



UML Foundation for C Self-Trimming

Version 2.6

September 28, 2003

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. System-Level Control..... 1
- 3. Targeted Implementation Artifacts..... 1
 - Domain1
 - Class2
 - State Machine.....2
 - Association2
- 4. Sample..... 2
- 5. Template Techniques 3

1.Introduction

This Technical Note describes the optimizations to be deployed in RAM and code space optimization in the UML Foundation for C implementation architecture. It lists the target structures and accessor code to be eliminated when it is not needed.

This optimization is heavily affected by the analysis class-based code generation control property SuppressGeneration (TRUE/FALSE). Any class that is "suppressed" (not generated) because its SuppressGeneration == "TRUE" may potentially eliminate the access/reference/use of some other non-suppressed artifacts. The property SuppressGeneration is available for class services (however if a class has SuppressGeneration == "TRUE", all class services will automatically have their SuppressGeneration set to "TRUE"). The property SuppressGeneration is also available for domain services, and for subsystems where all classes and domain services in a subsystem with SuppressGeneration == "TRUE" will automatically have their SuppressGeneration set to "TRUE".

(Please note this technical assumes Subsystem Phase 2 capabilities as a base.)

2.System-Level Control

The system level property "SelfTrim" (default value is "FALSE") will control this feature. If "SelfTrim" is FALSE, then full code will be generated regardless of whether it is needed by the PAL actions. If "SelfTrim" is "TRUE" then the implementation artifacts in section 3. Targeted Implementation Artifacts will only be generated is explicitly needed by PAL actions.

To help illuminate the set of trimmed elements, when "SelfTrim" is "TRUE a trimming log named <system name>_<domain prefix>_trimming_results.txt is created for each domain in the top level generated code output directory.

3.Targeted Implementation Artifacts

The purpose of these optimizations is to reduce RAM and code space optimization in the UML Foundation for C implementation based on the actual needs of the model, in terms of what is accessed. The overall strategy is simple: while the overall implementation architecture calls for a full set of structures and functions to support all possible model (PAL) actions, only those that are actually required by an action currently present in the domain will be generated.

The difficulty in this feature comes from the need to initially examine all classes to see if any are suppressed (the SuppressGeneration property). The use of this property prevents these optimization efforts from relying on the accessor tracking data automatically constructed by Springboard, requiring that templates be constructed to navigate the non-suppressed PAL and construct their own data.

All implementation artifacts listed below will only be generated if they are actually accessed/referenced/used.

Domain

(All Domain Services will be generated to support external use.)

Class

Class-Extent Instance Tracking Structure (for FIND CLASS...)

Class-Extent Instance Tracking Functions

Attributes

Class Services (via explicit invocation, or if the target of a ServiceHandle)

Constructor/Destructor Functions

Various class support functions, including unplug, and unplugDispatcher

internalInfo_ (combination of rttiNumber_, state_ and used_ fields)

Class (if no aspects of a class are referenced, then nothing will be generated for the class)

State Machine

Transitions (If a transition's event is never GENERATED in PAL, then the transition/guard functions are not generated)

State Action Functions (If all transitions into a state are optimized away, and no class CREATES reference it as an initial state, then all state action function for the state will not be generated)

Association

Participant Instance Tracking Structures (in each participant class)

Participant Link/Unlink Functions (in each participant class)

Participant Navigation Functions (in each participant class)

***Note:** due to the automatic unlink aspect of class instance delete, any delete of a class constitutes a navigational access of that association from that participant. If there are no other association accesses from that participant's side and it is desired to avoid generating participant overhead from that side, the automatic unlink implementation can be forced to use a class-extent search of it's related participant. This is done by setting the association's "Optimize" property to "Small" (the default is "Fast").

4.Sample

In the following sample model segments from the Rooms domain, a simple example shows the elimination of a class and related code elements via the SuppressGeneration property.

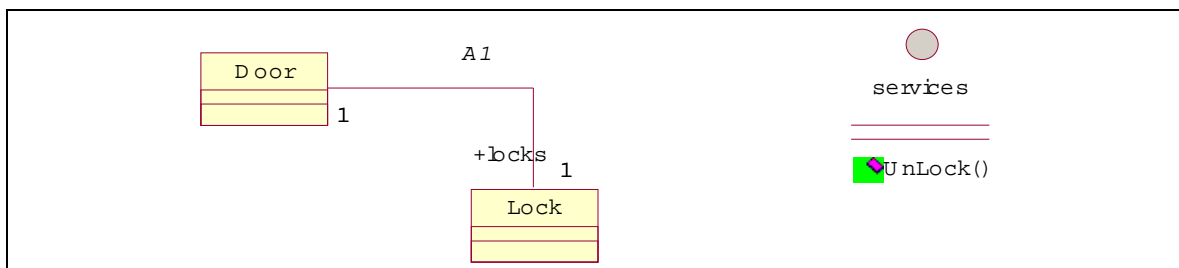


Figure 1 Class Model

The Room:Unlock() service contains:

```
. . .  
Ref<Lock> lck = FIND front_door->A1;  
. . .
```

Without eliminating the Lock class, the generated structure definition for Door contains a pointer supporting A1, and a corresponding navigation function:

```
struct Rooms_Door  
  
    . . .  
  
    /* Analysis associations */  
  
    Rooms_Lock_handle_t acrossA1_to_locks;  
  
    . . .  
  
Rooms_Lock_handle_t Rooms_Door_navigate_acrossA1_to_locks(Rooms_Door_handle_t start);
```

Now let us presume we wish to deliver a product variant without any Locks for doors. The following properties.txt file segment shows the elimination of the Lock class:

```
Object,TechNoteSample.Rooms.Lock,SuppressGeneration,TRUE  
DomainService,TechNoteSample.Rooms.Unlock,SuppressGeneration,TRUE
```

Now the files Rooms_Lock.c and Rooms_Lock.h are no longer generated, and the generated structure definition for Door has neither the pointer supporting A1, nor the corresponding navigation function.

5.Template Techniques

The Springboard translation engine has built-in data fields for many analysis elements listing their accessors. These accessor lists could be very useful in determining if each analysis element is required. However the impact of the analysis class and domain-service code generation control property SuppressGeneration (TRUE/FALSE) is unknown at model extraction time to Springboard, and these lists will not be accurate at the start of target file production (code generation).

In order to allow template logic processing to affect the model information population, a new Springboard directive has been made available from version 4.03.015 forward. This directive is [DELETE analysis_element]. It removes the specified analysis element from the model information, including eliminating it from any lists it may be referenced in. For example DELETing an object will remove it from its domain's objects field.

By making a "pre-processing" pass over the analysis elements, all Suppressed and unreferenced elements can be removed. This removal technique will be used for all elements, except for associations where only one side is used. In this case the relationship and both of it's participants must exist, but only the elements of the association that are used for the actually exercised navigations will be generated.

