Component Definition Document (CDD) for the AMI4CCM Example Component Assembly

Rev. -

December 15, 2010

Prepared By:
Northrop Grumman Corporation
Electronic Systems
Baltimore, MD

Table of Contents

1 Introduction

1.1 Scope

This document captures the specification and design for the AMI4CCM (Asynchronous Messaging Interface for the CORBA Component Model) Example software component assembly. This example assembly is targeted for deployment on the Scalable Node Architecture (SNA) real-time component framework. As such, it must be compliant with SNA Component Based Architecture (CBA) design guidelines.

This specification defines the component assembly's functional, interface and performance requirements, the context in which it must operate, and any design constraints it must adhere to. It provides criteria for verifying compliance, but it does not state methods for achieving results.

This is intended to be a relatively informal living document, to be included in same CM repository and package as the component source code. This CDD will initially be populated by a system engineer or software architect/lead to define component design constraints & guidelines. Over time, it will transition to enhance the "to be built" specification sections with "as built" design information documenting the final component product.

2 Applicable Documents

2.1 Applicable Government Documents

Document No.	Title

2.2 Other Applicable Documents

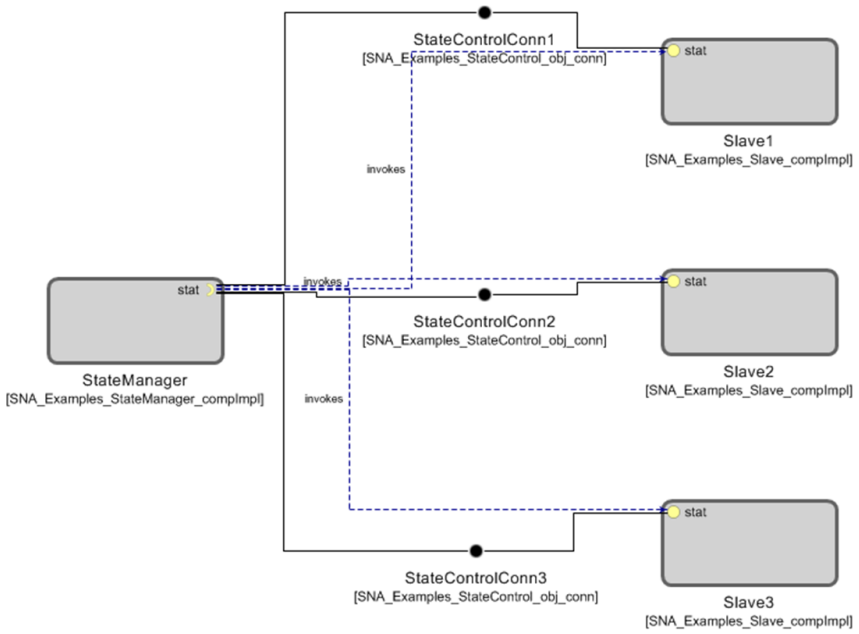
Document No.	Title

3 Component Description

3.1 Overview

The AMI4CCM Example component assembly is one of the component source examples included in the SNA SDK for reference, testing and experimentation. It illustrates a hierarchical two-level chain of two component designs, with the component at the top level making asynchronous/non-blocking client calls to service ports offered by lower level components. The asynchronous client ports demonstrate use of the AMI4CCM connector. The assembly containing these components is shown in Figure 3-1 below.

Figure 3-1 – AMI4CCM Example Component Deployment Assembly



This example assembly includes two component designs/implementations. The top level StateManager_comp runs a local timer to kick off a periodic call sequence. For every even number of occurrences of the timer event, StateManager_comp uses AMI4CCM to send a changeState command to the three instances of Slave_comp. Once the responses are collected, a new timer is scheduled. For the odd number of timer events, StateManager_comp uses synchronous CORBA invocations and afterward schedules a new timer.

3.2 Operational Context

The assembly is completely self-contained, with no external connections. The driving event is a periodic timer internal to the StateManager_comp component. It is designed to operate solely within the constraints of the SNA Platform's development environment to allow a new user/developer to step through the SNA component based development (CBD) process of loading a component assembly into the SNA SDK IDE, building/compiling it, and then executing it.

The example is provided with an appropriate set of SNA configuration files and a deployment plan to support its execution within a single-host SNA SDK "localhost" Virtual Machine (VM). Alternative variations on the default supplied design and deployment are possible via experimentation by a software developer.

4 Component Interfaces

The example assembly defines one internal service port specification, with one matching client port that accesses these services using an AMI4CCM connector. The component assembly has no external interfaces. The internal ports and component design functionality are described below.

4.1 Service Ports

4.1.1 StateControl_obj Service (internal)

This service is provided by the stateControlRecept port on a Slave_comp component, of which there are intended to be many to illustrate call fan-out and parallel state change, as well as the advantages of using the AMI4CCM connector to implement this pattern. This example includes a deployment with three instances of the Slave_comp component. The StateControl_obj service offers a changeState() method that is called by the StateManager_comp component to actually implement a multi-component state change for a notional software subsystem. The changeState() method returns a ReturnStatus enum indicating SUCCESS or FAILURE of each Slave_comp's state change request. The service port does not know or care whether the client makes the call synchronously or asynchronously.

4.2 Client Ports

A matching client port for the service described in the prior section is reflectively available on the opposite monolithic component.

4.2.1 StateControl_obj Client (internal)

This client port is used on the SystemManager_comp component to *implement* (vs. initiate) a notional subsystem state change by kicking off lower-level asynchronous calls to multiple Slave_comp instances in parallel. See StateControl_obj Service above. The client port is designated to be "asynchronous" in the design tools (Studio or CX), which causes it to be implemented using an AMI4CCM "connector." For more on AMI4CCM connectors, see the prior client port description.

4.3 Publisher Ports

This example assembly includes no publisher ports.

4.4 Subscriber Ports

This example assembly includes no subscriber ports.

5 Component Functionality

Reference the component assembly diagram in Figure 3-1 above. The critical functionality of each of the two monolithic component designs in this assembly will be described separately.

5.1 StateManager_comp Component Functionality

The StateManager_comp component will create a timer which on each timeout will alternate between AMI4CCM (sendc_changeState) and synchronous CORBA (changeState) invocations on the three instances of Slave_comp.

When using synchronous connections, the component's timer callback will log the responses and then schedule a new timer immediately. When using AMI4CCM, the timer callback will invoke the non-blocking, asynchronous version of changeState (sendc_changeState) and then return. After each Slave_comp instance completes its state transition, the AMI4CCM connector will invoke the changeState callback in the AMI4CCM reply handler with the return value of the call.

The reply handler will be shared within each set of three asynchronous invocations by passing the same instance as the first argument of the sendc_changeState call. The reply handler will keep a scoreboard of total expected responses as well as the number of successful and failed state transitions. Once the total number of expected responses is received, the reply handler will schedule a new timer. On the next set of asynchronous calls, a new reply handler will be created thus resetting the scoreboard. The cost of dynamically allocating the AMI4CCM reply handler is small with an average time required of approximately one microsecond.

As AMI4CCM is asynchronous, in some use cases there may be problems with out-of-order responses, such as mistakenly changing the scoreboard across two or more different commands. However, in this example the use of a different reply handler instance for each set of calls nullifies this problem due to each instance then having its own scoreboard.

When the StateManager_comp component uses the synchronous CORBA connections, then the scoreboard still exists but is intrinsic to the loop since each call is blocking until the response is received.

5.2 Slave_comp Component Functionality

In this assembly, there are three instances of this component type. The changeState() service callback implementation for each will simply return a typical SUCCESS or FAILURE response after checking if the commanded state transition is valid after a short (two second) simulated wait period.

6 Configurable Parameters

Since this component assembly is intended to support run-time experimentation, it is also packaged with a full set of SNA compliant run-time execution configuration and deployment files.

7 Design Constraints

1. This example assembly will implement all client ports using an AMI4CCM connector.
2. The AMI4CCM example will be driven by a periodic event defined by an internal timer.
3. The AMI4CCM example will illustrate the use of both the synchronous *and* asynchronous calls offered by an AMI4CCM connector.
4. The AMI4CCM example will generate SNA compatible log messages to allow a user to track the progress of the asynchronous call sequence and to indicate its overall success or failure.
5. A full set of SNA compliant configuration and deployment files will be provided with this example in order to support run-time execution in a default single-host target deployment.
6. This example will utilize the standard SNA APIs to perform all functions.

8 Component Test

This 4-component assembly must be executable on a single “localhost” development computer, to include the SNA SDK x86-64 VM at a minimum. Users can experiment with the deployment plan to redeploy to an alternative 2-host target environment if desired.

9 Component Dependencies

The AMI4CCM Example assembly is self contained and has no dependencies on any other application components. Its only dependency is on the SNA SDK's run time execution environment, including the CCM+DDS real-time component framework and its integrated AMI4CCM connector type.