>
<td>0 to 30</td>
<td>(10) Length of time the primary FLASHING DON’T WALK light is on after the WALK light has been on</td>
</tr>
<tr>
<td>Primary Turn Green Time (PT2)</td>
<td>0 to 90</td>
<td>(20) Length of time the primary turn light is GREEN. Note: only valid when the Primary Turn Light parameter is TRUE.</td>
</tr>
<tr>
<td>Primary Turn Yellow Time (PZ2)</td>
<td>0 to 10</td>
<td>(5) Length of time the primary turn light is YELLOW. Note: only valid when the Primary Turn Light parameter is TRUE.</td>
</tr>
</tbody>
</table>

Table A-2: Default Cycle Times for Mode 2

<table>
<thead>
<tr>
<th>Turn Lane</th>
<th>Ped Signal</th>
<th>Green</th>
<th>Yellow</th>
<th>Red</th>
<th>Walk</th>
<th>Don’t Walk</th>
<th>Turn Green</th>
<th>Turn Yellow</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>30</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>30</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>50</td>
<td>5</td>
<td>0</td>
<td>15</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>50</td>
<td>5</td>
<td>0</td>
<td>15</td>
<td>5</td>
<td>15</td>
<td>5</td>
</tr>
</tbody>
</table>

RoadRunner™ Traffic Light Control System

The values in Table A-2 are true for each direction, independently. Thus, if the primary road has a car waiting in its turn lane and a pedestrian walking, but the secondary road has neither, then the following timing diagram represents the cycle times for simultaneous turn lane mode (i.e. the turn lanes in both directions for a road turn together and the straight traffic doesn’t begin until the turn lanes have cycled to Red).
Mode 3: Responsive Cycle Mode
Mode 3 provides a fixed cycle time when secondary road is triggered by either pedestrian or vehicle. That is, it operates exactly like Mode 2 except that in the absence of cross-traffic signals (vehicle or pedestrian), primary road or primary pedestrian signals, the system shall cycle to the default condition: primary through lights are GREEN, primary turn lights are RED, secondary through and turn lights are RED, and all pedestrian lights DON'T WALK. As long as the signal is refreshed within the GREEN or WALK times, the currently active vehicle or walk light shall be refreshed. If the appropriate interval elapses without a fresh vehicle or pedestrian signal, then the intersection shall cycle to the default state. The default shall be maintained for at least the minimum duration of the Primary Green Time, as specified by the Mode 3 parameters. If there are both turning vehicles and pedestrians waiting along the same road, they shall be handled in same order as in Mode 2 (turn lane first, then pedestrians). If a road’s turn lane is GREEN and the pedestrian signal occurs along the same road, then the turn lane shall cycle to RED, and the pedestrian lights shall cycle to DON'T WALK. If there is waiting traffic in both roads of the intersection, then the intersection shall cycle as if it were in Mode 2 until no signals are received for the cycle times associated with the signals.

The same parameter set is used as in Mode 2, except the system responds to them with Mode 3 behavior, when Mode 3 is the selected mode.

Mode 4: Adaptive Mode
Mode 4 is for intersections with higher density traffic. In this mode, the intersection adapts to the local history of traffic by adjusting the cycle times depending on traffic density. This requires vehicle-counting behavior from the vehicle detectors. This is similar to Mode 2 and has the same parameters, plus:

Table A-3: Mode 4 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Averaging Interval (AIP)</td>
<td>10 .. 120 minutes</td>
<td>Period over which traffic volume is averaged to compute the relative density between the two roads</td>
</tr>
<tr>
<td>Minimum Density (MDP)</td>
<td>10 .. 1000</td>
<td>(100) Specifies the minimum number of vehicles, from both roads, that must have traversed the intersection before adaptive extension will be employed</td>
</tr>
<tr>
<td>90%</td>
<td>0 .. 60 seconds</td>
<td>(30) Length of time that a road’s green time may be extended due to higher traffic volume when the road’s traffic is 90% of total intersection traffic volume</td>
</tr>
<tr>
<td>80%</td>
<td>0 .. 60 seconds</td>
<td>(20) Length of time that a road’s green time may be extended due to higher traffic volume when the road’s traffic is 90% of total intersection traffic volume</td>
</tr>
<tr>
<td>70%</td>
<td>0 .. 60 seconds</td>
<td>(10) Length of time that a road’s green time may be extended due to higher traffic volume when the road’s traffic is 90% of total intersection traffic volume</td>
</tr>
<tr>
<td>60%</td>
<td>0 .. 60 seconds</td>
<td>(5) Length of time that a road’s green time may be extended due to higher traffic volume when the road’s traffic is 90% of total intersection traffic volume</td>
</tr>
</tbody>
</table>

The Vehicle Detector
Three types of Vehicular Detectors shall be supported: subsurface passive loop inductors (SPLIs), above-surface infrared sensors (ASIs) and above-surface radars (ASRs).
Subsurface detectors shall use a wired interface to communicate with the controller, while ASIs and ASRs shall support both wired and secure wireless communication. All vehicle detectors shall be able to perform vehicle counting.

In addition, ASIs and ASRs shall be able to receive directional transmissions from priority vehicle and emergency vehicle transmitters. The maximum range of such reception shall be no less than 250 feet and no more than 1000 feet.

Figure A-3 shows the relevant measures for both ASI and ASR detectors. When a vehicle enters the detection area (shown as the shaded area in the figure), the detector shall report the presence of a vehicle. Separate detectors are used for each lane in each direction.
**Vehicular Traffic Light**

Two kinds of traffic lights are supported: the standard three-light model and the 4-light model, the additional light being for a green turn arrow. When the intersection is allowing turn lane turns with the 4-light model, then the green arrow shall be on.

**Pedestrian Light and Sensor**

The pedestrian light is a two-signal light that can either be in the state of WALK, FLASHING DON'T WALK, or DON'T WALK, as shown in Figure A-5. If the light no longer detects that it is communication with the RIC, it shall go to a state of DON'T WALK within 10 seconds of the communication or RIC failure.
RoadRunner™ Traffic Light Control System

Front Panel Display
The front panel display (FPD) is an enclosed front panel display secured with a mechanical key access lock. The FPD provides an LCD for viewing information and keypad for entering information. The front panel display also has a wired Ethernet interface so that a local or remote network can view or set the values known to the system.

The FPD shall support the following choices as menus.

- Intersection Configuration Setup
- Mode 2 (Fixed Cycle Time) Setup
- Mode 3 (Response Cycle) Setup
- Mode 4 (Adaptive Cycle) Setup
- Intersection Statistics Display
- Device Manufacture Display

The FPD shall have a set of front panel keys and knobs as shown in Figure A-6. The turn knobs are used for menu selection and item selection. The numbers are for entering values; the arrow keys are for moving from field to field within segments of a parameter (such as mm:dd:yy for month:day:year date format). The Item Selection knob allows the selection of a parameter to change (if changeable). Before a parameter can be changed, it must be selected and then the user must press the EDIT key. Now the values can be entered with the keypad (if numeric) or selected from a list (if enumerated). When a change is made, it must be either confirmed by pressed the CONFIRM key or the change aborted with the ABORT key. The RESET key returns the value to the factory default if a current item is in the edit mode; or resets all parameters on the page to their defaults if no item is currently being edited.

Intersection Configuration Setup
Mode 2 (Fixed Cycle Time) Setup
Mode 3 (Response Cycle) Setup
Mode 4 (Adaptive Cycle) Setup
Intersection Statistics Display
Device Manufacture Display

The FPD supports 4 Ethernet ports for external interfacing and digital I/O ports for interfacing with the lights and sensors. Software may be uploaded via the Ethernet power from a service technician’s laptop.

The FPD supports secure wireless communication with up to 50 devices. Security includes 64-bit encryption of data and MAC address filtering. The wireless devices are primarily used to interact with infrared and radar vehicle sensors, but may also be used for connecting with a service technician’s PC. The set up information for wireless, other than the frequency, is not available on the FPD and must be uploaded from the service technician’s laptop. The range of the wireless connection shall be no less than 200 feet nor more than 1000 feet with an unobstructed line of sight.
Remote Communications
The RIC provides wired 10/100 Ethernet interfaces for remote monitoring and management (see the FPD). All parameters available on the front panel may be requested or set via this interface. In addition, traffic statistics may be viewed or reset via this interface.

Power
All components of the RIC that require power will accept 220-240 volts. Uninterruptible power supply (UPS) option is available to provide 1 hour or 10 hour functionality in the absence of line power. When the power is on, the UPS battery shall recharge. When running on UPS, the power light on the front panel display shall be RED. When running on main power, but charging the battery (and battery is at less than 90% capacity), the power light shall be AMBER. When on main power and battery is at 90% or more capacity, or when on main power and the UPS option is not present, the power light shall be GREEN.
Starting Point: Use Cases

Starting Point: Requirements Diagram

Requirements contained within the Detect Vehicle use case

Three types of Vehicular Detectors shall be supported:
- sub-surface passive-loop inductors (SPLIs),
- above-surface infrared sensors (ASIs),
- and above-surface radars (ASRs).

Separate detectors are used for each lane in each direction.

Subsurface detectors shall use a wired interface to communicate with the controller.

In addition, ASIs and ASTs shall be able to receive directional transmissions from priority vehicle and emergency vehicle transmitters.

ASIs and ASRs shall support both wired and secure wireless communication.

The maximum range of such reception shall be no less than 240 feet and no more than 1000 feet.

All vehicle detectors shall be able to perform vehicle counting.

Figure 2: Infrared and Radar Vehicle Detector

Figure 2 shows the relevant measures for both ASI and ASR detectors. When a vehicle enters the detection area (shown as the shaded area in the figure), the detector shall report the presence of a vehicle.
Starting Point: Use Case Scenario

User Case: Responsive Cycle Mode
Scenario: Pedestrian Cross Traffic

Preconditions:
- Mode: Responsive
- Green Time: 20s
- Yellow Time: 5s
- Red Delay Time: 0s
- Walk Time: 20s
- Green Turn Time: 20s
- Green Yellow Time: 5s

Starting conditions:
- Primary is Green
- Primary walklight is Don't Walk

Starting Point: System Architecture

Road Runner System Architecture:
Mission: Show subsystem architecture with operations
Exercise

- **Apply 3 strategies** to the creation of an intersection Traffic Control System (*use case: Detect Vehicle*), that can detect vehicles using 3 modalities: subsurface passive loop inductors, above-surface radar, and above-surface infrared. The system must also be able detect and handle both emergency vehicles and priorities vehicles (e.g. buses) using separate wireless communications.

- **Guidelines:**
  - Create a “semantically deep” model with an eye towards execution
  - Should have attributes, operations, relations, maybe states and transitions (depending on the strategy). Don’t do the *napkin drawing (i.e. semantic-free)* approach!
  - In actual application, you don’t draw all objects from a strategy and then begin to execute. Instead you begin to execute as you add classes and objects from the strategy, executing each incremental step. Use a single package to hold your classes but one diagram per strategy then an additional diagram with the “merged” results

- **Answer the following:**
  - What objects/classes/attributes were identified from more than one strategy?
  - What objects/classes/attributes were only found by one strategy?
  - Is the collaboration complete (i.e. does it meet all the requirements)?

---

**Design in Harmony**

- **Prespiral Planning**
  - Iterative Prototypes

- **Testing**
  - Integration Testing
  - Validation Testing
  - Increment Review (Party)

- **Design**
  - Architectural Design
  - Mechanical Design
  - Detailed Design

- **Analysis**
  - Object Analysis
  - Prototype Definition

- **Implementation**
  - Translation
  - Unit Testing
  - Peer Review

- **Control Project**
Design Activities

• Architectural design
  – Define strategic design decisions
    • Logical model
    • Physical model
      – Subsystem model
      – Concurrency model
      – Distribution model
      – Safety and reliability model
      – Deployment model

• Mechanistic design
  – Optimizes individual collaborations by applying design patterns and adding “glue” objects

• Detailed design
  – Defines and optimizes object internals

Design Workflow

Criteria and patterns are identified for all 5 aspects of architecture
Optimizes collaborations by applying collaboration-scope design patterns
Optimizes individual objects in terms of their structure or behavior
Physical Architectural Views

- Construct architectural design models
  - Subsystem Model
  - Concurrency Model
  - Distribution Model
  - Safety and Reliability Model
  - Deployment Model
- Capture with
  - Class Diagrams
  - Package Diagrams
  - Subsystem Diagrams
  - Task Diagrams
  - Deployment Diagrams
Coyote UAV System Architecture

Coyote UAV Aircraft Architecture
Subsystem and Component View

- A component
  - is the basic reusable element of software
  - organizes objects together into cohesive run-time units that are replaced together.
  - provides language-independent opaque interfaces
  - A kind of structured class (class with parts – UML 2.0)

- A subsystem
  - is a large object that provides opaque interfaces to its clients and achieves its functionality through delegation to objects that it owns internally
  - contains components and objects on the basis of common run-time functional purpose
  - a kind of component (UML 2.0)

Distribution Architecture

- Distribution model refers to
  - Policies for distribution objects among multiple processors and communication links, e.g.
    - Asymmetric distribution (dedicated links to objects with a priori known location)
    - Publish-Subscribe
    - CORBA and Broker symmetric distribution
  - Policies for managing communication links
    - Communication protocols
    - Communication quality of service management
Distribution Architecture with Ports (Example)

- We want to add to our system, a distribution architecture that
  - Allows efficient exchange of data and service requests over a network or distribution middleware
  - Does not in any way affect the internal structure or code for each subsystem
  - Does not break the encapsulation boundaries
    - I.e. requires the use of explicitly defined interfaces that type explicitly specified connection points (ports)

- Answer: Port Proxy Pattern
Port Proxy Pattern Context

Problem: How do I connect these two elements over a network that doesn't know anything about the application semantics and so cannot provide or require the necessary interfaces?

Port Proxy Pattern Structure

Port Proxy Pattern Solution Structure
Interface Proxy Pattern

Here is an example of how one can have two different implementations of I_TrackProcessing

- Std – Implements run-of-the-mill code
- PubSub – Implements specialized pub / sub code

Application Of the Interface Proxy Pattern

By implementing a port interface proxy that adheres to the interface specification, we can do pub/sub across a link.

Once received via the port implementation, the inside parts don’t know or care that the data was received via pub/sub vs a standard function call!
Safety and Reliability Model

- Safety and reliability model refers to the structures and policies in place to ensure
  - Safety
    - Freedom from accidents or losses
  - Reliability
    - High MTBF
    - Fault tolerance
- Safety and fault tolerance always require some level of redundancy

Safety and reliability of object models is described more completely in *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns*
Safety and Reliability Architecture

Subsystem Deployment Model

Note: this is in the subsystem model Deployment diagram - using a class diagram to Model deployment, as it is done in SysML. Stereotypes identify the "kind" of element it is - Physical (undifferentiated), mechanical, electronic, Or software (default).
Concurrency Architecture

- Refers to
  - Identification of task threads and their properties
  - Mapping of passive classes to task threads
  - Identification of synchronization policies
  - Task scheduling policies

- Unit of concurrency in UML is the «active» object
  - «active» objects are added in architectural design to organize passive objects into threads
  - «active» objects contain passive semantic objects via composition and delegate asynchronous messages to them

Basic Definitions

- Concurrency
  Concurrency refers to the simultaneous execution of action sequences

- Concurrency unit
  A Concurrency Unit (task or thread) has a sequence of actions in which the order of execution is known however the order of execution of actions in different concurrency units is “don’t know – don’t care” (except at explicit synchronization points)
Concurrency

• What’s the order of execution?
  – A then X then Alpha?
  – Alpha then Beta then Gamma then X then Y then A?
  – A then B then X then Y then D then Alpha?
  • ALL ARE CORRECT

If you care about the order between the sequences, then concurrency was the wrong choice!

Basic Definitions

• Urgency

**Urgency** refers to the nearness of a deadline

• Criticality

**Criticality** refers to the importance of the task’s correct and timely completion
Basic Definitions

• Deadline

A **deadline** is a point in time at which the completion of an action becomes incorrect or irrelevant.

• Priority

**Priority** is a numeric value used to determine which task, of the current ready-to-run task set will execute preferentially.

Basic Definitions

• Arrival Pattern

The **arrival pattern** for a task or triggering event is either time-based (periodic) or event-based (aperiodic).

• Synchronization Pattern

**Synchronization pattern** refers to the how the tasks execute during a rendezvous, e.g. synchronous, balking, waiting, or timed.
Basic Definitions

- Blocking Time

The **blocking time** for a task or action is the length of time it may be kept from executing because a lower priority task owns a required resource.

- Execution Time

The **execution time** for a task or action is the length of time it requires to complete execution.
**Blocking**

- What is the blocking time for Task Z?
- What is the blocking time for Task Y?
- What is the blocking time for Task X?
- What is the blocking time for Task A?
- Will Task A always meet its deadlines?

This illustrates *unbounded priority inversion* – this is **ALWAYS** a bad thing!

**Priority Inheritance**

- The *Priority Ceiling* for a resource is the priority of the highest priority task that can ever access the resource (in this case “1”)
  - While a lower priority task accesses the resource, it’s priority is temporarily escalated to its resource ceiling and deescalated once it releases the resource
- What is the blocking time for Task Z?
- What is the blocking time for Task Y?
- What is the blocking time for Task X?
- What is the blocking time for Task A?
- Will Task A always meet its deadlines?
Basic Definitions

- **Timeliness**
  
  *Timeliness* refers to the ability of a task to predictably complete its execution prior to the elapse of its deadline.

- **Schedulability**
  
  A task set is schedulable if it can be guaranteed that in all cases, all deadlines will be met.

Task Diagram

- A task diagram is a class diagram that shows only model elements related to the concurrency model:
  - Active objects
  - Semaphore objects
  - Message and data queues
  - Constraints and tagged values

- May use opaque or transparent interfaces
Concurrency Model

- Active object is a stereotype of an object which owns the root of a thread
- Active objects normally aggregate passive objects via composition relations
- Standard icon is a class box with heavy line

```
1 aThread: DataAcqThread
    | Heavy side border indicates «active» object
    1 dSensor
    1 dDatabase
    1 dqDataQueue

1 dThread: DisplayThread
    1 nvNumericView
    1 wdvWaveformDisplayView
```

Task Diagram

Example Task Diagram with ISR, Resource, and Mutex
Task Performance Budgets

- The context defines the end-to-end performance requirements

- This determines the overall task budget
  - Computation of total budget may *not* just be simply adding up the times due to concurrency

- Individual operations and actions within tasks must be assigned portions of the overall budget
  - Action budgets should be checked during unit test
  - Action budgets should take into account potential blocking

Required / Offered QoS
### Required / Offered QoS

#### Task Identification

**Task Identification Strategy** | **Description**
---|---
Single event groups | for simple systems, you may define a thread for each event type
Event source | group all events from a single source together for a thread
Related information | For example, all numeric heart data
Interface device | For example, a bus interface
Event properties | Events with the same period, or aperiodic events
Target object | For example, waveform queue or trend database
Safety Level | For example, BIT, redundant thread processing, watchdog tasks

---

*best for schedulability*
Key Concepts

- What is a design pattern?
- Advantages of design patterns
- Characteristics of design patterns
Design Patterns

- Design patterns are
  1. Generalized solutions to commonly occurring design problems
  2. Parameterized collaborations of objects, where the object roles are the *formal parameters* and the objects that play those roles are the *actual parameters* when the pattern is instantiated

- UML has a notation to explicitly depict design patterns.
  - Shown as a dotted ellipse with dotted lines to the collaborating objects or classes

Example Analysis Model Collaboration
Why Use Design Patterns?

- Reuse effective design solutions
- Provide a more powerful vocabulary of design concepts to developers
- Develop “optimal” designs for specific design criteria
- Develop more understandable designs

How do I Apply Design Patterns???

1. Construct the initial model
2. Identify the design criteria
3. Rank the design criteria in order of importance
4. Identify design patterns that optimize the system (architectural) or collaboration (mechanistic) for the critical design criteria at the expense of the lesser important ones
   - Architectural patterns apply *system-wide*
   - Mechanistic patterns apply *collaboration-wide*
   - State behavioral patterns apply *object state machine-wide*
5. Incorporate selected design patterns into your model
6. Validate your design model
   - Ensure you haven’t broken the functionality
   - Ensure you’ve achieved your optimization goals
Example of Pattern Selection

Architectural Patterns:
- Static priority Pattern
- Fixed Block Memory Allocation Pattern
- Channel Pattern
- Triple Modular Redundancy Pattern

Example of Pattern Selection (2)

Architectural Patterns:
- Recursive Containment Pattern
- Cyclic Executive Pattern
- Dynamic Memory Pattern
- Container-Iterator Pattern
4 Aspects of Patterns

• Problem
  – What problem does the pattern address

• Applicability
  – What design criteria the pattern is trying to optimize
  – When is the pattern applicable

• Solution
  – Structure of the collaborative elements in the pattern
  – How the structure supports the dynamic behavior required

• Consequences
  – Pros
  – Cons

Pattern Solutions Have Aspects

• Structural model of the pattern, containing
  – Provided pattern elements
  – Formal parameters
    • These are classes from your model
  – Relations among elements

• Behavioral model of the pattern, containing
  – Sequence diagrams illustrating the interaction of the elements within the pattern
  – State machines or activity diagrams specifying the behavior of some of the individual pattern elements
  – Activity diagram specifying the overall interaction of the pattern elements
Architectural Design Workflow

Key views include:
- Subsystem and component view
- Concurrency and resource view
- Distribution view
- Safety and Reliability view
- Deployment view

Not all views may be relevant within the mission of a given prototype, but all will be addressed before the project is completed.

Architecture: Optimize Architectural Model

This task focuses on the optimization of the entire system based upon a specific architectural view.

The purpose of this task is to optimize the model by adding in strategic (i.e., architectural) design decisions.

The input to the task is the un-optimized PIM (plus architectural decisions from previous prototypes). This task adds additional relevant architectural optimization decisions. Architecture is divided up into different subject matter views (see Guidance, below) each of which has its own specific design patterns. This step optimizes the system by selecting and applying patterns in one or more views.

Steps:
- Identify design criteria
- Rank design criteria
- Select design approach
- Apply design patterns
- Refine scenarios
**Artifact: Physical Architecture**

- System is organized into run-time artifacts *subsystems*, *components*, and *tasks*
- Subsystems contain instances of classes specified in the domains
- Subsystems specify only classes used to organize and deploy domain classes
- Consists of
  - Software run-time organizational units
  - Deployment hardware

---

**Defining the Concurrency Model**

- Use "Task Identification Strategies"
- Define Concurrency and Resource Architecture
  - Preconditions: There is an analysis model with at least one collaboration of classes or object.
  - Postconditions: An analysis model refactored into a design model with concurrency
- Task properties include:
  - Periodic or aperiodic?
  - Worst-case execution time
  - Period
  - Minimum interarrival time
  - Deadline
  - Blocking time

- For each task:
  - Specify task properties
  - Add "passive" objects into task
  - Refactor relations across tasks as necessary, possibly adding ports
  - Identify scenarios

- Identify tasks
- Make an &lt;active&gt; object for each Task
- Select resource-sharing patterns
- Apply resource-sharing patterns
- Apply synchronization patterns
- Select synchronization patterns
- Identify synchronization points
- Apply scheduling patterns
- Determine scheduling patterns
- Test via execution
- Apply Schedulability analysis
Exercise: Concurrency Architecture

- Add concurrency architecture to the object analysis model by
  - Identifying task threads
  - Constructing «active» classes
  - Moving the passive objects into the «active» objects via composition

- Guidelines
  - See the Concurrency Architecture Workflow presented previously
  - *Use the task identification patterns* presented previously – that’s why they’re there!
  - Identify shared resources and make them “passive” (execute in the context of the caller)
  - Draw the active classes with PIM classes as parts
    - *Hint: you’ll have to make the parts objects that execute in the context of the active class*

Mechanistic Design

- Identifies patterns and optimizations for object collaborations
- Concern is to optimize the individual collaboration against it’s required QoS
- Add design-level classes and objects
  - Container classes
  - Iterators
  - Adaptors
  - Interfaces
  - Helpers
  - Mechanistic design patterns
  - Local error handling policies
Mechanistic Design-level Patterns

- Domain-level classes already exist from analysis
- Design-level classes
  - Add the glue to stick them together
  - Resolve details of object interaction
  - Optimize collaboration against QoS
- Design-level classes most often come via the application of mechanistic design patterns
  - See Gamma, Helm, Johnson, Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

Mechanistic Design Workflow

<table>
<thead>
<tr>
<th>Work Breakdown</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Breakdown Element</strong></td>
</tr>
<tr>
<td>Select Object Analysis Collaboration</td>
</tr>
<tr>
<td>Optimize Mechanistic Model</td>
</tr>
<tr>
<td>Validate Collaborative Model</td>
</tr>
</tbody>
</table>
Example: Adapter Pattern

- **Problem**
  - You want to adapt the interface of a class to meet a client need
- **Applicability**
  - When you have an existing class that meets the *semantic* need but has an incompatible interface
  - When you want to reuse an existing class in as-yet-determined circumstances
- **Solution**
  - Create a class subclasses both the expected and actual interfaces and does the impedance matching, or
  - Create an object that refactors and forwards the requests
- **Consequences**
  - Class adapters work with only the specific class not subclasses
  - Object adapters work with class and subclasses
  - Class adapters introduce only a single object
  - Object adapters introduce a new object for every server object
Adapter Pattern Example (Class)

Adapter Pattern Example (Object)
Exercise: Mechanistic Design

• Apply Adapter Pattern to the object analysis collaboration to optimize the ability of the system to use different kinds of sensors from different manufacturers, even if they don’t provide the “right” interface

• Guidelines
  – Only show the elements on the class diagram that are affected by the application of the pattern
  – Think about adapting interfaces for Passive Loop Inductors from Acme™ and from Penultimate™ vendors. They have different interfaces, yet we have to connect them to our system so that we can use those devices to detect arriving vehicles.

Detailed Design

• Refine the details of classes themselves
  – Define visibility for attributes and behaviors
  – Define operation protocols
  – Define data structures
  – Define algorithms
  – Refine object references
  – Define default values and constructor/destructor
  – Define error handling
  – Responsibility for instance creation/destruction
Detailed Design

- Decide on time/space/safety tradeoffs
- Dynamic polymorphic operations require 1 additional pointer dereference
- Static polymorphic operations have no run-time overhead
- Inlines may be faster, but may increase memory size
- Run-time checks slow code but make it more fault tolerant and safe
Define Operation Protocols

- Preconditional invariants
  - Things client claims are true before operation
  - Determine responsibility to ensure adherence
- Operation signature
  - Number and type of parameters
  - Parameter order
  - Default value if optional
  - Polymorphic signatures
- Operator Sequence
  - Relative to other operations (usually via statechart)
- Postconditional invariants
  - Things server claims are true after operation
  - Determine responsibility to ensure adherence
- Quality of service constraints

Define Data Structures

- What do you want to optimize?
  - Reuse
  - Read access time
  - Write access time
  - Space complexity
  - Development effort
- Add private operations to support internal data structure manipulations
- Example: Tree
  - Unbalanced?
  - AVL Tree?
  - Red-Black?
Operation QoS Adherence

- A complete set of operations makes more likely for reuse
- For the Liskov Substitutability Principle to hold, QoS must be satisfied by subclasses
  - Subclass constraints should remain conformant with superclass constraint

![Diagram showing QoS adherence and subclass constraints]

Implementation in Harmony

- Analysis: Object Analysis, Prototype Definition
- Design: Architectural Design, Mechanical Design, Detailed Design
- Implementation: Translation, Unit Testing, Peer Review
- Prespiral Planning, Iterative Prototypes
Implementation Workflow

Testing in Harmony

Analysis
- Object Analysis
- Prototype Definition

Design
- Architectural Design
- Mechanical Design
- Detailed Design

Prespiral Planning
- Iterative Prototypes

Testing
- Integration Testing
- Validation Testing
- Increment Review (Party)

Implementation
- Translation
- Unit Testing
- Peer Review
Integration Workflow

Validation Workflow
Increment Review (Party) Workflow

Adopting MDA/MDD
What you need to Effectively Adopt MDA

• Tools that provide
  – Ability to validate models (requirements, analysis, design…)
  – Ability to generate code (forward engineering)
  – Facility to provide transparent bi-directional model-code associativity (forward and backward engineering)

• Technical Knowledge
  – Training in effective use of the UML for
    • Requirements Capture
    • Model organization
    • Architecture
    • Real-time and embedded features
    • Design
    • Test

What you need to Effectively Adopt MDA

• Processes that support model-based development
  – Managing model artifacts
  – Capturing IP in models instead of text and code
  – Incremental development and construction of your systems
  – Validating models all the way through the project not just at the end
  – Identification and management of risks

• Planning
  – Software and systems development plans cognizant of model approaches
  – Estimation, scheduling, and tracking of iterative, incremental development projects
  – Risk management and risk avoidance

• Assessments
  – In-project assessments with replanning & redirection as necessary based on actual project data (actual vs planned)
  – Identification of looming risks and risk reduction activities
Adopting MDA

• You can do it yourself
  – “Learning by making mistakes"
  – Experience shows benefits can be achieved by the 3rd project

• You can do it with help
  – “Learning by leveraging expertise"
  – Experience has shown that properly applied, you can realized benefits on the very first project

What Kind are Aids are Available?

• Tools
  – Modeling
  – Requirements management
  – Testing

• Initial Assessment
  – Knowledge
  – Process
  – Risk

• Knowledge transfer
  – Targeted training
  – UML/technology training
  – Advanced training (design patterns, advanced behavioral modeling, scheduability analysis, requirements analysis, estimation & scheduling, …)

• Process Improvement
  – Process deployment plan

• On-going Project mentoring

• Periodic project assessments
References

Also look for whitepapers at www.telelogic.com\modeling

Appendix: Answers to Modeling Exercises
Answers: Object Analysis (underline the nouns)

Table 10.3 Default cycle times for Mode 2

<table>
<thead>
<tr>
<th>Turn Lane</th>
<th>Ped Signal</th>
<th>Green</th>
<th>Yellow</th>
<th>Red</th>
<th>Walk</th>
<th>Don’t Walk</th>
<th>Turn Green</th>
<th>Turn Yellow</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>30</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>50</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>50</td>
<td>5</td>
<td>0</td>
<td>15</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>50</td>
<td>5</td>
<td>0</td>
<td>15</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The values in Table 10.3 are true for each direction, independently. Thus, if the primary road has a car waiting in its turn lane and a pedestrian walking, but the secondary road has neither, then the following timing diagram represents the cycle times for simultaneous turn lane mode (i.e., the turn lanes in both directions for a road turn together and the straight traffic doesn't begin until the turn lanes have cycled to Red).

Answers: Object Analysis (underline the nouns)

Table 10.4 Roadrunner fixed cycle mode nouns

<table>
<thead>
<tr>
<th>Noun Phrase</th>
<th>Element Type</th>
<th>Element Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode 2</td>
<td>state</td>
<td>System object</td>
</tr>
<tr>
<td>operational mode</td>
<td>state</td>
<td>System object</td>
</tr>
<tr>
<td>mode</td>
<td>state</td>
<td>System object</td>
</tr>
<tr>
<td>lanes</td>
<td>class</td>
<td>Vehicle Light Assembly</td>
</tr>
<tr>
<td>sequences</td>
<td>set of states</td>
<td>in various statecharts</td>
</tr>
<tr>
<td>fixed intervals</td>
<td>timeout</td>
<td>in various statecharts</td>
</tr>
<tr>
<td>system</td>
<td>object</td>
<td>System object</td>
</tr>
<tr>
<td>traffic light</td>
<td>class</td>
<td>Traffic Light Assembly</td>
</tr>
<tr>
<td>lights</td>
<td>class</td>
<td>Traffic Light Assembly</td>
</tr>
<tr>
<td>cross traffic</td>
<td>actor</td>
<td>Vehicle</td>
</tr>
<tr>
<td>pedestrian lights</td>
<td>class</td>
<td>Pedestrian Light Assembly</td>
</tr>
<tr>
<td>turn lane times</td>
<td>timeout</td>
<td>in statechart of Traffic Light Assembly</td>
</tr>
<tr>
<td>pedestrian timers</td>
<td>timeout</td>
<td>in statechart of Pedestrian Light</td>
</tr>
</tbody>
</table>
Answers: Object Analysis (real-world items)

Answers: Object Analysis (merged)
Answers: Concurrency Architecture

Answers: Mechanistic Design