Persistence Alpha Notes

of MC-3020

ROX Software, Inc.

The paper explains generated code infrastructure changes being made to support class instance persistence.

1. Persistence

The June 2004 alpha release contains code intended to reveal model compiler infrastructure changes in MC-3020 required for support of Persistence.

1.1. Class-Based Create/Delete/Init

The primary infrastructure change involves the centralization of the create, delete and initialization of class instances. The following paragraphs will explain the need for the change and its design. A description of the implementation of the generated code will follow with an explanation of the effects on speed, space and execution.

MC-3020 3.3 and before has a class-based create, delete, init design. The generated code contains a create accessor, delete accessor and factory initializer for each class in the modeled system. When an instance needs to be created, a routine specific to the class is invoked to allocate the implementation memory for the class instance. The attributes for the class instance are given reasonable default initial values. Identifying attributes of type unique_id are initialized to values certain to be unique with the generated system. The initial state of the new instance state machine is set.

In MC-3020 3.3 and before, when an instance needs to be deleted, a routine specific to the class is called to deallocate the implementation memory for the instance and do any required clean-up.

MC-3020 3.3 generates "object factory initialization" routines. There is one of these methods for each class in the modeled domains. These methods are called exactly once at bring-up time to initialize the instance memory data pool for the collection of instances for the class. This routine took into consideration the fact that some of the instances in a collection may be allocated and initialized statically as preexisting

instances. [Note: The June 2004 alpha release does not support preexisting instances defined in data. The beta release will restore this support.]

The class-based create/delete/init approach is very flexible. It allows the model compiler designer the liberty of considering only the class in hand when generating the code. Different profiles of deployment code can be generated based upon the type of class (passive, active, associative, having preexisting instances, etc). However, having the create, delete and init accessor dispersed to every class is troublesome when considering a persistent restore operation from centralized non-volatile storage (NVS).

1.2. Centralized Create/Delete/Init

MC-3020 3.4 and beyond centralizes the create, delete and initialize operations. This is accomplished by moving the intelligence of the class-based create/delete/init procedures into a class-based data structure. This class_info array contains information about each class such that centralized operations can intelligently create, delete and initialize classes and instances.

The information contained in the class_info structure includes:

active

: The head of the list of active instances is used to collect the active instances together.

inactive

: The head of the list of inactive instances is also maintained and used during create and delete operations.

container

: Container refers to the head of the list of container elements used to maintain collections. The base of this list is used to calculate indexes into the array of instances.

pool

: The base of the array representing the pool of instance data is accessible from the class_info array. The base of this list also is used to calculate indexes into the array of instances.

size

: The size of the class (in bytes) is necessary for initialization and for persistence support.

initial_state

: For active classes, the starting state of state machines must be kept for initialization of newly created active instances.

population

: Population is the number of instances in the instance collection.

[preexisting instances]

: Is used when there are preexisting instances defined in data. This element will inform the global initialization routine how many active instances to expect at power up so that initialization of

collections may proceed accordingly.

With the above information, it is possible to have centrally defined create, delete and initialization routines. These central routines rely on class-specific information from the class_info array rather than on class-specific generated procedures.

1.3. Implications to Persistence Support

The persistence restore operation will rely heavily upon the centralized create and initialization routines. Assuming for a moment a system that has no preexisting instances defined in data, a system having persistent classes must restore the class instance data from a central non-volatile storage unit (NVS).

When a system powers up, it will perform the standard system level initialization to bring up event queues and timer support and other system level infrastructure. The system then initializes the collection mechanisms for all of the classes in the generated system. This involves linking the instance storage into lists (singly or doubly linked).

At this point in a non-persistence supporting system the initialization functions would be invoked to allow application level user initialization to occur. In a persistence supporting system, persistent class instances will be restored from non-volatile storage before the user initialization functions are run.

The process of restoring class instances involves invoking the central create routine and populating the newly created instance with data from NVS. The data in the NVS needs to be rich enough such that the create routine knows which type of class to create. Once the type is communicated, the class_info array provides the necessary details to enable the correct instantiation of the new instance.

1.4. Implementation

In MC-3020 3.3 the files `*_object.c` contain methods to create, delete and perform factory initialization of class instances. To compare implementations, view one of these files in 3.3 generated code.

In MC-3020 3.4 and beyond, these routines are centralized. In the alpha release the routines are found in `sys_init.c`. The class_info structure is simply an array of pointers to elements of the structure which reside in the individual `*_object.c` files.

All domains in the system are coordinated such that a unique class number exists for each class regardless of domain. A single class_info array exists for the system containing information for all classes in all domains. During translation a unique number is assigned to each class. This number serves as an index into the class_info array thus allowing centralized create, delete, init and other other routines to look up statistics on any class in the modeled system.

1.5. Effects

Centralizing the create, delete and init functions has the intended effect of enabling a persistent restore operation that can rely on these class-independent routines (namely create). There are positive side-effects of this centralization.

Code space savings is the primary positive side effect. Instead of three routines for each class, there are now three routines for the entire system. This saves substantially on code size in the generated system. This effect will be greatest on systems with a large number of classes.

Other positive side effects include greater flexibility in adding features such as persistence. Speed has not yet been measure but is expected to be effectively the same as before.