



Incident Priority

Version 1.4
February 18, 2004

PathMATE Technical Notes

Pathfinder Solutions LLC
33 Commercial Drive, Suite 2
Foxboro, MA 02035 USA
www.PathfinderMDA.com
888-662-7284

Table Of Contents

- 1. Introduction..... 1**
- 2. Design Approach..... 1**
 - Specification of Class and Incident Priority.....1
 - PfdIncident Priority and Scheduling1
 - Local ServiceHandle Limitations1
- 3. Modeling Considerations 3**
 - Testing Considerations.....3
 - Delayed Events*3
 - Self-directed Events*4
 - Starvation*4
 - Prioritized Service Handles*4
 - Untriggered Transitions*.....4
- 4. Specifying a Complete Response Path..... 4**

1. Introduction

This Technical Note describes how the specification of priority for Incidents (Events and ServiceHandles) will be supported, and how this capability can be applied to achieve better control over the scheduling behavior of the PfdTask (SW_Task in C). This capability would be used to reduce the number of actual RTOS tasks used in a specific Structural Design, and still achieve the most rapid responses possible for certain specified Incidents. Essentially we are moving the management of some tasking priorities from the RTOS task scheduler to the PfdTask, where the problem is simplified and we have more control.

2. Design Approach

There are 2 key aspects to implementing Incident Priorities: priority specification and priority capture/scheduling.

Specification of Class and Incident Priority

Through the use of a properties.txt file, the Incident property "Priority" will be specified by the designer, and assigned to Events and Services (Service priorities are only considered when a ServiceHandle is created, or if a domain service is invoked in an inter-task context). Any literal integer constant value or symbolic constant (positive or negative) is supported. Higher priority numbers result in more rapid scheduling than lower priority numbers. The default value is the Priority of the Incident's defining class.

The Class property "Priority" is provided for convenient control over related groups of Events and Services. The default value is 0. The class' priority value specifies the default priority value for all Events and Services defining for that class. Please note, in the case of polymorphic events, it is the defining class (the supertype) and not the receiving class (the subtype) that specifies an Event's default priority.

PfdIncident Priority and Scheduling

A new data member of the PfdIncident class (SW_Incident in C) will be populated with the specified Incident priority upon incident creation. When an Incident is queued with a specific PfdTask, the priority is used to determine where to insert the Incident in the incident queue – the Incidents will be arranged in descending order of priority.

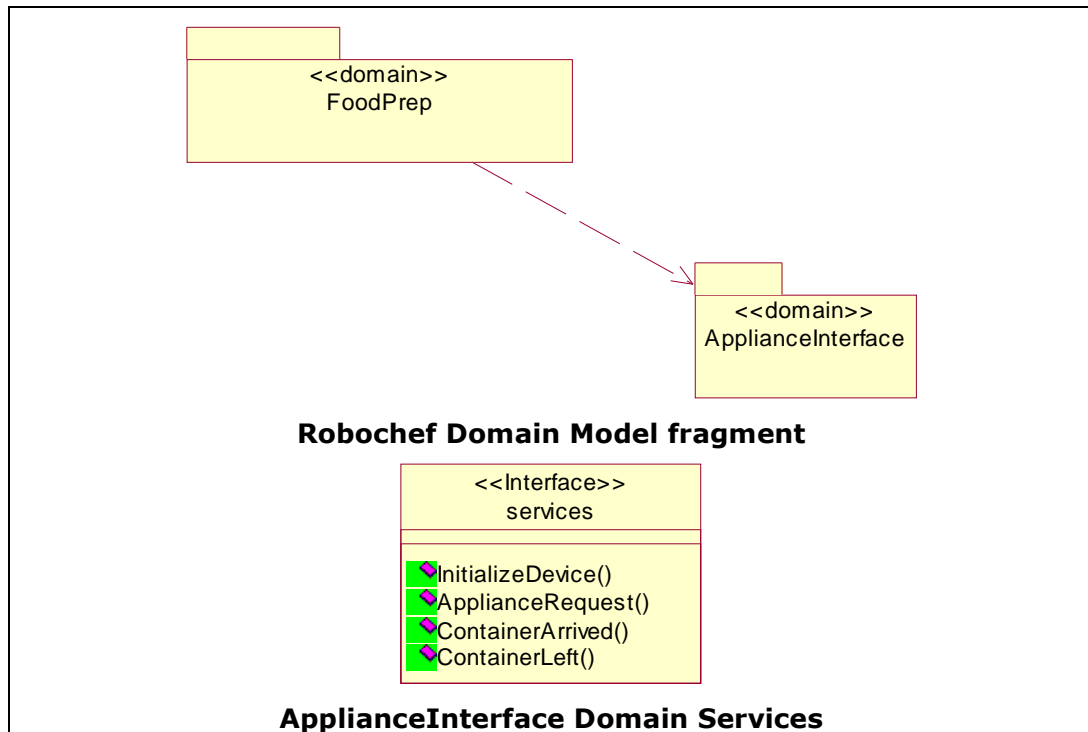
Incidents will continue to be processed in the order they appear in the incident queue.

(If run-time performance issues arise, a simple optimization can be applied to speed incident insertion into the appropriate queue position: a list or array of list insertion positions can be maintained, one for each priority. However it is not anticipated that such an optimization will be required.)

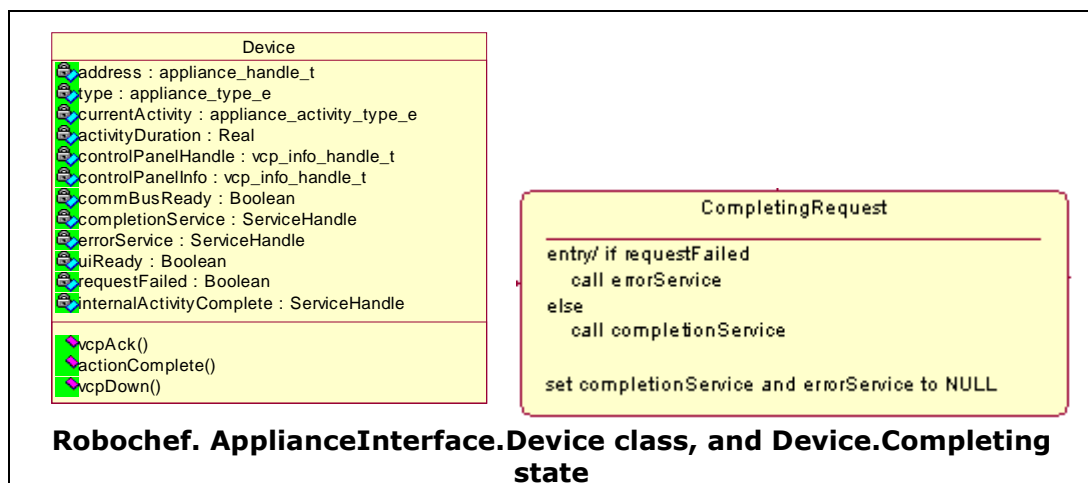
Local ServiceHandle Limitations

An Incident is an Event or a ServiceHandle. All Events go into the PfdTask(SW_Task) incident queue - intertask and task-local. However only intertask ServiceHandles go into the PfdTask's incident queue. Task-local

ServiceHandles are dispatched directly - they do not go into the queue. For example, in the Robochef system, ServiceHandles are needed to communicate with the ApplianceInterface domain:



In the ApplianceInterface domain, the ServiceHandle parameters provided to the AI:ApplianceRequest domain service are saved as attributes of the Device class:



When a request has been successfully completed, the Device.completionService is CALLED from the Device.Completing state entry action:

Device.Completing state entry action (partial):

```
CALL completionService();
```

This CALL is generated to the following code for C++:

```
if(this->completionService)
{
    this->completionService->deliver();
}
```

The PfdIncident::deliver() method determines if the destination task is the same as the sending task (local) or not. If it is not local, then the incident gets queued in the destination task's incident queue, and therefore is subject to incident priority.

If it is local, the incident's virtual sendLocal() method is called. In the case of a PfdEvent, the sendLocal() method calls PfdTask::enqueueSelfEvent() or PfdTask::enqueueEvent(), and again is subject to incident priority. However if the incident is local and it is a PdfServiceHandle, then PfdIncident::deliver() calls PdfServiceHandle::sendLocal(), which calls System::RouteServiceInvocation() – avoiding the local task's incident queue, and therefore avoiding any incident priority **because it is dispatched immediately – synchronously**. *(This local-case synchronous dispatch is a key aspect of the execution semantics of the ServideHandle.)*

3. Modeling Considerations

A significant rule of OOA is the all Events sent by one Class instance to another Class instance must be received in the order that they are sent. It is up to the analyst to ensure that the specification of non-default priority for any Event does not potentially create a scenario in violation of this rule.

Some conventions can help avoid this issue, or identify problematic situations. One or more of the following can help:

- Ensure all events received by a class all have the same priority by using the class priority value.
- Ensure all classes in an active hierarchy all have the same priority.
- Ensure all events sent from one class to another all have the same priority.
- Use non-default priorities sparingly.

Testing Considerations

Delayed Events

Since delayed events are currently put into the event queue after their timer expires in all designs, delayed events will now be entered into the event queue based on their priority. This may have the additional side effect of delaying the event further.

Self-directed Events

Priority will override the self-directed event semantics. A self directed event with a lower priority will be placed behind a non-self directed event with a higher priority.

Self directed events will be placed in the queue based on event priority and destination. A self directed event will be inserted before another non-self directed event with the same priority and to the same destination.

Self directed test cases with no event priorities specified should operate the same,

Starvation

Higher priority cycles of events may keep lower priority events from being handled.

Prioritized Service Handles

Service handles are not mixed in with the event queue currently. Intertask service handles are kept in a separate queue and are processed in `PfdTask.processAsynchronousInputs`. This approach processes one event for every one external service handles.

Setting priority on service handles will have no discernable effect until both are combined into one event queue.

Untriggered Transitions

Untriggered transitions are a special case of self-directed events. Untriggered events need to be processed before any other event to that instance. Since the untriggered "event" will be placed in the event queue in front of any other events to the instance. This is accomplished by setting the priority of untriggered events to one higher than the highest event priority on the class.

4. Specifying a Complete Response Path

This section outlines the steps needed to initiate a response from realized code all the way from realized code at the edge of our system through the chain of high-priority incidents in the modeled domains. For this example let's return to the `Robochef.ApplianceInterface.ApplianceRequest` service. Presume our Design requires that all error case reporting from appliance hardware must be acted upon as soon as possible.

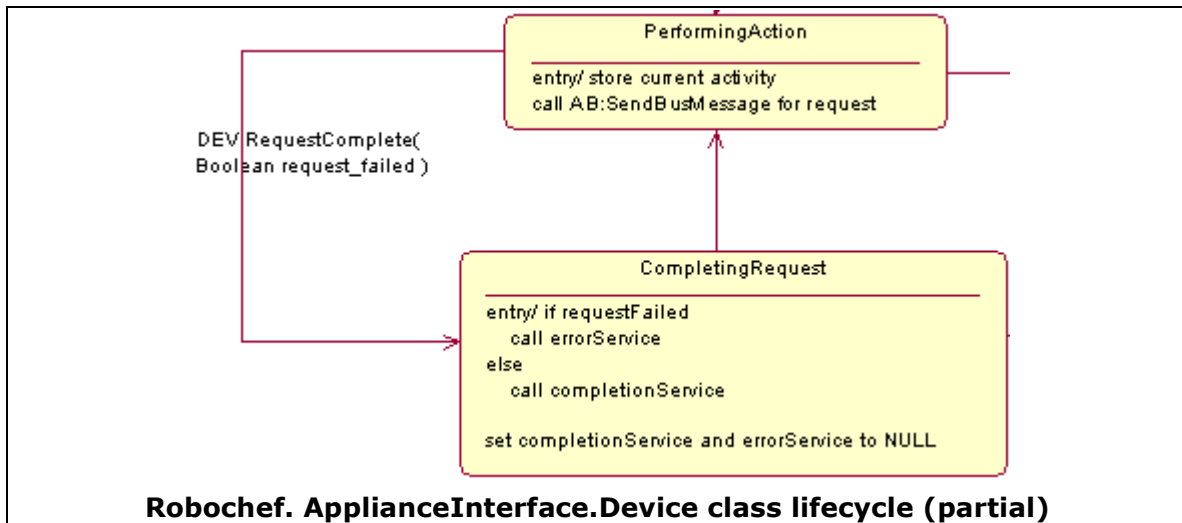
In this case we specify the entire communication chain from the realized Appliance Bus code up to the `FoodPrep.Active` recipe. Starting from the bottom up we see the `ApplianceBus` realized code calls a `ServiceHandle` to the class service `AI.Device.actionComplete`.

`ApplianceBus.SendBusMessage` domain service (C++, partial):

```
actionCompleted->sendLocal();
```

Please note: In general hand-written code should invoke the `PfdIncident::deliver()` method, so the interface from realized code does not change. As noted in section 2.3, `deliver()` will either call `sendLocal()` for local-task incidents or queue the incident to a remote destination tasks's incident queue.

The ApplianceInterface.DEV:RequestComplete event must receive high priority.

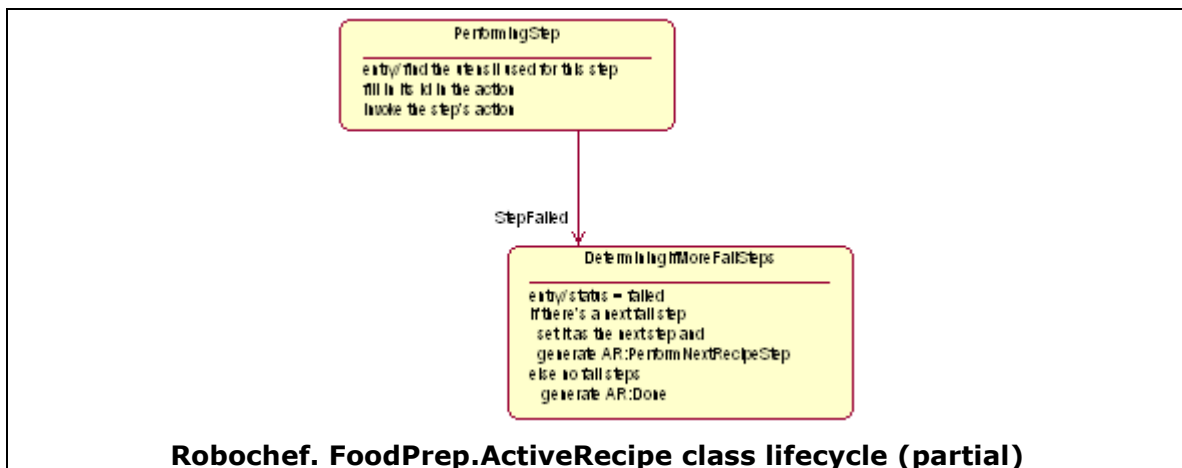


From here we can see the Device.errorService ServiceHandle must point to a high priority service. We see in the FoodPrep.Appliance.performOperation class service that the Device.errorService handle points to FoodPrep.Appliance.opFailed:

FoodPrep.Appliance.performOperation class service (partial):

```
actionFailed = CREATE ServiceHandle(this = this) TO APPL:opFailed;
```

The FoodPrep.Appliance.opFailed class service generates the event APPL:OpFailed, which then causes the generation of AR:StepFailed.



Properties.txt should include:

```
Event,Robochef.AI.DEV.RequestComplete,Priority,1
ObjectService,Robochef.FP.APPL.opFailed,Priority,1
Event,Robochef.FP.APPL.OpFailed,Priority,1
Event,Robochef.FP.AR.StepFailed,Priority,1
```

This we full specify the response path from realized code (AB_services.cpp) to the final response action (ActiveRecipe.DeterminingIf MoreFailSteps entry).